情報処理学会論文誌　Vol. 49　No. 10　3493–3507 (Oct. 2008)

# ソフトウェア設計に対するモデル検査のための検証パターン

金井勇人[†1]　岸　知二[†1]

本稿ではモデル検査技術を用いたソフトウェア設計検証のための検証パターンを提案する．ソフトウェア設計検証においては，時相論理式等が必要だが，その記述はソフトウェア技術者にとって扱いにくい作業である．本研究では，ソフトウェアの構造によって確認したい典型的な性質があり，またその確認方法にはいくつかの定石があることに注目し，ソフトウェア構造と性質とを合わせて体系づけたパターンを提案する．また，このパターンの例題を用いた評価も合わせて報告する．

# Verification Patterns for Model Checking Software Design

Hayato Kanai[†1] and Tomoji Kishi[†1]

In this paper, we propose verification patterns for software design verification utilizing model checking techniques. In verifying software design model, we have to develop target model and define properties depending on the target model. As, typical software structures have their own listing of important properties, it is useful to define each verification pattern as a set of software structure, its important properties, and verification techniques. We introduce verification patterns based on the idea, and demonstrate their usefulness based on a case study.

## 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あげられる．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．たとえば我々は高信頼性組み込み用オブジェクト指向設計技術プロジェクト[4]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[3]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[8]．しかしながら，モデル検査技術ではソフトウェアが満たしてほしい性質，もしくは満たしてはならない性質を確認するために，性質を時間的な概念を持たせた論理式（以下，時相論理式と呼ぶ）で記述することが必要であり，これは通常のソフトウェア技術者にとっては一般に困難な作業である．我々は，ソフトウェアのそれぞれの構造によって確認したい典型的な性質があり，またその確認方法にはいくつかの定石があることに注目し，それをパターンとして提示することで設計検証の支援を行う研究を進めている．

本稿は，以前我々が提案した UML 設計モデルを対象にした構造付き検証パターン[11),13)]を改良した内容である．改良した点は，適用対象を示す検証パターン中の構造の表記方法である．ソフトウェア設計の検証パターンを利用する際に，検証を行いたい設計モデルに適応できる検証パターンをパターンカタログから探す必要がある．ここで設計モデルの構造は静的構造と動的構造から構成されるが，検証パターン中に示される構造は，特定の具体的な 1 つの構造を表すのではなく，その検証手法が適用できる構造の持つ制約を示すものでなければならない．たとえば動的構造の表記を例にとると，我々は従来，**図 1** に示すような通常のステート図を用いてパターン構造を記述していた．

検証対象がこれと同一のステート図を持たなくても，このステート図が示している特定の条件を満たすならば検証パターンを適用できるのであるが，この表記ではそれが適切に表現

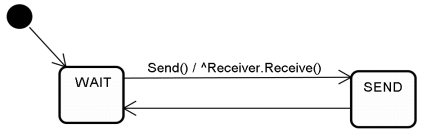


**図 1**　従来のパターン構造表記
Fig. 1　Old notation of our pattern structures.

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

されていない．本稿では検証パターンが適用できる構造の制約を示すための表記法を提案し，それを用いて検証パターンを定義した．

## 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．なお，本稿はモデル検査ツールとしてSPIN[1),2)]を利用しており，以下それをふまえて説明する．

### 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 時相論理式 LTL

時相論理式とは，通常の論理式の構成要素である原始命題，論理積，論理和，論理否定の組合せに時間的な概念を持った時相演算子を加えたものである．SPIN では時相論理式の 1 つである LTL を性質の記述に利用する．LTL で記述する時相演算子と意味を以下に示す．ここで p，q は任意の式である．

- U
  p U q . . . q が真になるまで，ずっと p は真である．
- []
  [] p . . . ずっと p は真である．
- <>
  <>p . . . いつか p は真である．
- ->
  p ->q . . . p が真ならば，q は真である．

下記は LTL 式の記述例であり，「つねに，p が真ならばいつか q は真である」ということを意味している．

```
[](p -> <>q)
```

#### 2.1.3 ソフトウェア設計に対する典型的な検証手順

設計モデルとして UML を利用した場合の典型的な検証手順例を示す．

(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[6),7),10)]．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) 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. 構造付き検証パターンの提案

### 3.1 従来の代表的な検証パターン

こうした検証を支援する 1 つの方法として，ソフトウェア設計においてデザインパターン[9)]を提示するように，ソフトウェア検証でもパターンを提示することが考えられる．現在までに提唱されている代表的な検証パターンとして，Dwyer らの検証パターン[5)]がある．この検証パターンはよく使われる時相論理式の記述を性質ごとに分類しパターン化している．分類を**図 2** に示す．

この検証パターンでは SPIN で利用する LTL 式以外の時相論理式に対してもパターン化

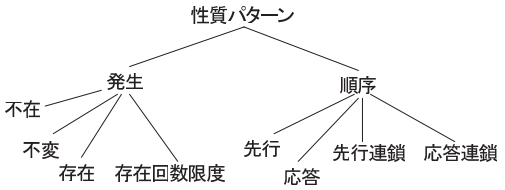


**図 2**　Dwyer らが提唱している検証パターンの分類
Fig. 2　Category of verification patterns proposed by Dwyer, et al.

を行っている．下記はパターン化された LTL 式の一例である．

- 不在 ... P はずっと偽である．
  [](!P)
- 先行 ... P が真になる前に S が真になる．
  ([]!P) || (!P U S)

### 3.2　従来の検証パターンの特徴と問題点

Dwyer らの検証パターンは対象システムを特定していないので，汎用的に利用できるという特徴を持つ．問題点として，汎用的なので特定のソフトウェア構造上の具体的な性質の表現にどう利用するかが明確に示されていない点があげられる．また，この検証パターンは時相論理式のみのパターン化なので，対象システムのモデル化方法等には言及していない．

### 3.3　構造付き検証パターンのねらい

我々はソフトウェア設計の検証を対象とした構造付き検証パターンを検討している[11),13)]．本検証パターンの特徴はソフトウェア設計の特定の構造に対して，その構造において重要な性質をリストアップし，それぞれの性質を検証するためのパターン化を行っている点である．このように構造と検証方法を合わせてパターン化することにより，実際のソフトウェア構造に関わる具体的な性質を検証するための検証方法のパターンを提示することができ，ソフトウェア技術者にとって利用しやすいパターンとなることが期待される．

本検証パターンは，以下の 2 つから構成される．

(1)　設計モデルに多出する構造を抽象化した構造の記述
(2)　上記の構造で確認が必要となる典型的な性質の検証方法の記述

(1) と (2) をセットとしたものを検証パターンカタログとしてまとめた．検証パターンカタログの具体的な内容については後述する．

本検証パターンを使った典型的な検証手順は以下のようになる（**図 3**）．

(1)　設計モデルを作成する．
(2)　検証パターンカタログより対応する構造の検証パターンを見つける．
(3)　対応した検証パターンの中で対応する性質を見つける．
(4)　設計モデルを仕様記述言語へ変換する．
(5)　検証パターンカタログの時相論理式等を設計モデルに依存した形で特殊化し，検証を行う．

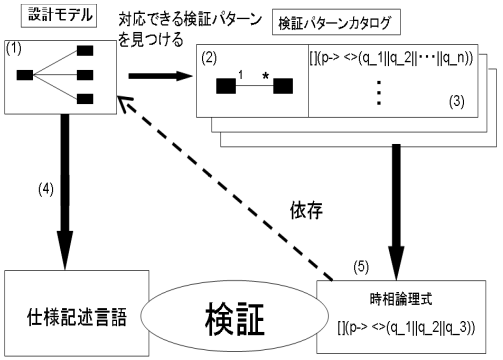


**図 3**　本検証パターンを使った検証方法
Fig. 3　Verification procedure on our verification patterns.

## 4.　パターン構造の表記方法

本章では，従来の方法からの改良点である検証パターンにおける構造記述の方法を述べる．構造記述は特定のソフトウェア構造を示すものではなく，その検証パターンを適用できるソフトウェア構造が満たすべき制約を示すものでなければならないと考える．以下，4.1 節ではどのような制約を記述できなければならないかを示し，4.2 節ではその記述方法を提案する．また，クラス図，ステートマシン図の各要素の説明には，通常の利用方法で記述されている要素に関しては説明を省略する．

### 4.1　制　　約

静的構造，動的構造ごとに制約すべき項目について以下に示す．

- 静的構造
  - 必要なクラスと，そのクラスが持つべき属性と操作
  - 任意のクラス（構造中に存在してもよいクラス）
  - クラス間の関連と多重度
- 動的構造
  - 必要な状態と遷移（ガード，イベント，アクションも含む）
  - 任意の状態，遷移と必要な状態と遷移の関係

### 4.2　表　　記

上述した制約を示すために，以下の UML 表記法を用いてパターン構造を記述する．静的

構造は，クラス図を用いる．動的構造は，メタモデルを用いる．

表記する項目とステレオタイプ等の意味について説明する．

#### 4.2.1 静的構造

(1) クラス

– 必須クラス

この検証パターンを利用するために必要なクラスを必須クラスと呼ぶ．**図 4** に表記を示す．

ステレオタイプは OMG による UML Profile for Schedulability Performance and Time Specification（SPT profile）[12] を利用する．

本稿で利用するステレオタイプは，SPT profile 中で定義されているもののうち**表 1** に示すもののいずれかである．

典型的には，これらのステレオタイプはそのクラスが以下を表す場合に使われる．

- ⟨⟨CRconcurren⟩⟩
  タスク，active クラス
- ⟨⟨SAschedRes⟩⟩
  タスクや active クラスで，セマフォや Mutex を持つもの
- ⟨⟨SAResource⟩⟩
  passive クラスで，セマフォや Mutex を持つもの

– 任意クラス

必須クラス以外のクラスを任意クラスと呼ぶことにする．必須クラスの役割を行うクラスとその他の役割のクラスの関係を示すために記述する．任意クラスは，“Arbitrary_Class_⟨ 番号 ⟩” というクラス名をつける．ここで ⟨ 番号 ⟩ 部分は任意クラスを識別するための番号とする．**図 5** に表記を示す．

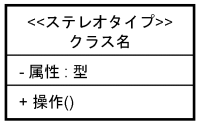


**図 4**　必須クラスの表記
Fig. 4　Notation of required class.

**表 1**　クラスのステレオタイプ記述と意味
Table 1　Notation and semantics of class stereotype.

| ステレオタイプ | 意味 |
|---|---|
| ⟨⟨CRconcurrent⟩⟩ | 並行動作単位. |
| ⟨⟨SAschedRes⟩⟩ | 並行動作単位だが，排他機構を持つ. |
| ⟨⟨SAResource⟩⟩ | 排他機構を持つ. |

(2) 関連

関連の表記を**図 6** に示す．

関連のステレオタイプは通信方法を表す．本稿で利用するステレオタイプは，**表 2** に示すもののいずれかである．

なお，同期通信か非同期通信かによって，検証性質に影響がある．たとえば，メッセージの到達性の検証の場合，非同期通信では「送信後いつかは受信」という性質になるが，同期通信では，「送信と受信は同時に起きる」という性質となる．

送信クラスから受信クラスへ矢印の関連とする．両端のクラスとも送受信する場合，矢印の表記はしない．

以上の表記方法を利用して記述した静的構造の例を**図 7**，**図 8** に示す．図 7 は，CRConcurrent である Sender と Receiver という 2 つの必須クラスがあり，その間には任意個の CRConcurrent である Arbitrary クラスが存在しうるという構造上の制約を示している．

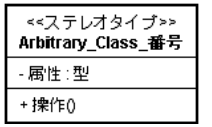


**図 5**　任意クラスの表記
Fig. 5　Notation of arbitrary class.

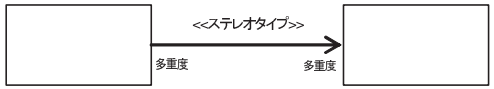


**図 6**　関連の表記
Fig. 6　Notation of relation.

**表 2**　通信方法のステレオタイプ記述と意味
Table 2　Notation and semantics of communication stereotype.

| 通信方法 | 意味 |
|---|---|
| ⟨⟨SYNC⟩⟩ | 同期通信 |
| ⟨⟨ASYNC⟩⟩ | 非同期通信 |

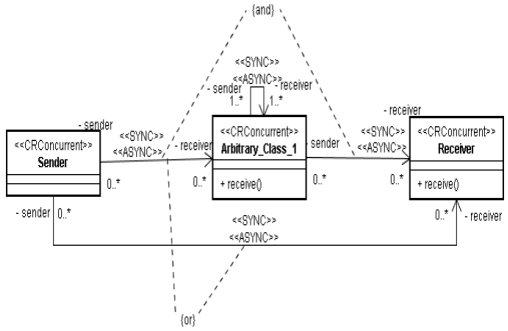


**図 7**　静的構造の表記例 1
Fig. 7　Description example 1 of static structure.

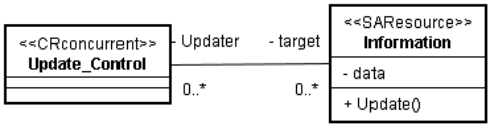


**図 8**　静的構造の表記例 2
Fig. 8　Description example 2 of static structure.

Sender，Arbitrary，Receiver 間のやりとりは，非同期通信または同期通信で行われる．多重度により各クラスのオブジェクトが任意個であることを示している．また制約 or により，Sender と Arbitrary，Sender と Receiver のどちらかの関連がなくてはならないことを示している．制約 and は Sender と Arbitrary，Arbitrary と Receiver のどちらかの関連がある場合，どちらも関連がなければならないことを示している．

図 8 は，Update_Control と Information という 2 つの必須クラスがあり，その間には任意のクラスが存在してはならないという構造上の制約を示している．Update_Control と Information 間のやりとりは，通信方法に依存しない．

#### 4.2.2　動 的 構 造

UML2.0 のステートマシン図のメタモデルを用いて，各要素の関係を示す．

要素名の後ろに記述してある「from ···」はその要素が UML2.0 で定義されている箇所を示している．

(1) 状態

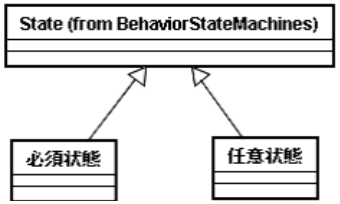


**図 9**　状態の表記
Fig. 9　Notation of state.

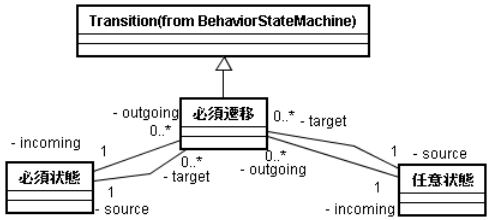


**図 10**　遷移の表記
Fig. 10　Notation of transition.

– 必須状態

この検証パターンを利用するために必要な状態を必須状態と呼ぶ．必須状態には，その状態を示す状態名を記述する．

– 任意状態

必須状態以外の状態を任意状態と呼ぶ．必須状態との関係を示すために記述される．“Arbitrary_State_⟨ 番号 ⟩” という状態名をつける．ここで ⟨ 番号 ⟩ 部分は任意状態を識別するための番号とする．

必須状態，任意状態の表記法を**図 9** に示す．

(2) 遷移

– 必須遷移

この検証パターンを利用するために必要な遷移を必須遷移と呼ぶ．必須状態には，その状態を示す適した状態名を記述する．**図 10** に表記法を示す．

– 必須イベント

この検証パターンを利用するために必要なイベントを必須イベントと呼ぶ．その遷移が発

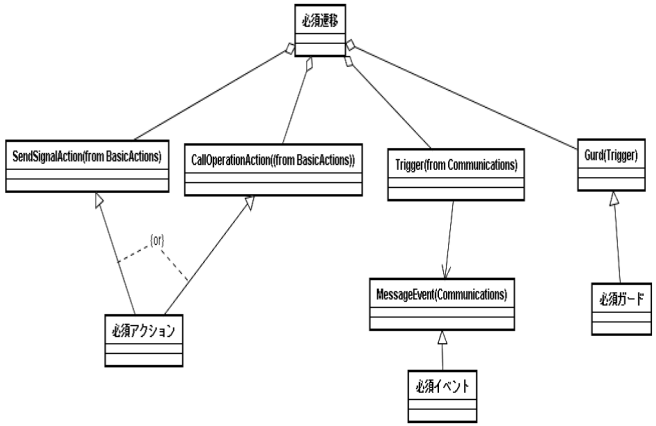


**図 11**　イベント，ガード，アクションの表記
Fig. 11　Notation of event, guard and action.

火する要因となる．そのイベントに対応づけられる操作名を “操作 ()” と表記する．

– 必須ガード

この検証パターンを利用するために必要なガードを必須ガードと呼ぶ．その遷移の発火の条件を与える．表記は条件式で表す．

– 必須アクション

この検証パターンを利用するために必要なアクションを必須アクションと呼ぶ．本検証パターンでは，他のオブジェクトに対して，メッセージを送るアクションのみを記述する．“クラス名．操作名 ()” と表記する．必須イベント，必須ガード，必須アクションの表記法を**図 11** に示す．

上記の表記方法を利用した動的構造の記述例を**図 12** に示す．

図 12 は以下の構造上の制約を表現している．

- 必須状態
  – Sending
- 任意状態
  – Arbitrary_State1
- 必須遷移
  – Sending-Arbitrary_State_1


**図 12**　動的構造の表記例
Fig. 12　Description example of behavior structure.

  – Sending_Sending
  – Transition_inside_Sending
- 必須イベント
  – なし
- 必須ガード
  – なし
- 必須アクション
  – ^ Receiver.receive()

また，関連に対する制約について説明する．

- or
  どれか 1 つの関連が成立していればよい．図 12 の例では，必須遷移である Sending-Arbitrary_State_1，Sending_Sending，Transition_inside_Sending のうちどれか 1 つが ^ Receiver.receive() を持っていればよい．

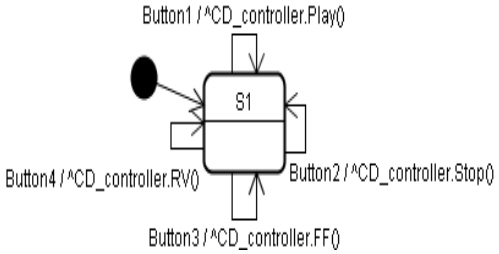


**図 13**　適用できる例 1
Fig. 13　Applicable example 1.

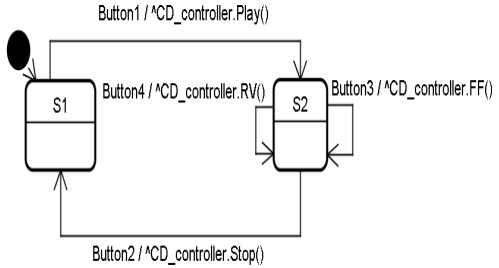


**図 14**　適用できる例 2
Fig. 14　Applicable example 2.

- mapped
  関連しているクラスのインスタンスが同じであればよい．図 12 の例では，必須遷移 Sending_Sending の両端のインスタンスが必須状態 Sending の同じインスタンスでなければならないことを示している．これはつまり自己遷移でなければならないことを示している．

図 12 は特定の動的構造ではなく，動的構造が満たすべき制約を示すものである．以下，この制約に適合するステート図と適合しないステート図をいくつか例示する．

**図 13** は，任意状態がない場合であり，制約をすべて満たしている例である．

**図 14** は，任意状態がある場合であり，制約をすべて満たしている例である．

**図 15** は，ステート図の中で少なくとも 1 つの遷移のアクションは SendSignalAction である条件を満たしていない．したがって，制約を満たしていない．

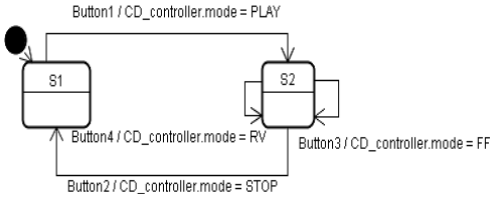


**図 15**　適用できない例
Fig. 15　Non applicable example.

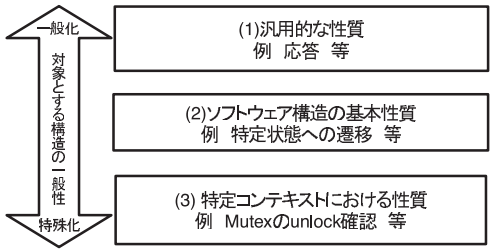


**図 16**　本研究で定義する性質体系
Fig. 16　Category of property defined by this paper.

## 5. 検証性質の分析

### 5.1 ソフトウェア設計全般における検証性質の分類

本研究では，検証性質を，それが対象とするソフトウェア構造に注目し，構造の一般性の観点から，**図 16** に示すように 3 つに分類する．

ここで，(1) がより一般的であり，(2)，(3) となるにつれ，性質が特殊化される．図 16 での各分類についてそれぞれ説明する．

(1) 汎用的な性質
　検証対象に関しての制約を持たない一般的な性質．3 章で紹介した Dwyer らによる検証パターンが対象にしている性質はこの分類にあたる．

(2) ソフトウェア構造の基本性質
　ソフトウェア設計上での基本的な構造に関する性質．たとえば，メッセージの送受信や状態遷移等に関する性質はこの分類にあたる．

(3) 特定コンテキストにおける性質

ソフトウェア設計上の特定の問題解決のための構造に関する性質．たとえば，Douglass による Real-Time Design Patterns（RTDP）[14] でパターン化されている特定構造等に関わる性質はこの分類にあたる．

(2) の性質は，(3) より汎用的な構造に関わる性質である．したがって，(2) の性質を検証する手法は (3) の性質を検証するための土台になると考えられる．本研究では，(2) の性質をパターン化した．

### 5.2　構造付き検証パターンで扱う性質

一般にモデル検査で検証することのできる UML 上の性質は，動的な側面に関する性質である．たとえば，Kruten による 4＋1 ビューモデルにおけるプロセスビューで記述されるような実行時構造に関わる性質である．具体的な例でいえば，複数のプロセス間におけるメッセージ通信に関する性質等がそれに相当する．

## 6. 検証パターンカタログ

5 章で述べた (2) に関する性質に注目した検証パターンを集め，検証パターンカタログという形で整理した．

### 6.1　検証パターンの記述項目

以下に検証パターンに記述される内容について説明する．

- 名前
  その検証パターンが対象とするソフトウェアの構造が簡潔に連想できる名前を示す．
- 検証目的
  検証で確認する性質群を示す．
- 構造
  - 静的構造
    UML のクラス図で示す．
  - 動的構造
    メタモデルを使って，UML のステートマシン図の遷移，状態等を示す．
- 協調動作例
  それぞれの構成要素が責任分担を遂行するためにどのように協調するか，シーケンス図で例をあげる．
- 検証方法
  本検証パターンで定義している検証性質と検証方法を示す．

### 6.2　構造付き検証パターンの一覧

実際のソフトウェア構造を検討し，そこで多出する構造についてパターン化した．具体的には，企業より提供されたカーオーディオ・システムを分析して，そこで多出する構造や，必要な検証項目に注目してパターン化した．

(1)　2 オブジェクト間のメッセージ送受信（**図 17**）
　Sender クラスから Receiver クラスへのメッセージの到達性を検証するパターン．

(2)　オブジェクト間連続メッセージ送受信（**図 18**）

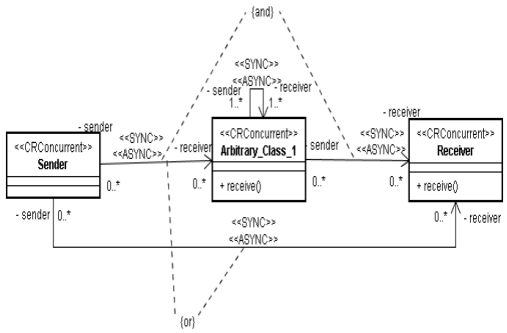


**図 17**　Sender クラスから Receiver クラスへのメッセージの到達性を検証するパターンの静的構造の表記例
Fig. 17　Description example of static structure in the verification pattern for reachability of message from Sender class to Receiver class.

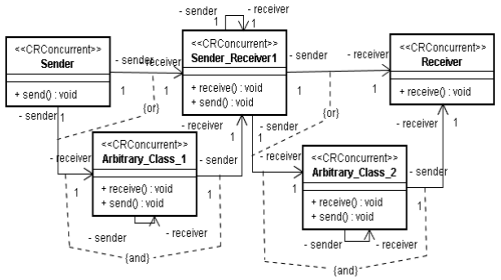


**図 18**　Sender クラスから Sender_Receiver クラス，Receiver クラスへのメッセージ到達性の順番を検証するパターンの静的構造の表記例
Fig. 18　Description example of static structure in the verification pattern for order of reachability of message from Sender class to Sender_Receiver class and Receiver class.

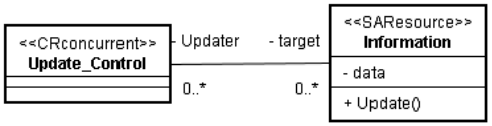


**図 19**　Informaion クラスの属性 data の値を検証する静的構造の表記例
Fig. 19　Description example of static structure in the verification pattern for value of attribute data in Information class.

Sender クラスから Sender_Receiver クラス，Receiver クラスへのメッセージ到達性の順番を検証するパターン．「2 オブジェクト間のメッセージ送受信」は，Sender クラスから Receiver クラスまでの間に介在するクラス群は検証対象ではないが，このパターンではそのクラス群も検証対象となる．

(3)　2 オブジェクト間の属性値（**図 19**）
Informaion クラスの属性 data の値を検証する．たとえば，ある特定の値になること，特定の値にはならないこと等を検証する．

## 7. 評　　価

構造付き検証パターンの妥当性について評価を行った．各評価方法の説明と結果について述べる．

**ソフトウェア構造の基本性質のパターンの評価**

RTDP は，組み込みソフトウェアでよく使われる構造をパターン化したものであり，組み込みソフトウェアで実際によく使われるソフトウェア構造を一定の網羅性をもってカタログ化したものである．そこで本評価では，この RTDP の全例題に対して，検証パターンがどれだけ適合するのか以下のように調査を行った．

- 静的構造の適合度
RTDP の各パターンのクラス図が，本パターンの静的構造のいずれかと適合するかどうかを調査した．なお，前述したように本検証パターンはプロセスビューに関わるものであるため，論理ビューを表す Subsystem and Component Architecture Patterns，配置ビューを表す Safety and Reliability Patterns は調査対象外とした．
**表 3** に，2 オブジェクト間のメッセージ送受信パターンへの適合するパターンの数を示す．なお，RTDP のパターンは Concurrency，Memory 等 4 つのパターングループに分類されており，ここではそのパターングループごとに適合する数を示した．また，

**表 3**　2 オブジェクト間のメッセージ送受信
Table 3　Message passing between two objects.

| パターングループ | 対象数 | 適合数 |
|---|---|---|
| Concurrency | 7 | 3 |
| Memory | 6 | 0 |
| Resource | 6 | 0 |
| Distribution | 6 | 6 |

**表 4**　オブジェクト連続メッセージ送受信
Table 4　Sequential message passing between objects.

| パターングループ | 対象数 | 適合数 |
|---|---|---|
| Concurrency | 7 | 1 |
| Memory | 6 | 0 |
| Resource | 6 | 0 |
| Distribution | 6 | 1 |

**表 5**　2 オブジェクト間の属性値
Table 5　Value of attribute between two objects.

| パターングループ | 対象数 | 適合数 |
|---|---|---|
| Concurrency | 7 | 1 |
| Memory | 6 | 0 |
| Resource | 6 | 6 |
| Distribution | 6 | 5 |

**表 4** に，オブジェクト連続メッセージ送受信に適合するパターンの数を示す．さらに，**表 5** に，2 オブジェクト間の属性値パターンの結果を示す．

- 動的構造の適合度
上記の静的構造の調査で適合した RTDP パターンを対象に，その RTDP パターンの例題に記載してあるシーケンス図から，ステートマシン図を作成し，そのステートマシン図が各構造付き検証パターンの動的構造と適合するかどうか調査を行った．その結果を 2 オブジェクト間のメッセージ送受信パターンは**表 6** に，オブジェクト連続メッセージ送受信パターンは**表 7** に，2 オブジェクト間の属性値パターンは**表 8** にそれぞれ示す．なお以下，パターングループである Concurrency，Memory，Resource，Distribution をそれぞれ，C，M，R，D と，パターングループを PG と略記する．
- 検証性質の適合度
検証性質の適合度について調査した．本評価では，検証パターン中に示されている LTL

**表 6**　2 オブジェクト間のメッセージ送受信
Table 6　Message passing between two objects.

| PG | パターン | 対象数 | 適用数 |
|---|---|---|---|
| C | MessageQueuing | 2 | 0 |
| C | GurddedCall | 2 | 0 |
| C | Rendezvous | 2 | 0 |
| D | SharedMemory | 2 | 0 |
| D | RemoteMethodCall | 2 | 2 |
| D | Observer | 4 | 4 |
| D | DataBus | 2 | 2 |
| D | Proxy | 10 | 10 |
| D | Broker | 16 | 16 |

**表 7**　オブジェクト連続メッセージ送受信
Table 7　Sequential message passing between objects.

| PG | パターン | 対象数 | 適用数 |
|---|---|---|---|
| C | MessageQueuing | 3 | 0 |
| D | SharedMemory | 3 | 0 |

**表 8**　2 オブジェクト間の属性値
Table 8　Value of attribute between two objects.

| PG | パターン | 対象数 | 適用数 |
|---|---|---|---|
| C | GurddedCall | 2 | 2 |
| R | CriticalSection | 2 | 2 |
| R | ProioirityInheritance | 2 | 2 |
| R | HighestLocker | 2 | 2 |
| R | PriorityCeiling | 2 | 2 |
| R | SimultaneousLocking | 2 | 2 |
| R | OrderedLocking | 2 | 2 |
| D | SharedMemory | 2 | 2 |
| D | Observer | 2 | 2 |
| D | DataBus | 6 | 6 |
| D | Proxy | 2 | 2 |
| D | Broker | 10 | 10 |

の述語を規則に沿って，特殊化するだけで適合できた場合を，本検証パターンに適合できると判断し，特殊化するだけでは適合できない場合を，本検証パターンに適用できないと判断した．以下に適合できる場合の例として Observer Pattern の例題を**図 20** に示す．

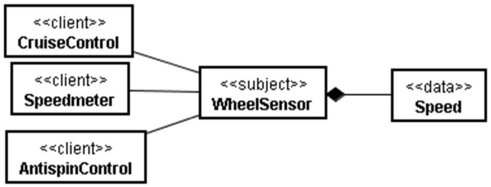


**図 20**　例題
Fig. 20　Example.

上記の例で以下の性質を検証する場合を考える．subject オブジェクトが複数の client オブジェクトにメッセージ accept を送信したら，複数の client オブジェクトがメッセージ accept を受信する．
上記の性質に対して「2 オブジェクト間のメッセージ送受信パターン」の「4. いつか特定した複数のメッセージの受信が行われる.」を適用した例を示す．
–適用したパターン

述語を以下のように定義する．

特定の Sender オブジェクト 1 が特定の
メッセージ 1 を送信した Sending 状態
...Sending1

特定の Sender オブジェクト 2 が特定の
メッセージ 2 を送信した Sending 状態
...Sending2

…

特定の Send オブジェクト N が特定の
メッセージ N を送信した Sending 状態
...SendingN

```
特定の Receiver オブジェクト 1 が特定の
メッセージ 1 を受信した Receiving 状態
...Receiving1

特定の Receiver オブジェクト 2 が特定の
メッセージ 2 を受信した Receiving 状態
...Receiving2

    ...

特定の Receiver オブジェクト N が特定の
メッセージ N を受信した Receiving 状態
...ReceivingN


([]((<>Sending1 &&  <>Sending2 && ...
     && <>SendingN)
    -> <>Receiving1 && <>Receiving2 &&
        ... && <>ReceivingN)))
```

–適用例

```
#define Sending1 SentAccept

#define
    Receiving1 client_ReceivedAccept[0]
#define
    Receiving2 client_ReceivedAccept[1]
#define
    Receiving3 client_ReceivedAccept[2]

[]((<>Sending1) ->
  <>Receiving1
  && <>Receiving2 && <>Receiving3)
```

**表 9**　2 オブジェクト間のメッセージ送受信
Table 9　Message passing between two objects.

| パターングループ | 対象数 | 適用数 |
|---|---|---|
| Concurrency | 3 | 3 |
| Distribution | 6 | 6 |

**表 10**　オブジェクト連続メッセージ送受信
Table 10　Sequential message passing between objects.

| パターングループ | 対象数 | 適用数 |
|---|---|---|
| Concurrency | 1 | 1 |
| Distribution | 1 | 1 |

**表 11**　2 オブジェクト間の属性値
Table 11　Value of attribute between two objects.

| パターングループ | 対象数 | 適用数 |
|---|---|---|
| Concurrency | 1 | 1 |
| Resource | 6 | 6 |
| Distribution | 5 | 5 |

上記のように LTL 式は received を「&&」で追加するだけで，LTL 式を作成することができる．適応できない具体例は，時相論理演算子を追加する必要がある場合，パターン定義されていない意味の述語を追加する必要がある場合である．
上記の静的，動的構造の調査で適応したパターンに対して，その例題の性質と適合するかどうか調査を行った．
**表 9** に，2 オブジェクト間のメッセージ送受信パターンの結果を示す．また，**表 10** に，オブジェクト連続メッセージ送受信パターンの結果を示す．さらに，**表 11** に，2 オブジェクト間の属性値パターンの結果を示す．

## 8. 考　　察

### 8.1 適合の網羅性

RTDP は，3 つの粒度に分類し，パターン化されている．たとえば，Memmory Patterns

は細粒度である詳細設計に該当するが，提案する検証パターンはこの粒度の例題には適用できなかった．一方，中程度の粒度であるメカニズム設計に該当する Concurrency Patterns, Resource Patterns や，大粒度であるアーキテクチャ設計に該当する Distribution Patterns に含まれる例題の多くに対してはパターンが適用できることが分かった．

上記より，本検証パターンは中粒度，大粒度のソフトウェア構造に対して有効性があると考えられる．なお，2 オブジェクト間のメッセージ送受信構造パターン，2 オブジェクト間の属性値検証パターン，多くのパターンの例題に対応できた．しかし，複数のオブジェクト間のメッセージ送受信構造パターンは，多くのパターンの例題に対応することができなかった．これは，対象にしたパターンの例題の中に，3 つ以上の関連したメッセージ送受信構造が少なかったためだと考えられる．

### 8.2　検証の正確さ

本検証パターンを利用した検証について，正確さの観点から考察する．

7 章で評価したように，組み込みソフトウェアでよく使われる構造のうち，中粒度・大粒度のものに対しては構造や性質がが一定の適合性を持っていることが分かった．これらに対しては，パターン中の性質を基本的にはパターン中の性質をそのまま利用して検証ができることから，検証を専門としないソフトウェア技術者がアドホックに性質を記述する場合に比べてより正確な検証が行えると考えられる．

また，本稿で提案した構造記述により，特定の構造が検証パターンの想定する構造に適合するかどうかの判断が，従来の提案に比べてより適確に行えるようになり，パターンのより適切な適用につながると考える．なお，今回はステート図のみメタモデルで定義した．これはステート図はクラス図に比べインスタンス構造のバリエーションを表現しにくかったからである．しかしながら利用しやすさの観点からは，静的構造と動的構造の記述レベルがそろっていることが望ましい．両者の記述レベルをそろえることは今後の課題である．

## 9. おわりに

本稿では，我々が検討してきた構造付き検証パターンに関し，特に構造記述の新しい方法を提案するとともに，検証パターンの有効性に関する評価を行った．今後，このパターンを改良し今回は対応できなかった詳細設計のような細かい粒度に適合できるようにすることを考えている．そして，この成果をベースに，さらに対象を (3) の性質まで拡張することも検討している．前述したように，(3) のパターンは (2) のパターンを組み合わせることによって定義できると推測しており，その体系だった組合せ方法を提案することによって，(3) の性質を検証するためのパターンを提示したいと考えている．

また，本検証パターンは検証する際の補助になるだけでなく，設計においても活用することが可能であると考えている．本検証パターンで示している構造は，検証を考慮した構造であるため，この構造をベースとして設計することにより，検証容易な設計を導くことができると考えており，検証を行いながら設計を進める検証指向の設計法へとつながると期待している．

## 参考文献


1) http://spinroot.com/spin
2) Holzmann, G.J.: *The SPIN Model Checker*, Addison-Wsley (2004).
3) Clarke, E., Grumberg, O. and Peled, D.: *Model Checking*, MIT (1999).
4) 岸　知二，青木利晃，中島　震，野田夏子，片山卓也：プロジェクト紹介：高信頼組込み用オブジェクト指向設計技術，情報処理学会ソフトウェア工学研究会，SE146-7, pp.41–46 (2004).
5) Dwyer, M.B., Avrunin, G.S. and James, C.: Patterns in Property Specifications for Finite-state Verification, *The 21st International Conference on Software Engineering* (May 1999).
6) Schafer, T., et al.: Model Checking UML State Machines and Collaborations, *Workshop on Software Model Checking* (2001).
7) Lilius, J. and Paltor, I.P.: vUML: A Tool for Verifying UML Models, TUCS Technical Report, No.272 (1999).
8) Kishi, T., et al.: Project Report: High Reliable Object-Oriented Embedded Software Design, *The 2nd IEEE Workshop on Software Technology for Embeddedand Ubiquitous Conputing Systems* (*WSTFEUS'04*) (2004).
9) Gamma, E., Helm, R., Johnson, R. and Vlissides, J.: *Design Patterns Elements of Reusable Object-Oriented Software*, ADDISON-WESLEY (1995).
10) Schafer, T., Knapp, A. and Merz, S.: Model Checking UML State Machines and Collaborations, *Electronic Notes in Theoretical Computer Science*, Vol.55, No.3 (2001).
11) 金井勇人，岸　知二：UML 設計モデル検査技術のための検証パターンの提案，情報処理学会ソフトウェア工学研究会，SE152-3, pp.17–24 (2006).
12) OMG: UML Profile for Schedulability, Performance and Time Specification version1.1.
13) 金井勇人，岸　知二：UML 設計に対するモデル検査のための検証パターンの提案と評価，情報処理学会ソフトウェアエンジニアリングシンポジウム，pp.185–192 (2006).
14) Powel, D.B.: *Real-Time Design Patterns*, Addison-Wsley (2003).

## 付　　録

一例として，“2 オブジェクト間のメッセージ送受信検証パターン” を以下に示す．ここでは，“性質と検証方法” は 5 つある中から 2 つのみ示す．

- 名前
  2 オブジェクト間のメッセージ送受信検証パターン
- 検証目的
  2 オブジェクト間のメッセージ送受信に関する以下のことを満たすか検証する．
  – 1 つのメッセージの特定
  – 複数のメッセージの特定
  – 1 つのメッセージの厳密な特定
  – 時間差のある複数のメッセージの特定
  – 特定メッセージの送信または受信の否定
- 構造

– 静的構造（**図 21**）
– 動的構造
・ Sender（**図 22**）
・ Receiver（**図 23**）

- 協調動作例（**図 24**）

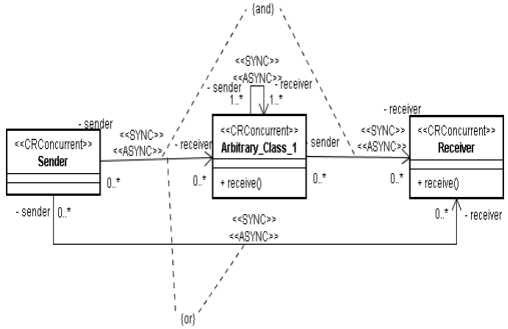


**図 21**　静的構造
Fig. 21　Static structure.

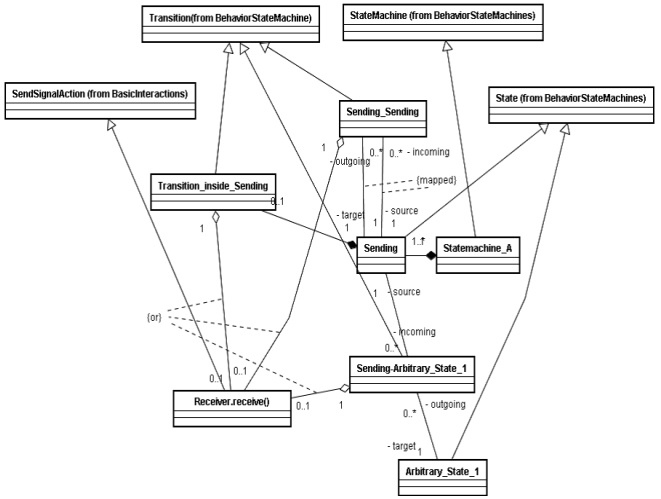


**図 22**　Sender
Fig. 22　Sender.

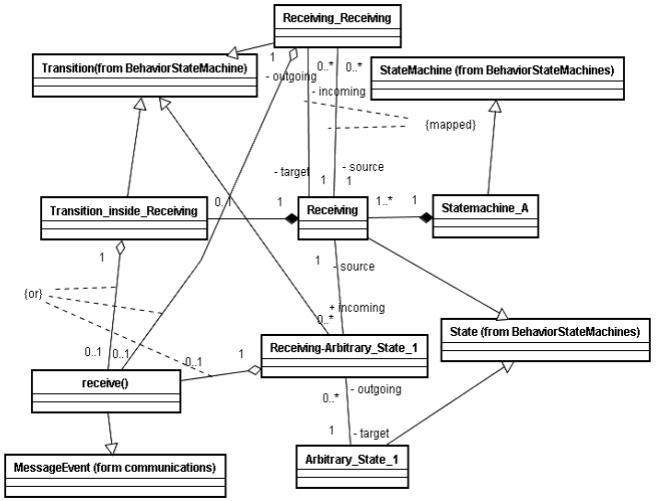


**図 23**　Receiver
Fig. 23　Receiver.

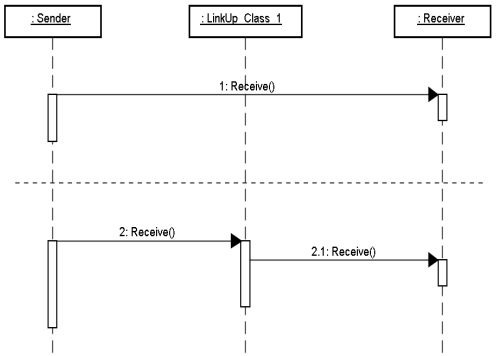


**図 24**　協調動作例
Fig. 24　Collaboration example.

- 性質と検証方法

1．1 つのメッセージの特定

– 性質

特定した 1 つのメッセージを含んだメッセージの送信と受信が行われる．

– 検証方法

以下の LTL 式で検証する．

**LTL 式**

つねに送信後はいつかは受信する

```
[](Sending -> <> Receiving)
```

なおここで述語は以下を表す．

・Sending... 特定の Send オブジェクトが特定のメッセージを送信した Sending 状態

・Receiving... 特定の Receiver オブジェクトが特定のメッセージを受信した Receiving 状態

2．複数のメッセージの特定

– 性質

特定した複数のメッセージを含んだメッセージの送信と受信が行われる．

– 検証方法

以下の LTL 式で検証する．

**LTL 式**

つねに複数のメッセージ送信後はいつかは複数のメッセージを受信する

```
[]((Sending1 || Sending2 ||
     … || SendingN)
     -> <>(Receiving1 || Receiving2 ||
     … || ReceivingN))
```

なおここで述語は以下を表す．

・Sending1... 特定の Sender オブジェクト 1 が特定のメッセージ 1 を送信した Sending 状態

・Sending2... 特定の Sender オブジェクト 2 が特定のメッセージ 2 を送信した Sending 状態

…

・SendingN... 特定の Sender オブジェクト N が特定のメッセージ N を送信した Sending 状態

・Receiving1... 特定の Receiver オブジェクト 1 が特定のメッセージ 1 を受信した Receiving 状態

・Receiving2... 特定の Receiver オブジェクト 2 が特定のメッセージ 2 を受信した Receiving 状態

…

・ReceivingN... 特定の Receiver オブジェクト N が特定のメッセージ N を受信した Receiving 状態

なお，同名のオブジェクト，同名のメッセージは必ずしも同じオブジェクト，メッセージでなくてもよい．たとえば，上記の “Sending1” の定義で利用している “メッセージ 1” と “Receiving1” の定義に利用している “メッセージ 1” は同じメッセージでなくてもよい．

(平成 20 年 1 月 10 日受付)
(平成 20 年 7 月 1 日採録)

**金井　勇人**（学生会員）

2004 年桐蔭横浜大学工学部電子情報工学科卒業．2006 年北陸先端科学技術大学院大学情報科学研究科博士前期課程修了．現在，同大学院情報科学研究科博士後期課程に在籍．ソフトウェアに対する設計手法，検証手法の研究に従事．

**岸　　知二**（正会員）

1982 年京都大学大学院工学研究科情報工学専攻修士課程修了．2002 年北陸先端科学技術大学院大学情報科学研究科博士後期課程修了．日本電気勤務を経て，2003 年より北陸先端科学技術大学院大学情報科学研究科特任教授．博士（情報科学）．組み込みソフトウェアを主対象に，ソフトウェアアーキテクチャ，ソフトウェアプロダクトライン，設計検証等の研究に従事．ACM，IEEE CS 各会員．