ソフトウェアテストシンポジウム 2005
JaSST'05: Japan Symposium Software Testing 2005

Webアプリケーションのクラス設計仕様に対するモデル化と検証(注†)

崔　銀惠† 渡邊　宏†

† 産業技術総合研究所 システム検証研究センター　〒 661–0974 尼崎市若王寺 3–11–46
E-mail: †{e.choi,hiroshi-watanabe}@aist.go.jp

あらまし　本研究では Web アプリケーションのクラス設計仕様に対するモデル化と検証法を提案する．オブジェクト指向型言語による Web アプリケーション開発でクラス設計仕様は設計仕様書群の中でも実装直前に作成し，直接コーディングに使われるものとして位置付けられる．したがって，クラス設計仕様は実装に最も近い仕様である．Web アプリケーションの設計段階で作成するクラス仕様を検証できて，実装前に不具合を検出できれば，Web アプリケーションの品質向上につながると共に後のテスト工程で見つかる不具合を改修する手戻りを削減できる．本研究ではまず，Web アプリケーションのクラス設計仕様からクラスの動的な振舞いを表すモデルを作成する方法を提案する．次に，モデル検査の手法を用いて，提案するモデルを利用したクラス設計仕様の検証方法を 2 つ提案する．第 1 の方法は，クラス仕様が Web アプリケーションの基本的な設計仕様である画面遷移仕様を満たすか否かを検査する．第 2 の方法は，クラス仕様と UML アクティビティ図などの他の動作仕様とを比較する方法である．最後に，実際の Web アプリケーションの設計仕様の事例に対してモデル化と検証法を適用した結果を報告し，提案法の有効性を示す.
キーワード　Web アプリケーション，検証，オブジェクト指向，クラス設計仕様，画面遷移，UML アクティビティ図，モデル検査.

Modeling and Verification of Class Specification for Web Applications

Eun-Hye CHOI† and Hiroshi WATANABE†

† Research Center for Verification and Semantics, AIST　Nakoji 3–11–46, Amagasaki, 661–0974 Japan
E-mail: †{e.choi,hiroshi-watanabe}@aist.go.jp

Abstract　This paper proposes a framework for modeling and verification of class specifications for Web applications. Class specification such as a class diagram is indispensable in an object-oriented design for Web applications, and thus verification of the class specification is very useful for building reliable Web applications. In this paper, we first propose a method for modeling the dynamic behavior of classes from the class specification. We next present two methods for verifications of the class specification based on model checking over the proposed model. The first verification is to check whether the class specification satisfies the requirement of page transition, which is one of the most essential specifications for Web applications. The second verification is to check the compatibility between the class specification and the other dynamic models such as an UML activity diagram. We also apply the proposed methods to real specifications of a Web application used in a certain company and show the efficiency of the proposed methods through experimental results.
Key words　Web application, verification, object-oriented design, class specification, page transition, UML activity diagram, model checking.

1. ま え が き

近年，Web アプリケーションの需要は急速に増加し，その振舞いも複雑になってきているため，Web アプリケーションの信頼性を保証する検証技術の重要性がますます高まってきている．Web アプリケーションを含めたソフトウェアの品質保証のためにはレビューやテストといったソフトウェアの検証が必要である．従来のソフトウェアの検証としては実装後のプログラムの

(注1)：文部科学省科学技術振興調整費（若手任期付支援），関西電力株式会社との共同研究の研究成果である.

コードに対しておこなうテストが多かったが，近年では実装前の設計段階でソフトウェアの設計仕様をモデル化して検証するというアプローチが注目を集め始めている．ソフトウェアの設計仕様を検証して実装前に不具合を発見することは，ソフトウェアの品質保証と共に，後のテスト工程で見つかる不具合を修正する手戻りの削減につながる．

本研究では，Web アプリケーションの設計仕様の内，クラスの設計仕様に注目する．近年のソフトウェアの設計開発ではオブジェクト指向が広く採用されてきており，オブジェクト指向言語の Web アプリケーションの設計ではクラス図と各クラスのメソッド仕様といったクラスに関する仕様の作成が必要不可欠である．また，クラスの仕様は直接コーディングに使われるので実装に最も近い仕様である．本研究では Web アプリケーションのクラス設計仕様に対するモデル化と検証法を提案する．

提案法ではまず，Web アプリケーションのクラス設計仕様からクラス全体の動的な振舞いを表すモデルを作成する．クラスの最も基本的な設計仕様であるクラス図はクラスの静的な構造を表す仕様であるため，クラスの動的な振舞いは表せない．クラスの動的な振舞いを表すモデルとしては UML [9] のステートチャート図などがあるが，クラス全体の振舞いを表すモデルではないため，そのままでは検証に利用できない．そこで，クラス図と各クラスのメソッド仕様書からクラス全体の動的な振舞いを表すモデルを作成する方法を提案する．本論文では，クラス図とメソッド仕様書をまとめてクラス仕様，クラス仕様から提案法を用いて作成するモデルをクラスモデルと呼ぶこととする．

次に，クラスモデルを利用したクラス仕様の検証法を 2 つ提案する．提案する検証法ではモデル検査 [1] を用いる．モデル検査とは，有限状態遷移系でモデル化されたシステムが時相論理で表現した性質を満たすか否かをシステムが取り得る全状態空間を探索して網羅的に検査する手法である．既存のモデル検査ツールを利用してある程度の大きさのシステムに対しては全自動的に検査できるという長所があり，近年最も注目されているシステム検証技術の 1 つである．提案する第 1 の検証手法では，クラス仕様が Web アプリケーションの基本的な設計仕様の 1 つである画面遷移仕様を満たすか否かを検査する．この検証では，画面遷移仕様が表す性質を時相論理 CTL(Computation Tree Logic) の式で表現し，クラスモデルがその CTL 式で表現した性質を満たすか否かをモデル検査する．

提案する第 2 の検証手法では，クラス仕様と別の動作仕様の整合性を検査する．例えば UML アクティビティ図のように，動作仕様は一般的に遷移図で表されることが多い．この検証では，まず，クラスモデルと比較する動作仕様のモデルとを合成したモデルを作成する．ただし，クラスモデルと動作仕様のモデルで同じ処理を表す部分が同期するように合成する．次に，合成したモデルの振舞いにデッドロックがあるか否かをモデル検査する．合成モデルにデッドロックがあればクラス仕様と比較する動作仕様の振舞いに整合しない部分があることが分かり，モデル検査の結果の反例を分析して不整合部分を特定できる．

最後に，実際の Web アプリケーションの設計仕様に対する適用実験を通して，提案法の有効性を確認する．事例の設計仕様にはクラス仕様（クラス図とメソッド仕様書），画面遷移図，UML アクティビティ図が含まれていた．適用実験では，クラス仕様からクラスモデルを作成し，クラス仕様が画面遷移図の仕様を満たすか否かの検査と，クラス仕様と UML アクティビティ図の比較検査をそれぞれ行った．今回の実験では，提案法でのモデル検査に UPPAAL [10] を用いた．実験の結果，クラス仕様が画面遷移仕様を満たさない不具合とクラス仕様と UML アクティビティ図の間の不整合がそれぞれ短時間で幾つか発見できた．

2. モデル検査

モデル検査とは，有限状態遷移系でモデル化したシステムが時相論理で表現した性質を満たすか否かをシステムが取り得る全状態空間を探索して網羅的に検査できる手法である．検査したいシステムを状態遷移系のモデルとして表現し，検査項目を時相論理式で表現すれば，既存のモデル検査ツールを利用して全自動的に検査できる．また，モデル検査の結果，検査項目が正しくない場合は反例が出力されるので不具合部分を特定するためにも有効な手法である．こうした理由から，モデル検査を用いたシステム検証は近年ますます注目を集めており，有名なモデル検査ツールとしては UPPAAL [10]，SMV [6]，SPIN [5] などがある．

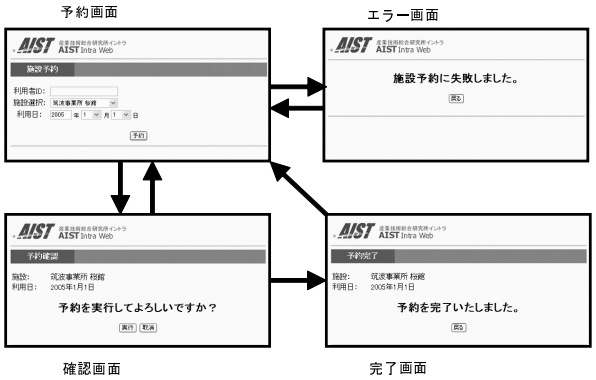
本研究の適用実験では，モデル検査ツール UPPAAL を用いる．UPPAAL はスウェーデンの Uppsala 大学とデンマークの Alborg 大学が共同で開発したツールで，システムの状態遷移モデルを GUI で描くことができるのでモデルの記述が簡単であるという特徴をもつ．UPPAAL は付録で説明するラベル付き遷移システムとチャネル同期を使った並列合成が記述できる．そのほかにも，共有変数，時間などを記述できる．また，モデルのシミュレータも備わっているのでモデル検査結果の反例をシミュレーション機能を用いて分析できる．モデル検査の検査項目としては命題時相論理 CTL の一部の論理式を対象とするが，安全性や活性，デッドロックといったほとんどの検証したい性質は記述可能である．また，UPPAAL は Java で実装されているため，SMV や SPIN と比較すると動作速度が遅いが，ある程度の大きさのシステムに対しては十分に有用である．

3. クラス仕様のモデル化

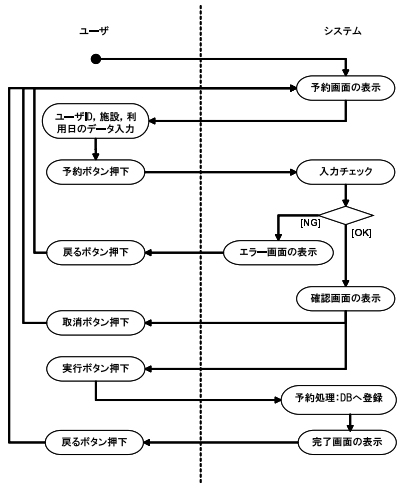
3.1 クラス仕様

本論文では，クラス図とメソッド仕様を合わせて**クラス仕様**と呼ぶ．簡単のため，クラス仕様で各クラスのインスタンスは高々 1 つしか生成されないと仮定する．また，メソッドの処理が実行中のとき，処理が終了するまで同じオブジェクトのメソッドが新たに呼び出されないことを仮定する．適用実験で対象としたようなオブジェクト指向の MVC モデル [4] に基づいて開発される一般的な Web アプリケーションのクラス仕様には上の仮定があてはまる．

［**例 1**］ 図 1 は簡単な Web アプリケーションの画面遷移仕様とその処理の流れを表す UML アクティビティ図の例である．図 2 は図 1 の仕様の Web アプリケーションを設計するためのクラス図とメソッド仕様書の例である．画面遷移仕様の予約画面，エラー画面，確認画面，完了画面に対応して，クラス図には $ReservAction$, $ErrorAction$, $ConfirmAction$, $FinishAction$ というクラスがある．各クラスは対応する画面にあるボタンを押したときの処理を行うメソッドを持つ．メソッド仕様書にはメソッドが行う処理内容が書かれている．この例では簡単のためにクラス変数は使用していない． □



(1) 画面遷移仕様



(2) UML アクティビティ図

図 1　Web 画面遷移仕様とアクティビティ図の例

3.2　クラスモデルの作成法

この章では，クラス仕様からクラス全体の動的な振舞いを表すモデルを作成する方法を説明する．本論文では，次の手順でクラス仕様から作成するモデルを略して**クラスモデル**と呼ぶ．

クラスモデルの作成手順は次の 2 つのステップから成る：(Step 1) 各クラスの振舞いを表すラベル付き遷移システムを作成する，(Step 2) クラス同士の連携を考慮して，Step 1 で作成したラベル付き遷移システムを同期合成する．(ラベル付き遷移システムと合成に関しては付録を参照されたい.)

Step 1: 個々のクラスのモデル化

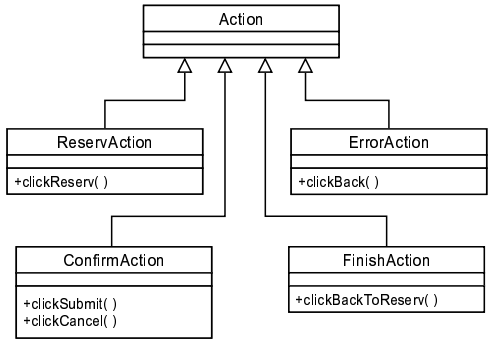
各クラスごとの振舞いをラベル付き遷移システムで記述する．n 個のメソッド $M_1, \ldots, M_n$ を持つクラスについては，次の (1) から (4) の手順で，すべての遷移にラベル τ が付いたラベル付き遷移システムを一つ作成する．

（1） 初期状態 $*$ とインスタンスが生成された状態 1 を用意する．初期状態 $*$ から状態 1 へ遷移 $* \xrightarrow{\tau} 1$ を作る．

（2） 各メソッド M_i に対して，新たにメソッドが呼び出された状態 x_i とメソッド処理が終了した状態 y_i を状態に加える．また，遷移 $1 \xrightarrow{\tau} x_i$ を作る．

（3） 各メソッド M_i に対して，メソッド仕様書をもとに，メソッドが呼び出された状態 x_i からメソッド処理が終了した状態 y_i に至るまで適宜状態を加えて状態遷移を作成する．ただし，遷移にはすべてラベル τ を付ける．

（4） 各メソッド M_i に対して，次のいずれかを行う：



(1) クラス図

クラス名	メソッド名	処理内容
ReservAction	clickReserv	1. 利用者 ID のチェックを行う. 1.1. 誤りがあればエラーページへ転送する. 2. 確認画面へ転送する.
ErrorAction	clickBack	1. 予約画面へ転送する.
ConfirmAction	clickSubmit	1. 予約処理を行う.
		2. 完了画面へ転送する.
	clickCancel	1. 予約画面へ転送する.
FinishAction	clickBack	1. 予約画面へ転送する.

(2) メソッド仕様書

図 2　クラス図とメソッド仕様書の例

- メソッドの処理が終了した後にインスタンスが不要になる場合（ページ転送処理を行うメソッドの場合など）は，遷移 $y_i \xrightarrow{\tau} *$ を作成する．
- それ以外の場合は，遷移 $y_i \xrightarrow{\tau} 1$ を作成する．

Step 2: モデルの合成

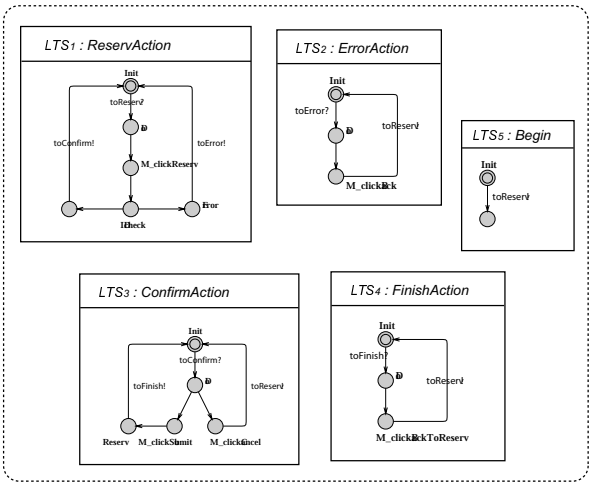
Step 1 で作成したラベル付き遷移システムは一つ一つのクラスのインスタンスの振舞いを表わすにすぎない．クラス全体の動的な振舞いは，メソッドが別のクラスを呼び出してインスタンスが生成されるなど，クラス間の連携動作で実現される．そこで，作成したラベル付き遷移システムをチャネルを使って同期させ，クラス全体の動作を模擬する遷移システムを作成する．k 個のクラス $C_1, \ldots, C_k$ から作ったラベル付き遷移システムに対して次の手順でラベルを付け換える：

（1） 各クラスに対応する入力チャネルと出力チャネルを用意する．クラス C_i についてはチャネル a_i を用意し，出力チャネルと入力チャネルのラベル $a_i!$，$a_i?$ を用意する．

（2） それぞれのクラスについて次の作業をする：クラス C_i のラベル付き遷移システムの遷移 $* \xrightarrow{\tau} 1$ のラベル τ を $a_i?$ に取り換える．

（3） クラス $C_1, \ldots, C_k$ の各メソッドについて次の作業をする：メソッドが処理中にクラス C_j を呼び出す場合，メソッド呼出し部分のラベル付き遷移 $s \xrightarrow{\tau} s'$ のラベル τ を $a_j!$ に取り換える．

上の (2)，(3) を行った後に，付録にある方法でラベル付き遷移システムをチャネルを使って並列合成すると，(3) のメソッドがクラスを呼び出す遷移と (2) のクラスが生成される遷移が同期して動作する (ラベルのない) 遷移システムが得られる．この遷移システムがクラス全体の振舞いを表現するクラスモデルである．



クラスモデル $LTS = LTS_1 \parallel LTS_2 \parallel LTS_3 \parallel LTS_4 \parallel LTS_5$

図 3　クラスモデルの例

3.3　クラスモデルの作成例

ここでは，図 2 のクラス図とメソッド仕様書を例にクラスモデルの作成方法を説明する．図 3 の LTS_1, LTS_2, LTS_3, LTS_4 は，それぞれ，図 2–(1) のクラス図の各クラス $ReservAction$, $ErrorAction$, $ConfirmAction$, $FinishAction$ をモデル化したラベル付き遷移システムを表す．ただし，LTS_5 は初期画面（予約画面）に対応するクラス $ReservAction$ を呼出すために作成したモデルである．

たとえば，クラス $ConfirmAction$ のモデル LTS_3 は次の手順で作成される：

（1） 初期状態 `Init`，インスタンスが作成された状態 `Do`，遷移 `Init` → `Do` を作成する．

（2） このクラスには 2 つのメソッド $clickSubmit$ と $clickCancel$ があるので，それぞれのメソッドが呼ばれた状態 `M_clickSubmit` と `M_clickCancel` と遷移 `Do` → `M_clickSubmit`, `Do` → `M_clickCancel` を作成する．

（3） メソッド $clickSubmit$ の仕様書を参照して,「予約処理を行う」状態 `Reserv` を作成し，`M_clickSubmit` → `Reserv` とする．予約処理を行った後メソッド処理は終了し完了画面へ転送するので，`Reserv` → `Init` とする．同様に，`M_clickCancel` → `Init` を作成する．

個々のクラスに対するモデル LTS_1, LTS_2, LTS_3, LTS_4 を作成した後は，クラス呼出しに対応する遷移にチャネルをつけて同期合成する．ここでは，各クラス $ReservAction$, $ErrorAction$, $ConfirmAction$, $FinishAction$ に対して，チャネル `toReserv`, `toError`, `toConfirm`, `toFinish` を用意する．たとえば，クラス $ConfirmAction$ のモデル LTS_3 に関しては，このクラスが呼出されることに対応する遷移にラベル `toReserv?` をつけて，`Init` $\xrightarrow{\texttt{toReserv?}}$ `Do` とする．また，他のクラス $FinishAction$, $ReservAction$ の呼出しに対応する遷移である `Reserv` → `Init`, `M_clickCancel` → `Init` のそれぞれに対してラベル `toFinish!`, `toReserv!` を付ける．

このように作成したモデルの合成 $LTS_1 \parallel LTS_2 \parallel LTS_3 \parallel LTS_4 \parallel LTS_5$ はクラス全体の動的な振舞いを表す．

4.　検証 1：画面遷移仕様の検査

ここでは，提案したクラスモデルとモデル検査法を用いて，クラス仕様が画面遷移仕様を満たすか否かを検査する方法を説明する．我々は文献 [2] で，画面遷移仕様を時相論理の CTL 式を用いて表現し，その CTL 式を検査項目としてモデル検査を行うことで画面遷移仕様を検証するという手法を提案した．次の 4.1 で画面遷移仕様のモデル検査について紹介し，4.2 でクラスモデルに適用する．

4.1　画面遷移仕様のモデル検査

画面遷移仕様のモデル検査 [2] は，画面遷移仕様と画面遷移仕様通りに振舞うべき遷移システムの両者を比較して検査する．画面遷移仕様とは画面の集合 X と画面間の遷移の集合 $R(\subseteqq X \times X)$ の対 $V = (X, R)$ で与えられるものである．また，遷移システムの各状態には画面 (X の要素) が一つ対応することを仮定する．例えば，Web アプリケーションのアクテビティ図はこのような仮定を満たすので検査対象に成る [2]．クラスモデルもこの仮定を満たす．

提案法では，画面の集合 X をモデル検査の検査式を構成する原始命題の集合と見なし，遷移システムの各状態に画面を一つ付記したクリプキ構造の上で次の二種類の CTL 式を検査項目としてモデル検査を行う．

(e1)　全ての画面遷移 $(x, y) \in R$ について，CTL 式 $EF(x \wedge EXy)$,

(e2)　各画面 $x \in X$ について，CTL 式 $AG(x \rightarrow AX(x \vee \bigvee_{(x,x') \in R} x'))$.

最初の (e1) の $x \wedge EXy$ の部分は,「画面 x を持つ状態から画面 y を持つ状態へ遷移する」を表現していて，全体の式 $EF(x \wedge EXy)$ は，初期状態からそのような状態へ到達可能であることを表現する．この検査式の検査結果が真になると，画面遷移仕様に定義されている画面遷移が遷移システムで起こることが確認できる．

次の (e2) の CTL 式は,「画面 x を持つ状態の次の状態は，画面 x か，あるいは $(x, y) \in R$ となる画面 y である」を意味している．この検査式が真になると，画面遷移仕様に定義されていない画面遷移が遷移システムで起こらないことが確認できる．

次の 4.2 章ではクラスモデルに対して UPPAAL を使って本検査と同等の検査を行う方法を説明する．

4.2　UPPAAL を用いた模擬検査

ここでは，図 2 のクラス仕様が図 1–(1) の画面遷移仕様を満たすか否かを UPPAAL を用いて検査する．クラスモデルは 3.3 章で説明した手順で作成した図 3 のクラスモデルを例として使用する．

まず，UPPAAL の共有変数を使って画面を値として持つ変数を用意し，この変数を利用してクラスモデルの各状態に画面の情報をつける．ただし，今回の例では各クラスがちょうど一つの画面と対応しているので，この共有変数は使用していない．

また，UPPAAL では (e1)，(e2) の EX, AX が記述できないので,「次の状態」の画面情報を表す変数 `next_view` を用意し，クラスモデルで画面が移る遷移が起こるとき，共有変数 `next_view` の値を更新させる．例では，予約画面，エラー画面，確認画面，完了画面を表す定数値をそれぞれ `ReservPage`, `ErrorPage`, `ConfirmPage`, `FinishPage` として，画面遷移仕様検査のためのクラスモデルは図 4 のように作成した．たとえば，クラス

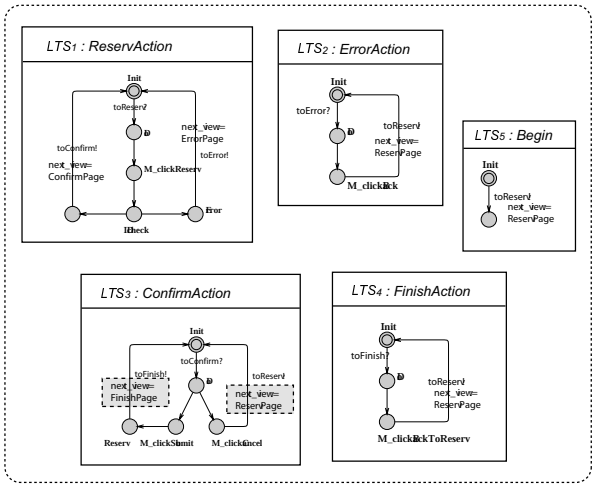


図 4　画面遷移仕様検査のための画面付きクラスモデル

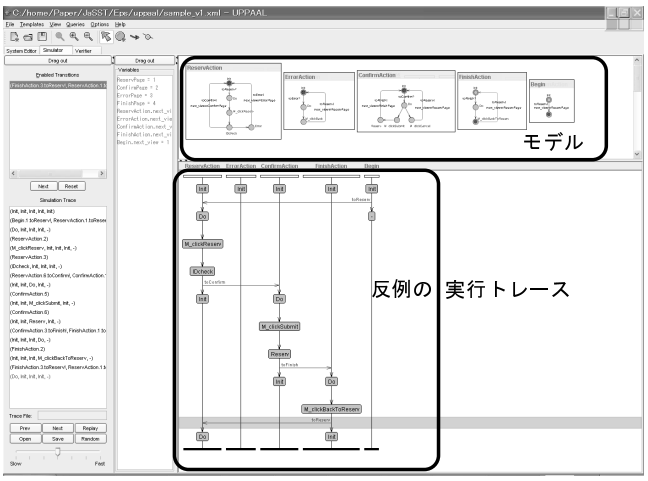


図 5　UPPAAL の出力：クラス仕様の画面遷移仕様検査

ConfirmAction のモデル LTS_3 の四角の破線で記した個所が示しているのは，それぞれ，完了画面へ転送する遷移と予約画面へ転送する遷移である．それぞれの遷移が起こるとき，共有変数の更新 `next_view := FinishPage`, `next_view := ReservPage` が行われる．

次に，図 1–(1) の画面遷移図から UPPAAL 向けの検査式を作成する．検査項目 (e1),(e2) を UPPAAL の検査式に翻訳すると，検査項目 (e1) は (1)–(6)，検査項目 (e2) は (7)–(10) になる．

（1） `E <> ReservAction.next_view == ErrorPage`
（2） `E <> ReservAction.next_view == ConfirmPage`
（3） `E <> ErrorAction.next_view == ReservPage`
（4） `E <> ConfirmAction.next_view == ReservPage`
（5） `E <> ConfirmAction.next_view == FinishPage`
（6） `E <> FinishAction.next_view == ReservPage`
（7） `A[] (ReservAction.next_view == ReservPage ∨ ReservAction.next_view == ErrorPage ∨ ReservAction.next_view == ConfirmPage)`
（8） `A[] (ErrorAction.next_view == ErrorPage ∨ ErrorAction.next_view == ReservPage)`
（9） `A[] (ConfirmAction.next_view == ConfirmPage ∨ ConfirmAction.next_view == FinishPage ∨ ConfirmAction.next_view == ReservPage)`
（10） `A[] (FinishAction.next_view == FinishPage ∨ FinishAction.next_view == ReservPage)`

UPPAAL で図 4 のモデルに関して上記の 10 個の検査項目をモデル検査した結果は全て「Property is satisfied.」となり，図 2 のクラス仕様が図 1–(1) の画面遷移仕様を満たすことがわかる．

画面遷移仕様が満たされない例も見せる．画面遷移仕様に完了画面から予約画面への遷移がないと仮定して，上記の検査項目 (10) を `A[] FinishAction.next_view == FinishPage` に変更する．UPPAAL でこの検査項目をモデル検査すると，結果は「Property is not satisfied.」と偽になり，図 5 のように反例の実行トレースがシミュレータに出力される．反例を分析すれば，画面遷移仕様にはない完了画面から予約画面への遷移がクラス仕様にあることが読みとれる．

5.　検証 2：動作仕様との比較検査

ここでは，提案したクラスモデルとモデル検査法を用いて，クラス仕様と他の動作仕様の整合性を検査する方法を説明する．一般に設計では，クラス仕様のほかに，UML のアクティビティ図，ユースケース図，シーケンス図などといった動的な振舞いを記述する仕様を併用することがある．クラス仕様とそれらの動作仕様の整合性を検査できれば，クラス仕様の不具合が発見できる．5.1 章と 5.2 章で，クラス仕様と他の動的仕様の整合性をモデル検査を用いて検査する方法と UPPAAL を用いた模擬検査例をそれぞれ説明する．

5.1　クラス仕様と動作仕様の整合性検査

動作仕様は一般的にフローチャートのような遷移図で表されることが多い．クラス仕様と動作仕様が整合していることは，動作仕様の動作の流れに対応した状態の遷移列がクラスモデル上にも存在し，クラス仕様の動作の流れに対応した状態の遷移列が動的仕様の遷移図上にも存在することを意味する．提案法では，与えられたクラス仕様から作成したモデルと動的仕様から作成したモデルを並列合成し，合成したモデルでデッドロックの有無をモデル検査する．本検査でクラス仕様と動的仕様の整合性を検査する．

ここでは，検査の対象となるクラス仕様のクラスモデル LTS と動作仕様をモデル化したラベル付き遷移システム F が与えられているとする．クラスモデルは 3.2 章にあるように，そのクラス仕様を構成する各クラスについて作成したラベル付き遷移システム $LTS_1, \ldots, LTS_n$ を並列合成して与えられるものとする．これらのラベル付き遷移システム F, およびクラスモデル $LTS_1 || \cdots || LTS_n$ それぞれはデッドロックしないことを仮定する．

次のような手順で検査を行う．

（1）　イベントを列挙する：クラス仕様の中の基本的なイベントを列挙する．とくに，ラベル付き遷移システム F の遷移図で枝わかれしている遷移に対応するイベントは含める．

（2）　ラベルを付ける：上で列挙したイベントに対しチャネルを用意し，F の対応する遷移には出力チャネルのラベルを付ける．また，クラスのラベル付き遷移システムのイベントに対応する遷移には入力チャネルを付ける．

（3）　F とクラスのラベル付き遷移システム $LTS_1, \ldots, LTS_n$ を並列合成し，デッドロック検査をする．

もしデッドロックが起こる場合，並列合成の定義から，デッド

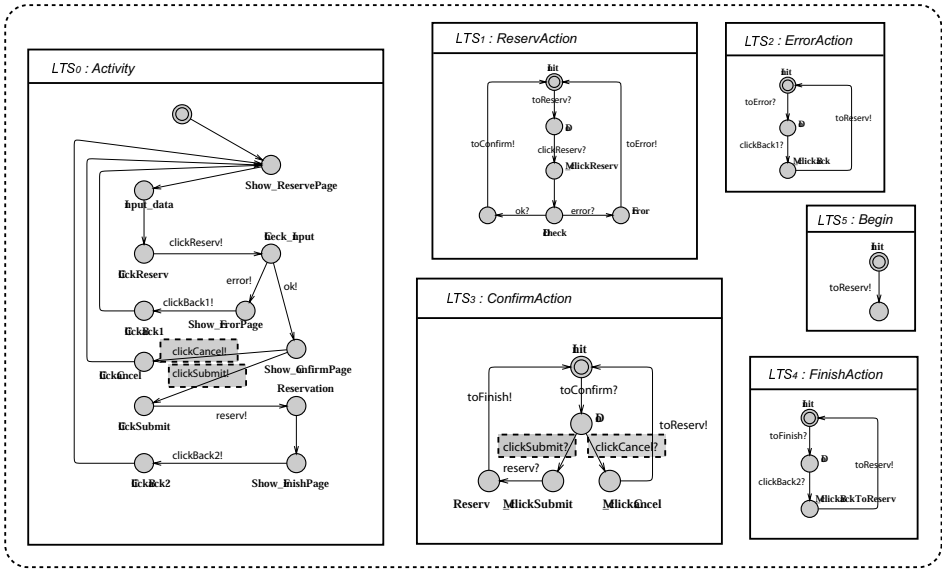


図 6　クラスモデルとアクティビティモデルの合成

ロックの状態について次が分かる：

(1) ラベル付き遷移システム F は出力チャネルのラベルが付いた遷移しか起こらない状態にある．また，どのラベル付き遷移システム $LTS_1, \ldots, LTS_n$ も入力チャネルのラベルの付いた遷移しか起こらない状態にある．(2) F で起こる遷移の出力チャネルと $LTS_1, \ldots, LTS_n$ で起こる遷移の入力チャネルは共通するチャネル名を持たない．

(1)，(2) より，デッドロックの状態では，動作仕様で起こるイベントとクラスモデルで起こるイベントが対応しないことがわかり，クラス仕様と動的仕様の不整合が検出できる．

しかし，提案法は，検査手順の中のイベントの選び方に任意性があり，もちろん完全な検査ではない．この検査によってクラス仕様と動作仕様の間の全ての不整合部分を検査できるわけではないことに注意されたい．

5.2　UPPAAL を用いた模擬検査

ここでは，図 2 のクラス仕様と図 1–(2) のアクティビティ図の例を用いて提案する整合性検査を UPPAAL を用いて行う．

まず，クラスモデルとアクティビティ図のモデルの合成モデルを作成する．(以降では，アクティビティ図のモデルを略してアクティビティモデルと呼ぶ.）クラスモデルは図 3 のように作成される．一方，アクティビティ図に対しては，図 1–(2) の遷移図をそのままラベル付き遷移システムとしてモデル化する．図 6 は作成したモデルである．図の LTS_0 はアクティビティモデルを，残りはクラスのモデルを表す．それらを合成するために，アクティビティモデルとクラスのモデルで同じ処理を表す遷移にチャネルを用いた同期ラベルを付けている．たとえば，アクティビティモデルと $ConfirmAction$ クラスのモデルには「実行ボタンを押す」と「取消ボタンを押す」という同じ処理を表す遷移がそれぞれにあり，図の破線の四角で記したように，`clickSubmit` と `clickCancel` というチャネル名のラベルを付けている．

次に，合成モデルのデッドロック検査を行う．UPPAAL で図 6 のモデルを記述して，デッドロックがないことを表す UPPAAL の検査式「$A[]$ not deadlock」をモデル検査する．この例では検査結果は「Property is satisfied.」となり，デッドロックはない．

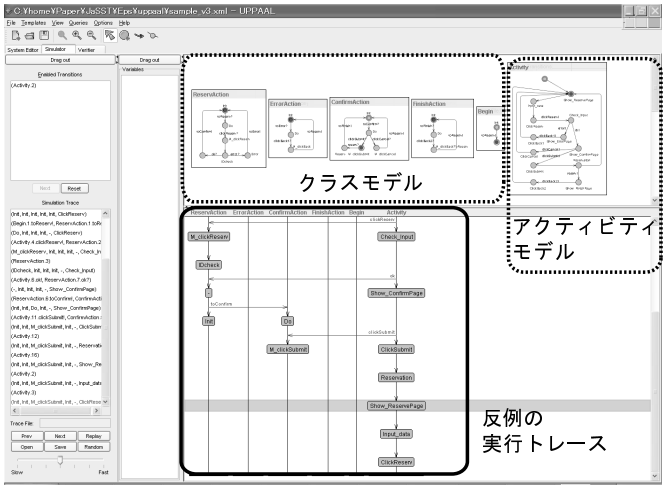


図 7　UPPAAL の出力：クラス仕様と動作仕様の比較検査

模擬的に不具合の検出例も見せる．仮に，図 1–(2) のアクティビティ図で，状態「予約処理 : DB への登録」から「予約画面の表示」へ遷移があるとしよう．アクティビティモデルに新しい遷移を追加して同様にデッドロック検査を行うと，今度は「Property is not satisfied.」という結果となり，図 7 のように反例が出力される．反例を分析すれば，アクティビティ図の「予約処理」の後の「予約画面の表示」という動作はクラス仕様のメソッド仕様書の「予約処理」の後の「完了画面の表示」と整合しないという仕様間の不具合が検出できる．

6.　適 用 実 験

提案法の有効性を評価するため，実際にある企業で稼働中の Web アプリケーションのクラス設計仕様を対象に適用実験を行った．当該 Web アプリケーションは Struts [7] のフレームワークを用いて開発された Java ベースの業務処理アプリケーションである．その設計仕様は上位と下位に分かれており，上位の設計仕様には画面遷移図とアクティビティ図が，下位の設計仕様には

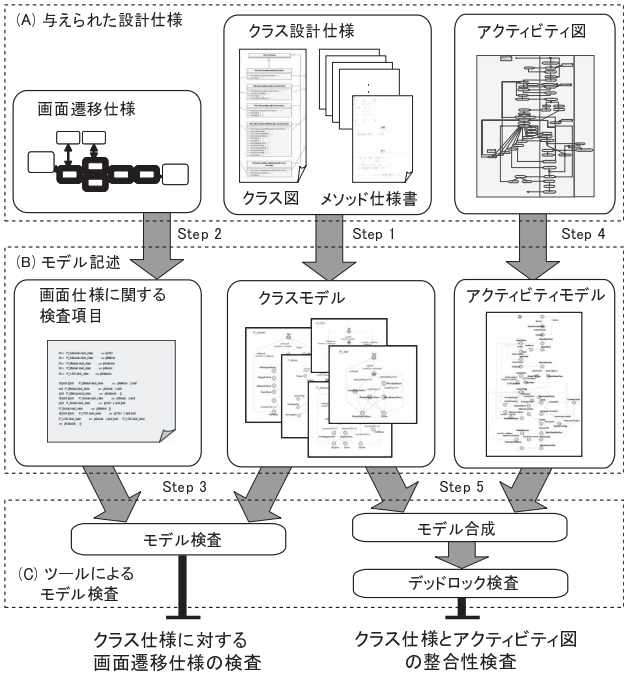


図 8　実 験 手 順

クラス図とメソッド仕様書が含まれていた．クラス図とアクティビティ図は UML で，画面遷移図は状態遷移図で書かれていた．一方，メソッド仕様書は自然言語の文章で書かれていた．それらの設計仕様を対象に，クラス仕様からクラスモデルを作成し，クラスモデルを用いた画面遷移仕様の検査とアクティビティ図との整合性検査をそれぞれ提案法を用いて行った．提案法におけるモデル検査には UPPAAL を用いた．

図 8 は実験の手順を表す．まず，図 8-(A) の設計仕様について説明する．本実験では，当該 Web アプリケーションの数十個のモジュールの内の 1 つに関する設計仕様を対象とした．表 1 は使用したクラス図，画面遷移図，アクティビティ図のそれぞれの大きさを表す．メソッド数に関しては，各クラスごとのメソッド数の組を表記している．クラス図の 5 つのクラスの内 4 つは画面遷移図の画面に 1 つずつ対応しており，残りの 1 つは初期画面の設定に関するクラスである．

実験では，最初に，クラス仕様からクラスモデルを作成し，UPPAAL を用いてモデルを記述した (図 8 の Step 1)．表 2 は個々のクラスに対して作成したラベル付き遷移システムの大きさと，それらのラベル付き遷移システムを同期合成するために使用したチャネル同期ラベルの数を表す．各クラスのモデルにおけるチャネル同期ラベルの数は，そのクラスのメソッドで他のクラスを呼出す回数 + 1（自分自身のクラスが呼出されてインスタンスが作成されることを表す遷移に対応する）と等しい．個々のクラスのモデルを作成してクラス呼出しの対応部分にチャネルをつければ，同期合成は UPPAAL が行う．なお，クラスモデルの作成と記述にはクラス仕様の作成者でない第三者一人で約 2 時間かかった．

次に，クラス仕様に対する画面遷移仕様の検査を行った．画面遷移仕様の検査では，画面遷移図から検査項目を作成し (図 8 の Step 2)，クラスモデルで検査項目が成り立つかをモデル検査する (図 8 の Step 3)．表 3 は (e1)，(e2) のそれぞれに関する検

表 1　実験対象の設計仕様

クラス図	クラス数	5
	メソッド数	(1,2,2,10,2)
画面遷移図	状態数	4
	遷移数	9
アクティビティ図	状態数	66
	遷移数	83

表 2　作成したクラスモデル

各クラスのモデル	状態数	遷移数	同期ラベル数
C_1	8	13	2
C_2	12	19	3
C_3	10	13	3
C_4	18	33	3
C_5	10	15	4

表 3　画面遷移仕様の検査結果

	(e1)	(e2)	合計
検査項目数	9	4	13
偽判定項目数	1	1	2

査項目数と検査結果が偽判定の項目数を示す．(e1) の検査項目は，画面遷移図の画面遷移がクラスモデルにあることを検査する項目で，検査項目数は画面遷移図の遷移数（=11）に等しい．一方，(e2) の検査項目はクラスモデルの画面遷移が画面遷移図上にあることを検査する項目で，検査項目数は画面遷移図の状態数（=4）に等しい．モデル検査の結果，(e1) と (e2) のそれぞれに関して 1 つの検査項目が偽と判定され，画面遷移に関する不具合部分が見つかった．なお，検査項目の作成と記述には第三者一人で約 5 分かかった．また，UPPAAL によるモデル検査に必要な実行時間は 1 秒以内であった．

最後に，クラス仕様とアクティビティ図の整合性検査を行った．この検査では，アクティビティ図のモデルを記述し (図 8 の Step 4)，クラスモデルとアクティビティ図のモデルを合成してデッドロック検査を行う (図 8 の Step 5)．実験では，UPPAAL でアクティビティ図の遷移図をそのまま記述し，アクティビティ図の遷移と各クラスのモデルの遷移の対応する部分にチャネルの同期ラベルをつけてクラスモデルと合成した．アクティビティ図とクラスモデルの間でチャネルをつけたラベル遷移数は 26 個である．UPPAAL を用いたデッドロック検査の結果は偽であり，反例を分析した結果，クラス仕様とアクティビティ図の間で幾つかの不整合部分が見つかった．不整合の詳細は述べられないが，あるボタンを押したときのクラス仕様とアクティビティ図の画面遷移先が異なっていたり，ある処理の後にクラス仕様にはエラー表示画面への遷移にあるが，アクティビティ図ではエラーへ遷移しないといった種類の不整合が見つかった．なお，アクティビティ図からのモデル記述と反例分析による不具合の特定には，第三者一人でそれぞれ約 1 時間と 30 分がかかった．また，UPPAAL によるデッドロック検査の実行時間は 1 秒以内であった．

7. 関 連 研 究

Web アプリケーションを対象とした形式的検証法はまだほとんど提案されていない．文献 [3] は，Web アプリケーションの画

面遷移関係を含む振舞いを Web オートマトンと呼ばれるオートマトンとしてモデル化し，Web オートマトンをテストに用いる手法を提案している．Web オートマトンは Web アプリケーションの MVC モデルに基づいて作成され，その振舞いからテストパターンが作成される．しかし，形式的検証は今後の課題となっている．また，文献 [8] では，Web アプリケーションの画面遷移をグラフモデルとして作成しモデル検査を行う形式的手法について述べている．しかし，文献 [8] と本研究の提案法では注目する検証対象と性質が異なる．文献 [8] では Web アプリケーションの保守性に注目して画面遷移が満たすべき好ましい性質を CTL 式で記述し検証するというフレームワークが提案されている．

8. ま と め

本研究ではオブジェクト指向型言語によって開発される Web アプリケーションのクラス設計仕様を対象としたモデル化と検証法を提案した．まず，モデル化では，クラス設計仕様であるクラス図とメソッド仕様書からクラス全体の振舞いを表すクラスモデルをラベル付き遷移システムの合成として表した．次に，提案したクラスモデルとモデル検査を用いた 2 つの検証法を提案した．1 つ目の検証法では，クラス仕様が画面遷移仕様を満たすか否かを検査する．この検査では，画面遷移仕様を時相論理式を用いて表現し，クラスモデルがその検査式を満たすか否かをモデル検査する．2 つ目の検証法では，クラス仕様と UML アクティビティ図のような他の動作仕様の整合性を検査する．この検査では，動作仕様をラベル付き遷移システムとしてモデル化し，そのモデルとクラスモデルとの合成モデルにデッドロックがないかをモデル検査する．デッドロックがあればクラス仕様と動作仕様の間で不整合部分があるため，モデル検査結果の反例を分析することで不具合部分を特定できる．最後に，提案法の有効性を評価するため，実際にある企業で稼働中の Web アプリケーションのクラス設計仕様に対して適用実験を行った．実験の結果，クラス仕様と画面遷移仕様の検査，クラス仕様とアクティビティ図の検査のそれぞれに対して提案した検査法を用いて短い時間で幾つかの不具合を発見できた．

文 献

[1] E. M. Clarke, O. Grumberg, D. Peled, “Model Checking,” MIT Press, 1999.

[2] 崔銀惠, 河本貴則, 渡邊宏, “画面遷移仕様のモデル検査,” 日本ソフトウェア科学会 コンピュータソフトウェア (採録済).

[3] Keishi Kato, Shouji Yuen, Daiju Kato, and Kiyoshi Agusa, “Web automaton: A behavioral model of Web applications based on the MVC Model,” 日本ソフトウェア科学会 ディペンダブルソフトウェア研究会 (DSW’04), pp. 111–120, 2004.

[4] I. Singh, B. Searns, M. Johnson, and the Enterprise Team, “Designing Enterprise Applications with the J2EE Platform, Second Edition,” 2002.

[5] G. J. Holzmann, “The Model checker SPIN,” *IEEE Trans. Softw. Eng.*, vol. 23, no. 5, pp. 279–295, 1997.

[6] K. L. MacMillan, “Symbolic Model Checking,” Kluwer Academic, 1993.

[7] Struts, http://struts.apache.org/.

[8] E. D. Sciascio, Francesco M. Donini and Marina Mongiello and Giacomo Piscitelli, “Web Applications Design and Maintenance Using Symbolic Model Checking,” *Proc. of the 7th European Conference on Software Maintenance and Reengineering (CSMR 2003)*, pp. 63–72, 2003.

[9] UML Revision Taskforce, OMG UML Specification v. 1.4., Object Management Group, 2001.

[10] UPPAAL, http://www.uppaal.com/.

付 録

A. ラベル付き遷移システムと合成

チャネルの名前の集合を Ch とする．各チャネル名 $ch(\in Ch)$ に 記号 ! を付けた出力チャネル $ch!$，記号 ? を付けた入力チャネル $ch?$，あるいは特別な記号 τ のことをラベルと呼ぶことにする．ラベルを集めた集合 $\Delta = \{ch!, ch\ |ch \in Ch\} \cup \{\tau\}$ をラベルの集合という．

状態の集合 S，ラベル付き遷移の集合 $T \subseteqq S \times \Delta \times S$，初期状態 $* \in S$ から成る三つ組 $(S, T, *)$ のことをラベル付き遷移システムと呼ぶ．また，ラベル付き遷移 $(s, l, s') \in T$ を $s \xrightarrow{l} s'$ と書くことにする．

ラベル付き遷移システムを並列合成すると (ラベルのない) 遷移システムが得られる．n 個のラベル付き状態遷移システム $LTS_i = (S_i, T_i, *_i), (1 \leqq i \leqq n)$ を並列合成して得られる遷移系 $(S, T(\subseteqq S \times S), init)$ は次で与えられる:

- $S = S_1 \times \cdots \times S_n$
- 遷移 $((s_1, \ldots, s_n), (s'_1, \ldots, s'_n)) \in T$ は次のいづれか:

– どれか一つの LTS_i に遷移 $s_i \xrightarrow{\tau} s'_i$ があり，s_i 以外は変化しない．すなわち $s'_k = s_k (k \neq i)$.

– LTS_i に遷移 $s_i \xrightarrow{ch!} s'_i$ があり，LTS_i とは異なる LTS_j に遷移 $s_j \xrightarrow{ch?} s'_j$ があり，s_i, s_j 以外は変化しない．すなわち $s'_k = s_k (k \neq i, j)$.

- 初期状態 $init$ は組 $(*_1, \ldots, *_n)$.

ラベル付き遷移システム $LTS_i = (S_i, T_i, *_i), (1 \leqq i \leqq n)$ を並列合成して得られる遷移システムを $LTS_1 || \cdots || LTS_n$ と書く．

[**例 2**] 図 A·1 の 2 つのラベル付き遷移システム，

$LTS_1 = (\{1, 2\}, \{(1, \tau, 2), (2, ch!, 1)\}, 1)$,

$LTS_2 = (\{3, 4, 5\}, \{(3, \tau, 4), (4, ch?, 5), (5, \tau, 3)\}, 3)$,

について考える．ただし，$\Delta = \{ch!, ch?, \tau\}$．図において，二重丸は初期状態を表し，ラベルが表示されていない遷移にはラベル τ が省略されているものとする．LTS_1 と LTS_2 を並列合成した結果は右の図になる．ただし，初期状態から到達しない状態は図示していない． □

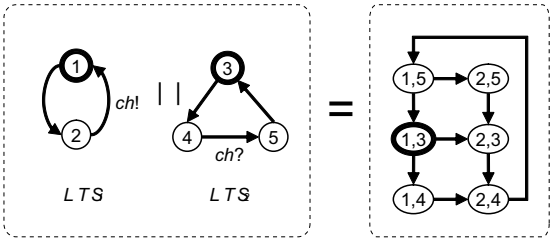


図 A·1 ラベル付き遷移システムの同期合成例