4R−12

分散制御システムのデバッグ手法
—動作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
三菱電機(株)中央研究所

1 はじめに

分散制御システムのデバッグ支援を目的として、システムの動作仕様をもとに実行時のイベントヒストリを検査する手法を考案した。また、この手法をもとにしたモニタシステムを試作した。本手法では、イベントヒストリを検査し、実行順序の誤りや不当なデータアクセスによるエラーを検出する。

2 背景

分散制御システムのデバッグ手法として、イベントヒストリ(対象システム中の各タスクから収集した実行履歴)を検査する手法が知られている[1]。この手法では、実行時のイベントヒストリを、予めプログラマがイベント記述言語を用いて定義した検査用のイベント系列(エラーパターンなど)と比較することにより、イベント発生順序の誤りなどによるエラーを検出することができる。

しかし、デバッグ時に検出すべきエラーごとに検査用の仕様を新たに定義することが必要であり、また、システムの動作仕様をプリミティブなイベント系列で記述しなければならないため、プログラマに対する負担が大きい。

筆者らは、これらの問題点を解決するため、システム分析及びソフトウエア設計時に定義した動作仕様を用いてイベントヒストリを検査し、エラー検出する手法を考案した。また、この手法に基づいてモニタシステムを試作し、その有効性を確認した。

本手法は、要求仕様で定義した情報を用いて実行時の検査を行なうため、以下の利点がある。

- テスト用の動作仕様の定義を行なう必要がない
- 要求仕様レベルで動作の検査をすることができる

3 動作仕様を用いたデバッグ手法

図1に本手法をもとに試作したデバッグ支援用モニタシステムの構成とデバッグ対象とするシステムの開発手順を示す。

3.1 対象システム

デバッグの対象とするシステムは、プラント制御に用いられる分散制御システムである。この種のシステムは、トラッキング機能、制御機器に対する設定値の計算や各種実績データの収集など複合的な機能から構成される。各機能はセンサーなど外部からの制御信号によって起動され、並行実行される。

対象システムは、以下の手順で開発する。

- システム分析

  システム分析では、分散システムの各機能を分離し、各々を制御アジェンダとして要求仕様を定義する。図2に、鉄鋼プラント制御システムにおける設定計算(制御機器に対する設定値の計算)の機能を簡単化し、制御アジェンダとして記述した例を示す。

  各制御アジェンダの要求仕様は、Wardの変換スキーマ[2]に従って定義する。ただし、実行時のイベント発生順序を検査するため、データストアへのアクセス順序を明示的に示すための表記法を追加した。

  動作検査はアジェンダごとに定義した要求仕様に従って行なう。

- ソフトウエア設計

  要求仕様で定義した情報を整理し、システムを実現するためのソフトウエア仕様を定義する。タスク構成は、実行効率などを考慮しながら、要求仕様中の各ノードをタスクへ割り付けていくことにより決定する。タスク仕様及びタスク間のインタフェース仕様などは、各タスクにマッピングされたノード間の関係から決定される。

- 実装、テスト

  ソフトウエア仕様に従って各タスクを制御システム記述用の言語で実現する。モニタシステムへのイベント送信処理は、システムコールのレベルで行なわれるものとする。

  動作検査の対象となるイベントはファイル入力(read)、ファイル出力(write)、メッセージ送信(send)、メッセージ受信(receive)の4種類とする。各タスクでこれらの操作に対応するシステムコールが実行される際には、以下の情報がモニタシステムに送信される。

  “イベント名:引数:発生タスク”

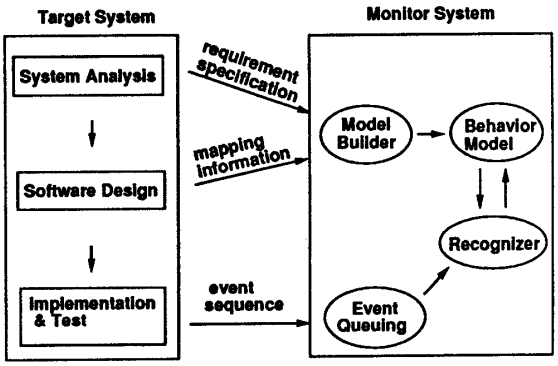

図1: モニタシステムの構成

3.2 モニタシステム

モニタシステムは、実行時に各タスクから収集したイベントを検査し、実行順序の誤りによるエラーや不当なデータアクセスによるエラーを検出する。このために要求仕様とソフトウエア仕様から

Debugging of Distributed Control Syetem:
Checking the Event History using Behabior Specification
Kenji HIRAI, Akira SUGIMOTO, Shigeru ABE
Mitsubishi Electric Corp.

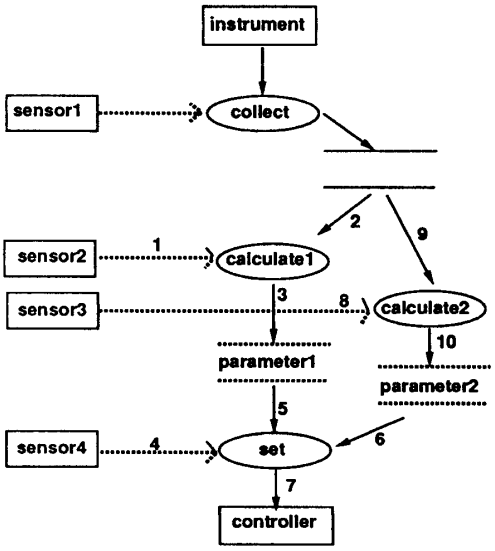

図 2: 制御アジェンダの記述例

実行時のイベント発生順序を検査するための検証用モデルを生成する。検証用モデルとしては、並行システムの動作表現に適したペトリネットを採用した。

4 イベントヒストリの検査方法

4.1 ペトリネットの生成

要求仕様の情報はデータや制御の流れを表すだけであり、実行時のイベントとは直接対応していない。従って、動作検証用のペトリネットを生成するためには、要求仕様から実行時にイベントとして観測できる情報を抽出し、それらの発生順序を決定することが必要である。

イベントとして観測できる情報は、要求仕様とソフトウエア仕様とのマッピング情報から決定する。検証用ペトリネットは、各トランジションごとに発火に必要なイベントを決め、イベント発生順に状態遷移を行なうように構成する。

検証用ペトリネットの生成手順

- サブネットの生成
  1つのデータ変換ノードに関係する全てのフローに対して順にペトリネットへの変換規則（フローの種類とその両端ノードの種類に応じて定義）を適用し、変換結果を制御フロー、入力フロー、出力フローの順で状態遷移を行なうように結合する。
- サブネットの合成
  複数のデータ変換ノードが同じデータバッファにアクセスする場合や、データ変換ノード間でフローがある場合には参照関係をもとにサブネットを合成する。
- トランジションへのイベント割り当て
  トランジションの発火に必要なイベントは、フローがマッピングされたソフトウエア仕様の情報から決定する。

図3に図2の動作仕様から生成した検証用ペトリネットの一部を示す。図は3つのデータ変換ノードのサブネットを合成したものである。要求仕様中の各フローは、同じ番号のトランジションに変換されている。

4.2 ペトリネットを用いた動作検査

検証用ペトリネットは、実行時に収集したイベントによって各トランジションを順に発火させることによって実行する。

イベント発生順序やデータアクセスの正当性を検査するためにトランジションの発火条件と発火規則を以下のように定義する。

発火条件 全ての入力プレースにトークンが存在し、出力プレースにトークンが存在しない

発火規則 イベント発生時に対応するトランジションが上記の発火条件を満たすときは、そのトランジションを発火させる。

イベント発生順序に誤りがある場合には、対応するトランジションが発火できないため動作エラーが検出される。

4.3 エラー検出例

図 3に図 2の動作仕様に従ってイベントヒストリを検査した結果を示す。データ変換ノード calculate1,calculate2,set は、それぞれ Task1,Task2,Task3 にマッピングされているものとする。

イベントの発生順序に従って対応するトランジションが発火している。図のイベントヒストリでは6番目のイベントの検査時に、トランジションＴ６の発火に失敗し、エラーが検出されている。これは、要求仕様では学習計算処理（calculate2）がデータバッファ（parameter2）に値を出力する前に設定処理（set）がデータ入力をしてエラーが発生したことに対応している。

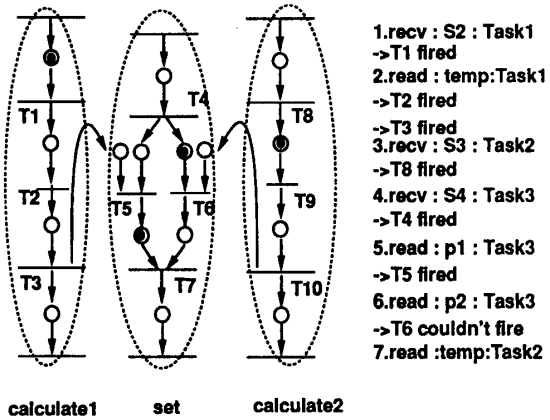

図 3: 検証用ペトリネットとその検査例

5 おわりに

本稿では動作仕様を用いた分散制御システムのデバッグ手法について述べた。実行時のエラーを動作仕様のレベルで確認するためのビジュアルモニタも試作完了している。今後の課題としては、分散環境でのイベント到着順序についての問題が残されている。

参考文献

[1] Charles E. Mcdowell and David P. Helmbold, "Debugging Concurrent Programs", ACM Computing Surveys,Vol.21,No.4,Dec 1989, pp593-pp622

[2] Paul T. Ward, The Transformation Schema: An Extension of the Data Flow Diagram to Represent Control and Timing, IEEE Tr.SE 12-2,1986,pp198-210