組込みソフトウェアの制御機構に対するモデル検査の適用に関する評価実験

石黒 正揮 †1　　中島 震 †2　　梅村 晃広 †3　　田中　一之 †4

†1 (株) 三菱総合研究所　　†2 国立情報学研究所　　†3 株式会社 NTT データ
†4 日立ソフトウェアエンジニアリング

あらまし　製品化されるブルーレイディスクのモード制御機構に対してモデル検査システム SPIN を適用し、モード状態の整合性に関する検証を行った。モデル検査により、従来のテストケースによる方法では発見が困難な不具合を発見した。検証対象の形式モデリング、検証性質の記述および不具合箇所の特定についてまとめる。また、形式手法の導入によるコストと効果に関する評価結果を示す。

An Evaluation of Application of Model-Checking to Conrtol Mechanizm of Embedded Software

Masaki Ishiguro†1　　Shin Nakajima†2　　Akihiro Umemura †3
Kazuyuki Tanaka †4

†1 Mitsubishi Research Institute, Inc
3-6, Otemachi 2-Chome, Chiyoda-ward, Tokyo 100-8141, Japan
†2 National Institute of Informatics　　†3 NTT Data Corporation
†4 Hitachi Software Engineering

1　はじめに

近年、情報システムが社会に深く浸透するに伴い、ソフトウェアの不具合に起因する大規模な損害事故が増えている [1, 2]。また、組込みシステムの普及、マルチスレッドやマルチコアを利用したソフトウェアの並列処理や WEB アプリケーション等の分散処理の普及により、従来のソフトウェア・テストでは不具合の発見が困難なソフトウェアが増加している。このような背景から、ソフトウェアの正しさを論理的に検証し、その信頼性やセキュリティを確保するための方法として、フォーマルメソッド（形式手法）に注目が集まっている。　しかし、形式手法は、一部の開発現場においては適用が試みられているが、形式手法の技術的な敷居の高さや、形式手法の費用対効果に関する情報が不足しているために、ソフトウェア開発への導入が進んでいない。本研究では、実際に製品化されるブルーレイディスクのモード制御機構に対して、モデル検査システム SPIN を適用し、形式手法の有効性を実証するケーススタディを行った。その結果、従来のテストケースによる方法では発見が困難な不具合の検出や、設計仕様に関する曖昧性を取り除くなどの成果が得られた。また、形式手法の導入による費用対効果について評価し、その有効性について検討した。

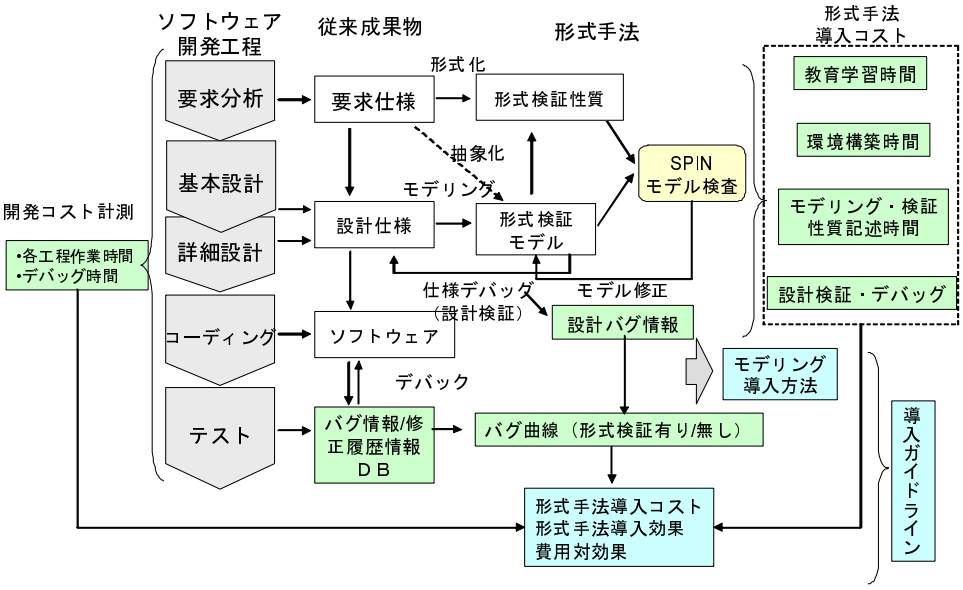

図 1: ケーススタディの概要

2　ケーススタディの概要

本研究におけるケーススタディの概要を図 1 に示す。

ソフトウェア開発工程における要求分析において要求仕様を抽出する。また、設計工程における設計書を用いて、検証対象の形式検証モデルを定義する。形式検証モデルと要求仕様に基づき、検証性質を LTL（線形時相論理）式を用いて形式的に記述する。形式検証モデルと検証性質を入力として SPIN を用いて、モデル検査を行い、不具合検出および要求仕様の検証を行う。モデル検査により検出された不具合は、形式検証モデルおよび設計仕様にフィードバックし修正を行う。以上の検証プロセスの作業時間等を記録し、形式手法の導入効果の評価に利用する。

3　検証対象

検証対象ソフトウェアは、ブルーレイディスク製品を開発するために提供されるミドルウェアである。　図2は、ブルーレイディスク全体のシステム構成を示している。システムは、ブルーレイ用ドライブ、制御用プロセッサーのハードウェアの上に、リアルタイム制御を行う RTOS が実装され、その上に Linux ベースのミドルウェアが実装される。ミドルウェアは、モード制御用の Blu-ray 系インタフェースと、Blu-Ray ミドルウェアモジュールから構成される。

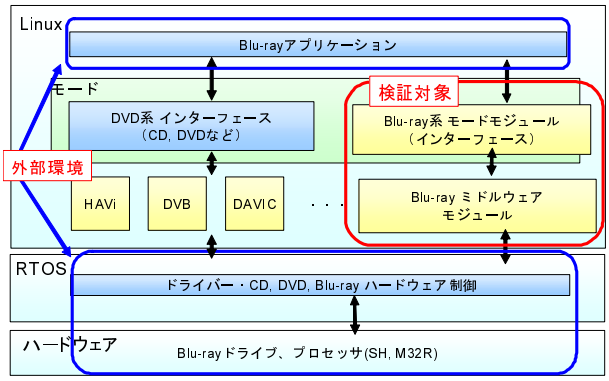

図 2: 検証対象システムの構成

システム全体のうち、Blu-Ray 系モードモジュール (インターフェース) と Blu-Ray ミドルウェアモジュールを対象として、それらのモジュールにおける再生、停止、早送り等の制御における動作モードの整合性に関する検証を行う。モードモジュールは、外部環境である Blu-Ray ディスクのアプリケーションに対して、再生、

停止等の Blu-Ray ディスク操作に関する基本的な 24 種類の API を提供する。モードモジュールは、モード制御に関わる状態を、「停止」「再生」の 2 状態で管理し、その状態に基づき、要求された API に応じて、ミドルウェアモジュールに対して要求の送信を判定する。ミドルウェアモジュールは、独自に管理するモード制御状態に基づき、ディスクプレーヤーなど実際のデバイスに対して要求を出す。ミドルウェアモジュールの状態は、モードモジュールよりも詳細に、「停止」「再生」「再生移行中」「ポーズ中」「早送り中」「正スロー中」「早戻し中」「逆スロー中」の 8 種類の状態を区別し管理する。ミドルウェアモジュールの「停止」は、モードモジュールの「停止」に対応し、ミドルウェアモジュールの「停止」以外のすべての状態は、モードモジュールの「再生」に対応する。ミドルウェアモジュールの状態は、モードモジュールからのリクエストやデバイスからの非同期な再生終了などのイベントにより遷移する。従って、モードモジュールとミドルゥエアモジュールの状態間にずれが生じることが設計段階から想定されている。

以上のような、モードモジュールおよびミドルウェアモジュールのモード状態制御において、アプリケーションから任意の API を任意の順序で送信しても、モードモジュールとミドルウェアモジュールの状態のずれにより、デッドロックや意図した処理が実行されないなどの制御機構に関わる不具合を検出することが本検証の目標である。

4 形式モデリング

本研究では、以下の方針に基づき検証目的を明確化し、検証に関連しない制御機構を捨象することにより、モデルを抽象化する。

- 本研究では、モードモジュールとミドルモジュールの制御状態の整合性について検証するため、モードモジュールをミドルモジュールを独立プロセスとしてモデリングする。また、ミドルモジュールとデバイス（デバイスドライバーを含む）は、単一のプロセスとしてモデリングする。
- デバイスからの割込みは、非決定的に発生し、ユーザ要求処理とは独立して動作するため、「コールバックプロセス」として独立プロセスとする。
- モードモジュールの並列性は、アプリケーションのスレッド数によって決まる。アプリケーションのスレッド数は任意の固定個としてモデリングする。
- アプリケーションは外部環境であり、ブラックボックスとしてモデリングするため、各スレッドからモードモジュールに任意の API 要求が発生する。

Promela を用いたシステムの形式モデリングにおいては、並行動作する主体をプロセスとして特定し、それらの間の相互作用をメッセージ通信として明確化することが重要である。検証対象とするモード制御ソフトウェアの基本構造を、プロセスとメッセージ通信を用いてモデリングしたものが図 3 である。

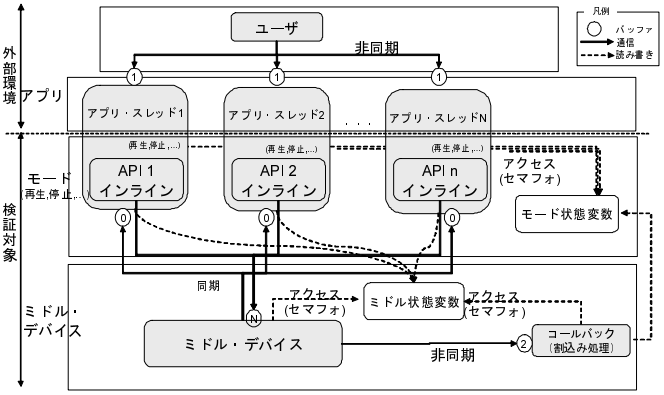

図 3: 基本アーキテクチャのモデル

プロセスは、ブルーレイディスクに対する操作要求を行うユーザ、アプリケーションを構成する複数固定個からなるスレッド、ミドルウェアとデバイスと抽象化した「ミドル・デバイス」プロセス、割込み処理を行うコールバックプロセスから構成される。検証対象として注目するモード制御は、アプリ・スレッドから同期的に呼

出されるAPI郡から構成されるモードモジュールと、モードモジュールから呼出されるミドルモジュールの２つに大きく分類される。図3の左端に示したとおり、APIインラインから下が検証対象となる、アプリ・スレッドより上位は外部環境となる。　モードモジュールは、ブルーレイディスの基本状態として、「再生」と「停止」のいずれかの状態を保持するモード状態変数を持つ。また、ミドルモジュールは、モード状態変数よりも、詳細なブルーレイディスクドライブの状態として、「停止」、「再生移行中」、「再生」、「早送り」、「早戻し」、「正スロー」、「逆スロー」などの状態を持つ。コールバックプロセスは、ディスクプレーヤーなどのデバイスからの割込みに応じて発生する状態遷移を実現するプロセスである。モードモジュールに相当するAPIインライン、ミドル・デバイス、コールバック等のプロセスの状態遷移を、設計書に基づき状態遷移表として表現し、Promelaによる記述を行う。

5　検証仕様の形式記述

モードモジュール、ミドル･デバイスモジュール、コールバックモジュールに関して、Promelaによる形式記述のポイントをまとめる。

モードプロセスは、アプリケーションに対してブルーレイディスクの機能を提供するAPIをモデル化する。アプリスレッドに対応した独立したプロセスから、実行されるインライン形式で定義される（図４）。

各APIからは、プロセス全体で共有されるモード状態変数に対して、セマフォを用いて排他制御によるアクセスを行う。

　ミドル･デバイスプロセスは、モードプロセスからの要求を受け、デバイスでの処理に係わる制御をモデル化している。モデルは、ブルーレイディスク関する操作リクエストに係わる定義と、デバイス割込みの発生に係わる定義を行っている (図 5)。

コールバック・プロセスは、ブルーレイディスク・ドライブから発生するハードウェア割込みに対する処理を行う制御機構をモデル化している。割り込みの発生は、ブルーレイディスクが、メディアの終わりに到達し、再生から停止に制御が移る場合などに発生する。　ここでは割り込みの種類は、モード状態が、再生に移行する割込みと停止に移行する２種類の割込みをモデル化している。

```
inline modef_play() {     /* API再生 */
  Mode_req ! midf_play(resp_chan); resp_chan ? ret ->
                                   /* ミドル関数に対するリクエストの送受信 */
progress_modef_play:
  if
    ::(ret == SUCCESS) ->
      P(Sem_mode);                 /* セマフォを用いてモード状態を排他的に更新 */
      Mode_state = mode_play;
      V(Sem_mode)
    ::(ret == ERROR)->  /*エラーを返す */
      skip           /* mode_stateは変更しない */
  fi
}
inline modef_stop() {  /* API停止 */
  if
    ::(Mode_state == mode_play) ->  /* モード状態が再生の時  */
      Mode_req ! midf_stop(resp_chan); resp_chan ? ret ->
      if
        ::(ret == SUCCESS) ->
          P(Sem_mode);             /* セマフォを用いてモード状態を排他的に更新 */
          Mode_state = mode_stop;
          V(Sem_mode)
        ::(ret == ERROR)-> skip /*エラーの場合  */
      fi
    ::(Mode_state == mode_stop) -> skip /* モード状態が再生の時 */
  fi
}
```

図 4: モードモジュールの形式記述 (一部)

```
active proctype middle()   /*ミドル･プロセス（ミドル+デバイス）Middle Process */
{
  mtype req_type;   /* モードからミドルへのリクエスト */
  chan resp_chan;   /* モードから渡される応答用のチャネルへの参照変数  */
end_middle:
  do
  ::true ->  /*  (1)モードからのリクエストを処理する場合  */
      Mode_req ? req_type(resp_chan) ->
      P(Sem_mid);
      atomic {
        if  /* ミドル要求イベントによる場合分け（13通り） */
          ::(req_type == midf_play) ->  /* (A)ミドル再生要求の時 */
          if   /* ミドル状態による場合分け（9通り） */          アプリケーション
            ::(Mid_state == mid_stop) ->     /* (a)ミドル状態が停止の時  */   リクエスト処理
                Mid_state = mid_to_playing;
                resp_chan ! SUCCESS
            ::(Mid_state == mid_play) ->     /* (b)ミドル状態が再生の時 */
                Mid_state = mid_to_playing;
                resp_chan ! SUCCESS
-------------------------- 省略 --------------------------
  ::true ->  /* (2) デバイス割込みで状態遷移する場合(非決定的に発生) */
    if  ::(Mid_state == mid_stop) ->     /* (a)ミドル状態が停止の時 */
          skip                          /* 状態遷移は無し */
        ::atomic{(Mid_state == mid_play) ->    /* (b)ミドル状態が再生の時 */
              P_in_atomic(Sem_mid);                                  デバイス割込み
              Mid_state = mid_stop;                                  処理
              V(Sem_mid);
        }
          Mid_req ! CB_stop
```

図 5: ミドルモジュールの形式記述 (一部)

6　モデル検査

6.1　到達性解析

到達性解析により、Promelaで記述した状態遷移システムに関して、意図しない状態における停止や到達しない状態を自動的に検出する。前者は、デッドロックの検出を意味し、後者はデッドコード（到達しないコード、起こりえない場合分け）を意味する。デッドロックの検出は、後述するLTL式による記述は難しい。到達性解析は、LTL式を与えることなく、対象システムのモデルを記述するだけで、自動で検証を行うことができるため、最初に行う検証として有効である。また、並行プログラム、マルチスレッドプログラムにおける原因不明のストールは、デッドロックによることが多いため、到達性解析の有用性は高い。本ケーススタディでは、到達性解析により、デッドロックが発生することが検出された。原因は、ミドルモジュールの状態変数に対して、ミドルモジュールがアクセスのためにロックをかけた状況で、割込みの発生によりコールバックプロセスが、その状態を書き換えようとすることが原因であった。このような状況を解決することにより、デッドロックが発生しないことが確認された。

6.2　LTL式の検証

LTL式を用いることにより、到達性解析および進行性解析では検査できない具体的な機能に係わる検証を行うことができる。本研究では、開発現場からの開発中のソフトウェアの検証に関する要望に基づき以下のような性質について形式的な記述を行い検証を行った。

6.2.1　状態のズレによる不具合の不在性

モードモジュールとミドルモジュールの状態のずれにより問題が生じないかどうか検証する。一般的に表現すると、「ミドルが処理できない要求をモードが流しても、いつかは待機状態に戻る。」となる。具体的には、ミドル状態と、モードからミドルへの要求イベントの組合せに対して検証することができる。たとえば、「いつでも、ミドル状態が「停止中」で、かつ、ミドルに「ポーズ」要求が来ても、いつかは待機状態に戻る」を検証すればよい。この性質はPromelaを用いて以下のように記述した。

```
[]((Mid_state==mid_stop) &&
   (Mode_req ? [midf_pause(ch1)])  ->
    <>(Mode_state == mode_stop))
```

6.2.2　ミドルへのリクエストの制約

ミドル状態において処理できない要求を、モードは流さないかどうか検証する。実際には、反例が存在することが予想されていたため、個々では反例を発見することを目的とした検証である。具体的には、「ミドル状態が「再生移行中」で、かつ、モードがミドルに「ポーズ」要求を出していることはない。」ことを検証すればよい。Promelaを用いて以下のように記述した。

```
[]!((mid_state == mid_to_playing) &&
    (Mode_req ? [midf_pause(ch1)]))
```

6.2.3　モード状態とミドル状態の一致性

モード状態とミドル状態はずれることが無いかどうか検証する。実際には、反例が存在することが、予想されていたため、ここでは、反例の具体例を発見することが目的である。

6.2.4　機能の実行到達性

ある操作を実行したときに、いずれはその機能が実行されるという、一般的な機能要件を検証したい。具体的には、「モード状態が「停止中」の時、いつでも「再生」Iを要求すれば、いずれは再生ミドル関数が実行される。」を検証すればよい。Promelaを用いて以下のように記述した。

```
[]((Mode_state == mode_stop) &&
   (Mode_req ? [midf_play(ch1)]) ->
   <>(Mid_state == mid_play))
```

以上の検証項目の内、機能の実現性について、アプリケーションから再生 API を実行したにも関わらず、再生が実行されない場合が存在することが発見された。原因は、「再生移行中」に別の API を実行することにより、再生に至らないケースであることが分かった。これに関しては、一般的な利用環境で、「再生移行中」の短い時間に別の操作をすることは通常考えられないため、許容できる仕様とみなされた。

7　効果の評価

情報家電メーカーは、出荷後に不具合が発見されることによる損害リスクを抱えている。形式手法の適用によりソフトウェアの信頼性が向上することで期待される効果を試算する。出荷後の不具合による改修コストが発表されている携帯電話の場合を参考とする。携帯電話の場合、改修コストは 120 億円かかっている。本検証の対象ブルーレイディスクの場合、ミドルウェアとして複数の製品で利用されることから出荷数は多く、また、一台当りの価格も携帯電話より大きい。そこで、本事例の場合の出荷後の改修コストを $C = 100$ 億円と仮定する。

出荷後に不具合が発見される確率 p、従来のテスト後に潜在する不具合のうちモデル検査により不具合が検出される比率 q は、一般に決定できないため、ここでは以下のようなパラメータ値で仮定する：

$$p = \frac{1}{50} \quad (1)$$

$$q = \frac{4}{5} \quad (2)$$

モデル検査により低減される不具合の発生による損害額の期待値 L は、

$$\begin{aligned} L &= 100(\text{億円}) \times p * q \quad (3) \\ &= 1.6(\text{億円}) \end{aligned}$$

形式検証の人件費を 1 人月 200 万円と仮定すると、不具合発生による損害額の回避による効果は、約 53.3 人月の人件費を投じても割に合うことが計算できる。実際には、不具合発生により損なわれる企業のブランド価値 [1, 2] を考慮すると、これ以上の投資を行う効果が得られると判断できる。

7.1　まとめ

製品化される Blu-Ray ディスクに対してモデル検査による検証のケーススタディを実施し、以下のような成果が得られた。

- 従来のソフトウェアテストは発見が困難な不具合を検出

  ユーザ要求とデバイス割込み処理における状態変数への排他アクセスにより、デッドロックが発生する可能性を検出した。また、操作の順序により、要求された処理が実行されない状況が存在することが検出された。

- 設計仕様の曖昧性を 60ヶ所明確化

  設計仕様の作成が収束した後、形式モデリングのプロセスにおいて、設計仕様に関する 60 項目程度の曖昧性を排除することができた。これにより、開発者の間でシステムに対する正確な理解が可能になった。

- コスト削減、信頼性向上等

  テスト工程におけるバグ発生の収束期間の評価から、テスト工程期間が 2 ヶ月から 1 ヶ月に短縮されることが確認できた。また、出荷後の不具合による損害リスクを 1.6 億円回避できることが確認できた。

参考文献

[1] Ishiguro, M., Tanaka, H. and Matsuura, K.: The Effect of Information Security Incidents on Corporate Values in the Japanese Stock Market, *The Workshop on the Economics of Securing the Information Infrastructure (WESII)* (2006).

[2] 石黒正揮：IT 事故および情報セキュリティ対策の企業価値に及ぼす影響，経済産業省リスク定量化ワークショップ (2007).