Vol.2012-SE-176 No.10
Vol.2012-EMB-25 No.10
2012/5/21

UML アクティビティ図から SPIN モデル検査用コードの自動生成における並列処理拡張と Ajax アプリケーション設計への適用

山田 豊[†]　　和﨑克己[††]

筆者らが，以前に試作した UML アクティビティ図から PROMELA で記述された検証用モデルに自動変換する変換器において，UML アクティビティ図のフォークノード・マージノード等の並列処理記述を扱えるように拡張し，非同期通信により並列処理を伴う Ajax アプリケーションのモデルも扱うことができるようにした．提案手法の評価のため，Ajax を用いた業務アプリケーションの画面遷移設計に適用実験を行った．

Extension for the Parallel Processing in Automatic Generation from UML Activity Diagram to SPIN Model Checking Code and its Application to Ajax Application Design

YUTAKA YAMADA[†]　KATSUMI WASAKI[††]

We extended the converter that converts automatically the UML activity diagram into the SPIN model checking code PROMELA, and enabled it to treat parallel processing description, such as a fork node and merge node of an UML activity diagram. The model of Ajax application with parallel processing by asynchronous communication can also be treated. We modeled the screen transition design of an Ajax application as an evaluation of the proposed method.

1. はじめに

ソフトウェアの上流設計仕様を形式言語でモデル化し，検査ツールを用いて動作検証，あるいは，反例を検出するというアプローチが注目を集めている. モデル検査により，設計段階でモデルの欠陥を検出することにより，ソフトウェアの信頼性・安全性の向上が期待できる．

ソフトウェアの分析・設計には，UML(統一モデリング言語)[1]を用いることが多い．UML はダイアグラムを多用した, モデルを視覚的にとらえるための表記法であるため，UML でモデリングしたモデルを, そのまま検査ツールで検証することはできない. UML で記述した分析・設計用のモデルとは別に，形式言語で記述した検証用のモデルを用意する必要がある．ソフトウェア開発の上流工程において，仕様の見直しは頻繁に行われるので，そのたびに UML モデルから検証用モデルを手動で，変換・導出することは非常なコストと時間がかかるため，実開発の場面では現実的でない．筆者らは，UML アクティビティ図に着目し，アクティビティ図をモデル検査ツール SPIN[2][3]の仕様記述言語である PROMELA に自動変換する手法を提案した[4].

従来のオンラインショッピングやオンラインバンキング等に代表されるインターネット上のアプリケーションのみならず，企業の業務アプリケーション等様々なシステムが Web アプリケーションで構築されており, Ajax 技術を用いたデスクトップアプリケーションと同等のユーザーインタフェースを持つ Web アプリケーションも広く普及している．これに伴い，Web アプリケーションの信頼性・安全性に対する要求も益々高まっている．

本報告では，以前に試作した UML アクティビティ図から PROMELA で記述された検証用モデルに自動変換する変換器を拡張し，UML アクティビティ図のフォークノード・マージノード等の並列処理記述を変換可能とした点について説明する．これにより，非同期通信により並列処理を伴う Ajax アプリケーションのモデルも扱うことができる．さらに，並列処理には欠かせない同期機構の扱いについても検討したので，合わせて報告する．最後に，提案手法の評価のため，Ajax を用いた業務アプリケーションの画面遷移設計に適用実験を行ったので，その結果について述べる．

2. SPIN モデル検査ツール

SPIN は，G.J.Holzmann が中心になって開発，公開しているモデル検査ツールである．並列システムの振る舞い仕様に着目するモデル検査法に基づいた自動検証ツールの一つで，産業界でも多数の適用事例が報告されている．

2.1 SPIN モデル検査の概要

図 1 に SPIN ツールによるモデル検査の流れを示す. SPIN は PROMELA 言語で記述されたモデルから，無限長の語を扱う有限オートマトン(Buchi オートマトン)を生成し，検証器と呼ばれる C 言語で書かれたプログラムを生成する．C コンパイラを使ってコンパイルした後，検証器を実行すると，検証器はあらゆる状態遷移を生成し，モデルが所定の

[†] (株)プラグマティック・テクノロジーズ
Pragmatic Technologies Co. Ltd.
[††] 信州大学工学部情報工学科
Department of Computer Science & Engineering. Faculty of Engineering, Shinshu University.

Vol.2012-SE-176 No.10
Vol.2012-EMB-25 No.10
2012/5/21

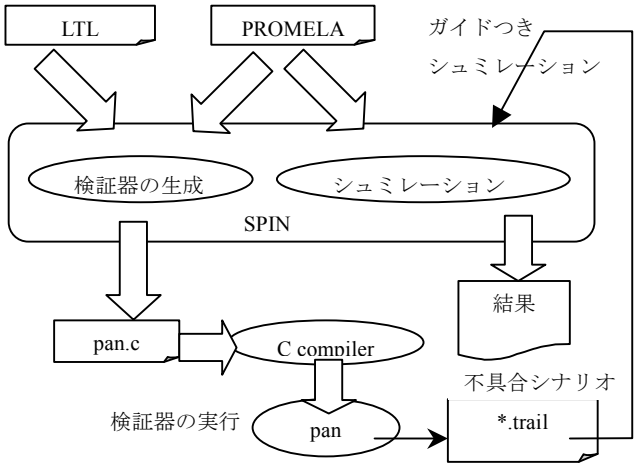

図 1　SPIN モデル検査の流れ

Figure 1　Process of SPIN model checking.

```
if
  :: ガード 1 －＞ 文の列 1
  :: ガード 2 －＞ 文の列 2
        …
  :: ガード N －＞ 文の列 N
fi;
```

図 2　ガードコマンド

Figure 2　Guard commands.

性質を満たしているかを検証する．

SPIN では，モデルの正当性を表現するために，表明(assert)と線形時相論理(LTL)を用いることができる．表明はプロセスの任意の箇所に assert(条件式)文を記述することで，その時点において，モデルが取ってはならない反例を示すことができる．線形時相論理は，命題論理式に，時の概念を表現できる時相演算子を加えた論理体系である．これにより，PROMELA で記述したプロセスが時系列に沿って成り立つべき性質を論理式の形で指定できる．実行中の検証器は，オートマトンが取り得る全ての状態を網羅的に探査し，正当性の要求条件を満たしているか否かを検証する．満たしていない場合，初期状態からの反例パスを求めた後，出力し，設計者へフィードバックする．

2.2 PROMELA 言語

PROMELA は対象システムを，チャネルを用いて通信するプロセスの集まりとして表現する．並列実行されるプロセスは，基本的にはオートマトンの考え方で，対象システムの振る舞い仕様を表現する．プロセスの振る舞い記述の基本的な考え方は，ガードコマンド(Guarded Commands)である．図 2 にガードコマンドの記述例を示す．

ガード条件が成り立つと，アクションを構成する文の列が実行される．ガードが常に真となるように(あるいはガードを省略)記述すると，検証において，取りうる全ての選択の中から，非決定的に選択されるようになる．

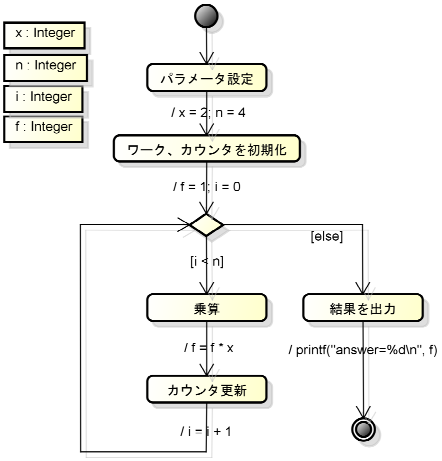

図 3　アクティビティ図の記述例

Figure 3　Drawing example of activity diagram.

3. UML アクティビティ図から PROMELA への自動変換

UML アクティビティ図は，UML2.0[1]で独立して定義された振る舞い図の一つで，処理の実行順序や条件，制御などを表現できる．図 3 にアクティビティ図の例を示す．このアクティビティ図は，単純なべき乗を計算するモデルである．アクティビティ図では，モデリング対象が行う個々の活動をアクションとして記述し，アクション間の順序関係を制御フローで連結する．条件に応じてフローが分岐する場合は，ディシジョンノードを使う．ディシジョンノードの各分岐には，ガードと呼ばれる要素に分岐条件を記述する．

3.1 自動変換の手法

UML アクティビティ図では，モデルを視覚的に表現するため多くの図要素が定義されているが，アクティビティ図から PROMELA モデルの導出を考えたとき，それら全てを対象にする必要はない．PROMELA で表現できることをアクティビティ図から抽出して，検証用モデルを作成することになる．これは，人手で行う場合でも同様である．

PROMELA の振る舞い記述の基本は，ガードコマンドであり，アクティビティ図では，制御フローがそれに該当する．制御フローには，ガードとアクションの記述が出来るので，その流れを PROMELA 上に再現すれば良い．

PROMELA 記述は，あらかじめテンプレートを用意しておき，アクティビティ図を上から辿り，出現するノードに対応するテンプレートを組み合わせていくことで変換を行う．図 4 に UML アクティビティ図の要素と，PROMELA 記述の対応例を示す．

これらの変換が基本になるが，逆に PROMELA の表現でアクティビティ図の要素では記述できないものもある．

Vol.2012-SE-176 No.10
Vol.2012-EMB-25 No.10
2012/5/21

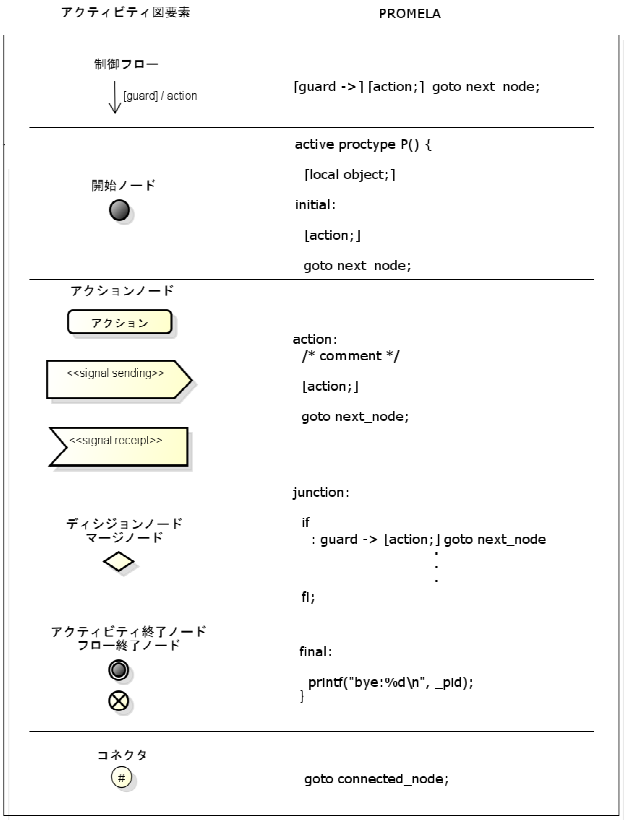

図 4　アクティビティ図要素と PROMELA の対応例
Figure 4　The map of an activity diagram element and PROMELA

表 1　PROMELA 変換用に定義したタグ付き値
Table 1　Tagged value defined for PROMELA conversion.

アクティビティ図全体に付与するタグ	
Spin.process	プロセス名(省略時は P1,P2,…)
Spin.number_of_processes	プロセス数(省略時は 1)
Spin.ltl_spec_pattern	仕様パターン(省略可)
アクションに付与するタグ	
Spin.label	プロセス中のラベル(省略可能). LTL の記述に利用することを想定.
Spin.pre_condition	事前条件(省略化)
Spin.post_condition	事後条件(省略化)
Spin.semaphore	P または Pn (n=1,2,3...) V または Vn (n=1,2,3...)
シグナル送信アクションに付与するタグ	
Spin.channel_input	チャネルに送信するメッセージ
Spin.channel	チャネル名(省略時は ch0,ch1,…)
Spin.channel_buffer_size	バッファサイズ(省略時は同期通信)
Spin.channel_message_type	メッセージタイプ(省略時は, spin.channel_input の型で生成)
イベント受信アクションに付与するタグ	
Spin.channel_output	チャネルから受信したメッセージの出力先

それらについては，UML の「タグ付き値」を利用するようにした．「タグ付き値」は，UML の拡張メカニズムの一つで，モデル要素に任意の情報を{tag=value}の形式で定義することができる．表 1 に，PROMELA 変換用に定義した「タグ付き値」の一覧を示す．定義された「タグ付き値」が付与されている場合は，対応する PROMELA のテンプレートを組み合わせて，変換を行う．

アクティビティ図は，多様な処理の流れを表現できるが，アクションの記述やその抽象度の扱いなどに厳密な決まりはないため，特に自然言語を用いた場合，その解釈次第で実装が異なるという曖昧さを含んでおり，これが自動変換時の難しさの一因となっている．UML では，この仕様の曖昧さを軽減するために，オブジェクト制約言語(Object Constraint Language)[5]を用意している．ダイアグラムでは表現しきれない制約条件などを記述できるため，振る舞いについての外部仕様を明確化し，開発者間で，齟齬のないように設計を進める上で有効である．しかしながら，OCL は，その記述の複雑性などから，現時点で広く普及しているとは言い難い．

本研究では，アクティビティ図を SPIN の仕様記述言語 PROMELA に自動変換し，上流工程での仕様の検証を効率化することを目的としているため，アクションやガードなどの記述に，PROMELA の構文を直接用いることで，形式化と自動変換の簡素化を目標とすることにした．

PROMELA は，変数の代入，四則演算や論理演算などの構文が C 言語に似ているため，ソフトウェアの設計者にとって構文の再習得の必要がなく，自然な記述ができる．

3.2 自動変換の手順

図 5 に，UML から PROMELA モデルへの自動変換の流れを示す．変換器は PHP 言語を用いて実装した．

UML 記述ツールは，株式会社チェンジビジョンの astah* Professional[6]を使用した．ツールの機能によって，記述した UML を XML 形式のファイルとして出力する．XML ファイルには，アクティビティ図の構成要素が，タグとして格納されている[7].

変換器は，XML ファイルを受け取ると，XML パーサーにより，メモリ上に XML の木構造を展開する．次にアクティビティパーサーがアクティビティ図の要素を抜き出し，PROMELA フォーマッターが，各要素に対応するテンプレートを使って PROMELA の形式で出力する．

出力された PROMELA ファイルを SPIN に入力してモデルの検証を行う．

Vol.2012-SE-176 No.10
Vol.2012-EMB-25 No.10
2012/5/21

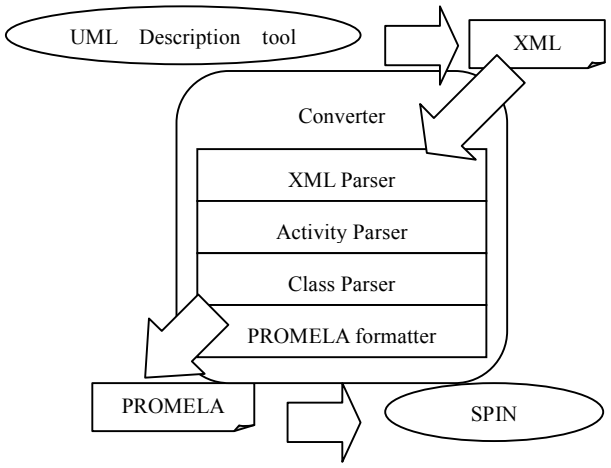

図 5　自動変換の流れ

Figure 5　Flow of automatic conversion.

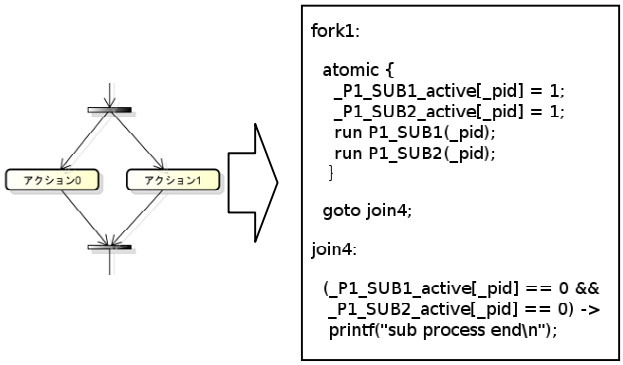

図 6　フォークノード・マージノードの変換例

Figure 6　Example of converting fork node and marge node.

3.3 フォークノード・マージノード並列処理記述の変換

Ajax アプリケーションで見られる非同期通信による並列処理を扱えるようにする為，アクティビティ図のフォークノード・マージノードによる並列処理記述も変換できるように拡張した．以下に，フォークノード・マージノードの変換手順を示す．

(1) フォークノード・ジョインノードで挟まれた処理を抜き出し，別プロセスとして，PROMELA 上に出力する．ジョインノードがない場合は，終了ノードまでを抜き出す．
(2) フォークノードでは，抜き出したプロセスを run() で起動する．
(3) ジョインノードで，同期を取る場合は，子プロセスの終了をフラグ(大域変数)管理する．
(4) 親プロセスと子プロセスの両方で使用している変数は大域変数として変換する．

図 6 にフォークノード・マージノードの変換例を示す．

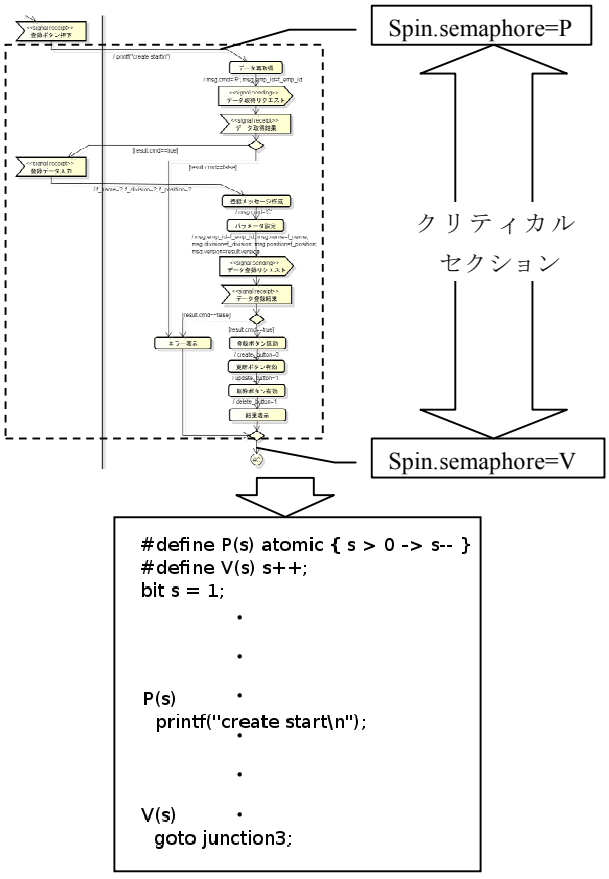

図 7　セマフォア設定と PROMELA 出力例

Figure 7　Example of setting semaphore and POMELA.

3.4 同期機構セマフォアの導入

並列処理を取り扱うため，モデリングにおいても資源の同時アクセスについて考慮することが必要である．今回は基本的な同期機構であるダイクストラのセマフォアを導入した．UML アクティビティ図の要素に「タグ付き値」で，クリティカルセクションを指定する．図 7 にセマフォア設定と PROMELA の出力例を示す．

4. Ajax アプリケーションへの適用実験

提案手法の評価のため Ajax を用いた社員管理システムの画面遷移設計を取り上げ，試作した変換器を用いて適用実験を行った．

4.1 社員管理システムの仕様

今回，事例で扱う「社員管理システム」は，社員情報の登録・検索・変更・削除を行うアプリケーションである．図 8 に画面の例を示す．このシステムは，ブラウザ上で動作するクライアントサイドと，データを保持・管理するサーバーサイドで構成される．

このシステムでは，社員番号・氏名・部署・役職等のデータを取り扱うが，社員番号は一意でなくてはならない．そこで，社員番号を入力した時点で，Ajax の非同期通信を使いサーバーからデータを取得して，登録が無ければ，「新規登録」ボタンを，既に登録があれば，「変更」「削除」

Vol.2012-SE-176 No.10
Vol.2012-EMB-25 No.10
2012/5/21

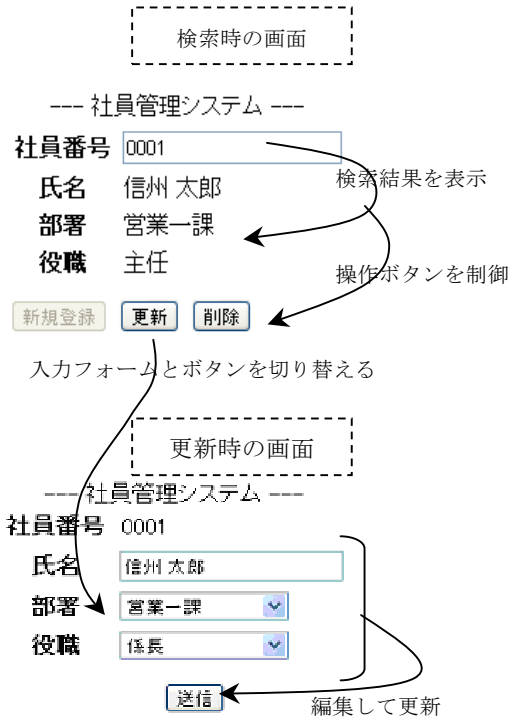

図 8　社員管理システム

Figure 8　Employee management system.

ボタンを押せるようにする．状況に応じて，有効な操作ボタンを表示することで，オペレーションミスを防ぐ仕様とした．

また，このシステムのユーザーは，人事担当者で，数名が同時に利用する可能性がある．そのため，別の作業者によるデータの上書きや，編集中の削除などの不整合が起きないよう考慮する必要がある．

4.2 同時アクセスによる不整合の検出

複数ブラウザにより，同一社員番号で，新規登録，更新，削除の操作が重なった場合の不整合を検出するために，更新バージョンを管理するようにした．

図 9 に同時アクセスによる不整合検出の手順を示す．サーバーサイドでは，新規登録，更新，削除の処理のたびに，version をカウントアップする．クライアントサイドは，新規登録，更新，削除の操作に入る直前で，その時点の version を取得し，更新系のリクエストをする際に，この version も一緒に送信する．サーバーサイドでは送られてきた version と保持している version を比較して，古ければ，そのクライアントの更新操作の間に，別のクライアントにより変更されたことが検出できる．

4.3 社員管理システムのモデル化

4.1 で説明した仕様のモデルをアクティビティ図で作成した．図 10 に社員管理システムのクライアントサイドのモデルを，図 11 にサーバーサイドのモデルを示す．クライアントサイドのアクティビティ図は，レーンを２つに分け，ユーザー操作と，ブラウザの動き(Javascript)が区別できるように記述した．

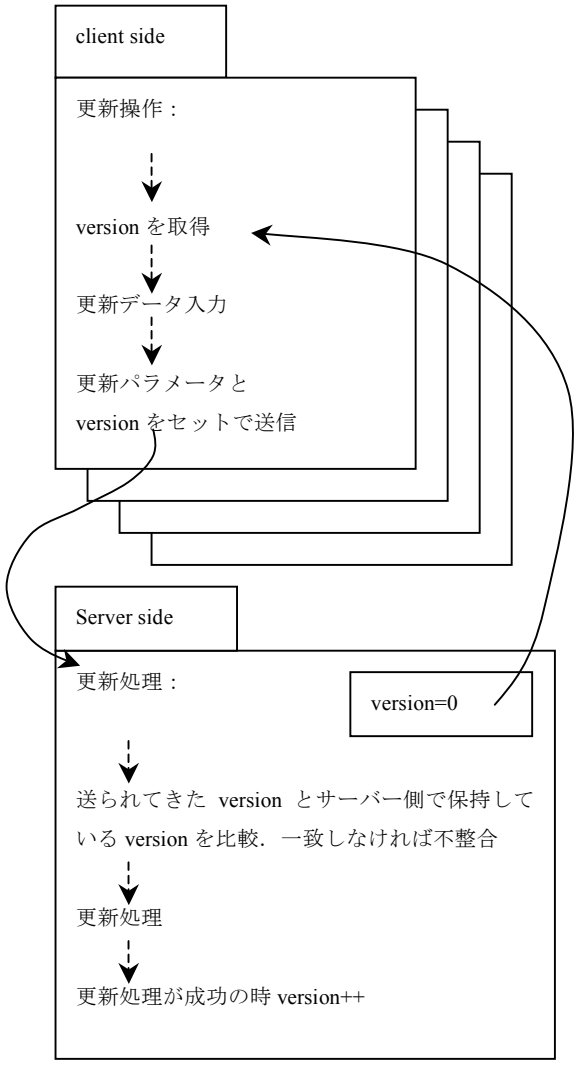

図 9　同時アクセスによる不整合の検出

Figure 9　Detection of the error by concurrent access.

クライアントサイドのモデルとサーバーサイドのモデルは，チャネル通信を使って情報をやり取りしながら，処理を進めていく．

図 10 の点線で囲まれた部分が Ajax の非同期通信をモデル化した部分である．データ取得のリクエストをサーバーサイドに送信し，非完了待ちで，すぐ制御が戻る様子を表現している．フォークノードで制御が二つに分かれ，一方はメインに戻り，もう一方はサーバーからのデータ受信を待つ．受信処理が終わるとブラウザ上のデータを更新して，フローは終了する．

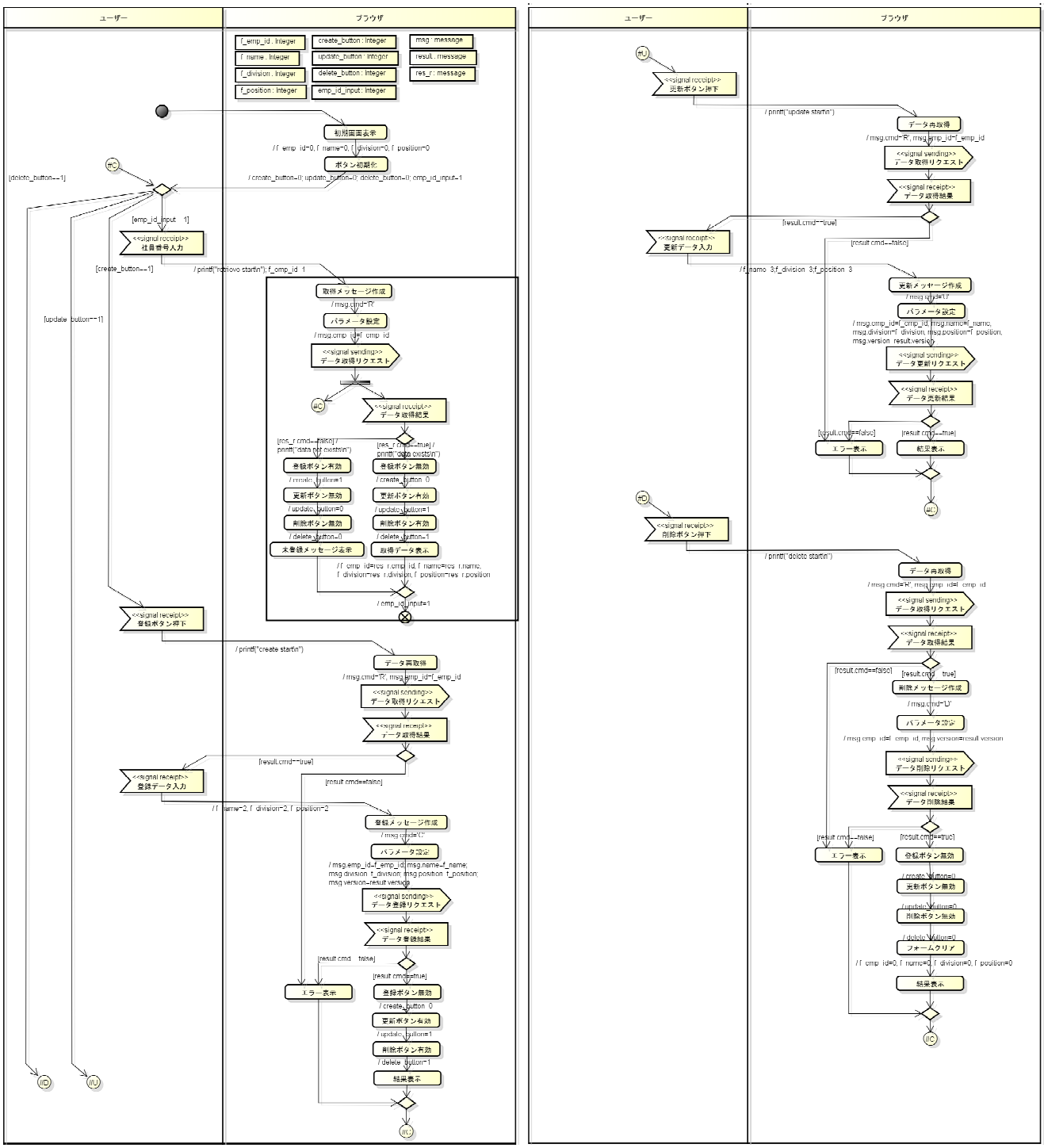

図 10　クライアントサイドのモデル

Figure 10　The model of client side.

複数ブラウザによる同時更新の対策として，更新系の操作の入り口と出口の間をクリティカルセクションとし，セマフォアの設定を行った．クライアントシステム全体で高々1 つのプロセスだけが実行可能な部分となる．

実際のシステムでは，個々のクライアントは，別々のコンピュータ上で動作するため，サーバーサイドを経由してデータベースのテーブルをロックするような実装を想定して抽象化している．

4.4 モデル検査の検証項目

今回の実験では，モデル検査の検証項目を以下の 3 項目とした．

1. 取得データの状況に応じて，次に実行可能な操作を決めるので，デッドロックなどの状態異常がないか．

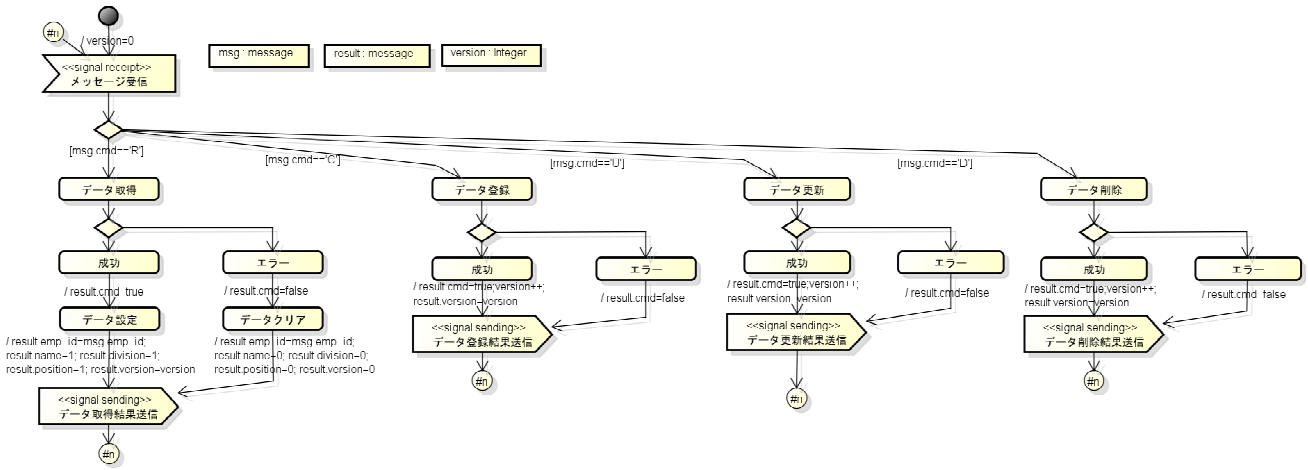

図 11　サーバーサイドのモデル

Figure 11　The model of server side.

```
Depth= 1016396 States=    1e+06 Transitions= 1.27e+06 Memory=
 179.384        t=  2.55 R=   4e+05
pan: assertion violated (msg.emp_id!=0) (at depth 1614020)
pan: wrote employee10.pml.trail

(Spin Version 5.2.2 -- 7 September 2009)
Warning: Search not completed
        + Partial Order Reduction

Bit statespace search for:
        never claim             - (none specified)
        assertion violations    +
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      +

State-vector 460 byte, depth reached 1614024, errors: 1
 1915758 states, stored
  668285 states, matched
 2584043 transitions (= stored+matched)
   44269 atomic steps
```

図 12　SPIN による検証結果

Figure 12　Results of checking by SPIN.

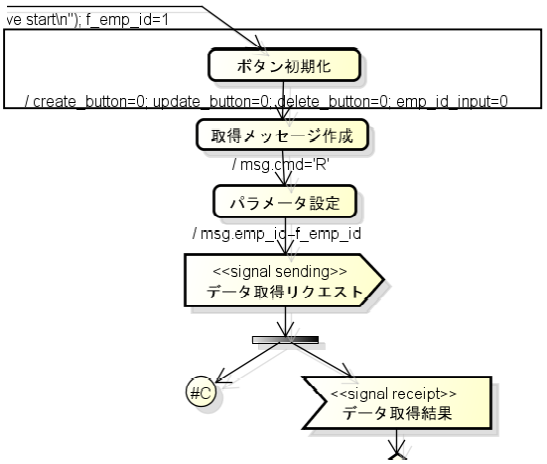

図 13　モデル employee2　(抜粋)

Figure 13　The model of employee2 (excerpt).

2. Ajax の非同期通信による並列動作により，変数の上書きによる不具合が起きないか．
3. 複数ブラウザにより，登録，更新，削除の操作が重なっても不整合は起きないないか．

検証項目 1 は, SPIN の基本的な検査(到達性解析)により，デッドロック，ライブロックなどの状態がないか検査できる．検証項目 2 は，クライアントサイドモデルにおいて，Ajax の非同期通信処理で更新される入力フォームの値を検証する．この値は，更新系操作のサーバーサイドへのリクエストパラメータに利用されるため，リクエスト送信の事前条件として不正な値でないかを検査する．検証項目 3 は，サーバーサイドにおいて，更新バージョンを管理し，新規登録・更新・削除処理の事前条件として，更新バージョンの不一致がないことを検査する．事前条件は，spin.pre_condition タグに記述し，その内容は，PROMELA では表明(assert 文)に変換される．

4.5 検証結果 1　(Ajax 非同期通信によるデータの上書き)

アクティビティ図のモデルから変換器により，PROMELA モデルを生成し, SPIN を使って検証したところ，図 12 の様なエラーが検出された．

出力された Trail から不具合シナリオによるシュミレーションで調べたところ，検証項目 2 のパラメータ異常を検出したことが分かった．削除操作において，ユーザーが入力フォームに入力した値を，リクエスト用の変数に移している途中で，並列動作しているデータ取得の受信処理が入力フォームの内容を書き換えるため，不具合が起きると考えられた．

そこで，クライアントサイドのモデルを修正し，データ取得リクエストの直前で，一旦，各ボタンの状態を無効にする処理を追加した．図 13 に，アクティビティ図に付加した部分を示す．この修正により，データ取得が完了するまでは，更新系の操作を抑制できる．修正後のモデル employee2 を SPIN で検証したところ，先程のエラーは解消

```
Depth= 1096558 States=    1e+06 Transitions= 1.03e+06 Memory=
  181.768       t=  6.51 R=   2e+05
pan: assertion violated (msg.version==version) (at depth 1177473)
pan: wrote employee2.9.pml.trail

(Spin Version 5.2.2 -- 7 September 2009)
Warning: Search not completed
        + Partial Order Reduction

Bit statespace search for:
        never claim             - (none specified)
        assertion violations    +
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      +

State-vector 684 byte, depth reached 1177481, errors: 1
 1074032 states, stored
   39272 states, matched
 1113304 transitions (= stored+matched)
   26352 atomic steps
```

図 14　SPIN による検証結果 2

Figure 14　Results 2 of checking by SPIN.

された.

4.6 検証結果 2 (複数ブラウザによる更新の競合)

次に，複数の作業者が同時にこのシステム利用した場合を想定して，クライアントサイドのプロセス数を 5 に設定し，再度検証を行った．同時生成プロセス数の指定は spin.number_of_porcesses タグで行う.

この検証結果を図 14 に示す．ここで検出されたエラーは，検証項目 3 の更新バージョンの不一致である．出力された Trail から不具合シナリオによるシュミレーションを実行して調べたところ，図 15 のように，二つのプロセス(pid:2 と 4)の間で，削除操作と更新・新規登録の操作が競合している様子が観察できた.

アクティビティ図を見直してみたところ，新規登録操作と更新操作にはセマフォアの設定により，クリティカルセクションになっていたが，削除操作には設定が抜けていたことが分かった.

削除操作にもセマフォアを設定し，再度，SPIN による検査を行ったところ，今度は，エラーが検出されなくなり，モデルの正当性が検証された.

5. まとめと今後の課題

本報告では，筆者らが以前に試作した UML アクティビティ図から PROMELA で記述された検証用モデルに自動変換する変換器を拡張し，フォークノード・マージノードといった並列処理記述を取り扱う手法を提案した．これにより, 並列処理を伴う Ajax アプリケーションのモデルも扱うことができる. 改良した変換器を用いて Ajax を用いた業務アプリケーションに適用し，良好な結果が得られた.

適用実験では，エラーが検出されなくなるまで，モデル修正・変換・検証のサイクルを何度か繰り返したが，SPIN で検出されたエラーや不具合シナリオによるシュミレーション結果から，アクティビティ図上での不具合部分を特定する際に，出力された PROMELA コードとアクティビティ図を見比べることもあった．SPIN のエラー情報は PROMELA を対象としているため，ある程度，やむを得ない面もあるが，今後は，SPIN のエラー情報から，アクティビティ図の該当部分を簡単に特定出来るような仕組みを検討したい.

```
delete_start:2          // 削除操作の開始
        ・
        ・
received_data:2         // 削除操作のバージョン取得
        ・
        ・
update_start:4
        ・              // 更新操作
update_end:4
        ・
        ・
create_start:4
        ・              // 新規登録操作
create_end:4
        ・
        ・
spin: line 697 "employee2.9.pml", Error: assertion violated
spin: text of failed assertion: assert((msg.version==version))
spin: trail ends after 1177474 steps
```

図 15　不具合シナリオ

Figure 15　The scenario of fault.

参考文献

1) J. Rumbaugh, I. Jacobson, G. Booch: The Unified Modeling Language Reference Manual (2nd Edition), Pearson Higher Education (2004).
2) G.J. Holzmann: THE SPIN MODEL CHECKER, Addison Wesley (2003).
3) 中島　震: SPIN モデル検査, 近代科学社 (2008).
4) Yutaka Yamada, Katsumi Wasaki: Automatic Generation of SPIN Model Checking Code from UML Activity Diagrams, IJACT: International Journal of Advancements in Computing Technology, Vol. 3, No. 8, pp. 189- 197 (2011).
5) OMG OCL, http://www.omg.org/spec/OCL/2.3/Beta2/
6) 株式会社チェンジビジョン astah* professional, http://astah.change-vision.com/ja/product/astah-professional.html
7) OMG XMI, http://www.omg.org/spec/XMI/2.4/Beta2