修士学位論文

題目

非同期モデルのシーケンス図に含まれるメッセージ順序の曖昧性除去手法

指導教員
楠本 真二 教授

報告者
楠　野明

平成 27 年 2 月 6 日

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 26 年度 修士学位論文

非同期モデルのシーケンス図に含まれるメッセージ順序の曖昧性除去手法

楠　野明

内容梗概

ソフトウェア開発において品質保証は重要であり，そのための手法としてソフトウェアテストや欠陥の自動修正手法なと様々な手法が提案されている．しかし，ソフトウェアが修正すべき欠陥を含んでしまうのは，実装の段階だけとは限らない．例えば，仕様設計の段階でその原因が発生してしまう場合がある．開発が進み，そのような欠陥が実装を行う段階で発覚した場合，開発者は仕様設計の段階まで遡って修正し，再び実装しなければならなくなる場合もある．この場合，欠陥の修正にかかるコストは大きくなってしまう．

本研究は，仕様設計の段階で用いられるシーケンス図において欠陥を発見，および修正する手法を提案する．本研究が修正の対象とする欠陥は非同期モデルを表すシーケンス図において行われるメッセージ送受信の順序が逆転してしまうことによる欠陥とした．また，提案手法は発見した箇所が必ずしも修正すべきものとは限らないため，開発者に修正すべきかを確認しながら修正する半自動的な手法とした．手法は，まずシーケンス図からモデル検査に用いるモデルを生成する．次にモデルに含まれるオブジェクト間で行われるメッセージのやり取りから，モデル検査のもうひとつの入力である検査式を生成する．ここで生成される検査式はオブジェクト毎に，そのオブジェクトに関連するメッセージの順序が逆転しないかをチェックするものである．そのためシーケンス図が含むオブジェクトとメッセージの数に応じて複数得られる．次に生成したモデルと検査式を用いてモデル検査を行う．この時，モデルが検査式に表された検査したい性質を満たさない場合，反例が得られる．反例が得られた場合には提案手法は開発者に対して，その反例が起こらないようにシーケンス図を修正する方法を提案する．これはシーケンス図に含まれる情報だけではこの修正方法を採用するかどうかを判断することはできないためである．そのため開発者に提案し，修正を適用するかどうかを選んでもらう．すべての検査式に対してモデル検査を行い，反例が出力されなくなれば終了する．

提案手法をツールとして実装し，いくつかのシーケンス図に適用して 2 種類の評価を行った．その結果，1 つ目の評価ではシーケンス図が含むオブジェクトの数やメッセージの数に対して，提案手法の生成する修正案の数や実行時間を評価した．2 つ目の評価では実際の開発現場で用いられるシーケンス図に対して，ツールが出力する修正が正しいことが確認できた．

主な用語

シーケンス図，モデル検査，LTL 式，XML

目 次

図 目 次

表 目 次

1　まえがき

ソフトウェアの信頼性は重要であり，ソフトウェアの品質を保証するための手法として欠陥の検出手法や修正手法に関する研究が盛んに研究が行われている．例えば，プログラムのソースコードとテストを入力として，プログラムに含まれる障害箇所，および障害箇所に対する修正案を出力する手法 [1] や，プログラムのソースコードを既存のコードを再利用することで修正する手法 [2–4] などソースコードを入力として様々な手法が提案されている．しかし，ソフトウェアが欠陥を生じるのは実装の段階だけとは限らない [5]．例えば，仕様設計の段階で欠陥を生じてしまう場合もある．そのような欠陥が実装を行う段階で初めて発覚した祭は，開発者は仕様設計の段階まで遡って修正し，再び実装しなければならない場合もある．この場合，欠陥の修正にかかるコストは大きくなってしまうため，欠陥を早期に発見することを目的とした研究も盛んに行われている [6–12]．

本研究は，仕様設計の段階で使用されるシーケンス図においてメッセージ順に関する欠陥を発見，および修正する手法を提案する．対象とする欠陥は非同期モデルを表すシーケンス図において行われるメッセージ送受信の順序が状況により逆転してしまうことによる欠陥とした．この欠陥を本研究では “メッセージ順序の曖昧性” と定義して以降の説明を行う．提案手法が発見するメッセージ順序の曖昧性は必ずしも修正すべきものとは限らないため，開発者に修正すべきかを確認しながら修正する半自動的な手法としている．

手法は，まずシーケンス図からモデル検査に用いるモデルを生成する．次にモデルに含まれるオブジェクト間で行われるメッセージのやり取りから，モデル検査のもうひとつの入力である検査式を生成する．ここで生成される検査式はオブジェクトごとに，そのオブジェクトに関連するメッセージにメッセージ順序の曖昧性が存在するかをチェックするものである．そのためシーケンス図が含むオブジェクトとメッセージの数に応じて，検査式は複数得られる．次に生成したモデルと検査式を用いてモデル検査を行う．モデルが検査式に表された検査したい性質を満たさない場合，つまりモデルがメッセージ順序の曖昧性を持つ場合，提案手法は開発者に対してそのメッセージ順序の曖昧性を除去できるシーケンス図の修正候補を提案する．シーケンス図に含まれる情報だけではこの修正を行うべきかどうかを判断することはできない．そのため開発者に提案し，その修正を適用するかどうかを選んでもらう．すべての検査式に対してモデル検査を行い，修正候補が提案されなくなれば終了する．

提案手法をツールとして実装し，2 つの評価を行った．1 つ目の評価では既存研究やツールで用いられているシーケンス図を収集して，ツールを適用した．その結果，ツールが生成する修正案の数や実行時間を計測し評価を行った．2 つ目の評価では実際の開発現場の経験を元にして得られたメッセージ順序の曖昧性を持つシーケンス図を対象としてツールを実行した．その結果，2 つのシーケンス図に対して 10 個の修正候補が提案された．この内，1 個

の修正候補に関しては実際に開発現場で想定されるエラーシーケンスを解決できる修正であり，開発者に確認したところ，この修正が正しいことが確認できた．

以降，2 章では，研究の背景や論文中で登場する用語の説明を行う．3 章では，研究の動機を説明する．4 章では，提案手法が行う処理の内容について述べる．5 章では，実施した評価の方法と結果を説明する．7 章では，既存の研究について触れる．6 章では，本研究の妥当性の脅威について述べる．8 章では，この研究のまとめを行い，今後の課題を示す．

2　研究背景

本章では研究の背景となる諸技術と関連研究について簡単にふれる．

2.1　UML

UML(Unified Modeling Language) [13] は，Object Management Group(OMG) がオブジェクト指向システムの開発のために標準化した仕様記述言語である．UML2.0 [14] では構造図と振る舞い図の 2 つの分類で以下に示す 13 種類の図が定義されており，システムの構造の可視化や振る舞いの特定などに用いられている．

- 構造図

  クラス図： システムのクラス構造を表現する．

  オブジェクト図: クラスをより具体化したオブジェクト同士の関係を表現する．

  パッケージ図: クラスなどをグループ化して整理された関係を表現する．

  コンポジット構造図: クラスやコンポーネントの内部構造を表現する．

  コンポーネント図: 物理的な構成要素 (ファイル，ヘッダ，ライブラリなど) からシステムの構造を表現する．

  配置図: システムの物理的な構成を表現する．

- 振る舞い図

  ユースケース図： 外部からの要求に対するシステムの振る舞いを表現する

  アクティビティ図: システムにおける処理のフローや状態遷移を表現する

  状態遷移図: 1 つのオブジェクトの状態に注目し，その変化を表現する

  シーケンス図: クラスやオブジェクト間の相互作用を時系列に表現する

  コミュニケーション図: クラスやオブジェクト間の関連と応答を視覚的に表現する

  相互作用概要図: ユースケース図やシーケンス図を構成要素として広域な処理の流れを表現する

  タイミング図: クラスやオブジェクトの状態遷移を時系列で表現する

UML の定義は Meta-Object Facility (MOF) [15] のメタモデルを用いて行われている．UML モデルは，QVT(Queries/Views/Transformations) [16] などの変換言語を使って Java などに自動的に変換できる．この機構を使うことで，クラス図からソースファイルのスケルトンを生成することもできる．OMG は UML を 4 階層のアーキテクチャで定義している．

1. MOF: M3 層に相当し，UML メタモデルを記述するための言語を定義する．
2. UML メタモデル: MOF のインスタンス．M2 層に相当し，UML モデルを記述するための言語を定義する．
3. UML モデル: UML メタモデルのインスタンス．M1 層に相当し，オブジェクトモデルを記述するための言語を定義する．
4. オブジェクトモデル: UML モデルのインスタンス．M0 層に相当し，特定のオブジェクトを表現する．

一般に用いられている UML は M1 層が相当する．

2.2 シーケンス図

シーケンス図とは UML において定義されている図の 1 つである．ライフラインやメッセージなどの構成要素によって，オブジェクト間の相互作用を時系列に表現する．ライフラインはシーケンス図の上部に四角で表す．オブジェクト名は四角の中に書かれる．各オブジェクトからは時系列を表す点線が引かれ，オブジェクトが実行状態であるときは実行仕様が四角で表現される．メッセージは実行仕様から，実行仕様に対して矢印で表現される．また，メッセージの送信，受信はそれぞれ送信イベント，受信イベントと呼ばれる

図 1 にシーケンス図の例を示す．図 1 ではオブジェクト A からオブジェクト B に向かって，“メッセージ 1” が送信されている．メッセージが送信される際，オブジェクト A が実行状態になり，実行仕様が四角で表現されている．また，オブジェクト B もメッセージを受信する際は，実行状態になるため実行仕様が四角で表現されている．シーケンス図の仕様は，UML2.0 と UML1.4 [17] で異なっており，UML2.0 の仕様ではシーケンス図において，if 節を用いることやループを記述することが可能となっている．本研究ではツールの入力とする xml ファイルの仕様に合わせて UML1.4 に対応している．

シーケンス図はソフトウェア開発においては，仕様設計の段階で使用されており，これを対象とした研究も盛んに行われている [12, 18].

2.3 XML

Extensible Markup Language(XML) [19] とは World Wide Web Consortium (W3C) によって策定，勧告された文書やデータの意味や構造を記述するためのマークアップ言語の 1 種である．マークアップ言語とは “タグ” と呼ばれる特定の文字列で地の文に情報の意味や構造，および装飾を埋め込んでいく言語のことである．XML は開発者が独自にタグを指定できるため，マークアップ言語を作成するためのメタ言語とも言われている．統一的な記法

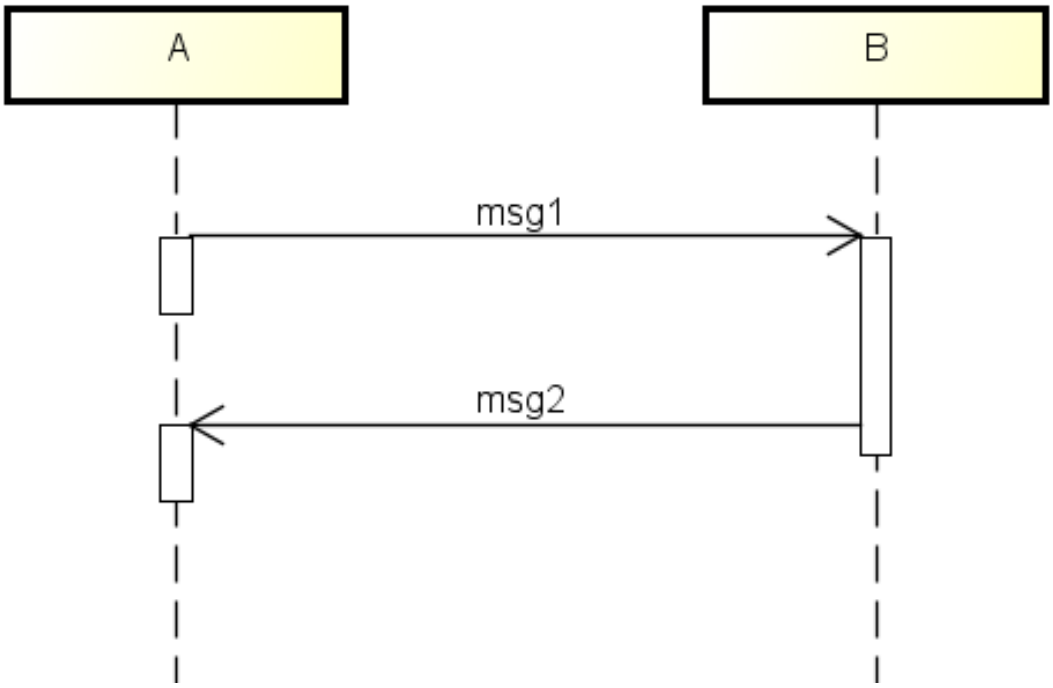

図 1: シーケンス図の例

を用いながら独自の意味や構造の定義も可能であるため，ソフトウェア間の通信に用いるデータ形式や，様々な種類のデータを保存するためのファイルフォーマットなどにも用いられている．例えば，astah* professional [20] は UML のクラス図や状態遷移図，およびシーケンス図などを XML 形式で入出力することが可能である．本研究では XML で記述されたシーケンス図を入力として用いる．

2.4 モデル検査

モデル検査とは形式的手法の 1 つである [21]．形式的手法は数学的，および論理的基盤に基づいて検査したい性質の正しさを証明する．モデル検査においては，システムを検査対象として状態遷移モデルを有限オートマトンに対応付けて表現する．モデル検査の特徴は状態遷移モデルにおけるすべての遷移系列に対して網羅的に検証を行うことである．このため，通常のソフトウェアテストなどでは発見しにくい問題も発見することが出来る．また，モデルが検査したい性質を満たさない場合，反例を得ることが出来る．反例とはモデルが検査したい性質を満たさない際の実行の流れが示されたものである．この反例を解析することで検査対象のシステムがもつ問題を発見することが出来る [22]．モデル検査を行うためのツールはモデル検査器と呼ばれており，SPIN モデルチェッカ [23] や NuSMV [24]，UPPAAL [25]

など様々なモデル検査器が提案されている．

2.4.1 SPIN モデルチェッカ

SPIN モデルチェッカとは，ソフトウェアのモデル検査を行うためのツールである．SPIN では非決定的な振る舞いを専用の記述言語 Promela で記述する．SPIN は Promela で記述された処理を網羅的に探索可能な検査器に変換する．この検査器は C 言語で記述されている．この検査器をコンパイルし，実行することで指定した性質が成立するかどうかを自動的にチェックすることが出来る．この検査器が検査する性質はラベルや表明，および性質オートマトンで指定可能である．また，SPIN には線形時相論理式を性質オートマトンに変換する機能も組み込まれている．検査の際に検査したい性質に違反した場合は，反例が生成される．

2.5 線形時相論理

線形時相論理 (Linear Temporal Logic) とは時間に関する様相をもつ様相時相論理である．様相論理はシステムの状態を時間の推移に合わせて多様に表現できる．モデル検査においては，線形時相論理 (LTL) や計算機論理 (CTL) が用いられることが多く，SPIN モデルチェッカにおいては LTL が使われる．LTL では古典的な論理演算子に加えて時間演算子が使用可能である．各演算子の説明を表 1 に示す．LTL では単項演算子 X，G，F，および二項演算子 U が使用可能である．

表 1: LTL における時間演算子の意味

文字表記	記号表記	説明
Xϕ	$\circ\phi$	ϕ が次状態で真となる
Gϕ	$\Box\phi$	ϕ は今後常に真となる
Fϕ	$\Diamond\phi$	ϕ は将来のいずれかの時点で真となる
ψUϕ	$\psi U\phi$	ϕ は将来のいずれかの時点で真となり，かつ ψ はその時点まで真である

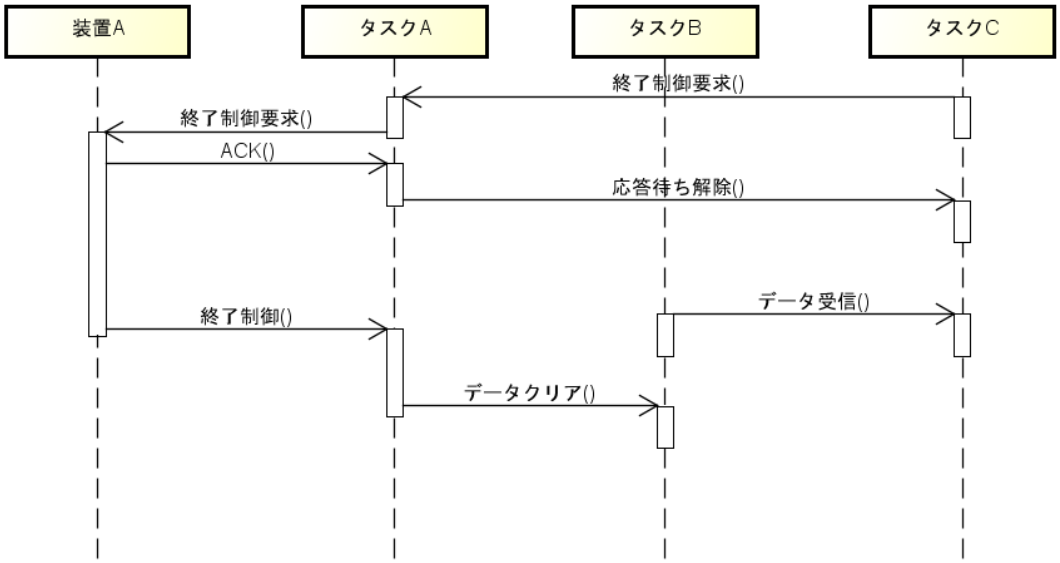

図 2: 開発現場で使われるシーケンス図の例

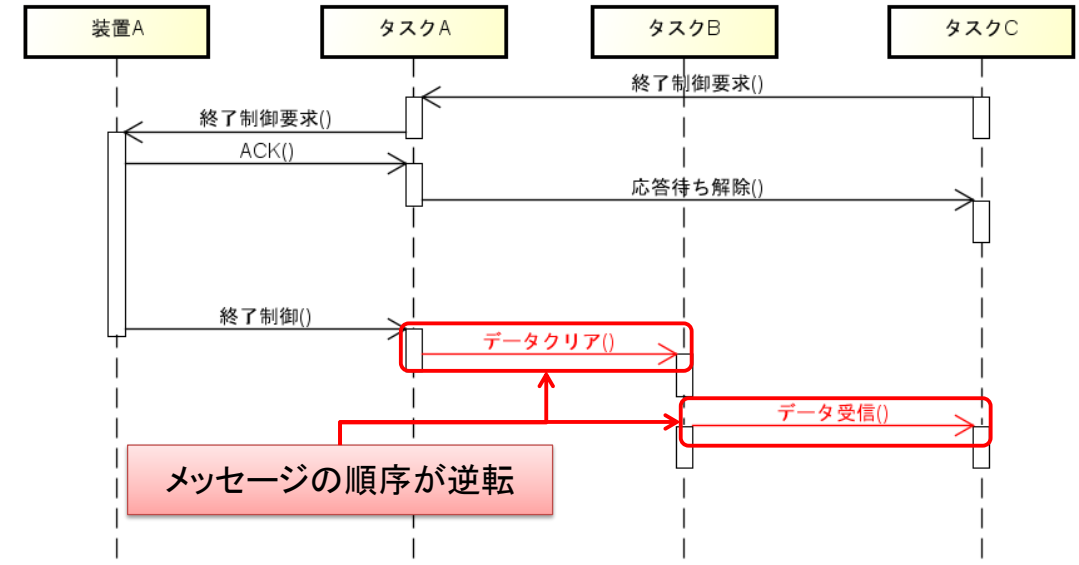

図 3: 想定されるエラーの例

3　研究動機

図 2 はソフトウェア開発において使われているシーケンス図である．この図は要求定義や仕様設計の段階で用いられるものであり，開発者はすでに欠陥は取り除いたと考えている．しかし，開発の途中で，図 3 のような欠陥が発見された．これはタスク B からタスク C へ

“データ受信” というメッセージが送られる処理と，タスク A からタスク B へ “データクリア” というメッセージが送られる処理の順序が逆転してしまうというものである．この逆転によりデータ受信が完了する前にタスク B が保持していたデータが消えてしまい，データ受信の処理が失敗するという欠陥が起こっている．

シーケンス図では，各オブジェクトが処理を行う際に各オブジェクトのライフラインが実行状態になる．例えば図 2 の装置 A においては，タスク A から “終了制御要求” のメッセージを受信する処理，タスク A に “ACK” のメッセージを送信する処理，タスク A に “終了制御” のメッセージを送信する処理の 3 つの処理が 1 つの実行仕様で行われている．また，タスク B においてはタスク C に “データ受信” のメッセージを送信する処理が 1 つの実行仕様で，タスク A から “データクリア” のメッセージを受信する処理が 1 つの実行仕様で行われており，2 つの処理が異なる実行仕様で行われている．このような異なる実行仕様で行われる処理の間に依存関係がない場合は並列に実行されるように実装されることがある．そのように実装された場合，非同期モデルのシーケンス図では各処理の依存関係が存在しないために処理が行われる順序の逆転が発生してしまうのである．

本研究では，図 2 のようなある程度設計が進んだシーケンス図から “メッセージ順序の曖昧性” を除去する手法を提案する．

4　提案手法

4.1　概要

提案手法は半自動的にシーケンス図の修正を行い，シーケンス図が持つメッセージ順序の曖昧性を除去する．手法の入力はシーケンス図を表す xml ファイルで，出力は入力として与えられたシーケンス図からメッセージ順序の曖昧性を除去したシーケンス図を表す xml ファイルである．また，メッセージ順序の曖昧性はシステムの挙動に悪影響を与えない場合も存在する．これは入力として与えられる情報だけでは判断できないため，開発者に発見したメッセージ順序の曖昧性を除去するか判断してもらうようにした．そのため，提案手法は半自動的な手法となっている．手法の対象は非同期モデルを表し，メッセージ交換が行われているシーケンス図である．また，手法で用いるシーケンス図のバージョンは UML1.4 [17] とした．提案手法は図 4 のような流れで，以下の 4 つの STEP で行われる．

STEP-1: モデル記述生成

STEP-2: 検査式生成

STEP-3: 修正候補群導出

STEP-4: シーケンス図修正

以降では，行う処理を各 STEP ごとに説明する．

4.2　STEP-1: モデル記述生成

この STEP では，入力として与えられたシーケンス図を表す xml ファイルからモデル検査で用いるモデル記述を生成する．提案手法が用いる xml ファイルは astah* professional [20] を用いて取得する．また，モデル記述は SPIN モデルチェッカ [23] で用いる promela 言語で記述されたものを生成する．生成方法は Lima らの手法 [26] を参考にしており，シーケンス図における表現から表 2 に従って promela における記述が生成される

図 5 を例として説明する．図 5 の上部はシーケンス図，下部はシーケンス図から生成したモデル記述を略記したものである．シーケンス図に含まれるオブジェクト B には 3 つの実行仕様が存在するので，プロセス B-1，プロセス B-2，プロセス B-3 としてそれぞれモデル記述が生成されている．また，オブジェクト B からオブジェクト C に対しての “msg1” の送信は，プロセス B-1 では 8 行目のオブジェクト C に対する “msg1” の送信に変換されている．また，プロセス C-1 においては 14 行目にオブジェクト B から “msg1” を受信する処理が記述されている．

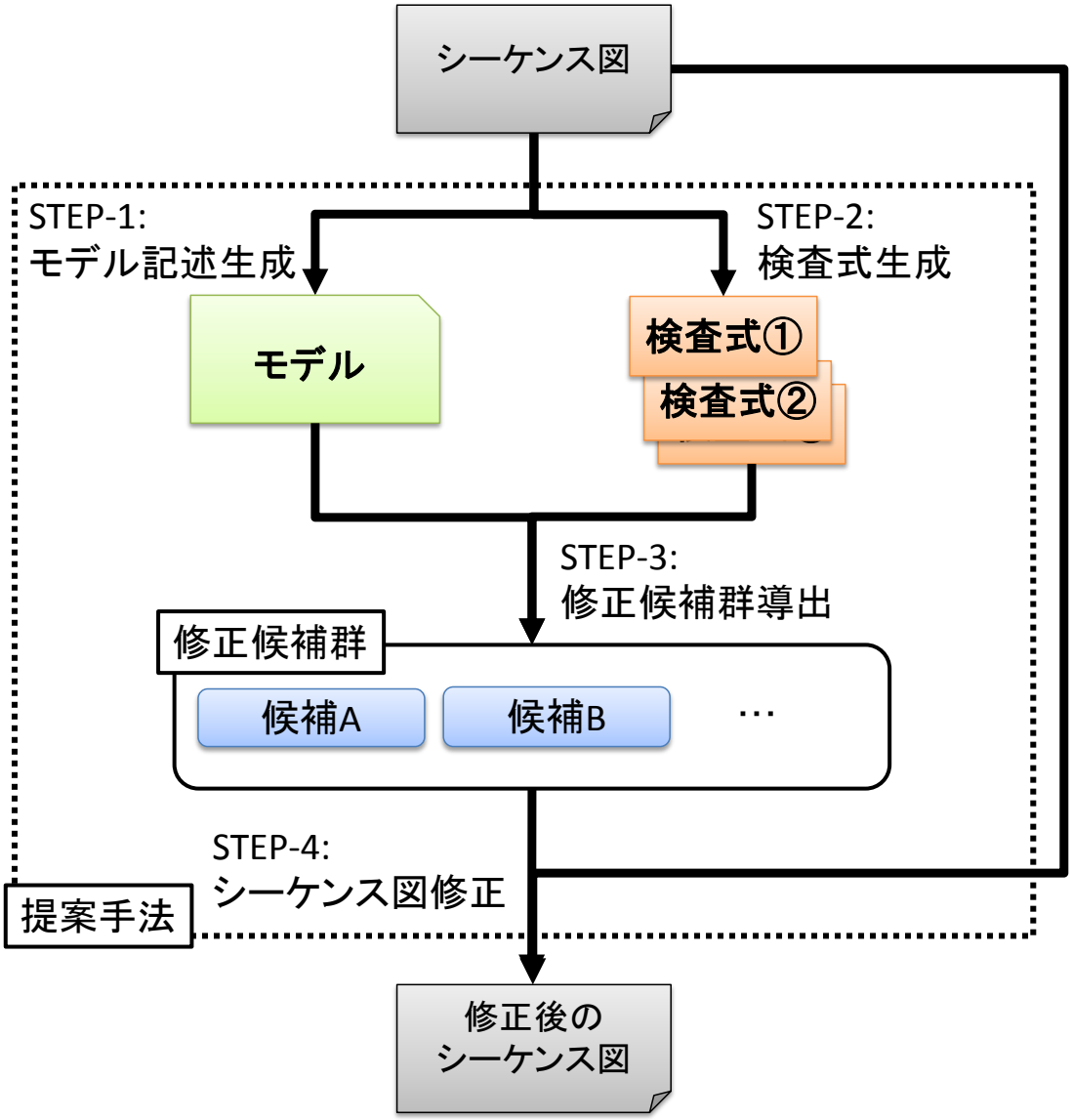

図 4: 提案手法の流れ

また，図 5 ではモデル記述を略記しているが，メッセージの送信と受信に関するモデル記述は実際には図 6 のように生成される．図 6 は図 1 から生成された promela 記述である．シーケンス図上の実行仕様は図 6 においてそれぞれ proctype の A_1，A_2，B_1 に対応する．シーケンス図におけるオブジェクト A からオブジェクト B に対する msg1 の送信は 13 行目に，msg1 の受信は 17 行目に対応する．またオブジェクト B からオブジェクト A に対する msg2 の送信は 18 行目に，msg2 の受信は 15 行目にそれぞれ対応している．

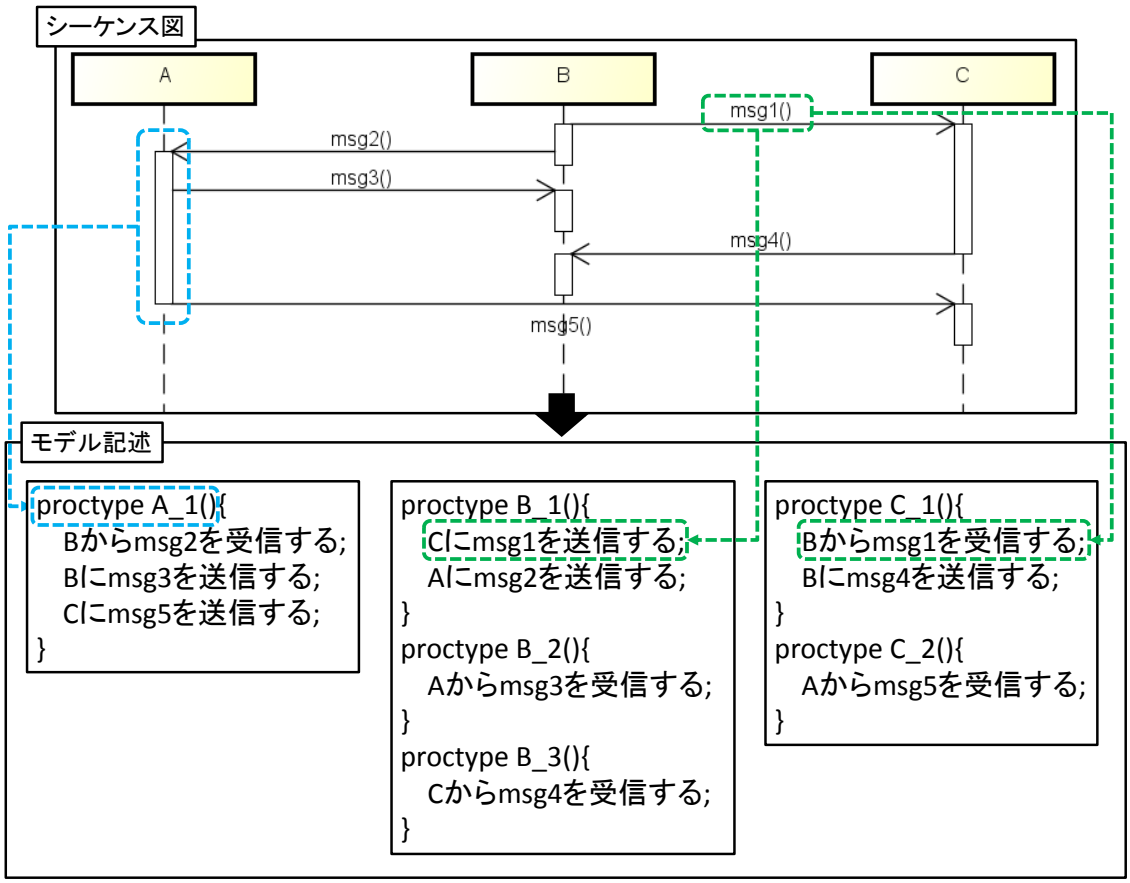

図 5: シーケンス図からモデル記述への変換例

4.3 STEP-2: 検査式生成

この STEP では，入力として与えられたシーケンス図を表す xml ファイルからモデル検査で用いる検査式を生成する．検査式は時相論理式 (LTL 式) を用いる．検査式はシーケン

表 2: シーケンス図と promela 記述の対応

シーケンス図での表現	promela の要素	promela コードでの記述
実行仕様	プロセス	proctype {...}
メッセージ (メッセージのラベル)	メッセージ	mtype={m1,...,mn}
コネクタ (メッセージの矢印)	チャネル	chan chan=[1] of {mtype}; ...; chan chann=[1] of {mtype};
送信イベント	送信	chan!m
受信イベント	受信	chan?m

```
1  /* Auto Generated Promela File */
2  /* Message Declaration */
3  mtype = {msg1, msg2};
4  /* Channel Declaration */
5  chan to_A = [20] of {mtype};
6  chan to_B = [20] of {mtype};
7  /* Variable for send and receive */
8  bool send = false;
9  bool receive = false;
10 mtype msg;
11 /* Process Declaration */
12 active proctype A_1(){
13   d_step{ send=true; receive=false; msg=msg1; to_B!msg1;} }
14 active proctype A_2(){
15   d_step{ to_A?msg2; send=false; receive=true; msg=msg2;} }
16 active proctype B_1(){
17   d_step{ to_B?msg1; send=false; receive=true; msg=msg1;}
18   d_step{ send=true; receive=false; msg=msg2; to_A!msg2;} }
```

図 6: 実際のモデル記述生成例

ス図がメッセージ順序の曖昧性を持つかどうかを検査する．隣接する全てのメッセージ組に対してメッセージ順序の曖昧性を検査するので，検査式はメッセージ，およびオブジェクトの数に応じて複数生成される．これら全ての検査式に対して検査したい性質への違反がなければ，モデルがメッセージ順序の曖昧性を持たないことが保証される．

例えば，図 5 のシーケンス図から検査式を生成する場合を考える．シーケンス図において，オブジェクト B は 4 つのメッセージ送受信を行っている．このオブジェクトに関して検査したい性質は以下のとおりである．

1. “msg1” を送信する前に，“msg2” を送信するか
2. “msg2” を送信する前に，“msg3” を受信するか
3. “msg3” を受信する前に，“msg4” を受信するか

以上の 3 つの検査したい性質に対して，それぞれ以下に示す検査式が生成される．

1. $\neg(msg2$ を送信する$)U(msg1$ を送信する$)$

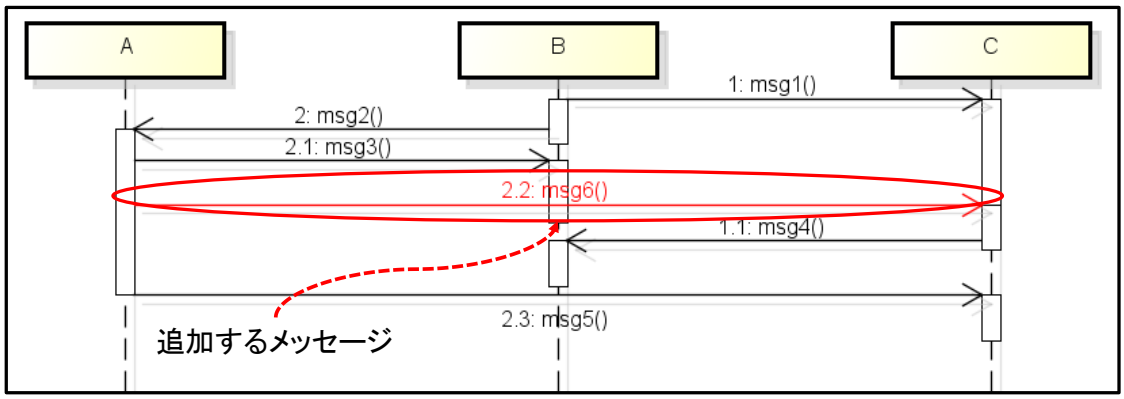

図 7: 修正候補例

2. ¬(*msg*3 を受信する)*U*(*msg*2 を送信する)

3. ¬(*msg*4 を受信する)*U*(*msg*3 を受信する)

4.4 STEP-3: 修正候補群導出

この STEP では，STEP-1 で取得したモデル記述と STEP-2 で取得した検査式を用いてモデル検査を行い，修正候補の導出を行う．修正候補とは，シーケンス図をどのように修正するかを示すものである．どのメッセージ送受信の組にメッセージ順序の曖昧性があるかの情報を保持しており，それらのメッセージ送受信の間にメッセージを追加するという修正を示す．メッセージの送信元は，メッセージを追加される組において先にメッセージ送受信を行うオブジェクトであり，メッセージの受信先は組において後にメッセージ送受信を行うオブジェクトとなる．

修正候補はモデル記述が，検査式が表す検査したい性質を満たさない場合に生成される．そのため，モデル検査の結果に応じて修正候補は複数生成される．修正候補は入力として使った検査式から導出される．検査式にはメッセージ順序の曖昧性が存在するかを判定したい対象として，2 つのメッセージ交換が書かれている．モデル検査の結果，モデルがメッセージ順序の曖昧性をもつと示されたとき，提案手法は検査式に書かれたメッセージ交換の間にメッセージの送受信を追加するという修正候補を生成する．例えば，図 7 では，モデルに対して，

$$A \text{ が “} msg3'' \text{を送信する前に, } C \text{ が “} msg4'' \text{を送信するか}$$

という性質を検査した結果，メッセージ順序の曖昧性が存在すると判定されている．そこでオブジェクト A には検査したい性質の前半に書かれている “msg3” を送信した後に，“msg6” を送信する処理が追加されている．また，オブジェクト C には検査したい性質の後半に書

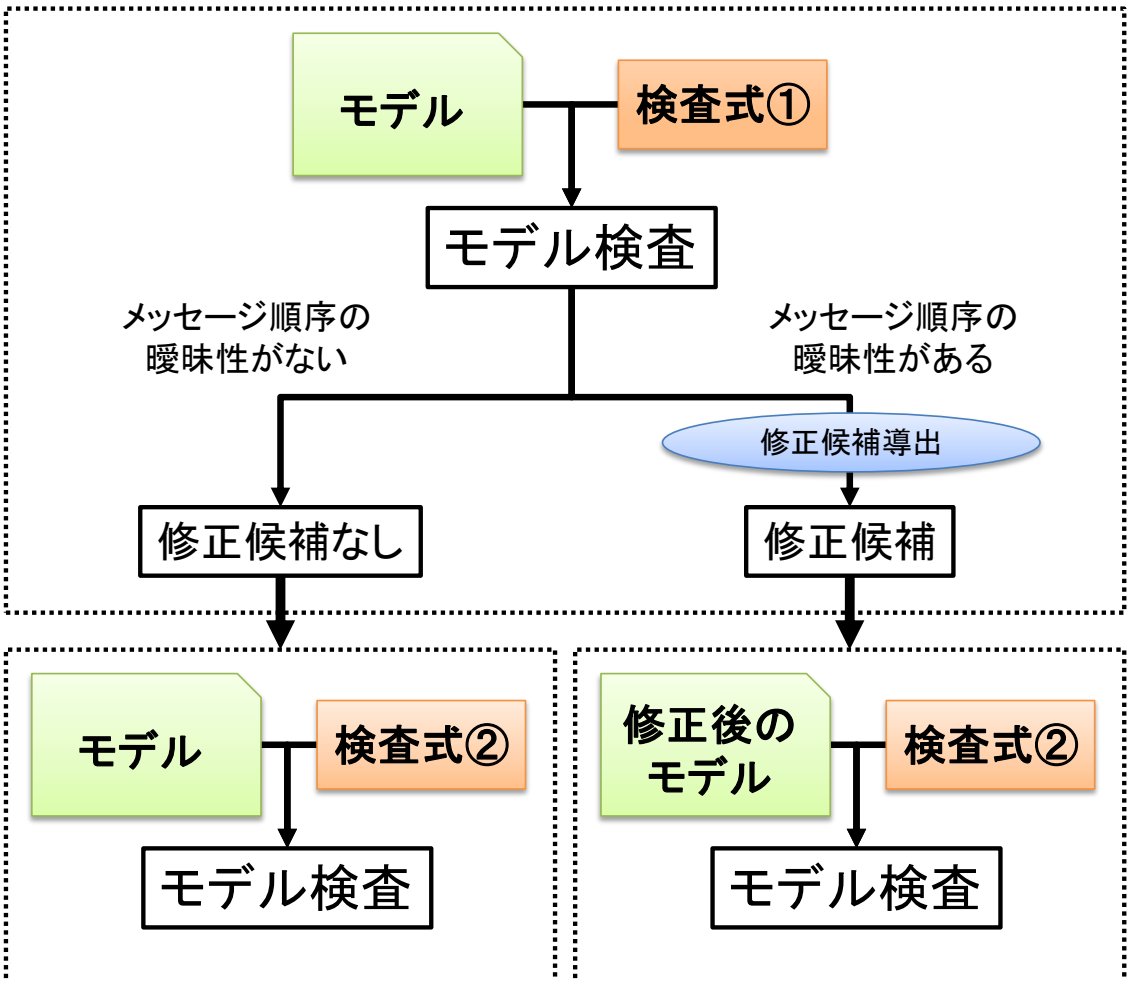

図 8: 修正候補導出の流れ

かれている “msg4” を送信する前に “msg6” を受信する処理が追加されている．

修正候補の導出は図 8 のように全ての検査式に対して行われる．まず複数存在する検査式から 1 つを選び，その検査式とモデル記述を入力として SPIN モデルチェッカでモデル検査を実行する．検査の結果に応じて，以下のように異なる処理を行う．

メッセージ順序の曖昧性がない 同じモデルと別の検査式を用いてのモデル検査に移る

メッセージ順序の曖昧性がある 入力した検査式から修正候補の導出を行う．修正候補を適用するかを開発者に判断してもらい，適用すると判断された場合はこの修正候補を保持する．また，適用するかどうかにかかわらず，同じ反例がでないようにモデルを修正する．そして修正したモデルと，別の検査式を用いて再びモデル検査を行う．

これを繰り返し，全ての検査式に対するモデル検査が終了すればこの STEP を終了する．

4.5 STEP-4: シーケンス図修正

STEP-3 で開発者に適用すると選ばれた修正候補を用いて，入力として与えられた xml ファイルに修正を加える．修正候補は追加するメッセージの名前とメッセージを追加する箇所を示しているので，メッセージの定義やシーケンス図の各ライフラインに関するメッセージ情報をそれぞれ修正する．

開発者に適用すると判断された修正候補全てに対して，xml ファイルの修正が終わればこの STEP を終了する．

5　評価

本研究では提案手法の性能を評価するために，Java 言語を用いて手法をツールとして実装した．また，実装したツールを用いて以下に示す 2 種類の評価を行った．

- 評価 1: 生成したシーケンス図による生成する性能評価
- 評価 2: 実際のシーケンス図に対して適用

なお，評価に用いた環境は以下の通りである．

- OS: Windows 7 Professional
- CPU: Intel Xeon E5607 2.27GHz × 2
- Memory: 16.0GB
- SPIN: version 6.3.2

また，SPIN モデルチェッカの状態ベクトルのサイズはデフォルトの 1024bytes とした．

以降 5.1 章から 5.2 章では各評価について，詳細に説明する．

5.1　評価方法

5.1.1　評価 1

シーケンス図に関連する既存研究 [27–31] やツール [32, 33] において，対象や例として示されているシーケンス図を収集した．また，収集したシーケンス図の内 2 つに対して，オブジェクト数，およびメッセージ数を増やす変更を加え，新たにシーケンス図を生成した． 生成したシーケンス図を図 9 に示す．図 9 は 10 を元にして生成したシーケンス図である．生成にあたっては，図 10 中の青い点線の四角で示されている箇所を複製しメッセージ数とオブジェクト数を増やすという変更を加えた．増やした箇所は図 9 においては緑色の点線の四角で示されている．同様の変更を加える形で他のシーケンス図も生成した．これらのシーケンス図に対してツールを適用し，シーケンス図に含まれるオブジェクト数やメッセージ数，およびツールが生成する修正候補の数と実行時間を計測した．

この評価を行う際は，開発者はツールが提案する修正候補を全て採用すると仮定して，実行時間を計測した．

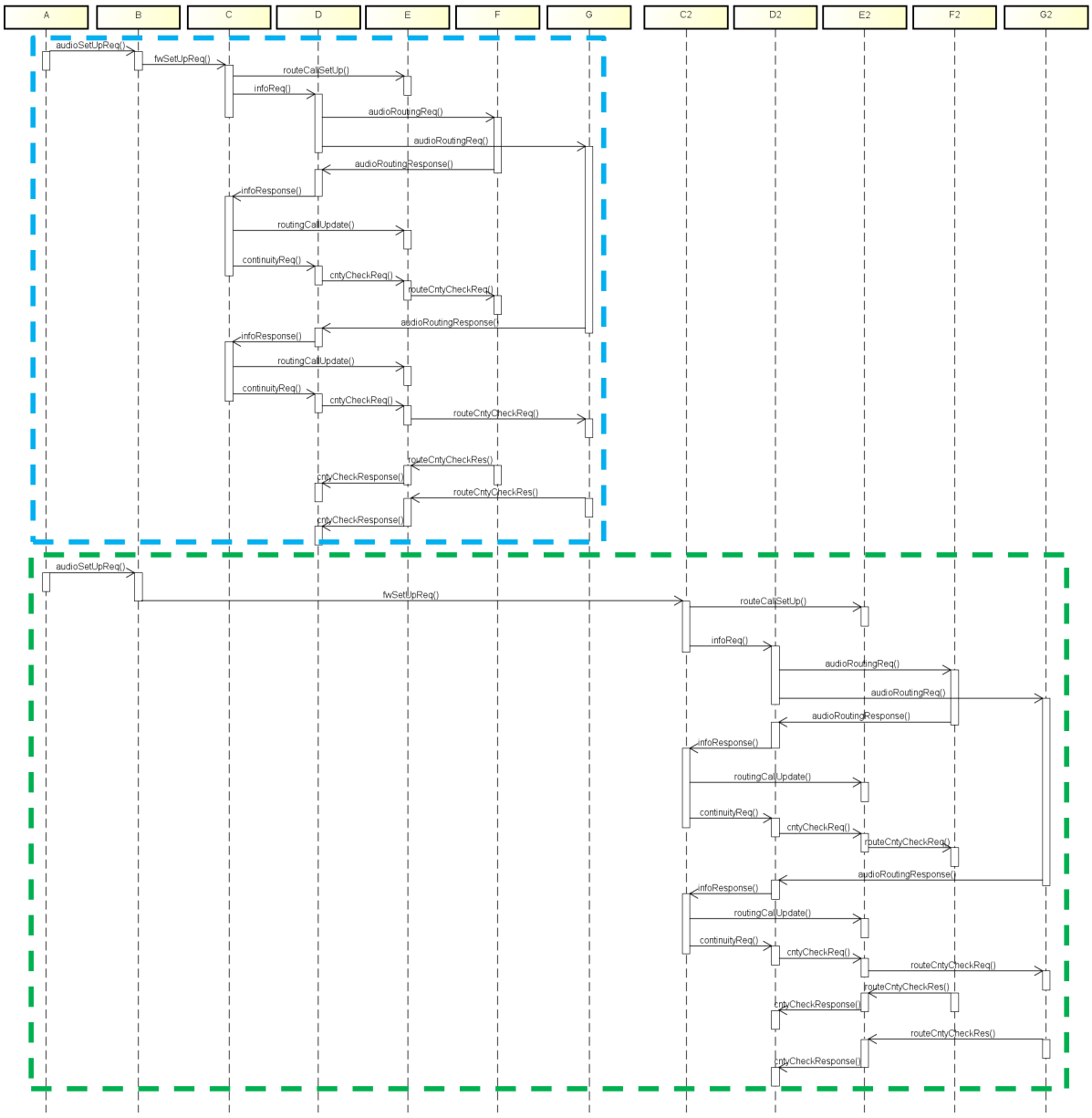

図 9: 評価 1 のために生成したシーケンス図

5.1.2 評価 2

実際の開発現場で使われているシーケンス図を表す xml ファイルを astah* professional [20] によって取得し，ツールを適用する．ツールによって図 3 に示したようなエラーシーケンスが発見できるかどうかを確かめる．また，ツールが生成する修正候補を開発者に確認してもらい，ツールが出力する修正候補が正しいかどうかを確認した．

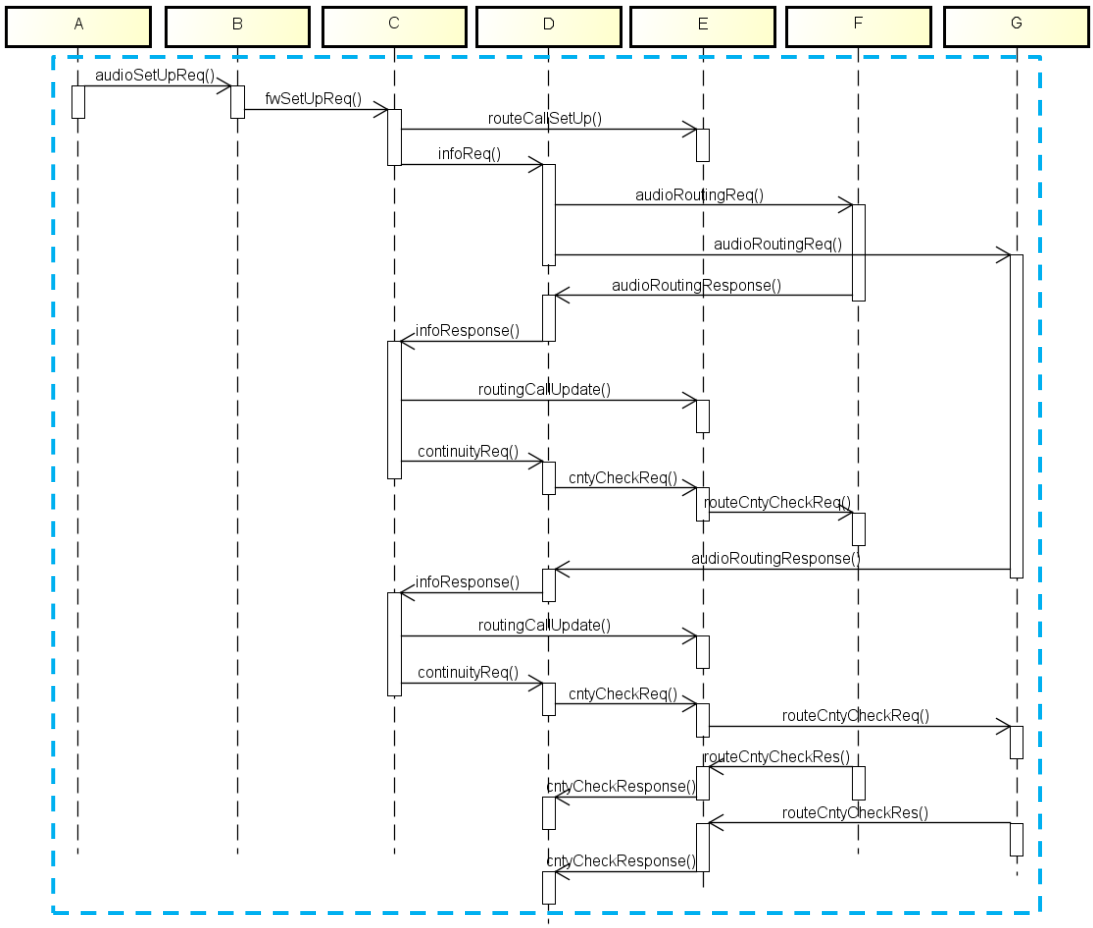

図 10: 図 9 の生成元シーケンス図

5.2 結果

5.2.1 評価 1

7 個のシーケンス図を収集し，その内の 1 つに対して変更を加え 4 個のシーケンス図を生成した．これら 11 個のシーケンス図にツールを適用した．評価の結果は表 3 のようになった．表 3 の 1-3 列目は対象としたシーケンス図の情報を，4-7 列目までは対象にツールを適用した結果を示している．表の 1 列目はシーケンス図の取得元を，2 列目の “obj” はシーケンス図に含まれるオブジェクト数を，および 3 列目の “msg” はシーケンス図に含まれるメッセージ数をそれぞれ示している．また，4 列目の “修正候補数” はツールが生成した修正候補の数を，5-7 列目はツールの適用にかかった実行時間をそれぞれ示している．実行時間に関しては STEP ごとに計測した．ただし，5 列目の STEP-1，2 に関しては，STEP-1 の処理と STEP-2 の処理が同じシーケンス図を入力として，並行して処理が行われるため 2 つの

STEP の実行時間を合わせて計測している．

STEP-1，2 の実行時間は対象による大きな差はないが，[31]-2,3,4 の値は比較的大きくなっている．これはシーケンス図に含まれるオブジェクトの数やメッセージの数が大きいためだと考えられる．また，STEP-3 の実行時間はシーケンス図に含まれるオブジェクト数やメッセージ数の値が大きくなるほど，永くなっていることがわかる．STEP-4 の実行時間についても大きな差はないが，生成した修正候補の数に応じて大きくなっていることが考えられる．

表 3 において [31]-4 の STEP-3 の実行時間はオブジェクト数やメッセージ数の値が非常に大きいにもかかわらず，[31]-3 の値に比べて非常に小さくなっている．これは [31]-4 のシーケンス図を表すモデルを用いてモデル検査を行った際にメモリ不足でモデル検査が行えなかったためである．そのため，修正候補数の値も 0 となっている．

5.2.2 評価 2

ソフトウェアの開発現場で実際に使われるシーケンス図を 4 つ提供してもらい，その中でメッセージ順序の曖昧性に関するエラーシーケンスが想定される 2 つに対してツールを適用した． ツールを適用したシーケンス図を，図 11 と図 12 に示す．ツールを適用した結果は表 4 のようになった．表 4 が含む情報は表 3 と同様になっている．

図 11 のシーケンス図にツールを適用する流れを説明する．STEP-1 の結果，付録 I に示す promela 記述が得られた．

表 3: 評価 1 の結果

	obj	msg	修正候補数	実行時間 (秒)		
				STEP-1,2	STEP-3	STEP4
[27]	7	11	3	0.44	20.21	0.43
[28]	4	12	2	0.42	25.19	0.41
[29]	3	8	0	0.39	15.55	0.41
[30]	3	4	0	0.37	5.96	0.40
[32]	5	6	1	0.42	8.95	0.41
[33]	6	24	0	0.49	56.15	0.42
[31]	7	22	6	0.46	52.08	0.48
[31]-1	12	44	9	0.58	124.77	0.45
[31]-2	22	88	18	0.98	9136.59	0.52
[31]-3	32	132	24	0.97	14054.17	0.56
[31]-4	37	154	0	1.20	744.73	0.56

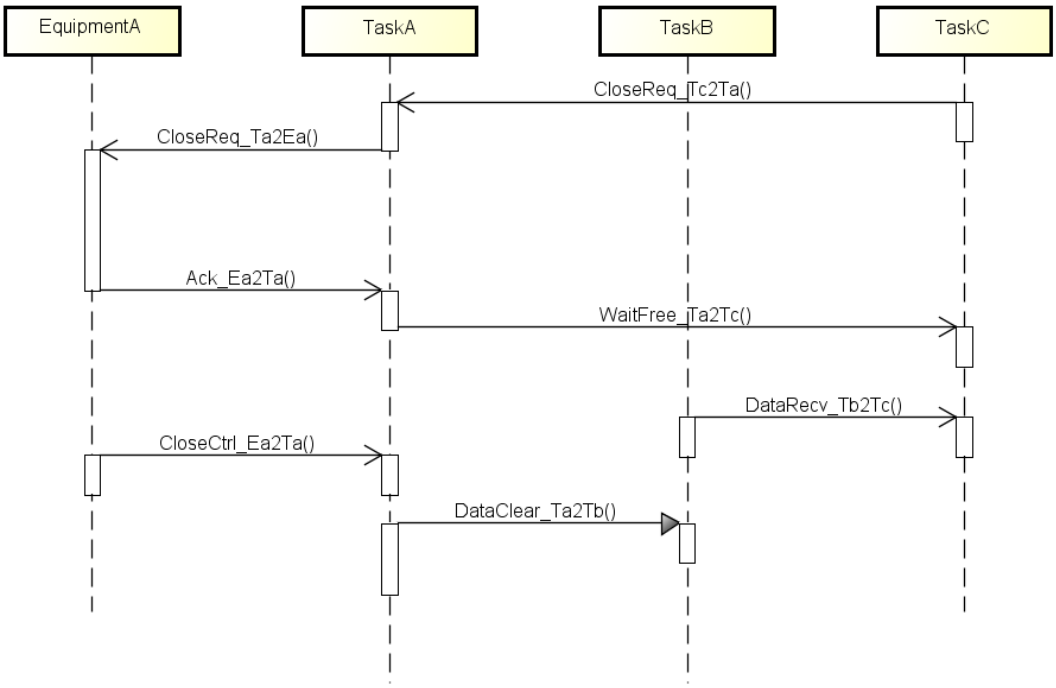

図 11: 開発現場で使われるシーケンス図の例 1

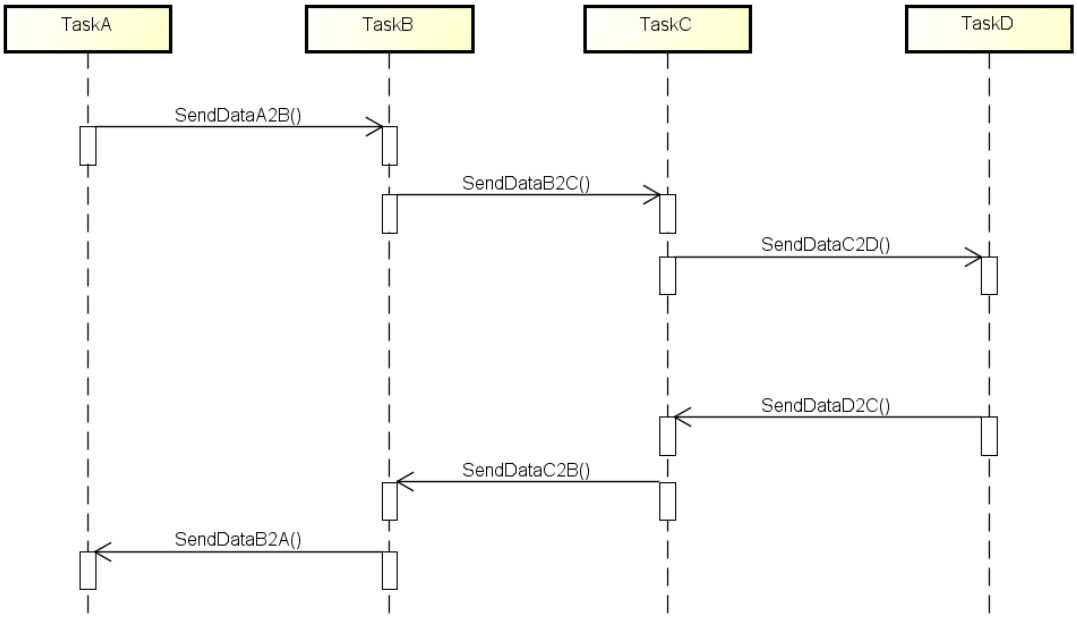

図 12: 開発現場で使われるシーケンス図の例 2

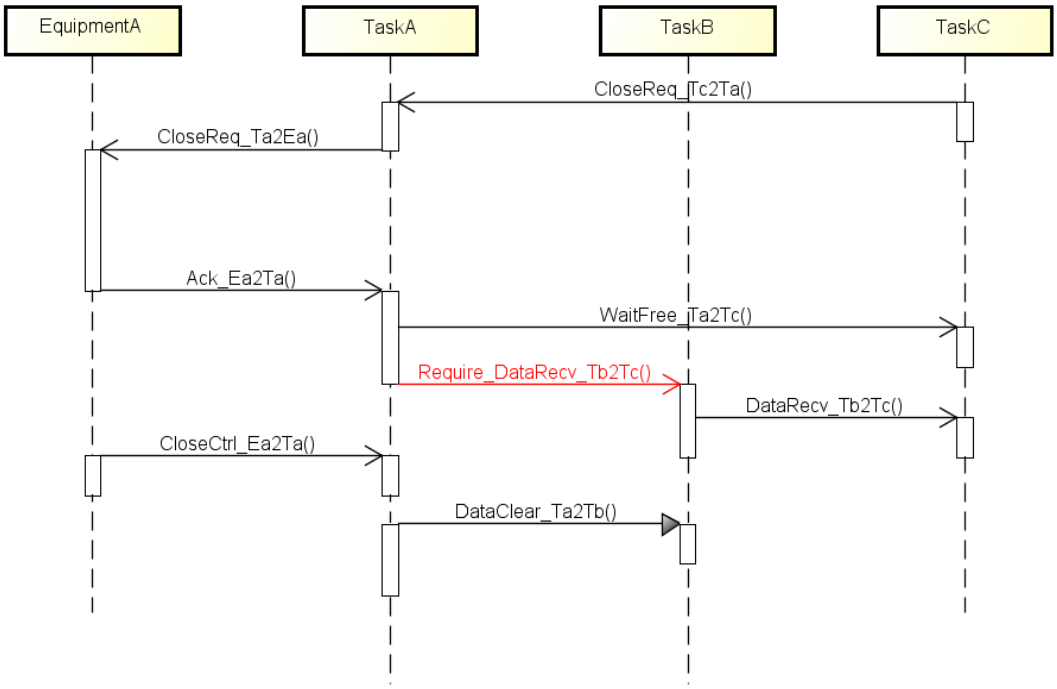

図 13: 修正候補 1

ツールによって得られた 10 個の修正候補を図 13-22 に示す．図 16 を例として，修正候補について詳細に説明する．図 16 が示す修正候補は，“TaskC” が “DataRecv_Tb2Tc” のメッセージを受信する処理と，“TaskA” が “DataClear_Ta2Tb” のメッセージを送信する処理の間に，“TaskC” が “Require_DataClear_Ta2Tb” のメッセージをを送信する処理と，“TaskA” がそのメッセージを受信する処理を追加する修正を示している．この修正候補は図 3 が示した “データ受信” のメッセージ交換と “データクリア” のメッセージ交換の順序が逆転してしまうエラーシーケンスを元に生成されている．よって，提案手法は図 3 のような欠陥を発見できるといえる．

表 4: 評価 2 の結果

	obj	msg	修正候補数	実行時間 (秒)		
				STEP-1,2	STEP-3	STEP4
図 11	4	7	5	0.40	14.04	0.62
図 12	4	6	5	0.41	12.15	0.52

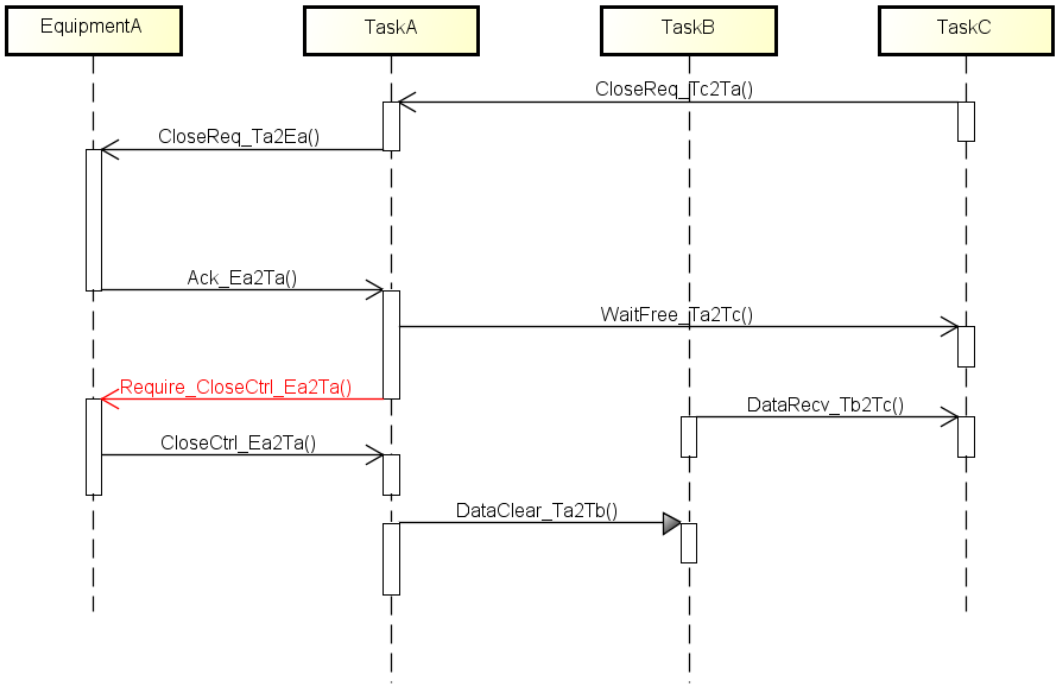

図 14: 修正候補 2

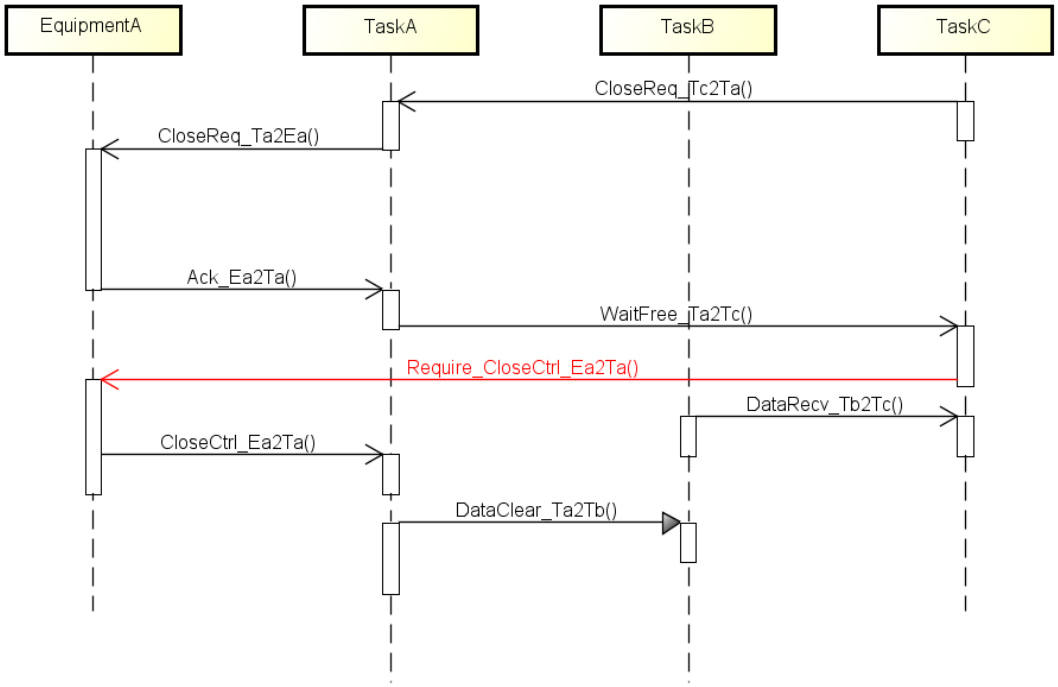

図 15: 修正候補 3

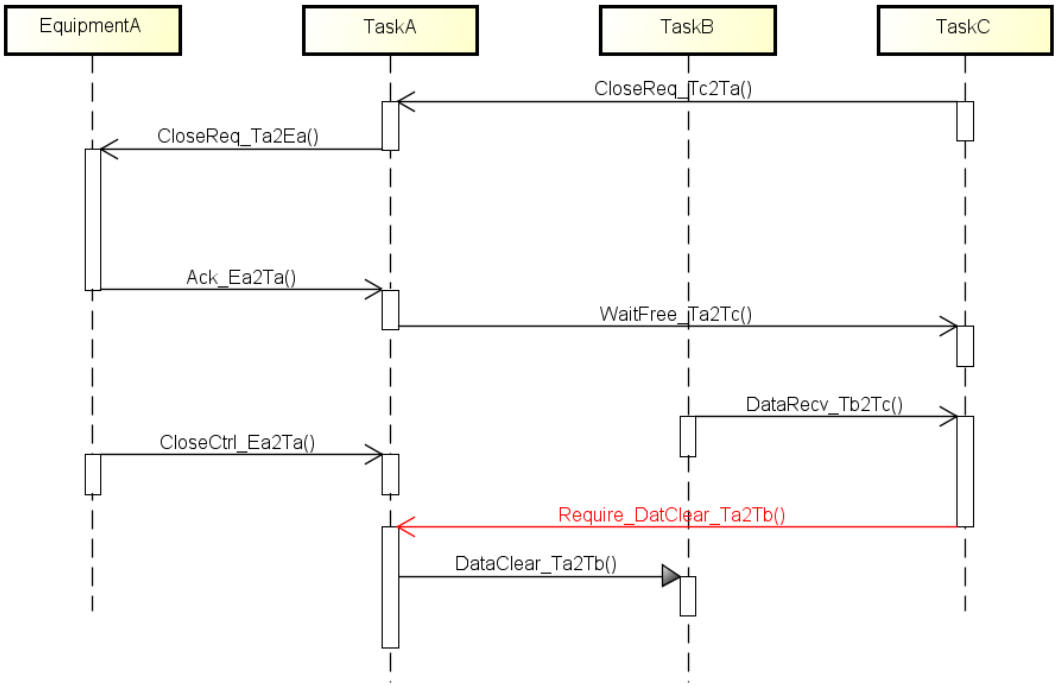

図 16: 修正候補 4

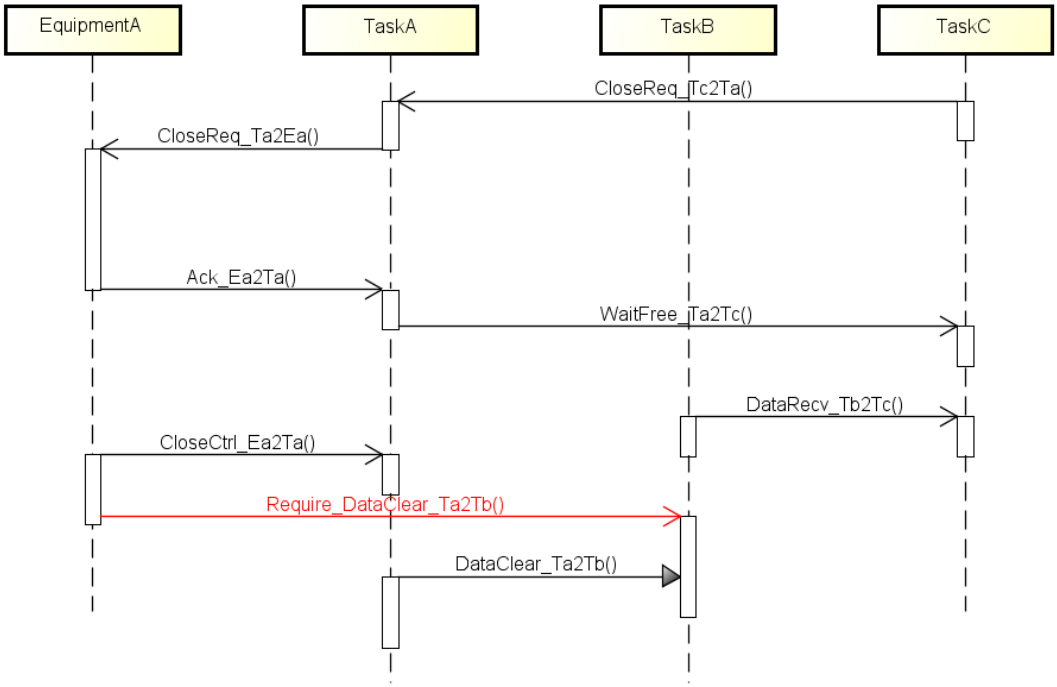

図 17: 修正候補 5

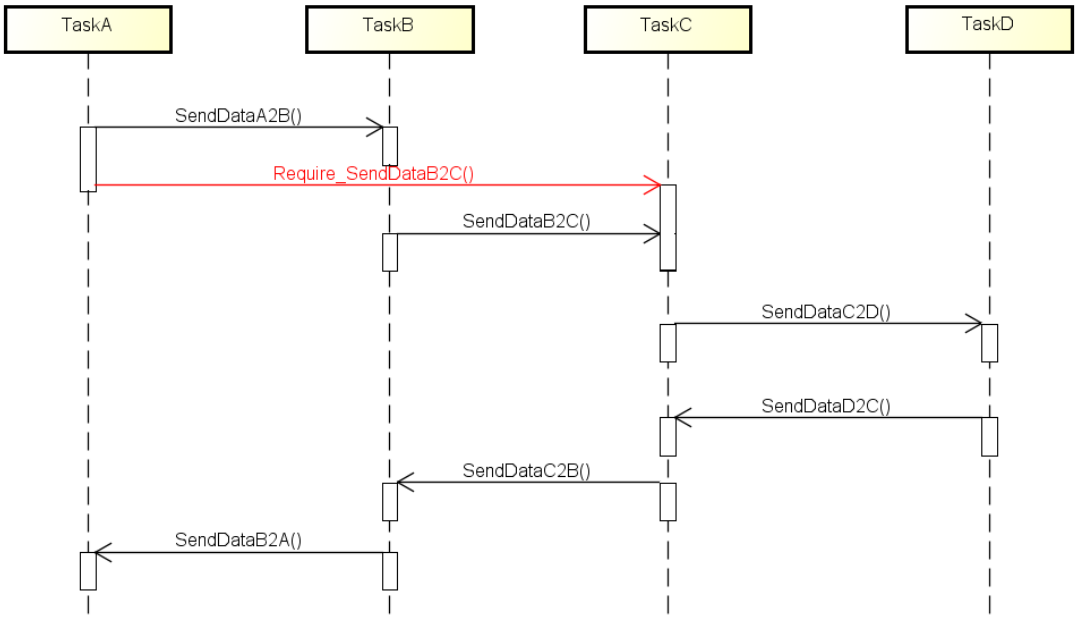

図 18: 修正候補 6

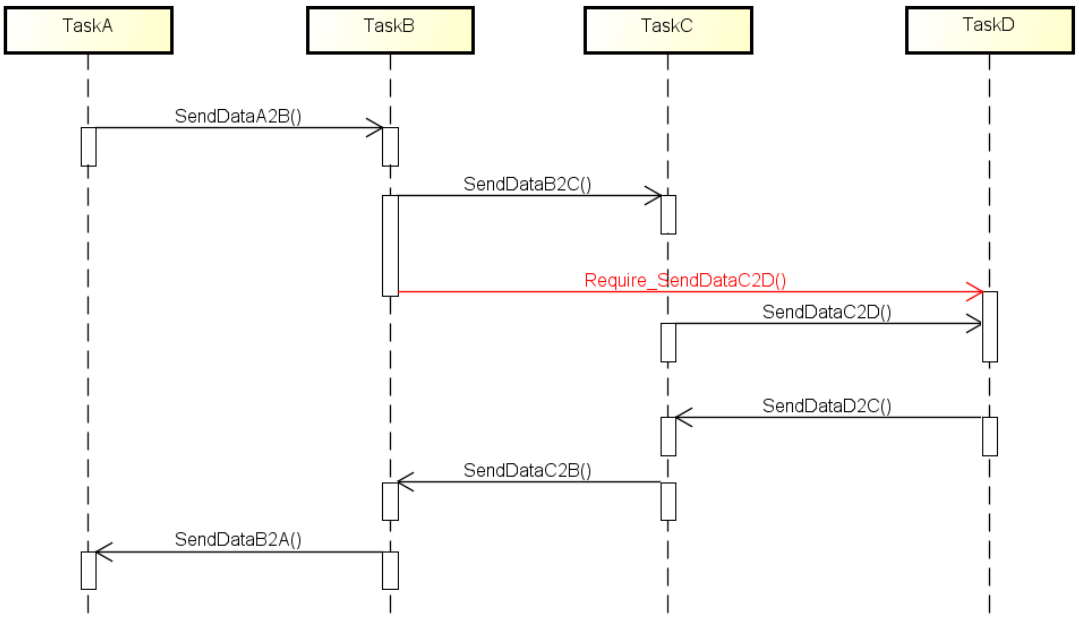

図 19: 修正候補 7

6　妥当性の脅威

評価 1 では既存研究やツールからシーケンス図を収集し，ツールを適用することでツールが生成する修正候補の数や実行時間を評価した．今回，手法を適用したシーケンス図は 11

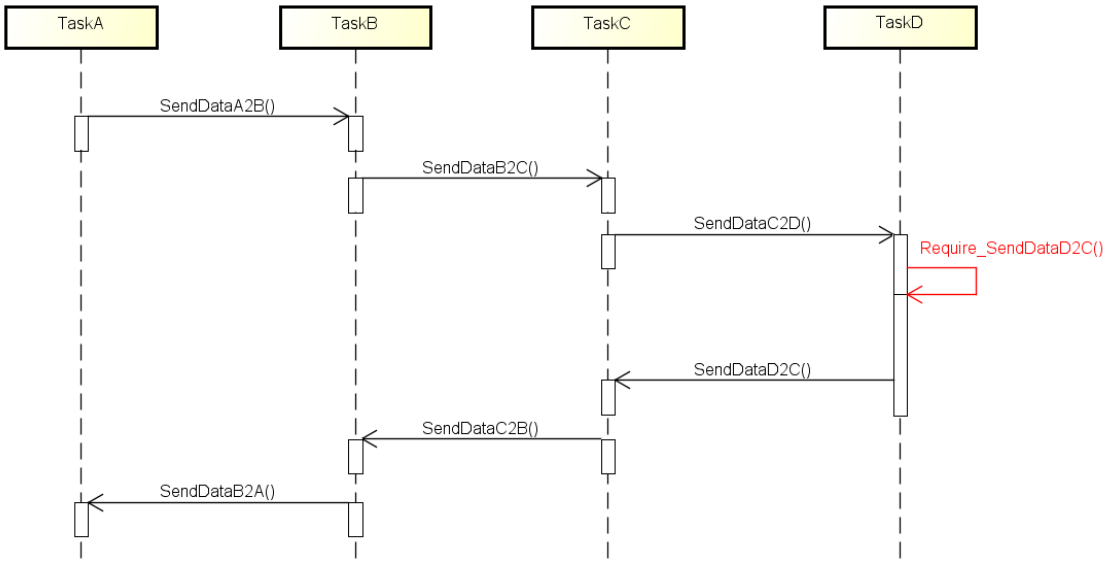

図 20: 修正候補 8

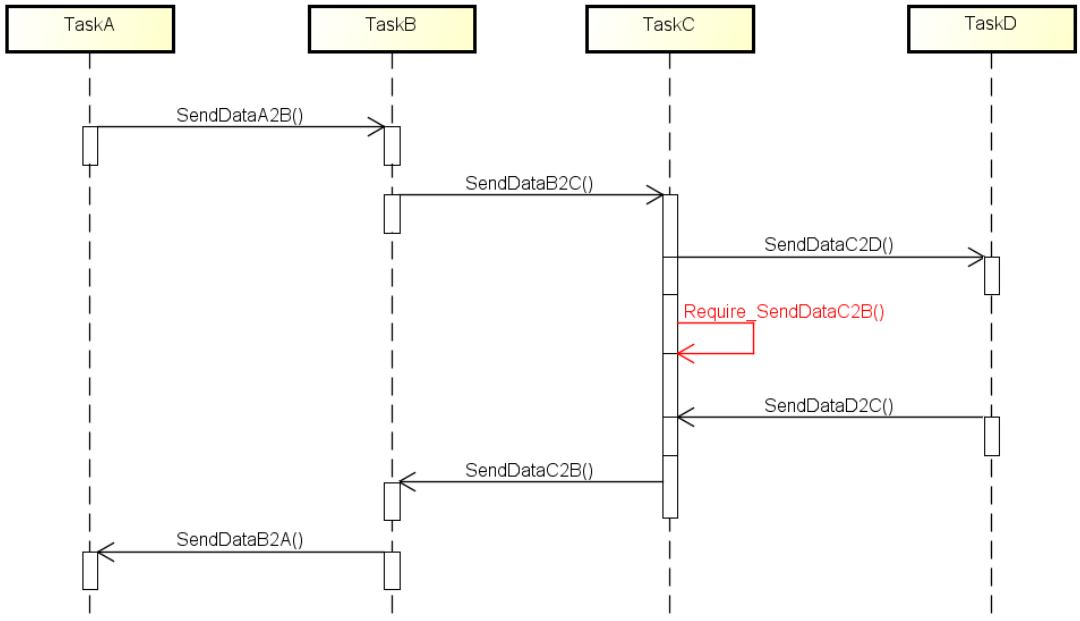

図 21: 修正候補 9

個であり，またシーケンス図に含まれるオブジェクト数やメッセージ数も限られた規模のものとなっている．そのため，規模の異なるシーケンス図を評価に用いた場合は異なる結果が得られる可能性がある．

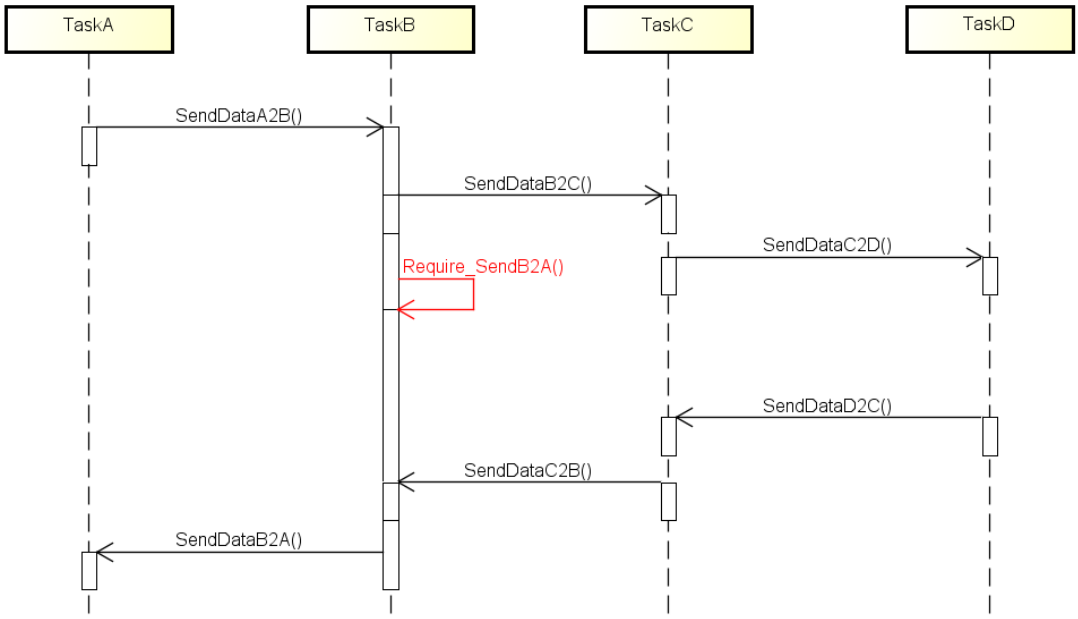

図 22: 修正候補 10

次に，評価 2 では，実際に開発現場で発生した事態を元に想定されたエラーシーケンスを発生させる可能性を持ったシーケンス図を用いて，提案手法がエラーシーケンスを発見し，修正候補を提示できるかを確認した．今回の評価では対象が 2 つであるため，より一般的な結果を得るためにはより多くのシーケンス図を開発現場から取得し，評価を行う必要がある．

7　関連研究

Lima らや Amirat らは，シーケンス図から promela 記述を生成し，モデル検査によって欠陥を早期発見する手法を提案している [6,26]．Lima らの手法では UML2.0 の形式で書かれたほとんどのシーケンス図に対して，promela 言語における表現形式を提案している．また，この生成方法を eclipse プラグインとして実装し，適切な検査式を与えることで欠陥を発見できることを確認している．Amirat らも同じくシーケンス図の要素を promela において表現する形式を定義している．また Amirat らは AToM3 [34] を用いて，シーケンス図から promela 記述への変換を実装している．

宮本らは作成された状態遷移図と配置図で記述されたプログラムの仕様を promela 言語で記述された仕様に変換する手法を提案している [11]．この手法では，astah* professional が出力した状態遷移図と配置図を示す xml ファイルを入力として，配置図に含まれるインスタンスをプロセスに，状態遷移図に含まれる状態遷移を各プロセスが行う処理に変換することで promela ファイルを生成する．また，UML に含まれる仕様パターンの記述を LTL 式に変換することで，複雑な式を書くことなく SPIN でモデル検査を実行することを可能にしている．

山田らはソフトウェアの仕様を記述したアクティビティ図を変換し，promela 記述を自動生成する手法を提案している [7]．この研究では astah* professional [20] で記述したアクティビティ図の xml ファイルに含まれる各要素を promela 言語でどのように記述するかの形式を述べるとともにその変換を php 言語で変換器として実装している．また，実装した変換器を評価するために，Ajax を用いた社員管理システムの画面遷移設計に対して適用実験を行っている．

Tiwari らはアクティビティ図を用いてテストケースを生成する手法を提案している [35]．この手法は，システムの処理の流れを表現するアクティビティ図からシステムが正常終了する条件を取得，その条件を反転してシステムの故障木図を取得する．これらを変換することでシステムの正常終了時のテストケース，および異常終了時のテストケースを生成する．

長田らはシーケンス図などで記述された通信プロトコルの仕様から，通信プログラムのソースコードを生成する手法を提案している [36]．この手法では，シーケンス図やメッセージ形式定義言語で通信プロトコルを定義し，定義から通信プログラムを構築する．また構築の際，故障木図から例外処理を導出し，正常時処理に付加する．これにより，通信プログラムが例外発生時に行うべき例外処理の実装漏れを削減している．

Kaleeswaran らはプログラムとテストスイートから欠陥を検出し，その欠陥に対する修正候補を提示する手法を提案している [1]．この手法は欠陥箇所特定ツール Zotlar [37] を用いて特定した欠陥箇所に対して，ソースコードを変化させて生成した修正候補を複数生成し，

開発者に提示する．開発者は提示された修正候補から選んで修正を行うため，半自動的な手法となっている．

8　あとがき

本研究では，非同期モデルでメッセージ交換が行われるシーケンス図を対象として，そのようなシーケンス図が持つメッセージ順序の曖昧性を除去する手法を提案した．手法は，まず既存研究の方法を参考にしてシーケンス図からモデル検査用コードを生成する．また，シーケンス図に含まれる各メッセージに対して，メッセージ順序の曖昧性を検査する検査式を生成する．次に，生成したモデルと検査式を用いてモデル検査に基づく方法で修正候補を導出する．ここで導出した修正候補は適用するかどうかを開発者に選んでもらっている．最後に適用すると選ばれた全ての修正候補が示す修正を入力のシーケンス図に加え，シーケンス図を自動修正する．

また，2 つの評価を行った．1 つの評価では提案手法が実際のシーケンス図に含まれるメッセージ順序の曖昧性を検出，および修正できることを確認した．もう 1 つの評価では提案手法が生成する修正候補の数や実行時間を計測し，評価を行った．今後の課題としては，以下の 3 点である．

- より多くの実際に開発で使用されるシーケンス図に提案手法を適用する．
- 手法が対応できる欠陥の種類を増やす.
- 手法が示す修正候補のバリエーションを増やす.

謝辞

本研究を行うにあたり，日頃より理解あるご指導，ご助言を賜り，多大な励ましを頂きました楠本 真二 教授に深く感謝を申し上げます.

本研究の全過程を通して，細部にわたる貴重な御指摘と熱心かつ丁寧なご指導を頂きました岡野　浩三 准教授に心より感謝の意を表します.

本研究に多大なるご助言，およびご協力を頂きました井垣 宏 特任准教授に深く感謝致します.

本研究に関して，的確かつ有益なご指摘，およびご助言を頂きました肥後 芳樹 助教に心より感謝を申し上げます.

本研究に対し，アイデアや評価実験の対象を提供していただいた三菱電機　原内　聡様と村上　享平様に厚くお礼申し上げます

本研究において，様々な形でご助言やご協力，励ましを頂きました楠本研究室の皆様に深く感謝致します.

また，本研究に至るまでの様々な演習や講義，実習等におきましてご指導を頂きました大阪大学基礎工学部情報科学科の諸先生方にこの場を借りて，心より御礼を申し上げます.

参考文献

[1] Shalini Kaleeswaran, Varun Tulsian, Aditya Kanade, and Alessandro Orso. MintHint: Automated Synthesis of Repair Hints. In *the Proceedings of the 36th International Conference on Software Engineering*, pp. 266–276. ACM, June 2014.

[2] Westley Weimer, ThanhVu Nguyen, Claire Le Goues, and Stephanie Forrest. Automatically Finding Patches Using Genetic Programming. In *the Proceedings of the 31st International Conference on Software Engineering*, pp. 364–374. IEEE Computer Society, May 2009.

[3] Hoang Duong Thien Nguyen, Dawei Qi, Abhik Roychoudhury, and Satish Chandra. SemFix: Program Repair via Semantic Analysis. In *the Proceedings of the 35th International Conference on Software Engineering*, pp. 772–781. IEEE Press, May 2013.

[4] Yuhua Qi, Xiaoguang Mao, Yan Lei, Ziying Dai, and Chengsong Wang. The Strength of Random Search on Automated Program Repair. In *the Proceedings of the 36th International Conference on Software Engineering*, pp. 254–265, June 2014.

[5] Robert H Bourdeau and Betty HC Cheng. A formal semantics for object model diagrams. *IEEE Transactions on Software Engineering*, Vol. 21, No. 10, pp. 799–821, 1995.

[6] A. Amirat, A. Menasria, M.A. Oubelli, and N. Younsi. Automatic Generation of PROMELA code from Sequence Diagram with Imbricate Combined Fragments. In *the Proceedings of 2012 Second International Conference on Innovative Computing Technology*, pp. 111–116, Sep. 2012.

[7] 山田豊, 和崎克己. 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と Web アプリケーション設計への適用. 電子情報通信学会技術研究報告. SWIM, ソフトウェアインタプライズモデリング, Vol. 110, No. 427, pp. 23–28, Feb. 2011.

[8] Nianhua Yang, Xinshun Guo, and Wenjie Wang. Formal Verification of a UML State Chart Diagram with Uppaal. *International Journal of Hybrid Information Technology*, Vol. 5, No. 4, pp. 55–60, 2012.

[9] Mouna Ait_Oubelli, Nadia Younsi, Abdelkrim Amirat, and Ahcene Menasria. From UML 2.0 Sequence Diagrams to PROMELA code by Graph Transformation Using

AToM 3, 2011.

[10] Yefei Zhao, Zongyuan Yang, Jinkui Xie, and Qiang Liu. Quantitative Analysis of System Based on Extended UML State Diagrams and Probabilistic Model Checking. *Journal of Software*, Vol. 5, No. 7, pp. 793–800, 2010.

[11] 宮本直樹, 和崎克己. UML 記述の仕様から SPIN モデル検査用 PROMELA モデルへの自動変換. 情報科学技術フォーラム講演論文集, Vol. 9, No. 1, pp. 311–314, Aug. 2010.

[12] 宮本直樹, 和崎克己. Uml シーケンス図の構造記述から線形時相論理式への自動変換手法. 情報科学技術フォーラム講演論文集, Vol. 10, No. 1, pp. 311–314, Sep. 2011.

[13] Martin Fowler. *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional, 2004.

[14] UML2.0 Superstructure Specification. http://www.omg.org/cgi-bin/doc?formal/05-07-04.

[15] Object Management Group. Meta Object Facility (MOF) 2.0 Core Specification, 2004. http://www.omg.org/docs/ptc/03-10-04.pdf.

[16] Object Management Group. Queries/Views/Transformations. http://www.omg.org/spec/QVT/1.1/PDF/.

[17] UML1.4. http://www.omg.org/spec/UML/1.4/.

[18] Stefan Leue and Peter B Ladkin. Implementing and Verifying MSC Specifications Using PROMELA/XSPIN. In *the Proceedings of the 2nd International Workshop on the SPIN Verification System*, Vol. 32. American Mathematical Society, 1997.

[19] Tim Bray, Jean Paoli, C Michael Sperberg-McQueen, Eve Maler, and François Yergeau. Extensible markup language (XML). *World Wide Web Consortium Recommendation*, p. 16, 1998.

[20] astah* professional. http://astah.change-vision.com/ja/product/astah-professional.html.

[21] Edmund M Clarke, Orna Grumberg, and Doron Peled. *Model checking*. MIT press, 1999.

[22] Thomas Ball, Mayur Naik, and Sriram K. Rajamani. From Symptom to Cause: Localizing Errors in Counterexample Traces. In *the Proceedings of the 30th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '03, pp. 97–105. ACM, Jun. 2003.

[23] Gerard J Holzmann. The model checker SPIN. *IEEE Transactions on software engineering*, Vol. 23, No. 5, pp. 279–295, 1997.

[24] Alessandro Cimatti, Edmund M. Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In *the Proceedings of the 14th International Conference on Computer Aided Verification*, CAV '02, pp. 359–364. Springer, July 2002.

[25] Kim G Larsen, Paul Pettersson, and Wang Yi. Uppaal in a Nutshell. *International Journal on Software Tools for Technology Transfer*, Vol. 1, No. 1, pp. 134–152, 1997.

[26] Vitor Lima, Chamseddine Talhi, Djedjiga Mouheb, Mourad Debbabi, Lingyu Wang, and Makan Pourzandi. Formal Verification and Validation of UML 2.0 Sequence Diagrams using Source and Destination of Messages. *Electronic Notes in Theoretical Computer Science*, Vol. 254, pp. 143–160, 2009.

[27] Paul Baker, Paul Bristow, Clive Jervis, David King, Robert Thomson, Bill Mitchell, and Simon Burton. Detecting and Resolving Semantic Pathologies in UML Sequence Diagrams. In *the Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering*, pp. 50–59. ACM, Sep. 2005.

[28] Simona Bernardi, Susanna Donatelli, and José Merseguer. From UML Sequence Diagrams and Statecharts to Analysable Petri Net models. In *the Proceedings of the 3rd international workshop on Software and performance*, pp. 35–45. ACM, July 2002.

[29] David Harel and Shahar Maoz. Assert and Negate Revisited: Modal Semantics for UML Sequence Diagrams. *Software & Systems Modeling*, Vol. 7, No. 2, pp. 237–252, 2008.

[30] Hui Shen, Ram Krishnan, Rocky Slavin, and Jianwei Niu. Sequence Diagram Aided Privacy Policy Specification. *IEEE Transactions on Dependable and Secure Computing*, pp. 1–1, 2014.

[31] Bill Mitchell. Characterizing Communication Channel Deadlocks in Sequence Diagrams. *IEEE Transactions on Software Engineering*, Vol. 34, No. 3, pp. 305–320, 2008.

[32] Lucidchart. https://www.lucidchart.com/.

[33] tracemodeler. http://www.tracemodeler.com/.

[34] Juan de Lara and Hans Vangheluwe. AToM3: A Tool for Multi-formalism and Metamodelling. In *the Proceedings of the 5th International Conference on Fundamental Approaches to Software Engineering*, FASE '02, pp. 174–188. Springer-Verlag, Apr. 2002.

[35] Saurabh Tiwari and Atul Gupta. An Approach to Generate Safety Validation Test Cases from UML Activity Diagram. In *the Proceedings of the 20th Asia-Pacific Software Engineering Conference*, pp. 189–198. IEEE, Dec. 2013.

[36] 長田知之, 原内聡, 北村操代, 山地勉, 上野泰秀. 故障木図からの例外処理の導出による通信プログラム構築手法. 情報科学技術フォーラム講演論文集, Vol. 11, pp. 45–48, Sep. 2012.

[37] Tom Janssen, Rui Abreu, and Arjan J. C. van Gemund. Zoltar: A Toolset for Automatic Fault Localization. In *the Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, ASE '09, pp. 662–664. IEEE Computer Society, Nov. 2009.

付録

I　生成された promela 記述

```
/* Auto Generated Promela File */
/* Message Declaration */
mtype = {CloseReq_Tc2Ta__qpk, CloseReq_Ta2Ea__1ems,
Ack_Ea2Ta__1xjo, WaitFree_Ta2Tc__3cui,
DataRecv_Tb2Tc__4hj2, CloseCtrl_Ea2Ta__7u1z,
DataClear_Ta2Tb__8ao0};

/* Channel Declaration */
chan to_EquipmentA = [20] of {mtype};
chan to_TaskA = [20] of {mtype};
chan to_TaskB = [20] of {mtype};
chan to_TaskC = [20] of {mtype};

/* Variable for send and receive */
bool send = false;
bool receive = false;
mtype msg;

/* Process Declaration */
active proctype EquipmentA_1(){
/* EquipmentA receive CloseReq_Ta2Ea__1ems */
d_step{
to_EquipmentA?CloseReq_Ta2Ea__1ems;
send = false;
receive = true;
msg = CloseReq_Ta2Ea__1ems;
}
/* EquipmentA send Ack_Ea2Ta__1xjo */
d_step{
send = true;
receive = false;
msg = Ack_Ea2Ta__1xjo;
```

```
to_TaskA!Ack_Ea2Ta__1xjo;
}
}
active proctype EquipmentA_2(){
/* EquipmentA send CloseCtrl_Ea2Ta__7u1z */
d_step{
send = true;
receive = false;
msg = CloseCtrl_Ea2Ta__7u1z;
to_TaskA!CloseCtrl_Ea2Ta__7u1z;
}
}

active proctype TaskA_1(){
/* TaskA receive CloseReq_Tc2Ta__qpk */
d_step{
to_TaskA?CloseReq_Tc2Ta__qpk;
send = false;
receive = true;
msg = CloseReq_Tc2Ta__qpk;
}
/* TaskA send CloseReq_Ta2Ea__1ems */
d_step{
send = true;
receive = false;
msg = CloseReq_Ta2Ea__1ems;
to_EquipmentA!CloseReq_Ta2Ea__1ems;
}
}

active proctype TaskA_2(){
/* TaskA receive Ack_Ea2Ta__1xjo */
d_step{
to_TaskA?Ack_Ea2Ta__1xjo;
```

```
send = false;
receive = true;
msg = Ack_Ea2Ta__1xjo;
}
/* TaskA send WaitFree_Ta2Tc__3cui */
d_step{
send = true;
receive = false;
msg = WaitFree_Ta2Tc__3cui;
to_TaskC!WaitFree_Ta2Tc__3cui;
}
}

active proctype TaskA_3(){
/* TaskA receive CloseCtrl_Ea2Ta__7u1z */
d_step{
to_TaskA?CloseCtrl_Ea2Ta__7u1z;
send = false;
receive = true;
msg = CloseCtrl_Ea2Ta__7u1z;
}
}

active proctype TaskA_4(){
/* TaskA send DataClear_Ta2Tb__8ao0 */
d_step{
send = true;
receive = false;
msg = DataClear_Ta2Tb__8ao0;
to_TaskB!DataClear_Ta2Tb__8ao0;
}
}

active proctype TaskB_1(){
```

```
/* TaskB send DataRecv_Tb2Tc__4hj2 */
d_step{
send = true;
receive = false;
msg = DataRecv_Tb2Tc__4hj2;
to_TaskC!DataRecv_Tb2Tc__4hj2;
}
}

active proctype TaskB_2(){
/* TaskB receive DataClear_Ta2Tb__8ao0 */
d_step{
to_TaskB?DataClear_Ta2Tb__8ao0;
send = false;
receive = true;
msg = DataClear_Ta2Tb__8ao0;
}
}

active proctype TaskC_1(){
/* TaskC send CloseReq_Tc2Ta__qpk */
d_step{
send = true;
receive = false;
msg = CloseReq_Tc2Ta__qpk;
to_TaskA!CloseReq_Tc2Ta__qpk;
}
}

active proctype TaskC_2(){
/* TaskC receive WaitFree_Ta2Tc__3cui */
d_step{
to_TaskC?WaitFree_Ta2Tc__3cui;
send = false;
```

```
receive = true;
msg = WaitFree_Ta2Tc__3cui;
}
}

active proctype TaskC_3(){
/* TaskC receive DataRecv_Tb2Tc__4hj2 */
d_step{
to_TaskC?DataRecv_Tb2Tc__4hj2;
send = false;
receive = true;
msg = DataRecv_Tb2Tc__4hj2;
}
}
```