社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2006－SE－151（19）
2006／3／24

シーケンス図からの時間性能モデル検査用オブザーバ生成手法

長谷川哲夫*　　深澤良彰**
*株式会社 東芝/早稲田大学　　**早稲田大学

多種のシステムを低コストで短期間に開発しなければならない組込みソフトウェアなどの分野でも、ソフトウェアの大規模化が進む一方で大量販売により出荷後の不具合修正コストは大きく、モデル検査手法のような設計検証の自動化技術がより容易に使えるようになることに期待が高まっている。設計者にとってモデル検査の敷居が高い理由のひとつは、検証したい性質を時相論理式で与える点がある。本稿では、モデル検査ツール UPPAAL を対象として、シーケンスチャートから検証したい性質を表現するオブザーバを機械的に生成する手法を提案する。注目するシーケンスに対応する可能性のある挙動の完全な監視と、注目しない事象の透明化を特徴とする。また、本手法を通信プロトコルのアプリケーションに適用してオーバーヘッドの評価を行った。

A method generating an observer for model checking from a sequence chart

Tetsuo Hasegawa*　　Yoshiaki Fukazawa**
*Toshiba corporation/Waseda university　　**Waseda university

In the software field of mass products such as embedded systems, an error cause high correction cost after shipping. But many kinds of systems must be developed in a short term and low-cost. Then it may be impossible comprehensive design review by human resources. So model checking that is one of automatic design verification techniques is expected to be easy to use for software designer. In this paper, a method generating an observer for the model checking tool UPPAAL from a sequence chart, which is usually made by designers, is proposed. Also it is reported the result of applying this method to a communication systems.

1. はじめに

モデル検査とは、システムの振る舞いモデルを網羅的に調べることによって誤りを発見する自動検証手法である。近年、多くのモデル検査ツールが開発されるとともに、モデル検査を産業界の実システムに適用した事例報告も増えてきた。これまでのモデル検査の適用先は、原子力発電所の制御、航空管制など不具合が社会的な問題となり開発に大きなコストをかけられる高信頼システム分野が主であった。しかし、多種のシステムを低コストで短期間に開発しなければならないソフトウェア分野でこそ早い段階での誤り抽出は重要である。例えば、組込みソフトウェア分野のように大量販売される製品においては出荷後の不具合修正はリコールなど非常にコストがかかる。そして、ネットワーク家電のようにネットワークに接続される機器が増え、ソフトウェアは大規模化、複雑化が進みつつある。その結果、ソフトウェアの全動作パターンを人海戦術で検証する従来型の設計レビューは膨大な作業工数が必要となってきており、もはや現実的ではなくなってきている。

そのため、モデル検査手法のような設計検証の自動化技術が、より容易に使えるようになることに期待が高まっている。

本稿では、特にシステムの時間性能に注目した時間性能モデル検査ツールを対象として、設計者が通常設計時に作成するシーケンスチャートとステートチャートを基にしてモデル検査の活用を容易化する手法について述べる。

2. 性能モデル検査

性能モデル検査では、時間的な要素も含めたシステムの振る舞いと検証したい性質を入力とし、システムの振る舞いを網羅的に調べて入力された性質が常に満たされるかどうかを自動検証する。システムの振る舞いは時間オートマトンで記述し、検証したい性質は時相論理とよばれる論理体系の式で表現す

るのが一般的である。

時間オートマトンは、ステートチャートに詳細な時間制約を付加することにより表現できる。ステートチャートは、組込み系ソフトウェア分野では設計過程で作成されることが多く、また UML における設計書式の1つにもなっており、設計者にとって比較的理解が容易である。一方、時相論理式は多くの設計者にとっては敷居が高く、記述、理解が容易とは言い難い。

設計時に作成されるシーケンスチャートは、特定のシナリオに基づくシステム挙動であり、システムが満たすべき時間制約を表現することができる。UML においては、時間制約に対する標準書式として profile for SPT[1]が提供されている。そこで、このシーケンスチャートから検証性質としての時相論理式導き出す研究が行われている[2]。しかし、この場合でもモデル検査の検証結果を分析する際には時相論理式の理解が必要となり、それだけでは設計者がモデル検査を容易に活用できるとは言い難い。

一方、検証したい性質を時相論理式のみで記述する他に、オブザーバと呼ばれる時間オートマトンと簡単な時相論理式の組み合わせで表現することも可能である。オブザーバとは、検証対象の時間オートマトンと一体となって動作し、そのロケーションや遷移を監視するものである。

図 1 にオブザーバを用いた性能モデル検査の例を示す。本例は、プロセス 1、2 間で排他制御を行うもので、各プロセスは id という外部変数に自プロセスの番号を代入した後一定時間(k)経過後に id の値が自プロセス番号であれば排他領域に入り、排他領域から出る際に id に 0 を代入する。待機中に id の値が 0 になった場合は、再度 id に自プロセスの番号を代入して再度待機するというものである。この処理を時間オートマトンで表したものが、図 1 の(1)および(2)であり、初期ロケーション(図では二重丸のロケーション)、排他領域に入るための処理を開始するロケーション(req)、排他領域に入れるか確認中のロケーション(wait)、排他領域に入っているロケーション(CS)から構成される。図 1(3)のシーケンスチャートは、プロセス 1 が排他領域に入ろうとした後にプロセス 2 が排他領域に入ろうとし、プロセス 1、プロセス 2 の順番で

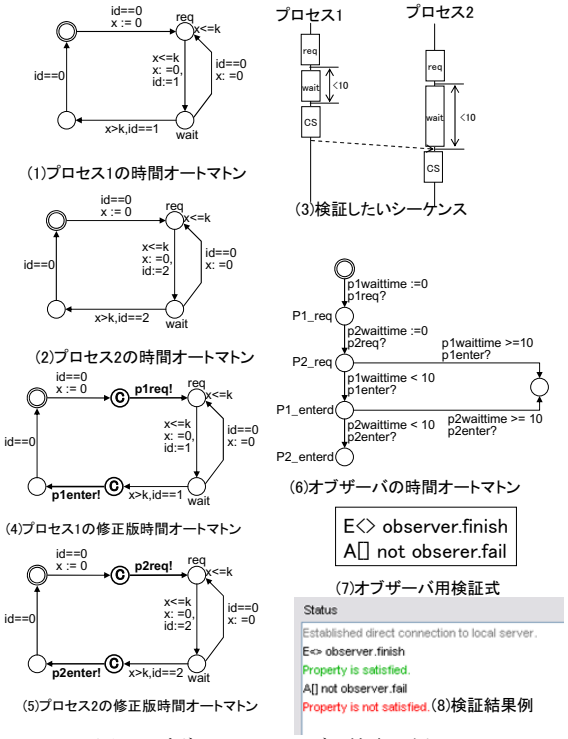

図1 オブザーバによるモデル検査の例

それぞれ 10 時間ユニット未満に排他領域に入れる実行シーケンスを示している。この様な実行シーケンスが存在するか、および、どちらのプロセスも必ず 10 時間ユニット未満で排他領域に入れるかを検証するためのオブザーバとして図 1(6)の時間オートマトンを作成する。一方、プロセス 1、2 の時間オートマトンに対しても、オブザーバに自プロセスのロケーション遷移を通知するために図 1 の(4)、(5)に示す様に変更を行う。プロセス 1 の時間オートマトンは、req、CS に遷移する際にそれぞれ p1req、p1enter という通信イベントを送信し、プロセス 2 は同様に p2req、p2enter という通信イベントを送信する。オブザーバはこれらを受け取って自身のロケーションを進める。p1req のイベントを受信すると、時間変数である p1waittime をクリアし、プロセス 1 が req に遷移したことを意味するロケーション(P1_req)に遷移する。プロセス 2 が req に遷移し、自身が P2_req に遷移した後、時間変数 p1waittime が 10 時間ユニット未満の間にプロセス 1 が CS に遷移して p1_enter の通信イベントを受信するとオブザーバは P1_entered に遷移し、引き続きプロセス 2 が CS に遷移するのを待つ。一方、p1waittime が 10 時間ユニット以上になった後にプロセス 1 が CS

に遷移して p1_enter の通信イベントを受信するとオブザーバは fail ロケーションに遷移する。その後プロセス 2 が req から CS に 10 時間ユニット以内に遷移すれば finish ロケーションに遷移し、10 時間ユニット以上経過してから CS に遷移した場合は fail ロケーションに遷移する。ここで、fail ロケーションはすべての実行シーケンスで到達してはいけないロケーション、finish ロケーションはある実行シーケンスでは到達できるロケーションである。そして、必ず fail ロケーションに到達しない、finish ロケーションに到達できる、という性質を検証するための検証式として図 1(7)に示す 2 式を定義することが多い。この例では、finish に到達する実行シーケンスは存在するが、fail に到達する実行シーケンスもあり得るという検証結果が得られる。このように、オブザーバは時間オートマトンで表現されており、検証結果の分析のための理解、あるいは、さらなる分析のためのオブザーバの修正等も比較的容易である。このオブザーバをシーケンスチャートから生成する検討もある[3]。しかし文献[3]で提案されている方式では、シーケンスチャートに対応する振る舞いすべてを検証できない場合があった。

本稿ではシーケンスチャートから機械的にオブザーバを生成するより現実的な手法を提案する。3 章で期待される要件を整理し、4 章以降で代表的な性能モデル検査ツールである UPPAAL 向けのオブザーバを生成する手法を説明する。

3. 期待される要件

シーケンスチャートからオブザーバを機械的に生成する際の要求としては以下の項目が考えられる。

■ 検証する性質として定義できることが望ましい事項

- プロセス間イベントのタイミングや順序
- プロセス内部のアクションシーケンス

検証したい性質として、必須と考えられるのは、プロセス間同期にかかわるイベントである。たとえば、プロセス間通信やグローバル変数の更新/参照などがあり、これらのイベント発生時刻に対して相互の順序関係や何時間単位以内などの時間制約を定義できる必要がある。さらに、設計者による検証という観点からは、プロセス内でこのようなアクションシーケンスの結果のイベントだからこのタイミングに起きなくてはならない、あるいは、このタイミングに発生したイベントに基づいてこのような処理をしなくてはならないなどのプロセス内部のアクションシーケンスも同時に検証できることが望ましい。

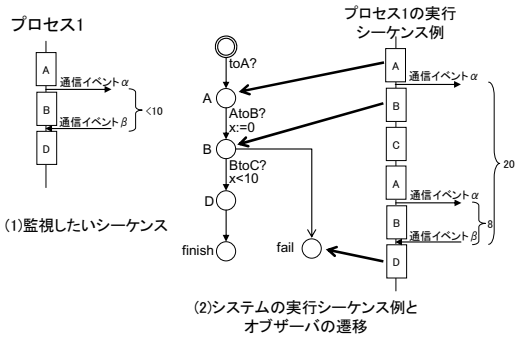

図2 監視に失敗する例

■ オブザーバの監視能力

- 注目するシーケンスに対応する可能性のある挙動の完全な監視

オブザーバは、監視対象のプロセスのロケーション遷移に合わせて自身のロケーションを遷移させるため、図 1 に挙げたオブザーバのように監視したいシーケンスのみを持つオブザーバは不都合が生じることがある。例えば、監視したいシーケンスと一部(前半)のみが同一の挙動に対してオブザーバ自身のロケーションを遷移させてしまうと、途中で実行シーケンスが監視したいシーケンス異なった時点でオブザーバが停止してしまう、あるいは、図 2 のように監視したいシーケンスの続きに相当する実行シーケンスが別途現われる場合に、連続したシーケンスと見なして誤った時間制約の判断をしてしまうなどである。

- 注目していない事象の透明化

注目しているシーケンスに関係しない、他のプロセスのアクションの遷移やそのタイミングにオブザーバが影響を受けないことが必要である。

3. ターゲットモデル検査ツール UPPAAL

UPPAAL[4] は、Uppsala University と Aalborg University が共同開発し、初版は 2001 に公開されたモデル検査ツールであり、以下の特徴を持つ。

- 検証対象のシステムは並行プロセス群で構成され、1 プロセスに 1 時間オートマトンが対応する。

- 時間変数は、時間の経過に応じて値が増える変数である。値の増加はシステム全体に対して互いに同一ペースである。
- 時間オートマトン上に定義する時間制約は、遷移に対するガード、および、各ロケーションでの滞在時間制約として規定する。x, y などの時間変数に対し、x < 5、x − y < 3 などの式で表す。
- 検証式は代表的な時相論理式の表現形式の 1 つである CTL のサブセットで与える。Gp(常に p が成立)、Fp(いつか p が整理)の前に、A(全ての実行パスで成り立つ)、または E(ある実行パスで成り立つ)が付き、AG などの後ろに、時間変数に関する論理式が書ける。
- プロセス間同期は、チャネルを介した同期型通信でモデル化されている。チャネル名を指定して送受信を行うと待ち合わせによる同期がとられる。1 対 1 に同期するチャネルと、送信側がブロックされず待ち合わせしている全プロセスと同期する同期型放送通信チャネルがある。また、データはグローバル変数を介してやり取りする。時間オートマトンの遷移にチャネル名+"!"を記述することで送信、チャネル名+"?"を記述することで受信を表す。
- 特殊なロケーションとして、時間が経過しない Urgent ロケーションと、時間が経過せずかつ他に比べて優先的に遷移する Committed ロケーションがある。

4. オブザーバ生成手法

オブザーバ生成の入力となるシーケンスチャートの表現、オブザーバに対して、各プロセスのロケーション遷移を通知するための修正方法、そして、オブザーバの生成方法について述べる。

4.1 入力とするシーケンスチャート

監視対象として注目するシーケンスは、各プロセスのアクションシーケンスで示す。ここで、アクションはプロセスの時間オートマトンの各ロケーションに1対1に対応するものとする。シーケンスチャートは、他プロセスとのやり取りであるイベントのシーケンスを主としてプロセス内部の詳細なアクションシーケンスは記載しない場合もある。しかし、設計者が検証を行う場面では、シーケンスチャートを詳細にすることは無理が無いと思われる。

時間制約に関しては、以下の様に表現する。UPPAAL の時間オートマトンでは、各プロセスの通信や外部変数の参照・更新イベントは時間オートマトンのロケーション遷移時に実施される。したがって、時間制約は、アクションシーケンスにおける特定のアクション遷移と、同一または他のプロセスのアクションシーケンスにおけるアクション遷移間の半順序関係または相対的な時間関係で示すことができる。

個々の時間制約に対して、常に成立しなくてはならない制約(all 制約と表現する)なのか、あるいは、その制約を満たす実行シーケンスが存在すれば良い制約なのか(exist 制約と表現する)を示す。

4.2 監視対象プロセスの時間オートマトンに対する修正

監視対象プロセスの時間オートマトンに対して、オブザーバ自身のロケーション遷移を通知するための修正が必要となる。オブザーバもシステムの一部となって動作するため、通知によって待ちが発生するなどプロセスの動作が変わらないように放送通信を用いて通知する。簡単のために時間オートマトンの各ロケーションの名前は対応するアクション名がついており、システムで一意とする。修正は、図 3 に示すように各遷移に対して、以下の遷移を Committed ロケーションで接続して挿入するものである。

- 遷移前後のロケーション名の組を名前とする放送通信チャネルへの送信
- 遷移前のロケーション名を名前とする放送通信チ

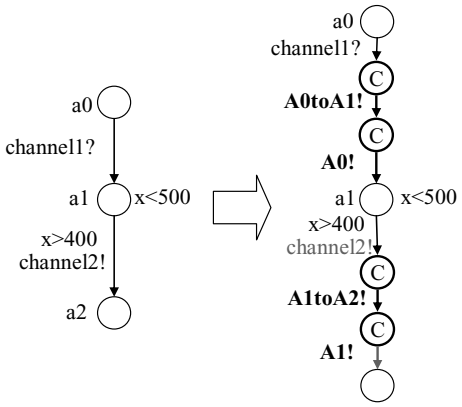

図3 プロセスの時間オートマトン修正例

ャネルへの送信

通知の利用方法と 2 種類の通知が必要な理由は、次節で述べる。

4.3 オブザーバの生成方法

まず 1 プロセスを対象とした場合のオブザーバの生成手法について述べ、その後に複数プロセスを対象とする場合について述べる。

図 4 と図 5 にシーケンスチャートと対応して生成されるべきオブザーバの例を示し、以下に生成手順を述べる。

1) シーケンスチャートのアクションに対応するロケーションとアクション遷移に対応するロケーション遷移を配置し、最初のアクションを初期ロケーションとする。さらに、finish ロケーションと fail ロケーションを追加して、最後のアクションに対応するロケーションから finish ロケーションへの遷移を追加する。
2) 各遷移において、前後のアクションの組を名前とする放送通信チャネルからの受信を付加する。
3) 各ロケーションと初期ロケーションの間を、対応するアクション名を名前とする放送通信からの受信を持つ遷移で接続する。
4) 時間制約毎に時間変数を定義し、時間制約の起点となる遷移にその時間変数のクリア処理を付加する。
5) 時間制約を判定すべき遷移に、満たすべき条件をガードとして持つ遷移を Urgent ロケーションで接続して挿入する。
6) 時間制約が all 制約であった場合は、図 5 に示すように、遷移前のロケーションと fail ロケーションを、満たすべき条件の否定をガードとして持つ遷移と遷移前後のロケーションに対応するアクションの組を名前とする放送通信チャネルからの受信を持つ遷移で接続する。

図 4 のオブザーバ例を用いて動きを説明する。プロセスがアクション A1 を終了して A2 を開始する、すなわち時間オートマトンにおいて対応するロケーション A1 からロケーション A2 に遷移すると、オブザーバは A.1toA.2 のチャネルから受信してアクション A2 に対応するロケーションに遷移する。その後、プロセスがアクション A2 を開始してからの経過が時間ユニット 300 以上 350 以内にアクション A3 を開始すると、オブザーバは A.2toA.3 のチャネルから受信してアクション A3 に対応するロケーションに遷移する。しかし、時間経過が時間ユニット 300 未満または 350 より大きかった場合、および、次のアクションが A3 以外の場合は、監視したいシーケンスとは異なると判断しての処理となる。その場合はオブザーバの A3 に対応するロケーションへの遷移にはガードに反する、または、チャネルからの受信ができずに遷移せず、引き続きプロセスの時間オートマトンから送信される A2 のチャネルから受信して初期ノードへ遷移する。その後、再び、アクション A1 からの実行シーケンスが開始されるのを待つ。

一方、時間制約が図 5 の例のように all 時間制約であった場合は、次のアクションはシーケンスと同じだが時間制約に反した場合には fail ロケーションへの遷移を行い、次のアクションがシーケンスと異なる場合には上の例と同様に初期ノードに戻って、再び監視対象の実行シーケンスが開始されるのを待つ。

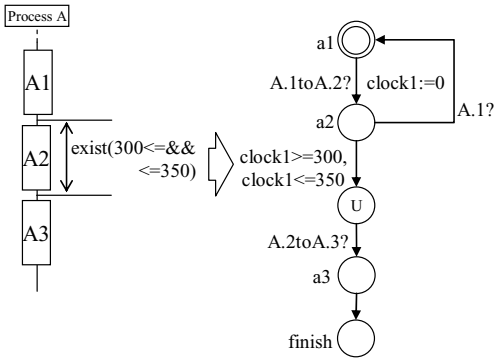

図4 exist時間制約に対するオブザーバ生成例

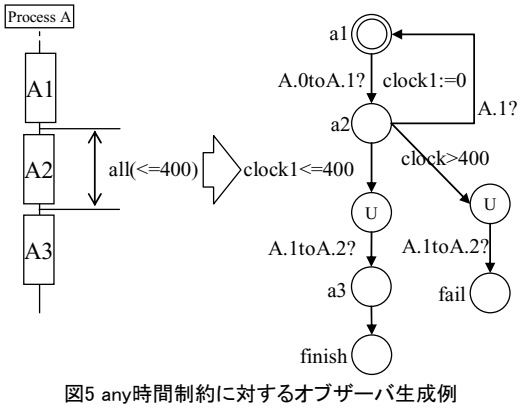

図5 any時間制約に対するオブザーバ生成例

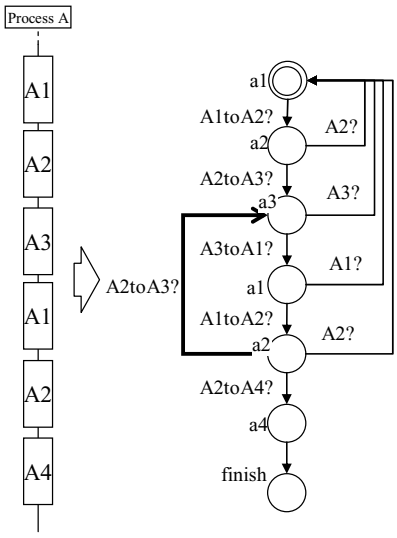

図6 オブザーバ生成例

ここで、シーケンスチャートにおいて、図 6 のアクション A1～アクション A2 のように先頭からのアクションシーケンスのパターンが、シーケンスの途中に再び出現する場合を考える(重複シーケンスと呼ぶ)。この場合は、手順(3)の後に更にそのロケーションに対して以下の手順が必要となる。

3')2 回目以降の重複シーケンスの最後のロケーションと、先頭の重複シーケンスに引き続くロケーションの間を、対応するアクションの組を名前とする放送通信からの受信を持つ遷移で接続する。

図 6 の例で A1,A2,A3,A1,A2,A3,A1,A2,A4 というアクションシーケンスは、監視したいシーケンスを含んでいるが、この遷移が無い場合は 2 回目の A3 の次の A1 で初期ロケーションに戻ってしまい監視できないが、この遷移の追加で正しく監視できるようになる。

次に複数プロセスを対象とする場合について述べる。まず、個々のプロセスに対してこれまで述べてきた手法でオブザーバを生成する。その上で、以下の手順でこれらのオブザーバに修正を加える。ここでは、時間制約が 0 より大きいという条件の場合を特に順序制約と呼ぶことにする。

1) オブザーバ毎に、現在ロケーションを示すグローバル変数を定義し、シーケンスチャートのアクション遷移に対応する各遷移に対して、次のロケーションを示す値の代入処理を付加する。
2) 時間制約または順序制約毎に、グローバル時間

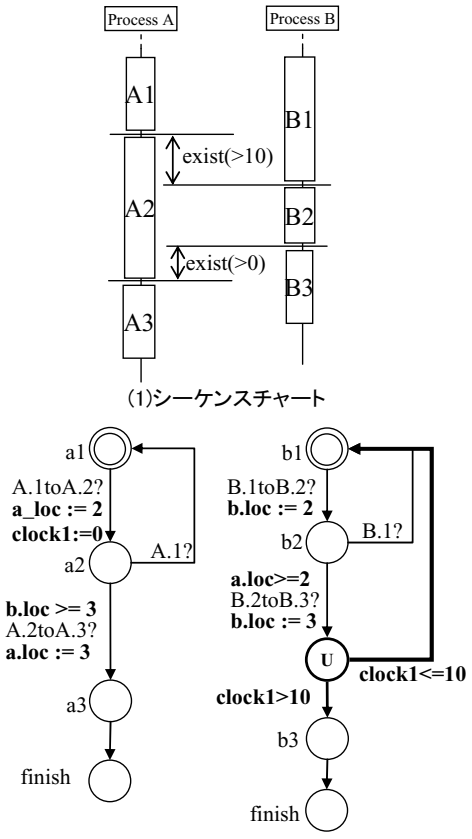

図7 複数プロセスの遷移間の
時間制約、順序制約に対するオブザーバ例

変数を定義し、時間制約の起点となる遷移に時間変巣をクリア処理を付加する。

3) 時間制約または順序制約を判定すべき遷移に対して、起点となる遷移を持つプロセスの現在ロケーションが起点となる遷移以降であることを示す条件をガードとして付加する。制約が all 制約の場合は現在ロケーションが起点となる遷移より前であることを示す条件のガードと、遷移前後のロケーションに対応するアクションの組を名前とするほう送通信チャネルからの受信を持つ遷移で fail ロケーションと接続する。
4) 時間制約の場合は、判定すべき遷移にさらに時間制約を条件とするガードを付加した遷移を Urgent ロケーションで接続して挿入する。時間制約が exist 時間制約の場合は Urgent ロケーションと初期ロケーションの間を、all 時間制約の場合は Urgent ロケーションと fail ロケーションの間を、それぞれ時間制約の否定条件をガードとして持つ遷

移で接続する。

図 7 に本手順でオブザーバを生成した例を示す。図において太字で示した部分がこの手順で追加された部分である。

最後に、検証式は図 1 の例と同様で、

- E<>((オブザーバ名).finish)
- A[]((オブザーバ名).fail)

の 2 種類である。

4. アプリケーションへの適用例と評価

提案手法を UPPAAL の適用事例として報告されているオーディオ機器間の通信プロトコル[5]に適用して、動作確認を行った。本アプリケーションは、Receiver と複数の Sender がバスを介して接続されたシステムを対象としている。データはビット列の 1 と 0 を、バス電圧変化で伝える。図 8 に示すように、888

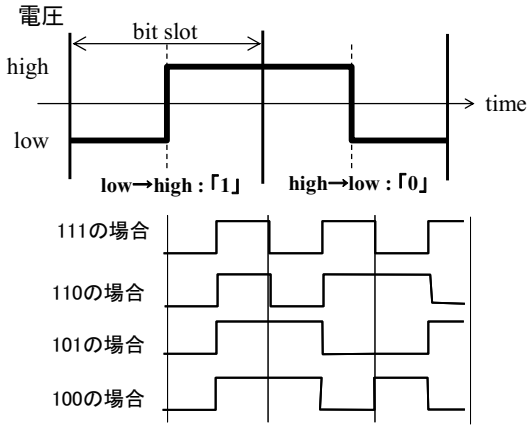

図8 データ送信の例

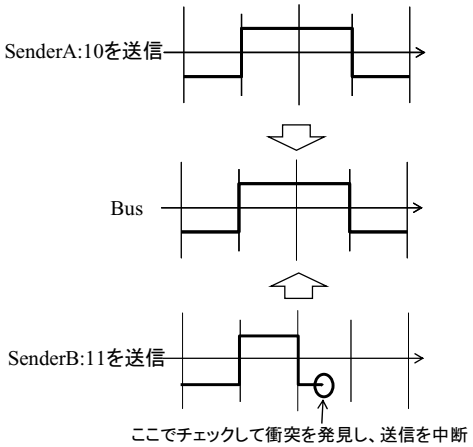

図9 Senderが衝突を発見する例

マイクロ秒をビットスロットと呼び、ビットスロットの中間、444 マイクロ秒時点で電圧が low から high に変化すれば 1、high から low に変化すれば 0 を意味する。この繰り返しでデータのビット列を送信する。複数の Sender が同時に送信した場合、それぞれが設定した電圧の or がバスの電圧となる。Sender は図 9 に示すように、ビットスロットの 1/4 または 3/4 時点で自身が low を設定していた場合はバスの電圧をチェックし、high になっていた場合は他の Sender も同時にデータを送信中であり衝突が起きていると判断して、送信を中断する。送信を開始する時は電圧が low であることを確認してから 20 マイクロ秒以内に電圧を high にする。この 20 マイクロ秒の間に他の Sender も同時に送信を開始する可能性があり、ビットスロットは Sender 間で最大 20 マイクロ秒ずれる。さらに、Sender が電圧を変化させるタイミング、バスの電圧をチェックするタイミング、Receiver がバスの電圧をチェックするタイミングはそれぞれ最大 5%ずれる可能性がある。文献[5]ではこのような条件でプロトコルが正しくデータ転送をできるかをモデル検査で検証している。その結果不具合を発見したシステムを対象に、さらに、タイミングのずれが 0.1%まで少ないと仮定した場合でも、衝突が発見できなくなるいくつかのシーケンスが起こりえることを検証した。図 10 に検証したいシーケンスを示す。このシーケンスは、SenderA が電圧をチェックするビットスロットの 1/4 から 3/4 の間に、SenderB のビットスロット 1/2 までの電圧 high の期間が収まってしまい、衝突が発見できなくなるケースである。図 10 の HF,LS1,LS2,LF1,LF2 はそれぞれのタイミングで Sender が行っているアクションである。シーケンスチャートは図 11 に示すようになり、これをもとに提案した手法で生成したオブザーバを図 12 に示す。Sender、Receiver、Bus の時間オートマトンも提案手法で修正し、UPPAAL で検証したところ、図 13 の実行シーケンスで図 10 に示すシーケンスが発生することがわかった。具体的には、SenderA が常にタイミングが遅れ、SenderB が常にタイミングが早くなるケースで、タイミングのずれが累積して生じることがわかった。また、SenderB のシーケンスにはアクション HF-LF1 の重複パターンが存在するが、図 13 に示すように直前に監視していたシーケンスの HF-LF1

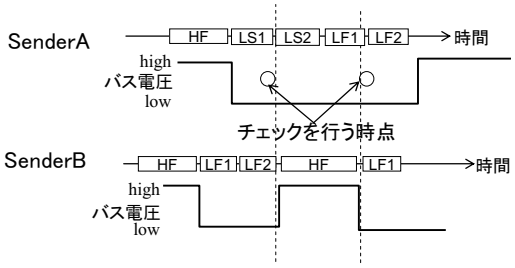

図10 検証したい振る舞い

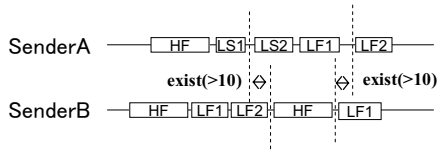

図11 シーケンスチャート

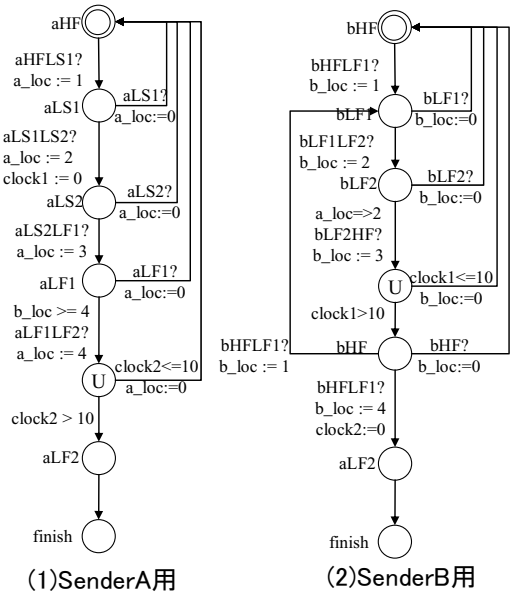

図12 オブザーバ

図13 発見されたシステムの振る舞い

を含むシーケンスを発見できた。

本アプリケーションは、プロセスとして Sender と Receiver の他に、バス動作をシミュレーションするプロセスなどから構成される。Sender の時間オートマトンは 12 ロケーション、19 遷移であったところをオブザーバへの通知のための修正により 50 ロケーション、57 遷移に増える。Receiver は 10 ロケーション、16 遷移が、42 ロケーション、48 遷移に、Bus は 2 ロケーション 4 遷移が 8 ロケーション 10 遷移に増える。Sender は 2 プロセスあるので、合計でロケーション数は約 4.2 倍、遷移数は約 3 倍、合計では 3.4 倍となる。

モデル検査においてはロケーション数や遷移数が増えることは一般には状態空間を著しく増やして検証効率を下げてしまうが、本手法で追加しているのはすべて Committed ロケーションとその前後の遷移であり、通常のロケーションに比べると影響がはるかに少ないことが報告されている[5]。

影響を調べるために、テスト用の簡単な性質に対して、元の時間オートマトンでのモデル検査にかかる時間と、修正後の時間オートマトンでのモデル検査にかかる時間を計測、比較した。すると、1.8 秒かかっていたところが、6.5 秒かかり、約 3.6 倍となっている。この数値は、ロケーションと遷移の合計の倍数に近いものであり、実用的には許容範囲と言える。今後他の例で評価を進めていきたい。

5. おわりに

組み込み型のソフトウェア設計時に作成するシーケンスチャートから機械的にオブザーバを生成し、モデル検査を行う手法を提案した。今後は、多くの事例に適用して、オーバーヘッドの評価を進めるとともに、より効率的な生成手法を検討していきたい。

参考文献

[1] ″UML Profile for Schedulability, Performance and Time″,URL:http://www.omg.org/cgi-bin/doc?formal/2005-01-02

[2]″ Automated Check of Architectural Models Consistency Using SPIN″, Paola Inverardi, Henry Muccini, Patrizio Pelliccione, Proc. of ASE2001

[3] ″Timed Sequence Diagrams and Tool-Based Analysis A Case Study″, Thomas Firley, Michaela Huhn, Karsten Diethers, Thomas Gehrke, and Ursula Goltz, Proc. of UML1999

[4] ″A Tutorial on Uppaal″, Gerd Behrmann, Alexandre David, and Kim G. Larsen. Proc. of SFM-RT'04.

[5] ″Verification of an Audio Protocol with Bus Collision Using Uppaal″. Johan Bengtsson, W. O. David Griffioen, Kare J. Kristoffersen, Kim G. Larsen, Fredrik Larsson, Paul Pettersson and Wang Yi, Proc. of CAV1996.