社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2008－SE－161 (7)
2008／9／26

モデル検査のためのアスペクト指向メカニズム切り替え手法の提案

金井勇人† 岸 知二†

本稿では，UML 設計モデルを対象として，処理の方式であるメカニズムを手続き的に切り替えて，モデル検査する手法を提案する．検査対象モデルを変更し，繰り返し検証することはよく行う作業である．特に分散している変更箇所は，体系的な方法を用いて正確に変更を行うことが重要である．本研究では，組み込みソフトウェアのメカニズムに注目し，手続き的に変更可能なアスペクトテンプレートを利用した手法を提案する．

Aspect-Oriended Method to Change Mechanisms for Model Checking

HAYATO KANAI† and TOMOJI KISHI†

In this paper, we propose method to systematically change mechanisms for UML software design verification utilizing model checking techniques. We are heavy in verifying changing verification model at many a time. In particular, it is important that systematically changing crosscut points. We introduce that systematic changing aspect template for embedded software.

1. はじめに

近年，組み込みソフトウェアの開発・検証手法は従来のものでは十分に対応しきれなくなっている．その背景として，様々なところで組み込みソフトウェアが使用されるようになり，その信頼性が社会的な問題となっていることが挙げられる．また組み込みソフトウェアは大規模，かつ複雑になってきており，従来の開発手法の限界が指摘されている．よって，経験則による手法だけでなく，形式的手法等，科学的手法の導入が期待されている．例えば我々は高信頼性組込み用オブジェクト指向設計技術プロジェクトにおいて，形式的検証手法の適用について研究を進めている．

形式的検証手法の１つにモデル検査技術[2)]がある．この検証技術は，検証対象を表現する有限状態モデルが，論理式で表現された性質を満たすかどうかを状態の網羅的な探索によって検証を行う技術のことである．モデル検査技術を UML 等で記述される設計モデルに適用することにより，ソフトウェアの信頼性を高めることが期待されている．

検査対象モデルを変更し，繰り返し検証することはよく行う作業である．このとき，その変更ごとに毎回検査対象モデルを手動で作成することは，非常に煩雑な作業である．大きな検査対象モデルになれば，なお変更箇所が多くなり，正確に変更が行われていることすら，疑わしくなる．このような作成者の意図が正確に反映されているか分からない検査対象モデルを検査することは無意味である．そのため，体系的な方法を用いて変更を自動的または手続き的に，間違いなく正確に行うことが重要である．

本研究では，その変更点となるものとして，メカニズムに注目する．本研究におけるメカニズムとは，ある機能を実現する方式を指す．例えば，排他的リソースを取得する場合を考える．今排他的リソースがロックされている時，そのセマフォを取得しようとしたタスクが，そのセマフォを開放するまで，待つ場合と待たない場合といった方式が考えられる．

本研究では，実験的に組み込みシステムのドメインにおいて典型的なメカニズムを対象として，ベースとなる検査対象モデルをユーザーが直接変更することなく，手続き的に変更する方法を提案している．具体的には，その変更方法をアスペクトのテンプレートとして定義し，そのテンプレートをユーザーの検査対象モデルに手続き的に融合させる手法である．

2. モデル検査技術による設計検証

モデル検査技術による設計検証の概要を説明する．

† 北陸先端科学技術大学院大学
Japan Advanced Institute of Science and Technology

なお，本稿はモデル検査ツールとして SPIN[1] を利用しており，以下それを踏まえて説明する．

2.1 モデルと性質の記述

2.1.1 仕様記述言語 PROMELA

SPIN では，対象システムを仕様記述言語 PROMELA で記述する．対象システムは並行動作する複数のプロセスの集合としてモデル化される．下記は PROMELA におけるプロセス宣言の例である．

```
proctype Example(){
    int n;
    n=1;
}
```

2.1.2 ソフトウェア設計に対する検証手順

本研究で想定している検証手順例を示す．設計モデルとして UML の利用を前提とする．

(1) モデル化
UML 設計モデルをモデル検査技術で使用できる形にモデル化する．このモデルをモデル検査ツール SPIN が使用できる仕様記述言語 PROMELA に変換して検査を行う．なお，本研究では UML 設計モデルとしてクラス図とステートマシン図を用いる．

(2) 性質の記述
システムが満たすべき性質，もしくは満たしてはならない性質を時相論理式 LTL を用いて記述する．この LTL は PROMELA で記述された対象システムの性質を記述するものであるため，その記述は PROMELA の記述に依存する．

(3) メカニズムの変更
同じ機能でも色々な方式で実装することが考えられる．つまり，メカニズムの方式を変更しては，検証することを繰り返すので，(3) と (4) は繰り返し行われる．

(4) 検証
記述した性質が成立するか，もしくは成立しないかをモデル検査ツールを用いて検証する．

3. アスペクトメカニズムテンプレートの提案

3.1 対象メカニズムの抽出

今回は組込みソフトウェアのメカニズムに注目した．以下のような抽出作業を行った．

(1) メカニズムの収集
MARTE[3] は OMG が提唱している組込み向け UML プロファイルである．MARTE では，組込みシステムでよく利用されるメカニズムが分類されている．本研究ではその中から，モデル検査，振る舞いの検証に影響あるメカニズムに注目し，抽出した．

(2) メカニズムの選定本稿では，動的なメカニズムの変更に注目する．そのため，(1) で収集したメカニズムから μ ITRON[4] の変更サービスコールがあるものを抽出した．それは，μ ITRON のサービスコール定義されているということは，組込みソフトウェア全般で動的に変更が必要なものであると考えることができるからである．本稿では，その抽出されたメカニズムの中から排他的リソースの取得方法のみを対象とした．

3.2 本テンプレートの前提

本手法では，メカニズムの表記は 3.1 節で述べた MARTE の表記を利用する．利用するステレオタイプ，タグ値の意味をそれぞれ表 1，表 2 に示す．

表 1 クラスのステレオタイプ記述と意味

ステレオタイプ	意味
⟨⟨SwSchedulableResource⟩⟩	並行動作単位
⟨⟨SwMutualExclusionResource⟩⟩	排他リソース

表 2 タグ値の記述と意味

タグ値	意味
acquireServices	SwSchedulableResouce を取得するサービス

それぞれ具体例を挙げると，⟨⟨SwSchedulableResource⟩⟩ はタスク，⟨⟨SwMutualExclusionResource⟩⟩ はセマフォである．本稿では，タグ値はコメントで表記する．

3.3 メカニズム切り替えの問題点

1 章で触れたが，排他的リソースの取得方法の例を具体的に説明する．例題のクラス図を図 1 に示す．ステート図は図 2 にクラス A のみ示す．

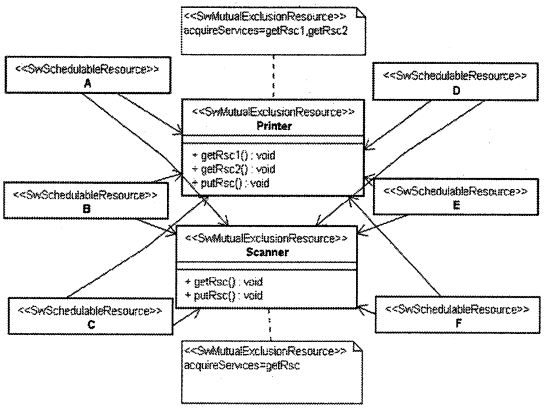

図 1 例題:クラス図

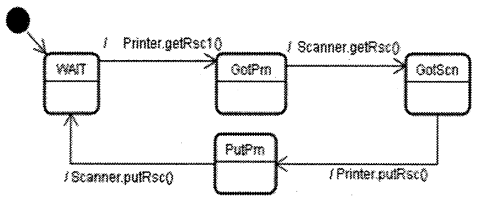

図 2 例題：A のステートマシン図

この場合，次の変更することを考える

- Printer に操作 getBlcRsc() を追加し，ステレオタイプ ⟨⟨SwSchedulableResource⟩⟩ のクラスが状態 WAIT のときのみ，操作 getRsc1() ではなく操作 getBlcRsc() を呼ぶように変更する．

この場合，ステレオタイプ ⟨⟨SwSchedulableResource⟩⟩ のクラスが 6 つあり，その中で，全ての操作 getRsc1() の呼び出しを変更するのではなく，呼び出し側の状態 (この場合，WAIT) と呼びだされるクラス (この場合，Printer) を考慮して変更しなくてはならない．このような複数箇所の複雑な条件下の変更は，非常に煩雑な作業であることが分かる．

3.4 アスペクトメカニズムテンプレートのねらい

ねらいの 1 つ目は，アスペクトの利用である．なぜ，アスペクトを利用することが必要なのかというと，前節でも述べたが，多くの箇所に変更箇所が分散している場合，体系的に変更する方法が必要なためである．

2 つ目は，テンプレート化である．本テンプレートの特徴は組込みソフトウェアでよく利用されるメカニズムのアルゴリズムとそのメカニズムが変更される箇所，変更方法を合わせてテンプレート化している点である．ユーザー毎に可変性のある部分は，変更される箇所を示す pointcut の部分だけである．pointcut の部分以外は，テンプレートにより定型化されたものを利用すればよい．また，pointcut においてもいくつかのテンプレートを提供している．よって，テンプレートを利用することでユーザーが行うメカニズム変更作業が軽減すると考えられる．

本研究で提供するものは次のものである．

- アスペクトメカニズムテンプレート
- UML から PROMELA への変換規則
- アスペクトメカニズムテンプレートからアスペクト PROMELA への変換規則

このテンプレートを利用した検証対象モデル変更手順は以下のようになる．(図 3)

(1) UML から PROMELA への変換規則を用いて PROMELA へ手続き的に変換する．

(2) アスペクトメカニズムテンプレートからアスペクト PROMELA への変換規則を用いて ASPECTPROMELA へ手続き的に変換する．

(3) (1) と (2) で変換したものを AspectPROMELA のウィーバーに入力する．

(4) AspectPROMELA のウィーバーによって，自動でウィービングされた PROMELA が生成される．

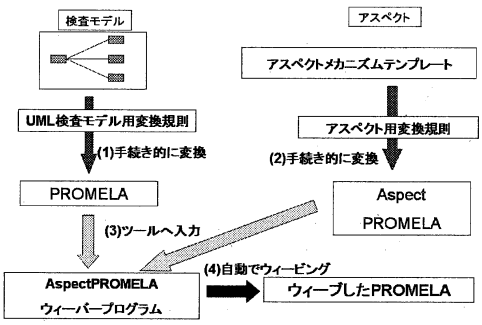

図 3 本手法を利用した場合の手順

以上により変更は手続き的に変更することができる．

3.5 テンプレートに利用する関連技術

3.5.1 UML アスペクト記述

本稿で利用する UML アスペクト記述は Stein ら[6] により提案された UML 上での AspectJ のための記述方法である．AspectJ で定義されている advice，pointcut，introduction を UML 上で表現できるようにしている．

図 4 に例を示す．

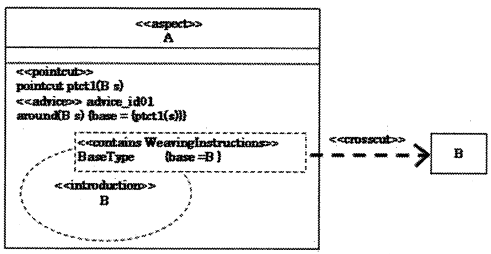

図 4 UML アスペクト記述例

pointcut，advice はそれぞれ，ステレオタイプ ⟨⟨pointcut⟩⟩，⟨⟨advice⟩⟩ の部分である．introduction は UML 表記の template で記述される．この template は，この手法では特別なバインディングメカニズムを利用している．通常は，1 つのモデル要素に対

して，バインドされる1つのパラメータが決まるが，Aspectの特性上，1つのモデル要素に対して，バインドされる複数のパラメータを取る場合がある．そのため，パラメータは1つのモデル要素ではなく，モデル要素の集合を示している．タグ値baseへの値がそれであり，この例では，クラスBになる．

introductionの例を図5に示す．

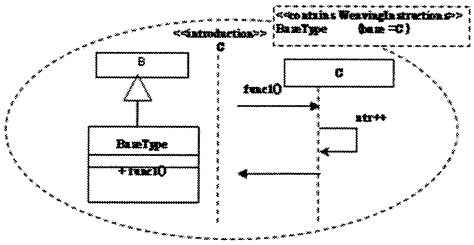

図5 introduction例

3.5.2 アスペクトPROMELA

アスペクトPROMELAは，大野ら[5]により提案されたアスペクト指向PROMELA言語である．特徴としては，AspectJと同様joinpointモデルを採用している．以下に記述例を示す．

```
around :
allStmnt(proctype("A"))
    && chan("ch","!","")
{# ch!msg1 #}
```

1行目がアドバイス，2，3行目がpointcut，4行目が挿入文になる．この例の意味は，"proctype A内のチャネルchを使ったメッセージ送信全てを，ch!msg1に置き換える"となる．

4. アスペクトテンプレートを利用したメカニズム切り替え手法

4.1 アスペクトメカニズムテンプレート

本研究のアスペクトテンプレートは2つに分けられる．

- アスペクト構造テンプレート
- pointcutテンプレート

次の小節より，それぞれについて説明する．

4.1.1 アスペクト構造テンプレート

本稿では排他的リソースの取得方法についてのみ，テンプレート化を行った．排他的リソースの取得方法については，以下の3つについてテンプレート化した．

- ブロック方式テンプレート
- ノンブロック方式テンプレート
- 条件付ブロック方式テンプレート

この3つのアスペクトは共通の抽象アスペクトクラスを継承する．その抽象アスペクトクラスを図6に示す．

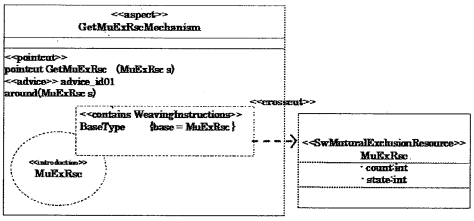

図6 テンプレート共通親クラス

図7にMuExRscのパターンステートマシン図を示す．これは，属性countとstateの関係を定義するためである．つまり，ユーザーはcountが0の時，stateはLOCKになる，countが1以上の時，stateはUMLOCKになることを守らなければならない．

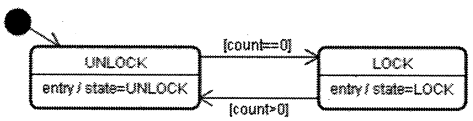

図7 MuExRscのパターンステートマシン図

ポイントカットは実装なし．アドバイスも挿入部分は実装なし．

次に，各テンプレートの内容を説明する．ここでは，ブロック方式テンプレートのみ，説明する．図8に示す．

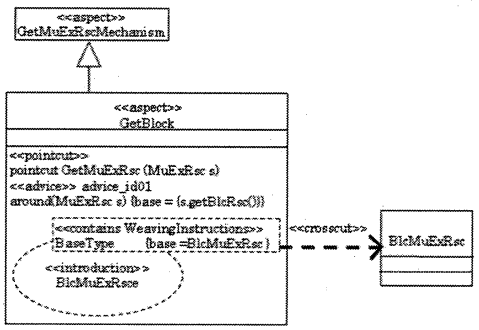

図8 ブロック方式アスペクトテンプレート

先ほどのGetMuExRscMechanismを継承して，テンプレートを記述している．アドバイスの挿入部分を

実装している．ブロック方式のアルゴリズムは，以下のシーケンス図によって記述されている．図９に示す．

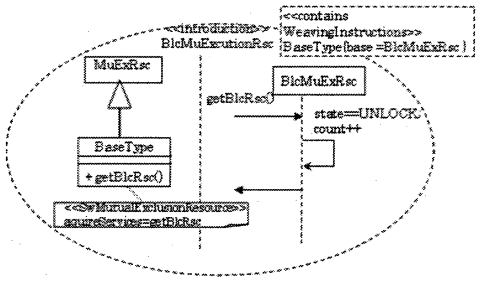

図 9 introduction

ユーザーの利用例を図１０，１１に示す．図のように利用したい方式のアスペクトクラスを継承し，pointcut のみをユーザーが考えて記述すればよい．

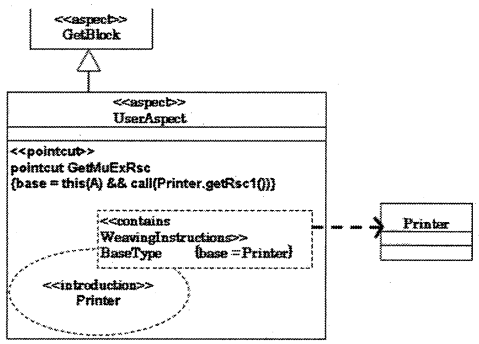

図 10 ユーザーの利用例

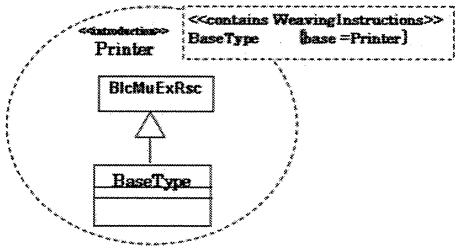

図 11 ユーザーの利用例

4.1.2 pointcut テンプレートと変換規則

pointcut はユーザーが記述する必要がある．その記述を支援するため，pointcut のテンプレートを提供する．このテンプレートに当てはめて，pointcut を記述すれば，手続き的に AspectPROMELA へ変換することができる．pointcut のテンプレートを以下に示す．

(1) 特定の排他的リソースを取るアクションの呼び出し全て

アスペクトテンプレートでの記述
call(X.acquireServices ())
(X には，任意の排他的リソースクラス名が入る)

アスペクト PROMELA へ変換後
around : label("_acquireServices_ X ")
{# Y (X) #}
(Y には，getBlcRsc，getPolRsc，getConRsc のいずれかが入る)

(2) 特定の排他的リソースを取るアクションの特定の呼び出しに限定

アスペクトテンプレートでの記述
call(Z.Y ())
(Y には，getBlcRsc，getPolRsc，getConRsc のいずれかが入る)
(Z には，任意の排他的リソースクラス名が入る．)

アスペクト PROMELA へ変換後
around : label("_Y_Z ")
{# X() #}
(X には，getBlcRsc，getPolRsc，getConRsc のいずれかが入る)

(3) 特定の状態に限定

アスペクトテンプレートでの記述
if(Y.state==X) &&
(Y には任意のクラス名が，X にはそのクラスの任意の状態名が入る)

アスペクト PROMELA へ変換後
allStmnt(proctype("Y")) && allStmnt(label("X")) &&

(4) 特定のクラスに限定

アスペクトテンプレートでの記述
this(X) &&

(X には任意のクラス名が入る)

アスペクト PROMELA へ変換後

allStmnt(proctype("X")) &&

4.1.3 UML から PROMELA への変換規則

この変換規則はクラス図とステートマシン図を対象にする.

クラス単体の変換規則

メンバ変数は proctype 内の変数に変換する.. クラス自体は次のようにステレオタイプに依存し変換する.

- ⟨⟨SwSchedulableResource⟩⟩ → proctype クラス名
- ⟨⟨SwMutualExclusionResource⟩⟩ → proctype クラス名

操作も次のようにステレオタイプに依存し変換される.

- ⟨⟨SwSchedulableResource⟩⟩ クラスの public 操作 → mtype= {操作名}
- ⟨⟨SwShedulableResource⟩⟩ クラスの private 操作 → インライン関数 クラス名操作名 ()
- ⟨⟨SwMutualExclusionResource⟩⟩ クラスの acquireServices である操作 → インライン関数 操作名 (byte rscCnt)
- ⟨⟨SwMutualExclusionResource⟩⟩ クラスの public 操作 → mtype= {操作名}
- ⟨⟨SwMutualExclusionResource⟩⟩ クラスの private 操作 → インライン関数 クラス名操作名 ()

クラス関連の変換規則

双方向の場合 2 つのチャネルに変換する. 以下のように非同期, 同期, 関数呼び出し, チャネルの共有, 非共有はステレオタイプに依存し, 変換される.

- ⟨⟨SYNC⟩⟩ → chan 送信クラス名_受信クラス名=[0] of {mtype, 引数の型 1, ••• }
- ⟨⟨ASYNC⟩⟩n → chan 送信クラス名_受信クラス名=[n] of {mtype, 引数の型 1, ••• }

ステートチャート図の変換

状態はラベルに変換し, その状態を全体をで囲む.

また, entry アクションは状態のの最初の行, exit アクションは他の状態に遷移する goto 文の直前の行に変換する.

アクションの変換について以下に示す.

- acquireServices である操作呼び出し

  クラス名. 操作名 () → _acquireServices_クラス名:_操作名_クラス名:

- ⟨⟨SwShedulableResource⟩⟩, ⟨⟨SwMutualExclusionResource⟩⟩ クラスの public 操作呼び出し

  呼び出し先クラス名. 操作名 () → 呼び出し元クラス名_呼び出し先クラス名!操作名;

- ⟨⟨SwShedulableResource⟩⟩, ⟨⟨SwMutualExclusionResource⟩⟩ クラスの private 操作呼び出し

  クラス名. 操作名 () → _クラス名操作名:

一部の操作をラベルに変換するのは, Aspect-PROMELA の weaver が inline 関数に対応していないため. ただ, weave 後に sed に定型のルールを与えて自動でラベルから inline 関数に変換することができる.

5. 適 用 例

図 1 を例題とし, テンプレートを適用した.

ウィーブ前の PROMELA の一部を示す.

```
 active proctype A()
{
 WAIT:{
  A_state=WAIT;

_acquireServices _Printer:
_getRsc1_Printer:
   skip;
  goto GotPrn;
 }

...

active proctype B()
{
 WAIT:{
  B_state=WAIT;

_acquireServices _Printer:
_getRsc2_Printer:
```

```
 skip;
goto GotPrn;
 }

...
```

5.1 特定の取得方法を変更する

以下の変更の場合を考える.

- Printer に操作 getBlcRsc() を追加し，ステレオタイプ ⟨⟨SwSchedulableResource⟩⟩ のクラスが状態 WAIT のときのみ，操作 getRsc1() ではなく操作 getBlcRsc() を呼ぶように変更する.

アスペクトクラス GetBlock を適用する．pointcut を pointcut テンプレートの (1) と (3) を適用する．図１２，１３に示す.

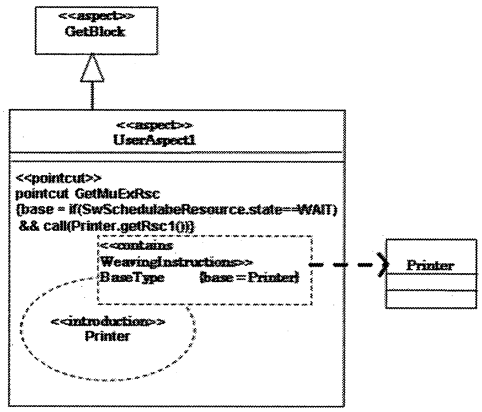

図 12 適用例 1

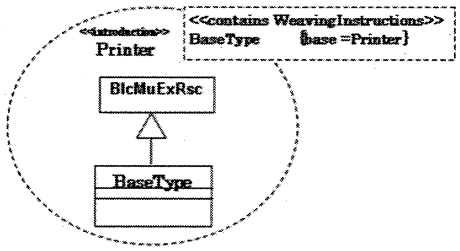

図 13 introduction

図１２，１３より以下の AspectPROMELA が生成される.

```
around : allStmnt(labe("WAIT"))
          && label("_getRsc1_Printer")
  {# _getBlcRsc_Printer: #}
```

ウィーブ後の PROMELA を示す.

```
 active proctype A()
{
 WAIT:{
  A_state=WAIT;

_acquireServices _Printer:
_getBlcRsc_Printer:
  goto GotPrn;
 }

 ...

active proctype B()
{
 WAIT:{
  B_state=WAIT;

_acquireServices _Printer:
_getRsc2_Printer:
  skip;
goto GotPrn;
 }

 ...
```

クラス A の"_getRsc1_Printer:skip;"が"_getBlcRsc_Printer:"に変更された．クラス B は Printer の取得に操作 gerRsc2() を呼び出しているので，操作 gerRsc2() の呼び出しは変更されない.

5.2 全ての取得方法を変更する

以下の変更の場合を考える.

- Printer に操作 getBlcRsc() を追加し，ステレオタイプ ⟨⟨SwSchedulableResource⟩⟩ のクラスが状態 WAIT のときのみ，全ての aquireServices の操作呼び出しを操作 getBlcRsc() の呼び出しに変更する.

アスペクトクラス GetBlock を適用する．pointcut を pointcut テンプレートの (1) と (3) を適用する．図１４に示す．また，introduction は図１３と同様.

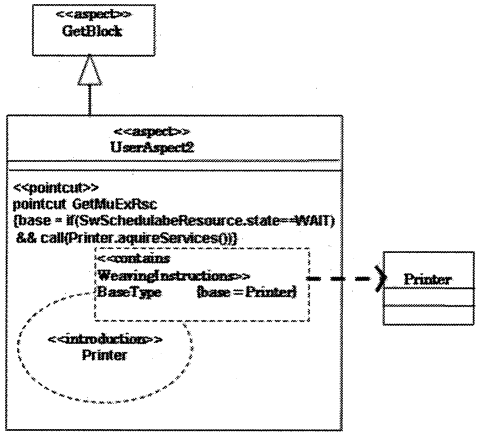

図 14 適用例 2

図１３，１４より以下の AspectPROMELA が生成される．

```
around : allStmnt(labe("WAIT"))
         && label("_aquireServices_Printer")
{# _getBlcRsc_Printer: #}
```

ウィーブ後の PROMELA を示す．

```
active proctype A()
{
WAIT:{
 A_state=WAIT;

_getBlcRsc_Printer:
 goto GotPrn;
}

...

active proctype B()
{
WAIT:{
 B_state=WAIT;

_getBlcRsc_Printer:
 skip;
goto GotPrn;
}

...
```

この場合は，すべての acquireServices の呼び出しがフック対象になるので，クラス A の"_getRsc1_Printer: skip;"，クラス B の"_getRsc2_Printer:skip;"共に変更された．

6. まとめと今後の課題，展望

本研究では，組込みドメインにフォーカスしたメカニズム切り替えの手法を提案した．適用例より，この手法を用いると手続き的にメカニズムを切り替えることが可能であることを確認した．

本研究の課題について述べる．５章で示したように，pointcut で排他的リソースを指定して切り替えることは可能だが，現在排他的リソースを指定しないと切り替えることができない．例えば，操作 getBlcRsc が呼ばれている全ての joinpoint を pointcut で記述することはできない．適用例で述べたような，特定の排他的リソース (適用例では，Printer) を取得するために操作 getBlcRsc が呼ばれている joinpoint のみ pointcut で記述することはできる．全ての排他的リソースを取得する場合でも，特定の取得方法 (例えば，操作 getBlcRsc) の呼び出しを pointcut にしたい場合も考えられるので，今後対応していきたい．

また，展望について述べる．１つ目は，テンプレートの追加である．今回は，排他的リソースの取得方法のみをテンプレート化した．同様の方法で，他のメカニズムに対しても適用し，テンプレート化を行っていきたい．２つ目は，自動化である．UML，AspectUML から PROMELA，AspectPROMELA への変換規則のみを示したが，将来的にはツール化し，自動で変更ができるようにしたいと考えている．

参 考 文 献

1) G.J,Holzmann. : The SPIN Model Checker. Addison-Wsley 2004.
2) E.Clarke,O.Grumberg,and D.Peled : Model Checking, MIT 1999.
3) OMG : A UML Profile for MARTE, Beta1,2007
4) トロン協会: μ ITRON4.0 仕様,Ver.4.02.00,2004
5) 大野真一朗，岸知二：モデル検査のためのアスペクト指向でのモデル記述支援環境，情報処理学会ソフトウェア工学研究会，SE159-6, pp41-48, 2008.
6) Stein, D., et. al. :" A UML-based Aspect-Oriented Design Notation For AspectJ ". Proceedings of the 1st International Conference on AOSD. Enschede, the Netherlands April 2002. PG106-112.