分散制御システムのデバッグ手法：要求仕様を用いたイベントヒストリの検査†

平井健治†† 杉本 明†† 阿部 茂††

本論文では，システム稼働時に収集したイベントヒストリをシステムの要求仕様を用いて検査することにより，タスクの実行順序などの分散制御ソフトウェアの誤りを検出する手法について述べる．従来のようにイベント検査のために煩雑なテスト用イベントパターンを準備する必要はない．抽象度が大きく異なる要求仕様とイベントとの対応をつけるため，タスクのインタフェースを仲介にしてマッピングテーブルを生成する方法を考案した．また，検査モデルの生成や，デバッグ作業を支援するためのビジュアルモニタの表示にも工夫を行った．鉄鋼プラント制御システムを簡単化した例を用いて，本手法の有効性を示す．

1. はじめに

分散制御システムでは，複数の計算機が互いに通信しながら生産ラインを制御する．しかも，非同期に発生する要求を実時間で処理する必要から，一つの計算機内においても多数のタスクが並行実行される．個々のタスク内の処理は比較的単純であり，あらかじめ十分に単体試験を行うことができる．しかし，タスク間の相互作用は複雑であり，分散システム全体としての動作の検査は容易でない．また，一般に動作に再現性がないため，トラブルが生じた時のエラー原因の解析が困難である．

分散システムの動作を検査する手法には，システム稼働前にプログラムの挙動を静的解析により検査する方法とシステム稼働時に収集したイベントヒストリ（処理の履歴）を検査する方法とがある[1]．静的解析手法では，例えば，S. Shatz らは Ada ソースからペトリネットを生成し，動作解析を行うツールを提案している[2]．この手法では，ペトリネットの可達木を生成することによって実行時に生じる状態を求め，プログラム自体に含まれるエラーを検出できる．しかし，プログラムと仕様との不一致によるエラーは検出できない．

イベントヒストリを用いた手法では，タスク間の通信や制御対象からのセンサ信号，オペレータ要求などのイベントを収集する．収集するイベントヒストリの量は膨大であり，誤り検出をするためには適切な支援方法が必要である．このため，イベントヒストリの解析を対話的に支援するためのブラウジング機能やリプレイ機能を備えたシステム[3]，また，あらかじめプログラマが定義したイベントパターンとの比較によってイベントヒストリの検査を自動化する手法[4),5]が提案されてきた．

イベントヒストリの検査を自動化する手法では，検査の対象となるイベントとその発生順序をあらかじめイベント記述言語で定義するが，次のような問題点がある．

（1） 仕様をもとに分散システムの動作をイベント系列のパターンとして定義することは相当な能力と多大な作業が必要である．

（2） 全体の検査仕様をイベントの時系列パターンとして直接記述した場合，検査仕様自体に誤りが混入する可能性が高い．

一方，分散制御システムのソフトウェア開発へのCASEツールの導入が進んでいる．仕様の誤りを防ぐため，要求仕様や設計仕様の CASE 手法による記述が試みられている．しかし，最終的なプログラムの自動生成までには至っていない．このため，プログラムの作成は多数の人手に頼っており，このプログラミング段階での誤りを統合テストの段階でデバッグする手法が大きな課題となっている．

本論文では，従来のイベント検査手法の問題点（1），（2）を解決するために，CASE ツールにより作成された要求仕様を用いてイベントヒストリを検査する手法を提案する．本手法では，要求仕様から生成した動作モデルとマッピングテーブルを用いてイベントの検査を行うため，煩雑なテスト用イベントパターンを用意することなく，プログラム作成時のエラーが検出できる．システムの要求仕様としては，制御システム向けの構造化分析手法に従って定義したデータ／制

† Debugging of Distributed Control Systems: Checking an Event History with Requirement Specifications by KENJI HIRAI, AKIRA SUGIMOTO and SHIGERU ABE (Central Research Laboratory, Mitsubishi Electric Corporation).

†† 三菱電機(株)中央研究所

御フロー図を用いる．

タスク内で発生する個々のイベントと要求仕様で定義する処理とは抽象度が異なるため，イベント検査を行う場合にはその対応づけが大きな問題であった．これに対して，われわれはタスク間のインタフェースを定義したタスク構成図と，インタフェースの種類ごとに定義したマクロイベント仕様を用いることによって，イベントヒストリとデータ／制御フロー図のフローとを対応づけるためのマッピングテーブルを生成する手法を考案した．また，データ／制御フロー図から自動生成されるペトリネット上でマッピングテーブルを用いながらイベントを検査する手法を明らかにした．

以下，第2章でイベント検査手法の概要，第3章ではイベントヒストリを要求仕様に対応づける方法とペトリネットを用いた動作検査の方法を示す．第4章では本手法によるイベント検査例を示し，その有効性を明らかにする．第5章では本手法によるデバッグ作業を支援するため試作したビジュアルモニタについて述べる．

2. イベント検査手法の概要

2.1 イベントモニタシステム

イベントモニタシステムは，分散制御システムの動作検査を，対象システムから収集したイベントヒストリを検査することによって行う．**図1**にイベントモニタシステムの一般的な構成を示す．

イベントモニタシステムは，対象システム内の各タスクからイベントを収集するイベント収集部，収集したイベントを格納するためのイベントキュー，システムの仕様をもとに生成した動作モデル，イベントヒストリを動作モデルと比較チェックするイベント検査部からなる．

イベントモニタシステムでは，イベントの発生順序

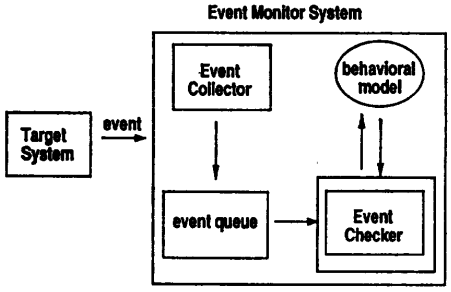


図1 イベントモニタシステムの構成
Fig. 1 Configuration of event monitor system.

などを検査する．

このようなイベントモニタシステムを構築するには，次のことを検討する必要がある．

- 対象イベントとその収集法
- 動作モデルの生成法
- イベント検査法

以下，われわれの採用した方法について述べる．

2.2 対象イベントとその収集法

一般にイベントモニタシステムで収集すべきイベントは，動作検査の方法によって異なる．われわれの手法では，タスクのインタフェースを仲介としてイベントと要求仕様を対応づけて検査するため，イベント収集部では各タスクのインタフェースを実装する際に用いるシステムコールを収集の対象とした．これらの履歴はシステムコールのスタブ関数を用いて収集できる．すなわち，システムコールが実行されるとき，その副作用としてモニタシステムに次の形式のイベントを送信する．

"ev-no. op : arg at task-id"

ここで，ev-no：イベント番号，op：システムコール名，arg：引数，task-id：発生タスクである．

例えば，Task1でTask2へのデータ送信処理(send-to)を実行した場合にイベント収集部に送られるイベントは次のようになる．

"1. send-to : Task2 at Task1"

イベント番号は，各タスク内でイベント発生ごとに与えられ，発生順序を管理するために用いられる．異なるタスク間のイベントについては，半順序関係が満たされるように論理クロック[6]の概念を用いてイベント番号を管理している．すなわち，タスク間でメッセージ通信などを行う場合には，送信側タスクから受信側タスクにイベント番号の値を送信し，受信側のタスクが自タスクのイベント番号と比較して大きい方の値にイベント番号を更新する．

分散環境では，イベントの到着順序が発生順序と一致しない場合があるため，イベント収集部では，到着したイベントをイベント番号に従って並び替える．

2.3 動作モデルの生成法

制御システム向けの要求仕様の定義手法としてWard[7]やHatley ら[8]の手法がある．これらの手法では，データ／制御フロー図，状態遷移図，プロセス仕様，データ関連図によってシステムの要求仕様を定義する．

本手法で用いる動作モデルは，これらの仕様の中で

データ／制御フロー図をもとにして，各変換プロセス(図中楕円で表される）の処理をペトリネット[9]で表現したものである．ペトリネットは，データ／制御フロー図で定義した各フローの実行順序を表す．付録にデータ／制御フロー図からペトリネットを生成するための変換規則を示す．変換規則はWardの変換スキーマを対象としたものであり，フローのパターンごとに定義している．

なお，データ／制御フロー図は階層的に定義されるが，本手法では階層的記述を展開した後に得られるフロー図を用いる．

2.4 イベント検査法と検査範囲

イベント検査部はイベントキューから順にイベントを入力し，その発生順序などを検査する．

検査の基本方針は，イベントをペトリネットの動作モデル上で解釈することである．ペトリネットではシステムの実行状態はマーキングの分布により表されるため，発生順序などの検査は，観測したイベントごとにマーキングを移動させることにより行える．

イベント検査によって検出できるエラーの主なものを以下に示す．

（1） 実行順序や時間制約に関するエラー
タスク間の相互作用の順序やファイルへのアクセス順序の誤り，および時間制約を満たさなかった処理など

（2） 各種の実装ミス
異常終了やタスクの実装ミスなどによって，要求仕様で定義したデータフローあるいは制御フローに相当する処理が実行時に発生しない場合

（3） 未定義処理の発生
非想定タスクの起動など要求仕様で定義されていない処理が発生した場合

第5章で紹介するビジュアルモニタでは，これらの原因となったプログラム上の誤りが，要求仕様のどの部分で生じたかをデータ／制御フロー図上でユーザに知らせてくれる．

3. 要求仕様を用いたイベント検査

本章では，前章で述べたイベント検査部においてイベントと動作モデルとの対応づけを行うための方法と動作検査の手順について述べる．

3.1 要求仕様を用いたイベント検査の概要

図2に提案する手法の主要部の構成を示す．本手法では，要求仕様としてデータ／制御フロー図(a)，設計仕様としてタスク構成図(b)があらかじめCASEツールを用いて定義されていることを前提としている．以後の説明では，データ／制御フロー図の表記法としてWardの変換スキーマを，タスク構成図の表記法としてGomaa[10]の手法を用いるが，本手法は他の表記法を用いた場合にも適用可能である．

動作モデルとイベントとの対応づけを行うためにマクロイベント仕様(c)を定義し，これとタスク構成図からマッピングテーブル(d)を生成する．イベントキューから入力したイベントヒストリ(e)はマクロイベントに変換された後，マッピングテーブルを用いて動作モデルと比較チェックされる．

以下，各部について順に説明する．

（a） データ／制御フロー図（図3参照）
データ／制御フロー図に付録の変換規則を適用することによってペトリネット表現の動作モデルが生成される．

（b） タスク構成図（図4参照）
タスク構成図は，タスク，ファイル，外部機器の三つの要素を用いてシステムのソフトウェア構成と構成要素間のインタフェース操作を定義したものである．以後の説明では，このインタフェース操作をタスクIFと呼ぶ(図中のI1からI11)．タスクIFは，複数のシステムコールの組合せによって実現される．

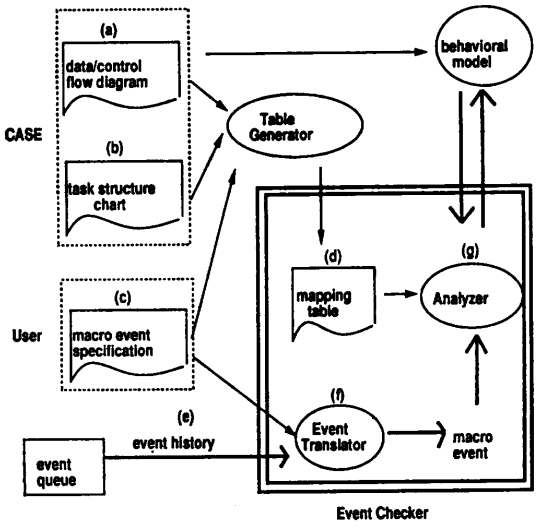


図2 イベント検査手法
Fig. 2 The event check method.

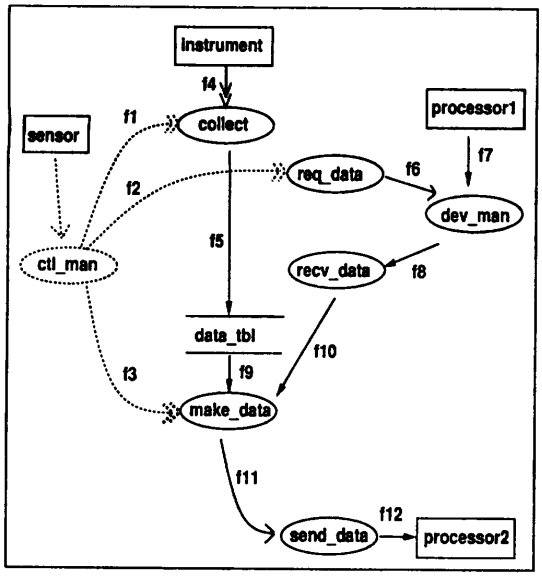


図 3 データ／制御フロー図の例
Fig. 3 An example of data/control flow diagram.

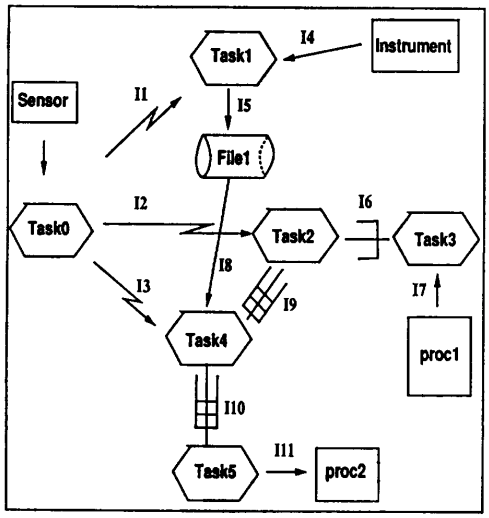


図 4 タスク構成図の例
Fig. 4 An example of task structure chart.

発生順序		イベント名
e1.	wait	: Task0 at Task2
e2.	wait	: Task0 at Task4
e3.	signal	: Task2 at Task0
e4.	send-to	: Task3 at Task2
e5.	write	: File1 at Task1
e6.	signal	: Task4 at Task0
e7.	send-to	: Task4 at Task2
e8.	read	: File1 at Task4
e9.	recv-from	: Task2 at Task4

図 5 イベントヒストリの例
Fig. 5 An example of an event history.

表 1 マクロイベント仕様
Table 1 Macro event specification.

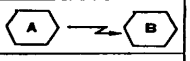
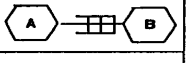
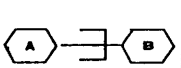
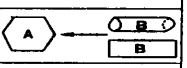
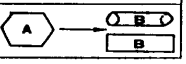
タスクIF（マクロイベント）の種類	イベント仕様	タスク構成図の表記法
シグナル操作 (I1,I2,I3)	(wait:A at B) ; (signal:B at A)	
疎結合型通信 (I9,I10)	(send-to:B at A) & (recv-from:A at B)	
密結合型通信 (I6)	(send-to:B at A ; recv-from:B at A) & (recv-from:A at B ; send-to:A at B)	
デバイス入力 (I4,I7,I8)	read:B at A	
デバイス出力 (I5,I11)	write:B at A	

表 2 マッピングテーブル
Table 2 Mapping table.

フロー	タスク IF (マクロイベント)	マクロイベントのパターン
f 1	I 1	(wait : Task0 at Task1) ; (signal : Task1 at Task0)
f 2	I 2	(wait : Task0 at Task2) ; (signal : Task2 at Task0)
f 3	I 3	(wait : Task0 at Task4) ; (signal : Task4 at Task0)
f 4	I 4	read : Inst at Task1
f 5	I 5	write : File1 at Task1
f 6 f 8	I 6	(send-to : Task3 at Task2 ; recv-from : Task3 at Task2) & (recv-from : Task2 at Task3 ; send-to : Task2 at Task3)
f 7	I 7	read : Proc1 at Task3
f 9	I 8	read : File1 at Task4
f 10	I 9	(send-to : Task4 at Task2) & (recv-from : Task2 at Task4)
f 11	I 10	(send-to : Task5 at Task4) & (recv-from : Task4 at Task5)
f 12	I 11	write : Proc2 at Task5

（c） マクロイベント仕様（表 1 参照）

マクロイベント仕様はタスク構成図で定義したタスク IF の種類ごとに，その実装に用いるシステムコールを出現順序も含めてイベントパターンの形式で定義したものである．マクロイベント仕様から求まる個々のタスク IF のイベントパターンをマクロイベントと呼ぶ．マクロイベント仕様の定義手法については 3.2 節で述べる．

(d) マッピングテーブル（**表2**参照）
データ／制御フロー図のフローとマクロイベントとの対応表となる．マッピングテーブルを生成する手順は3.3節で述べる．

(e) イベントヒストリ（**図5**参照）
イベントヒストリは，マクロイベント仕様の定義に用いたシステムコールの履歴を収集し，論理クロック順に並び替えたものである．イベントヒストリには複数のタスクのイベント系列が混在している．

(f) イベント変換部
イベントヒストリ(e)をマクロイベント仕様(c)とパターンマッチングすることによって，マクロイベントの系列に変換する．

(g) 動作検査部
マッピングテーブル(d)を用いて，マクロイベントをデータ／制御フロー図に対応づけ，ペトリネットの動作モデルによって発生順序などを検査する．

3.2 マクロイベント仕様の定義

マクロイベント仕様は，2.2節で述べたイベントと同様の形式で定義する．システムコールの出現順序は，逐次型の実行を“;”，並行型の実行を“&”オペレータを用いて**表3**に示すように定義する．表中の X, Y はタスク IF の両端のノードを表す．

(1) 並行型
ノード X, Y がタスクの場合で，これらのタスクでシステムコール op1, op2 が並行実行されることを示す．

(2) 逐次型
タスク内または，タスク間でシステムコール op1, op2 が逐次実行されることを示す．

これらのパターンを組み合わせてタスク IF の種類ごとにマクロイベント仕様を定義する．

例えば，メッセージ通信を表すタスク IF に対するマクロイベント仕様は次のようになる．なお，X, Y はそれぞれ，送信側，受信側のタスクを表す．

(send-to : Y at X) & (recv-from : X at Y)

X は Y にデータ送信，Y は X からデータ受信を行い，これらが並行実行されることを表す．

イベントヒストリとマクロイベント仕様とのパターンマッチングを行うことにより，タスク構成図中の各タスク IF の発生（マクロイベント）を観測することができる．

表3 マクロイベント仕様で用いる基本パターン
Table 3 Basic patterns in macro event specification.

	イベントパターン
並行型	(op1 : Y at X) & (op2 : X at Y)
逐次型	(op1 : Y at X); (op2 : X at Y)
	(op1 : Y at X); (op2 : Y at X)

3.3 マッピングテーブルの生成法

要求仕様としてデータ／制御フロー図が与えられると，システムエンジニアはこの図の各ノードに対してタスクやファイルを割り付けることにより，設計仕様としてタスク構成図などを作成する．タスク割付け表は，このようなデータ／制御フロー図のノードとタスク構成図のノードとの対応関係を示す．

各フローは，その両端ノードの割付け結果に従って，以下のいずれかの処理に対応づけられる．

(1) 両端ノードが異なるタスクに割り付けられた場合
タスク間の操作を表すタスク IF（メッセージ通信やタスク起動など）

(2) 一端がタスク，他端がファイルや外部機器に割り付けられた場合
タスクとファイルや外部機器との間の操作を表すタスク IF（ファイルアクセスなど）

(3) 両端ノードが同一タスクに割り付けられた場合
タスクの内部処理

タスク構成図中のあるノード間に一つのタスク IF しか定義されていない場合には，タスク割付け表からフローとタスク IF との対応関係は自動的に求められる．しかし，ノード間に複数のタスク IF が定義されている場合には，利用者が対応関係を指定することが必要である．

(1), (2)の場合，フローの発生は対応づけられたタスク IF の実行をマクロイベントとして観測することにより推定することができる．(3)の場合，フローの発生はイベントとして観測できないため他のフローからその発生を推定する．

マッピングテーブルの生成は次の手順で行う．

(1) マクロイベントを求める．
マクロイベント仕様を用いてタスク構成図のすべてのタスク IF に対するマクロイベントを求める．

（2） タスク割付け表をもとにタスク IF に対応するフローを求める．

一つのタスク IF には一つのフローが対応する場合と，次のように複数のフローが対応する場合がある（**図6参照**）．

（a） フローの合流，分岐がある

（b） フローの選択がある

（c） フローの実行に順序関係がある

マクロイベントとタスク IF は1対1に対応するため，マクロイベントとデータ／制御フロー図のフローとの対応がとれる．

3.4 動作検査法

まず，イベントヒストリをマクロイベントの系列に変換する．各イベントは論理クロック順のイベント番号に従って並び替えられ，イベントキューに格納されている．したがって，イベントキューの先頭のイベントから順にパターンマッチングしていくことによってマクロイベントへの変換が行える．

次に，各マクロイベントに対応するフローをマッピングテーブルから求め，その発生順序をペトリネット上で以下の手順で検査する．

（1） マクロイベントに一つのフローが対応する場合

フローに対応するトランジションの発火条件を検査する．トランジションは，すべての入力プレースにトークンが存在するときに発火可能とする．

(1-a) トランジションが発火可能なときそのトランジションを発火させ，マーキングを移動する．

(1-b) トランジションが発火可能でないとき

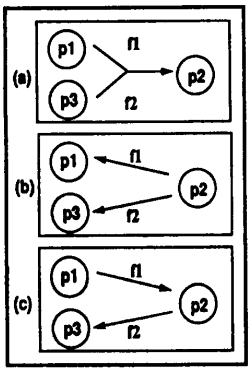


図 6 タスク IF に対応するフローのパターン
Fig. 6 Flow patterns corresponding to task interface.

実行順序のエラーとする．

（2） マクロイベントに複数のフローが対応する場合

前節で述べた3種類の場合（図6参照）があり，各ケースとも発火の候補となるトランジションが複数ある．各々の検査は以下のように行う．

（a） 候補となるトランジションがすべて発火可能の時にマーキングを移動させ，他の場合は動作エラーとする．

（b） 候補となるトランジションの中で発火可能なものを一つ選択し，マーキングを移動させる．すべてのトランジションが発火可能でないときは動作エラーとする．

（c） 候補となるトランジションについて順に発火条件を検査する．発火可能な場合はマーキングを移動させ，次のトランジションを検査する．発火可能でないものが一つでも含まれる場合には動作エラーとする．

イベントヒストリをマクロイベントに変換してから動作検査を行うことにより，2.4 節で述べた各エラーは以下のように検出できる．

（1） 実行順序に関するエラー

ペトリネットのトランジションの発火に失敗した場合に検出される．

（2） 実装ミス

マクロイベントへの変換時にマクロイベント仕様とのパターンマッチングが途中で失敗した場合は，対応するタスク IF の実装ミスとする．

（3） 未定義処理

マクロイベント仕様とのパターンマッチングに成功したイベント系列の中で，タスク構成図上で対応するタスク IF をもたないものを未定義処理とする．

4. 例　　題

本手法によるイベント検査の具体例を紹介する．対象とするシステムは鉄鋼プラント制御における実績データの収集および送信部分を簡単化したものである．

4.1 データ／制御フロー図とタスク構成図の入力

まず，CASE ツールを用いて作成されたデータ／制

御フロー図（図3）とタスク構成図（図4）を用意する．タスク構成図では図3の各ノードに対して表4のようにタスク割付けを行っている．

図3で f1 から f3 は制御フローであり，f4 から f12 はデータフローである．変換プロセス recv_data は f8，f10 の順に処理を行うことが定義されている．

以下の節では，ソフトウェアの作成ミスのため f8 の前に f10 に相当する処理が実行されたケースについてイベント検査を行った結果を示す．

4.2 マクロイベント仕様の入力

次に利用者はタスク構成図をもとに3.2節の表記法に従ってマクロイベント仕様を定義する（表1）．

図4のタスク構成図では5種類のタスク IF が用いられている．各タスク IF は，メッセージ送信 (send-to)，メッセージ受信 (recv-from)，シグナル送信 (signal)，事象発生待ち (wait)，デバイス入力 (read)，デバイス出力 (write) の6種類のシステムコールで実装されている．

4.3 マッピングテーブルと動作モデルの生成

システムでは，図4のタスク構成図と表1のマクロイベント仕様からすべてのマクロイベントを求める．この場合にはタスク構成図中のどのノード間にも一つのタスク IF しか定義されていない．したがって，3.3節のテーブル生成手順を用いると表4のタスク割付け表から表2のマッピングテーブルが自動生成される．

また，2.3 節の手法を用いることにより，図3の仕様からイベントヒストリを検査するためのペトリネットが自動生成される．図7に得られたペトリネットの一部を示す．

4.4 動作検査の実行

システムは，表1のマクロイベント仕様の定義に用いた6種類のシステムコールの実行履歴をイベントヒストリとして収集する．図5に収集した結果の一部を示す．e1 から e9 はイベントの発生順序を表す．

システムは，このイベントヒストリをマクロイベントに変換した後，表2のマッピングテーブルを用いて，データ／制御フロー図に対応づける．図8に対応づけた結果を示す．図の中段はマクロイベントへの変換結果を表し，下段は図3のフロー番号を表す．図8では，イベントヒストリが五つのフロー f2，f5，f3，f9，f10 に対応づけられている．例えばイベント e1，e3 は，マクロイベント I2 に変換された後，フロー f2 に対応づけられている．

図7のペトリネットによりこれらのフロー（f3，f9，f10）の実行順序を検査する．図でトランジション t3 は発火可能であり，フロー f3 の発生により，マーキングをプレース p3 へ移動できる．さらに，t4 も発火できるため，マーキングを p4 に移動させる．この結果，t9 の発火条件が成立するため，f9 の発生時に p9 へマーキングを移動できる．一方，t10 は発火可能でないため f10 の発生時にエラーが検出される．以上のことから，フロー f3 と f9 は，仕様どおりの順序で実行されているが，フロー f10 はプログラム上のミスにより，f8 より前に実行されたことがわかる．

4.5 従来手法との比較評価

提案手法と従来のイベント記述言語を用いた手法[5]

表 4 タスク割付け表
Table 4 Task assignment table.

要求仕様のノード	タスク構成図のノード
ctl_man	Task0
collect	Task1
req_data recv_data	Task2
dev_man	Task3
make_data	Task4
send_data	Task5
data_tbl	File1

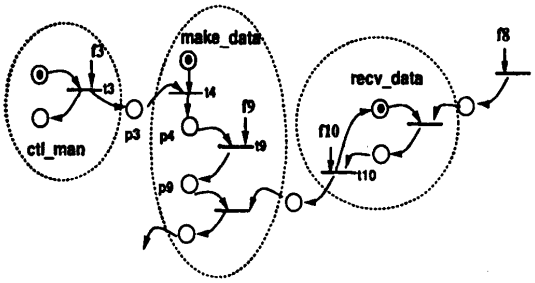


図 7 動作モデルの一部
Fig. 7 A part of behavioral model.

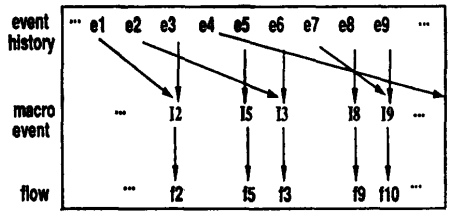


図 8 イベントヒストリからマクロイベントへの変換
Fig. 8 Conversion from an event history to macro events.

表 5 イベント検査手法の比較
Table 5 Comparison of event check methods.

	提案手法	従来手法
仕様の提示	CASEで作成されたデータ／制御フロー図とタスク構成図	左と同じあるいは同等の仕様
動作モデルの生成法	データ／制御フロー図を自動変換して得られるペトリネット	上の仕様をもとに利用者がイベント記述言語を用いてイベントの時系列パターンをモデルとして記述
イベントとモデルの対応づけ	タスク割付け表とマクロイベント仕様の定義が必要	上の段階で利用者が考慮する
検査範囲	システム全体の動作を検査するのに適する	一部のタスク間の動作検査には良いが，全体の検査はモデル記述が大変
信頼性	マクロイベント仕様の入力以外の大部分が自動化されるので信頼性が高い	仕様から時系列パターンの作成を行う人への依存大
誤り原因の究明	ビジュアルモニタで遡及が容易	利用者が因果関係を考える
省力化	CASEを併用すれば大部分のプロセスが自動化可能	イベントの時系列パターンの記述にかなりの人手を要する

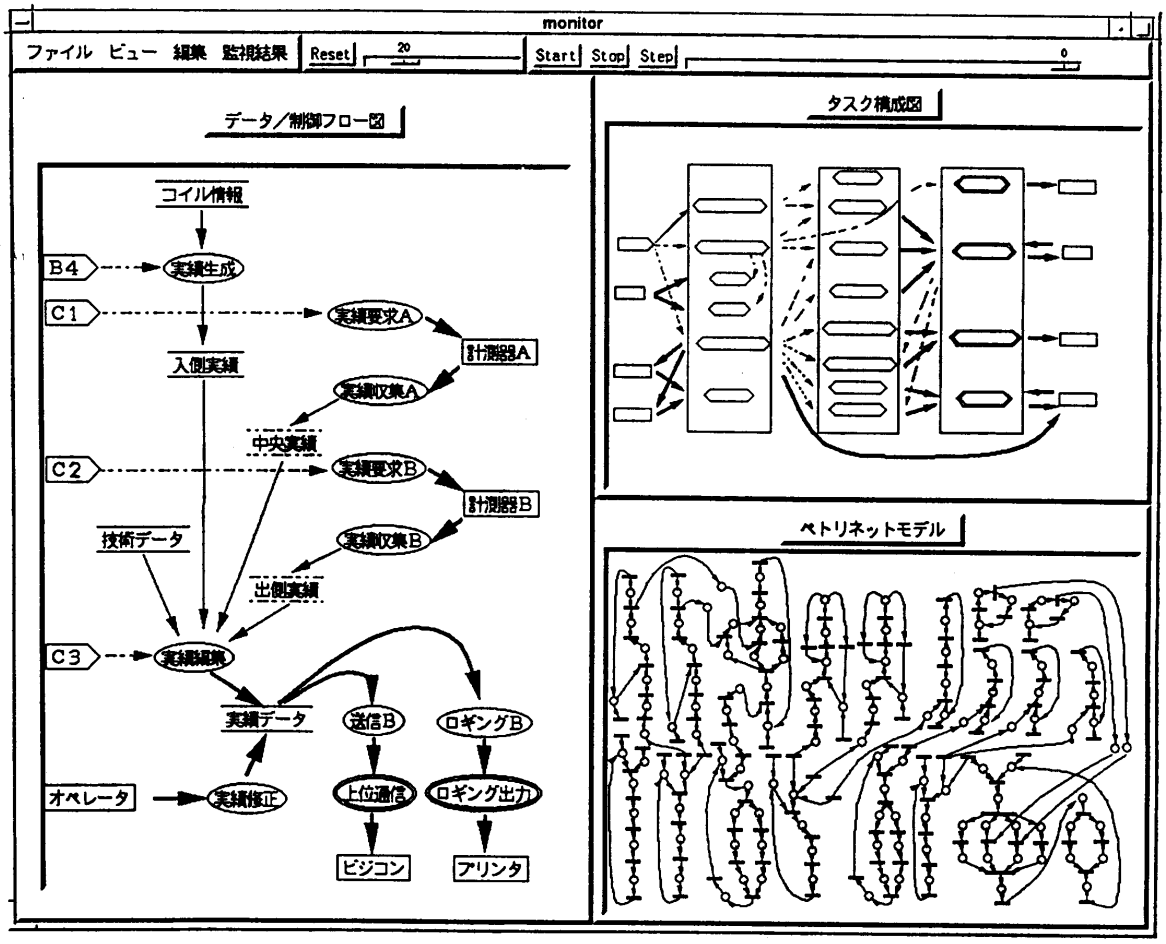


図 9 ビジュアルモニタの画面例
Fig. 9 A snapshot of visual monitor.

との主な相違点をまとめて**表5**に示す．ここでは，今後 CASE の利用が高まり，仕様の形式的記述が容易に得られる状況を想定している．このような状況下では提案手法はかなりの部分を自動化できるため，従来法に比べて大幅な省力化が期待できる．

一方，従来手法では，モデル生成のためにはイベント記述言語自体の修得が必要である．また，個別にイベントパターンを定義していくため検査洩れが生じる可能性が高い．

5. ビジュアルモニタ

本手法によるデバッグ作業を支援するために試作したビジュアルモニタについて述べる．**図9**にビジュアルモニタの画面例を示す．ビジュアルモニタでは，以下の機能を提供し，エラー原因の解析を支援している．

（1） データ／制御フロー図上での実行過程の表示
イベントヒストリをマクロイベントに変換し，その結果をマッピングテーブルを用いてデータ／制御フロー図上で示す．各変換プロセスの実行状態（実行中 or 停止中）を別の色で示し，フローの実行による実行状態の変化も表示する．

（2） タスク構成図上での実行過程の表示
マクロイベントの発生ごとにタスク構成図上の対応するタスク IF を表示する．変換プロセスのときと同様にタスクの実行状態も色変化で表す．タスクに割り付けられた変換プロセスが一つでも実行中であれば，そのタスクの状態も実行中とする．

（3） 対応関係の表示
各タスクに割り付けられた変換プロセスの集合を表示する．また，各タスク IF に対応するフローの集合をマッピングテーブルをもとに表示する．

（4） エラー箇所の表示
イベント検査により検出したエラー箇所をデータ／制御フロー図とタスク構成図の各レベルで表示する．リプレイ機能を用いてエラーに至るまでの過程を遡って確認することができ，デバッグを効率的に行える．

6. おわりに

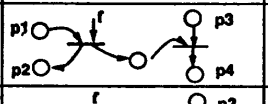
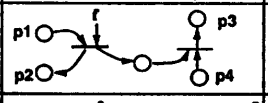
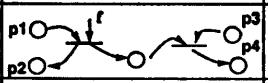
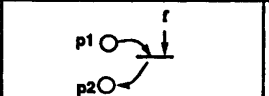
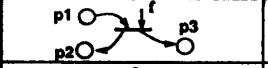
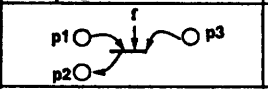
本論文では，分散制御ソフトウェアのデバッグ手法として一般的なイベント検査において煩雑なテスト用イベントパターンの準備を必要としない，新しい手法を提案した．本手法の特長は，CASE で用いられる要求仕様を用いてイベント検査を行う点にあり，これに必要な要求仕様とイベントとの対応をつけるマッピングテーブルの生成法，要求仕様からの動作モデルの生成法を明らかにした．本手法を用いれば，分散制御システムのソフト開発時に問題となる並行タスクの実行順序エラーなどを簡単に検査できる．また，要求仕様上でエラー部分を容易に特定できるビジュアルモニタも紹介した．今後，時間制約の検査機能などを拡充し，普及の広がりを見せる CASE ツールの新しい機能として発展させたいと考えている．

参考文献

1) Mcdowell, C. E. and Helmbold, D. P.: Debugging Concurrent Programs, *ACM Comput. Surv.*, Vol. 21, No. 4, pp. 593-622 (1989).
2) Shatz, S. et al.: Design and Implementation of a Petri Net Based Toolkit for Ada Tasking Analysis, *IEEE Trans. on Parallel and Distributed Systems*, Vol. 1, No. 4, pp. 424-441 (1990).
3) Leblanc, R. J. and Robbins, A. D.: Event-Driven Monitoring of Distributed Programs, *Proc. of the 5th International Conference on Distributed Computing Systems*, IEEE, pp. 515-522 (1985).
4) Bates, P.: Debugging Heterogeneous Distributed Systems Using Event-Based Models of Behavior, *Proc. of Workshop on Parallel and Distributed Debugging*, ACM, pp. 11-22 (1988).
5) Helmbold, D. P. and Luckham, D. C.: TSL: Task Sequence Language, *Ada in Use, Proc. of the Ada International Conference*, ACM, pp. 255-274, Cambridge University Press (1985).
6) Lamport, L.: Time, Clocks, and the Ordering of Events in a Distributed System, *Comm. ACM*, Vol. 21, No. 7, pp. 558-564 (1978).
7) Ward, P. T.: The Transformation Schema: An Extension of the Data Flow Diagram to Represent Control and Timing, *IEEE Trans. Softw. Eng.*, Vol. 12, No. 2, pp. 198-210 (1986).
8) Hatley, D. J. and Pirbhai, I. A., 立田（訳）：リアルタイムシステムの構造化分析，p. 377，日経 BP 社（1989）．
9) Peterson, J. L., 市川，小林（訳）：ペトリネット入門，p. 293，共立出版（1984）．
10) Gomaa, H.: A Software Design Method for Real-Time Systems, *Comm. ACM*, Vol. 27, No. 9, pp. 938-949 (1984).

付録　ペトリネットへの変換規則

変換プロセスの入出力フローは，その種類に応じた変換規則によってペトリネットに変換される．同一の変換プロセスが複数の入出力フローをもつときは，各フローに対するペトリネットのプレースの接続順序を変換プロセスの仕様に従って決める．

data/control flow pattern	petri net pattern	marking before	marking after
A —f⇢ B		p1 p3	p2 p4
A —f⇠ B		p1 p4	p2 p3
A —f→ B		p1 p3	p2 p4
A —f→ ||		p1	p2
A —f→ □			
A —f→ |		p1	p2 p3
A —f→ |		p1 p3	p2

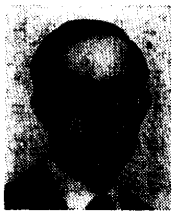
(平成3年7月 5 日受付)
(平成4年1月17日採録)

平井　健治（正会員）

昭和 61 年京都大学工学部情報工学科卒業．昭和63年同大学院工学研究科情報工学専攻修士課程修了．同年4月三菱電機(株)入社．中央研究所にて分散システムの研究に従事．

杉本　　明（正会員）

昭和 52 年京都大学理学部卒業．昭和 54 年同大学院工学研究科（数理工学）修士課程修了．同年三菱電機(株)入社．以来同社中央研究所にて，設計支援システム，オブジェクト指向言語，分散システムなどの研究に従事．電子情報通信学会，ACM 各会員．



阿部　　茂（正会員）

昭和 46 年東京大学工学部電子工学科卒業．昭和 51 年同大学院工学系研究科(電気工学)博士課程修了．工学博士．同年4月三菱電機(株)入社．中央研究所にて，電力系統システム，オブジェクト指向システム，幾何情報システムの研究に従事．昭和60年電気学会論文賞受賞．IEEE，電気学会，電子情報通信学会各会員．