April 21, 2010

RT0902

Computer Science 9 pages

Research Report

時間制約を持つ仕様に対するUML/SysMLモデルの動的検査手法

小野康一、河原亮、中村宏明、石川浩

IBM Research - Tokyo
IBM Japan, Ltd.
1623-14 Shimotsuruma, Yamato
Kanagawa 242-8502, Japan

1 はじめに

UML[3] や SysML[2] の組込みシステム/リアルタイムシステム開発への適用が試みられている [1]．上流工程の開発成果物としてのモデルは，対象のシステム/ソフトウェアの構造や振舞を正しく表現し，かつ，曖昧さや矛盾などがないように意味を厳密に与えることが不可欠である．UML 2.0 以降では，振舞の意味を与えるためにステートマシン図やアクティビティ図を用いて実行可能モデルを表現することができる．

このような実行可能モデルによる振舞の定義を，その厳密性を利用して検査/検証に利用することができる．検証の目的は主に 2 つあり，1 つはモデル自身の正当性検証であり，もう 1 つはモデルから生成・実装されたコードのモデルに対する正当性検証である．検証手法としては，形式手法などを用いた静的な検査/検証と，実行トレースの取得などによる動的な検査/検証がある．

形式手法は強力な手法ではあるが，必ずしもすべてに有効な手法とはいえない．組込みシステム/リアルタイムシステムでは，ソフトウェアは独立した要素ではなく，物理的特性を持つ他の要素と関連しながら動作することが多い．そのようなシステムでは，物理的特性を持つ要素は機械系や電気系であり，ソフトウェアはそれらに対する制御系 (コントローラー系) として構成される．前者の機械系や電気系をプラント系と呼ぶこともある．そのような複合的なシステムのモデリングでは，ソフトウェアによる制御系は離散系モデルとして定義し，機械系/電気系は連続系モデルとして定義する．ソフトウェアの離散的振舞のモデリング言語としては UML/SysML が主に用いられ，連続系を記述するモデリング言語としては Simulink などがある．そのような混交したモデルによるシステムの振舞を検証するには，制御系の離散的振舞のみを形式手法で静的に検証するだけでは十分ではない．たとえば，プラント系 (連続系) のある属性値 (たとえば速度や温度など) がある値に到達した時にそれを監視している制御系 (離散系) がある状態 A から状態 B に遷移するとして，状態 A になってから状態 B になるまでの時間がある時間制約を満たしている，といった単純な振舞の仕様ですら，制御系を時間オートマトンで表現して UPPAAL などでモデル検査しても検証できない．したがって，このようなシステムにおいては，モデルの直接実行もしくはモデルから生成・実装されたコードの実行による動的検証が有効である．なお，モデルの実行においては，制御系のモデルとプラント系のモデルの連携による実行を必要とする．

以降では，モデルの直接実行，もしくは，モデルから生成・実装されたコードの実行による振舞を実行トレースとして取得し，この実行トレースが振舞の仕様に対して正当であるかを検証することを課題として考える．振舞の仕様は，UML/SysML によるモデリングの文脈ではシーケンス図を使用することが多い．シーケンス図による振舞の仕様は，対象システムが機能的に満たすべき制約であるとみなすことができる．また，UML Profile for MARTE (Modeling and Analysis of Real Time and Embedded systems) [4] を適用することで，対象システムが満たすべき時間制約もシーケンス図に記述することができる．我々は，あるイベントの区間が時間制約を持つ仕様を前提として，モデルの直接実行，もしくは，モデルから生成・実装されたコードの実行による振舞を観測して得た実行トレースを動的に検証する手法を提案・実装した [6][7]．仕様は時間制約を持つシーケンス図の集合で与えられる．ここのシーケンス図はテストシナリオやユースケースにおけるイベント列全体ではなく，その短い断片の集まりとして定義される．本稿では，この時間制約の動的検証手法の拡張について述べる．最初に 2 節で，提案済みの時間制約の動的検査手法の概要について述べる．次に 3 節で，手法の拡張について述べる．

2 時間制約の動的検査手法

2.1 解決すべき課題

振舞の仕様は，複数の小さなシーケンス図の集合で与えられる．個々のシーケンス図はイベントの系列を与えて，これはシステムの機能的な制約となる．また，個々のシーケンス図にはイベントの発生に関する時間制約も与えられる．このように，時間/機能制約によって定義されるシーケンス図の集合を振舞の仕様とすると，それに対する実行トレースの動的検証では 2 つの課題が生じる．

1 つは，実行トレース中の個々のイベントについて，それが検証の対象となるシーケンス図を選択する必要がある，という課題である．言い換えると，個々のシーケンス図が仕様として適用可能であるかどうかを，実行トレースの各イベントについて判断する必要がある．シーケンス図を細かい粒度で与えるため，個々のシーケンス図に与えるイベント列は，それが生じるための文脈や前提が存在する図 1．したがって，そのような文脈や前提条件から外れている場合には，そのシーケンス図による検証はおこなわないように動的検査手法が制御する必要がある．このために，UML 2.x のシーケンス図に対して拡張をおこなった．具体的には，適用可能であるための前提条件をシーケンス図が持つように拡張した．前提条件が満たされている場合に限り，そのシーケンス図は動的検証で用いられる．満たされていない場合には，実行トレース中にそのシーケンス図と一致しないイベント系列があったとしても不整合とはみなさない．さらに，振舞の仕様として定義されるシーケンス図の集合は，すべての生じうるイベントを網羅することは要求しないとする．つまり，特定の状況下での振舞のみを検証の対象として，それに対応するシーケンス図の集合のみを仕様として定義することもありうる．そのような目的にも前提条件は使われる．

もう 1 つは，前提条件が満たされて検査に使用されるシーケンス図との照合の必要性である．特定のテストシナリオに沿って実行するという前提が成り立たない以上，実行トレースの個々のイベントがどのシーケンス図の振舞に従っているのかを事前に決定することができない．このため，実行トレースの検査は，トレース中のイベントを時系

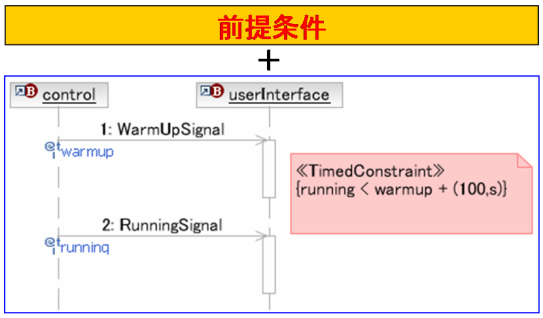

図 1　シーケンス図と前提条件

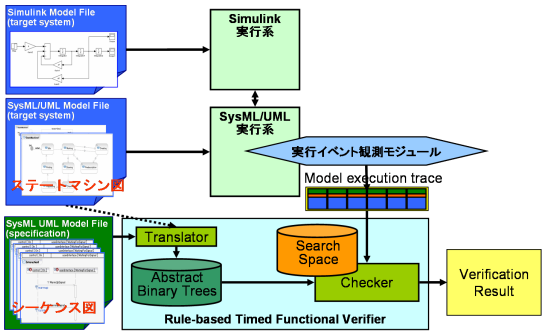

図 2　時間/機能制約検査の概略

列に沿って辿りながら，そのイベントの時点で適用可能なシーケンス図を特定する処理をおこないつつ，そのシーケンス図とイベントの照合をおこなう，という処理になる．実行トレースのあるイベントの時点では，前提条件を満たすシーケンス図は複数ありえる．つまり，あるイベントの時点では，1 つのイベントの照合の対象となるシーケンス図が複数ありえる．個々のシーケンス図においては，その最後のイベントまでの一致が判定された時点で初めて，そこまでに至るイベント列の振舞がそのシーケンス図の定義に従っていたことが確定する．それまでは，照合の対象となるシーケンス図が複数ある以上は，個々のシーケンス図はあくまで振舞の仕様としての候補であり，そのシーケンス図と異なる振舞であっても不整合であるとは決定できない．検証に用いられる振舞の仕様の候補から除外されるだけである．候補が 1 つしか残っていない時点でそのシーケンス図と異なる振舞となった場合にのみ，不整合を検出したことになる．

2.2　時間/機能制約検査手法

本手法の概略を図 2 に示す．ここでは，モデルの直接実行結果を実行トレースとして取得し，振舞の仕様であるシーケンス図の集合と照合する手法として示す．1 節で述べたように，組込みシステム/リアルタイムシステムのモデリングでは，UML/SysML による制御系の離散的振舞だけでなく，プラント系の機械的/物理的現象を連続的振舞として Simulink などで定義する場合がある．離散系の実行においては連続系の実行と連携していることを前提とする．本手法では，制御系の振舞はステートマシン図を用いて定義してあるものとする．それとは別に時間/機能制約による振舞の仕様をシーケンス図の集合として定義してあるものとする．シーケンス図の集合は，実行可能モデルの中で与えてもよいし，別のモデルとして与えてもよい．論理的には，検証対象と仕様なので別のモデルともみなせる．

2.2.1　モデル実行系

図 2 に示したように，離散系の UML/SysML モデルの実行系と連続系モデルの実行系の連携によって，システム全体のモデルが実行される．我々は，この 2 つの実行系を連携するための技術を新たに開発した [5] ．この連携技術は，SysML がもつシステムエンジニアリングへのサポートを利用しながら Simulink による連続的な振舞い記述を SysML に取り込むことにより，制御系を含む組込みシステムの開発の上流工程におけるシミュレーションを実現することを目的としている．なお，本稿の動的検査手法自体は，離散系モデルが UML/SysML モデルであることは前提としているが，連続系モデルが Simulink であることを前提としない．本手法は離散系モデルと連続系モデルが連携して動作する実行系であれば適用可能である．

2.2.2　実行トレースの動的検証

本手法では，実行エンジンによってモデル実行トレースを取得し，それを，シーケンス図の表現で時間/機能制約として記述した振舞の仕様に対して検査する．実行トレースは離散系モデルにおけるオブジェクトの挙動にともなうイベント情報として記録される．連続系モデルで生じた変化は 2.2.1 節で述べた連携技術により，対応する離散系のモデル要素へのイベントとして伝達される．この際に時間管理モジュールによって連続系モデルで経過した時間は離散系モデルに対して同期される．したがって離散系モデルのイベント情報である実行トレースには，連続系モデルでの振舞の情報が含まれている．図 2 では，実行エンジンに対して実行イベント観測モジュールを付加して動作させてイベント情報を取得するように示してある．各イベントはそれが生じた時刻のタイムスタンプと，各ステートマシンの状態遷移およびシグナルによる相互作用の情報を持つ．

検証系からは実行トレースは単なるイベント列であればよい．したがって，モデル実行結果の観測イベントを記録・保存したものを実行トレースとして用いてもよいし，リアルタイムの観測イベントをストリーム的に実行トレースとして用いてもよい．

2.2.3　シーケンス図による振舞の仕様

振舞の仕様はシーケンス図の集合によって定義する．2.1 節で述べたように，個々のシーケンス図には，それが検証に適用可能であるかどうかを判断するための前提条件が付加されている．前提条件の表記は，UML 2.x に対する拡張になっている．2.2 節で述べたように，本手法では，制御系の振舞はステートマシン図を用いて定義してあると仮定している．そこで我々は前提条件の表現を，各オブジェ

クトの状態による条件式と定義した．つまり，実行トレースのあるイベントの時点において，あるシーケンス図が検証に適用可能であるためには，その時点で，そのシーケンス図が前提とする状態に各オブジェクトがなっている必要がある．その前提とする状態になっていない時点では，そのシーケンス図は検証に適用されない．図 3 に前提条件を付与したシーケンス図の例をしめす．この例では，control オブジェクトの状態が On であり，かつ，userInterface オブジェクトの状態が WaitingForSignal であることが前提条件として与えられている．したがって前提条件を状態の条件式として表現すると次のようになる (ただし，“.state” はそのオブジェクトの状態を表すとする)．

control.state=On ∧
userInterface.state=WaitingForSignal

この例では 2 つのオブジェクト (control, userInterface) のライフライン間で一組のメッセージの送受イベントのみが示されているが，本手法で仕様として与えるシーケンス図はオブジェクト/ライフライン/イベントの数に制限はない．仮に他のオブジェクト (たとえば heaterControl や waterTank など) のライフラインが共にシーケンス図中に存在してそれらとのメッセージ送受が記述されていた場合も，それを仕様として適用可能である．また，本手法はメッセージの方向性やメッセージ種別にも制限を設けていない．この例では control オブジェクトのライフラインから userInterface オブジェクトのライフラインへの非同期メッセージ WarmUpSignal および RunningSignal の送信が示されているが，この他に userInterface オブジェクトのライフラインから control オブジェクトのライフラインへメッセージ送信が記述されていても仕様として適用可能であるし，また，これらのメッセージが同期メッセージやその応答であるリプライメッセージとして記述されていても適用可能である．このモデルでは，加湿器を構成する複数のオブジェクトが独立したスレッドで並行動作することを意図して，それらのクラスを active class として定義している．それらのオブジェクト間の相互作用は非同期になるため，この例では非同期メッセージでシーケンス図を記述した．一般的に組込みシステムには独立して動作する計算ユニットや電子系/機械系ユニットが構成要素として存在することが多くあり，それらの構成要素間の相互作用はしたがって非同期メッセージで表現する．しかしながら前述のように，本手法が検証に用いる仕様としてのシーケンス図では，非同期メッセージ以外のメッセージ種別でも用いることができる．

2.2.4 時間制約と制約評価

シーケンス図は，0 個以上の時間制約も持ちうる．時間制約は UML における Constraint 要素として表現されている．より厳密には，Constraint 要素に対して，UML Profile for MARTE[4] の TimedConstraint ステレオタイプを適用し，時間制約であることをモデル中で表現している．時間制約式は，MARTE で定義している VSL (Value Specification Language) で表現される．具体的には，シーケンス図中のイベントやオカレンスに対して UML の TimeObservation を対応付けておくことで，イベントの発生時刻を表すシンボルが導入される．それを用いた式を記述することで，二つのイベントの間の経過時間に対する制約などが表現できる．図 3 では，TimeObservation として，シグナル WarmUpSignal が control オブジェクトから userInterface オブジェクトへ送信された時点を warmup，シグナル RunningSignal が control オブジェクトから userInterface オブジェクトへ送信された時点を running に対応付けている．シーケンス図中に置かれた Constraint 要素は時間制約であり，そこに VSL で記述された時間制約式

running < warmup + (100,s)

は，warmup に対応する時点 (シグナル WarmUpSignal を送信) から running に対応する時点 (シグナル RunningSignal を送信) までの経過時間が 100 秒未満であることを述べている．

3 時間制約検査の拡張

2 節で述べた手法の課題は，仕様が多数のシーケンス図で構成される場合のイベント照合のコストである．したがって，照合候補の絞込みや検査失敗の判定を早期におこなうことが重要となる．本稿では，時間制約の成否判定を早期におこなうための拡張について述べる．

図 3 の時間制約を例に挙げる．この時間制約式には 2 つの TimeObservation を表す時刻変数 running，warmup が用いられている．したがって，この時間制約式の真偽が確定するには，warmup に対応するイベント (シグナル WarmUpSignal の送信) および running に対応するイベント (シグナル RunningSignal の送信) が生じる必要がある．しかしながら，この 2 つのイベントの発生後にこの時間制約式の成否を naive に判定する方法では効率がよくない．シグナル WarmUpSignal の送信イベントが生じてからシグナル RunningSignal の送信イベントが生じるまで長い時間経過があるかもしれないし，あるいは，シグナル RunningSignal の送信イベントが発生しないかもしれない．仕様には複数のシーケンス図が記述されており，この動的検査手法ではそれらのシーケンス図を同時並行に検査する．成否が確定しない時間制約式を持つシーケンス図は，確定するまでイベント照合の候補として残り続けることになるため，検査コストが低減できない．

そこで，次のアイデアに基づいて時間制約式の成否判定を早期におこなう．各シーケンス図は特定のシナリオにおける振舞を表現しているため，それに関係するイベントのみが記述されている．たとえば，図 3 には 2 つのイベントのみが記述されているが，実際の振舞では，そのシナリオとは無関係の他のイベントが並行して生じている．すべてのイベントは時系列に沿って発生しているため，図 3 の最初のイベント (シグナル WarmUpSignal の送信イベント) が生じてから次のイベント (シグナル RunningSignal の送信イベント) が生じるまでの間で他のイベントも生じている．そのような中間イベント e の発生時刻を $To(e)$ で表すとすると，以下の関係が成立することになる．

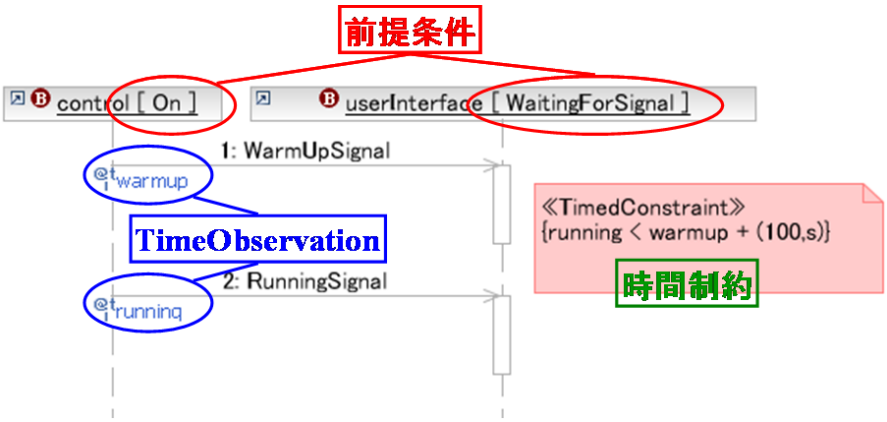

図 3　状態の前提条件を付与したシーケンス図

$$warmup < To(e) \wedge To(e) < running \qquad (1)$$

この関係から次のことがいえる．2 番目のイベント (シグナル RunningSignal の送信イベント) の発生以前であっても，最初のイベント (シグナル WarmUpSignal の送信イベント) が発生以降のイベント e について，図 3 の時間制約式

$$running < warmup + (100, s) \qquad (2)$$

中の $running$ に $To(e)$ を代入して得られる以下の式

$$To(e) < warmup + (100, s) \qquad (3)$$

が満たされないならば，式 2 も満たされない．よって，2 番目のイベント (シグナル RunningSignal の送信イベント) の発生以前であっても，図 3 の時間制約式の真偽を判定することができる．

3.1　対象とする時間制約式のクラス

この技術で対象とする時間制約式のクラスを定義する．組込みシステム/リアルタイムシステムの振舞の仕様としての時間制約は，主に，ある処理に要する時間がある範囲内である，という形になる．したがって，2 つの時刻変数の差分、時間変数、定数との加減算の結果の比較演算式を基本式として構成される制約式を対象式のクラスとする．BNF による定義は以下のようになる．

```
<時間制約式> ::=
      <時間制約項>
    | <時間制約式> <論理和演算子> <時間制約項>
<時間制約項> ::=
      <時間制約因子>
    | <時間制約項> <論理積演算子> <時間制約因子>
<時間制約因子> ::=
      <時間時刻制約単純式> | ( <時間制約式> )
    | ! ( <時間制約式> )
<論理和演算子> ::= ||
<論理積演算子> ::= &&
<時間時刻制約単純式> ::=
      <時間制約単純式> | <時刻制約単純式>
<時間制約単純式> ::=
      <時間演算式> <比較演算子> <時間演算式>
<時刻制約単純式> ::=
      <時刻演算式> <比較演算子> <時刻演算式>
<比較演算子> ::= == | != | < | <= | > | >=
<時間時刻演算式> ::=
      <時間演算式> | <時刻演算式>
<時間演算式> ::=
      <時間演算項>
    | <時間演算式> <加減演算子> <時間演算項>
<時間演算項> ::=
      <時間演算因子>
    | <時間演算項> <乗除演算子> <算術演算因子>
    | <算術演算項> <乗除演算子> <時間演算因子>
<時間演算因子> ::=
      <時間変数> | <時間定数> | ( <時間演算式> )
<時間変数> ::=
      /*DurationObservation で導入される変数名*/
<時間定数> ::= ( <数値> , <時間単位> )
<時間単位> ::= h | m | s | ms | us | ns
<加減演算子> ::= + | -
<乗除演算子> ::= * | /
<時刻演算式> ::=
      <時刻演算項>
    | <時刻演算式> <加減演算子> <時間演算項>
<時刻演算項> ::= <時刻変数> | <時刻定数>
<時刻変数> ::=
      /*TimeObservation で導入される変数名*/
<時刻定数> ::= <時> : <分> : <秒> : <秒以下>
<算術演算式> ::=
      <算術演算項>
    | <算術演算式> <加減演算子> <算術演算項>
```

<算術演算項> ::=
<算術演算因子>
| <算術演算項> <乗除演算子> <算術演算因子>
<算術演算因子> ::= <数値> | (<算術演算式>)

3.2 技術的課題の定義

技術的課題を以下に定義する．この技術の目的は，各イベントにおける時間制約式評価にかかる実行時間コストの低減である．このために，評価する制約式を簡略化することを考える．時間制約式の一般形は以下のように与えられる．

$$p(t_1, t_2, \cdots, t_n) \tag{4}$$

ここで t_i はイベント e_i の発生時刻を表す変数シンボル，または，経過時間を表す変数シンボルである．時間制約式中の変数シンボルに対応するイベントが発生する以前に時間制約式の真偽判定が可能になるのは，これらの変数シンボルのうち，1 つだけが未確定 (対応するイベントが未発生) で，残りが確定している (対応するイベントが発生している)，という条件のときである．その条件下であれば，いま仮に t_k が未確定であるとする．確定しているイベントの発生時刻を $c_i (i \neq k)$ とすると，式 4 は以下のように式変形して簡略化できる．

$$p_k(t_k) = p(c_1, c_2, \cdots, c_{k-1}, t_k, c_{k+1}, \cdots, c_n) \tag{5}$$

その時刻 t_k に対応するイベント e_k の代わりにあるイベント e の時刻 $To(e)$ を用いて評価するということは，$p_k(To(e))$ を評価するということになる．したがって，技術的課題は以下の 2 点である．

1. 与えられたシーケンス図の時間制約式をどのように変形 (簡略化) するか
2. 変形 (簡略化) した時間制約式をどの時点/イベントで判定するか

以降では特に前者について述べる．なお，後者については，時間制約式を満たせなくなった瞬間が判明しなくてもいいので，すべてのイベントにおいて時間制約式を評価しなくてもいい．たとえば，イベントの平均的な発生頻度と，現在時刻から時間制約式が満たされなくなる時刻の間の時間から，時間制約式が満たされなくなるまでに観測されるイベント数を予測し，その予測回数のイベントが観測されるまでは時間制約式を評価しない，という方法が考えられる．

3.3 時間制約式の簡略化

時間制約式は検査開始前に簡略化しておく．制約式の簡略化は式変形操作であり計算コストが大きいため，検査中に動的におこなうべきではない．また，どのような簡略化をおこなうべきかは事前に静的に決められる．

式 4 のイベント $e_1, e_2, \cdots e_n$ のどれが時系列上で最終イベントになるかを事前に決定可能かどうかは与えられたシーケンス図の構造に依存する．UML 1.x までのシーケンス図は構造化されていないので，最終イベントはイベント列の最後のイベントであることが一意に決まる．UML 2.x のシーケンス図は複合フラグメント (combined fragment) による制御構造が導入されているため，複合フラグメントの種類によっては最終イベントが一意に定まらない可能性がある．具体的には，alt フラグメント，opt フラグメント，seq フラグメント (弱シーケンス) などである．よって，1 つの時間制約式から複数の簡略化した式を事前に作成しておき，検査の際にそのどれが最終イベントとして残るかが確定するので，その式を用いて評価する．

図 4 に alt フラグメントを持つシーケンス図の例を示す．この例では 4 つのイベントに相当する TimeObservation が示されている (warmup，running，cooldown，standby)．これらのイベントのうち，最終イベントになる可能性のあるものは standby と running に対応するシグナル StandBySignal 送信イベントおよびシグナル RunningSignal 送信イベントの 2 つである．よってこの例では 2 つの簡略化した式 ($p_{standby}(standby)$ および $p_{running}(running)$) を事前に作成しておく．

$$p_{standby}(standby) = \\ standby < running - warmup + cooldown \tag{6}$$

$$p_{running}(running) = \\ running > standby - cooldown + warmup \tag{7}$$

シーケンス図とその時間制約式から，そのシーケンス図で最終イベントになりうる候補を求めて，そのイベントに対応する TimeObservation の時間/時刻変数を求めるアルゴリズムを図 5 に示す．また，各複合フラグメントについてのアルゴリズムを図 6 〜図 10 に示す．

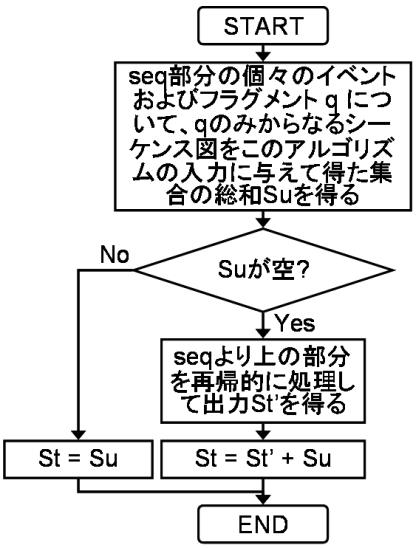

図 6　複合フラグメント (seq) の処理

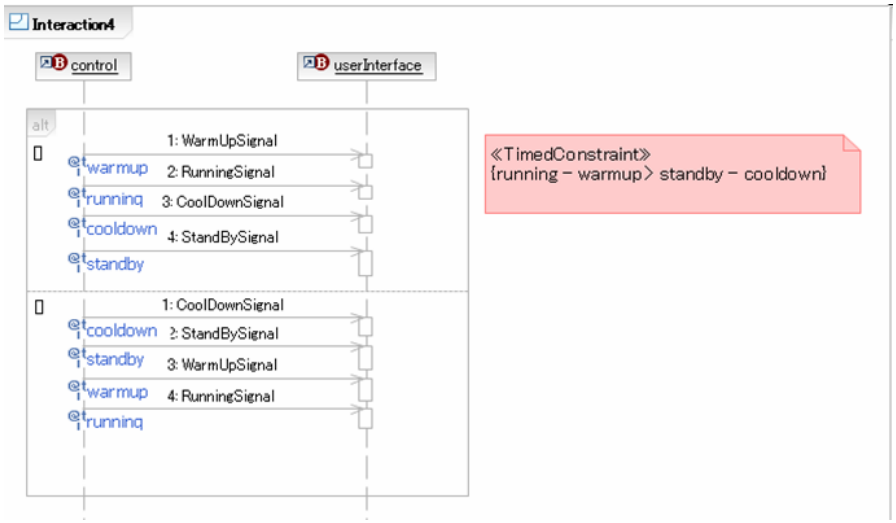

図 4　イベント順序が一意にならない例 (alt)

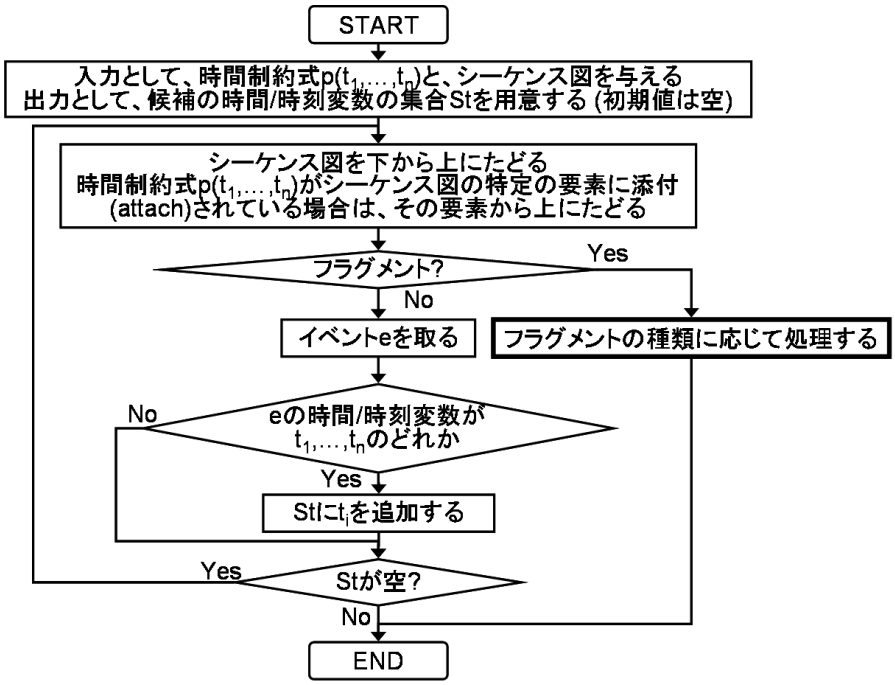

図 5　最終イベントになりうるイベントの時間/時刻変数を求めるアルゴリズム

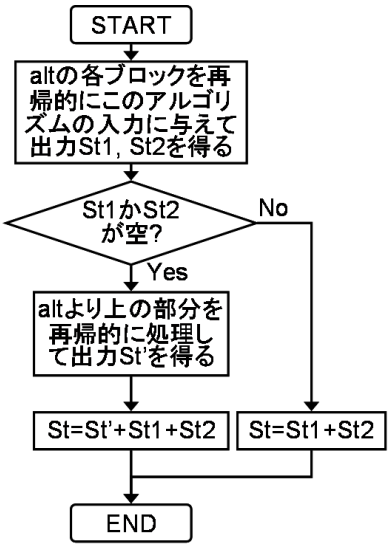

図 7　複合フラグメント (alt) の処理

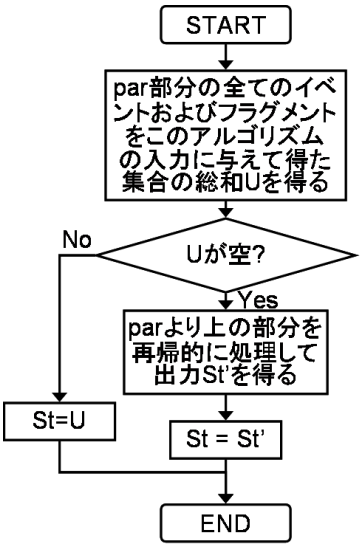

図 9　複合フラグメント (par) の処理

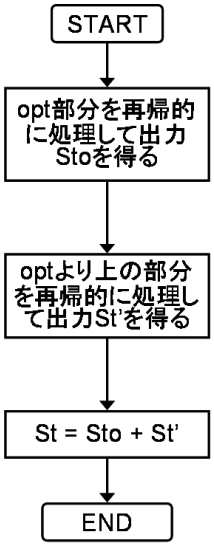

図 8　複合フラグメント (opt) の処理

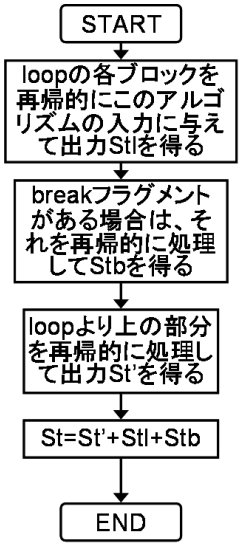

図 10　複合フラグメント (loop) の処理

4　まとめ

UML/SysML によるモデルの実行トレースが時間制約を持つ仕様に対して動的検査する技術の拡張を提案した．検査のためにイベント列と多数のシーケンス図との照合をおこなうが，照合にかかるコストを低減するために，時間制約式に現れる時間変数に対応するイベントの発生よりも前の時点のイベントで時間制約式の真偽を判定するための拡張について述べた．本稿では，時間制約式の簡略化のために，時間制約式に現れる時間/時刻変数に対応するイベントの中の最終イベントだけに着目したが，複数の時間/時刻変数間に依存関係があるなら，互いに独立ではないので，より早い段階での時間制約式の真偽判定の可能性がある．これについては将来の課題と考えている．

参考文献

[1] L. Lavagno, G. Martin, and B. V. Selic, editors. *UML for Real: Design of Embedded Real-Time*

Systems. Springer, 2003.

[2] Object Management Group. *OMG System Modeling Language (OMG SysML) Specification Version 1.1*, November 2008.

[3] Object Management Group. *OMG Unified Modeling Language (OMG UML) Superstructure Specification Version 2.2*, February 2009.

[4] Object Management Group. *A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems*, 1.0 edition, November 2009.

[5] 河原亮 and 中村宏明. SysML-Simulink 協調シミュレーションツールによる制御システムの仕様検証. In **組込みシステムシンポジウム** *2009 (ESS2009)* **講演集**, 2009.

[6] 小野康一, 中村宏明, and 石川浩. 時間/機能制約による仕様に対する実行可能な UML/SysML モデルの動的検査手法. In **日本ソフトウェア科学会第** *25* **回大会論文集**, pages 8B–2, 2008.

[7] 小野康一, 中村宏明, and 石川浩. 時間/機能制約による仕様に対する実行可能な UML/SysML モデルの動的検査手法. **コンピュータソフトウェア**, to appear.