修士論文

非同期通信を行うWebアプリケーションにおける資源競合問題の検証手法の提案

電気通信大学大学院情報システム学研究科
社会知能情報学専攻

1151023 都丸　卓也

主任指導教員： 田原康之 准教授
指導教員： 大須賀昭彦 教授
指導教員： 栗原聡 教授
平成26年7月18日（金） 提出

概 要

Web アプリケーションの開発では，クライアント側のユーザビリティ向上が重要になってきており，Ajax により非同期通信が使われることで，クライアント側の操作をブロックせずに動作することが可能となっている．しかしながら，非同期通信によって予期しない順序でサーバ処理が行われる可能性がある．本研究では，特に資源の獲得・解放と非同期通信の使用時に起こる可能性のある問題と解決方法を示す．従来はこのような問題に対し，モデル検査などの形式検証手法が有効であると考えられてきたが，専門知識が必要でかつ検証の労力が大きいという課題があった．本論文では，モデル検査による検証手法を，一般の技術者にも容易に利用可能とするためにソースコードから PROMELA 記述の変換方法を提案した．これにより開発者は資源競合の検証を行う際に，Web アプリケーションのモデルを記述しないで良くなるので，専門知識が不要となり，かつ検証時の労力の大幅な軽減が可能となった．例題に手法を適用し，非同期通信時にはデッドロックが発生することを確かめた．

目 次

図 目 次

表 目 次

第1章　はじめに

従来のWebアプリケーションではブラウザはサーバから送られてきたページを表示しているだけであり，表示を変更するためにはクライアント・サーバ間通信が不可欠となっている．このような通信方法では，通信の間はユーザ操作が出来ないためユーザビリティを低下させていた．

この問題を解決するためにAjax（Asynchronous Java Script and XML）が使われ，非同期通信により，現在の表示にかかわらず，通信することが可能となった．しかしながら，非同期通信では，任意のユーザアクションが要求と対応する応答の間に起こりうるので，デッドロックなどの問題が発生するという問題がある．

さらに，Webアプリケーションが複数台のサーバから構成されており，それらが共有して資源を使用していた場合，複数の要求が異なるサーバに伝達され，サーバ間で競合が発生する可能性がある．

このような問題を確認するための方法として，現在主流のソフトウェアテストと形式検証の二つの方法がある．ソフトウェアテストでは，テスト設計者が規定したテストケースを用いて不具合がないかどうかを確認することが出来るが，今回の場合，ユーザアクションは任意であり，テストケース数は膨大になる．そのため，抜けや漏れの可能性があり，網羅的に検証することができる方法が求められている．SPINによるモデル検査法を使えば網羅的に検証することが可能である．これは，専用の記述言語であるPROMELAを入力し，検証性質が成り立つかどうかを出力する．

本研究ではモデル検査法によるRIA（Rich Internet Application）の資源競合の検出を目指し，資源へのアクセスをモデル化するとともに，Ajaxの非同期通信のモデルを使用する．これまで研究されてきたモデル検査法では専用の記述言語を使用して検証モデルを作成しなければならず，記法の習得が難しいことやRIAに特化した記法ではないために記述が複雑・増加しやすいという欠点があった．そこで本論文では，ソースコードからPROMELA

記述を生成することができる方法を提案する．これによって開発者が資源のリクエストの記述をしなくて済むことはもちろん，本質的な情報を抽出することが可能である．また，ソースコードとの対応を示すことで，検証結果で不具合が検出された際にソースコードの修正箇所を特定しやすくした．

評価として図書館システムにおいてクロール，ソースコード解析によるモデル化が可能であることを示し，非同期通信時になった場合の不具合を検出することが出来た．また todo リストアプリケーションにおいてクロール，ソースコード解析によるモデル化が可能であることを示した．

本論文の構成は次の通りである。第 2 章では，本研究が対象とする背景知識である RIA，Ajax，モデル検査について述べる．第 3 章では，Ajax アプリケーションの課題について述べる．第 4 章では，提案手法について述べる．第 5 章では，提案手法の評価と考察について述べる．第 6 章では，関連研究について述べる．第 7 章では，今後の課題について述べる．最後に第 8 章でまとめる．

第2章　研究背景知識

本章では，本研究が対象とするRIA（Rich Internet Application），Ajax（Asynchronous JavaScript + XML）およびモデル検査について説明する．すなわち，本研究が対象とする非同期通信の問題点とその発生原因，ならびに検証方法と検証を行うにあたっての課題について述べる．

2.1　従来のWebアプリケーションとは

Webアプリケーションとは，Webの機能を利用したアプリケーションのことである．ここではサーバおよびクライアントが使われる．サーバとは，様々な資源を一元管理し，これを使うことでサービスを提供するコンピュータのことである．クライアントとは，サーバに要求を出し，サービスを受けるコンピュータのことである．図2.2のように，このサーバとクライアント間での通信をサーバ-クライアントと呼ぶ．Webアプリケーションにおいてはクライアント側でブラウザを，サーバ側でWebサーバを使用する．

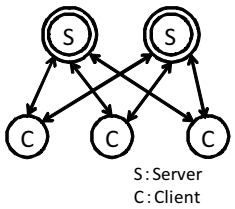

図 2.1: サーバ-クライアント

従来のWebアプリケーションはページ遷移によって動作が行われた．すなわち，Webアプリケーションは HTTP リクエストを投げると，サーバ側で処理がなされてから HTML ページが返され，ブラウザが HTML ページを表示する．この時ブラウザはサーバ処理が終わることをまたなければならなく，また，レスポンスが返ってくるとページ全体を更新し

なければならなかった（図 2.2）．これではユーザビリティが低い．そのため，ページの一部のみを更新する方法が必要となった（図 2.3）．

2.2　RIA とは

RIA とは，データをサーバおよびクライアントで処理することが出来る Web アプリケーションの一種である．特徴として，一部の画面を再表示したり，クライアントの処理が継続されたり出来るように非同期通信を行うことが挙げられる．クライアント側で処理を行うことにより，デスクトップアプリケーションと同じようなルック・アンド・フィールを提供できたり，通信量の削減ができたりするというメリットがある．従来の Web アプリケーションでは，ユーザの操作によって通信が起こり，その応答結果を待っていた．それに対して，RIA ではユーザ操作をエンジンが監視しており，ユーザ操作があり，かつサーバ通信が必要な場合にのみ通信を行うようになった．これによって，通信結果が想定と違う順番で返ってくる可能性がある．

RIA は様々な技術が使われており，ブラウザ上かクライアントの実行環境で実行される．例えば Flash や Silverlight，Java Applets，Ajax はブラウザ上で動作する．また，バイナリファイルをダウンロードし，実行環境を構築し，そこで実行される Java Web Start や Adobe AIR などがある．

2.3　Ajax とは

本研究では Ajax による非同期通信を実装した Web アプリケーションを検証対象とする．この Ajax は，Garrerr[12] によって提唱されたもので以下の定義がなされている．

- XHTML と CSS を使った標準規格のプレゼンテーション
- DOM（Document Object Model）を使った動的な表示とインタラクション
- XML と XSLT を使ったデータ変換と操作
- XMLHttpRequest を利用した非同期なデータ取得

- JavaScript が様々な技術を統合する

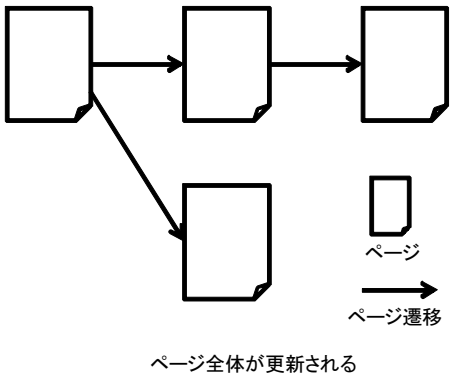

図 2.2: 従来の Web アプリケーションによる動作

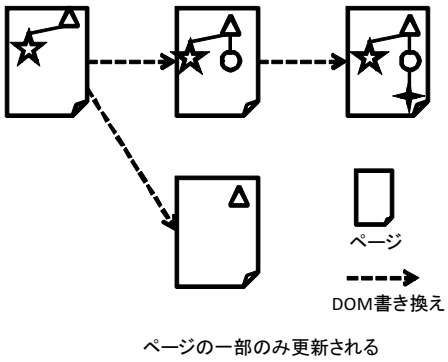

図 2.3: RIA による動作

2.4 従来の Web アプリケーションと RIA との違い

ここで従来の Web アプリケーションの構築方法とどこが異なるかについて述べる．

RIA では非同期通信を実現するために，Ajax エンジンと呼ばれる仲介者を挟む．ユーザインタフェースからの処理要求を Ajax エンジンが処理し，必要に応じて，サーバ側にリクエストを送る構造となっている（図 2.4 参照）．

この非同期通信を支える技術である XML Http Request について述べる．XML Http Request は，非同期通信を行うためのオブジェクトである [23]．図 2.5 のように，従来では，

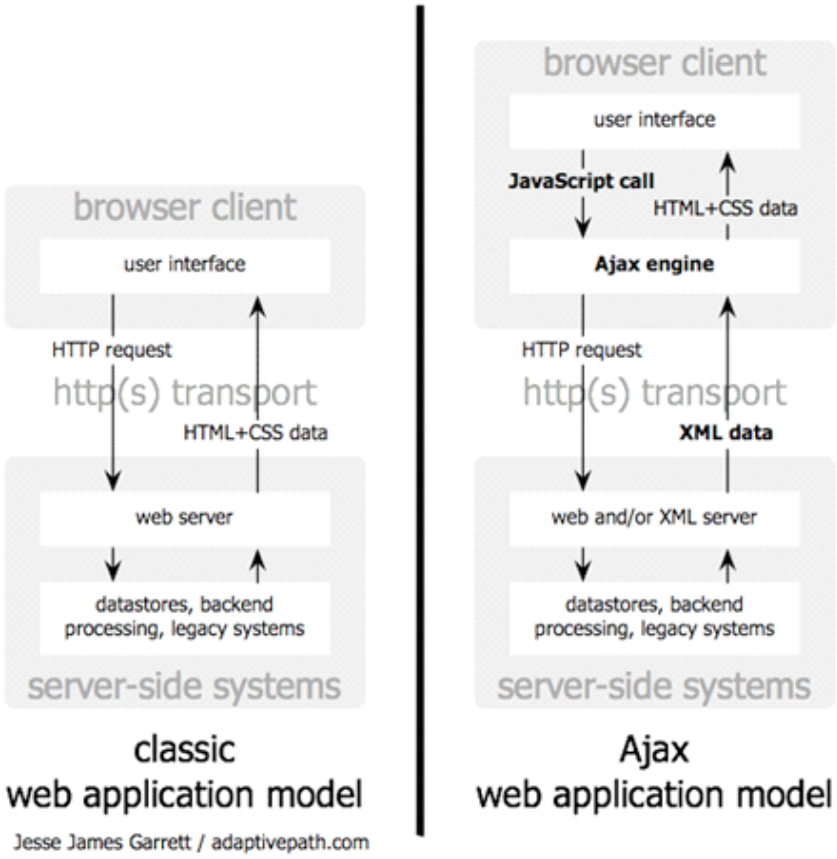

図 2.4: 従来の Web アプリケーションモデル（左側）と Ajax モデル（右側）の比較 [12]

このモデルにおいてはブラウザ側で処理をすることが不可能であり，ユーザビリティが低下するという問題点があった．このため，ユーザが待たなくてもすむ方法が提案された．すなわち，ユーザではなく Ajax エンジンが要求を常に受け付けることで，ユーザの負担をなくしたのである．

2.5 形式検証

本章では，本論文で用いるモデル検査の属する形式検証の基礎知識を述べる．

2.5.1 形式検証

形式検証技術とは，システムの信頼性向上を目的とする技術で計算機支援を前提として数学的な枠組みでモデル化し，正しさを保証するという技術である．形式検証の代表例としては，演繹法とモデル検査法の 2 種類がある．演繹法では，証明の概念を持つ論理体系でモデル化可能であれば，どのようなシステムの振舞いの正しさでも検証することが可能であるが，完全に自動的に証明を行うことができず，人手による証明のしやすさを考慮すると，システムの振る舞いや性質を限定しなければならないことがある．モデル検査法は検証に必要な証明の作業を自動的に行うため，導入の敷居が低く，有望視されている手法である．

2.5.2 モデル検査法とは

モデル検査法は，システムの振舞い仕様をモデルで，検証性質を命題で表現し，モデルと命題との間の充足関係を検証する手法である．また，対象が誤りを含む場合は反例を見つけることができる．反例は誤り箇所を特定するための重要な情報を持っているため誤りの早期発見に威力がある．ただし，網羅的な探索なので状態数が有限でなくてはならない．また，コンピュータの計算資源の制約もある．たとえば，状態空間がメモリに入りきるか，実用的な時間に答が得られるかどうかなどである．このため，モデリングが重要となる．予め対象のシステムを抽象化し適切な規模の状態遷移システムを得ることが必要不可欠となる．モデル検査法を行うツールとして，SPIN（Simple Promela INterpreter）[14, 13] が挙げ

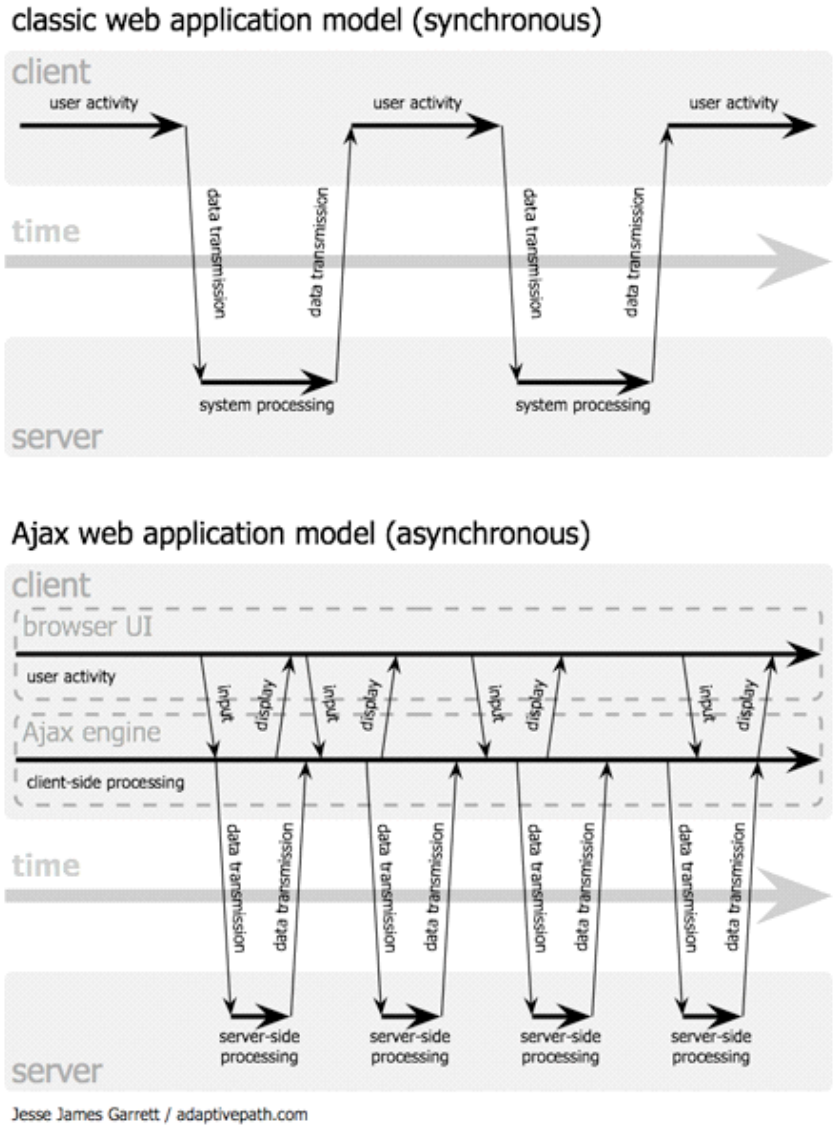

図 2.5: 従来の Web アプリケーションの同期通信パターン（上側）と Ajax アプリケーションの非同期パターン（下側）との比較 [12]

られる．SPIN で使われる記述言語は PROMELA（PROtocol/PROcess MEta LAnguage）である．そして，検査対象のシステムは PROMELA で記述することによりモデル化され，SPIN により検証される．PROMELA は，プロセス同期および協調のモデル化に重点が置かれている．

2.5.3 モデル検査の例

ここで，本論文でも検出例としている並行システムのデッドロックが検出されるモデル検査例について説明することを通し，モデル検査による障害検出について説明する．

本例題のシステムは，スキャナーから画像を読み取り，プリンタによって出力する．このシステムの利用者は 2 人おり，各々が 2 つの資源 Printer と Scanner を使用する．利用者は Printer および Scanner を同時に利用する必要がある．もし，片方の利用者が Printer もしくは Scanner あるいは両方を使用していた場合には，もう片方の利用者はシステムを利用することはできない（図 2.6 参照）．

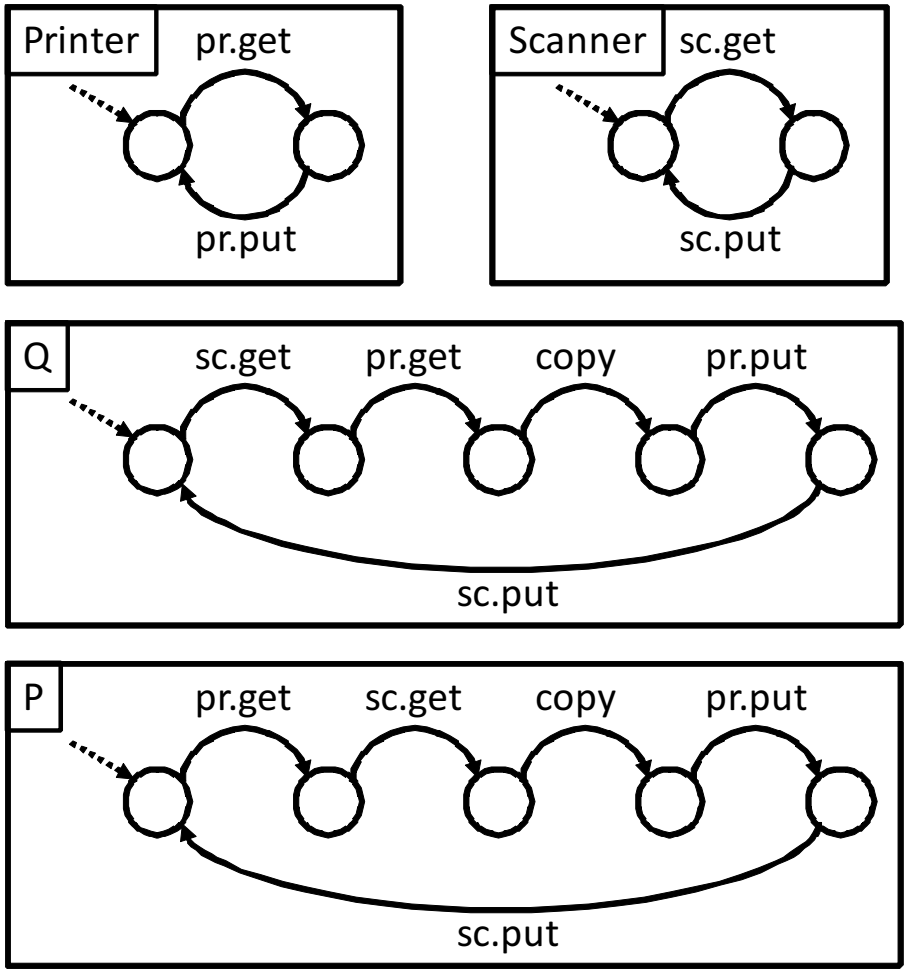

図 2.6: 利用者 P と Q が Printer と Scanner を利用するときの振舞い

第3章　Ajaxアプリケーションの課題

Ajax アプリケーションには，非同期通信に起因する障害の可能性があるが，その検出は困難である．本章ではその詳細について説明する．

3.1　非同期通信によって発生する障害

非同期通信によって発生する障害の例を述べる．図3.1にUser，Browser，Server，Databaseの4層からなる同期・非同期通信時のモデルを示す．このモデルでの記号は次の意味を持っている．

- ユーザはアクション（A）を実行でき，実行結果（V）を見ることが出来る
- ブラウザはリクエスト（R）かレスポンス（P）を処理できる
- サーバはリクエストを処理（X）できる
- データベース（DB）はサーバから資源をリクエストされる
- 矢印の元・先は処理の実行前・後を示す（破線矢印は実行できないことを示す）

このモデルでは，ユーザは，アクション A1 を実行し，A1 をキャンセルする操作 A1c を実行後，アクション A2 を実行する．ここで A1c が同期通信か非同期通信かによって振舞いが異なる．

1. 図 3.1(a) に A1c が同期通信の場合の処理を示す．A1c が発火されると，処理が返される（V1c）まで他のアクションは実行することが出来ない．これによって，必ずX2が実行される前にX1はキャンセル処理を行うことになり，資源解放が行われる．

2. 図 3.1(b) に非同期通信の場合の処理を示す．非同期通信と X1 の実行によって，X1c が実行される前に，X2 が実行されている．X1 は DB1 の資源を獲得し，次に DB2 の資源を要求している．X2 は DB2 の資源を獲得し，次に DB1 の資源を要求している．これによって，X1 と X2 の間でデッドロックが起こっている．

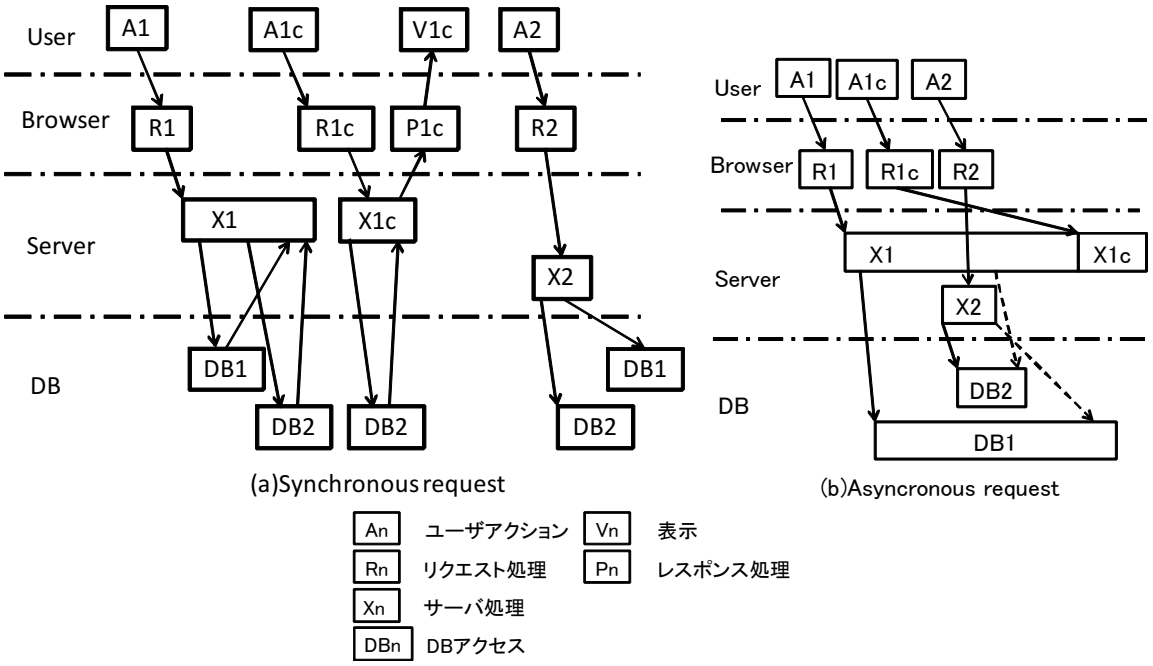

図 3.1: User, Browser, Server, DataBase 間で起こりえるインタラクション

3.2　Ajax アプリケーションの課題

Ajax アプリケーションを検証するために，玉田ら [9] は AjaxEngine，XHR_Obj，Inner_Obj，Server の 4 つのプロセスによって Ajax アプリケーションをモデル化した（図 3.2 を参照）．このモデルでは，ユーザインタフェースの部分に主眼を置いており，ユーザ要求が AjaxEngine によって通信が行われずブラウザで処理される場合は Inner_Obj プロセスを生成し，非同期通信が行われる場合には XHR_Obj プロセスを生成した．また，Server は，要求を受け取ると処理を実行し結果を返すというモデルとなっている．

図 3.2 では AjaxEngine は XHR_Obj と Inner_Obj のプロセス管理，XHR_Obj は XHR のためのオブジェクト，Inner_Obj は通信を行わないブラウザ内部の処理，Server はサーバ側の処理をそれぞれモデル化したものである．玉田らのモデルではサーバ側のモデルは処理を受け取り返すのみでデッドロックの検出は手動で変更を加えない限り行えない．

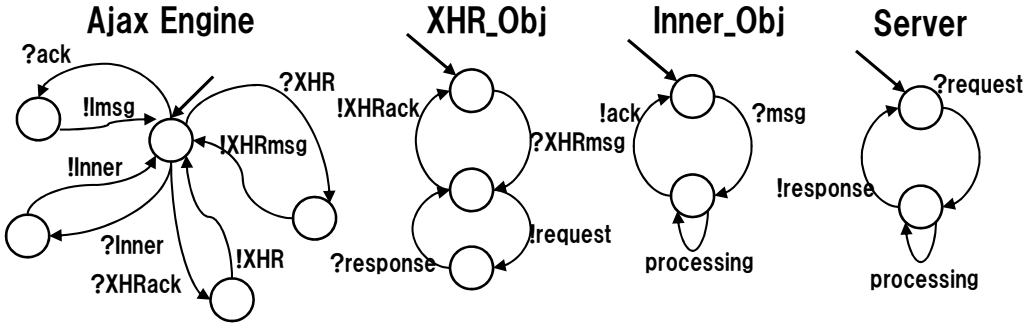

図 3.2: Ajax アプリケーションの非同期通信検証モデル [9]

まず Web アプリケーションは一般的にステートフルな振る舞いをする．よって，状態遷移システムを考えなければならない．さらに，Ajax アプリケーションにおいては要求と対応する応答の間に任意のユーザアクションを起こすことが出来る．これによって，複数の要求がサーバ側に送られ，意図しない処理が行われる可能性がある．特に，資源の獲得・解放は処理手順によってデッドロックが発生する可能性があり，要求順序が入替わる可能性のある Ajax アプリケーションでは，この検出は必要である．

3.3 非同期通信によって発生する障害の検出

この節では非同期通信によって発生する障害を検出する方法について述べる．

3.3.1 デッドロックの検出

User，Browser，Server，Database の 4 層モデルをモデル検査ツール SPIN を使用して検証する．障害を検出する Promela 記述の一部をソースコード 3.1 に示す．ここで使用している条件分岐 if と通信チャネルについて説明する．まず条件分岐 if について説明する．条

件分岐 if は，if で始まり fi で終る．条件と命令の組を::で区切って列挙することにより，条件が成り立つ命令が実行される．複数の条件が成り立つ場合には，成り立つ命令の内のいずれか一つが実行される．次にチャネルについて説明する．SPIN では複数のプロセスが生成でき，それらのプロセス間で通信が行える．このときチャネル経由でメッセージを送受信するが，その記述は以下のようにする．

- 変数!メッセージ

  変数に格納されているチャネルにメッセージを送信する

- 変数?メッセージ

  変数に格納されているチャネルからメッセージを受信する

ソースコード 3.1 の記述内容について説明する．クライアント側からのリクエスト req によって，サーバ側の動作を分岐させている．もし，キャンセル処理（req==x1c）ならばグローバル変数のキャンセルフラグを真にし，各プロセスがキャンセル処理を行うようにしている．キャンセル処理が行われる場合は，割り込みなしにデータベースを解放する．

ソースコード 3.1: server 側の Promela 記述

```
1  proctype Server(byte me){
2  mtype req;
3   request?me, req−>
4   if
5    ::(req == x1)−>db1!get;
6        if
7            ::atomic{(cancel==true) −> db1!release; cancel=false;}reply!req;
8            ::atomic{(cancel==false) −> db2!get;}
9            if
10             ::atomic{(cancel==true) −> db1!release; db2!release;cancel=false;}reply!req
                   ;
11             ::else ;db1!release;db2!release;reply!req;
12           fi;
13         fi;
14    ::(req == x2)−> db2!get;db1!get;skip;db1!release;db2!release;reply!req;
15    ::(req == x1c)−> cancel = true;reply!req;
16   fi;
17  }
```

3.3.2 順序制約に違反した場合の検出

別の例として，画像アプリケーションを考える．このアプリケーションでは画像の編集，保存，削除を行うことが可能である．

図 3.3 のように，このアプリケーションをユーザが編集，削除の順で操作する時を考える．もし同期通信が使われた場合には，ユーザが意図した通りに編集，削除の順で実行されるため問題はない．しかしながら，非同期通信が使用された場合には，削除，編集の順で操作がなされる場合がある．この場合には，ユーザは編集した結果を想定しているのにも関わらず，編集結果が反映されずに削除されるため，不具合と認識する．そのため，操作順序に制約が存在することとなる．

この制約を発見するには，すべての操作順序を考慮しなければならなく，なかなか発見し難いと考えられるため，モデル検査器を使用する．

Promela 記述をソースコード 3.2 に示す．ここでは編集・保存・削除のいずれかが起こった場合には資源を取得し操作を行い開放を行う．

本例題で問題となっているのはレスポンスの順番であるため，レスポンスが識別できるようにあらかじめラベルをふっておく．リクエストの到達時には s を頭文字とし，以降に操作の名前を記述する．レスポンスの送信時には e を頭文字とし，以降に操作の名前を記述する．これによって，サーバにリクエストが到達した時と，サーバからレスポンスが返送される時が明確になる．

このラベルを使い検証性質の記述を行う．本例題では編集が行われた際にはいつか保存が行われるという性質のためソースコード 3.2 の最下行の LTL 式になる．

ソースコード 3.2: server 側の Promela 記述

```
1  active proctype UI(){
2  endUI:
3    Ajax(0, edit);Ajax(1, save);CallBack();CallBack()
4  }
5  proctype Server(byte me){
6  mtype req;
7       request?me, req->
8       if
9                 ::(req == save )->
10 ssave: resource1!get; resource1!put;
11 esave: reply!req;
12                ::(req == delete )->
13 sdelete: resource1!get; resource1!put;
```

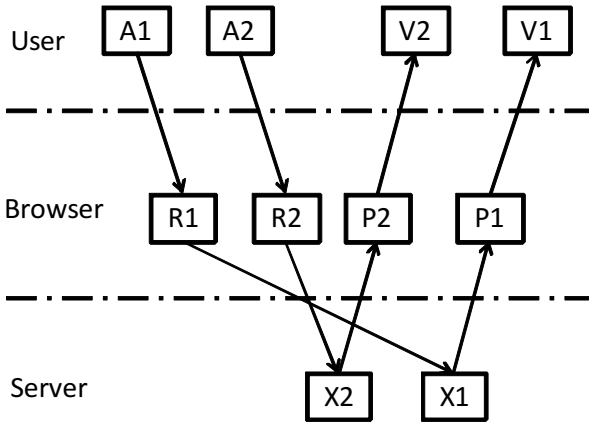

図 3.3: 非同期通信による処理順序の変更

```
14 edelete: reply!req;
15                  ::(req == edit )->
16 sedit: resource1!get;resource1!put;
17 eedit: reply!req;
18       fi;
19 }
20
21 active proctype RESOURCE1(){
22 endDB:
23   do
24     :: resource1 ? get -> resource1 ? put;
25   od
26 }
27
28   ltl c1 { [](( Server[4]@sedit) -> <>(Server[6]@ssave)) }
```

3.3.3　検証における問題点

このように SPIN を用いて検証を行うことが可能であるが，以下の欠点が挙げられる．

- Promela は、並行プロセスと非決定的な振舞いの記述が中心であるため、一般の技術者には習得が難しい．
- 資源のリクエスト・レスポンス順序以外の記述が必要．例えば，変数の値によってはサーバ側で障害が発生する可能性がある．
- 資源が多くなるとコード行数が増加し，関係性が読み取りにくくなる．Promela 記述はプロセスに着目しており，資源に着目していない．

3.4 UML とソースコードの制約

崔ら [18] は画面遷移図とプログラムを記述したフローチャートから検証モデルを作成し，画面遷移の仕様を命題時相論理 CTL（Computation Tree Logic）で表現して検証を行っている．

大須賀ら [20] の研究では，アプリケーションの振舞いをセッション管理に関する処理と限定することで，UML モデルからの情報抽出および抽象化を行った．彼らが述べているようにデータの読み書きや共有変数を扱うような動作を行うものなどには未対応である．

ソースコードを静的解析し検証モデルを作成するツールが存在するが，それらには Java 言語のみといった言語対応の制約が存在する．しかしながら，非同期通信時に利用される Ajax は開発方法論の一つであり，特定の言語に特化したツールでは対応できない．

3.5 モデル検査適用における問題点

モデル検査にあたりモデルを作成する必要がある．モデル検査を現実のシステムに適用する場合の問題点について述べる．

3.5.1 開発現場におけるモデル検査の適用の問題点

Web アプリケーションの製作はウォーターフォールモデルよりも，スパイラルモデル（図 3.4 参照）が採用される．

ウォーターフォールモデルにおいてはソフトウェア開発工程を上流工程から順番に行う．すなわち，ソフトウェアが作成され廃棄されるまでのステップにおいて，一度ステップを踏んだあとには原則として後戻りすることができないと捉え，各工程に十分な時間をかける．

スパイラルモデルにおいてはソフトウェアの一部分の設計・実装を行い，ソフトウェア開発工程を反復する．

現在 Web アプリケーションは，短期の納期が求められている．また厳密な仕様書の作成がないこともあり，従来のモデル検査が適用不可能である場合がある．本論文ではソースコードから Promela 記述に変換しているため，Web アプリケーションから手法適用の導入敷居が低い．

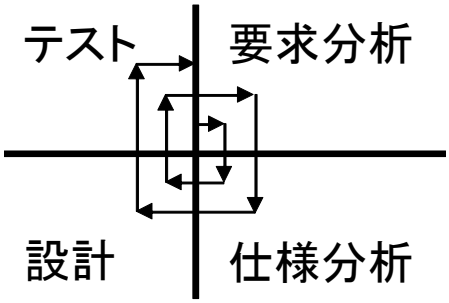

図 3.4: スパイラルモデル

3.5.2　モデル作成方法

モデルの作成方法には大別して二通りある．従来のモデル検査とモダン・モデル検査である．

従来のモデル検査においては，ソフトウェア開発工程の上流工程において作成されたモデルが設計者の意図した通りになっているかを確かめるために行う．ここでは，SMV，SPIN，LTSA といったツールが利用される．

この手法においては検査対象であるシステムの振舞いをもれがないように検証者が正確に記述しなければならない．モデル作成には専用の言語が必要であり導入の敷居が高い．

本論文では SPIN を用いているがソースコードから Promela 記述に変換しているためモダン・モデル検査に近い．しかしながらデータベースアクセスや UI 動作以外は捨象しているため抽象化の度合いが大きい．

一方モダン・モデル検査は開発工程のテスト・分析工程でデバッグの一種として利用されている．モダン・モデル検査では，デザイン記述やプログラムから，自動的な抽象技法を用いて，有限状態機械を抽出し検証を行う．この領域では，SLAM，JPF など多くのツールが考案され，実際に利用されている．

この手法においては，実装したソフトウェアが，設計者が期待する動作を行うかどうか確認することができる．自動的な抽出法では人間による誤りが入らないという利点がある．しかしソースコードが大きくなると時間がかかることやソースコードがない場合には適用不可能であるという問題がある．

第4章　提案手法

従来のWebアプリケーションではクライアントからのある要求があった場合，その要求をサーバが処理し終わるまで，サーバは次の要求を処理しなかった．しかしながら，RIAでは，非同期通信でクライアントからの要求がなされるため，サーバが処理する順序が変わる可能性がある．この動作を表現するためには非同期通信を表現する層をモデルに組み込まなければならない．すなわち，非同期通信を実現するAjaxEngineとXHRオブジェクトをプロセスとして定義し，検証を行うこととした．図4.1に想定しているRIAにおける通信の全体構造を示す．ここでXHRオブジェクトは非同期通信ごとに生成される．プロセスが新しく生成されることで，他のプロセスとの実行順序が変わる可能性がある．

検証手順の全体について説明する（図4.2参照）．次のようなステップで変換を行う．

1. ソースコードからSQL-Treeを抜き出す[21].
2. クロールによってユーザ動作を抽出する．
3. SQL-Treeおよびクロール結果をPromela記述に変換する（提案手法）．
4. SPINによって検証を行う．
5. SPINの出力結果に基づき不具合修正を行う．

4.1　共通利用モデル

この節では全てのモデルで共通に利用するモデルについて述べる．図4.3に示すAjax_Engineプロセスは，ユーザとサーバの間の通信を中継する．ユーザまたはサーバから非同期通信の要求を受け取り，サーバまたはユーザに送信する役割を持つ．ユーザ・サーバにいつ送信す

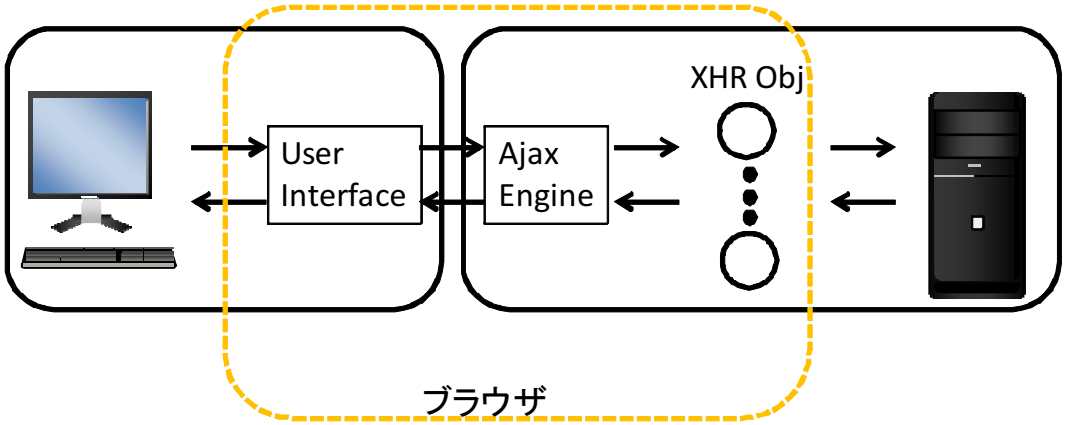

図 4.1: プロセス全体の構成

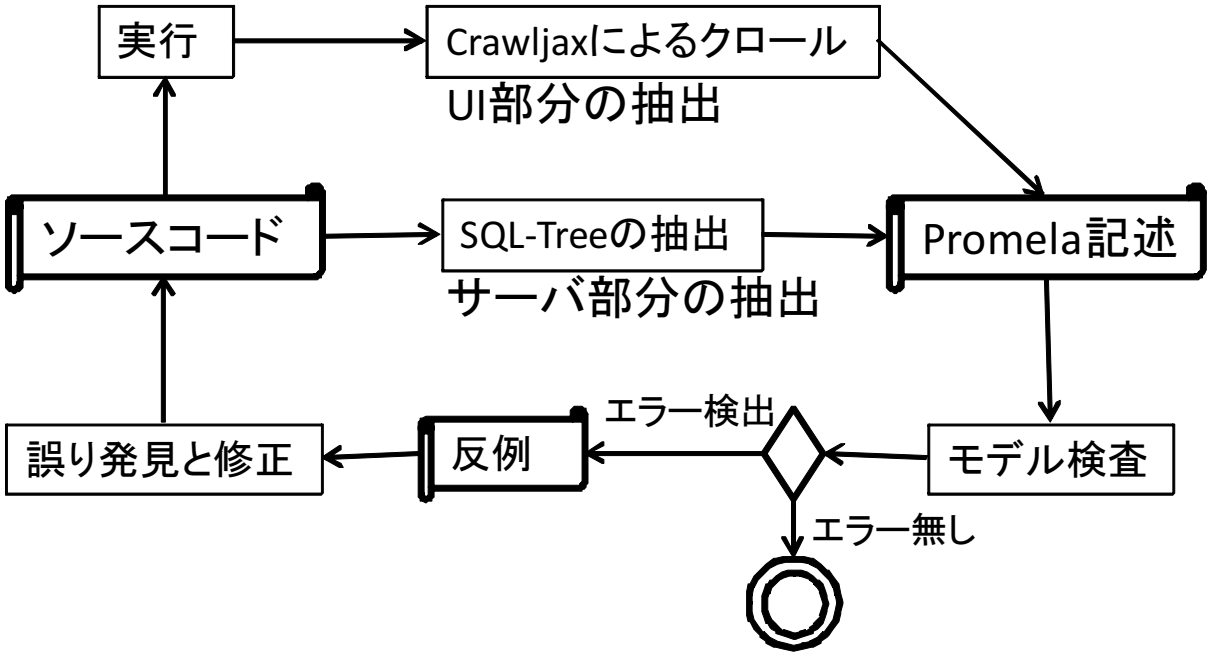

図 4.2: 検証の流れ

るかは決まっていないため，直接ユーザ・サーバ間を橋渡しするのではなく，XHR_Obj を生成することで実現した．また，コールバックもいつ呼ばれるかわからないため CallBackRes プロセスを生成することで表現した．

各モデルがソースコードのどの部分に当たるのかについて述べる．Ajax_Engine プロセスは付録のソースコード 5.1 において 124 行目から 131 行目にあたる．XHR_Obj は 133 行目から 138 行目にあたる．CallBackRes は 41 行目から 44 行目にあたる．

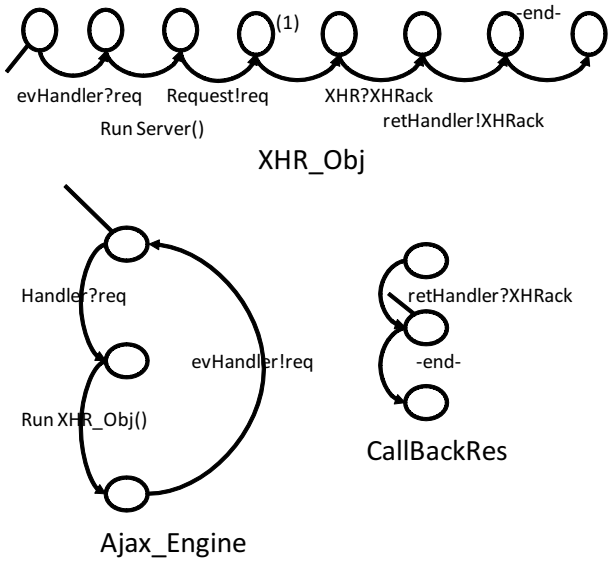

図 4.3: 共通利用モデル

4.2 ユーザ動作の抽出法

ユーザプロセスはサーバに対してリクエストの送信を行う．通常，ユーザが操作できるのはブラウザである．アプリケーションによってページ内にリンクや入力などがあり，それによって動作することがあるが，ブラウザの進む・戻る・更新などのボタンや URL を直接入力することなどによって，アプリケーションが意図しないリクエストを受信する可能性がある．

ユーザプロセスをモデル化する際には大きく分けて二通りの方法がある．すなわち，クローラによって RIA を自動的に操作・実行・記録することでモデルを作成する方法と，静

的解析によってモデルを作成する方法がある．前者のメリットは，検証者がアプリケーションのソースコードを必要としないという点である．後者のメリットはアプリケーションの挙動を厳密に定義することが可能であるという点である．

クローラによる抽出方法ついて説明する [9]．Crawljax を用いて Ajax アプリケーションから振舞いを抽出し，State-flow graph を作成し，手動で非同期通信部分を追加する．State-flow graph から Promela へ変換する．

詳細を Algorithm1 に示す．SFG とは State-flow graph の略である．Count はノードの数を数える関数である．親ノードから変換を開始し，ノードが 1 つの場合か複数の場合かにより処理が分岐する．1 つの場合には USER プロセスからアクションを受け取り，非同期通信か同期通信でサーバを呼び出し，User プロセスにレスポンスを返す．複数の場合には if 文にて分岐させそれぞれで 1 つの場合と同様の処理を行う．

Algorithm 1 UI 部の変換法

```
for u ∈ SFG do
  if 1 = Count[Adj[u]] then
    Algorithm2 を呼び出す
  else if 1 < Count[Adj[u]] then
    Adj[u] 個の if 文の分岐を用意する
    for 用意した分岐それぞれについて do
      Algorithm2 を呼び出す
    end for
  else
    goto endIndex
  end if
end for
```

4.3　サーバ動作の抽出法

サーバプロセスは，ユーザからの要求を受け取り，要求された処理を行い，結果を返すアプリケーションの挙動を示す．サーバプロセスでは，ユーザのリクエストごとに動作が変わる．リクエストによってサーバがどのように動作するのかを記述することは検証の上で重要である．サーバの動作記述を適切に行うことで，システムの検証が可能となる．

Algorithm 2 UI 部の変換法（各リクエストごとの変換）

Operation?input; と記述する /* USER プロセスからアクションを受け取る */
if $XHR == on$ **then**
Ajax(s, me) と記述する．/* Promela では次のマクロ文を呼び出す． inline Ajax(s, me){ Handler!s,me->CallBack(me); } s には呼び出すリクエスト名を指定し，me には一意の数字を指定する．*/
else
Sync(s, me) と記述する． /* Promela では次のマクロ文を呼び出す． inline Sync(s, me){ mtype server;run Server();request!s,reply[me];reply[me]?server;} */
end if
Operation!output; と記述する /* レスポンスを返す */
次の状態に goto 文にて遷移する

資源を操作する方法には様々な方法がある．変数やファイルやデータベースなどである．アプリケーションではデータベースに格納されるデータが，永続的なデータであり，そのアプリケーションの振る舞いを表す上で重要であると考えられる．そこでデータベースへのアクセスに着目することで検証範囲を狭めることが考えられる．これによって，検証モデルを小さくすることで状態爆発により検証不可能な場合をなくすとともに，もし反例が出現しても比較的容易に解析することができる．

4.3.1 データベースへのアクセス方法とトランザクションの識別方法

データベースへのアクセス方法の抽出のためには，SQL 文がソースコードのどの位置に出現するのかを調べる必要がある．すなわち SQL 文の出現位置を調べ，SQL 文どうしがどのような構造になっているのかを明らかにする [21]．ソースコードから条件分岐を考慮し SQL 文を抽出するには図 4.4 のようにする．ここでは SQL Tree という条件分岐構造を抽出し，解析することにより得られる同時に実行される SQL の集合がトランザクションファンクションである．SQL-Tree には三種類のノードがある．

- Code Node：ソースコードに対応するノード
- Branch Node：条件分岐に対応するノード
- SQL Call Node：SQL 呼び出しに対応するノード

これらのうち SQL Call Node が資源の取得開放のモデルを作成するうえで重要である．また Branch Node は条件分岐が多様なため手作業で変換することとした．しかしながら Branch Node では，Promela 記述では条件なしで分岐させるようにすれば効率は落ちるものの自動化がより高まることが期待される．本論文では SQL-Tree 抽出結果をもとに自動生成を行う．

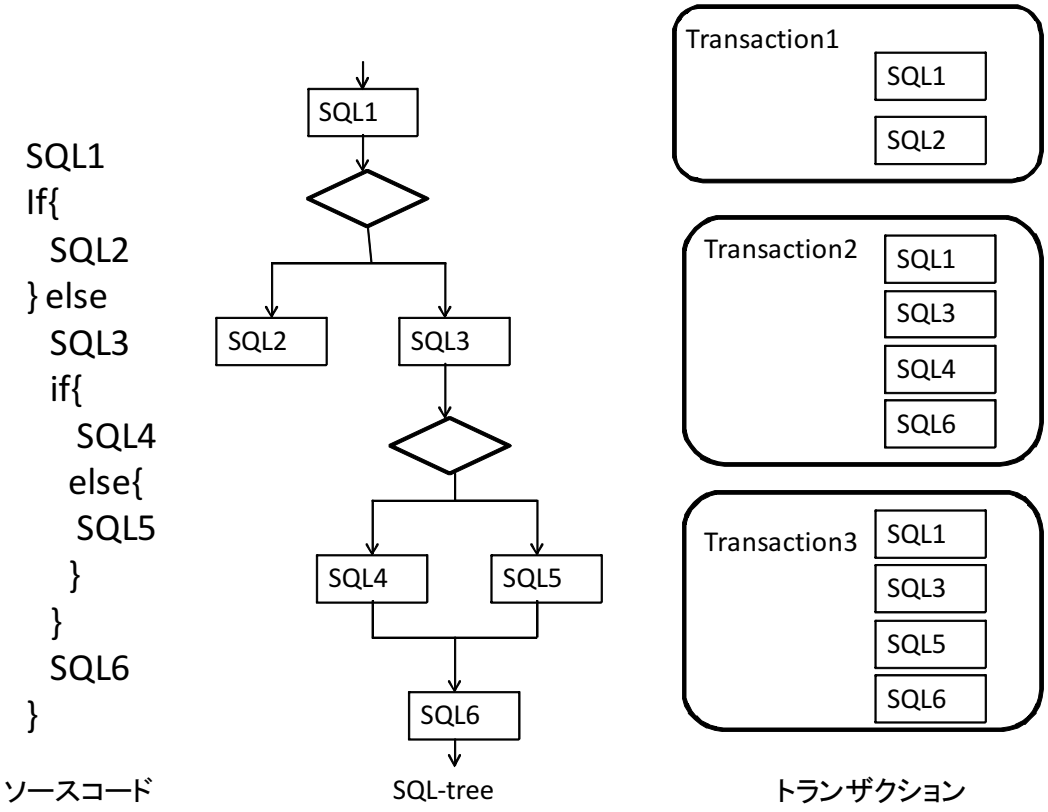

図 4.4: SQL Tree とトランザクション

4.3.2　SQL-Tree から Promela 記述の自動生成

SQL-Tree から Promela 記述への自動変換の方法を行う手順を Algorithm3 に示す．SQL-Tree の集合の中から一つ選び変換を開始する．SQL-Tree の親ノードから変換を開始し，遷移先が一つか複数かにより場合分けを行う．1 つの場合には資源の取得を行う．複数の場合には if 分で場合分けし，条件を書き，それぞれで資源の取得を行う．

Algorithm 3 Server 部の変換法

1: **for** $SQLTree \in SQLTrees$ **do**
2: **for** $u \in SQLTree$ **do**
3: 親ノードから変換を開始する
4: **if** $1 = Count[Adj[u]]$ **then**
5: 子ノードの資源取得を行う．例えば，Promela では次のようになる．
 db!get;db!release;
6: **else if** $1 < Count[Adj[u]]$ **then**
7: Count[Adj[u]] 個の if 文の分岐を用意する
8: **for** 用意した分岐それぞれについて **do**
9: 適切な mtype 型の変数を追加し，一致したら分岐するようにする．例えば Promela では次のようになる．
 if::(req == index)-> ···
 ::(req==addForm)-> ···
10: 資源取得を行う．同期通信のデータベースアクセスは lock と unlock を挿入し、排他制御を行う
11: **end for**
12: **end if**
13: **end for**
14: replyChannel!req; と記述する　/* レスポンスを返す */
15: 子ノードの変換を行う
16: **end for**

4.4　ソースコードと Promela との対応付け

ソースコードから Promela 記述を生成した後，SPIN により検証を行い，検証結果をもとにソースコードの修正を行うことが考えられる．ここで Promela 記述とソースコードとの対応をつけることで，修正すべき箇所を特定することが容易になると考えられる．

非同期通信を実現する Ajax のコードはソースコード 4.1 のようになる．このコードは 2 行目の req.open メソッドにより，サーバーの login.php にリクエストを非同期で送信している．そして 3 行目で，コールバックメソッドが登録され，リクエストが返ってきた場合に実行される．

ソースコード 4.1: Ajax による非同期通信

```
1 var req = new XMLHttpRequest();
2 req.open(‘ GET ’,‘ login.php?name= ’+ username, true);
3 req.onreadystatechange = function () {
4       if(req.readyState == 4) {
5          // do something
6       }
7    };
8 req.send(null);
```

これを Promela 記述に書き直すとソースコード 4.2 になる．ソースコード中に XMLHttpRequest があった場合には，ソースコード 4.2 の Ajax，AjaxEngine，XHRObj の Promela 記述を生成する．ここでソースコード 4.1 の 2 行目の引数は，資源のリクエストに影響を及ぼさないため捨象する．

ソースコード 4.2: Promela による非同期通信

```
1  inline Ajax(no, s){
2    Handler!no, s;
3  }
4  active proctype Ajax_Engine(){
5    byte procid;
6    mtype req;
7  endAjax:
8    do
9      ::Handler?procid, req;run XHR_Obj(procid);evHandler!procid, req;
10   od
11 }
12 proctype XHR_Obj(byte me){
13   mtype req;
14   evHandler?me, req;run Server(me); request!me, req;
15 }
```

4.5 検証性質の定義

非同期通信が適切に行われているかどうかを調べるために，成り立つ性質を与える必要がある．

従来から考えられている検証項目は以下のとおりである．

- 到達可能性：例えば，ページ B からページ A に到達できる
- 最短経路：例えば，ページ A はページ B から少なくとも N 個のステップだけ離れている
- 順位：例えば，ページ A を訪れるためには，ユーザは先ずページ B を通過しなければならない
- デッドロック：処理が先に進まなくなる状況
- 順序制約：例えば，処理 A が行われてから処理 B が行われるという制約がある

第5章　提案手法の評価と考察

5.1　図書館管理システムによる実験

図書館管理システム bcat[24] に対して本手法を適用し，検証を行った結果について述べる．Promela 記述としてあらかじめ用意したコードは 51 行である．これに UI および Server 部分の変換結果を付け加えることで SPIN による検証を行うことが可能である．

図書管理システムのページ遷移を図 5.1 に示す．プログラムは Struts を使用し Java コード行数は 4934 行，JSP コード行数は 657 行である．

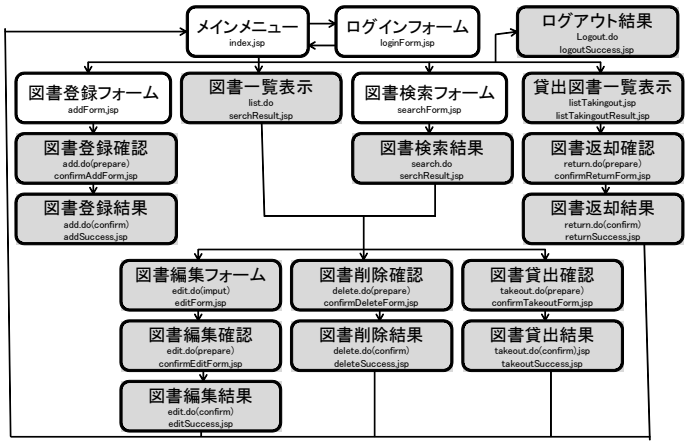

図 5.1: 図書管理システム

5.1.1　Promela への変換結果

検証に使った UI 部分の Promela 記述の行数は表 5.1 の通りである．crawljax による抽出結果を図 5.2 に示す．右上と右下が index.jsp，左上から左下が上から順番に loginForm.jsp，addForm.jsp，searchForm.jsp，list.do，listTakingout.do である．

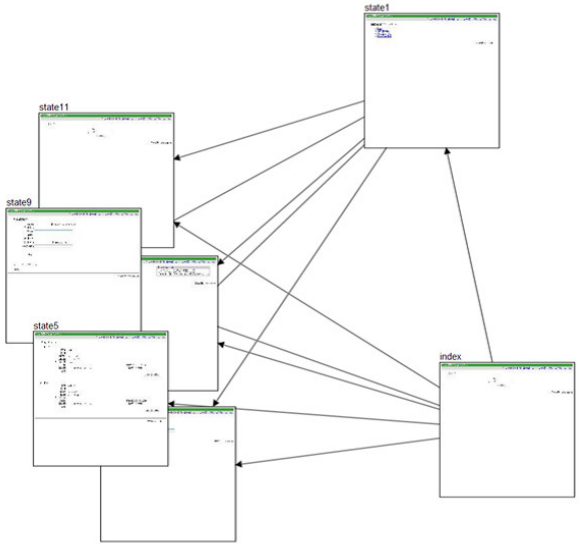

図 5.2: 図書管理システムのクロール結果

表 5.1: UI 部分の Promela 記述の行数

全体行数	42
自動生成した行数	13
手作業で追加した行数	24
手作業で修正した行数	5

表 5.2: Server 部分の Promela 記述の行数

全体行数	48
自動生成した行数	41
手作業で追加した行数	0
手作業で修正した行数	7

検証に使った Server 部分の Promela 記述の行数は表 5.2 の通りである．UI 部分において追加した行数 24 行は crawljax によってクロールできなかったページへの遷移である．また修正したのは遷移先が異なる場合が発生したということである．

Server 部分の Promela 記述において変更した 7 行はソースコード 5.1 の add_step==prepare と add_step==confirm のように条件分岐する箇所が 7 箇所あったということである．このソースコードは Algorithm3 において 8・9 行目の変換アルゴリズムで生成したものである．

ソースコード 5.1: 図書館管理システムの Promela 記述の一部

```
1 ::(req == add )−> true −>
2         if
3         ::(add_step == prepare)−>skip;replyChannel!req;
4         ::(add_step == confirm)−>bookcatalog!get;bookcatalog!release;
```

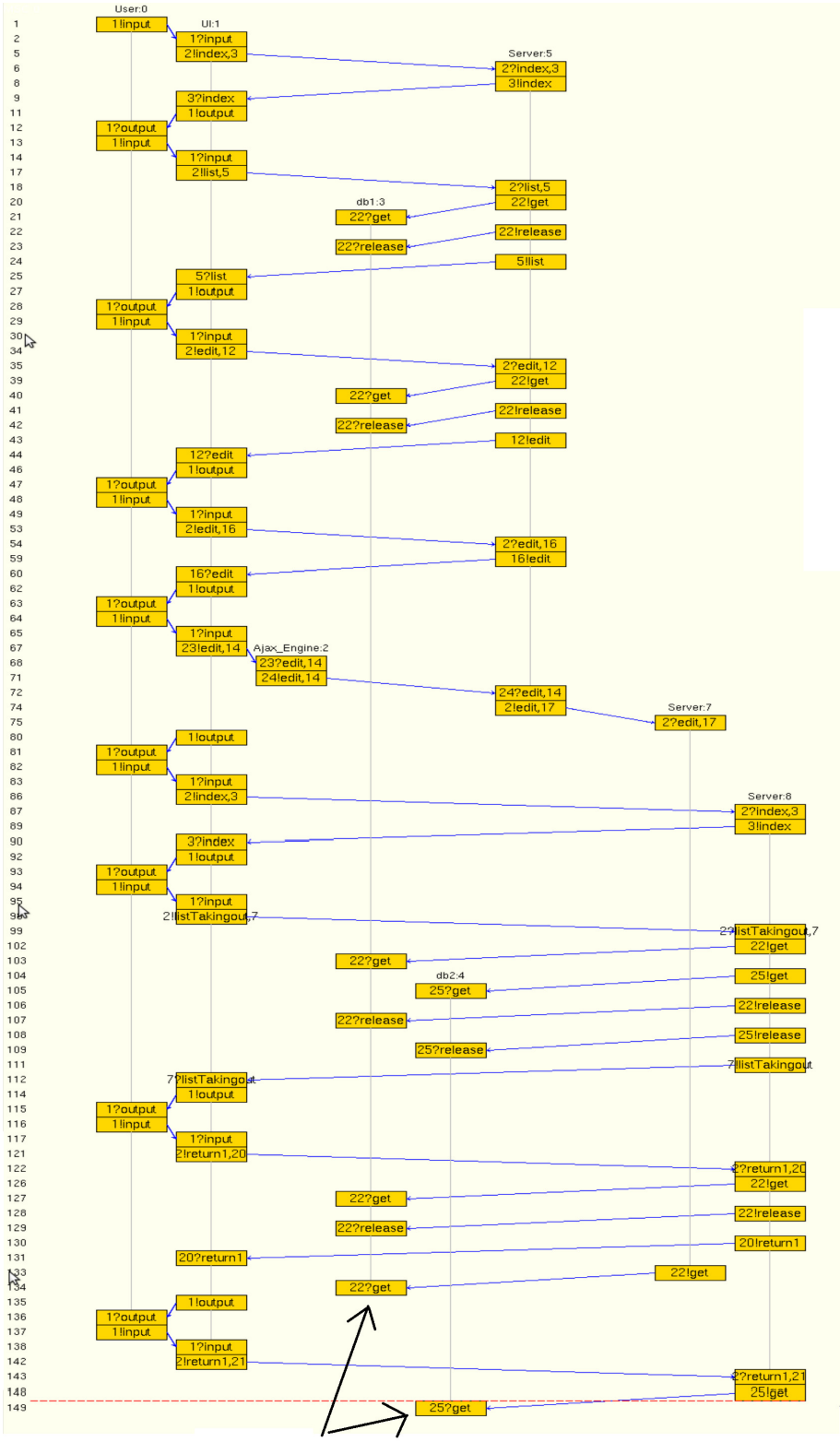
User:0
UI:1
Server:5
db1:3
Ajax_Engine:2
Server:7
Server:8
db2:4
editがdb1をreturnがdb2にアクセスし互いに異なるデータベースに
アクセスしようとしておりデッドロックが発生している

図 5.3: 反例

5.1.2 検証結果

Promela 記述に変換した後，形式検証を行った結果を表 5.3 に示す．同期通信時にはデッドロックがないことが確認できた．また，非同期通信時にデッドロックが発生する場合があることが確認できた．図 5.3 に edit 操作を非同期通信にした場合に発生したデッドロック時の反例を示す．edit が非同期通信で実行されており後で実行されていることが分かる．edit が db1 を取得し次に db2 にアクセスを要求し，return が db2 を取得し次に db1 にアクセスを要求しているため，互いに相手がほしい資源を取得しあっているというデッドロックが起こっている．

非同期通信時の検証結果を表 5.4 に示す．

表 5.3: 同期通信時の検証結果

総実使用メモリ	129MB
実行時間	0s

表 5.4: 非同期通信時の検証結果

総実使用メモリ	134MB
実行時間	0.09s

手作業で変更を行った後の全 Promela 記述を付録のソースコード A.1 に示す．

5.2 struts2todo による実験

todo リスト管理システム Struts2todo に対して本手法を適用し，検証を行った結果について述べる．Promela 記述としてあらかじめ用意したコードは 51 行である．これに UI および Server 部分の変換結果を付け加えることで SPIN による検証を行うことが可能である．

プログラムは Struts を使用し Java コード行数は 5693 行，JSP コード行数は 387 行，JavaScript コード行数は 22 行である．

表 5.5: UI 部分の Promela 記述の行数

全体行数	29
自動生成した行数	26
手作業で追加した行数	3
手作業で修正した行数	0

表 5.6: Server 部分の Promela 記述の行数

全体行数	17
自動生成した行数	17
手作業で追加した行数	0
手作業で修正した行数	0

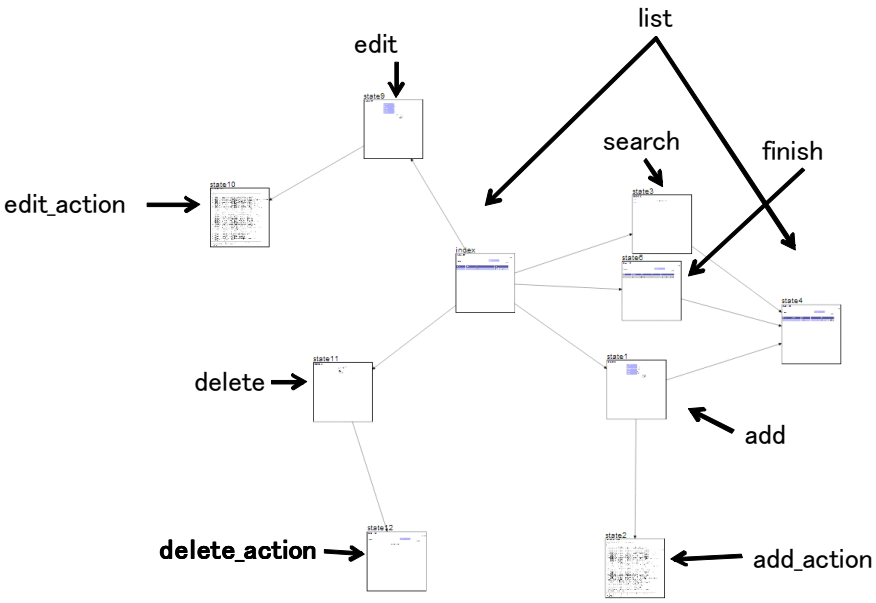

図 5.4: Struts2todo のクロール結果

5.2.1　Promela への変換結果

検証に使った UI 部分の Promela 記述の行数は表 5.5 の通りである．手作業で追加した 3 行はログイン画面の追加である．ログイン画面はユーザ ID とパスワードの入力が必要であるため汎用的な crawljax では抽出が出来なかった．

crawljax による抽出結果を図 5.4 に示す．

検証に使った Server 部分の Promela 記述の行数は表 5.6 の通りである．Server 部分は if 分による分岐がなかったため SQL-Tree から自動的に変換することが可能である．

手作業で変更を行った後の全 Promela 記述を付録のソースコード A.2 に示す．

5.2.2　検証結果

Promela 記述に変換した後，同期通信時の形式検証を行った結果を表 5.7 に示す．

edit_action，delete_action，add_action を非同期通信にして形式検証を行った結果を表 5.8 に示す．デッドロックは発生しないことを検証できた．

表 5.7: 同期通信時の検証結果

総実使用メモリ	129MB
実行時間	0.01s

表 5.8: 非同期通信時の検証結果

総実使用メモリ	228MB
実行時間	2.42s

5.3 考察

従来から Web アプリケーションに対する検証は報告されてきたが，いずれもページ遷移に基づくもので，検証性質は到達可能性を調べることがほとんどであった．しかし，本提案手法ではリクエストごとに条件を与えることが可能であり，生成した Promela 記述をもとに，より詳細に検証性質を与えることができる．本論文ではデッドロックの検証を行った．

提案したソースコードを Promela 記述に変換する手法を用いることで，従来では習得に時間がかかった Promela 記述の習得を軽減することが可能であると考えられる．どれだけ自動的に Promela に変換できるかについて述べると，UI 部分に関しては crawljax の抽出精度に依存し，もし抽出がうまくいっていた場合には手作業での修正が非同期通信を行う箇所の修正のみになる．Server 部分に関しては分岐の条件を考えなければすべて自動的に変換することが可能である．

図書館管理システムおよび todo リスト管理システムはどちらも Struts を使用している．todo リスト管理システムは JavaScript が使用されている．どちらのシステムの場合も UI 部分の抽出においてはクロールによるので違いがない．サーバー部分の抽出もどちらも SQL-Tree の抽出は Java で書かれたソースコードから行う．todo リスト管理システムの検証結果では使用メモリが同期通信の場合 129MB だったのが非同期通信では 228MB と 1.8 倍にもなった．これは非同期通信によりインタラクションが増えたことが原因であると考えられる．しかしながら三つの要求を非同期通信に変更しても十分に検証可能なメモリ容量の範囲となった．

第6章　関連研究

関連研究として，ソフトウェアテスト，モデル検査，競合の検出に関する研究が挙げられる．

6.1　Webアプリケーション・テスト

現在のWebアプリケーションのテストツール（Seleninum，WebKing，Sahi）は，ユーザの操作によって発生したイベントをキャプチャし，リプレイすることによって実現している．しかしながら，キャプチャ・リプレイツールは，ユーザにキャプチャの操作が必要なため，自動化が望まれている．

Webアプリケーションのモデルベーステストの手法はRiccaとTonella[1]によって提案された．彼らはモデルを作成するRewebとテストケースを生成するTestWebの二つのツールを示した．

別の手法として，Ajaxアプリケーションを有限状態機械としてモデル化する手法が，Marchettoら[2]によって提案された．彼らはモデルをトレースによって構築し，テスターがモデルを洗礼した後，テストケースを生成している．その後，Mesbashら[8]によりAjaxアプリケーションのユーザイベントをロボットが発生させ，それをクロールすることでモデルを作成する手法が提案された．

さらに，サーバー側のソースコードの静的解析によりアプリケーションの振舞いを抽象化する手法がある．

Artiziら[3]は，PHPスクリプトのバグを見つけるためにApolloと呼ばれるツールを提案している．Apolloは，コンコリック実行と呼ばれる手法によって，実行時エラーと不正なHTML出力を検出している．

また，Artziらは，フィードバック指向のテスト手法を提案し，Artemisというツールに

実装した [4]．彼らは，優先順位関数を定義し，網羅率が高くなるように，イベントハンドラの実行順序を入れ替えられるようにした．

しかしこれらの手法は従来型のテストであり，実行した振舞いに不具合があるかどうかは検証できるが，全ての振舞いを網羅的に調べるわけではない．本研究で使用したモデル検査法ならば厳密な検証が可能である．

6.2 モデル検査

Tkachuk ら [5] は，モデル検査ツール Java PathFinder を使って，網羅的にデータを生成し，テストケースを生成している．

Halle ら [6] は，NSM（Navigation state machines）を定義し，モデル検査ツール NuSMV を使って，モデルが検証性質を満たしているかどうかを確認することで，NSM プラグインを作成することを提案した．

本間らは，資源を変数を使い表すことで，状態遷移の条件や代入を定義し，モデル検査を行っている [10]．

以上で検証されている対象は，ページ遷移の Web アプリケーションであり，本研究のように非同期通信が対象ではない．

6.3 競合の検出

Zheng ら [11] は非同期呼び出しによって生じる非決定的な問題を検出する静的解析手法を提案した．しかしながら，サーバ側の動作は単純にリクエストを処理し返すのみで，サーバ側の内部の動作を表現していない．本研究では，サーバ側のデータベースアクセスによって不具合が検出されることを示した．

また，Liu ら [16] は原子性の違反検出および修正を自動的に行う手法を提案した．しかし，Web アプリケーションを対象とはしていない．

第7章　今後の課題

本章では本研究における今後の課題について述べる．

7.1　複雑なサーバ動作への対応について

提案した手法ではごく単純な資源のリクエストのみを記述したもので，一般的な RIA に適用可能とは言い難い．しかし，資源の競合の検証を手軽にモデル化できるという利点がある．

7.2　フレームワークとの連携について

Struts フレームワークを用いることにより，Struts アプリケーションの制御依存グラフであるページ遷移モデルとデータ依存グラフであるデータフローモデルの抽出が可能である [22].

第8章　おわりに

本研究では，User，Browser，Server に加え，資源をモデル化し，これらの間の振舞いを定義することで，非同期通信によって生じる資源の競合の検出が SPIN によって可能であることを示し，特に，資源共有のみを検証とする場合に，設計者が検証を容易に行える手法を提案した．資源の競合の検出が出来ることは，Ajax アプリケーションの品質の向上および再実装が減ることが予想される．

提案手法では UI 部分を crawljax によるクロールによってモデル化し，Server 部分をソースコードの静的解析によってモデル化した．この手法では UI 部分においては crawljax によるクロールに修正が必要なければ非同期通信部分の記述を除き自動で変換が行える．Server 部分においては条件分岐を手作業で書くという工程を踏まなければすべて自動で変換が行える．変換のほとんどを自動で行えることは，ソースコードからのモデル化の誤りを防ぐことができ，また，短期間で検証を行うことができるというメリットがある．本手法では導入の敷居が高かった Promela の記述を自動化することにより技術者に専門的な知識がいらなくなるという利点がある．

評価では図書館管理システムおよび todo リスト管理システムに手法を適用した．それぞれでソースコードから UI 部分・Server 部分の変換を行い，検証が可能であることを示した．

第9章　謝辞

電気通信大学大学院情報システム学研究科の大須賀 昭彦教授におかれましては，物事の本質を考え続けることの大切さや挑戦の機会を与えていただきました．厚く御礼申し上げます．

電気通信大学大学院情報システム学研究科の田原 康之准教授におかれましては，諦めそうになる私に「頑張って欲しい」との激励や，毎週のゼミや，ご多忙の中でも常に丁寧なご指導・ご指摘をいただきました．毎週のゼミだけではなく，訪問した時にはこころよく迎え入れていただいたことや，夜遅くのメールにも素早い対応をいただきました．厚く御礼申し上げます．

電気通信大学大学院情報システム学研究科の岡本敏雄教授におかれましては，適切かつ丁寧なご指導を頂きました．御礼申し上げます．

大阪大学大学院情報科学研究科情報システム工学専攻ディペンダビリティ工学講座の中川博之准教授におかれましては，研究のみならず様々な面でサポートして頂きました．海外出張でお忙しい中でも，メールによるご指導をいただきました，お礼申し上げます．

電気通信大学大学院情報システム学研究科の清雄一助教におかれましては適切かつ丁寧なご指導をいただきました．御礼申し上げます．

電気通信大学大学院情報システム学研究科の栗原聡教授におかれましては適切かつ丁寧なご指導をいただきました．御礼申し上げます．

大須賀・田原研究室の皆様，国立情報学研究所・東京大学の本位田研究室の皆様，早稲田大学の深澤研究室の皆様に感謝の意を表します．

最後に，本研究を進めるにあたり，家族を始め，多方面から支えてくださった皆様にもう一度，心からお礼を申し上げます．

参考文献

[1] F. Ricca and P. Tonella, Analysis and Testing of Web Applications, Proc. of ICSE'2001, International Conference on Software Engineering, pp. 25-34, 2001

[2] A. Marchetto,P. Tonella,F. Ricca,State-Based Testing of Ajax Web Applications,Proc. of ICST '2008,International Conference on Software Testing, Verification, and Validation,pp. 121-130,2008

[3] S. Artzi,A. Kiezun,J. Dolby,F. Tip,D. Dig,A. Paradkar,M. D. Ernst,Finding bugs in dynamic web applications,ISSTA'2008,international symposium on Software testing and analysis,pp. 261-272 ,2008

[4] S. Artzi,J. Dolby,S. H. Jensen,A. Moller,F. Tip,A framework for automated testing of javascript web applications,ICSE'2011,International Conference on Software Engineering,pp. 571-580 ,2011

[5] O. Tkachuk,S. Rajan,Automated driver generation for analysis of web applications,FASE'2011/ETAPS'2011, international conference on Fundamental approaches to software engineering: part of the joint European conferences on theory and practice of software,pp. 326-340 ,2011

[6] S. Halle,T, Ettema,C. Bunch,T. Bultan,Eliminating Navigation Errors in Web Applications via Model Checking and Runtime Enforcement of Navigation State Machines,ASE '2010, pp. 235-244 ,2010

[7] P. Fraternali,S. Comai,A. Bozzon,G. T. Carughi,Engineering rich internet applications with a model-driven approach,TWEB'2010, Volume 4 Issue 2,Article No. 7,2010

[8] Ali Mesbah and Arie van Deursen,Invariant-based automatic testing of AJAX user interfaces,ICSE '2009,,pp. 210-220,2009

[9] 玉田和洋, 中川博之, 中山健, 田原康之, 大須賀昭彦, モデル検査による ajax アプリケーション検証のためのモデルの提案, ソフトウェア工学の基礎 XV, 2009

[10] 本間圭, 高橋薫, 和泉諭, 阿部雄貴, 富樫敦, 変数を用いた Web アプリケーションのモデル化と形式的手法による検査, 電子情報通信学会ソフトウェアインタプライズモデリング研究会, 信学技報, Vol.109, No.298,2009

[11] Zheng, Yunhui and Bao, Tao and Zhang, Xiangyu,Statically locating web application bugs caused by asynchronous calls,WWW '2011,pp.805-814,2011

[12] J.J. Garrett, et al. Ajax: A new approach to web applications, adaptive path, Vol.18, 2005

[13] 吉岡 信和, 青木 利晃, 田原 康之,SPIN による設計モデル検証—モデル検査の実践ソフトウェア検証 (トップエスイー実践講座), 近代科学社, 2008

[14] G.J.Holzmann,The Spin Model Checker-Primer and Reference Manual, AddisonWesley, 2004

[15] W. E. McUmber and B. H. C. Cheng, A General Framework for Formalizing UML with Formal Languages, ICSE 2001, pp. 433-442

[16] Peng Liu, Charles Zhang, Axis: Automatically Fixing Atomicity Violations through Solving Control Constraints, ICSE2012,pp.299-309

[17] HiRDB Version 8 UAP 開発ガイド, http://www.hitachi.co.jp/Prod/comp/soft1/manual/pc/d635640/W3560001.HTM (2013 年 1 月 18 日)

[18] 崔銀惠, 画面遷移仕様のモデル検査, コンピュータソフトウェア, Vol.22, No.3, pp.146–153, 2005

[19] 藤原貴之，岡野浩三，楠本真二，SPIN による Struts アプリケーションの動作検証を目的としたモデル生成手法の提案，電子情報通信学会技術研究報告. SS, ソフトウェアサイエンス，Vol.105，No.491，pp.73–78，2005

[20] 大須賀隆彦，UML モデルからの変換による Web アプリケーションの形式検証に関する研究，早稲田大学基幹理工学研究科　情報理工学専攻　深澤研究室　修士論文，平成 23 年度

[21] 枝川拓人，静的解析による Web アプリケーションからのファンクションポイント自動計測手法の改善とその実験的評価，大阪大学　大学院情報科学研究科　コンピュータサイエンス専攻，平成 21 年

[22] 金子伸幸，桑原寛明，山本晋一郎，阿草清滋，StrutsLint：Web アプリケーションコーディングチェッカ，コンピュータソフトウェア，Vol.26，No.3，pp.34–43，2009

[23] XMLHttpRequest
http://www.w3.org/TR/2012/NOTE-XMLHttpRequest1-20120117/（2013 年 1 月 22 日）

[24] 中所武司，藤原克哉，Information&Computing-104 Java による Web アプリケーション入門-サーブレット・JSP・Struts-，サイエンス社，2005

[25] 本教材について http://si.comp.ae.keio.ac.jp/web_app_dev_material/index.html2014 年 7 月 12 日閲覧

付 録A　PROMELA記述

ソースコード A.1: 図書館管理システムの Promela 記述

```
1  mtype = {get,release};
2  mtype = {index, add, addForm, list, searchForm, listTakingout, Logout, loginForm, edit
       , delete, takeout, search, return1};
3
4  mtype = {input, output};
5  mtype = {XHRack};
6
7  chan reply [19] = [0] of {mtype};
8  chan Handler =[0]of{mtype, byte};
9  chan evHandler=[0]of{mtype, byte};
10 chan request=[0]of{mtype, chan};
11 chan Operation=[0]of{mtype};
12
13 chan XHR = [0] of {mtype};
14 chan retHandler = [0] of {mtype};
15
16 chan bookcatalog = [0] of {mtype};
17 chan takeoutrecords = [0] of {mtype};
18
19 bool cancel=false;
20 bool data=false;
21
22 mtype = {prepare, confirm};
23 mtype return_step,edit_step, delete_step,add_step,takeout_step;
24
25 mtype = {LOCKED, UNLOCKED};
26 mtype mutex = UNLOCKED
27
28 inline lock(m){
29   d_step{m==UNLOCKED -> m = LOCKED}
30 }
31
32 inline unlock(m){
33   m = UNLOCKED
34 }
35
36
37 inline CallBack(me){
38   run CallBackRes(me);
39 }
40
41 proctype CallBackRes(byte me){
```

```
42     mtype server;
43     reply[me] ? server;
44 }
45
46 inline Ajax(s, me){
47   Handler!s, me -> CallBack(me);
48 }
49
50 inline Sync(s, me){
51   mtype server;
52   run Server();
53   request!s, reply[me];
54   reply[me] ? server;
55 }
56
57 active proctype User(){
58 endUser:
59 /*
60   do
61   ::Operation!input->Operation?output;
62   od
63 */
64  Operation!input->Operation?output;
65   Operation!input->Operation?output;
66  Operation!input->Operation?output;
67   Operation!input->Operation?output;
68  Operation!input->Operation?output;
69   Operation!input->Operation?output;
70  Operation!input->Operation?output;
71   Operation!input->Operation?output;
72  Operation!input->Operation?output;
73   Operation!input->Operation?output;
74
75 }
76
77 active proctype UI(){
78 endUI:
79 indextag:
80 Operation?input->
81       Sync(index, 0);Operation!output;goto endindex1tag;
82 endindex1tag:
83   Operation?input->
84       if
85       ::Sync(addForm ,1);Operation!output;goto endadd1tag;
86       ::Sync(list ,2);Operation!output;goto end_edit_delete_takeout_tag;
87       ::Sync(searchForm ,3);Operation!output;goto endsearchFormtag;
88       ::Sync(listTakingout,4);Operation!output;goto endlistTakingouttag;
89       ::Sync(Logout ,5);Operation!output;goto indextag;
90       ::Sync(loginForm ,6);Operation!output;goto indextag;
91       fi;
92 endadd1tag:
93   Operation?input->add_step=prepare;Sync(add,7);Operation!output;goto
         endaddForm2tag;
94 endaddForm2tag:
```

```
95    Operation?input->add_step=confirm;Sync(add,8);Operation!output;goto indextag;
96  end_edit_delete_takeout_tag:
97    if
98    ::Operation?input->edit_step=input ;Sync(edit,9);Operation!output;goto endedit1tag;
99    ::Operation?input->delete_step=prepare;Sync(delete,10);Operation!output;goto
          enddelete1tag;
100   ::Operation?input->takeout_step=prepare;Sync(takeout,11);Operation!output;goto
          endtakeout1tag;
101   fi;
102
103 endsearchFormtag:
104   Operation?input->Sync(search,12);Operation!output;goto end_edit_delete_takeout_tag
          ;
105
106 endedit1tag:
107   Operation?input->edit_step=prepare;Sync(edit,13);Operation!output;goto
          endedit2tag;
108 endedit2tag:
109   Operation?input->edit_step=confirm;Sync(edit,14);Operation!output;goto indextag;
110
111 enddelete1tag:
112   Operation?input->delete_step=confirm;Sync(delete,15);Operation!output;goto
          indextag;
113
114 endtakeout1tag:
115   Operation?input->takeout_step=confirm;Ajax(takeout,16);Operation!output;goto
          indextag;
116
117 endlistTakingouttag:
118   Operation?input->return_step=prepare;Sync(return1,17);Operation!output;goto
          endreturn1tag;
119 endreturn1tag:
120   Operation?input->return_step=confirm;Sync(return1,18);Operation!output;goto
          indextag;
121
122 }
123
124 active proctype Ajax_Engine(){
125   mtype req;
126   byte me;
127 endAjax:
128   do
129     ::Handler?req, me;run XHR_Obj();evHandler!req, me;
130   od
131 }
132
133 proctype XHR_Obj(){
134   mtype req;
135   byte me;
136   evHandler?req, me;run Server(); request!req, reply[me];
137
138 }
139
140 proctype Server(){
```

```
141 mtype req;
142 chan replyChannel;
143     request?req, replyChannel->
144     if
145     ::(req == index )->replyChannel!req;
146     ::(req == addForm )->replyChannel!req;
147     ::(req == add )-> true ->
148       if
149       ::(add_step == prepare)->skip;replyChannel!req;
150       ::(add_step == confirm)->bookcatalog!get;bookcatalog!release;replyChannel!req;
151       ::else XHR!XHRack;
152       fi;
153     ::(req == list )->bookcatalog!get;bookcatalog!release;replyChannel!req;
154     ::(req == search )->true->
155       if
156       ::bookcatalog!get;bookcatalog!release;replyChannel!req;
157       ::bookcatalog!get;bookcatalog!release;replyChannel!req;
158       fi;
159     ::(req == searchForm )->replyChannel!req;
160     ::(req == listTakingout)->lock(mutex);bookcatalog!get; takeoutrecords!get;
161                                 bookcatalog!release;takeoutrecords!release;unlock(mutex);
                                        replyChannel!req;
162     ::(req == Logout )->replyChannel!req;
163     ::(req == loginForm )->replyChannel!req;
164     ::(req == edit )-> true ->
165       if
166       ::(edit_step == input )->bookcatalog!get;bookcatalog!release;replyChannel!req;
167       ::else if
168               ::(edit_step == confirm)->lock(mutex);bookcatalog!get; takeoutrecords!get
                        ;
169                                             bookcatalog!release;takeoutrecords!release;unlock
                                                (mutex);replyChannel!req;
170               ::else replyChannel!req;
171               fi;
172       fi;
173     ::(req == delete )->bookcatalog!get;bookcatalog!release;replyChannel!req;
174     ::(req == takeout )->true->
175       if
176       ::(takeout_step == prepare)->bookcatalog!get;bookcatalog!release;replyChannel!
              req;
177       ::else lock(mutex);takeoutrecords!get; takeoutrecords!release;
178               takeoutrecords!get; takeoutrecords!release;
179               bookcatalog!get;bookcatalog!release;
180               bookcatalog!get;bookcatalog!release;unlock(mutex);replyChannel!req;
181       fi;
182     ::(req == return1 )->true->
183       if
184       ::(return_step == prepare)->bookcatalog!get;bookcatalog!release;replyChannel!
              req;
185       ::(return_step == confirm)->lock(mutex);takeoutrecords!get; bookcatalog!get;
186                                     takeoutrecords!release;bookcatalog!release;unlock(
                                          mutex);replyChannel!req;
187       fi;
188     fi;
```

```
189 }
190
191 active proctype db1(){
192 endDB:
193   do
194     :: bookcatalog ? get −> bookcatalog ? release;
195   od
196 }
197
198 active proctype db2(){
199 endDB:
200   do
201     :: takeoutrecords ?get −> takeoutrecords ? release;
202   od
203 }
```

ソースコード A.2: Struts2todo の Promela 記述

```
1  mtype = {get,release};
2  mtype = {login, list, search, finish, add, edit, delete, edit_action, delete_action,
       add_action};
3
4  mtype = {input, output};
5  mtype = {XHRack};
6
7  chan reply [19] = [0] of {mtype};
8  chan Handler =[0]of{mtype, byte};
9  chan evHandler=[0]of{mtype, byte};
10 chan request=[0]of{mtype, chan};
11 chan Operation=[0]of{mtype};
12
13 chan XHR = [0] of {mtype};
14 chan retHandler = [0] of {mtype};
15
16 chan todo_user = [0] of {mtype};
17 chan todo_item = [0] of {mtype};
18
19 bool cancel=false;
20 bool data=false;
21
22 mtype = {prepare, confirm};
23 mtype return_step,edit_step, delete_step,add_step,takeout_step;
24
25 mtype = {LOCKED, UNLOCKED};
26 mtype mutex = UNLOCKED
27
28 inline lock(m){
29   d_step{m==UNLOCKED −> m = LOCKED}
30 }
31
32 inline unlock(m){
33   m = UNLOCKED
34 }
35
```

```
36
37 inline CallBack(me){
38   run CallBackRes(me);
39 }
40
41 proctype CallBackRes(byte me){
42     mtype server;
43     reply[me] ? server;
44 }
45
46 inline Ajax(s, me){
47   Handler!s, me -> CallBack(me);
48 }
49
50 inline Sync(s, me){
51   mtype server;
52   run Server();
53   request!s, reply[me];
54   reply[me] ? server;
55 }
56
57 active proctype User(){
58 endUser:
59 /*
60   do
61   ::Operation!input->Operation?output;
62   od
63 */
64  Operation!input->Operation?output;
65   Operation!input->Operation?output;
66  Operation!input->Operation?output;
67   Operation!input->Operation?output;
68  Operation!input->Operation?output;
69   Operation!input->Operation?output;
70  Operation!input->Operation?output;
71   Operation!input->Operation?output;
72  Operation!input->Operation?output;
73   Operation!input->Operation?output;
74
75 }
76
77 active proctype UI(){
78 endUI:
79 endindex1tag:
80
81 Operation?input->
82   Sync(login, 0 ); Operation!output;goto endlogintag;
83
84 endlogintag:
85 Operation?input->
86   Sync(list, 0 ); Operation!output;goto endlisttag;
87
88 endlisttag:
89   Operation?input->
```

```
90      if
91       ::Sync(search, 0); Operation!output;
92         goto endindex1tag;
93      ::Sync(finish, 1); Operation!output;
94           goto endindex1tag;
95      ::Sync(add, 2); Operation!output;
96            goto endaddtag;
97      ::Sync(edit, 3); Operation!output;
98           goto endedittag;
99      ::Sync(delete, 4); Operation!output;
100          goto enddeletetag;
101 fi;
102
103 endedittag:
104   Operation?input->Sync(edit_action, 5); Operation!output;
105          goto endindex1tag;
106
107 enddeletetag:
108   Operation?input->Sync(delete_action, 6); Operation!output;
109          goto endindex1tag;
110
111 endaddtag:
112   Operation?input->Sync(add_action, 7); Operation!output;
113          goto endindex1tag;
114 }
115
116 active proctype Ajax_Engine(){
117   mtype req;
118   byte me;
119 endAjax:
120   do
121     ::Handler?req, me;run XHR_Obj();evHandler!req, me;
122   od
123 }
124
125 proctype XHR_Obj(){
126   mtype req;
127   byte me;
128   evHandler?req, me;run Server(); request!req, reply[me];
129 }
130
131 proctype Server(){
132 mtype req;
133 chan replyChannel;
134      request?req, replyChannel->
135      if
136      ::(req == login )->lock(mutex);todo_user!get;todo_user!release;unlock(mutex);
             replyChannel!req;
137      ::(req == list )->lock(mutex);todo_item!get;todo_item!release;unlock(mutex);
             replyChannel!req;
138      ::(req == add )->lock(mutex);todo_user!get;todo_user!release;unlock(mutex);
             replyChannel!req;
139      ::(req == add_action )->lock(mutex);todo_user!get;todo_item!get; todo_user!release;
             todo_item!release;unlock(mutex); replyChannel!req;
```

```
140     ::(req == edit )->lock(mutex);todo_item!get;todo_item!release;unlock(mutex);
            replyChannel!req;
141     ::(req == edit_action )->lock(mutex);todo_item!get;todo_item!release;todo_item!get;
            todo_item!release;unlock(mutex); replyChannel!req;
142     ::(req == delete )->lock(mutex);todo_item!get;todo_item!release;unlock(mutex);
            replyChannel!req;
143     ::(req == delete_action)->lock(mutex);todo_item!get;todo_item!release;todo_item!get
            ;todo_item!release;unlock(mutex); replyChannel!req;
144     ::(req == finish )->lock(mutex);todo_item!get;todo_item!release;todo_item!get;
            todo_item!release;unlock(mutex); replyChannel!req;
145     ::(req == search )->lock(mutex);todo_item!get;todo_item!release;unlock(mutex);
            replyChannel!req;
146     fi;
147 }
148
149 active proctype db1(){
150 endDB:
151   do
152     :: todo_user ? get -> todo_user ? release;
153   od
154 }
155
156 active proctype db2(){
157 endDB:
158   do
159     :: todo_item ?get -> todo_item ? release;
160   od
161 }
```