NAIST-IS-MT9451123

修士論文

コマンドマクロ記述のための
ヴィジュアルプログラミング言語の設計とその実装

山本 雅哉

1996 年 2 月 16 日

奈良先端科学技術大学院大学
情報科学研究科 情報システム学専攻

本論文は奈良先端科学技術大学院大学情報科学研究科に
修士 (工学) 授与の要件として提出した修士論文である。

山本 雅哉

指導教官 ： 横矢 直和 教授
千原 國宏 教授
竹村 治雄 助教授

コマンドマクロ記述のための ヴィジュアルプログラミング言語の設計とその実装*

山本 雅哉

内容梗概

日常的な計算機利用においては, 複数のコマンドをある一定の順序で発行することがしばしばある. このような定型的な操作を手作業で行なう手間を省くためのツールが従来から提供されてきた. これらのツールの多くでは, 複雑なコマンド発行を自動化するために, 一種のプログラムであるコマンドマクロを利用した手法を用いる. しかし, プログラミングに関する知識を持たないユーザにとってはこのようなツールの利用は困難である.

そこで本研究では, コマンド発行自動化のためのツールを新たに提供する. ユーザは新たに設計したデータフロー型のヴィジュアルプログラミング言語 (Visual Programing Language, 以下 VPL と略記) を用いてコマンドマクロを容易に記述することができる. 記述されたコマンドマクロをインタプリタが解析することで, コマンド発行を自動化する. VPL を利用することによって, 複雑なコマンド発行の記述に対応するとともに, プログラミングについての学習の容易性をも提供する.

本論文では, 新たに設計した VPL の設計方針と文法規則について述べる. また, エディタやインタプリタなどから構成される, コマンド発行自動化のためのツールの実装について説明する.

キーワード

グラフィカル ・ユーザ ・インタフェース, コマンドマクロ, ヴィジュアル ・プログラミング ・ランゲージ, データフローモデル

*奈良先端科学技術大学院大学 情報科学研究科 情報システム学専攻 修士論文, NAIST-IS-MT9451123, 1996 年 2 月 16 日.

A Visual Programming Language for Describing Command Macros *

Masaya Yamamoto

Abstract

On daily computer use, users often give systematic command executions to computers. A number of tools have been presented for reducing user's load in such executions. The majority of the tools are based on command macros for automatic execution of complicated command sequences. However, it is difficult for computer users to use such tools, if they have little knowledge of programming.

In this research, a new tool for automatic command execution is presented. By using this tool, users can easily describe command macros by new designed data-flow type Visual Programming Language(VPL). An interpreter analyzes the described command macros , then commands are automatically executed. Using command macros makes it possible to describe complicated command sequences and presents ease to study programming.

This paper describes the design policies of the new VPL and its grammar rule as well as the implementation of the tool for automatic command execution.

Keywords:

Graphical User Intarface, Command Macro, Visual Programing Language, Data Flow Model

*Master's Thesis, Department of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-MT9451123, February 16, 1996.

目 次

図 目 次

表 目 次

1. はじめに

日常的な計算機使用においては, 複数のコマンドをある一定の順序で発行することがしばしばある. このような定型的な操作を毎回手作業で行なうのは労力が大きいため, 自動的にコマンドを発行するためのツールが提供されている. 例えば UNIX のシェルにおいて, シェルスクリプト [1] はこの目的で利用されている. シェルでは一種のプログラミング言語であるシェルスクリプトを用いてコマンド発行の際の引数や順序などをユーザがプログラムとして記述し, その記述内容を専用のツールが解析することによってコマンドが自動的に発行される. このようにコマンド発行の自動化を目的として記述されたプログラムのことをここではコマンドマクロと呼ぶ.

このコマンドマクロ記述のためのプログラミングは, 計算機の専門家以外の一般のユーザにとってはツールを利用する際の障害となり得る. これは, 一般ユーザは計算機についての教育を受けていないことが多く, プログラミングのための知識をあまり持っていないためである. しかし, 近年グラフィカル・ユーザ・インタフェース (GUI) の普及につれ, 一般ユーザが増加していることを考慮すれば, 一般ユーザにも利用可能なコマンド発行の自動化のためのツールを提供していく必要がある.

このような目的のために, 本研究ではコマンドマクロの記述のために特化したデータフロー型のヴィジュアルプログラミング言語 (Visual Programming Language: 以下 VPL と略記) を提案する. VPL とは, 一般の言語のように文字を主に利用するのではなく, 直線・四角形といった幾何学図形を含む図やアイコンを主に利用して記述を行なう, 視覚性を重視したプログラミング言語である [2, 3, 4, 5, 6]. 特にデータフロー型の VPL とは, 有向グラフを利用し, データフロー計算モデル [7] の概念に基づく VPL である [8, 9, 10, 11].

また本研究では, この VPL 専用の統合プログラミングシステムの開発を行なった. この統合プログラミングシステムでは, エディタとインタプリタとトレーサーの機能が提供される. エディタでは, 一般の描画ツールと同様の自然な操作によって, コマンドマクロを容易に記述することができる. インタプリタは記述されたコマンドマクロを解析することによって, コマンド発行を自動的に行なう. そして,

トレーサーではその解析の過程をインタラクティブに表示することによって, 記述されたコマンドマクロに対する理解の容易性を高める. 文法自身の視覚性の高さ, エディタでの記述の容易性, トレーサーでのインタラクティブなフィードバックなどによって, この統合プログラミングシステムはプログラミング知識を持たないユーザにとっても使用方法の習得が容易である.

本論文では, 第２章にて計算機におけるコマンド発行についての考察する. また, コマンド発行を自動化するための従来のツールや関連研究を紹介しその問題点等について述べ, その上で本研究で提案する自動化のための手法を説明する. 第３章では, 本研究で提案するコマンドマクロ記述のための VPL の文法設計方針を説明し, 更にその文法規則について述べる. 第４章では統合プログラミングシステムの設計方針や特長を説明し, その実装について説明する. 最後に第５章において本研究の成果をまとめ, 今後の課題等について言及する.

2. コマンド発行とその自動化

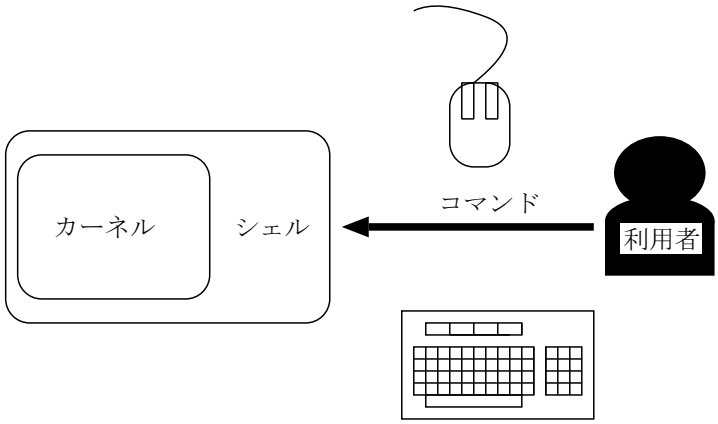


図 1　シェルの役割

本章では, 計算機におけるコマンド発行について考察する. また, コマンドの自動発行のための従来の手法を紹介し, 併せてその問題点を指摘する. そして, コマンドの自動発行のためのツールが満たすべき要件についてまとめた上で, 本研究で提供するツールにおけるコマンド発行の自動化の手法について述べる.

2.1 計算機におけるコマンド発行

計算機のユーザは計算機に命令を与えることで, 計算機に仕事を行なわせる. ここで, 計算機に対する命令のことを一般的にコマンドと呼ぶ. 計算機の利用者が発行したコマンドは, 図 1 で示すように, 計算機においてユーザインタフェースを司るシェルと呼ばれるプログラムが最初に受けとる. シェルではこのコマンドを解釈して, 計算機のユーザが必要とするプログラムを起動する. この, コマンド発行のための方法は, ユーザインタフェースの観点から, 大きく二つの方法に分類される.

- キャラクタ ・ユーザ ・インタフェース (CUI)
  キーボードからプログラムを特定するための文字列を入力することによっ

て, 計算機へコマンドを発行する.
UNIX のシェルや MS-DOS のコマンドラインがこの方法の例である.

- グラフィカル ・ユーザ ・インタフェース (GUI)
  画面上にプログラムを表すアイコンがあり, マウスなどのポインティングデバイスでそのアイコンをクリック（またはダブルクリック）することによって計算機へコマンド発行する.
  Machintosh のグラフィカル ・シェルや, MS-WINDOWS のプログラム ・マネージャがこの方法の例である.

また, プログラムによっては引数を取るものがある. 例えば,UNIX において cat コマンドは次のような書式[1]であり, 多くの引数を取る.

```
cat [-benstuvAET] [--number] [--number-nonblank]
 [--squeeze-blank] [--show-nonprinting] [--show-ends]
 [--show-tabs] [--show-all] [--help] [--version] [file...]
```

引数を取るプログラムでは, ユーザが引数を変化させることによって, プログラムの挙動を変更することができる.

プログラムに引数を与える方法は, シェルによって異なる. 一般的に CUI のシェルでは, プログラムを特定するための文字列に続けて, 与えたい引数を順次キーボードから入力することによって, 多くの引数をプログラムへと与えることができる. GUI のシェルでは, プログラムの起動の際にクリックではなく, ファイルを表すアイコンを目的のプログラムを表すアイコンへとドラッグ& ドロップ[2]することによって, 引数を一つに限ってプログラムへと与えることができる.

2.2 コマンド発行の自動化

コマンド発行を自動化する方法には大きく分けて二つの方法がある.

[1] この cat の書式は DEC OSF-1 上のものである
[2] アイコンの上でマウスのボタンを押下して, そのままマウスを目的のアイコンへと移動させ, その目的のアイコンの上でボタンを離すという一連の操作

- 操作履歴方式
  ユーザからの, キーボードやマウスによる操作をそのまま履歴としてツールが記録する. その操作履歴をツールが再生することによって, ユーザが行なったのと全く同じ操作が自動的に行なわれる. 記録する操作をコマンド発行のための操作とすることによって, コマンド発行を自動化することができる.

- コマンドマクロ方式
  コマンド発行の際の引数や順序を何らかのプログラミング言語によってユーザが記述する. その記述内容を専用のツールが解析することによって自動的にコマンド発行が行なわれる.

操作履歴方式は, 計算機のユーザにとっては普段の操作をそのまま行なうだけで, コマンド発行の自動化が可能となるために, 非常に直観的であり, その使用方法の習得も容易である. しかし, 記録された操作履歴は単なるキーボードやマウスのイベント列であり, そのイベント列からは記録されたコマンド発行の内容の理解は困難である. このために, 一部分の削除や変更などの編集作業は難しい. この方式のツールには, MS-WINDOWS に附属しているレコーダ [12] などがある.

コマンドマクロ方式は, 一種のプログラミング言語を利用するために, 編集作業は容易である. しかし, 普段の計算機利用の方法とは別に, プログラミング言語の文法について理解し, 更にプログラミング方法も習得する必要があり, プログラミング経験に乏しい計算機の初心者にはその利用は困難である. この方式のツールとしては, UNIX のシェルにおけるシェルスクリプト [1] や MS-DOS のバッチファイル [13] などがある.

また, この二つの方式の両方の長所を取り込んだ, Programing by Demonstration(例示によるプログラミング:以下 PBD と略記) または Programing by Rehearsal と呼ばれるシステムを利用したツールがある. PBD システム [14] とは, ユーザが何度か操作を行なうことによって, その操作から法則性を見い出して自動的にプログラムを作成するシステムのことを言う. この PBD システムを利用する手法では, 使用方法の習得の容易さと編集作業の容易さを両立させることができる. しかし現状では, ループや条件制御など, 複雑な操作の法則性の検出は困難で

ある [15]. この PBD を利用したコマンド自動発行のためのツールには Pursuit[16] などがある.

2.3 自動化のためのツールが満たすべき要件

近年の GUI の普及とともに, 計算機のユーザは益々増加している. これら新しいユーザの多くは, ワードプロセッサや表計算ソフトなど種々のアプリケーションソフトを活用することを目的とする, アプリケーションユーザと呼ばれる人々である. こうしたアプリケーションユーザであっても, 日々の計算機利用においては, 複数のコマンドをある一定の順序で発行する状況に遭遇することは多いと思われる. このために, コマンド自動発行ツールは, アプリケーションユーザにも必要である. しかし, アプリケーションユーザの多くは計算機についての専門教育を受けてはいないために, 従来型の言語によるプログラミングは困難であり, プログラミングを必要とするコマンドマクロ方式のコマンド自動発行ツールの利用は難しい. 従来, アプリケーションユーザを対象とするコマンド自動発行ツールの多くが, 操作履歴方式または PBD システムを利用した方式であったのは, このためである.

しかし, 条件分岐等の複雑な制御の記述や編集作業が可能なコマンド自動発行ツールは, 計算機に対する知識の差に関わらず, 全てのユーザに必要である. 特に, アプリケーションユーザのように, 計算機に対する知識を余り持たないユーザが利用するためのコマンド自動発行ツールには, 特別の配慮が必要である.

また, 最近の GUI を利用した OS はマルチタスク OS でもあることが多く, これらマルチタスク OS で必要とされるコマンドの自動発行のための制御には並列化を考慮した制御も必要である.

そして, アプリケーションユーザが発行するコマンド列は, 画像処理やデータベース処理と言った特定のプログラムに偏るのではなく, より広範な範囲に渡るものと思われる.

まとめると, 自動化のためのツールは次の要件を満たす必要があると考える.

(1) 操作方法の習得が容易である. 特に, プログラミング言語を利用するコマンドマクロ方式であれば, 特別の配慮が必要である.

(2) 編集作業が可能である
既に各々の方式で記述された一連のコマンド発行の一部分だけを削除したり, 別のコマンド発行と置き換えたりといった作業が可能でなければならない.

(3) マウスによる操作を中心とする
GUI を利用するアプリケーションユーザは, キーボード操作よりもマウスによる操作を好むことが多いため, キーボードの利用は必要最低限にとどめる

(4) 並列制御を含む, 種々の制御を可能とする
従来のコマンドマクロ方式のツールと同等の制御を可能とする. さらに並列制御も, マルチタスク OS では必要である.

(5) 各種コマンド発行への対応を可能とする
画像処理など特定のプログラムに特化するのではなく, 汎用的なコマンド発行を可能とする. このためには, 引数をコマンドへ与えることを可能とし, その与え方の制限も緩いものでなければならない.

2.4 関連研究など

先述した要件の多くを満たすツールが, 既にいくつか提供されている. しかし, これらのツールは要件の一部については満たさない.

Glaser らの提案する Psh[17] は, 要件の多くを満たしている. 彼らは, Prograph[18] という VPL を基として, コマンドマクロ記述のために新たに Psh 言語を設計した. Psh 言語では単なるコマンドマクロを越えて, 一般の GUI プログラムを記述することも可能である. しかし,Psh 言語は引数の与え方に関して強い制限があることや, 並列制御の記述が不可能なことなど, いくつかの点で要件を満たさない.

Modugno らが提案する Pursuit[16] は, PBD システムを利用する形式のツールである. PBD システムを利用するために, 理解容易性は高いものの, PBD システムによって推測可能な制御の数は多くないため, 実用上は問題があると思われる.

また, 市販品には SGI 社が提供する Iris Explorer[11] がある. このツールでも VPL を利用することによって, 操作方法の習得の容易性を高める工夫がなされて

いる. しかし, 本来画像処理のためのツールであり, 汎用的なコマンド発行には向かない.

2.5 本研究で提案する自動化のための手法

第 2.3 節で先述した各要件を満たすためには, いくつかの手法がある. 各要件ごとに考えられる手法を述べる.

- 要件 (1):プログラミング経験のないユーザにも利用可能
  - 操作履歴方式または PBD システムを利用する
  - コマンドマクロ方式において, 記述に VPL を利用する

  といった手法が考えられる. この選択肢の内では前者の方がより操作方法の習得は容易であると考えられる.

- 要件 (2):編集作業が容易
  プログラミング言語を利用したコマンドマクロ方式が非常に有効である.

- 要件 (3):マウス操作を中心
  選択肢としては要件 (1) と同様である. ただし, 要件 (1) の場合とは違い, どちらの手法でもマウスによる操作を中心とすることが可能である. 逆に, コマンドマクロ方式でも従来型の文字型言語を利用する方法では, キーボードでの操作が中心とならざるを得ない.

- 要件 (4):各種の制御記述を提供
  コマンドマクロの利用が有効. 特に, 並列制御のためには 状態遷移図 ・データフロー図 などに基づく VPL の利用が, 並列制御の視覚化を考慮すると有効である.

- 要件 (5):各種コマンド発行への対応
  操作履歴方式 ・コマンドマクロ方式のいずれにせよ, 対応は可能である. ただし, コマンドマクロ方式では記述言語の設計に依存する.

全ての要件をバランス良く満たすために, 本研究では Psh と同様に, VPL を利用したコマンドマクロ方式を採用することとした. そして, コマンドマクロを記述するための, 新たなデータフロー型の VPL を設計することとした. 特にデータフロー型の VPL を選んだのは, 並列制御の視覚化が容易であるということと, プログラミングの初学者にも理解の容易なフローチャートに似た構成のためである.

この新たに設計する VPL では要件 (5) を満たす必要がある. また, 要件 (4) に関しても設計する言語では各種の制御を記述できなければならない. さらに要件 (1) をより高度に満たすための工夫も必要である. この新たに設計する VPL についての詳細は第 3 章で述べる. また, VPL を記述するためのエディタや解析のためのインタプリタにおいても, 要件 (1) を更に満足するための工夫が必要である. このエディタやインタプリタの詳細は第 4 章で述べる.

また, PBD システムを利用した方法に関しては, 要件の多くを満たすものの, 現状では推測できる制御の数に問題があるために採用には至らなかった. ただし, PBD システムを利用することは提案手法と排反するわけではなく, PBD システムによってコマンドマクロの骨格を作成し, その後の編集作業や解析などに提案手法を用いることも可能である. 今後の PBD システムの進展によっては, 利用の可能性もあると思われる.

次章以降では, 本研究で提案する VPL の設計方針と文法規則, またその VPL 専用の統合プログラミングシステムの開発に関して述べる.

3. コマンドマクロ記述言語

本章では, 本研究で提案するコマンドマクロ記述のための VPL の設計方針について述べる. そして, 設計方針を元に設計した言語の文法規則について説明する.

3.1 言語の設計方針

言語を設計するにあたって, ツール全体の設計方針を基として, 更に詳細に設計方針を立てた. ここでは, それぞれの設計方針とその狙いについて説明する.

(1) 各種の制御の記述を可能にする
(ツール全体の設計方針の内で要件 (4) に対応する)
MS-DOS のバッチファイルや UNIX のシェルスクリプトなど, 従来のコマンドマクロ方式のツールが提供してきた制御の記述ついては, この記述を可能とする. この制御には, while ループ ・for ループ ・条件分岐などが含まれる. また, これに加えて並列制御の記述を可能にする. この並列制御には, 並列化 ・バリア同期などが含まれる.
これは, 従来から提供されてきた制御と同等のものは最低限記述できなければならないと考えるからである. また並列制御の記述は, マルチタスク OS におけるコマンド発行の自動化では重要と考えるからである.

(2) コマンド発行全般へ対応する
(ツール全体の設計方針の内で要件 (5) に対応する)
画像処理やデータベース処理といった特定のプログラムに特化したコマンド発行を目的とするのではなく, 全てのコマンドの発行へ対応する.
これは, アプリケーションユーザが発行するコマンド列は特定のプログラムに偏るのではなく, より広範な範囲に渡るものと思われるからである.

(3) 言語についての学習の容易性を高める
(ツール全体の設計方針の内で要件 (1) に対応する)
この設計方針を満たすために, 次のような工夫をする.

(a) 学習のために必要な知識を軽減する

データフロー型の VPL を利用するためには, ユーザはグラフ構造については最低限理解している必要がある. もしも, このグラフ構造の概念以外にユーザに求める知識があるとすれば, それはグラフ構造に似た知識であるか, 既に殆んどの人が知っていると思われる知識とする. プログラミングに関する教育を受けている人でなければ, 持ち得ないような知識はできるだけ利用しない.

言語を学習するために必要な知識を軽減することで, 言語についての学習そのものも容易となる.

(b) 段階的な学習を可能にする

言語について全てを完全に理解していなくても, 最低限必要な一部分だけを理解していれば, 簡単なコマンドマクロの記述は可能であるようにする. そのうえで, 更に文法を理解していけば, より複雑なコマンドマクロを記述することが可能であるようにする.

こうすることによって, 記述可能なコマンドマクロの種類を減らすことなく, 言語についての学習の中途段階でもとりあえずコマンドマクロを記述してみることを可能とする.

(4) 記述内容の直観性を高める

(ツール全体の設計方針の内で要件 (1) に対応する)

この設計方針を満たすために, 次のような工夫をする.

(a) 記述内容の全体的な把握を容易にする

記述されたコマンドマクロを一目見るだけで, 全体の概要を理解できるようにする. 逆に, 細部の詳細な記述の一覧性を犠牲にすることになる.

これは, プログラムの閲覧・理解はトップダウン式に行なわれるものであり, 全体把握の重要性が細部の理解の重要性に優ると考えるからである.

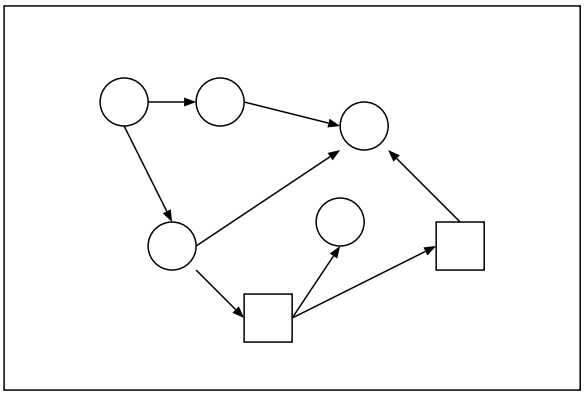


図 2 データフロー図

(b) グラフの正当性の判別を容易にする
データフロー図は一種の有向グラフであるが, VPL として利用可能な有向グラフは有向グラフ全体の集合の中の一部分である. ここで, どのような部分集合が VPL として正しい有向グラフであるかは VPL の設計に依存する. 今回設計する VPL では, この制約としてはグラフの局所的な性質に基づく制約のみを利用する.
これは大局的な性質に基づく制約は視覚的に分かりにくく, ユーザにとっては正しいかどうかの判定が困難なためである.

これらの方針を踏まえて設計した VPL について, 次節以降で説明する.

3.2 言語の概要

3.2.1 データフロー型の VPL

本研究で提供するコマンドの自動発行ツールではデータフロー型の VPL を用いてコマンドマクロを記述する. データフロー型の VPL とは, 図 2 のような有向グラフを用いるものである. データフロー型の VPL では, 記述されたプログラムの解析実行はデータフロー計算モデルの概念に基づく.

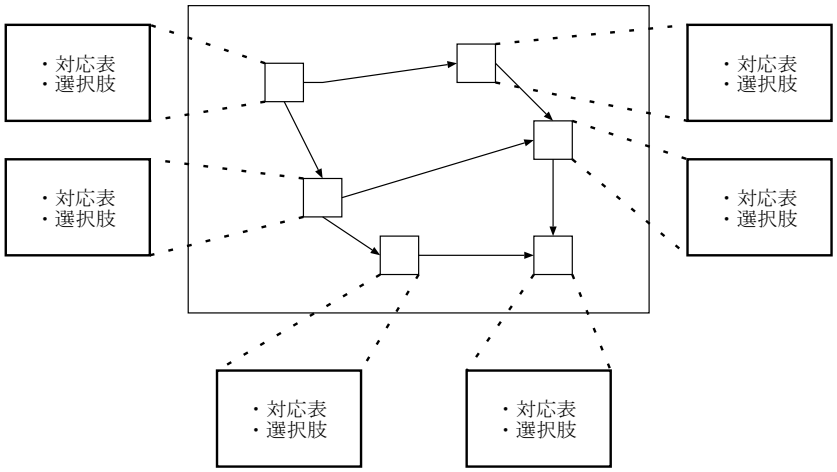


図 3 データフロー図と機能詳細記述図の関係

このグラフのノードはある特定の機能を実行することを表現している. ここで, 機能を表現しているノードのことをオブジェクトと呼ぶ. リンクはオブジェクトからオブジェクトへのデータの流れを表現している. コマンドマクロ記述のために設計されたこの VPL では, リンクを流れていくデータは, コマンド発行の結果やその結果を加工したものである. これらのデータが特定のオブジェクトへと渡されると, そのデータを引数としてコマンドが発行される. このようにして, コマンドが自動的に発行される. コマンドの発行以外にも, ユーザからの入力受付や受けとったデータの加工などの動作を表現するオブジェクトもある. また機能を表現しているオブジェクトとは別に, データフローの制御のためのオブジェクトも存在する.

設計した VPL では 1) オブジェクトとリンクを配置してデータフロー図を作成する 2) オブジェクトが表現する機能の詳細を記述する ことによってコマンドマクロを記述する. データフロー図は, 直線やアイコンなどを利用して視覚的に作成される. オブジェクトの機能の詳細は, 2-1) 対応表を作成する 2-2) 複数の選択肢から一つを選択する ことによって記述される.

データフロー図は, コマンドマクロの全体的な構造を表現している. 各オブジェ

クトの機能の詳細は, コマンドマクロの細部の動作を表現している. この状況を図 3 に示す. このように操作を階層的に分離することによって, 記述内容の全体的な把握を容易にするという設計方針の一つを満たしている.

3.2.2 データの形式

データフロー図において, リンクを流れオブジェクトへと渡されるデータのことをここではトークンデータと呼ぶ. 一つのトークンデータは複数のデータ項目から構成される. さらに, 各データ項目はデータ型とデータ内容から構成される. まとめると, トークンデータとは表 1のような形式のデータである.

n は定数であり, 一つのデータフロー図中においてリンクを流れるトークンデータは全て同数のデータ項目を持つ.

データ型として許されるのは次の３つに, 不使用を表す型を含めた４つのデータ型である.

1 整数値型　　数値を保持するデータ型である

2 文字列型　　文字列を保持するデータ型である

3 真偽値型　　真偽値を保持するデータ型である

データ内容にはデータ型に応じて, 整数値 ･文字列 ･真偽値が格納される. データ型が不使用の場合にはデータ内容は無視される.

表 1　トークンデータのデータ形式

項目番号	項目の型	項目の内容
1	データ型 1	データ内容 1
2	データ型 2	データ内容 2
⋮	⋮	⋮
n	データ型 n	データ内容 n

このように, トークンデータは汎用的な構造を持つ. これは, あらゆるコマンド発行の結果をトークンデータとして格納することを目的としており, これによって, コマンド発行全般への対応という設計方針を満たしている.

3.2.3 オブジェクトの機能

コマンドマクロの記述のために, 八種類のオブジェクトを提供する. この中には, 定型操作の基本単位となるプログラム起動を表現するオブジェクトや, データフロー計算モデルにおける制御を行なうためのオブジェクトなどが含まれる. ここで, 八種類のオブジェクトの各々についてその機能を述べる.

1 入力オブジェクト
ユーザから新たに入力を受け付けて, オブジェクトが受けとったトークンデータとともに, 次のオブジェクト[3] へと送る.

2 実行オブジェクト
オブジェクトが受けとったトークンデータを引数としてプログラムを起動し, プログラムの実行結果と受けとったトークンデータとを編成して, 次のオブジェクトへと送る.

3 中継オブジェクト
オブジェクトが受けとったトークンデータを加工して, 新たにトークンデータを作成し, 次のオブジェクトへと送る.

4 定数オブジェクト
オブジェクトに事前に定義されている数値や文字列などのデータを, オブジェクトが受けとったトークンデータとともに, 新たにトークンデータへと編成し, 次のオブジェクトへと送る.

5 単純分岐オブジェクト
オブジェクトが受けとったトークンデータをそのまま, 全ての次のオブジェクトに送る.

[3]データフロー図において, 目的とするオブジェクトとリンクで結ばれており, リンクの終了端側にあたるオブジェクトのことである.

6 条件分岐オブジェクト
オブジェクトが受けとったトークンデータの各データ項目を事前に定義されている条件と照らし合わせて全ての次のオブジェクトの中から一つを決定し, トークンデータはそのままの形で, その決定したオブジェクトへと受けとったトークンデータをそのまま送る.

7 AND バリアオブジェクト
全ての前のオブジェクト[4] からのトークンデータを受けとった段階で, 全てのトークンデータから新たにトークンデータを編成し, 次のオブジェクトへと送る. 前のオブジェクトのどれか一つからでも, トークンデータを受けとることができない時には, このオブジェクトは動作しない.

8 OR バリアオブジェクト
前のオブジェクトのうちの一つからトークンデータを受けとった段階で, そのトークンデータから新たにトークンデータを編成し, そのトークンデータを次のオブジェクトへと送る.

これら, 八種類のオブジェクトのうちで, 実行オブジェクトだけは意味のあるコマンドマクロ記述のためには必要である. その他七種類のオブジェクトについては, 記述する制御によっては利用の必要がないものもある. これによって, 全てのオブジェクトについて内容を理解することなく, 最低限必要なオブジェクトについてのみ理解していけば, とりあえず簡単なコマンドマクロを記述することが可能である. これは段階的な学習が可能という設計方針を満たしている.

また, 条件分岐 ・単純分岐 ・AND バリア ・OR バリアの四種類のオブジェクトは, 主に制御の記述を目的としたオブジェクトである. これら四種類のオブジェクトを利用することによって, 条件分岐 ・並列化 ・バリア同期 ・各種ループなど様々な制御を記述することが可能であり, これも各種制御の記述という設計方針を満たしている. 実際に各制御を記述したデータフロー図を次節に示す.

さらに, これらの制御構文のうちで, 条件分岐 ・並列化 ・バリア同期 ・while ループは次節の図にも示されている通り, 簡易に記述が可能であり, その記述は直観

[4] データフロー図において, 目的とするオブジェクトとリンクで結ばれており, リンクの開始端側にあたるオブジェクトのことである.

的である. また, 条件分岐オブジェクト ・単純分岐オブジェクト ・OR バリアオブジェクト ・AND バリアオブジェクトがこれらの制御にほぼ一対一に対応しており, 制御の記述とオブジェクトの配置操作が直観的に対応している.

この, オブジェクトの種類に関しては,

- できるだけ種類を少なくする
  逆に一つのオブジェクトが表現する機能は複雑になる
- できるだけ各オブジェクトが表現する機能を簡単にする
  逆に, 機能毎にオブジェクトを提供するために, 種類は増える.

という選択肢があった. しかし, 前者の方法では段階的な学習という方針を満たすのは難しく, 制御の記述とオブジェクトの配置操作が対応しない.

3.3 プログラムの記述例

この節では, 制御の記述とオブジェクトの配置の対応を見るために, いくつかのプログラムの例を示す.

図 4 の直列制御の例では, 入力されたトークンデータが実行オブジェクト１へと渡されて, プログラムが起動される. 実行オブジェクト１ではプログラムの終了後に, 実行オブジェクト２へとトークンデータを渡す. 実行オブジェクト２では, トークンデータを受けとった段階でプログラムを起動する.

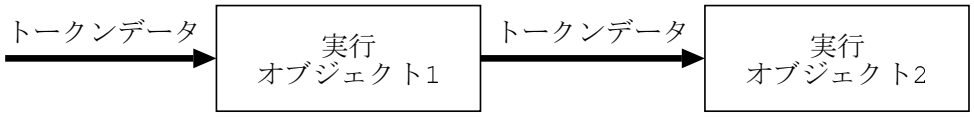


図 4 直列制御のプログラム例

図 5 の並列制御の例では, 入力されたトークンデータは単純分岐オブジェクトによって, 同時に実行オブジェクト１と実行オブジェクト２の両方に渡される. この時に両方の実行オブジェクトでプログラムの起動が行なわれる.

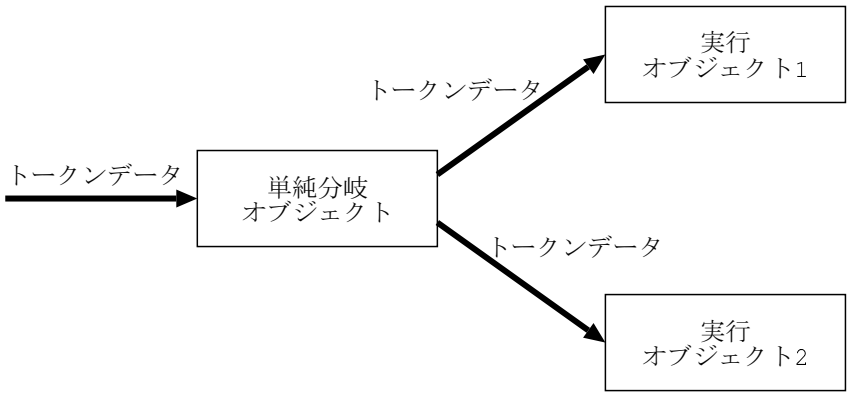


図 5 並列制御のプログラム例

図 6 の条件分岐制御の例では, 入力されたトークンデータは条件分岐オブジェクトによって, 実行オブジェクト１と実行オブジェクト２のどちらか一方に渡される. トークンデータを渡された方の, オブジェクトではプログラムが起動され, もう一方のオブジェクトでは何も起こらない.

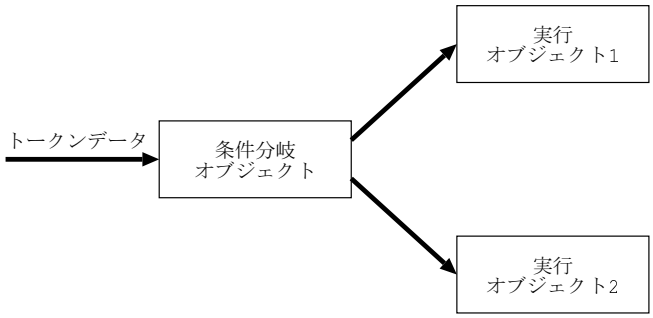


図 6 条件分岐制御のプログラム例

図 7 のバリア同期制御の例では, 実行オブジェクト１と実行オブジェクト２の両方からのトークンデータを受けとるまで, AND バリアオブジェクトは動作を開始しない. そして, 両方からのトークンデータを受けとった段階で, 実行オブジェクト３へとトークンデータを送る.

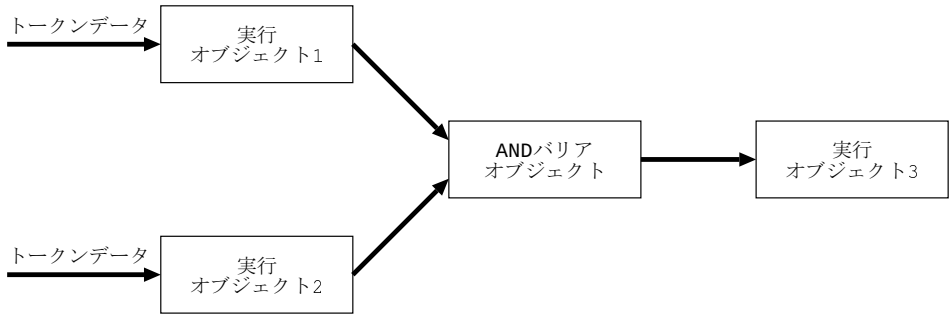


図 7 バリア同期制御のプログラム例

図 8 のループ制御の例では, 入力されたトークンデータは OR バリアオブジェクトによって, オブジェクト 1 へとそのまま渡される. ループ本体での動作を終えて, オブジェクト n から出力されたトークンデータは, OR バリアによって再びオブジェクト 1 へと渡される.

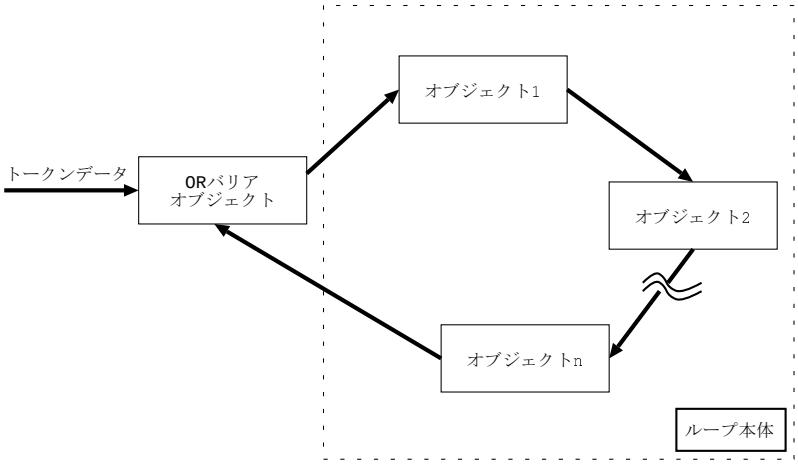


図 8 ループ制御のプログラム例

3.4 データフロー制御

3.4.1 発火規則

データフロー計算モデルにおける制御原理は, 発火規則に基づいたデータ駆動方式である.

トークンデータを受けとったオブジェクトは実行状態となり（発火と呼ぶ）, そのトークンデータを消費し, 実行の結果をトークンデータとして次のオブジェクトへと出力する. この様子を図 9に示す.

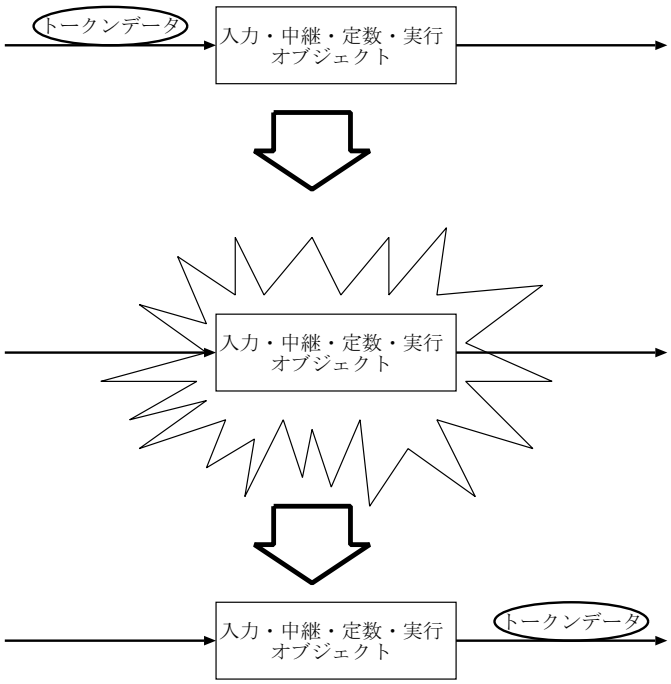


図 9 オブジェクトの発火の様子

3.4.2 オブジェクトの発火規則

本来, データフロー計算モデルでは入力される全てのリンクからのトークンデータを受けとった段階でオブジェクトが発火するのであるが, これを OR バリアオブジェクトにおいて変更する. OR バリアオブジェクトでは, 入力されるリンクの

うちの一つからでもトークンデータを受けとれば発火する. 同時に複数のリンクからトークンの入力があった場合にも, 一つずつ処理される. この様子を図 10に示す.

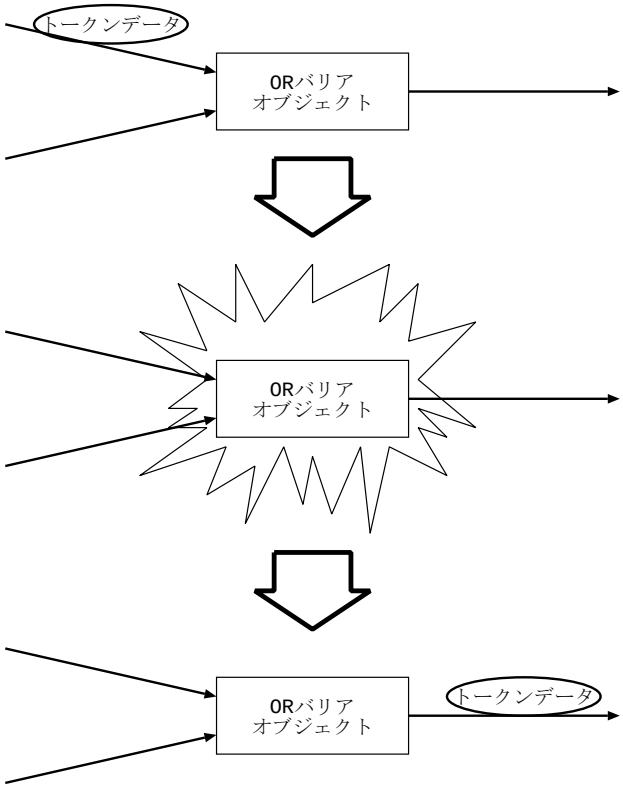
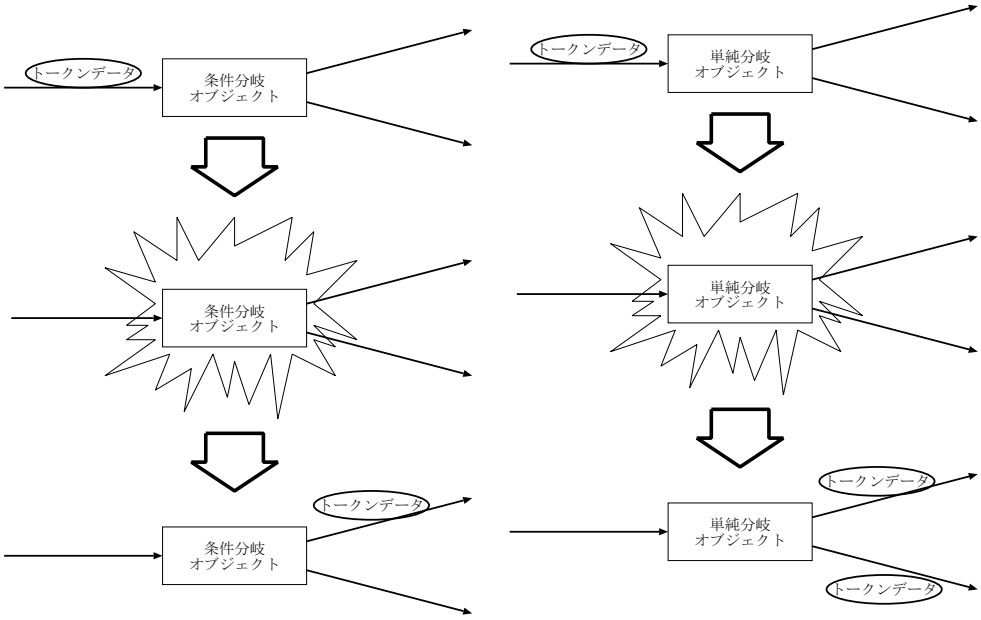


図 10 OR バリアオブジェクトの発火の様子

3.4.3 複数の出力リンクを持つオブジェクト

条件分岐オブジェクトと単純分岐オブジェクトのみが, 複数の出力リンクを持つことが可能である.

条件分岐オブジェクトでは実行結果のトークンを複数の出力リンクの内の一つのみに出力する. どのリンクに出力するかは条件分岐オブジェクトの機能の詳細記述に依存する. 単純分岐オブジェクトでは, 実行結果のトークンは, 複数の出力リンクの全てに同一のものが送られる. この様子を図 11に示す.



(a) 条件分岐　　(b) 単純分岐

図 11　分岐オブジェクトの発火の様子

3.4.4 出力リンクを持たないオブジェクト

実行オブジェクトは, 出力リンクを持たなくても構わない. このとき, 出力されるはずのトークンデータは消滅する. この様子を図 12に示す.

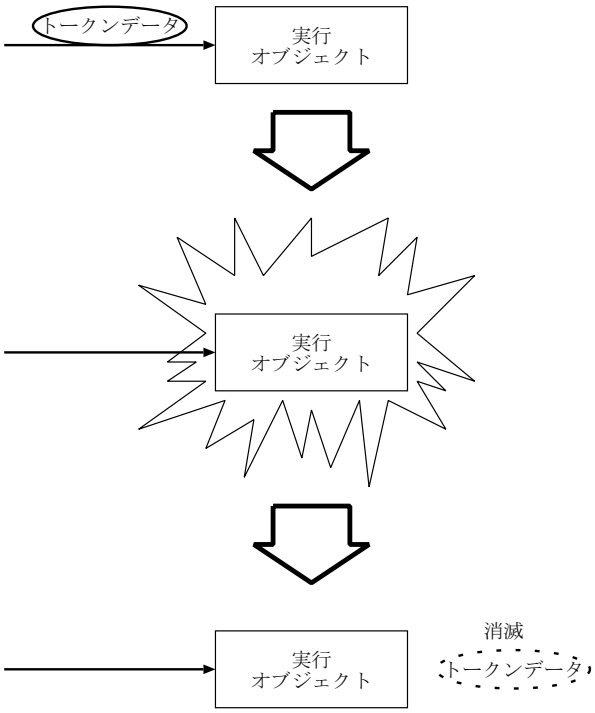


図 12　出力のないオブジェクトの発火

3.4.5　解析実行の開始と終了

データフローにおける解析実行の開始段階には, 入力リンクを持たないオブジェクトに, 全て空のデータ項目を持つトークンデータが与えられる. また, 全てのトークンデータが消滅した段階で解析実行は終了する.

3.5　データフロー図の制約

3.5.1　オブジェクトの配置制約

オブジェクトの配置に関しては特に制約はない. 適当な場所にどの種類のオブジェクトでも必要なだけ配置できる. このように, グラフの記述の際にオブジェクトの配置をユーザーが自由にできることから, ユーザが自分に取って最も理解

し易いように配置できる.

3.5.2 リンクの配置制約

ここで, あるオブジェクトを開始端として持つリンクのことをそのオブジェクトの出力リンクまたはオブジェクトから出ていくリンクと呼び, あるオブジェクトを終了端として持つリンクのことを, そのオブジェクトの入力リンクまたはオブジェクトへと入るリンクと呼ぶ.

さて, リンクの配置の際には, あるオブジェクトへと入るリンクの数に上限がある. この上限の数はノードに配置されているオブジェクトの種類によって異なる. 出ていくリンクに関しても同様である.

逆に, あるオブジェクトへと入るリンクの数には必要最低限の数が決まっている. この必要最低限の数はノードに配置されているオブジェクトの種類によって異なる. また, 終了端に関しても同様である.

この上限数と必要最低限数について表 2にまとめる.

表 2 リンクの上限数と最低限数

オブジェクト	出力		入力	
の種類	必要数	上限数	必要数	上限数
入力オブジェクト	0	1	1	1
実行オブジェクト	1	1	0	1
中継オブジェクト	0	1	1	1
定数オブジェクト	0	1	1	1
単純分岐オブジェクト	1	1	2	2
条件分岐オブジェクト	1	1	2	2
AND バリアオブジェクト	2	2	1	1
OR バリアオブジェクト	2	2	1	1

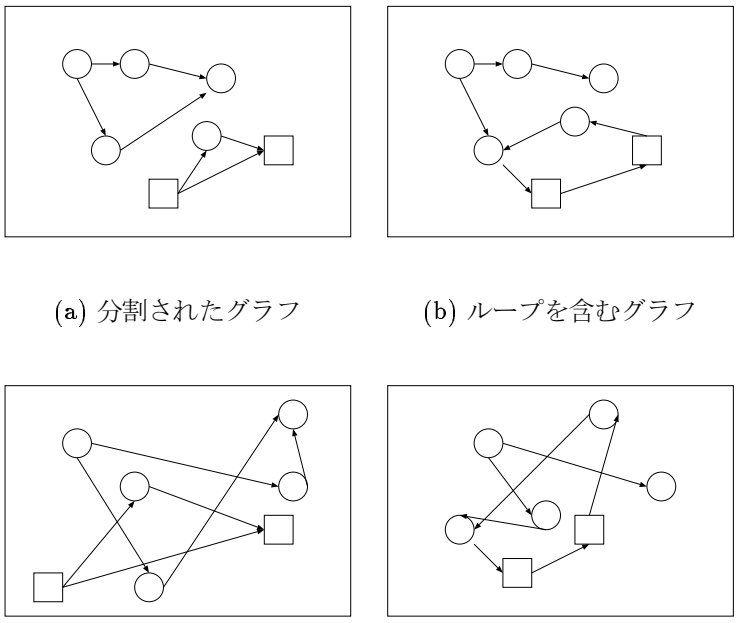
図を見ればわかるように, 実際には殆んどの部分が 1 または 2 であり, 必要数

と上限数が一致しないものも少ない. これはグラフの正当性の判別を容易にするためである.

3.5.3 その他の制約

リンクの配置に関して制約がある以外には, データフロー図の作成に当たっては制約はない. 特に, 大局的な制約は一切設けていない.

例えば, 図 13(a)(c) のように二つに分割されたグラフでも, 図 13(b)(d) のようにループしているグラフでも構わない. グラフの正当性の判別を容易にするためである.



(a) 分割されたグラフ　　(b) ループを含むグラフ

(c) わかりにくい分割　　(d) わかりにくいループ

図 13 制約を満たすデータフロー図

3.6 オブジェクトの機能詳細記述

第 3.2 節で先述したように, オブジェクトの機能の詳細は

- 対応表を作成する
- 複数の選択肢から一つを選択する

という操作によって記述される.

この対応表は, 入力されたトークンデータの各データ項目の扱いと出力されるトークンデータの各データ項目の作成方法などを定義する. 対応表において対応元と対応先の意味するものや, 対応表が満たすべき条件はオブジェクトの種類によって異なる.

また, 選択肢からの選択は

- 入力オブジェクトにおいて, ユーザから入力されるデータのデータ型
- 実行オブジェクトにおいて, 発行するコマンド
- 中継オブジェクトにおいて, データ加工の方法
- 定数オブジェクトにおいて, 定数データのデータ型
- 条件分岐オブジェクトにおいて, 分岐条件となるデータ項目

といったことを決定するために行なわれる.

各々のオブジェクトにおける詳細記述については付録 A に示す.

4. コマンドマクロ記述言語のプログラミングシステム

本章では, 本研究で設計したコマンドマクロ記述のための VPL 専用に実装した統合プログラミングシステムの設計方針について述べ, その実装について説明する.

4.1 統合プログラミング環境の概要

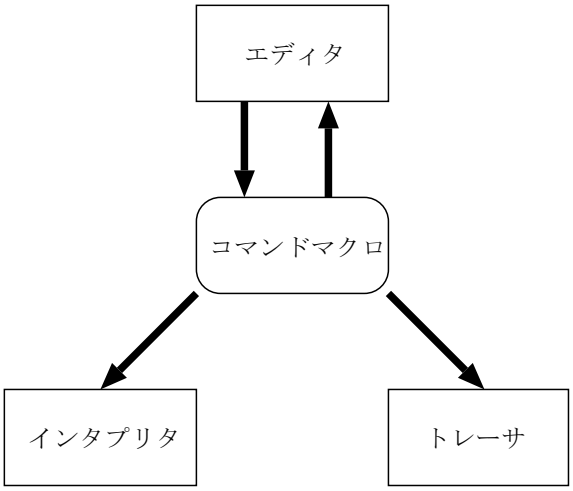


図 14 統合プログラミングシステムの構成図

本研究において実装した統合プログラミングシステムは図 14 に示す構成である.

- エディタ
  エディタでは, コマンドマクロを記述 ・編集する. エディタでの操作は図 ・アイコンなどを利用して視覚的に行なわれる.
- インタプリタ
  エディタによって記述されたコマンドマクロを解析してコマンドの自動発行を行なう.

- トレーサ
  インタプリタでの解析の様子を視覚的にフィードバックする．これによって, 記述されたコマンドマクロの内容を容易に理解することが可能であり, バグの発見に貢献する.

エディタによって記述されたコマンドマクロは, ファイルの形で保存される．このファイルには, グラフ ・対応表 ・選択肢などの情報が保存される.

このエディタやトレーサの設計方針はツール全体の設計方針に基づく．次節以降では, エディタとトレーサについてその特長を説明する．また操作方法などの実装についても説明する.

4.2 エディタの特長

統合プログラミングシステムにおいて実装したエディタは以下のような特長を持つ.

- グラフ記述への特化
  エディタにおけるグラフ記述のための操作は, 一般の描画ツールでの操作をベースとして, グラフ記述へと特化する方法で実装している．このために, 一般の描画ツールの描画コマンドから, 特にグラフ記述に利用できるものを取捨選択した．これによって, 多くの操作を覚えることなく, 最小限の操作方法を習得するだけでグラフの記述が可能である．また, その操作と対応する操作結果もグラフ記述に向くように設計している．この操作の詳細に関しては第 4.4 節で詳細に説明する.
  この特長によって, ユーザはグラフの作成を容易に行なうことができる.

- 対応表作成の容易さ
  エディタにおける対応表作成のための操作は, グラフ記述での操作とよく似たものとしている．これによって, 全体の操作の一体感を高めている．この操作の詳細に関しては第 4.4 節で詳細に説明する.
  この特長によって, ユーザはグラフ記述の操作方法という, データフロー型

のシステムを利用する以上最低限の知識さえ身につければ, 後は特に習得すべき操作方法はなく, システムを利用することができる.

- 誤った記述の排除
  エディタにおいて言語として誤った記述をユーザがしようとすると,
  - 言語として誤った記述には, エラーを意味する表示を行なうことで, プログラマに文法エラーを通知する.
  - 言語として誤った記述をしようとしても, その操作の結果は対応する結果とはならず, 操作が無かったかのように振舞う.

  などの方法で, 誤った記述を排除するようにしている.

  前者の例としては,
  - リンクの数の制約を満たしているオブジェクトとそうでないオブジェクトでの表示の違い. (図 15(a) を参照)
  - 選択すべき項目を選択していないオブジェクトと, 正しく選択されているオブジェクトの表示の違い

  などがある. 後者の例としては,
  - リンクの数の制約を満たさなくなるようにはリンクを張ることができない. (図 15(b) を参照)

  などがある.
  これによって, グラフの満たすべき制約などについて習得していなくても, エラーを減らすように操作を繰り返すことによって, 言語として正しい記述を行なうことができる.

- グラフの制約の視覚化
  グラフの作成においては, 第 3.5節で述べたように, リンクの配置に関して制約がある. この制約に関しては, オブジェクトの表示にリンクの配置点を示すことによって視覚化している.
  このために, ユーザは特にリンクの配置制約を覚える必要はない.

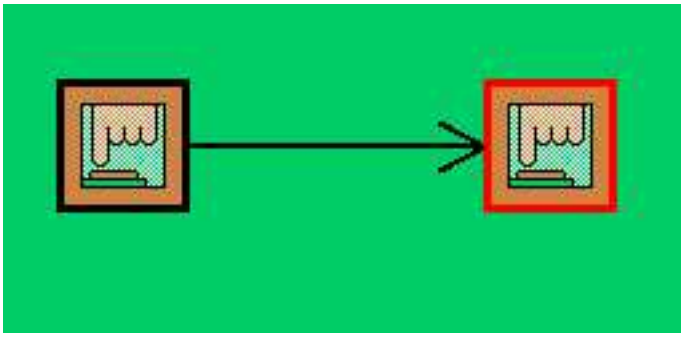
- データ項目への名前付け
  言語としては, トークンデータの各データ項目はデータ型とデータ内容を持つだけであり, 各データがどのような意味を持つのかについては全く不明である. これでは, プログラミングに当たって, 各データ項目の持つ意味を把握することが容易ではないために, 各データ項目に意味を表すタグをつけることができる. このデータ項目への名前付けは基本的には自動化されており, キーボードからの入力は最低限に押えられている.
  これによって, 記述されたコマンドマクロの動作についての理解の容易性を高めている.

- 変数機能の追加
  言語としては変数を扱ってはいないが, 便利な機能であるので変数を利用できるようにした. ただし, 変数を利用せずとも変数を利用したのと同じプログラムを作成することは可能である. ここで, 変数はプログラム中の全ての場所で共通の広域変数であり, 特にプログラマが指定することなくとも, 自動的に数値型 ・文字列型 ・真偽値型の各々に二つずつ定義されている. ただし, プログラマがこの数を増やしたり減らしたりすることはできない.
  この変数機能は, 追加の機能でありユーザはこの機能を理解しなくとも, システムの利用には全く差し支えはない.

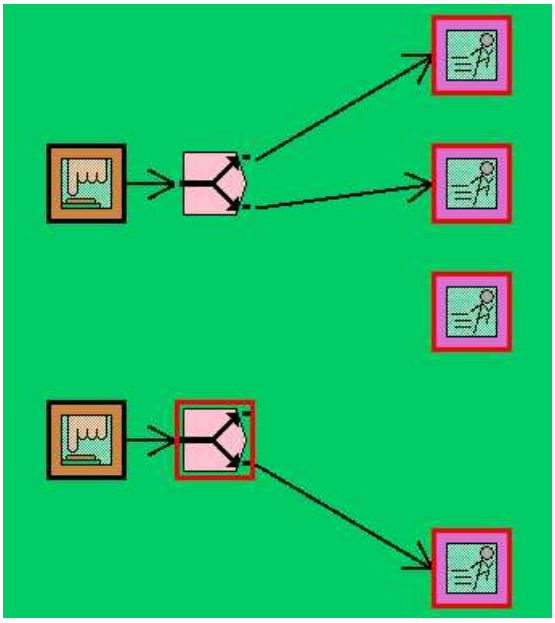
4.3 トレーサの特長

統合プログラミングシステムにおいて実装したトレーサでは, トークンデータが流れて発火が起こる度に, 一旦コマンドマクロの解析が停止する. ここで, ユーザがボタンを押すことによって, その発火によって引き起こされる処理が行なわれる. このときに, 必要であれば機能詳細記述ダイアログも表示される. 解析が再開された後に, トークンデータが次のオブジェクトへと渡されて発火が起これば, 再び解析が停止する.

こうすることで, ユーザはトークンが流れていく様子をインタラクティブに見ることが可能であり, 記述したコマンドマクロの動作状況の理解が容易になる.



(a) リンクの制約に関するオブジェクト表示の違い



(b) リンクの制約によるリンク配置操作の結果の違い

図 15　ユーザへのエラーの提示

4.4 エディタの実装

この節では, エディタでのコマンドマクロの記述の様子について説明する.

4.4.1 エディタのメイン画面

エディタのメイン画面を図 16 に示す.

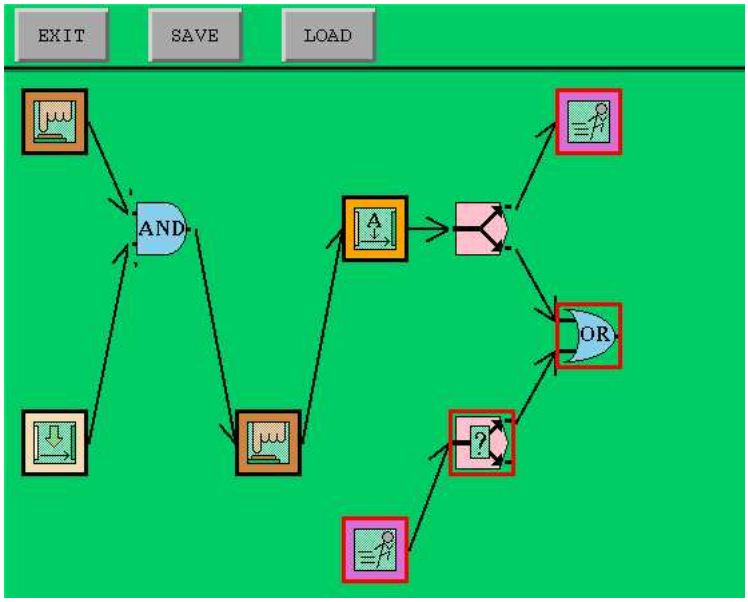


図 16 エディタのメイン画面

メイン画面では, データフロー図を記述する. また, セーブやロード ・終了のためのボタンも配置されている. データフロー図作成のためには, オブジェクトとリンクに関して合計６つの操作方法が提供されている. ここで, ６つの操作に関してその操作結果とともに説明する.

- オブジェクトの配置
  画面中のオブジェクトが何も配置されていないところをボタン１でクリックすることで, オブジェクトの種類を選択するダイアログボックス (図 17 を参照) を開いて, そこから希望のオブジェクトを選択する.

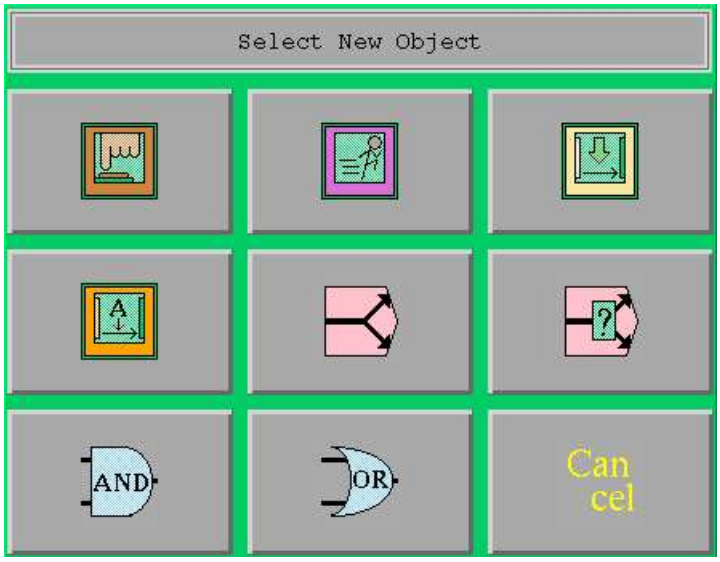


図 17　オブジェクト選択ダイアログ

- オブジェクトの削除
  オブジェクトをボタン２でクリックすることによって, 目的のオブジェクトを削除する. オブジェクトの削除と同時に, 接続されているリンクも全て削除される.

- オブジェクトの移動
  オブジェクトをボタン１でドラッグすることによって, 目的のオブジェクトを移動する. オブジェクトの移動と同時に, 接続されているリンクも全て自然な位置に移動される.

- リンクの配置
  二つのオブジェクトの間をボタン２でドラッグすることによって, リンクを配置する. オブジェクトとオブジェクトを結ぶ位置以外にはリンクを配置することはできない.

- リンクの削除
  リンクをボタン２でクリックすることによって, 目的のリンクを削除する.

接続されているオブジェクトには特に影響しない.

- オブジェクトの機能詳細記述ダイアログを開く
  各オブジェクトをボタン１でクリックすることによって, オブジェクト毎に機能詳細記述ダイアログを開く各ダイアログでの操作は次の節で説明する.

オブジェクトの配置を除く, ５つの操作方法に関して, 表3にまとめる.

4.4.2 各オブジェクトの機能詳細記述ダイアログ

各オブジェクトの機能詳細記述ダイアログでは, 対応表を作成することを主な目的とする. ただし, 条件分岐オブジェクトと単純分岐オブジェクトでは対応表の作成は行なわない.

対応表の作成のための操作は, グラフ作成のための操作と似通ったものであり, 画面上の点と点との間を直線で結ぶ操作を繰り返すことで対応表を作成していく. ここでは, 混乱を防ぐために, この点のことをコネクションポイントと呼び, 直線のことをコネクションと呼んで, メインウィンドウにおけるオブジェクト ・リンクと区別する. コネクションポイントは

1 各オブジェクトの入力トークンデータの各データ項目

2 各オブジェクトの出力トークンデータの各データ項目

3 入力オブジェクトのユーザからの入力データ項目

表 3 マウスでの操作方法

操作対象 オブジェクト／リンク	ボタン１		ボタン２	
	クリック	ドラッグ	クリック	ドラッグ
オブジェクト	ダイアログを開く	オブジェクトの移動	オブジェクトの削除	リンクの配置
リンク	-	-	削除	-

4 実行オブジェクトのコマンドへの引数データ項目

5 実行オブジェクトのコマンドの実行結果の各データ項目
ダイアログでは (1) と同一の形状である.

6 中継オブジェクトのフィルタへの入力データ項目
この表示の左端である.

7 中継オブジェクトのフィルタからの出力データ項目
この表示の右端である.

8 定数オブジェクトで定義されているデータ項目
ダイアログでは (3) と同一の形状である.

9 中継 ・定数オブジェクトの変数からの入力

10 中継 ・定数オブジェクトの変数への出力

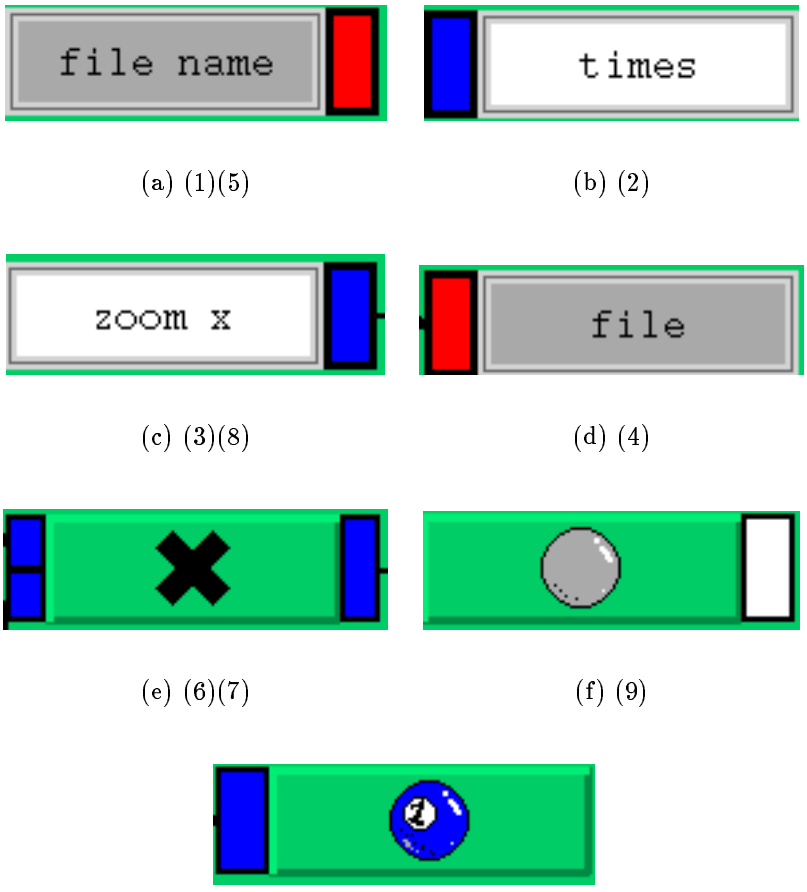
を表現している. これらコネクションポイントのダイアログでの表示を図 18 に示す.

コネクションポイントには色つきの四角形がある. この四角形の色はコネクションポイントが表すデータ項目のデータ型を表現している. データ型と色の対応は表 4 に示す. データ型を色で提示することによって, 各々のデータ項目がどのような型を持っているかが一目で識別できる. また, 各オブジェクトの機能詳細記述ダイアログには, 既にコネクションポイントが存在しており, これを新たに配置 ・削除 ・移動することはできない. プログラマがすべき操作はこのコネクションポイントの間に, コネクションを配置することである. このために,

- コネクションの配置
二つのコネクションポイントの間をボタン 2 でドラッグすることによって, コネクションを配置する. コネクションポイント同士を結ぶ位置以外には, リンクを配置することはできない.

- コネクションの削除
  コネクションをボタン２でクリックすることによって, コネクションを削除する. 接続されているコネクションには, 特に影響しない.

という二つの操作が提供されている.



(a) (1)(5)　(b) (2)

(c) (3)(8)　(d) (4)

(e) (6)(7)　(f) (9)

(g) (10)

図 18　コネクションポイントの凡例

また, コネクション配置における制約は付録 Aに示すオブジェクトの機能詳細記述における対応表の制約に従う. この制約に従わないような配置はできない.

これはメインウィンドウにおいて, 制約に反するリンクの配置ができないのと同様に, 操作方法の習得の容易性を向上させる.

また, 前述したコネクションポイントのうちで (2)(3)(8) については記述されている文字をユーザが変更することが可能である. この内で,(3)(8) については文字変更以外の操作も可能である. この操作については付録 B に示す. また,(1)(4)(5) についてはプログラマが行なうべき操作はない. (6)(7)(9)(10) については付録 B に示す.

また, (1)(4)(5) に表示される文字はユーザが指定することなく自動的に変更されるものである. 同様に (2) についても, ユーザによる明示的な変更とともに, コネクションを張ることによっても自動的に変更される.

これらの文字は, 各データ項目の持つ意味を表している. これによって, ユーザの記述内容への理解を容易にする. また, 自動的に変更されるものも多く, ユーザによる直接のキーボードによる指定は, 最低限に抑えられている.

表 4 データ型と色の対応

色	データ型
青	数値型
赤	文字列型
緑	真偽値型
白	不使用

5. むすび

本論文では, 計算機におけるコマンド発行の自動化について論じ, 新たなコマンド発行の自動化のための手法を提案した. 計算機におけるコマンド発行の自動化にはいくつかの手法があるが, 操作方法習得の容易性 ・記述可能な制御構造の自由度 ・保守編集機能の提供といったコマンド発行の自動化において必要な特徴の全てを完全に満たすものは存在しない. この中で本手法は, 各特徴をバランス良く満たしている.

また, 本手法において必要となる VPL の設計方針について論じ, この方針に基づいて新たに VPL を設計し提案した. VPL はプログラムをアイコン ・図などで視覚的に表示することによって従来の文字型言語に比べて視覚性を向上させている. このために, プログラム経験を持たないユーザにも, 言語の学習が比較的容易である. そして, 提案した VPL では, ループ ・条件分岐などを含む複雑な制御構造を持ったコマンド発行をも記述できることを確認した. しかし, コマンド発行の際に重要な引数の与え方については比較的制限が緩くはなったものの, 可変引数を取ることができないなど, 改善の余地がある.

更に, この VPL 専用の統合プログラミングシステムについて設計方針を述べ, その実装について説明した. このシステムに含まれるエディタでは, エラー箇所指摘機能やグラフ記述への特化などの特長によって, 容易にコマンドマクロを記述することができる. また, トレーサでは記述されたコマンドマクロの解析の過程を示すことで更に理解を深める工夫を行なった.

本研究では, この統合プログラミングシステムを UNIX 上において実装した. これは UNIX 上では既にコマンドマクロに利用することを目的とした, 質の高い各種のツールが提供されているためである. しかし, プログラミング経験を持たないユーザの多くは主に Machintosh や MS-WINDOWS などの OS を利用しており, これらの OS への移植も行なっていく必要がある.

謝辞

本研究の全過程を通して, 直接懇切なる御指導, 御鞭撻を賜ったソフトウェア基礎講座 横矢 直和教授 に衷心より感謝の意を表します.

本研究の遂行にあたり, 終始有益な御助言と御鞭撻を頂いた像情報処理学講座 千原 國宏教授, 並びにソフトウェア基礎講座 竹村 治雄助教授 に厚く御礼申し上げます.

さらに, 日頃より様々な御指導と御助言を頂いた大阪大学基礎工学部情報工学科 萩原 兼一教授 に感謝します.

そして本研究を通じて, 有益な御助言を頂いたソフトウェア基礎講座 岩佐 英彦助手, 並びに情報科学センター 片山 喜章助手 に厚く感謝します.

最後に, 物心両面において常に温かい御助言を頂いた, ソフトウェア基礎講座の諸氏, ならびに, ソフトウェア基礎講座事務補佐員 村上 和代嬢に深く感謝します.

参考文献

[1] 山口 和紀: The UNIX Super Text （上巻）, 第 30 章, pp.503–514, 技術評論社, 1992.

[2] S.K.Chang: "Visual languages: A tutorial and survey", IEEE Software, 4, 1, pp.29–39, 1987.

[3] Margaret M. Burnett and Marla J. Baker: "A classification system for visual programing language", Technical Report93-60-14, Oregon State University Dept of Computer Science, 1993.

[4] Margaret M. Burunett and David W.McIntyre: "Visual programing", IEEE Computer, 28, 3, pp.14–16, march 1995.

[5] Ephrain P.Gilnert and Steven L.Tanimoto: "Pict: An intertactive graphical programing enviroment", IEEE Computer, pp.7–25, November 1984.

[6] 長崎 祥, 田中 譲: "シンセティック ・メディアシステム:intelligentpad", コンピュータソフトウェア, 11, 1, pp.36–48, 1994.

[7] 村田 忠夫: ペトリネットの解析と応用, 第 2 章, pp.18–19, アルゴリズム ・シリーズ, 5, 近代科学社, 1992.

[8] Stefan Schiffer and Joachim Hans Fröhlich: "Concepts and Architecture of Vista - a Multiparadigm Programming Enviroment", Proc. IEEE Visual Programing, pp.40–47, 1994.

[9] G. Wirtz: "Modularization and Process Replication in a Visual Parallel Programming Language", Proc. IEEE Visual Programing, pp.72–79, 1994.

[10] T. D. Kimura: "Object-oriented dataflow", Proc. IEEE Visual Programing, pp.180–186, 1995.

[11] Margaret-Anne Halse: Iris Explorer, 1993.

[12] Microsoft: Microsoft Windows 機能ガイド, 第 12 章, pp.658–676, マイクロソフト社, 1993.

[13] Microsoft: Microsoft MS-DOS 6.2/V ユーザーズガイド, マイクロソフト社, 1993.

[14] B. A.Myers: "Visual programing, programing by example, and programing visualization: A taxonomy", Proc. ACM Conf. on Human Factors in Computing Systems (CHI '86), pp.59–66, 1986.

[15] A. Cypher: "Eager:programing repetitive tasks by example", Proc. ACM Conf. on Human Factors in Computing Systems (CHI '91), 1991.

[16] F.Modugno: "A state-based visual language for a demonstrational visual shell", Proc. IEEE Visual Programing, pp.304–311, 1994.

[17] Hugh Glaser and Trevor J. Smedley: "Psh - the next generation of command line interfaces", Proc. IEEE Visual Programing, pp.29–36, 1995.

[18] Prograph International: Prograph CPX User's Guide, 1993.

付録

A. 各オブジェクトの機能詳細記述における文法規則

ここでは, 各オブジェクト毎に機能詳細記述の文法規則に関して説明する.

A.1 入力オブジェクト

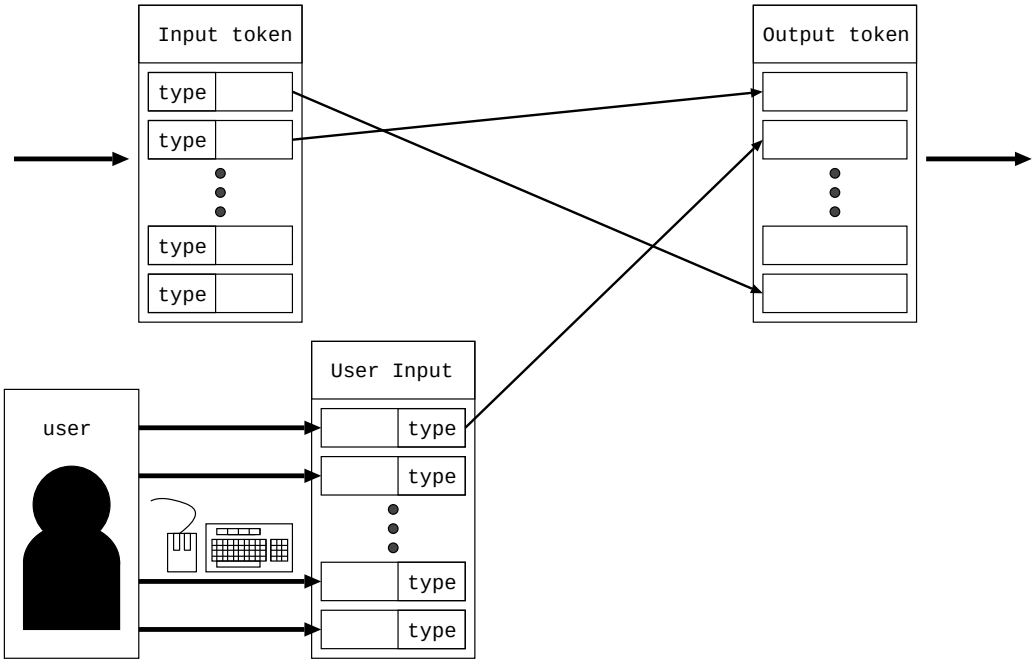


図 19 入力オブジェクトの概念図

入力オブジェクトは, プログラムの実行時にユーザからの入力を受け付けることを表現している. このときに, ユーザから受け付けるデータは表 5のような形式である.

ここで, ユーザから入力されたデータの各項目がどのデータ型を持つかを定義する必要がある. このために, ユーザからの入力データのそれぞれの項目について, データ型を数値型 ・文字列型 ・真偽値型から選択する. これによって, ユーザ

から入力されたデータの各項目は, トークンデータにおけるデータ項目と同じ形式になる.

また, 最終的に入力オブジェクトから出力されるトークンデータの各データ項目は,

- 入力オブジェクトへと受け渡されたトークンのデータ項目
- ユーザからの入力データに, 入力オブジェクトで定義されたデータ型を付与したもの

のうちの一つをコピーしたものとなる.

このコピー元とコピー先の対応をつけるために対応表を作成する必要がある. この対応表では,

- コピー先には対応するコピー元がなくてもよい
- コピー先に対応するコピー元がある場合には, それは唯一である.

という条件を満たしている必要がある. 逆に, コピー元には対応するコピー先がなくても構わないし, 複数のコピー先が対応してもよい.

A.2 実行オブジェクト

実行オブジェクトは, 記述されたコマンドマクロの解析・実行時に, コマンド発行を行なうことを表現している.

表 5 ユーザ入力のデータ形式

項目番号	項目の内容
1	データ内容 1
2	データ内容 2
⋮	⋮
n^5	データ内容 n

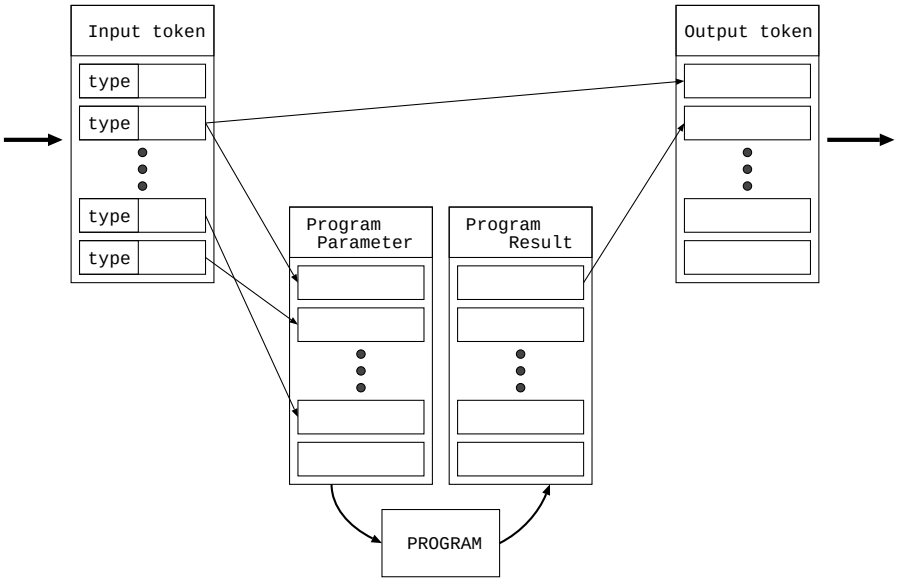


図 20　実行オブジェクトの概念図

実行オブジェクトでは発行可能な全てのコマンドの中から, 一つのコマンドを選択する必要がある.

コマンドが選択されると同時に, そのコマンドの引数の数と, 引数として数値型 ・文字列型 ・真偽値型のうちのどの型のデータを必要としているかが決定される.

また, 逆にコマンドの実行結果として, どのような型のデータがどれだけ出力されるかも決定される. この実行結果の各データは, 型を含めてトークンデータのデータ項目と同じ形式のものである.

ここで, コマンドの各引数には, 実行オブジェクトへと渡されたトークンデータのデータ項目の一つがコピーして渡される. このトークンデータのデータ項目と各引数との対応表は, 44ページにおいて述べた入力オブジェクトの対応表の制約と同じ制約を満たしている必要がある.

最終的に実行オブジェクトから出力されるトークンデータの各データ項目は,

- 実行オブジェクトへと受け渡されたトークンのデータ項目
- コマンドの実行結果の各データ

のうちの一つをコピーしたものとなる．この対応表も同じ制約を満たす必要がある．

A.3 中継オブジェクト

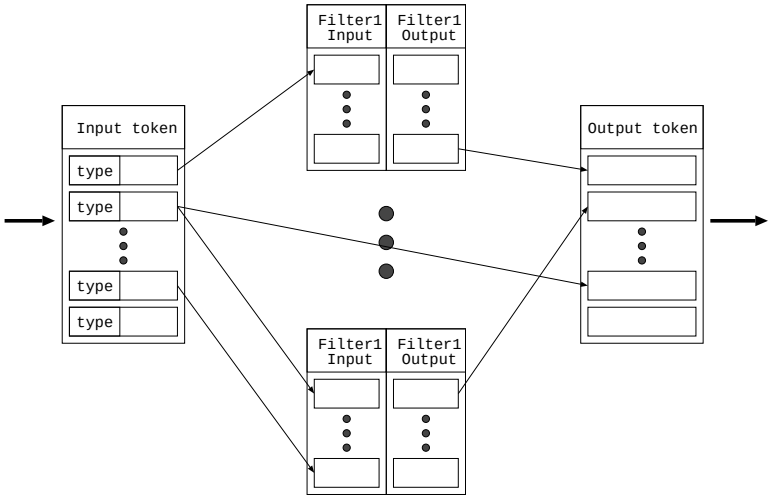


図 21 中継オブジェクトの概念図

中継オブジェクトは, 受け渡されたトークンデータの各データ項目を加工して, 出力するトークンデータの各データ項目を作成する機能を持つ．

データ項目をどのように加工するかを決定するために, ここでフィルタと呼ぶ一種の関数を選択する．フィルタは一つのオブジェクトに複数定義することが可能である．フィルタは各々に決まった数のデータ項目を受けとって, 決まった数のデータ項目を出力する.[6]

このフィルタでは一例として以下のような関数を用意している．

- 四則演算
- キャスト

[6]実装した統合プログラミング環境では理解容易性のために, フィルタが受けとるデータ項目の数は１または２に, 出力するデータ項目は１つに限定している

- ブール演算
- 文字操作関数

いずれのフィルタにせよ, フィルタを選択した段階で, その必要な入力データの数と入力データとしてどのような型を必要としているかが決定される. また, そのフィルタからの出力データの型も一意に決定される.

ここで, フィルタへの引数として与えられる各データ項目には, オブジェクトへと渡されたトークンデータのデータ項目の一つがコピーされる. このトークンデータのデータ項目とフィルタの引数との対応表は, 44ページにおいて述べた入力オブジェクトの対応表の制約と同じ制約を満たしている必要がある. また, さらに付け加えてコピー元とコピー先の両者のデータ型が一致している必要がある.

また, 最終的に中継オブジェクトから出力されるトークンの各データ項目は,

- 中継オブジェクトへと受け渡されたトークンのデータ項目
- フィルタの出力結果

のうちの一つをコピーしたものとなる. このコピー元とコピー先の対応をつけるために対応表を作成する必要がある. この対応表は上記の制約と同じ制約を満たしている必要がある. ただし, データ型の一致は必要ではない.

A.4 定数オブジェクト

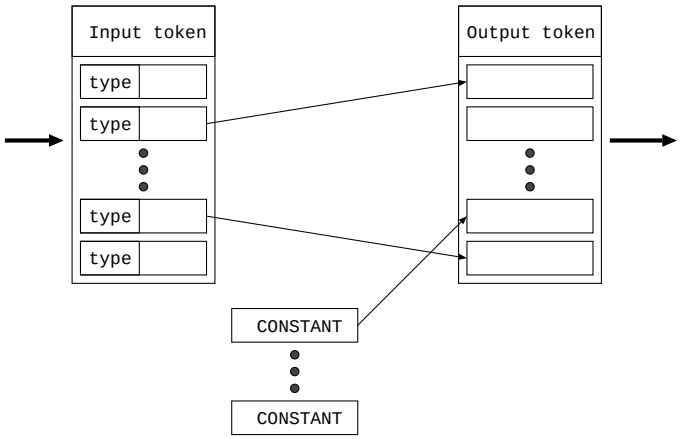


図 22 定数オブジェクトの概念図

定数オブジェクトは, オブジェクトに予め定義されている数値 ・文字列 ・真偽値をトークンデータへと付け加える動作を表現している.[7]

ここで, 予め定義されているデータ項目に関してそのデータ型とデータ内容を定義する必要がある. このために, 定義しておく各データ項目毎にデータ型を数値型 ・文字列型 ・真偽値型から選択する. また, データ内容に関しても定義しておく.

また, 最終的に定数オブジェクトから出力されるトークンデータの各データ項目は,

- 定数オブジェクトへと受け渡されたトークンデータのデータ項目
- 定数オブジェクトで定義されているデータ項目

のうちの一つをコピーしたものとなる. このコピー元とコピー先の対応をつけるために対応表を作成する必要がある. この対応表は, 44ページにおいて述べた入力オブジェクトの対応表の制約と同じ制約を満たしている必要がある.

[7] 丁度,C 言語におけるハードコーディングに相当する機能である

A.5 単純分岐オブジェクト

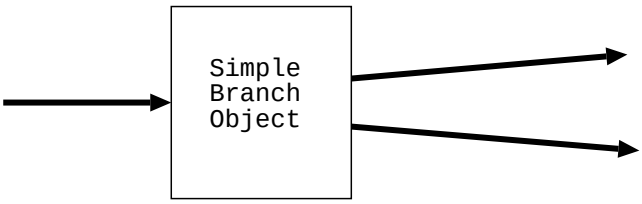


図 23 単純分岐オブジェクトの概念図

単純分岐オブジェクトでは, 受けとったトークンデータをそのまま全ての出力リンクへと送り出す.

このオブジェクトを利用することで, コマンドマクロの制御の並列化を記述することが可能である.

このオブジェクトでは, 特に定義することはない.

A.6 条件分岐オブジェクト

条件分岐オブジェクトでは, 受けとったトークンデータのデータ項目を元に, 全ての出力リンクの中から一つを選びだし, そのリンクへと受けとったトークンデータをそのままの形で送る. 選ばれなかったリンクにはトークンデータは送られない.

このオブジェクトを利用することで, コマンドマクロの流れの制御を記述することが可能である.

このオブジェクトでは, 受けとったトークンデータのどのデータ項目を分岐の条件とするかを選択する必要がある. このデータ項目としてはデータ型が真偽値型のもののみを選択することができる.

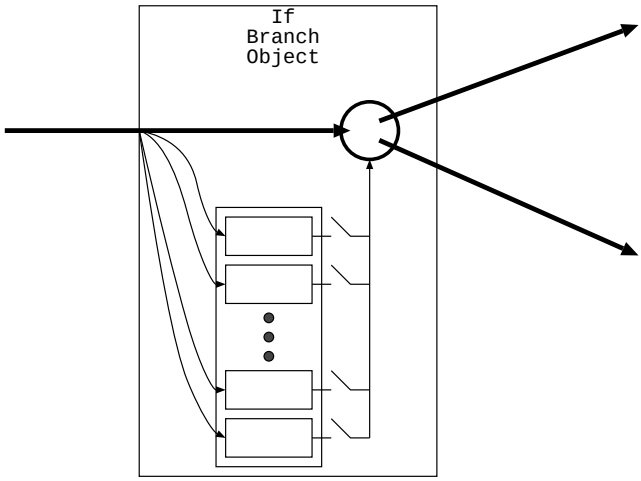


図 24　条件分岐オブジェクトの概念図

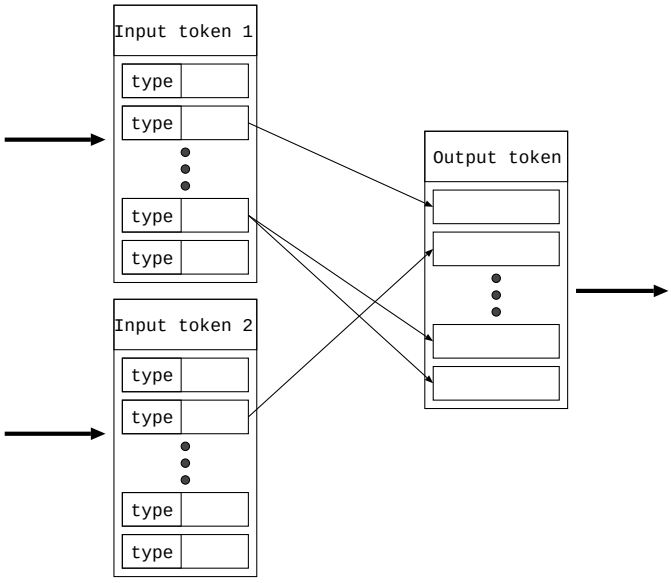


図 25　AND バリアオブジェクトの概念図

A.7 AND バリアオブジェクト

AND バリアオブジェクトは, 入力される全てのトークンデータが揃うのを待って, その全てのトークンデータを編成して次のオブジェクトへとトークンデータを送る. このために, このオブジェクトを利用することで, コマンドマクロの並列制御を記述することが可能である.

AND バリアオブジェクトから最終的に出力されるトークンの各データ項目は,

- AND バリアオブジェクトへと受け渡された全てのトークンデータのデータ項目

から一つをコピーしたものとなる. このコピー元とコピー先の対応をつけるために対応表を作成する必要がある. この対応表は 44ページにおいて述べた入力オブジェクトの対応表の制約と同じ制約を満たしている必要がある.

A.8 OR バリアオブジェクト

OR バリアオブジェクト, 全ての入力リンクのうちの一つからでもトークンデータを受けとると, そのトークンデータから新たにトークンデータを編成して, 次のオブジェクトへと送る. このときに, 出力されるトークンデータの各データ項目のデータ型は, 揃えられる. このために,OR バリアオブジェクトは様々な流れ制御に利用される.

OR バリアオブジェクトから最終的に出力されるトークンの各データ項目は,

- オブジェクトに受け渡されたトークンデータのデータ項目

の一つをコピーしたものとなる. このコピー元とコピー先の対応をつけるために対応表を作成する必要がある. この対応表は

- コピー先には対応するコピー元がなくてもよい
- コピー先に対応するコピー元がある場合には, 入力される各トークンデータ毎に一つのデータ項目がそれに対応し, それは各トークン毎に唯一である.
- コピー先に対応する全てのコピー元は全て同じデータ型である必要がある

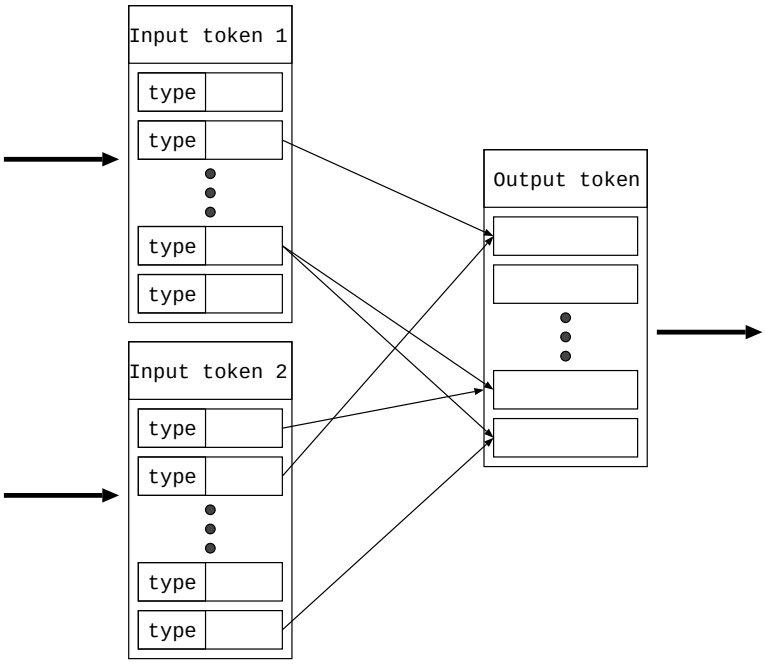


図 26 OR バリアオブジェクトの概念図

B. オブジェクト機能詳細記述ダイアログの実装

ここでは, 各オブジェクトの機能詳細記述ダイアログの実装について説明する.

B.1 入力オブジェクト機能詳細記述ダイアログ

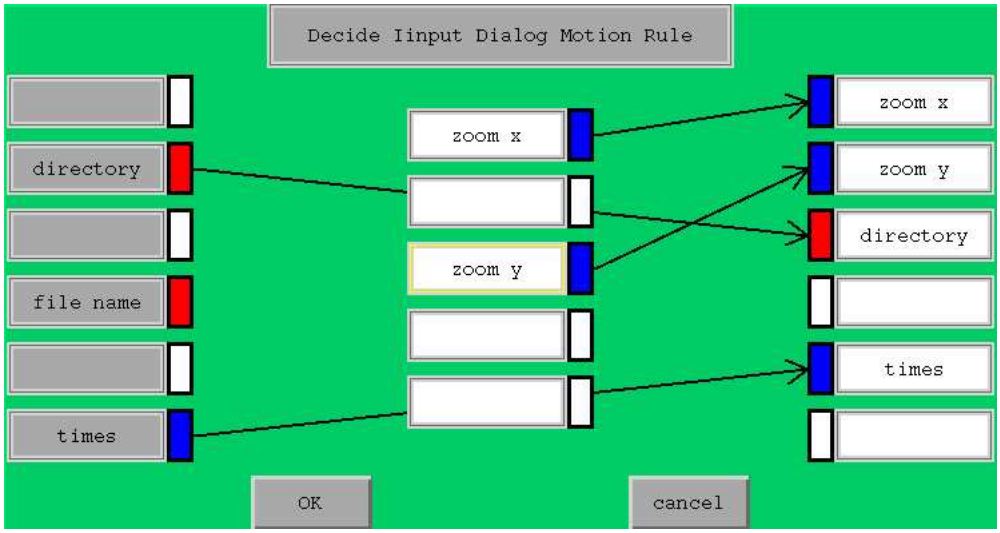


図 27 入力ダイアログの画面

入力オブジェクト機能詳細記述ダイアログ (入力ダイアログと略す) の画面を図 27に示す.

入力ダイアログにおいては, ユーザからの入力データ項目を表すコネクションポイントのデータ型を決める必要がある.

このためには, 入力データ項目を表すコネクションポイントをマウスのボタン２でクリックする. 一度クリックすることに色が変更され,

→ 数値型 → 文字列型 →
真偽値型 → 不使用 →

の順に変わっていく.

B.2 実行オブジェクト機能詳細記述ダイアログ

実行オブジェクト機能詳細記述ダイアログ（実行ダイアログと略す）の画面を図 28に示す.

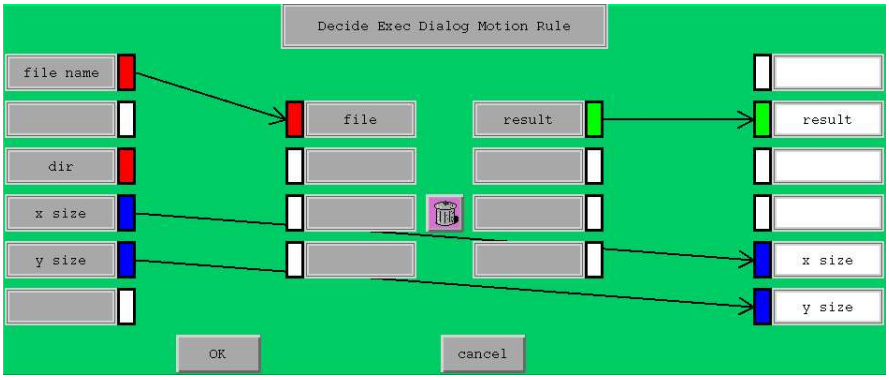


図 28 実行ダイアログの画面

実行ダイアログにおいては, そのオブジェクトが表すプログラムを決定する必要がある. このためには, ダイアログ中央のボタンを押して, プログラム選択ダイアログを開く. このプログラム選択ダイアログの画面を図 29に示す.

図 29 プログラム選択ダイアログの画面

B.3 中継オブジェクト機能詳細記述ダイアログ

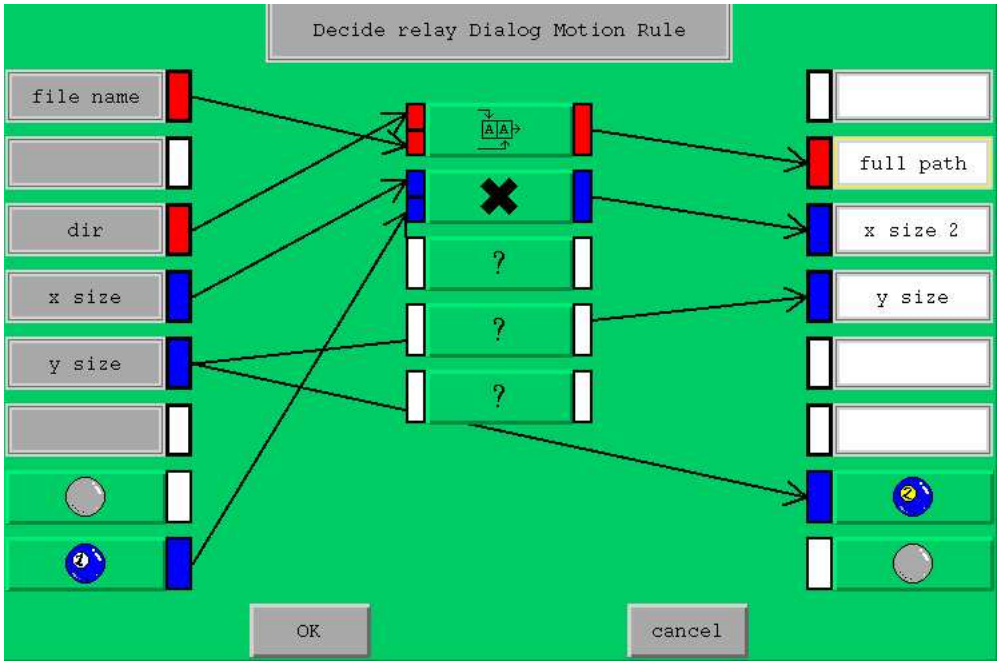


図 30　中継ダイアログの画面

中継オブジェクト機能詳細記述ダイアログ　(中継ダイアログと略す）の画面を図 30に示す.

中継ダイアログにおいては, オブジェクトにおいてデータを加工するフィルタの種類を決定する必要がある. このためには, ダイアログ中央列のボタンを押して, フィルタ選択ダイアログを開く. このフィルタ選択ダイアログの画面を図 31に示す.

また, 中継ダイアログでは, 変数への代入 ·変数からの値の取り出しを行なうことができる. そのために, どの変数への操作なのかを決定する必要がある. このためには, ダイアログの左下 ·右下にあるボタンを押して, 変数選択ダイアログを開く. この変数選択ダイアログの画面を図 32に示す.

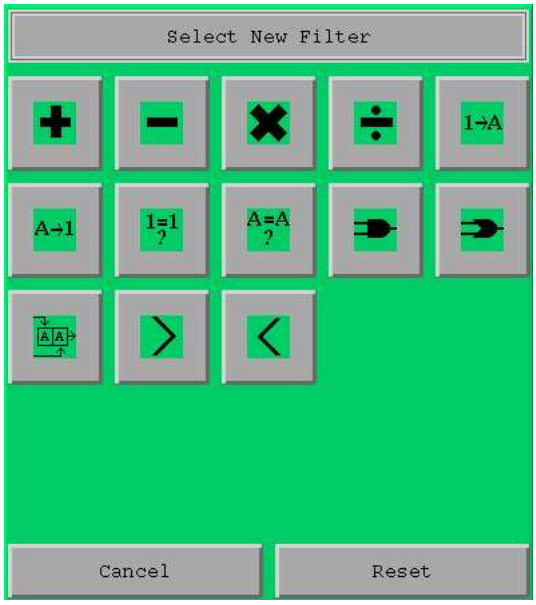


図 31　フィルタ選択ダイアログの画面

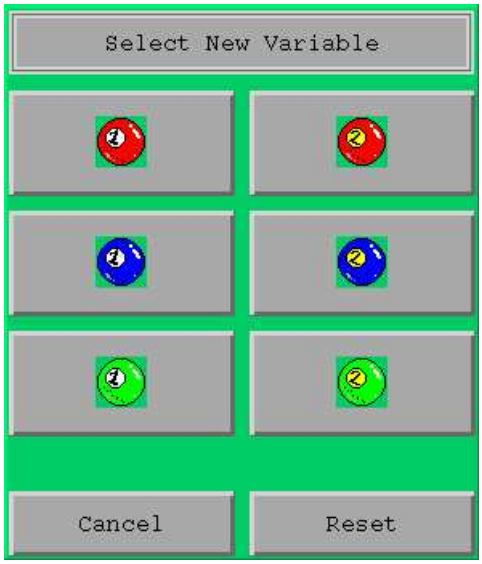


図 32　変数選択ダイアログの画面

B.4 定数オブジェクト機能詳細記述ダイアログ

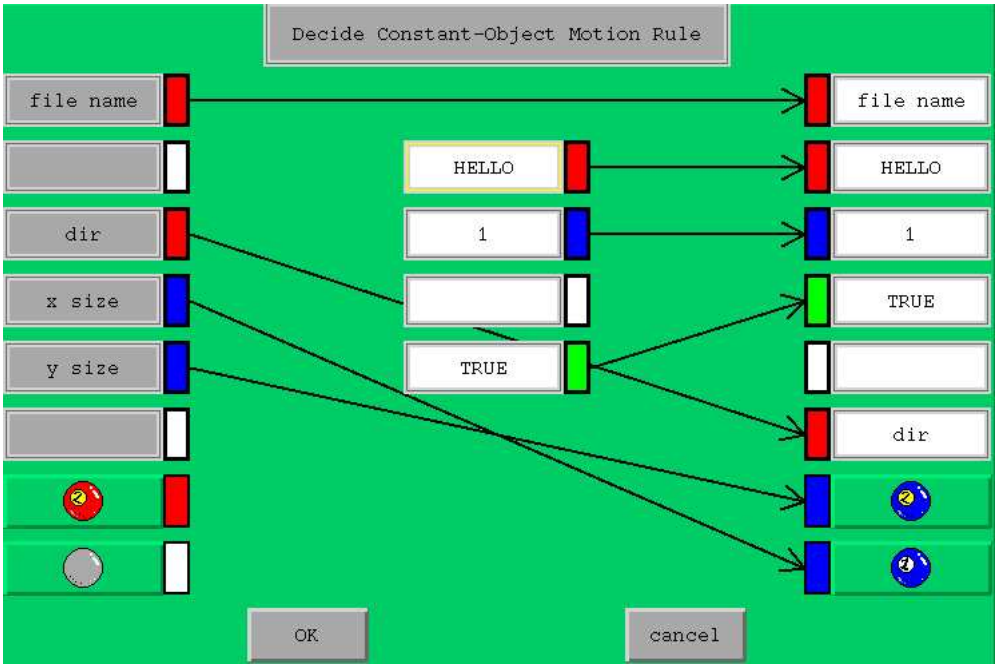


図 33 定数ダイアログの画面

定数オブジェクト機能詳細記述ダイアログ　定数ダイアログと略す）の画面を図 33に示す.

定数ダイアログにおいては, 数値 ・文字列 ・真偽値のデータを必要に応じて定義する. このために, まず定数データのデータ型を決定する.

このためには, 入力オブジェクトにおける入力データ項目のデータ型の設定と同様の操作で, データ項目を表すコネクションポイントのデータ型を変更する. さらに, データ内容をキーボードによって入力する.

また, 変数に関しては中継ダイアログでの操作と同様である.

B.5 条件分岐オブジェクト機能詳細記述ダイアログ

条件分岐オブジェクト機能詳細記述ダイアログ　条件分岐ダイアログと略す）の画面を図 34に示す.

図 34　条件分岐ダイアログの画面

条件分岐ダイアログでは, 分岐の際の条件となるデータ項目を選択する．ダイアログには条件分岐オブジェクトへと入力されるトークンデータの内でデータ型が真偽値となるデータ項目のみが表示される．このうちから一つを選択してボタンを押す.

B.6 AND バリアオブジェクト機能詳細記述ダイアログ

AND バリアオブジェクト機能詳細記述ダイアログ （AND バリアダイアログと略す）の画面を図 35に示す.

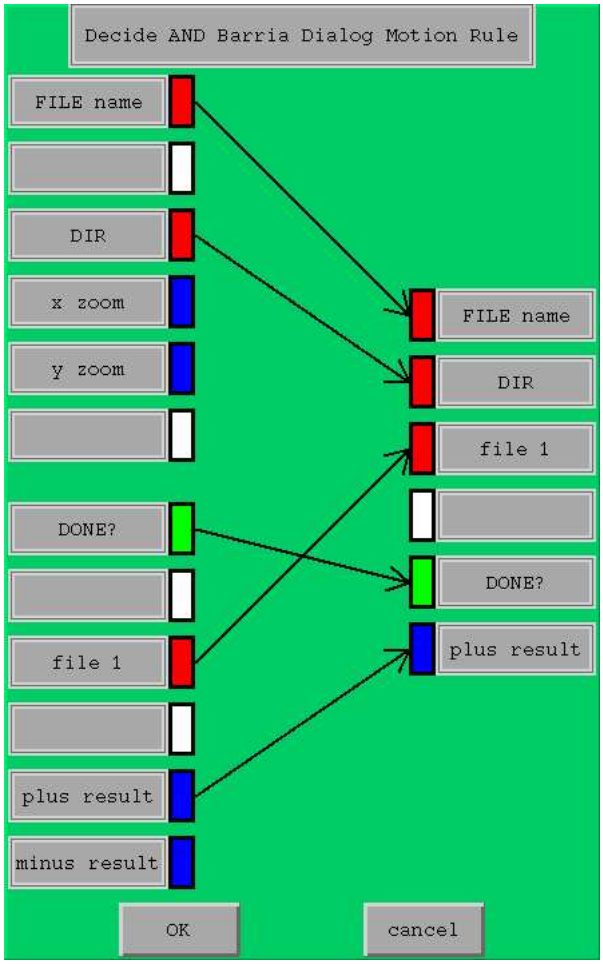


図 35 AND バリアダイアログの画面

AND バリアダイアログでは, 特に固有の操作はない.

B.7 OR バリアオブジェクト機能詳細記述ダイアログ

OR バリアオブジェクト機能詳細記述ダイアログ （OR バリアダイアログと略す）の画面を図 36に示す.

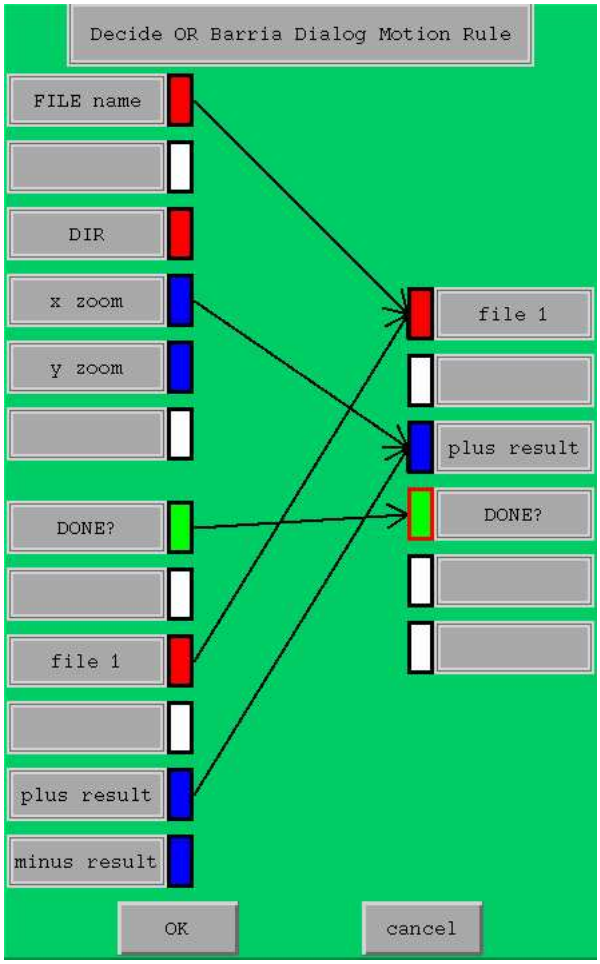


図 36 OR バリアダイアログの画面

OR バリアダイアログでは, 特に固有の操作はない.