分散システム開発におけるモデル検査への視覚的支援手法

田原康之[††] 吉岡信和[††]
大須賀昭彦[†] 本位田真一[††,†††]

インターネットやイントラネットなどの広域開放型ネットワークが普及するにつれて，Web サービスやエージェントといった，開放型ネットワーク環境を有効に活用するための分散システム開発技術が急速に進展している．一方，その急速な発展のため，これらの技術を用いて実際の開発を行う際に，このような大規模かつ複雑なシステムが，機能やセキュリティなどに関し，要求仕様を満足しているかどうかを確認することが困難になってきている．従来このような問題を解決するための最も有効な技術の 1 つとして，モデル検査手法などの形式的検証技術があるが，実適用は現在でも容易ではない．本論文では，分散システム開発において，視覚的な支援により，モデル検査の適用を容易にする手法を提案する．本手法では，開発ツール IPEditor で作成した視覚的モデルに対し，その形式的表現を与え，そのモデルからモデル検査ツール SPIN が検証の対象とする Promela プログラムの一部を自動的に生成する．これにより，開発者は記述が困難な Promela プログラムの全体を直接記述しなくても，IPEditor により理解が容易な視覚的モデルを作成することにより，SPIN ツールを利用したモデル検査を用意に実施することができる．

Visual Support of Model Checking and Development of Distributed Systems

Yasuyuki Tahara,[†] Nobukazu Yoshioka,[††] Akihiko Ohsuga[†]
and Shinichi Honiden[††,†††]

As wide-area open networks like the Internet and intranets grow larger, various technology of distributed system development is rapidly emerging such as the Web service technology and the agent technology. On the other hand, the growth is so rapid that people are reporting a number of difficulties in actually operating practical systems such as management of complexity, poor flexibility to cope with frequent changes of requirements and environments, and security issues. Although the vendors are considering such difficulties, one of the most effective techniques is still in an early stage of research, that is, formal verification including model checking. Application of formal verification techniques is thought of as difficult because it is not easy to create models to verify. In this paper, we propose a method that visually supports model checking and development of distributed systems. Our technique realizes integration of visual modeling and model checking by providing formal representations of the visual models and a procedure of generation of a part of Promela programs that are formal models for the model checking tool SPIN. We can create the visual models using IPEditor, a development support tool for multi-agent applications originally and also applicable to other technologies including Web services. Thus the developer do not have to describe the entire Promela programs directly and therefore easily use SPIN in system development. We demonstrate the advantage of our method with an example of mutual exclusion.

1. はじめに

インターネットやイントラネットなどの広域開放型ネットワークが普及するにつれて，そのような環境を有効に活用するための分散システム開発技術が急速に進展している．たとえば，XML や HTTP などの WWW 技術に基づいて標準化されたインタフェースを持つ，疎結合かつ大粒度のコンポーネントである Web サービスや，自律的かつ知的な動作により，状況変化に柔軟に対応するソフトウェアであるエージェントといったものがある．我々は，現在モバイルエージェントを利用したマルチエージェントフレームワー

† 株式会社東芝研究開発センター知識メディアラボラトリー
Knowledge Media Laboratory, Corporate Research and Development Center, Toshiba Corporation
†† 国立情報学研究所
National Institute of Informatics
††† 東京大学
The University of Tokyo

ク Bee-gent [10] の開発を行っており，さらに Web サービスへの対応も進めている．我々は，Bee-gent によるモバイルエージェント技術の実用化を目的とし，Bee-gent に多くの実用的特徴を組み込んでいる．マルチエージェントアプリケーションのためのグラフィカルな開発支援ツール IPEditor もその 1 つである．我々は，B to B EC 向け電子カタログシステムを含めた，Bee-gent の業務用アプリケーションの開発もいくつか行っている．

一方，その急速な発展のため，これらの技術を用いて実際の開発を行う際に，このような大規模かつ複雑なシステムが，機能やセキュリティなどに関し，要求仕様を満足しているかどうかを確認することが困難になってきている．従来このような問題を解決するための最も有効な技術の 1 つとして，モデル検査手法などの形式的検証技術がある．しかし，この技術はまだ研究段階であって実用性が低いと考えられており，実開発への適用例が少ない．実用化に対する最大の課題は，検証の対象とする形式的モデルを構築するのが困難である，という点である．

本論文では，分散システム開発において，IPEditor による視覚的な支援により，モデル検査の適用を容易にする手法を提案する．本手法では，IPEditor で作成した視覚的モデルに対し，その形式的表現を与え，そのモデルからモデル検査ツール SPIN [3] が検証の対象とする Promela プログラムの一部を自動的に生成する．具体的には，Promela プログラムの構成要素である各プロセスに対し，プロセスの状態の集合，各状態での他プロセスとのメッセージ交換シーケンス，およびメッセージ交換の結果発生する状態遷移の記述を視覚的に行うことができる．そして自動生成された Promela プログラムに対し，必要に応じて手作業で修正を行い，さらに必要に応じて検証条件を線形時相論理式で記述するなどにより検証作業を行う．これにより，開発者は記述が困難な Promela プログラムの全体を直接記述しなくても，IPEditor により理解が容易な視覚的モデルを作成することにより，SPIN ツールを利用したモデル検査を容易に実施することができる．

本論文の構成は次のとおりである．2 章では，我々の手法を説明する．3 章では，排他制御の例題への適用を通じて，我々の手法の有効性を示す．4 章では関連研究との比較を行い，5 章で結論と今後の課題を述べる．

2. 分散システム開発におけるモデル検査への視覚的支援手法

本章では，本論文で提案する手法を説明する．概要は以下のとおりである（**図 1** 参照）．

(1) IPEditor ツールを用いて動作仕様を作成する．
(2) 動作仕様から Promela プログラムを自動生成し，必要に応じて手作業で修正する．
(3) 要求仕様を SPIN の検証条件として，たとえば検証パラメータや線形時相論理（Linear Temporal Logic，LTL）の論理式などで表す．
(4) SPIN で検証を実行する．
(5) 動作仕様の欠陥を発見した場合は，その欠陥を修正し，検証作業を繰り返す．
(6) IPEditor ツールを用いて，システムの Java コードを自動生成する．現状の IPEditor では，Bee-gent のためのコード生成のみサポートしているが，Web サービスのコード生成のサポートも予定している．

2.1 開発支援ツール IPEditor

IPEditor は，GUI 上でコンポーネントの挙動を状態遷移図で記述できる開発支援ツールである．また本ツールは，メッセージシーケンス図（相互作用図と呼ぶ）によるエージェント間相互作用の記述支援，テンプレート形式による ACL メッセージ定義記述支援，およびエージェントのビヘイビアパターン[5),6)] もサポートしている．IPEditor の試用版は前述の Bee-gent パッケージに含まれており，無料で入手可能である．IPEditor のスクリーンショットの例は次章で提示する．

2.2 モデル検査ツール SPIN

次に，SPIN の概要を説明する．SPIN（Simple Promela INterpreter）は，Promela 言語で記述された動作仕様が，与えられた各種の性質を満たすかどうかを自動検証するツールである．

Promela（PROtocol/PROcess MEta LAnguage）は，並列・並行に動作する複数の単位（プロセスと呼ぶ）をそれぞれ手続き的に記述し，プロセスの集合でシステム全体の動作を記述する．また，プロセス間通信機構を，通信チャネルを用いて明示的に記述することもできる．

Promela で記述された動作仕様（Promela プログラムと呼ぶ）が満たすべき性質には，デッドロックが発生しないこと，不適切な無限ループにならないこと，および LTL（Linear Temporal Logic，線形時相論理）と呼ばれる形式論理における論理式で表される性質が

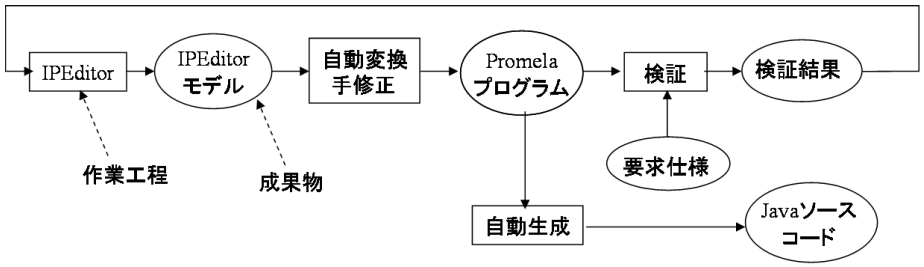

図 1 本手法の概要
Fig. 1 Summary of our development process.

ある．最後のものには，安全性（セキュリティ違反などの不適切な状態にならない）や活性（要求を受理すればそのうち必ず応答するなど）がある．

SPIN には，XSPIN と呼ばれる GUI がある．その機能としては，Promela プログラムや LTL 論理式の編集と表示，各種設定，SPIN 機能（シミュレーションや検証）の呼び出し，実行状態の表示（トレースや変数），プログラムのオートマトン表示，および実行トレースのシーケンス図による表示がある．

SPIN の検証機能は，モデル検査手法により実現されている．モデル検査とは，システムが遷移可能な状態を網羅的に列挙し，その全体に対して与えられた性質が満たされるかどうかを厳密に確認する手法である．

2.3 Promela プログラムの生成

IPEditor では，作成した動作仕様から Promela プログラムの一部が自動生成できる．詳しくは，形式的なグラフ構造で表現された動作仕様をプログラムに変換する．以下，動作仕様のグラフ表現を説明し，次にプログラムへの変換方法を述べる．

2.3.1 IPEditor 仕様のグラフ表現

IPEditor 仕様を Promela プログラムに変換するためには，何らかの仕様の形式的表現が必要である．本手法では，文献 11) で用いたのと同様なモデルを使用する．以下に，その詳細について説明する．

IPEditor 仕様は 3 つの構成要素，すなわち，各コンポーネント（IPEditor では「エージェント」と呼んでいるが，後で Promela プログラムに変換される際にプロセスに変換されるので，以下「プロセス」と呼ぶ）の状態遷移図，各状態の相互作用図，およびメッセージ定義から構成される．相互作用図は各状態ごとに 1 つずつ用意される．またメッセージ定義は，相互作用図の各メッセージ矢印ごとに与えられる．

以降で，各構成要素を形式的に表現し，その後どのように結合するかを検討する．各構成要素の形式的表現は以下のとおりである．

- プロセス A_i の状態遷移図は，有向グラフ $(N_i, E_i, \iota_i, \epsilon_i)$（ただし，$E_i \subseteq N_i \times N_i; \iota_i, \epsilon_i \in N_i$ はそれぞれ始状態と終状態）で表現する．各状態 $s \in N_i$ には 1 つの相互作用図が割り当てられる．その形式的表現は，メッセージ定義の列とメッセージ定義の集合との対（$\langle M_j | j = 1, 2, \ldots, n \rangle, Ms$）である．$Ms$ の要素は，この状態において最後に発生しうるメッセージ交換を表し，複数存在する可能性がある．次状態は，どのメッセージ交換が発生するかによって変化する．
- 上述した列に含まれるメッセージ定義 M_j は，組 (s, r, c) で表される．ここで s，r，c は以下のものを指す．
  - s はメッセージを送信するプロセス
  - r はメッセージを受信するプロセス
    ただし，s か r のいずれか一方は，状態遷移図で指定されている A_i
  - c はメッセージの内容を表す文字列
- Ms の要素であるメッセージ定義 M は，前述のメッセージ定義 M_j とは異なり，組 (s, r, c, ns) によって形式的に表現される．ここで，前述のメッセージ定義の形式的表現にはなかった，次状態 ns が含まれていることに注意する．これは，Ms の要素である M は，状態遷移を引き起こすイベントに相当するからである．したがって，M は当該状態の相互作用図においては終端のメッセージとなるので，「終端メッセージ定義」と呼ぶ．
  また，各 $M_1, M_2 \in Ms$，$M_i = (s_i, r_i, c_i, ns_i)$ に対し，$s_1 = s_2$，$r_1 = r_2$．すなわち，s と r は，すべての $M \in Ms$ について，それぞれ共通である．
  さらに，状態 s から s' への遷移を表すエッジ (s, s') が存在する場合は，遷移を引き起こすイベントとして，s' を次状態とする Ms の要素の存在が必要．また逆に，第 3 の状態 s'' に対し，次状態が s'' となるような Ms の要素が存在すれば，エッジ (s, s'') が存在することが必要である．

2.3.2 グラフ表現から Promela プログラムへの変換

次に，以下の手順によって，動作仕様のグラフ表現を Promela プログラムに変換する．

(1) メッセージ内容の文字列を `mtype` 型のデータ，すなわちメッセージを表す記号として宣言する．

(2) ローカル相互作用図で通信を行う各プロセスの対に対し，通信チャネルを定義する．プロセス P がプロセス Q にメッセージを送信するチャネルには，P`To`Q という名前を付け，サイズは 1 とする．

(3) 各プロセスに対し，起動されたプロセスであることを表す `active proctype` 宣言を生成する．

(4) プロセスの本体は以下のようにして生成する．

(a) 各状態に対し，メッセージグラフを以下のように文の列に変換する．

- 当該プロセスが他のプロセスにメッセージを送信するメッセージ定義を送信文に変換する．受信文も同様にして生成される．
- メッセージ定義の次ノードが 1 つしかない場合は，それらのノードから生成された文を，元のノードにおけるメッセージ交換の方向に応じて，送信の場合`;`，および受信の場合`->`で結合する．ここでメッセージ定義の「次ノード」とは，そのメッセージ定義に応じて次のように定義される．
  - 終端メッセージ定義でない場合，相互作用図において次のメッセージ定義があればそのメッセージ定義
  - 相互作用図において次のメッセージ定義がなければ，終端メッセージ定義
  - 終端メッセージ定義の場合は，次状態の最初のメッセージ定義

  なお，“`;`” も “`->`” も同じ逐次実行を表すが，Promela において一般に推奨される記述スタイルとして，このように使い分けている．また，本ケースでは状態遷移が決定的なので，逐次実行として変換する．
- ノードが 2 つ以上の次ノードを有する場合は，どちらのノードに遷移するかは非決定的とすべきであるため，生成される文の列は，そのような非決定的分岐を表す `if` 構造を含むことになる．詳しくは，ノード N_0 の次にノード $N_i(i = 1, \ldots, n)$ が続き，ノード $N_j(j = 0, \ldots, n)$ がそれぞれ文 s_j に変換される場合，以下のような文の列が生成される．

  s_0`;`
  `if`
  `::` s_1 `;`
  (s_1 の後に続く文の列)
  `::` s_2 `;`
  ...
  `::` s_n `;`
  (s_n の後に続く文の列)
  `fi`
- 最後に，Ms に含まれるメッセージ定義に対応する文の各々に対し，その次状態を S とすると，その後に “`goto` S” を追加することにより，次状態に遷移させる．

(b) 各状態名 S に対し，対応する文の列にラベル S`:`を付加することにより，`goto` 文で当該状態に遷移できるようにする．

(c) 以上で生成された文の列全体が，順番に並べられる．

なお，以上で自動生成したプログラムは，手作業で修正することが必要な場合がある．たとえば，自動生成したプログラムのままだと，`if` 文はメッセージを文字列としてみた場合の条件分岐しか行うことができない．したがって，メッセージの内容に応じて振舞いを変化させたい場合は，メッセージの内容に対する条件判断を行う記述を追加する必要がある．

さらに，以上の自動生成手続きにより，IPEditor 仕様の操作的セマンティクスが，次のように与えられていることになる．すなわち，IPEditor 仕様は，本手続きにより，完全に形式的な操作によって，IPEditor 仕様と意味的に同等な Promela プログラムまで，自動的に変換されることになる．一方，Promela プログラムには，文献 3) で示されているように，操作的セマンティクスが与えられている．したがって，自動生成手続きと，自動生成された Promela プログラムのセマンティクスを統合することにより，IPEditor 仕様に操作的セマンティクスが与えられることになる．なお，前述したメッセージ内容に対する条件判断の追加など，生成された Promela プログラムを手作業で

修正した場合は，元の IPEditor 仕様からセマンティクスが変化する可能性がある，という点には注意が必要である．

3. 適 用 例

本章では，本手法を並行プロセスの相互排除問題，および 2 相コミットトランザクション処理に適用した例を示す．

3.1 並行プロセスの相互排除問題

本例題の詳細は次のとおりである．

- 2 つのクライアントプロセス `client1` と `client2`，および 2 つのサーバプロセス `server1` と `server2` があるものとする．
- `client1` は 1 つ目のリクエストを `server1` に，2 つ目を `server2` に，そして最後に `server1` に送信する．
- `server1` は，`client1` からの上記 2 リクエストの間には，他のリクエストを受け付けないものとする（ただし，この制限は後でデッドロックを回避するために緩和される）．
- 以上の内容を，`client1` を `client2` に，また `server1` を `server2` に，それぞれ置き換えた場合にも適用する．

3.1.1 例題の IPEditor モデル

本例題の IPEditor モデルの一部を**図 2**，**図 3**，**図 4**，**図 5** に示す．これらの図において，`client1` のローカル相互作用図は `INIT` 状態のもので，メッセージ定義は `request` メッセージのものである．

本動作仕様から Promela プログラムが生成される．生成されたプログラムを付録 A.1.1 に記述する．

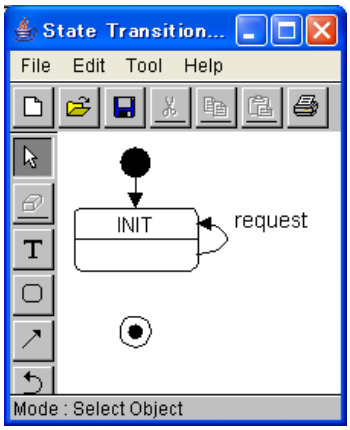

図 2 相互排除問題の IPEditor モデルの一部：`client1` の状態遷移図

Fig. 2 Part of the IPEditor model of the mutual exclusion problem: the state transition diagram of `client1`.

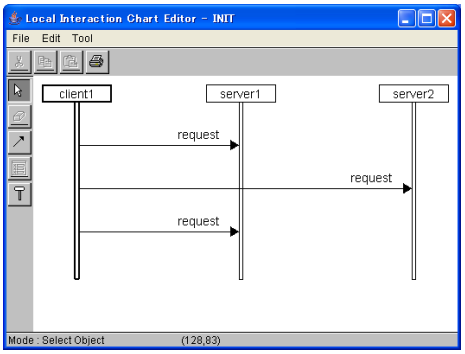

図 3 相互排除問題の IPEditor モデルの一部：`client1` のローカル相互作用図

Fig. 3 Part of the IPEditor model of the mutual exclusion problem: a local interaction diagram of `client1`.

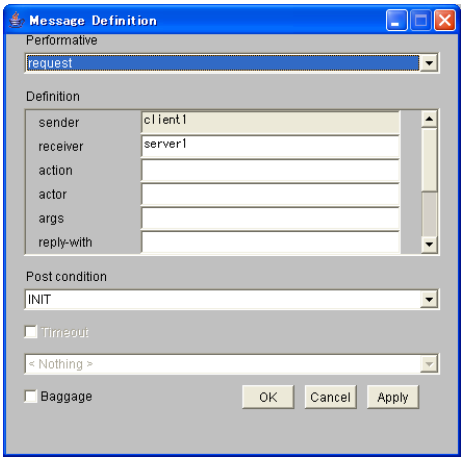

図 4 相互排除問題の IPEditor モデルの一部：`client1` のメッセージ定義

Fig. 4 Part of the IPEditor model of the mutual exclusion problem: a message definition of `client1`.

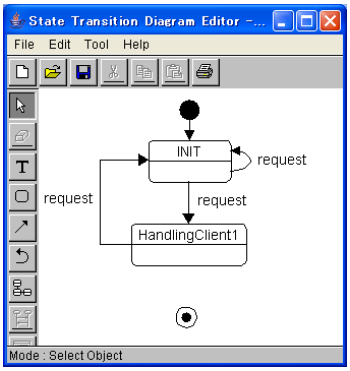

図 5 相互排除問題の IPEditor モデルの一部：`server1` の状態遷移図

Fig. 5 Part of the IPEditor model of the mutual exclusion problem: the state transition diagram of `server1`.

```
pan: invalid end state (at depth 62)
pan: wrote pan_in.trail
(Spin Version 4.1.1 -- 2 January 2004)
Warning: Search not completed
+ Partial Order Reduction

Full statespace search for:
never claim          - (not selected)
assertion violations - (disabled by -A flag)
cycle checks         - (disabled by -DSAFETY)
invalid end states +

State-vector 44 byte, depth reached 62, errors: 1
      65 states, stored
      32 states, matched
      97 transitions (= stored+matched)
       0 atomic steps
hash conflicts: 0 (resolved)
(max size 2^19 states)
```

図 6 デッドロック検出の報告
Fig. 6 Report of deadlock detection.

3.1.2 モデル検査と欠陥の修正

本システムにおいてデッドロックが発生するかどうかを確認するためには，SPIN を用いて “Invalid End-states” オプションを付けてプログラムを検証する．本例題では，XSPIN の利用により，デッドロックの可能性が，**図 6** の下から 7 行目においてエラーとして報告され，**図 7** のようにデッドロック発生時のトレースがメッセージシーケンス図として表示される．図 7 において，各縦線は左から `client1`，`client2`，`server1`，および `server2` の動作の時系列を表し，その間の矢印がメッセージ送受信を表す．そして最下部において色の異なる矩形がデッドロックとなっていることを表す．この図において，`client1` と `client2` が，それぞれ `server1` と `server2` にリクエストを送信した後，お互いに次のリクエストを別のサーバに送信できないため，デッドロックになっていることが分かる．

このような欠陥の修正のために，各サーバプロセスを変更して，両クライアントのリクエストを同時に処理できるようにする．ただしこの場合は，同時処理によって不整合が生じないように，サーバプロセスの実装を行う必要がある．`server1` の動作仕様は，**図 8** のように変更される．

修正後に Promela プログラム生成と検証を再度行うと，付録 A.1.2 のようなプログラムが得られ，今回はデッドロックが発生しないという検証結果が報告される．

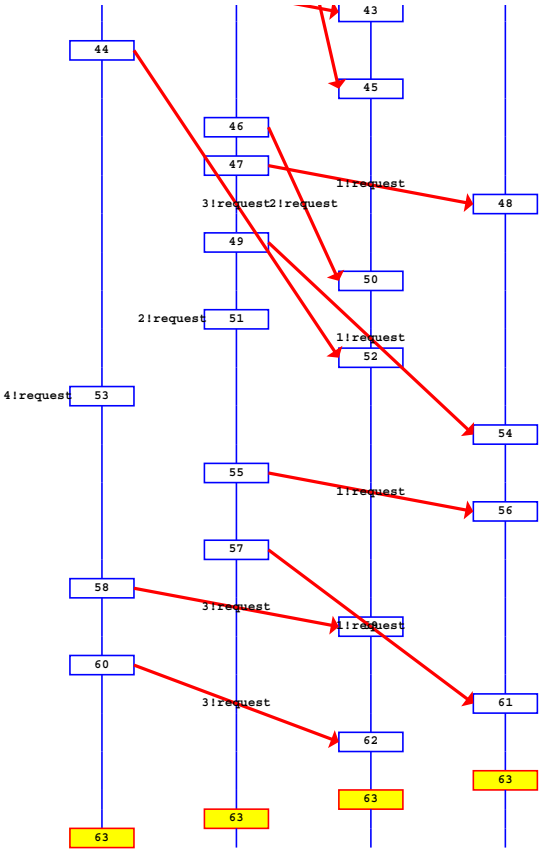

図 7 デッドロック発生までのトレースのメッセージシーケンス図
Fig. 7 Message sequence diagram of the trace until the occurrence of the deadlock.

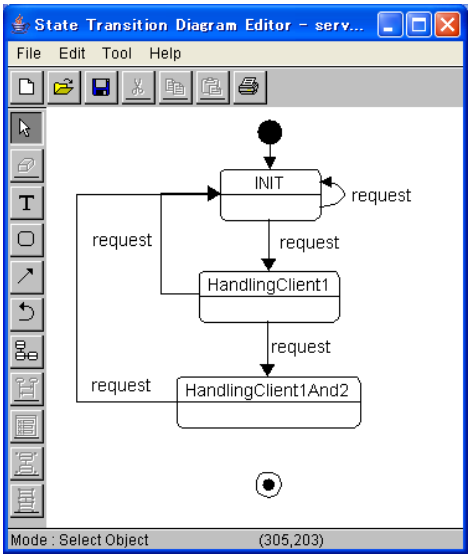

図 8 修正後の `server1` の動作仕様における状態遷移図
Fig. 8 State transition diagram of the `server1` behavior specifications after the modification.

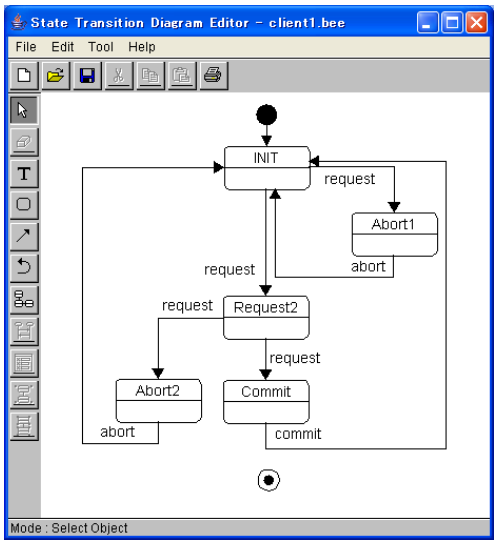

図 9 2 相コミットトランザクション処理例題の IPEditor モデルの一部：client1 の状態遷移図
Fig. 9 Part of the IPEditor model of the two-phase commitment problem: the state transition diagram of client1.

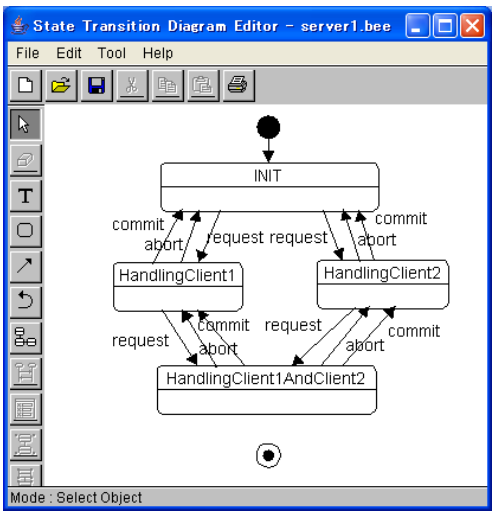

図 10 2 相コミットトランザクション処理例題の IPEditor モデルの一部：server1 の状態遷移図
Fig. 10 Part of the IPEditor model of the two-phase commitment problem: the state transition diagram of server1.

3.2 2 相コミットトランザクション処理

本例題の詳細は次のとおりである．

- 相互排除問題と同様に，2 つのクライアントプロセス client1 と client2，および 2 つのサーバプロセス server1 と server2 があるものとする．
- client1 は 1 つ目のリクエストを server1 に，そして 2 つ目を server2 に送信する．しかし各サーバは，リクエストを受け取った時点では，その処理を完了しない．
- client1 は各リクエスト送信後，何らかの原因で 1 つ目のリクエストを送る前の状態に復帰させたい場合がある．そのとき，すでにリクエストを送信したサーバにアボート処理依頼メッセージを送信する．
- client1 は，アボートの必要がなく両リクエスト処理を完了させたい場合に，各サーバにコミット処理依頼メッセージを送信する．それを受け取ったサーバは，リクエスト処理を完了する．
- 以上の内容を，client1 と client2 を，また server1 と server2 を，それぞれ交換した場合にも適用する．すなわち，client2 の動作は，リクエスト送信順が server2，server1 となること以外は，client1 と同様である．

本例題の IPEditor モデルの一部を**図 9** と**図 10** に示す．

本動作仕様から生成された Promela プログラムを付録 A.1.3 に記述する．本プログラムは，SPIN によりデッドロックがないことが検証される．

4. 関 連 研 究

本章では，本論文の手法と関連研究の比較を行う．

SPIN に対する XSPIN のように，GUI を備えたモデル検査ツールは存在するが，本論文の手法のように，モデル作成作業から視覚的に支援するものはない．一方，UML（Unified Modeling Language）で記述された視覚的モデルを Promela に変換して SPIN で検証可能とする手法[4] があるが，動作記述については状態遷移図しか扱っていないので，各状態内の動作は開発者が記述する必要がある．一方本論文では，状態内のメッセージ送受信動作についても，シーケンス図を扱うことにより，時系列情報は自動生成できる．また，BPEL4WS に代表される Web サービスの連携動作を記述する言語に対し，モデル検査などの自動検証手法を適用する研究が提案されている[1),2)]．BPES4WS に対しては，最近の CASE ツールの多くにより視覚的開発支援が可能になっており，本研究と同等の効率的なモデル検査適用が考えられるが，本研究のようにプロセス全体を一貫してサポートしたものはない．本論文の手法は，視覚的モデルの形式的な表現を与え，そのような表現に対するモデル検査のための Promela プログラムへの変換機能や，実行可能なソースコード生成機能により，そのような一貫したサポートを実現している．

5. おわりに

本論文では，分散システム開発において，IPEditorによる視覚的な支援により，モデル検査の適用を容易にする手法を提案した．本手法では，IPEditorで作成した視覚的モデルに対し，その形式的表現を与え，そのモデルからモデル検査ツールSPIN[3]が検証の対象とするPromelaプログラムの一部を自動的に生成する．これにより，開発者は記述が困難なPromelaプログラムの全体を直接記述しなくても，IPEditorにより理解が容易な視覚的モデルを作成することにより，SPINツールを利用したモデル検査を容易に実施することができる．

今後は以下のような課題を解決する方向で研究を進める．

- IPEditorはまだプロトタイプレベルのツールであり，さらに改良を進める必要がある．たとえば，プロセスのインスタンスではなくクラスを開発可能とすることや，メッセージの内容を文字列だけではなくさらに複雑な構造のものを記述できるようにすることがある．また，現在は検証条件を論理式で記述する部分の視覚的支援を行っていないので，文献8)のようにシーケンス図から時相論理式を生成する手法などを取り入れる必要がある．
- IPEditorは元来マルチエージェントアプリケーションのためのツールである．一方，マルチエージェント技術はWebサービスなどの他の技術との統合もさかんに研究されている[9]．我々は，文献7)においてマルチエージェントアプリケーションに対する形式的検証の枠組を確立しているので，本論文の手法との統合は有益であると考える．
- 本論文の例題は非常に簡単なものである．そこで，さらに現実的な例題への適用を行って，本論文の手法の有効性を実証してゆきたい．特に，どの程度の規模のアプリケーションにまで，現実的な検証を行うことが可能か，ということの検討が必要と考える．

参考文献

1) Foster, H., Uchiteland, S., Magee, J. and Kramer, J.: Model-based verification of Web service compositions, *Proc. ASE2003*, pp.152–163 (2003).
2) Fu, X., Bultan, T. and Su, J.: Analysis of interacting BPEL Web services, *Proc. WWW2004* (2004).
3) Holzmann, G.J.: *The SPIN model checker: Primer and reference manual*, Addison Wesley (2004).
4) Schäfer, T., Knapp, A. and Merz, S.: Model checking UML state machines and collaborations, *Electronic Notes in Theoretical Computer Science*, Vol.55, No.3 (2001).
5) Tahara, Y., Ohsuga, A. and Honiden, S.: Agent system development method based on agent patterns, *Proc. ICSE'99*, pp.356–367, IEEE (1999).
6) Tahara, Y., Ohsuga, A. and Honiden, S.: Behavior patterns for mobile agent systems from the development process viewpoint, *Proc. ISADS 2001*, pp.239–242, IEEE Computer Society (2001).
7) Tahara, Y., Ohsuga, A. and Honiden, S.: Mobile agent security with the IPEditor development tool and the Mobile UNITY language, *Proc. Agents 2001*, pp.656–662, ACM Press (2001).
8) van Lamsweerde, A. and Willemet, L.: Inferring declarative requirements specifications from operational scenarios, *IEEE TSE*, Vol.24, No.12, pp.1089–1114 (1998).
9) Walton, C.D.: Model checking multi-agent Web services, *Proc. AAAI Spring Symp. on Semantic Web Services* (2004).
10)（株）東芝：Bee-gent WWWページ．http://www2.toshiba.co.jp/beegent/
11) 田原康之，大須賀昭彦，本位田真一：IPEditor開発ツールとMobile UNITY言語の適用によるモバイルエージェントセキュリティの実現，情報処理学会論文誌，Vol.43, No.6, pp.1582–1597 (2002).

付　録

A.1 例題のPromelaプログラム

A.1.1 相互排除問題：欠陥修正前

```
mtype = { request };

chan client1ToServer1 = [1] of { mtype }
chan client1ToServer2 = [1] of { mtype }
chan client2ToServer1 = [1] of { mtype }
chan client2ToServer2 = [1] of { mtype }

active proctype client1() {
INIT:client1ToServer1!request;
  client1ToServer2!request;
  client1ToServer1!request;
  goto INIT
}
```

```
active proctype client2() {
INIT:client2ToServer2!request;
  client2ToServer1!request;
  client2ToServer2!request;
  goto INIT
}

active proctype server1() {
INIT:if
       :: client1ToServer1?request ->
          goto HandlingClient1
       :: client2ToServer1?request ->
          goto INIT
  fi;
HandlingClient1:
  client1ToServer1?request ->
  goto INIT
}

active proctype server2() {
INIT:if
       :: client2ToServer2?request ->
          goto HandlingClient2
       :: client1ToServer2?request ->
          goto INIT
  fi;
HandlingClient2:
  client2ToServer2?request ->
  goto INIT
}
```

A.1.2 相互排除問題：欠陥修正後

```
mtype = { request };

chan client1ToServer1 = [1] of { mtype };
chan client1ToServer2 = [1] of { mtype };
chan client2ToServer1 = [1] of { mtype };
chan client2ToServer2 = [1] of { mtype };

active proctype client1() {
INIT:client1ToServer1!request;
  client1ToServer2!request;
  client1ToServer1!request;
  goto INIT
}

active proctype client2() {
INIT:client2ToServer2!request;
  client2ToServer1!request;
  client2ToServer2!request;
  goto INIT
}

active proctype server1() {
INIT:if
       :: client1ToServer1?request ->
          goto HandlingClient1
       :: client2ToServer1?request ->
          goto INIT
  fi;
HandlingClient1:if
  :: client1ToServer1?request ->
     goto INIT
  :: client2ToServer1?request ->
     goto HandlingClient1And2
  fi;
HandlingClient1And2:
  client1ToServer1?request ->
  goto INIT
}

active proctype server2() {
INIT:if
       :: client2ToServer2?request ->
          goto HandlingClient2
       :: client1ToServer2?request ->
          goto INIT
  fi;
HandlingClient2:if
  :: client2ToServer2?request ->
     goto INIT
  :: client1ToServer2?request ->
     goto HandlingClient2And1
  fi;
HandlingClient2And1:
  client2ToServer2?request ->
  goto INIT
}
```

A.1.3 2相コミットトランザクション処理

```
mtype = { request, commit, abort };

chan client1ToServer1 = [1] of { mtype }
chan client1ToServer2 = [1] of { mtype }
```

```
chan client2ToServer1 = [1] of { mtype }
chan client2ToServer2 = [1] of { mtype }

active proctype client1() {
INIT:client1ToServer1!request;
  if
    :: goto Request2
    :: goto Abort1
  fi;
Request2:client1ToServer2!request;
  if
    :: goto Commit
    :: goto Abort2
  fi;
Abort1:client1ToServer1!abort;
  goto INIT;
Commit:client1ToServer1!commit;
  client1ToServer2!commit;
  goto INIT;
Abort2:client1ToServer1!abort;
  client1ToServer2!abort;
  goto INIT
}

active proctype client2() {
INIT:client2ToServer2!request;
  if
    :: goto Request2
    :: goto Abort1
  fi;
Request2:client2ToServer1!request;
  if
    :: goto Commit
    :: goto Abort2
  fi;
Abort1:client2ToServer2!abort;
  goto INIT;
Commit:client2ToServer2!commit;
  client2ToServer1!commit;
  goto INIT;
Abort2:client2ToServer2!abort;
  client2ToServer1!abort;
  goto INIT
}

active proctype server1() {
INIT:if
       :: client1ToServer1?request ->
          goto HandlingClient1
       :: client2ToServer1?request ->
          goto HandlingClient2
  fi;
HandlingClient1:if
       :: client1ToServer1?commit ->
          goto INIT
       :: client1ToServer1?abort ->
          goto INIT
       :: client2ToServer1?request ->
          goto HandlingClient1AndClient2
  fi;
HandlingClient2:if
       :: client2ToServer1?commit ->
          goto INIT
       :: client2ToServer1?abort ->
          goto INIT
       :: client1ToServer1?request ->
          goto HandlingClient1AndClient2
  fi;
HandlingClient1AndClient2:if
       :: client1ToServer1?commit ->
          goto HandlingClient2
       :: client1ToServer1?abort ->
          goto HandlingClient2
       :: client2ToServer1?commit ->
          goto HandlingClient1
       :: client2ToServer1?abort ->
          goto HandlingClient1
  fi;
}

active proctype server2() {
INIT:if
       :: client1ToServer2?request ->
          goto HandlingClient1
       :: client2ToServer2?request ->
          goto HandlingClient2
  fi;
HandlingClient1:if
       :: client1ToServer2?commit ->
          goto INIT
       :: client1ToServer2?abort ->
          goto INIT
```

```
       :: client2ToServer2?request ->
          goto HandlingClient1AndClient2
  fi;
HandlingClient2:if
       :: client2ToServer2?commit ->
          goto INIT
       :: client2ToServer2?abort ->
          goto INIT
       :: client1ToServer2?request ->
          goto HandlingClient1AndClient2
  fi;
HandlingClient1AndClient2:if
       :: client1ToServer2?commit ->
          goto HandlingClient2
       :: client1ToServer2?abort ->
          goto HandlingClient2
       :: client2ToServer2?commit ->
          goto HandlingClient1
       :: client2ToServer2?abort ->
          goto HandlingClient1
  fi;
}
```

(平成 16 年 5 月 19 日受付)
(平成 16 年 11 月 1 日採録)

田原 康之（正会員）

1966 年生．1991 年東京大学大学院理学系研究科数学専攻修士課程修了．同年（株）東芝入社．1993〜1996 年情報処理振興事業協会に出向．1996〜1997 年英国 City 大学客員研究員．1997〜1998 年英国 Imperial College 客員研究員．2003 年より国立情報学研究所に勤務．2004 年より同研究所特任助教授．博士（情報科学）（早大）．エージェント技術，およびソフトウェア工学等の研究に従事．日本ソフトウェア科学会会員．

吉岡 信和（正会員）

1993 年富山大学工学部電子情報工学科卒業．1995 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．1998 年同大学院大学情報科学研究科博士後期課程修了．博士（情報科学）．同年（株）東芝入社．2002 年より国立情報学研究所に勤務，2004 年より同研究所特任助教授，主にエージェント技術，およびソフトウェア工学の研究に従事．現在に至る．日本ソフトウェア科学会会員．

大須賀昭彦（正会員）

1958 年生．1981 年上智大学理工学部数学科卒業．同年（株）東芝入社．1985〜1989 年（財）新世代コンピュータ技術開発機構（ICOT）に出向．現在（株）東芝研究開発センター知識メディアラボラトリー主任研究員．工学博士（早大）．電気通信大学大学院客員教授ならびに大阪大学大学院非常勤講師兼任．主としてソフトウェアのためのフォーマルメソッド，エージェント技術の研究に従事．1986 年度情報処理学会論文賞受賞．電子情報通信学会，日本ソフトウェア科学会，IEEE CS 各会員．

本位田真一（正会員）

1953 年生．1976 年早稲田大学理工学部電気工学科卒業．1978 年同大学大学院理工学研究科電気工学専攻修士課程修了．（株）東芝を経て 2000 年より文部科学省国立情報学研究所．現在，研究主幹・教授，2001 年より東京大学大学院情報理工学系研究科教授を併任，現在に至る．2002 年 5 月〜2003 年 1 月英国 UCL ならびに Imperial College 客員研究員（文部科学省在外研究員）．工学博士（早大）．1986 年度情報処理学会論文賞受賞．エージェント技術，オブジェクト指向技術，ソフトウェア工学の研究に従事．IEEE，ACM，日本ソフトウェア科学会等各会員．本学会理事．