ソフトウェア工学 107−20
(1996. 1. 19)

# ネットワークソフトウェア開発のためのコンポーネントウェア *PARTY* の設計

平野 純二
南山大学 情報管理学科
*92b514@iq.nanzan-u.ac.jp*

野呂 昌満
南山大学 情報管理学科
*masami@iq.nanzan-u.ac.jp*

ネットワーク上の1つの実行単位(オブジェクト)を1つのコンポーネント(部品)ととらえ,それらを組み合わせることでネットワークソフトウェアを作成可能なコンポーネントウェア *PARTY* の設計をする.ネットワークソフトウェアは,ネットワーク上に分散したオブジェクトが協調しながら実行されてゆくので,同期通信・非同期通信などの通信の種類を考慮しなければならない.*PARTY* では,部品を組み合わせるさいに,通信の種類を使用者が決定できるようにする.また,型の不一致や違法な通信を,*PARTY* がソフトウェア作成中に見つけ警告することで,対話式な開発環境を提供する.本稿では,これ以外の設計において留意した点も述べる.

# Design of a componentware *PARTY* for network software construction

Junji Hirano
Department of Information System and Quantitative Sciences , Nanzan University

Masami Noro
Department of Information Systems and Quantitative Sciences ,Nanzan University

This paper describes design issues of a componentware *PARTY* which supports development of network software. In *PARTY*, an object which is an entity concurrently executed is a component towards network software construction. Severals of salient features of *PARTY* are component linkage handling and interactive consistency checking. *PARTY* handles linkages among objects for representing various types of communication the objects have. Interactive detection of illegal communication and typing violation will help easy network software construction.

## 1 はじめに

本研究の目的は, ネットワークソフトウェアが容易に作成できるコンポーネントウェアを設計することである. プリンタやディスク装置などの資源の共有を目的にネットワークを結ぶことが増えてきた. それとともに, ネットワークソフトウェアの需要が増えてきた. ネットワークソフトウェアを作成することは, プロトコルに関する知識や, OS が提供する機能に関する知識が必要であるので難しい. 我々は視覚的でわかりやすく, コードを記述することなくネットワークソフトウェアが作成可能なコンポーネントウェアの設計をする.

コンポーネントウェアとは,"コンポーネント"と呼ばれる部品を組み合わせることで, 応用ソフトウェアの作成が可能な視覚的開発環境である. ネットワークソフトウェアを作成可能なコンポーネントウェアの設計の基本方針は, 1 つの実行単位 (オブジェクト) を 1 つの部品とみなし, それらを組み合わせることでネットワーク上の関係を記述できるようにすることである.

*PARTY* の設計にあたり, 我々はコンポーネントウェアと呼ばれるものを調査し, コンポーネントウェアに必要とされる機能をまとめた. 設計するコンポーネントウェアのもとになる言語 (Visual Basic では Basic) は, 強い型づけ言語がよいと思った. これは, ソフトウェア作成時に型違いというフォールトを発見できるからである. 我々は,*PARTY* のもとにある言語を

- 強い型づけ言語
- ネットワークソフトウェアの作成を支援する言語

である $y$[5] とした.

部品を組み合わせる方法に,PARTS Workbench というコンポーネントウェアにある "リンク" という考えを取り入れる. PARTS Workbench では, コードを記述することなく, 部品を組み合わせることができるからである. さらに *PARTY* では, 通信の種類を決定できるようにすることと, リンクの種類を増やすことで, ネットワークソフトウェアの作成を可能にする.

*PARTY* で使用される部品は, 実行速度の向上とデータの抽象化を目的に, "逐次部品" とすることができる. また,*PARTY* では部品によって配置できる場所が異なるので視覚部品・非視覚部品・ウィンドウ部品に分類する.

我々は *PARTY* 自体も複数のオブジェクトから構成されているととらえ, 設計仕様書を OMT 法が示す図式を使って作成した. OMT 法では, 図式が豊富に提供されているので, 複数の開発者に設計の内容が正確に伝えることができる. これらの図式を利用すると, 分かりやすく, 変更しやすい設計書ができると判断したからである.

## 2 現在のコンポーネントウェア

*PARTY* の設計にあたり, 現在どのようなコンポーネントウェアがあるかを調査した. 調査したコンポーネントウェアは, 次に挙げる 3 つである.

**Intelligent Pad:** Sun OS 上で実現されている数少ないコンポーネントウェアの 1 つであり, 無償ソフトウェアとして提供されるので, 入手することができ調査した.

**Visual Basic:** 広く使われていて一般的であるので調査した.

**PARTS Workbench:** すべての部品も視覚化されており, コードを記述しなくても応用ソフトウェアが作成可能であるという特徴があるので調査した.

3 つのコンポーネントウェアの調査をもとに, コンポーネントウェアに必要とされる機能をまとめた. 3 つのコンポーネントウェアにはそれぞれ特徴がある. 我々はこれらのコンポーネントウェア相違点をまとめた. そしてコンポーネントウェアに必要とされる機能や, 便利だと思う機能を *PARTY* に取り入れたい機能としてまとめた.

### 2.1 Intelligent Pad

Intelligent Pad では, 部品は Pad と呼ばれている. Pad を組み合わせることで, コードを記述することなくソフトウェアを作成できる. 使用者が Pad を組み合わせて作ったものを新しい Pad として登録することができる. Intelligent Pad で作成したソフトウェアは,Intelligent Pad 上でのみ実行可能である. Pad を配置した時点でその Pad は実行を開始するので, 動的に部品を組み替えることが可能である.

コードを記述することなくソフトウェアを作成できるが, 用意されている部品の種類が少ないので本格的なソフトウェアを作成することは難しい. また, 作成したソフトウェアは Intelligent Pad 上でしか実行できないという欠点がある.

### 2.2 Visual Basic

Visual Basic では "フォーム" と呼ばれるウィンドウ上で "コントロール" とよばれる部品を組み合わせることでソフトウェアを視覚的に作成することができる. Basic でコードを記述して部品を組み合わせる. ソフトウェア作成中はインタプリタとして作成したソフトウェアを実行することができるので, ステップ実行などのディバッグ機能が充実している.

Visual Basic では, 部品の組合せのさいに,Basic でコードを記述しなければならない. Basic ではコンパイラの

制約により，部品の数や個数に制限があるので，大きなソフトウェアを作成することができない．

## 2.3 PARTS Workbench

部品をリンクという作業で組み合わせるだけで，コードを記述することなく応用ソフトウェアを作成することができる．使用者が部品を組み合わせて作ったものはネスティッドパートと呼ばれ，新しい部品として登録することができる．整数や文字列なども部品として登録されており，ソフトウェア作成中は視覚化されている．部品間の関係も線で表示されるのでわかりやすい．

PARTS Workbench は，もとになる言語が型なし言語(Smalltalk)であるので，型検査が行なわれない．型違いによるフォールトをソフトウェア作成時に検出できないので，故障箇所の特定に時間がかかると考えられる．我々はPARTS Workbench で，自動販売機のシミュレータを作成した．そのさい，比較的大きなソフトウェアでは，型検査機能があった方がよいと感じた．

## 2.4 コンポーネントウェアに必要とされる機能

3つのコンポーネントウェアの相違点をまとめた．それぞれのコンポーネントウェアは部品の組合せの方法などに違いがある．その中で我々が便利だと思った機能や必要だと思った機能を *PARTY* にとり入れたい機能としてまとめた．

表 1: 相違点

| | IP[1] | PW[2] | VB[3] |
|---|---|---|---|
| マウスによる部品の組み合わせ | ○ | ○ | × |
| 応用ソフトウェア作成時の部品同士の関係の表示 | × | ○ | × |
| 使用者が作成した部品を新しい部品として登録 | ○ | ○ | × |
| もとになる言語 | C++ | Smalltalk | Basic |
| 実行可能ファイルの作成 | × | ○ | ○ |

○：できる　　×：できない

3つのコンポーネントウェアを調査した結果，以下に述べる機能が必要であると考え，*PARTY* に取り入れたい．

- マウスを使用し，部品間に線を引くことで部品の組み合わせができる機能
  - ・コードを記述しなくてもよいので簡単に部品の組み合わせができる
- 応用ソフトウェア作成時の部品間の関係を線で表し視覚的に表現する機能
  - ・部品間の関係を視覚的にすることで部品の関係が一目で分かる
- 部品を組み合わせて新たな部品として登録できる機能
  - ・新しい部品を容易に作成することができる
- 部品倉庫に登録できる機能
  - ・部品の整理ができ，探索が速くなる
- 型検査機能
  - ・故障箇所が発見しやすい

# 3 *PARTY* の仕様

3つのコンポネントウェアの調査をもとに，*PARTY* の仕様を決定した．はじめに，プログラミング言語で記述するよりも簡単にネットワークソフトウェアが作成できるように，設計指針を決めた．そして，基本指針に基づき *PARTY* のリンク，プロパティ部品そしてコード生成に関する仕様を決定した．

## 3.1 設計指針

簡単にネットワークソフトウェアが作成できるように，*PARTY* では，

- 視覚的でわかりやすく，
- コードを記述することなく，

ソフトウェアが作成できる機能を取り入れる．

ソフトウェア作成中に視覚化するものは，すべての部品，および部品と部品の関係である．PARTS Workbench では，これらのものは視覚化されている．我々がPARTS Workbench を使用したさい，部品の関係が明確でわかりやすいと感じた．我々も *PARTY* でもこれらのものを視覚化する．

*PARTY* では，ネットワークソフトウェアの作成を可能にする．そのため *PARTY* では，

- 1つの実行単位をアイコンとして表示し，
- ネットワーク上での実行単位の関係を線で表示する

機能を取り入れる．

コードを記述することなくソフトウェアが作成できるように，

[1]Intelligetn Pad
[2]PARTS Workbench
[3]Visual Basic

- ネスティッドパートを作成できる
- マウスで線を引くことで部品と部品を関係づけられる

ことを *PARTY* では可能にする.

*PARTY* のもとになる言語を強い型づけ言語とする. 比較的大きなソフトウェアでは, 型違いから起こる故障箇所を実行時に特定することは難しい. 我々は, 型違いをソフトウェア作成時に見つることができた方が良いと判断し, *PARTY* のもとになる言語を強い型づけ言語とした.

*PARTY* ではもとになる言語を $y$ とした. $y$ は, ネットワークソフトウェアの作成を可能にすることを目的として, 開発された言語であり, 強い型づけ言語だからである.

設計した内容を,OMT(Object Modeling Technique) で示されている図式を使ってまとめた (4 章). OMT でいわれている図式を利用した理由は以下の 2 つである.

- OMT では, 開発結果として豊富なドキュメントを用意する. これにより複数の開発者に開発結果を正確に伝えることができる.
- 仕様の変更にともなうドキュメントの変更箇所が発見しやすい

## 3.2 リンク

"リンクする"という作業で, ある部品でイベントが起こった時に他の部品にメッセージを送る, ということを関係づけることができるようにする. リンクにより部品を組み合わせるという考えは PARTS Workbench から取り入れた. PARTS Workbench では, コードを記述することなく, 部品を組み合わせることとができるからである.

*PARTY* では, ネットワーク上で分散して存在するオブジェクト同士の通信も, リンクにより表現することができるようにする. ネットワークソフトウェアは, ネットワーク上のいくつかのオブジェクトが協調して実行されていくので, オブジェクト間通信の種類には同期・非同期などがあるといわれている, *PARTY* ではリンクを作るさいに通信の種類を決定できるようにする.

リンクの種類には PARTS Workbench で提供されているものを取り入れた. しかし,*PARTY* では,PARTS Workbench で提供されているリンクの種類だけでは, たりないので, リンクの種類にキューリンクを加えた.

型違いによるフォールトをリンク時に検出できるように, 型検査機能を取り入れる. メッセージの引数の型と,"引数リンク" をした結果得られる値の型に矛盾がないか検査する.

**通信の種類**

通信の種類には, 同期通信, 非同期通信があるといわれている. *PARTY* での通信の種類を考慮しなければならない.
*PARTY* では,

- メッセージ通信
- 同期通信
- 非同期通信

をリンクのさいに選ぶことができるようにする.

メッセージ通信とは, ある部品にイベントが起こった時に, 他のある部品にメッセージを送る場合に使用する. 同期通信との違いはメッセージ通信でリンクをした結果, リンク元の部品とリンク先の部品が逐次部品 (3.4 章) となることである.

使用者が同期通信, 非同期通信のリンクを選択した場合, そのリンクを

- 通常通信
- 優先通信

にするかどうか使用者が決定することができる. 選択したリンクを優先通信にした場合, 他の通信によるメッセージよりも優先してメッセージを送ることができる. これらの通信も使用者がリンクのさいに決定することにより, *PARTY* 簡単に実現できる.

**リンクの種類**

PARTS Workbench には, 以下のリンク種類が用意されていた.

- イベントリンク

  ある部品にイベントが起きた時, 他の部品にメッセージを送るということを表すリンク.
- 引数リンク

  メッセージの引数を得るさいに使用されるリンク. 他の部品にメッセージを送り, 送られた先の部品が返すオブジェクト (戻り値) が引数となる.
- 結果リンク

  ある 1 つの部品にイベントが起こった時に 2 つ以上のメッセージを送りたい場合に, その順序を明確にする目的で使用される.

引数を指定することや, メッセージの順序を使用者が指定できる機能が必要である考えたので, これらのリンクの種類を *PARTY* にも取り入れた.

*PARTY* ではさらにキューリンクというリンクの種類を取り入れる. *PARTY* では非同期でメッセージが送られ

る場合がある．非同期でメッセージを送った結果を保持するものを指定する必要があるので，*PARTY* では，キューリンクにより結果を保持するものを指定できるようにする．

結果を保持するところは，*PARTY* では結果キューという部品である．結果キューは並行オブジェクトである．使用者は非同期通信でリンクをした場合は，結果キューにキューリンクすることで，結果を保持するところを決定する．その結果が必要になったさいには，結果キューにメッセージを送り，結果をもらうことができる．

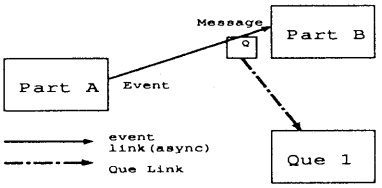


図 1: キューリンク

*PARTY* では引数リンクが行われる時に，型の検査を行なう．引数リンクによって呼び出そうとするメソッドの戻り値の型が，引数の型と合わない場合は，リンクができないということを使用者に警告する．これは，実行時には見つけにくい間違えをソフトウェア作成時に検出することで，ディバッグの時間を減らすことが目的である．

*PARTY* では型名が一致しないときでも代入可能な場合がある．*PARTY* のもとになる言語である $y$ では，C++ と同様スーパークラスへ代入が可能である．また，型変換演算子により型変換が行なわれることがある．$y$ が型検査すると同じメカニズムで *PARTY* は型検査をする．

## 3.3 プロパティ

プロパティとは部品の持つ属性であり，ボタンという部品であれば，その大きさや色などがプロパティである．使用者の要求に合わせ部品の仕様を変更することができるように，ソフトウェア作成時にプロパティの初期値を使用者が決定できるべきである．*PARTY* でもプロパティの初期値を変更をできる機能を取り入れる．

*PARTY* はもとになる言語が $y$ であり，強い型づけ言語であるので，プロパティの型は，使用者がプロパティの初期値として入力した型と一致しなければならない．使用者がプロパティの初期値として入力できる値の型は $y$ のリテラル定数で表すことができる型だけに制限をする．

プロパティが，座標を表す Point 型のように，$y$ のリテラル定数で表すことができない型であれば，はじめに Point クラスのインスタンスを生成しそのコピーによってプロパティを初期化する．Point クラスは 2 つの整数で表すことができるので，リテラル定数でそのプロパティを決定することができる．

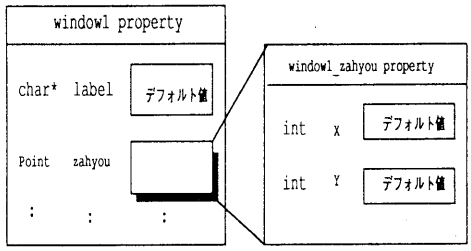


図 2: プロパティダイアログ

## 3.4 部品

*PARTY* では部品を並行部品または逐次部品として使うことができるようにする．並行実行する必要がない部品を逐次部品とすることで実行速度の向上とデータの抽象化をはかる．

部品は配置される場所により振舞いが変わるので，種類によって配置できる場所が異なる．*PARTY* では部品を視覚部品と非視覚部品とウィンドウ部品に分類する．

### 逐次部品と並行部品

*PARTY* では，オブジェクト間通信を行なう部品は並行部品として，行なわない部品は逐次部品とすることができる．ソフトウェアを構成するすべての部品を並行オブジェクトとすると，そのオブジェクトにメッセージを送信するさいには，ノードの特定やオブジェクトの特定などのオーバーヘッドがおこるので，実行速度が遅くなると考えられる．*PARTY* では，実行速度の向上とデータ抽象を実現することを目的に，並行実行する必要がないものは，逐次部品とすることができる．逐次部品と並行部品は，それぞれ $y$ でいう逐次オブジェクト，並行オブジェクトに相当する．

*PARTY* で使用される部品は，はじめは逐次部品か並行部品か決まっていない．使用者は部品を逐次部品として使用するか，並行部品として使用するか，リンクするさい決定できる．部品間をメッセージ通信でリンクをした結果，それらの部品は逐次部品となり，同期通信，非同期通信，または優先通信をした場合は，並行部品となる．

逐次部品は，他の並行オブジェクトとの通信のインターフェイスを持たないので，並行部品との通信はできない．逐次部品は他の部品の中にしか存在しない．よって，逐次部品はネスティッドパート (3.5 章) の中にしか存在し

ない.

既にメッセージ通信でリンクをしたことにより, 逐次部品となった部品から同期通信・非同期通信でリンクをしようとした場合は、そのリンクが違法であることを使用者に警告する。もし使用者がその警告にも関わらずそのリンクを同期通信・非同期通信によるリンクをしたい場合, *PARTY* はそのリンクに関連するすべての部品を並行部品にしても良いかを使用者に聞く. 使用者がそれらの部品を並行部品としてもよいと答えたら, *PARTY* はそのリンクに関連するすべての部品を並行部品とする. 並行部品からメッセージ通信でリンクをしようとした場合も, *PARTY* は同様のことをする. ソフトウェア作成中に違法な通信の種類を見つけながらも, 使用者の要求に柔軟に対話式に対応できるようにした.

#### 視覚部品と非視覚部品とウィンドウ部品

部品は種類によって, 配置できる場所が異なる. ボタンという部品の上にウィンドウという部品を配置することは, 不自然であるからである. またキューという部品のように実行時画面に現れない部品が, ウィンドウ上に配置されることも不自然であると考えた.

ウィンドウの位置が移動された時, ボタンの位置もそれに従い移動された方が自然である. ウィンドウのような部品と, ボタンのような部品と, キューのような部品は種類が違うと考えた.

*PARTY* では, すべての部品を視覚部品, 非視覚部品, ウィンドウ部品に分類する. *PARTY* では, 実行時画面に表示される視覚部品とし, 表示されない部品を非視覚部品とし, 視覚部品を配置できる部品をウィンドウ部品とする.

### 3.5 ネスティッドパート

*PARTY* では, 部品を組み合わせて作ったものを新しい部品とすることができるようにする. 部品を組み合わせるだけで新しい部品を作成できることで, プログラム言語を知らない使用者でも容易に新しい部品が作成できる. 部品を組み合わせて作成した部品をネスティッドパートと呼ぶ.

ネスティッドパートを作成した結果,*PARTY* は一つのクラスを作成する. 作業台上に配置された部品を, そのクラスのデータメンバとする. 作成されたネスティッドパートが受けとることができるメッセージをそのクラスのメソッドとする. ほかの部品と同様一つのクラスで表されているので, ネスティッドパートをさらにネストすることができる.

作成したネスティッドパートが並行部品として使われるか逐次部品として使われるかわからないので,*PARTY* はどちらの部品としても使用されるようなコードを生成する. また, 優先通信によって受けることができるメッセージは部品ごとに決まっている. ネスティッドパート作成時に, 部品が受けとることができるメッセージを優先通信で呼ぶことができるかどうかを決定しなければならない.

### 3.6 コード生成

*PARTY* でソフトウェアを作成した結果,*PARTY* が生成するコードを $y$ のソースコードと決めた. それは,

- 言語レベルの開発が可能
- ソースレベルのディバッグが可能

という理由からである. *PARTY* は,$y$ のソースファイル以外に, 作業台のリソースである `.contents` ファイルを生成する. また, ネスティッドパートを生成した場合は, さらに部品の仕様が記述されたファイルも生成する.

使用者が部品を配置したり, リンクした時点で $y$ のコードを生成すると, 部品が削除されたり, リンクが変更された場合に, その変更に対応する $y$ のコードを見つけることが困難であると考えられる. そこで, 中間的なデータとして作業台上の部品の情報とリンクの情報を保存する. そして, 使用者がコード生成を望んだ時に, 中間的なデータから目的コードを生成する.

中間的なデータとは,

- 部品リスト
- リンクリスト
- 外部仕様データ

である. 新たに部品が配置された場合は, 部品リストにその部品を表すノードを加えることで, 部品リストを更新できる. コード生成はこの部品リストとリンクリストからコードを生成する.

外部仕様データとは, ネスティッドパートを作成したさいに使用される. 外部仕様データには作成しているネスティッドパートのイベントリストとメッセージリストを保存する.

*PARTY* では `.contents` ファイルを目的ファイルの 1 つにする. `.contents` ファイルとは *PARTY* の作業台のリソースである. *PARTY* にしている作業を一旦やめて, 後でまたその作業の続きをしたい場合があると考えらえる. 作業台自体を `.contents` ファイルとして保存し, 必要な時にそのファイルをロードすることで, 作業の続きをすることができる.

## 4 設計内容の図式化

我々は, *PARTY* が複数のオブジェクトから構成されているととらえ, 設計した内容を OMT でいわれている図式を利用しドキュメント化した. OMT でいわれている図式を利用すると,

- 仕様の変更に対処しやすい
- 複数の開発者に対応できる

と判断したからである.

我々は, 識別したオブジェクトの役割の定義をデータ辞書に記述した. そして, オブジェクトモデル, 機能モデル, 動的モデルを作成した.

### 4.1 データ辞書

我々は *PARTY* に必要なオブジェクトを識別したところ, オブジェクトは 24 個になった. そして識別したそれぞれのオブジェクトすべての役割を定義した. そしてデータ辞書を作成した.

- 部品リスト
  部品リストは作業台上に配置されたすべての部品の情報を持つ. 部品リストはコード生成をする時や, 表示する時に使用される. 部品リストが持つ部品のデータは, 部品の名前, 配置された座標, プロパティと種類とその初期値である.

図 3: データ辞書

#### オブジェクトモデル

我々は,*PARTY* を構成するオブジェクトとして識別したものと, そのオブジェクト間の関係を明らかにするためにオブジェクトモデルを作成した. 作成したオブジェクトモデルの一部を以下に示す.

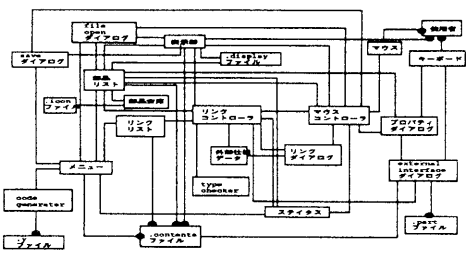


図 4: オブジェクトモデル

### 4.2 機能モデル

機能モデルは, オブジェクトが入力値に対してどんな出力値を出すのかを示している. 作成したデータフローダイアグラムの一部を以下に示す.

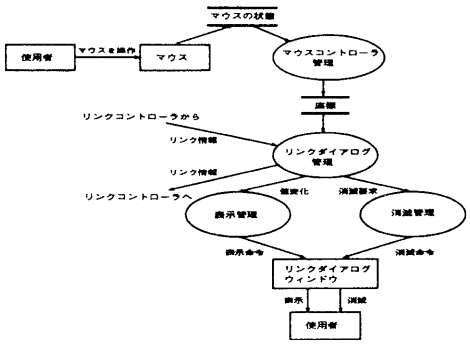


図 5: データフローダイアグラム

### 4.3 動的モデル

#### シナリオの作成

システムの動作を明確にし, オブジェクトの作業を定式化して記述するために *PARTY* のシナリオを作成した. 作成したシナリオの一部を以下に示す.

メニューでコード生成を選択した場合

- 使用者がメニューからコード生成を選択する
- メニューがコードジェネレータにコード生成が選択されたことを知らせる
- コードジェネレータが部品リストとリンクリストをみてコード生成する
- コードジェネレータ生成したコードを .y ファイルに書き込む

図 6: コードジェネレータのシナリオ

#### イベントトレース図の作成

イベントトレース図は, シナリオに対応して事象の発生順序を視覚的に表現している. 作成したイベントトレース図の一部を以下に示す.

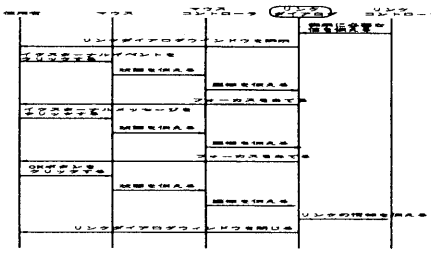

図 7: イベントトレース図

**状態遷移図の作成**

状態遷移図は, それぞれの事象を受信することによる, そのオブジェクトの状態遷移を視覚的に表現している. 作成した状態遷移図の一部を以下に示す.

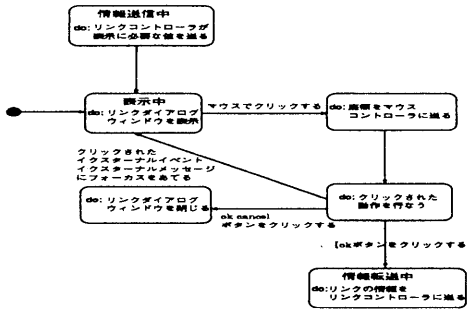


図 8: 状態遷移図

## 5 おわりに

我々は,

- ネットワーク上のオブジェクトを 1 つの部品とする
- オブジェクト間通信をリンクにより表現できる

ことにより, ネットワークソフトウェアの作成が可能なコンポーネントウェア *PARTY* の設計をしてきた. 3 章 2 節で述べたように, 通信の種類を使用者が決定できるようにしたことと, リンクの種類を増やしたことで, オブジェクト間通信をリンクにより表現できるようにした.

対話型ソフトウェア開発環境を提供できるように, *PARTY* では, リンク時やプロパティの設定時に型検査を行なうことや, 違法な通信の種類をリンク時に検出する機能をとりれた. 部品を逐次部品として利用できるとしたことにより, 実行速度の向上とデータの抽象化をはかった.

ディバッグに関する機能の設計は保留した. ディバッグ機能には

- ステップ実行
- ブレークポイントの設定
- 評価式の設定

があると, 一般的にいわれている. しかし, ネットワークソフトウェアは, ネットワーク上の複数のオブジェクトが異なるノードで実行される場合や, 非同期通信によって各オブジェクトが勝手に実行される場合がある. ネットワークソフトウェアの開発環境に, ステップ実行などのディバッグ機能をとりいれることは難しいと考えられる.

ブロードキャスト・マルチキャスト・サイマルキャストといった複数のオブジェクトに同時にメッセージを送るという通信の表現方法が考慮されていない. コンポネントウェアがインスタンスベースなソフトウェアの開発環境であるので, インスタンスを特定できない通信を表現することが難しかったからである.

*PARTY* が実現されれば, OS が提供する機能や通信のプロトコルに関する知識がない人でもネットワークソフトウェアを簡単に作成できるであろう. 今後はこの設計したことに基づいて実現する予定である.

## 謝辞

本論文の作成にあたり, 多くの助言を賜わり御指導して下さった, 大学院生の柴山具久氏, 山田亮介氏, 中原雅仁氏, また,PARTS Workbench を無償で貸して下さった株式会社 SRA に, 深く感謝致します.

## 参考文献

[1] 相沢文雄: *Visual Basic for Windows,* 操作ハンドブック: 株式会社ナツメ社,(1995)

[2] Microsoft: *Visual Basic,Programing System for Windows,* プログラミングガイド: マイクロソフト株式会社,(1994)

[3] 田中 譲: *Intelligent Pad* ユーザーズマニュアル: 北海道大学,(1995)

[4] Digitalk: *PARTS Workbench, User's Guide, PARTS Reference*: Digitalk Inco.(1993)

[5] 野呂 昌満, 崔 梗ろく, 原田 賢一: 並行オブジェクト指向プログラミング言語 $y$ の言語処理系の実現, ソフトウェアシンポジウム'95 論文, 1995.

[6] J.Rumbough,M.Blaha,W.Premerlani,F.Eddy, W.Lorensen:Object-Oriented Modeling and Design, Prentice Hall International,1991: 羽生田栄一監訳, オブジェクト指向方法論 *OMT* モデル化と設計, トッパン,1992.