組込みシステムのためのカラーペトリネットを利用した実装に基づくモデル化とテスト法

渡 辺 晴 美† 川 井 奈 央†† 山 本 修 二††

本論文では，組込みソフトウェアのために，割込みハンドラに対応したモデリングとテスト方法を提案する．提案する方法では，タスク状態図から割込みハンドラとコントローラ状態図を生成し，これらの状態図をカラーペトリネットへ変換しテストケースを得る．割込みハンドラに対応することにより，従来よりも設計と実装とが明確に関連付くため，設計から実装へのシームレスな開発，実際的なシミュレーションとテストケースの生成が可能になる．以上により，プロダクトの品質の向上と短期開発に貢献する．

A Modeling and Testing Method Using Colored Petri Nets for Embedded Software Development Based on Implementation

Harumi Watanabe,† Nao Kawai†† and Shuji Yamamoto††

In this paper, we propose a modeling and testing technique supporting interruption handlers for Embedded Software. By the method, interruption handler state charts and an interruption controller state chart are built from task state charts. The test cases are generated from colored Petri Nets which is translated from their state charts. Since the relation between design and implementation becomes clear than before using the method, seamless development from design to implementation, practical simulation and test cases generation become possible. Therefore, the method contributes to improvement quality of products and reducing development periods.

1. はじめに

組込みソフトウェアにとって，高い品質は必要不可欠であるが，近年の大規模化，複雑化，さらには短期リリースが求められ，品質を保持することが難しくなりつつある．したがって，従来よりも短い開発期間で従来以上の品質を保持でき，複雑なシステムに対しても対応できるテスト方法が求められている．

仕様どおり実装されていることを確認するために適合テストが一般に行われている．適合テストでは仕様を状態図で記述し，記述した状態図の実行経路からテストケースを生成する．テストケースをプログラムの入出力と対応付けることによりテストを行う．組込みソフトウェアに対して適合テストを行う際，2つの問題点がある．1つは，組込みソフトウェアが持つ並行性のため，テストケース生成時に状態数の爆発問題を起こすことである．もう1つは，状態図と実装との対応付けの困難さにある．

これまで，カラーペトリネット[21)~23)]を利用したテストケース生成法を提案することで，前者の問題に対応してきた[3),5),9)]．後者の1つに割込みハンドラの問題がある．従来，システム全体またはタスクごとに状態図を記述することは一般的であるが，実装で利用する割込みハンドラを状態図で記述しない場合が多い．このことが，分析から実装へのシームレスな開発，実際的なシミュレーションと見積もり，実践的なテストケースの生成を困難にする理由の1つとなっている．

割込みハンドラは，名称どおり割込みを処理するハンドラであるため，できるだけ短時間に処理されなければならない．処理時間がかかる場合，デッドラインミスや多重割込み問題などを引き起こす可能性がある．また，割込みは現在実行しているタスクと関連するため，割込みハンドラの処理は複数のタスクと関連を持つ．コンテキストスイッチ処理は時間を要するため，多くのリアルタイムOSでは，割込みハンドラは現在動作しているタスク資源を利用する．この場合，割込

† 東海大学開発工学部
Tokai University
†† キャッツ（株）
CATS Co., Ltd.

みハンドラを処理できる分のメモリやスタックを関連するタスクが残していなければならない．したがって，割込みハンドラを，従来から一般に広く行われているタスクの一種や，外部環境としてモデリングすることは適切でない．これらの方法では，割込みハンドラがどのタスクのどの状態と関連しているかを開発者が網羅的に把握しなければならない．

本論文では，これらの問題を解決するために，割込みハンドラに対応したモデリング方法とテスト方法を提案する．提案する方法では，従来記述していたタスク状態図から，割込みハンドラ状態図，割込みコントローラ状態図を構築し，シミュレーション，テストケース生成を行う．テストケース生成は，カラーペトリネットを用いた方法[3),5),9)]により行う．割込みハンドラを状態図に明示することにより，割込みハンドラを実行するのに必要な時間およびメモリを推測することが可能になり，従来より正確な時間およびメモリの見積もりが可能となる．さらに，割込みハンドラを自動生成することで，どのタスクのどの状態と関連するかが明らかになるため，開発者が網羅的に把握してきた問題を解決することができる．メモリやスタックを割込みハンドラに割り付けるOSでも，割込みがどのタスクのどの状態と関連するかを調べることは，割込みの波及効果を把握することであり，モデルの理解性，再利用性，信頼性を向上させるために重要である．

割込みハンドラを状態図として記述する場合，並行要素がタスクに加えてハンドラの分まで加わるため，状態図を合成してテストケースを生成する方法では非常に大きなテストケースを生成することになり，効果的でない．スライシングと合わせたカラーペトリネットのテストケース生成法により，効率的にテストケースを生成することが可能である．我々が提案してきたカラーペトリネットのスライシング方法を利用することでこの問題に対処できる[3)]．

複雑な組込みソフトウェアをタスク指向型開発で行うと，要求分析が不十分になり，再利用性が低下する[1)]．本論文で提案するテストケース生成法は，オブジェクト指向ソフトウェアについても対応している[4)]．したがって，上流でオブジェクト指向型モデリングを行い，テストケースを生成し，タスクを抽出した状態図に対して適合テストを行うことも可能である．

提案する方法をこれまでのカラーペトリネットに基づく研究成果に加えることで，以前よりも上流から下流へシームレスな開発を可能にし，プロダクトの品質の向上と短期開発に貢献する．

以下，次章では関連研究について述べ，3 章ではカラーペトリネットについて紹介する．4 章では，割込みハンドラのモデリング方法を提案し，5 章ではテスト方法とカラーペトリネットへの変換方法について述べる．6 章では，カラーペトリネットを利用した時間とメモリの見積もりについて説明する．7 章では，ストップウォッチの例に適用し，提案した方法を評価し，その結果を 8 章で議論する．

2. 関連研究

組込みソフトウェアの適合テスト，シミュレーションには実行可能な状態図が必要である．最も著名な実行可能な状態図は，UML[11)]に拡張されているHarelのステートチャート[17)]である．UMLのステートチャートは，リアルタイム組込みソフトウェアに応用できるように時間属性が添付され，拡張されている[12),13),16)]．ROOM[25)]は，割込み処理を考慮に入れている．ObTS[19),29)]はrun-to-completion型の割込み処理が可能である．これらの主な目的が，上流工程で機能を分析することにあるのに対し，我々の目的は，分析から実装へのシームレスな開発，すなわち，分析と設計間の適合テスト，および設計と実装の適合テストにある．実装で，割込みハンドラを利用することは一般的であるが，ハンドラを考慮した状態図は並行成分が多くなるため複雑度があがる．そこで，これらの実行可能な状態図上で直接テストケースを生成せずに，カラーペトリネットに変換してからテストケースを得る方法[5),9)]採用した．さらに，従来の方法では割込みハンドラと関連するタスクとの関係を明らかにすることはできないため，開発者がどのタスクのどの状態と関連するかについて把握する必要がある．本論文で提案する方法ではこの問題を解決する．

本論文で提案する割込みハンドラ状態図は，組込みソフトウェアのためのパターンの 1 つといえる．割込みパターンは文献 14) で提案されているが，具体的に状態図の形状を規定していない．割込みハンドラ状態図の生成については，我々がすでに提案しているが[2)]，カラーペトリネットへ変換する過程については述べていない．割込みハンドラを処理する部分は非常に複雑であり，これまで提案した方法では対応できない．本論文では，割込みハンドラを処理する部分と連動できるように割込みハンドラをCPNを変換する．処理部分については紙面の都合上省略し，本論文では割込みハンドラ部分のモデリングの重要性，モデリング方法，CPNへの変換方法について議論する．

3. カラーペトリネット

本論文で提案する方法は K. Jensen が提案したカラーペトリネット（Coloured Petri Nets: CPN）[21)~23)] を利用する．本章では CPN について紹介する．**図 1** に CPN の例を示す．CPN は，図 1-1 中の長方形内に記述したカラー宣言（color declaration）部分とグラフ部分からなる．カラー宣言部分では，カラーと呼ばれる型と変数の宣言を行う．図 1-1 では，int 型のカラー Int が宣言され，Int 型の変数 arg1，arg2 の宣言を行っている．

グラフ部分は図 1-1 に示すとおり，プレース，カラー，トランジション，ガード，アーク，アーク式，時間からなる．CPN では，システムの挙動を発火と呼ばれる仕組みにより表す．トランジションが発火することで，トークンはアークの方向に従いプレースからプレースへ移動する．図 1-1 のトランジション Operation が発火すると，プレース arg1 と arg2 にあるトークンはプレース Result に移動する（図 1-2 参照）．

発火は，発火可能なトランジションのうち 1 つが任意に起きる．発火可能とは，トランジションに入力しているアークのアーク式を満たすように，アークの始点となるプレースがトークンを有し，そのトークンの値がトランジションに添付されているガードも満たす場合のことである．

CPN には融合プレースという概念がある．このプレースを利用することにより，複雑な図を整理して記述したり，大域変数を表したりすることが可能である．融合プレース（FS: Fusion Sets place）の例を図 1-3 に示す．1-3 (a) と 1-3 (b) は同じ意味である．FS が添付された同じ名前のプレースは融合プレースであり，表現上は 2 個のプレースであるが，実際は 1 つのプレースである．

CPN の解析は到達グラフを用いて行う．図 1-1 に示した CPN の到達グラフを図 1-2 に示す．図 1-2 の到達グラフのノードは，CPN のマーキングの状態を示す．上のノードはトランジション Operation が発火する前の状態を表し，下のノードは発火後を表す．カラーペトリネットは，時間を扱うことが可能であるが，この機能は，複数の異なるクロック，すなわち CPU を並列に利用するシステムには適していないため，本方法ではこの機能を利用せず，時間を 1 つのカラーとして宣言する．

4. 割込みハンドラのモデリング

本章では，割込みハンドラを構築する方法について述べる．まず，割込みハンドラをモデリングする際の要件について明らかにし，次に，本論文で提案する方法が支援する部分と役割について説明する．そして，割込みハンドラ状態図，割込みコントローラ状態図の構築方法について記す．最後に，割込みハンドラ状態図の変更方法の今後の可能性について論じる．

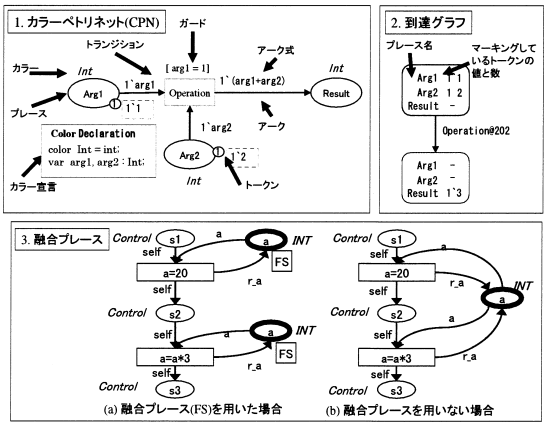


図 1 CPN，到達グラフ，融合プレースの例
Fig. 1 CPN, occurence graph and fusion sets place.

4.1 割込みハンドラのモデリング要件

割込みハンドラのモデリングは，システム全体のモデリングを行った後，タスク分割と同時に行うことが多い．割込みハンドラをモデリングするためには，まずハンドラで処理するイベントを決定する．本論文では，割込みハンドラで処理するイベントを割込みと呼ぶことにする．割込みハンドラで扱うイベントは，一般に以下の基準で選択する．

(1) 緊急性を要するイベントであること
(2) 十分に短い時間で処理できること
(3) イベントにともなう処理でデータが保護されること
(4) イベントにともなう処理に待ちをともなわないこと
(5) イベント間での直接的なメッセージ交換をともなわないこと
(6) 複数のタスクで共通に利用されるイベントであること

まず最初の基準の根拠であるが，割込みハンドラによる割込み処理の本来の目的は緊急性を要するイベントを処理することである．そのため，割込みハンドラの処理は，特別なタスク処理を除いて優先される場合が多く，オペレーティングシステムによっては，最も優先される．

2 番目，3 番目の基準を，すなわち十分に短い時間での処理を可能にし，データを保護するために，多く

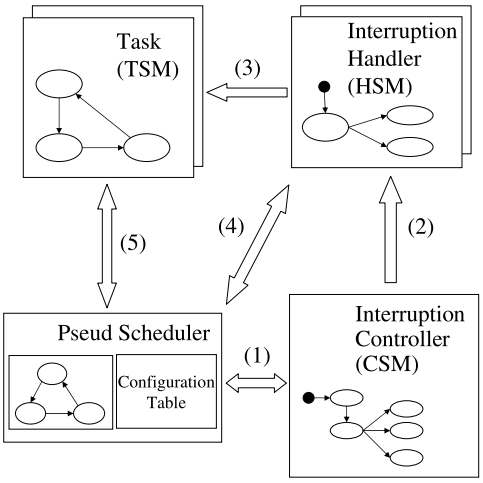
のリアルタイム OS では様々な制約事項を設けている．まず第 1 に，割込みハンドラではリアルタイム OS のシステムコールを利用することができない場合が多い．μITRON4.0[30]では，システムコール（カーネルサービスコール）を利用できる割込みとできないものを定義している．さらに，システムコールを利用できる割込みハンドラでも非タスクコンテキストとして分類し，一般のタスクで利用するシステムコールの一部のみが利用できる．また，非タスクコンテキスト専用のシステムコールも存在する．

これらの基本は，上記の基準の 4 番目と関連する．待ちをともなうシステムコール，およびデータの保護を阻害するシステムコールを利用することはできないということである．セマフォ資源の獲得やランデブーなど待ちを発生するシステムコールが存在すると，処理時間が保障できず，データの保護もできない．これらのシステムコールが利用できないことは，上記の 5 番目の事項と関連する．セマフォやランデブーなどのシステムコールが利用できないため，割込みハンドラ間の直接的なメッセージ交換はできない．

最後の基準であるが，複数のタスクで共通に利用されるイベントを割込みハンドラで対処することで，タスクで処理するよりも処理時間を短縮することが可能になる．したがって，1 つの割込みハンドラは，複数のタスクと関連を持つことになる．この場合，特に注意を払わなくてはならないことは，コンテキストスイッチには時間を有するため，OS での割込みハンドラの処理は，割込みが発生するまで動作していたタスクの資源を利用することが多いということである．したがって，割込みハンドラ起動時にタスクが利用していたメモリやスタックに割込みを処理する分の資源が残っていなければならない．

以上を考慮し，割込みハンドラをモデリングしなければならない．OS のシステムコールにともなう制約は，コンパイラで確認可能であるので，深刻な問題は起きない．上流工程で，抽象度の高いモデルを記述する際には，待ちをハンドラ内でともなわないように注意を払う必要がある．時間，データ保護に関しては，これまで我々が提案してきた方法に，本論文の 6 章で説明する時間とメモリの見積もり方法を利用すれば見積もることが可能である．

本論文では，特に，最後の基準である複数タスクに着目している．割込みハンドラは複数タスクと関連しているため，どのタスクのどの状態と関連するかを把握していなければモデリングできない．しかし，これを人手で網羅的に把握することは困難である．本論文で提案する方法では，割込みハンドラがどのタスクのどの状態と関連しているかを明らかにすることが可能である．割込みハンドラ用にメモリやスタックを準備できる OS でも，割込みがどのタスクのどの状態と関連するかを調べることは，割込みの波及効果を把握することであり，モデルの理解性，再利用性，信頼性を向上させるために重要である．また，多くのタスクと関連のあるイベントを抽出すること自体にも利用できる．



(1) Enable/Disable Interruption
(2) Receive Interruption
(3) Send Data
(4) Suspend Task, Ready Task
(5) Scheduling Tasks

図 2 割込み処理の概要
Fig. 2 Outline of interruption process.

4.2 割込みハンドラ処理の概要

本システムでは，タスク状態図から割込みハンドラ状態図と割込みコントローラ状態図を構築し，これらをカラーペトリネットへ変換することでテストケースを生成する．テストケース生成法については，5 章で述べる．

図 2 に割込み処理の概要を示す．本テストシステムでは，タスク状態図，割込みハンドラ状態図，コントローラ状態図スケジューラ間で，割込み許可・禁止，割込みの受信，データの送信，タスクの状態変化について状態図間でメッセージをやりとりして動作する．

割込みハンドラ状態図は，タスク状態図から生成し，どのタスクのどの状態と関連しているかが明示される．割込みコントローラは，ベクターテーブルを抽象化したものである．ベクターテーブルは，ハードウェアに強く依存し，再利用性がないため，抽象度を高くし，

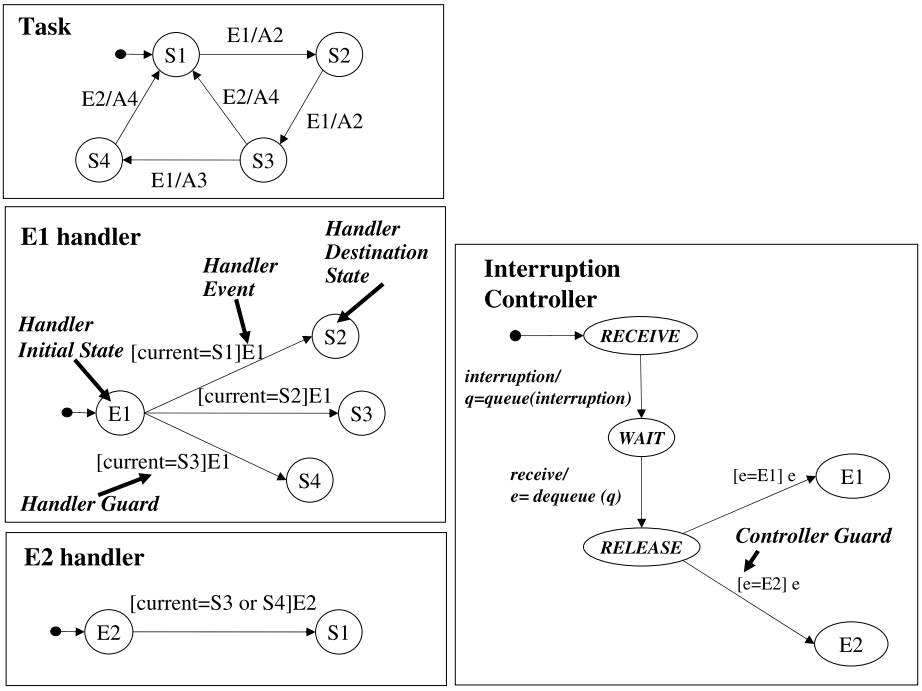


図 3 タスク状態図，割込みハンドラ状態図，割込みコントローラ状態図
Fig. 3 Task/interruption/interruption controler state machines.

割込みコントローラとした．

4.3 割込みハンドラ状態図の構築

本論文では説明を単純化するために状態図の入力イベントをすべて割込みとする．タスク状態図の集合 TSM 内のすべての入力イベントの集合を E とすると，**図 3** の入力イベントすなわち割込みの集合は，$E = \{E1, E2\}$ である．

以下，割込みハンドラの構築手順について**図 4** に示した行番号に対応して説明する．

(1) タスク内から 1 つのイベントを選択する．
各々のイベントごとに割込みハンドラ状態図 HSM を作成する．図 3 では，HSM は E1 handler と E2 handler である．
HSM は，割込み処理のための構成要素として，Handler Initial State, Handler Event, Handler Destination State, Handler Guard を持つ．

(2) Handler Initial State: i_h を作成する．
1 つの割込みに対し，1 つの i_h を作成する．図 3 の E1 handler では，$i_h = E1$ である．

(3) Handler Destination State: D_h を作成する．
イベントの遷移先を集め，Handler Destination State とする．図 3 の E1 handler では，D_h={S2, S3, S4} である．

HandlerCreation(TSM)
TSM: A set of task state machines begin
i_h: Handler initial state
E_h: A set of handler events
D_h: A set of handler destination states
g_h: Handler guard
1) Select an event $e \in E$ on TSM
2) i_h = CreateInitialState(e)
3) D_h = CollectDestinationNode(e)
4) for all $d_h \in D_h$ do
5) g_h=SourceNode(e, d_h)
6) Map(i_h, d_h, g_h, e)
end
end.

図 4 割込みハンドラの構築
Fig. 4 Handler construction.

(4) ハンドラの遷移をイベントの宛先ごとに作成する．

(5) ガード条件：g_h を作成する．
イベントとイベントによる遷移先状態の組から遷移元状態を探し，そのラベルからガード条件を作成する．すなわち，現在の状態を判断してハンドラが適切な状態へ遷移できるようにする．図 3 の E1 handler では，d_h=S2 であるとき，

g_h=“[current=S1]” である．

(6) i_h と d_h を連結し，その遷移にガード g_h とイベント e を添付する．図 3 の E1 handler で，d_h=S2 であるときは，Handler Initial State を S2 と連結し，g_h=“[current=S1]” と e=E1 を遷移に添付する．

本論文で自動生成した割込みハンドラ状態図をそのまま利用することもできるが，モデラーは必要に応じて状態を追加変更し，そのテストケースも得ることができる．本論文で提案した割込みハンドラは，実装のための最終形ではなく，詳細化を進めるための枠組みである．

4.4 割込みコントローラの構築

以下，割込みコントローラの構築手順について**図 5** に示した行番号に対応して説明する．

(1) RECEIVE 状態を作成する．
RECEIVE 状態は，割込みを待っている状態である．

(2) WAIT 状態を作成する．
WAIT 状態は，発生した割込みをイベントキューに待たせている状態である．

(3) RELEASE 状態を作成する．
イベントを受理できる通知を受け，イベントキューから割込みを外した状態である．

(4) RECEIVE から WAIT への遷移を作成する．
発生した割り込みをイベントキューに待たせる処理を行う．

(5) WAIT から RELEASE への遷移を作成する．
イベントを受理できる通知を受け，イベントキューから割込みを外す処理を行う．

(6) すべての割込みハンドラ状態図から，Handler initial state を集める．図 3 の Interruption Controller では，I= {E1, E2} である．

(7) 各々の Handler initial state ごとに遷移を作成する．

(8) ガードを作成する．
ガード式は，割込み e がどの割込みかを識別するので，Handler initial state のラベルをもとに g_c を作成する．図 3 の Interruption Controller で，$s = E1$ の場合，g_c=“[e=E1]” となる．

(9) RELEASE 状態から Handler initial state への遷移を作成する．

本論文で提案した割込みコントローラ状態図は，非常に単純であり，より，実際的な挙動のテストが必要な場合は，モデラーは開発するアーキテクチャに従い変更する必要がある．

ControllerCreation(HSM)
HSM: A set of handler state machines begin
I:A set of initial states in all handlers
E:A set of all events
g_c: Handler guard
  1) CreateNode(RECEIVE)
  2) CreateNode(WAIT)
  3) CreateNode(RELEASE)
  4) Map(RECEIVE, WAIT, “interruption/ q=queue(interruption)”)
  5) Map(WAIT, RECEIVE “receive/e=dequeue(q)”)
  6) I = CollectInitialState(HSM)
  7) for all $s \in I$ do
    8) g_c = GuardLabel(s)
    9) Map(RELEASE, s, g_c, “e”)
  end
end.

図 5 割込みコントローラの構築
Fig. 5 Interruption controller construction.

割込みコントローラのイベントは割込みベクターであり，コントローラ状態図は，割込みベクターテーブルの処理に対応した挙動を表している．

4.5 割込みハンドラ状態図の変更

4.3 節で述べた方法で構築した割込みハンドラは，変更可能であることを述べた．しかし，この方法では，状態図を変更するごとに割込みハンドラを構築することになり，開発者の変更を維持することができない．4.1 節で割込みハンドラの要件について述べたように，割込みハンドラは，短い処理，すなわち数行の処理しか記述することができず，非常に限られた命令しか使用することが許されない．したがって，割込みハンドラへの複雑な追加変更は本来してはならない．変更の多くは**図 6** に示すように，Interruption process p1 と EE1 を追加する程度である．

しかし，本節の最初で述べたように，状態図を変更するごとに割込みハンドラを修正することは労力を要する．今後，割込みハンドラで処理すべき内容を別に記述し，図 6 の p1，すなわち Interruption process 部分に合成できる仕組みをシステムに追加していく予定である．

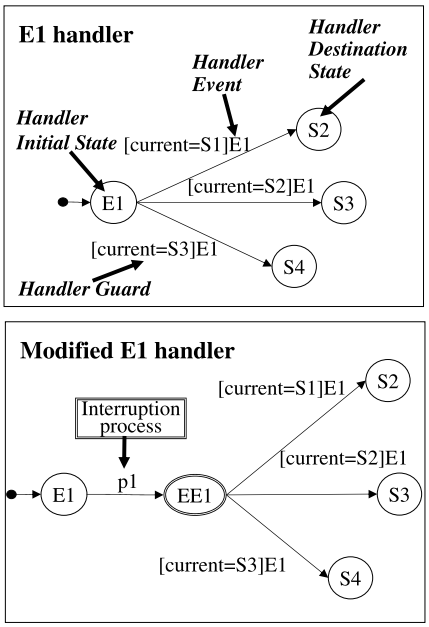


図 6 割込みハンドラ状態図の変更
Fig. 6 Modification of interruption handler.

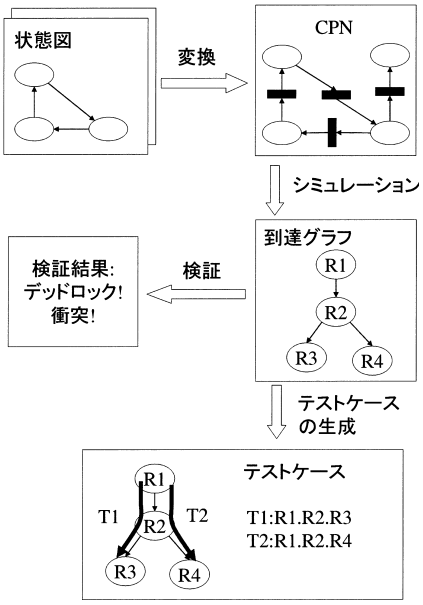


図 7 テストプロセスの概要
Fig. 7 Outline of testing process.

5. CPN を利用したテスト方法

本章では，まず想定するテスト方法の概要について述べ，次にテスト可能にするために必要な CPN への変換について述べる．最後に，時間とメモリの見積もり方法について概説する．

5.1 テスト方法の概要

本研究で利用するテストプロセスの概要を**図 7** に示す．本方法は，我々がすでに提案した方法に基づいている[5),9)]．状態図をカラーペトリネット（CPN）[21)]へ変換し，到達グラフ（Occurrence graph）へ変換する．到達グラフは 3 章に示したとおり，CPN の発火の軌跡であり，CPN の挙動を表す．変換した到達グラフからテストケースを生成する．到達グラフからテストケースを生成する方法では，CPN の発火を表す到達グラフの矢印は，テストケースの入力値に変換され，マーキングを表すノードが出力値に変換される．到達グラフを図 7 に示すようにたどることでテストケースを得る．テストケースは，プログラムが状態図の仕様書どおりに実装できたかを確認する適合テストに利用する．適合テストは，状態図から生成したテストケースの入力値をプログラムに入力し，プログラムの出力結果とテストケースの出力とを比較することで，仕様どおりに実装できたかを確認する．また，オブジェクト指向ソフトウェアからテストケースを生成する方法[4)]を利用すれば，上流でオブジェクト指向分析を行い，そこで生成したテストケースを下流でタスク指向で構築した状態図モデルの入力として用いることもできる．すなわち，オブジェクト指向分析で得た状態図モデルとタスク指向状態図モデルとの適合テストも可能である．ただし，我々が提案してきた方法では，テストケース生成の支援のみであり，比較には，シミュレーション等価性や弱双模倣性など様々な方法があるが，本方法では開発者の判断に任せている．

CPN は並行性をテスト検証するペトリネット[24)]を拡張したネット指向言語であり，階層化の概念，カラーと呼ばれる型付きの値を持つトークン（以下，カラートークン）の概念，そして時間の概念が拡張されている[21),22)]．CPN は様々な組込み機器に利用されている[23),27)]．

さらに，カラーペトリネットの同値仕様[22)]を利用することにより，状態爆発問題に対応したテストケースを生成することが可能である．到達グラフからデッドロックの検出などの可達性に関する検証することもできる[22)]．我々が提案したカラーペトリネットのスライシング技術[3)]を利用することで，機能に着目した部分テストケースを生成することも可能である．さらに，このスライシング技術は，検出した誤りについての原因追跡，誤り箇所の修正も支援する．

5.2 CPN への変換概要

図 8 に，タスクの並行動作を管理する仕組みの概要について示す．図 8 の下部は，「タスクを表す CPN」

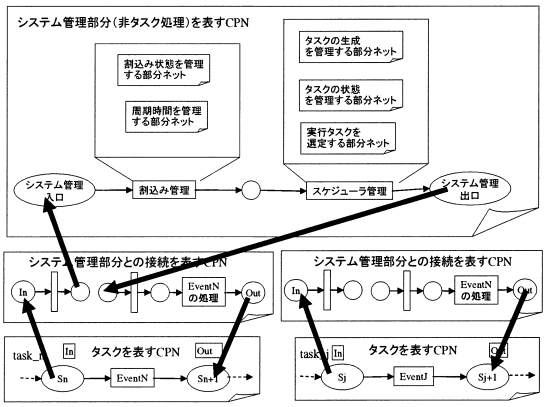


図 8 タスク管理の概要
Fig. 8 Outline of task control.

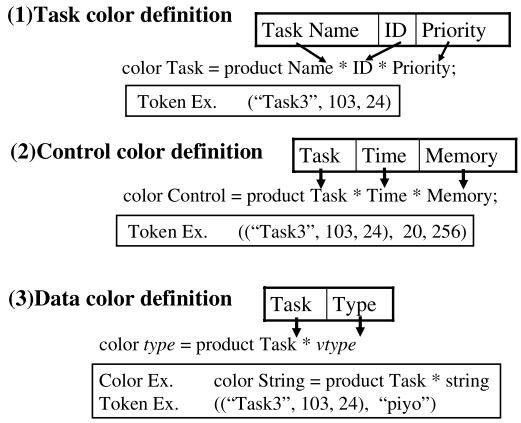


図 9 タスクトークンの定義
Fig. 9 Task token definitions.

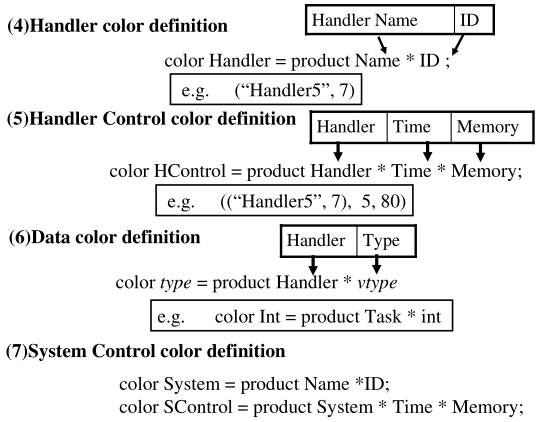


図 10 システムトークンの定義
Fig. 10 System token definitions.

であり，各タスクは各 CPN へ変換される．図中央の「システム管理との接続を表す CPN」は,「タスクを表す CPN」のトランジションを CPN の階層化機能を利用している．したがって，トランジションごとに「システム管理との接続を表す CPN」が作成される．トランジションは状態図の遷移ラベルであるイベントとアクションに相当する．すなわち，本システムでは，状態に到達すると，割込みを受け付けられる状態であるか，割込みがあるかどうか，タイムアップになっていないかなどを毎回調べる．

図 8 の上部の「システム管理を表す CPN」は，割込みを管理する部分とスケジューラを管理する部分からなる．各々は，さらに様々な役割を持つ部分 CPN からなる．これらシステムを管理する CPN は，図 2 で示した擬似スケジューラの状態図およびコンフィグレーションテーブルの内容に従い構築する．本論文では，割込みの処理方法のみに焦点を当てる．

5.3 トークン定義

図 9 に「タスクトークンの定義」，**図 10** に「システムトークンの定義」を示す．タスクを識別するトークンが Task カラーであり，タスク名，ID，プライオリティを持つ．Control カラーは，タスク内の処理をステップ実行させる役割を果たす．Control カラーの中に Task カラーの値を持つことができるため，タスク間同期を表現することができる．また，Control カラーには，時間とメモリのカラーを持つ．すなわち，時間とメモリについて計算することが可能である．また，内包している Task カラーにはプライオリティを持つため，プライオリティを動的に変更させることが可能である．データカラーは，演算に利用する．

Handler カラーは Task カラーと同様，ハンドラの識別に利用する．Handler Control は，割込みコントローラ部を処理するためのトークンカラーであり，ここでの Data カラーはハンドラ内部でのデータ処理を行う際に利用する．System カラーは，ハンドラ，タスク以外のシステム管理部分をステップ実行するためのトークンである．

5.4 CPN への変換

図 11 にタスクから CPN，**図 12** にハンドラーから CPN，**図 13** に割込みコントローラから CPN，そして**図 14** に割込み状態管理の CPN について示す．本方法では，各タスク，ハンドラごとに，各々図 11 と図 12 に変換する．ハンドラは割込みコントローラから受けた割込みに従い動作する．また，割込みコントローラは，割込み管理部で管理されている．以下，各々について述べる．演算や関数など基本的な変換規則は文献 4) に従う．

5.4.1 タスクから CPN への変換

状態図の状態は Control カラーのプレースに変換さ

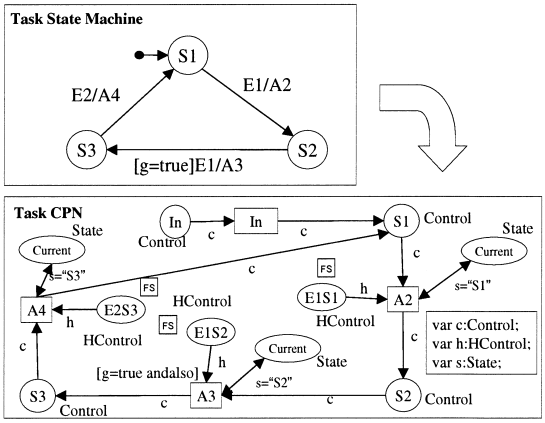


図 11 タスクから CPN への変換例
Fig. 11 An example of translating task into CPN.

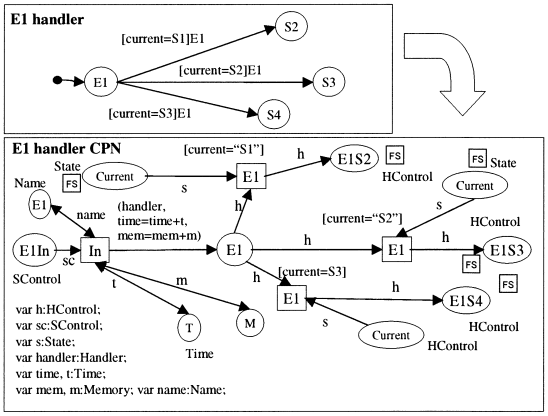


図 12 ハンドラーから CPN への変換例
Fig. 12 An example of translating handler into CPN.

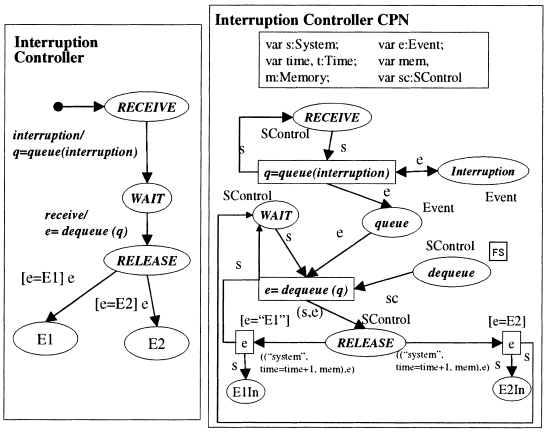


図 13 割込みコントローラから CPN への変換例
Fig. 13 An example of translating interruption controller into CPN.

れる．図 11 で，状態図の S1 は CPN の S1 である．状態図のイベントは CPN の HControl カラーのプレー

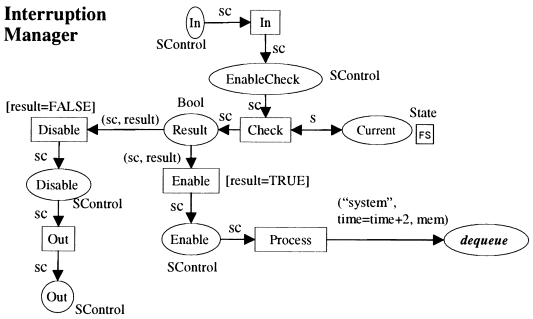


図 14 割込み状態管理の CPN
Fig. 14 Interruption manager CPN.

スに変換される．状態図の E1 は CPN の E1S1 である．E1S1 プレースは融合プレースであり，割込みハンドラと連結している．したがって割込みハンドラで処理がなされるとトークンが E1S1 プレースにやってくる．タスク内の Current プレースと割込みハンドラの Current プレースは融合プレースであるため，実際には 1 つのプレースであり．その役割は現在の状態の記録である．この記録により，発生した割込みが現在受理可能であるかを判断し，サスペンドしたタスク処理を続きから継続することを可能にしている．

本論文では議論を単純にするために，イベントをすべて割込みとして扱う．イベント部分にも式を記述することができる．割込みの場合は，イベントハンドラ内の処理として展開する．割込み以外のイベント処理および式の CPN での扱いについては文献 4) で提案している．状態図のアクションは CPN のトランジションで表される．状態図 A2 は CPN の A2 である．

5.4.2 ハンドラーから CPN への変換

図 12 の E1IN プレースで，割込みコントローラからの割込み E1 を受け付ける．割込みハンドラの処理に必要な時間とメモリを In で割り当てている．もし，割込みハンドラを表す状態図に時間やメモリが記されていれば，それらに対応するトランジションに付加される．Current プレースが有するトークンの値は，タスクの現在の状態を表すため，その値に従い，処理をタスクに戻す．

5.4.3 割込みコントローラから CPN への変換

図 13 の状態図は，割込み受理と開放がシーケンスに表現されている．実際の挙動では，各々独立に処理されるが，本方法では単純に表現している．実際にこの部分にモデラーが追加情報を記入しなければならないことは，イベントキューに連結する時間，リリースする時間であるため，記述をなるべく単純にした．CPN の方は，割込みの受理と開放は独立に動作する

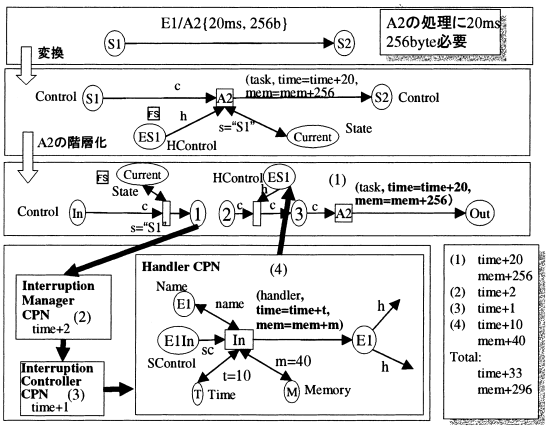


図 15 時間とメモリの見積もり例
Fig. 15 An example of time and memory estimation.

ように記述している．

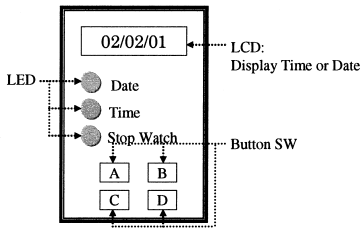
5.4.4 割込み状態管理の CPN

図 14 は，システムを管理する構成要素を表す CPN としてあらかじめ用意されている．この部分は，スケジューラ構築の際のコンフィグレーションにより変化する．割込み状態管理では，はじめに割込みを受付ける状態であるかどうかをチェックし，割り込みの受付けが可能であれば，割込みコントローラから割込みを開放する．

6. CPN を利用した時間とメモリの見積もり

図 15 に時間とメモリの見積もり例について示した．図の一番上は状態図であり，A2 の処理に 20 ms, 256 byte 必要であることが示されている．状態図は上から 2 番目の CPN へ変換され，さらに，A2 は階層化され，接続を表す CPN に変換される．ここで，接続を表す CPN を通して，スケジューラ，割込み処理がなされる．割込みを受け付けるプレースが，2 のプレースの後のトランジションに接続されている理由は，2 のプレースに，現在のタスク実行を継続するかどうかを決定する処理結果が戻されるためである．

これらの処理で消費される時間とメモリは到達グラフを生成することで得ることが可能である．図に示したとおり，タスク以外の部分でも様々な処理が必要であり，これらにも時間が必要である．本例題により，従来よりも正確な見積もりが可能であることが分かる．実際レベルで利用可能にするためには，システム管理部分に必要な処理時間は，ハードウェアや OS ごとのプロファイルが必要となる．



Initial Mode: Calendar Mode (LCD: Date)
Calendar Mode:
Push Button A
Change into stop watch mode.
Push Button B
If LCD shows date, it is changed into time.
If LCD shows time, it is changed into date.
Stop Watch Mode:
Push Button A
Change into calendar mode.
Push Button B
Start or Stop
Push Button C
Clear
Push Button D
Show lap time

図 16 ストップウォッチの例
Fig. 16 An example of stopwatch.

7. 適 用 例

本章では，提案した方法をストップウォッチの例に適用し，割込みハンドラ状態図が生成できることを確認する．**図 16** にストップウォッチの例を示す．図 16 は，ストップウォッチの構成とボタン操作について示している．ストップウォッチは，ディスプレイと 3 つのランプ，4 つのボタンからなる．ボタン操作により，日付を表示するカレンダーモードと時間を計測するストップウォッチモードを切り替える．

4 つの日付表示からストップウォッチのカウントを開始したりする．ボタン操作によるモードの切り替えディスプレイの表示変更について，図 16 の下部に示した．

ストップウォッチの機能をもとに構築したタスク状態図が**図 17** であり，タスク状態図から生成したイベントハンドラ状態図が**図 18**，**図 19** である．

本研究は，状態図からテストケース生成するツール[28]新規機能として試作中である．ツールで動作させた様子を**図 20** に示す．本ツールでは状態図を状態遷移表の形式で入力する．遷移表の中に示された矢印はテストケースの経路である．

割込みハンドラに対応しないテストケースの例が**図 21** であり，対応したテストケースの例が**図 22** である．今回の例では，タスクや割込みを管理する部分の処理を表す系列は非常に複雑であるため省略する．図 22 の例では，割込みのタイミングの様子が分かる．

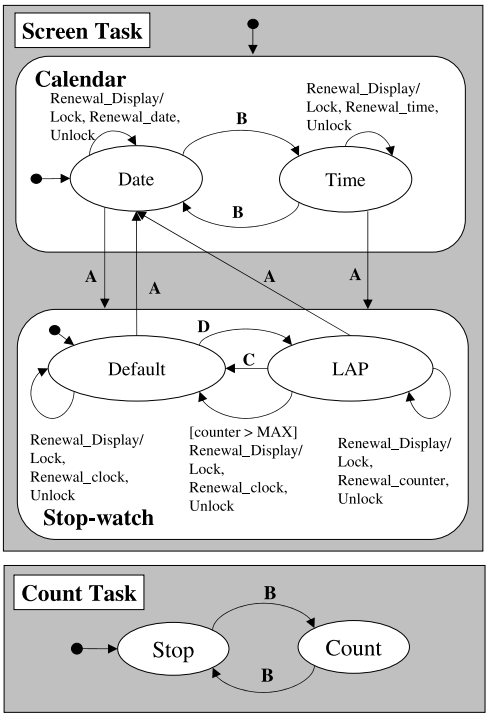


図 17 タスクの状態図
Fig. 17 Task state machines.

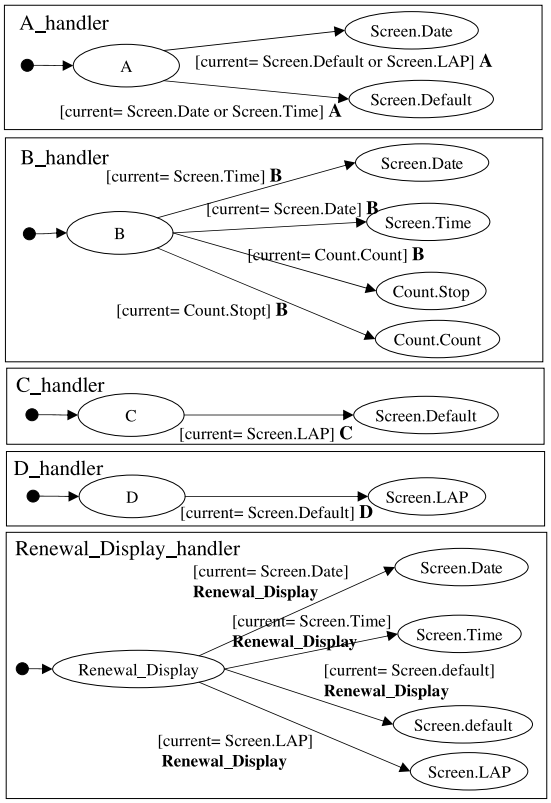


図 18 割込みハンドラの状態図
Fig. 18 Interruption handler state machines.

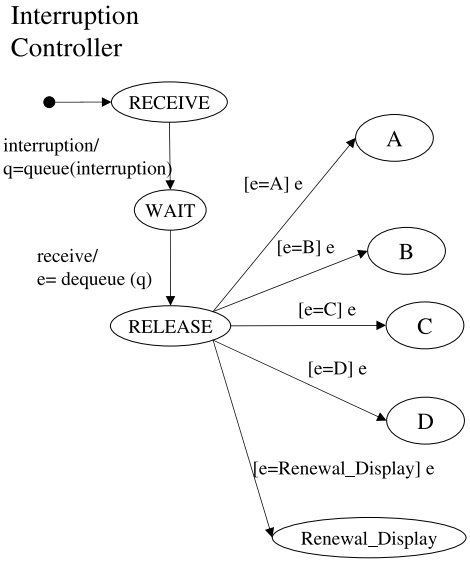


図 19 割込みコンローラの状態図
Fig. 19 Interruption controller state machine.

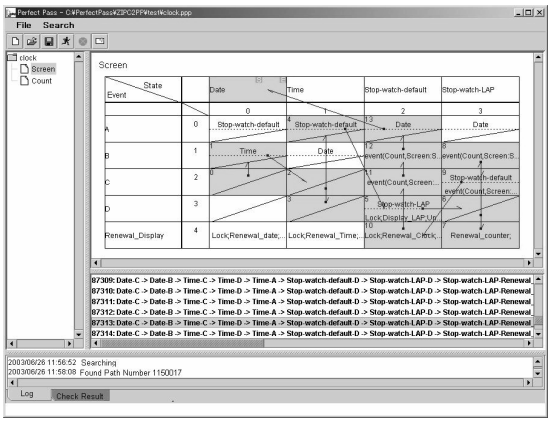


図 20 実行画面
Fig. 20 Screen shot.

ハードウェア制約を受ける組込みシステムではこのようなタイミングが示されることは非常に意味がある．タイミングによる誤差が致命的な欠陥につながるためである．さらに，図 21 に比べると実際の実行経路（＝テストケース）が詳細に表現され，割込みに費やされる以前に比べると実行時間が明確にできる．図 21 では，たった 10 個のイベント列であるが，図 22 では，30 個のイベント列からなる．また，図 18 から，割込みハンドラ状態図がどのタスクのどの状態と関連しているかを網羅していることが分かる．6 章で述べたメモリの見積もりを行えば，割込みハンドラが駆動できるメモリが残っているかを確認できる．

Date - A -> Stop-watch-default - B -> Stop-watch-default - C -> Stop-watch-default - D -> Stop-watch-LAP - B -> Stop-watch-LAP - Renewal_Display -> Stop-watch-LAP - D -> Stop-watch-LAP - C -> Stop-watch-default - Renewal_Display -> Stop-watch-default-A -> Date

図 21 タスク状態図から生成したテストケース
Fig. 21 A test case from task state machine.

Date - (RECEIVE - interruption -> WAIT - receive -> RELEASE) A -> Stop-watch-default - (RECEIVE - interruption -> WAIT - receive -> RELEASE) B -> Stop-watch-default - (RECEIVE - interruption -> WAIT - receive -> RELEASE) C -> Stop-watch-default - (RECEIVE - interruption -> WAIT - receive -> RELEASE) D -> Stop-watch-LAP - (RECEIVE - interruption -> WAIT - receive -> RELEASE) B -> Stop-watch-LAP - (RECEIVE - interruption -> WAIT - receive -> RELEASE) Renewal_Display -> Stop-watch-LAP - (RECEIVE - interruption -> WAIT - receive -> RELEASE) D -> Stop-watch-LAP - (RECEIVE - interruption -> WAIT - receive -> RELEASE) C -> Stop-watch-default - (RECEIVE - interruption -> WAIT - receive -> RELEASE) Renewal_Display -> Stop-watch-default - (RECEIVE - interruption -> WAIT - receive -> RELEASE) A -> Date

図 22 割込みハンドラに対応したテストケース
Fig. 22 A test case from state machine with interruption handler.

8. 議　　論

提案した方法により，割込みハンドラ状態図と割込みコントローラを構築することができ，割込みに考慮したテストケースが生成できることを 7 章の例で示した．例では，割込みのタイミングの様子が表せることを示した．従来の機能中心の状態図からはタイミングを見出すことができなかった．割込みに関係する処理が実行経路に詳細に表現され，割込みに費やされる実行時間を従来よりも正確に推測できるようになった．また，割込みハンドラ状態図が，どのタスクのどの状態と関連しているかを網羅できるかを示すことができた．メモリ見積もりを行えば，各々の状態において割込みハンドラが処理するメモリが残っているかを確認できる．

割込みハンドラに対応した状態図を生成することにより，状態図とプログラムとの対応付けが従来よりも明らかになり，設計から実装へのシームレスな開発の手助けとなる．実際の開発では，状態図でおおまかな挙動を理解し，タスクを利用せずに性能を向上させるために，割込みハンドラに処理の大半を記述するといったことが行われている場合がある．この場合，プライオリティインバージョンなど並行性に関する誤りが発生する．その誤りは従来一般に行われているプログラムに基づいて作成したテストケースによるテストでは再現性がないため，発見することが困難である．しかも，その場合の多くは状態図を記述せず，検証もなされていない．本方法により，タスク状態図からハンドラ状態図を生成することにより，このような良くない開発を防止することもできる．

また，タスクから自動生成する割込みハンドラ状態図に加えて異常系の割込みをモデラーが割込みハンドラ状態図として登録すれば，異常系割込みに関するテストケースの生成，シミュレーションが可能になる．

さらに，本論文で紹介した評価実験を行った際，はじめ Count Task を記述し忘れていた．割込みハンドラを抽出する段階で，実際に抜けを見つけることもできた．これは，タスクから割込みに視点が移ることにより，図 16 に示した機能との比較が容易に可能になったためである．

9. おわりに

本論文では，割込みハンドラに対応したモデル化技法とテスト方法を提案した．提案した方法により，割込みハンドラを利用する実装において，従来よりも設計と実装の関係が明確になり，設計から実装へのシームレスな開発が可能になることを示した．割込みに関係する処理が実行経路に詳細に表現され，割込みに費やされる実行時間と消費メモリを従来よりも正確に推測できるようになった．また，割込みハンドラと関連するすべてのタスクとの関連を把握しなければならなかった問題を，割込みハンドラの自動生成により，どのタスクのどの状態と関連するかを明らかにすることで解決した．さらに，評価実験において，割込みハンドラを構築する過程で，タスク側から割込み側に視点が移ることにより，状態図の誤りを発見することができた．

今後は割込みハンドラ内で，タスクを生成，削除する場合，タスクの状態を変化させる場合について検討し，時間見積り，メモリ見積もりへの影響について評価していく．また，実際的なハードウェア，OS のプロファイルを作成し，より現実的な計測を可能にしていく予定である．

参 考 文 献

1) 相沢一摩，海谷治彦，海尻賢二：組み込みシステム改造開発における機能単位設計法の適用と効果，組込みソフトウエアシンポジウム 2003

（ESS2003）論文集，情報処理学会シンポジウムシリーズ，Vol.2003, No.13, pp.8–13 (2003).
2) 渡辺晴美，川井奈央，山本修二：組込みソフトウエアのための割込みハンドラに対応したテストケースの生成，組込みソフトウエアシンポジウム 2003（ESS2003）論文集，情報処理学会シンポジウムシリーズ，Vol.2003, No.13, pp.39–46 (2003).
3) 渡辺晴美：オブジェクト指向ソフトウエアテストのためのカラーペトリネットスライシング技術，情報処理学会論文誌，Vol.44, No.6, pp.1461–1472 (2003).
4) 渡辺晴美，徳岡宏樹，Wu, W.，佐伯元司：カラーペトリネットによるオブジェクト指向ソフトウエアのテストと解析方法，電子情報通信学会和文誌 DI（情報・システム I-コンピュータ），Vol.J82-D-I, No.3, pp.1–19 (1999).
5) 渡辺晴美，工藤知宏：カラーペトリネットを用いた仕様記述からのテストケース生成方法，電子情報通信学会和文誌 A（基礎・境界），Vol.J80-A, pp.1073–1086 (1997).
6) 渡辺博之，渡辺政彦，堀松和人，渡守武和記：組み込み UML eUML によるオブジェクト指向組み込みシステム開発，翔泳社 (2002).
7) 渡辺政彦，飯田周作，石田哲史，山本修二，堀松和人，浅利康二：UML 動的モデルによる組み込み開発，オーム社 (2003).
8) Watanabe, H., Tokuoka, H., Wu, W. and Saeki, M.: A Technique for Analysing and Testing Object-oriented Software Using Coloured Petri Nets, *Asia Pacific Software Engineering Conference '98 Proceeding*, pp.182–190, (Dec. 1998).
9) Watanabe, H. and Kudoh, T.: Test Suite Generation Methods for Concurrent Systems Based on Coloured Petri Nets, *Asia Pacific Software Engineering Conference '95 Proceeding*, pp.242–251 (Dec. 1995).
10) Basanieri, F., Bertolino, A. and Marchetti, E.: The Cow_Swite Approach to Plannning and Deriving Test Suites in UML Projects, *<<UML2002>> — The Unified Modeling Language Model Enginieering, Concepts, and Tools*, LNCS2460, pp.383–397.
11) Booch, G., Rumbaugh, J. and Jacobson, I.: The Unified Modeling Language User Guide, Addison Wesley (1999).
12) Douglass, B.P.: *Doing Hard Time: Developing Real-Time Systems with UML, OBJECTS, FRAMEWORKS, And Patterns*, Addison Wesley (2000).
13) Douglass, B.P.: *Real-Time UML 2nd Edition Developing Efficient Objects for Embedded Systems*, Addison Wesley (1999).
14) Douglass, B.P.: *Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems*, Addison Wesley (2003).
15) Douglas, J.: Real-Time Manifesto, http://www.real-time.org.
16) Gomaa, H.: *Design Concurrent, Distributed, and Real-Time Applications with UML*, Addison Wesley (2000).
17) Harel, D. and Gery, E.: Executable Object Modeling with Statecharts, *ICSE-18*, pp.246–257 (1996).
18) Itou, K. and Katayama, T.: Evolutionary Development of Object Behaviors, International Symposium on Principles of Software Evolution (*ISPSE2000*), pp.65–74 (Nov. 2000).
19) Itou, K. and Katayama, T.: A Characterization of Inter-Object Behavior for High Reliability Systems, *5th International Conference on Information Systems Analysis and Synthesis* (*ISAS99*), *SCI'99* and *ISAS'99 Proceedings*, Vol.1, pp.554–561 (Aug. 1999).
20) Itou, K. and Katayama, T.: Evolution of Object System by Rewriting Behavior Expressions Internaional Workshop on Principles of Software Evolution (*IWPSE99*), pp.113–117 (July 1999).
21) Jensen, K.: *COLOURED PETRI NETS — Basic Concepts, Analysis Methods and Practical Use* — Volume 1, Springer-Verlag (1992).
22) Jensen, K.: *COLOURED PETRI NETS — Basic Concepts, Analysis Methods and Practical Use* — Volume 2, Springer-Verlag (1994).
23) Jensen, K.: *COLOURED PETRI NETS — Basic Concepts, Analysis Methods and Practical Use* — Volume 3, Springer-Verlag (1997).
24) Peterson, J.L.: *Petri Net Theory and the Modeling of Systems, Englewood Cliffs, New Jersey*, Prentice Hall Inc. (1981).
25) Selic, B., Gullekson, G. and Ward, P.T.: *REAL-TIME OBJECT-ORIENTED MODELING*, Wiley (1994).
26) Stewart, D.B. and Arora, G.: A Tool for Analyzing and Fine Tuning the Real-Time Properties of an Embedded System, *IEEE Trans. Softw. Eng.*, Vol.29, No.4, pp.311–326 (Apr. 2003).
27) CPN のホームページ，http://www.daimi.au.dk/designCPN/.
28) Perfect Pass and ZIPC のホームページ，http://www.zipc.com/.
29) ObTS のホームページ，http://kt-www.jaist.ac.jp/project/ObTS/index-e.html.
30) 坂村　健（監修），高田広章（編），μITRON4.0

仕様,（社）トロン協会 ITRON 部会 (1999).

(平成 15 年 10 月 27 日受付)
(平成 16 年 4 月 5 日採録)

渡辺　晴美（正会員）

1990 年東京工科大学情報工学科卒業．1990 年から 1993 年まで日本電気マイコンテクノロジー（株）勤務．1995 年東京工科大学大学院工学研究科システム電子工学専攻修士課程修了．1998 年東京工業大学情報理工学研究科計算工学専攻修了．博士（工学）．1998 年から 2000 年まで北陸先端科学技術大学院大学リサーチアソシエイト．2000 年から 2003 年まで立命館大学理工学部情報学科助手．2003 年から 2004 年まで南山大学数理情報学部情報通信学科客員研究員．2004 年から東海大学開発工学部情報通信工学科講師，現在に至る．ソフトウェアのテスト，組込みソフトウェア開発技術の研究に従事．日本ソフトウェア科学会会員．

川井　奈央

1976 年生，1998 年 The American University, The College of Arts and Sciences, Mathematical Major 卒業．1999 年日本マランツ株式会社入社．2002 年キャッツ株式会社入社，現在に至る．組込みのソフトウェア開発に興味を持つ．

山本　修二（正会員）

1961 年生．1986 年電気通信大学電気通信学部経営工学科卒業．1990 年京セラ株式会社入社．2001 年キャッツ株式会社入社，現在に至る．ソフトウェア工学，特にソフトウェアの論理検証法に興味を持つ．