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

2007－ＳＥ－156 (3)
2007－EMB－ 5 (3)
2007／5／28

デバイス擬似コンポーネントに基づいたUML設計の検査手法の提案

原 彬寛　上田 賀一

茨城大学 工学部 情報工学科

近年，組込みソフトウェアの大規模化，複雑化が進む一方で開発期間の短縮も求められている．その結果，組込みソフトウェアの品質を保証することが困難となってきた．組込みシステムではハードウェアとそれを制御するソフトウェアが密接に関係しており，ソフトウェア設計においてもハードウェアの存在を考慮して設計を行う必要がある．本研究では，UML設計モデル中のハードウェア部分をエミュレートするデバイス擬似コンポーネントを定義し，そのコンポーネントの動作ログを用いてUML設計モデルを検査する手法を提案する．本手法をエレベータ制御ソフトウェアに適用し，動作ログからUML設計モデルの不具合を発見することが可能であることを確かめた．

A Checking Technique for UML Design Using Device Emulation Component

Akihiro Hara and Yoshikazu Ueda

Ibaraki University

It is difficult that quailty of embedded software is guaranteed, because it must be developed in limited term while it is more complex and larger scale. Embedded system consisits of hardware and software to control it, and software design considers to constraint of hardware. In this paper,we propose a checking technique for UML design using logs of components emulating hardware device in UML model. This technique applies to elevator control software, as a result we confirm that faults in UML design model are discovered from logs output components.

1 はじめに

近年，携帯電話，電化製品，自動車などにおいて，製品の差別化，高機能化のために組込まれるソフトウェアは複雑化，大規模化が年々進んでいる．一方で，製品を迅速に市場に投入するために，開発期間の短縮が求められている．このため高機能化と開発期間の短縮を実現しつつ，組込みソフトウェアの品質を保証することは困難となってきている．

組込みソフトウェア開発と一般的なビジネスアプリケーション開発の大きな違いの一つは，ハードウェアとソフトウェアの並行開発である．開発期間短縮のため，ソフトウェア設計段階においてもハーソウェアの存在を考慮しなくてはならない．

膨大なコストがかかる専用のカスタムチップの開発を伴う大規模な組込みシステム開発ではなく，安価なセンサーやヒーターを用いて外部環境に影響を与え制御する電気ポットのように，既製の汎用デバイスを用いた小規模な組込みシステム開発では，ソフトウェア設計においては使用するセンサーの種類やCPUの処理性能といった大まかなハードウェア仕様は決定していても，ハードウェアそのものは存在していないことがある．

このためソフトウェア設計段階においてはセンサーやモーターの動作といったハードウェアデバイスの実行時の動作特性を検証することができず，テスト段階になってから検証される．そしてテスト段階で設計の不具合が発見され，手戻りの発生により開発期間の遅れや設計品質の低下を招いて

いる．組込みソフトウェアの品質を保証するために，結果としてテスト段階で膨大なコストと時間がかけられてしまう．

ソフトウェア設計段階ではハードウェアの動作特性を考慮した設計を行うことにより結果的にテスト段階での負担が減り，組込みソフトウェアの品質の向上と開発期間の短縮が可能となる．

そこで本研究では，UML 設計モデルの妥当性を検査するための検査手法を提案する．ソフトウェア設計上のハードウェア部分をエミュレートし，ソフトウェア設計の動作時の特性を計測する．得られた動作ログからソフトウェア設計の妥当性を検査する．

まず既存の UML モデリングツールで出力される組込みソフトウェア向けの UML 設計モデルを対象として，ハードウェア部分のエミュレートを行うためのデバイス擬似コンポーネントを考案した．このデバイス擬似コンポーネントによってハードウェア部分を UML 設計モデルから隠蔽する．デバイス擬似コンポーネントは UML 設計モデルを検査した際のロガー機能を有する．また UML 設計モデルを動作させるために必要な時間制約に関して既存の UML の拡張を行う．既存のモデリングツールでモデリングを行う際にデバイス擬似コンポーネントをステレオタイプを用いて UML 設計モデル中のハードウェアデバイスと対応づける．そして検査時に必要となる状態の時系列情報を用意することでハードウェアの動作をエミュレートしながら UML 設計モデルを検査する．コンポーネントを呼び出した時間や各オブジェクトの状態のログを出力し，この出力されたログをもとに UML 設計モデルの妥当性を検査する．

本研究の第 2 章では提案手法を具体的に述べ，第 3 章では本手法を倒立ロボットの設計モデルとエレベータの設計モデルに適用する場合の流れと考察を述べる．第 4 章において本手法について考察する．第 5 章では関連研究を述べる．

2 提案手法

2.1 提案手法の概要

本手法では UML 設計モデルの妥当性の検査を目的としてデバイス擬似コンポーネントを提案し，デバイス擬似コンポーネントと対応させた UML 設計モデルの動作ログを取得する．提案手法の概要図を図 1 に示す．また提案手法の大まかな流れは以下の通りである．

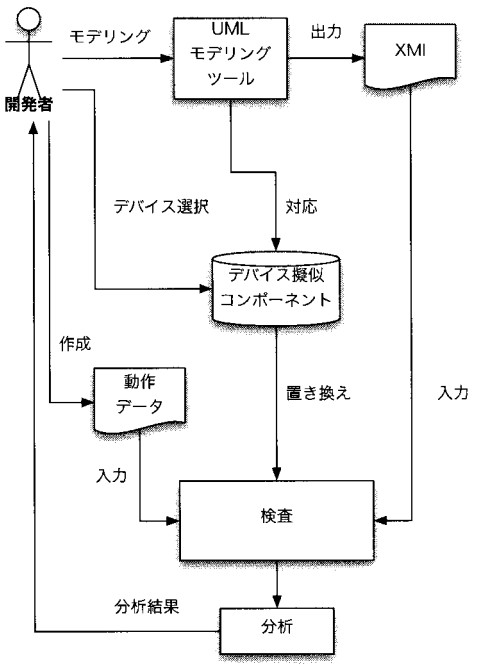

図 1: 本手法の流れ

1. 開発者は設計中のハードウェアデバイスに応じてデバイス疑似コンポーネントを選択する
2. 既存のモデリングツールでデバイス擬似コンポーネントと対応づけてモデリングする
3. 検査時に必要となる動作情報を設定する
4. デバイス疑似コンポーネントを用いて UML 設計モデルを検査し，動作ログを出力する
5. UML モデルの動作ログから設計の妥当性の分析する

まずデバイス仕様が記述されたデバイス疑似コンポーネントを選択する．デバイス擬似コンポーネントには一般的なハードウェア部分の仕様の記述がされているだけではなく，コンポーネントを用いた検査時に必要な仕様も記述されており，UML 設計モデルに存在するセンサーやモーターといっ

たハードウェアデバイスの隠蔽を目的としてコンポーネントを選択する．

また既存のモデリングツールを用いてモデリングを行う．このとき設定した擬似コンポーネントとモデルのオブジェクトを対応させてモデリングする必要がある．擬似コンポーネントはステレオタイプを用いてモデル中で一意に対応づけられる．また UML 設計モデルは検査が可能になるように本手法で定義した時間制約を付加する．

次に UML 設計モデルを検査する際に必要となる擬似コンポーネントに入力される時系列のデータを設定する．例えば開始してからいつスイッチが押されたか，ある時点でヒーターの状態はどんな状態にあるかといった検査時にコンポーネントが持つ必要があるデータを開発者が設定する．

そして設定した擬似コンポーネントでオブジェトを置き換えて UML モデルを検査し、各オブジェクトの状態推移を記録した動作ログから UML モデルの状態遷移の不具合を分析する．

2.2 デバイス擬似コンポーネント

本手法では，デバイス擬似コンポーネントを提案する．

デバイス擬似コンポーネントは組込みソフトウェアの UML 設計モデルにおいてハードウェア部分を隠蔽し，実機の存在無しに UML 設計モデルの検査を行うことを可能にする．

またデバイス擬似コンポーネントは検査時のロガー機能を持っており，各デバイス擬似コンポーネントの状態遷移を動作ログとして出力できる．動作ログを分析して発見した不具合を設計の時点で修正することでソフトウェア設計の品質を高める．組込みシステムにおけるソフトウェア設計中において必要となるデバイスはほぼ決まっており，その仕様を満たすデバイス擬似コンポーネントを選択する．デバイス擬似コンポーネント自体を開発者が作成する必要は無い．

デバイス擬似コンポーネントは一意に決められた名前を持っており，その名前をクラスに付加することで UML モデル設計中のデバイスと対応づける．デバイス擬似コンポーネントの名前が記述されたステレオタイプ

```
<<name_component>>
```

をクラスに付加することで参照を実現する．

図 2 ではエレベータの制御ソフトウェアのクラス図にステレオタイプを付加している．

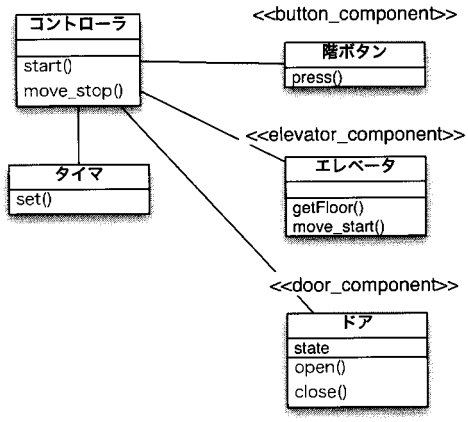

図 2: ステレオタイプが付加されたクラス図

UML 設計の検査を行うために選択されたデバイス擬似コンポーネントにはハードウェアデバイスの仕様や検査を行うための仕様が記述されている．デバイス擬似コンポーネント自体は XML 形式で記述されている．図 2 のクラス図におけるデバイス擬似コンポーネントはエレベータ，ドア，ボタンである．これらのデバイス擬似コンポーネントの記述例を図 3 に示す．また，記述例における XML タグの要素について表 1 に示す．

検査時にデバイス擬似コンポーネントの仕様を参照することでログの出力を行う．

表 1: XML タグの要素

タグ	説明
component	擬似コンポーネントの識別名
method	擬似コンポーネントのメソッド
time	メソッドの処理時間
input	動作情報の読み込み先の指定
logging	ログの出力先を指定
interrupt	割り込みメソッド

```
<?xml version="1.0" encoding="euc-jp"?>
  <component = "elevator_component">
    <method ="getFloor">
      <time ="msec">100</time>
      <input ="data.csv">elevator_floor</input>
    </method>
    <method="move_start">
      <time ="msec">4000</time>
      <loging ="log.csv">elevator</loging>
    </method>
  </component>

  <component = "door_component">
    <method="close">
      <time ="msec">2000</time>
      <loging ="log.csv">Door_state</loging>
    </method>
    <method="open">
      <time ="msec">2000</time>
      <loging ="log.csv">Door_state</loging>
    </method>
  </component>

  <component = "button_component">
    <method="start">
      <inturrupt>
        <input ="data.csv">button</input>
      <interrupt>
    </method>
  </component>
```

図 3: デバイス擬似コンポーネントの記述例

2.3 デバイス擬似コンポーネントの作成

デバイス擬似コンポーネント自体を開発者が作成することはないが，以下の流れでデバイス擬似コンポーネントを作成する．

- ハードウェアデバイスをシステム仕様書から抽出
- 必要となるメソッドの決定
- 決定したメソッドにデバイス擬似コンポーネントの仕様を付加

一度作成した擬似コンポーネントは保存しておき，擬似コンポーネントと対応するハードウェアデバイスの仕様が変更された場合，擬似コンポーネントにも変更を反映する．そのため頻繁に仕様が変更されるハードウェアデバイスが存在する場合は，擬似コンポーネントの抽象度を高く設定しておき，ハードウェアデバイスの仕様変更に対応する．

2.4 UML 設計モデルの検査時の情報

検査時情報の設定は開発者自身が設定する．検査時情報は開発者が選択したデバイス擬似コンポーネントの各状態を時系列で csv 形式で記述する．

例えばエレベータを制御するモデルにおいて以下の検査時の情報を設定する．

- エレベータが 6 階に止まっている
- エレベータの中から 6 階を指定する

図 2 ではデバイス擬似コンポーネントはエレベータ，ドア，ボタンなので，検査時情報はこれらの状態を時系列で並べた csv 形式となる．表 2 に検査時情報を示す．デバイス擬似コンポーネントの各状態を時系列で並べた検査情報をもとに UML 設計モデルの動作ログを出力する．

表 2: 検査時情報

msec	ボタン	現在のフロア
0000	-	6
0010	-	6
0020	-	6
0030	-	6
0040	-	6
0050	-	6
0060	-	6
0070	6	6

2.5 UML の拡張

本手法を用いて UML 設計モデルを検査するためには既存モデリングツールでモデリングする際にシーケンス図中のメソッドに時間制約を付加する必要がある．

付加する制約を表 3 に，制約を付加したシーケンス図を図 5 に示す．

これらの制約をもとに，メソッドの呼び出し順から動作ログを出力していく．

表 3: 付加する制約

項目	内容	使用例
時間制約	処理を完了するまでの時間	{constraint :50msec}
繰り返し	メソッドが繰り返し発生する場合の時間間隔	{loop :50msec}
割り込み	ハードウェア割り込みにより発生するメソッド	{interrupt}

2.6 UML 設計モデルの検査

デバイス擬似コンポーネントと検査情報を参照しながらシーケンス図中のオブジェクトの状態をログに残していく.

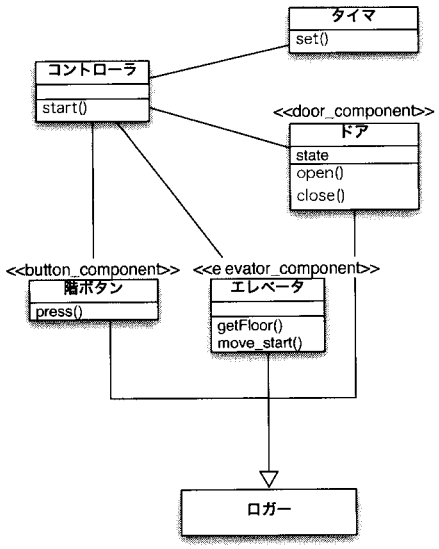

図 4: 置き換え後のクラス図

検査時には対象の UML 設計モデルのクラス図を擬似コンポーネント図で置き換える. 図 4 はエレベータの制御ソフトウェアのクラス図 (図 2) をデバイス擬似コンポーネントで置き換えたものである. 置き換え後, デバイス擬似コンポーネントの持つロガー機能を用いてログを出力する.

動作ログの形式は検査情報の形式と同様に csv 形式となっており, デバイス擬似コンポーネントの各状態を時系列で並べたものである. メソッドの呼び出しに応じて状態遷移を記録していく.

2.7 動作ログによる UML 設計モデルの分析

得られた動作ログから開発者は UML 設計モデルの分析を行う. 組込みソフトウェアは様々なドメインがあり, そのドメインで最適化するためにあるドメインで正しい処理であっても別のドメインでは異常となることがある. そのため, 本手法で得られた動作ログの分析は開発者自身が行う.

3 本手法の適用例

3.1 適用例

本手法をエレベータ制御ソフトウェアに適用する場合の流れを述べる. エレベータ制御ソフトウェアのクラス図を図 2 に, シーケンス図を図 5 に, ステートチャート図を図 6 に示す. エレベータ制御ソフトウェアから見たのハードウェア仕様は以下の通りである.

- エレベータボックス内のドア
- 行き先フロア指定ボタン

検査情報を図 2 に示す. エレベータ制御ソフトウェアを検査して得られた動作ログを図 7 に示す.

3.2 適用結果

エレベータの制御ソフトウェアを検査した動作ログから, 以下の不具合が発見された.

- エレベータが移動状態に遷移している

  実行ログの 2.180 秒において, エレベータは移動状態に遷移した (図 7 の (a) 参照). 実行時データではエレベータは 6 階に停止しており, 6 階を行き先としてボタンを押してい

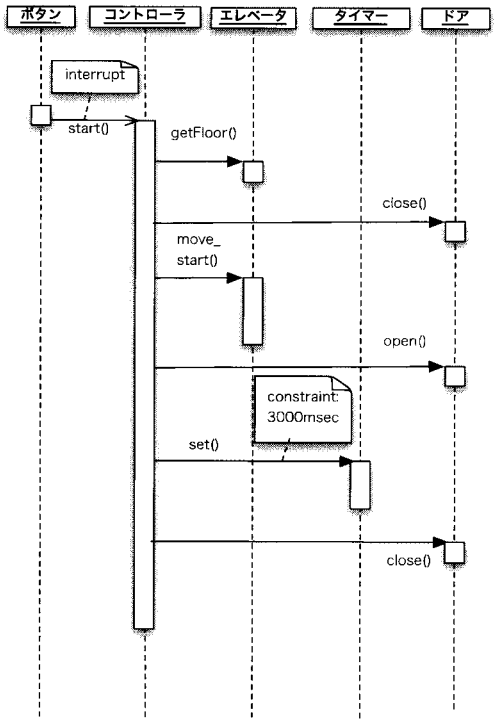

図 5: エレベータ制御のシーケンス図

る．本来ならば 6 階に停止している時にボタンを押しても，エレベータは移動状態に遷移しないことが望まれる．

- ドアが開いてしまう

  実行ログの 4.180 秒においてドアは開き，その後 3 秒間開いている（図 7 の (b) 参照）．これも本来ならば 6 階に停止ている時にボタンを押してもドアは開かないことが望まれる．

これらはエレベータ制御ソフトウェアのソフトウェア設計における不具合である．不具合が発生した原因は適用例のシーケンス図において停止している階から停止している階以外に移動することを前提としたことである． 実行したログから，エレベータ制御ソフトウェアの設計上の不具合を発見することが可能であった．

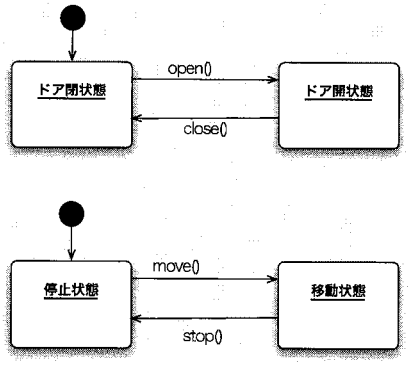

図 6: エレベータ制御のステートチャート図

4 考察

本節では適用例から得られた課題について述べる．

- 検査情報の設定方法

  検査情報の設定は開発者に委ねられている．しかし動作情報の設定によって発見できたりできなかったりすることが生じる．そのために開発者にとって容易にかつ有用な検査情報の設定方法に課題が残る．

- デバイス擬似コンポーネントの仕様決定

  センサーといったハードウェアの仕様をどの程度までデバイス擬似コンポーネントの仕様として取り入れるかが問題となる．UML 設計モデルから必要とするハードウェアの仕様は実際のハードウェアのすべての仕様が必要になるわけではない．デバイスごとにデバイス擬似コンポーネントにどの仕様を取り入れるかを検討する必要がある．

- デバイス擬似コンポーネント自体の作成の困難

  設計者は用意されたコンポーネントを使うことによって検査を行うが，動作情報の設定や擬似コンポーネント次第で UML 設計モデルの不具合を動作ログから発見できない場合ができてしまう．そのため，検査時の動作情報の作成方法も含むデバイス擬似コンポーネントの作成を的確かつ容易に作成できる手法が必要である．

```
time:msec  door  elevator_state
0000       -     s
0010       -     s
0020       -     s
0030       -     s
0040       -     s
0050       -     s
0060       -     s  //start
0070       -     s  //getFloor
0080       -     s
...        -     s
0160       -     s
0170       c     s  //close
0180       c     s
...        c     s
2160       c     s
2170       c     s
2180       c     m  //move_start  (a)
2190       c     m
...        c     m
4170       c     m
4180       o     s  //open      (b)
4190       o     s
...        o     s
5170       o     s
5180       o     s  //set
8180       c     s  //close
8190       c     s
...        c     s
1180       c     s
```

図 7: 動作ログ

5 関連研究

この節では本研究の関連研究として eUML, MATLAB/Simulink, co-design を紹介する.

5.1 eUML

eUML では, 組込みシステム開発のための分析, 設計についてのベスト・プラクティスをまとめており, 組込みステム開発のガイドラインとして示している [3].

eUML では既存の UML をそのまま使用している. 組込みシステム開発向けに必要な拡張を行っていない. そのため eUML だけで組込みシステムのモデリングが十分行える訳ではない.

しかし eUML が示すガイドラインは組込みシステム開発に十分有用であり, 本研究と組み合わせることによって組込みシステム開発の品質保証に寄与することが可能である.

5.2 MATLAB/Simulink

MathWorks 社は MATLAB/Simulink[4] を用いたモデルベースデザインを提案している. モデルベースデザインの基本は包括的なシステムレベルの数学的なモデルとなっており, Simulink 環境により, ブロック線図を利用してシステムおよびサブシステムのモデルを可視化する.

MATLAB/Simulink で行うシミュレートとはシステム自体のシミュレートであり, 本研究ではソフトウェア設計から見たハードウェアの動作特性をシミュレートする点で異なる.

本研究ではモデリング言語として UML を用いており, 様々なツールと連携, 再利用することが可能である. また, アルゴリズム的な処理を記述することが可能であるため数学的なモデルで記述できない処理を記述することが可能である.

5.3 co-design

co-design[4] では, 性能とコストのバランスをとりながら, ハードウェア設計とソフトウェア設計を同時協調的に行い, システム設計の最適化を行う. co-design におけるソフトウェアとハードウェアは同格であり, 専用 LSI を必要とする高機能, 大規模な組込みシステム開発に向いている.

しかし本研究ではソフトウェア設計からハードウェアの仕様, 特性を扱うという視点に立っており, 専用 LSI を必要とする大規模な開発ではなく, コスト制約が非常に厳しく, 安価なハードウェアを用いてソフトウェアによる制御が必要となる小規模な組込みシステムにおいては本手法は有効である.

6 おわりに

本研究ではデバイス擬似コンポーネントに基づいたソフトウェア設計の動的検証を行う手法を提案した.

そして本手法をエレベータ制御ソフトウェアに適用し, その結果からハードウェアデバイスの動作特性を含めたソフトウェア設計の妥当性を検査することが可能な動作ログを得た.

しかし組込みソフトウェア向けのUML設計モデルにおける様々な不具合を検査するためには，膨大な動作ログが必要となる．現在は膨大なログを人手で解析し、不具合を発見しなければならない．

そこで組込みシステムの対象ごとに異なった解析を行うツールで開発し，開発者に不具合をフィードバックしていく必要がある．また擬似コンポーネントの作成方法において開発者に有用なコンポーネントを用意することも必要であり，引き続き検討を行う．本手法を用いることで組込みソフトウェア設計の品質向上に寄与することが可能であることを確認した．

参考文献

[1] UML,http://www.omg.org/uml

[2] 渡辺博之，渡辺政彦，堀松和人，渡守武和記：組み込みUML，翔泳社（2002）．

[3] 本田晋也，冨山宏之，高田広章:システムレベル設計環境：SystemBuilder，電子情報通信学会論文誌，Vol.J88-D-I No.2 pp.163-174（2005）．

[4] MathWorks,MATLAB,
http://www.mathworks.com/