社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.
SS2001-41 (2002-01)

時間制約を持つGUI制御部の仕様記述の一手法

加藤雄一郎† 岡野浩三† 谷口健一†

† 大阪大学 大学院基礎工学研究科 情報数理系専攻
〒560–8531 大阪府豊中市待兼山町 1–3
Phone: 06-6850-6606 FAX: 06-6850-6609

E-mail: †{y-katou,okano,taniguchi}@ics.es.osaka-u.ac.jp

あらまし 本稿では, グラフィカルユーザインターフェース (GUI) アプリケーションの開発コストを小さくするために, GUI 制御部の仕様を形式的に記述する枠組を提案する. 本手法では, 各 GUI 部品 (オブジェクト) 名や GUI アプリケーションで実行されるタスク名を記述したコンポーネント部と, 時間制約を含む GUI 制御部の動作を記述した動作部の 2 つの記述から, タイムオートマトンで記述された GUI の全体動作記述を自動導出する. そして, その全体動作記述から時間制約処理部と各オブジェクトごとの動作仕様群を自動導出する. 本手法での利点は, (a) 時間制約が記述できること, (b) 全体仕様と各オブジェクトごとの動作仕様群の二つを導出することにより, 全体動作の把握と, GUI に特有なオブジェクト指向プログラミングによる実装が容易であること, (c) GUI 制御部に対して安全性などの自動検証が可能となることである.

キーワード GUI, 制御部, 仕様記述, タイムオートマトン

A Method for Describing Specification for GUI Controllers with Timing Resrictions

Yuichiro KATO†, Kozo OKANO†, and Kenichi TANIGUCHI†

† Department of Infomatics and Mathmatical Science, Graduate school of Engineering Science, Osaka University
1–3 Machikaneyama-cho, Toyonaka, Osaka, 560–8531 Japan
Phone: +81-6-6850-6606 FAX: +81-6-6850-6609

E-mail: †{y-katou,okano,taniguchi}@ics.es.osaka-u.ac.jp

Abstract We propose a method for designing for GUI controllers in order to reduce total cost of the development. In our method, we describe a behavioral description of GUI with timing restrictions (behavior description) and decsribe GUI objects corresponding to widgets and tasks used in the GUI application (component description). A whole specification (abstract level specificaition) of a GUI's controller is, then, derived automatically as a timed automaton from these descriptions. The timed automaton is automatically devided into a set of timed automata (low level specification), each of which expresses behavior of the GUI's object. Advantage of our method is following. (a) Timing restrictions such as time-out and so on, can be presented. (b) The method derives two levels of specification. The abstract level specification helps us to understand the whole behavior of the GUI applicaiton. The lower level specification helps us to implement the GUI application in the object oriented programming style. (c) Verification some properties of GUI controller, such as safety, can be performed automatically.

Key words GUI, controller, formal specificaiton, timed autumaton

1 はじめに

近年の大規模なグラフィカルユーザインターフェース (GUI) アプリケーションにおいて, GUI の制御はより複雑な傾向にある. GUI アプリケーションの開発においては, 一般的なソフトウェア開発とは異なり, 人手により行われる過程が多い. しかし, たとえば, GUI アプリケーションの初期状態から意図していない状態へと遷移することがないか, どのコマンドも実行不可能な状態に遷移することがないか, などの安全性の問題については, 原理的にはその GUI のすべての状態を調べる必要があるため, 人手による検証は非常に困難であり, 計算機の援用が望まれる.

そこで本稿では, これらのコストを総合的に小さくするために, GUI 制御部の仕様を形式的に記述する枠組を提案する. これまでにも GUI 開発の枠組に対する研究が行われているが, GUI 制御部に対する形式仕様記述と検証を扱ったものはほとんどない. 詳しくは 4. で述べる.

本手法では, GUI アプリケーション設計者は, 各 GUI 部品 (オブジェクト) 名 や GUI アプリケーションで実行されるタスク名をコンポーネント部として記述し, オブジェクトとタスクの対応関係および実行制御を GUI アプリケーションの動作部として記述する. GUI アプリケーション設計者が, システム全体の動作内容すべてを記述するのは非常に煩雑であると思われる. また, タイムアウトなどの時間処理を扱う GUI アプリケーションも数多く存在している. そこで本手法では, 動作部は GUI アプリケーションの代表的な振る舞いを時間制約込みで, たとえば, 「一時停止中に, 60 秒以内に pause ボタンが押されれば演奏を再開する, 押されなければ自動的に演奏を再開する」という動作の列として記述する. このような形で与えられた動作部からタイムオートマトンモデル [1] で記述された GUI アプリケーションの全体動作仕様を自動導出する. この全体動作記述はひとつのタイムオートマトンで記述される. 導出されたオートマトンにより, GUI アプリケーション全体の動作内容を正確に把握することが可能となり, またモデルチェッキングの手法を用いることで安全性などの自動検証が可能となる.

本手法では, さらに, 導出されたタイムオートマトンから, 時間制約オートマトンと各オブジェクトごとのオートマトンを自動導出する. 以後, 時間制約オートマトンと各オブジェクトごとのオートマトン (動作仕様) を集めたものを動作仕様群とする. 各動作仕様は互いに内部通信イベントを用いて, 部分的に同期しながら, GUI アプリケーションの全体仕様を記述したタイムオートマトンを模倣する. 本手法では時間制約を一つのオートマトンに集約することにより, 以後, 時間制約処理部と各オブジェクトに対するオートマトンからプログラムを具体的かつ自動的に実装することを容易にしている.

このように GUI 制御部の全体動作記述から時間制約部と各オブジェクトごとの動作仕様を自動導出する方法は著者らが文献 [2] で提案している. 本稿では, [2] では陽に述べていなかったオブジェクトおよびタスクの集合をコンポーネント部として記述すること, また, 分解アルゴリズムを明確にすることで, オブジェクト指向プログラミングによる実装を容易にしていること, 安全性などの検証を意識した GUI 制御部の仕様記述であること, などの点で拡張を行っている.

以降, 2. では, GUI 制御部の記述としてコンポーネント部および動作部の記述に関しての定義と記述例について述べる. 3. では, タイムオートマトンの定義, および, 上述の二つの自動導出について, 問題定義とアルゴリズムを与える. 4. では, 関連研究および実際への応用に関するいくつかの論点について議論する. 5. でまとめる.

2 GUI 制御部の記述

2.1 コンポーネント部

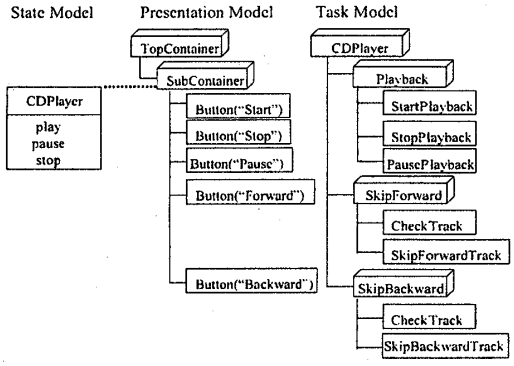

図 1: CD プレーヤーコンポーネント部

コンポーネント部とは, 対象となる GUI アプリケーションを構成するオブジェクト名とタスク名を記述したものである. 例として CD プレーヤーを考える. 図 1 は, CD プレーヤーのコンポーネント部を表したものである. 本稿では [3] にもとづき, コンポーネント部を 3 つのモデル (*State Model, Task Model, Presentation Model*) で記述する.

State Model は, GUI アプリケーションがタスク実行前後の状態を記述したものである. 本手法では, GUI アプリケーションを構成するオブジェクトに対してユーザからのイベントを制限したり (モーダル制御), 特定のオブジェクトに対してユーザからのイベントをフォーカスする (フォーカス制御) ために用いられる. クラスとその属性で構成される. 図 1 では, CDPlayer クラス内で, play, pause, stop が CD プレーヤーの状態名として定義されている.

Task Model は, GUI アプリケーションが実行するタスク (*task*) 名が記述されている. 図 1 では, CDPlayer は Playback, SkipFoward, SkipBackward の 3 つタスクから構成されている. また, Playback は StartPlaback, StopPlayback, PausePlayback の 3 つのタスクから構成されている.

Presentation Model は, GUI アプリケーションを構成するオブジェクト (*object*) の階層で表現される. 各

オブジェクトは, ウィジェット (*widget*) である *concrete object (CO)* と CO を抽象化した *abstract object (AO)* の 2 つのレベルで記述される. AO はそれぞれ, データの入力に使われる場合は *inputter*, データの出力に使われる場合は *displayer*, データの入出力に使われる場合は *editor* とする. 実装を行う際には, オブジェクトは CO で記述されなければならないので, AO は実装段階で, GUI アプリケーション開発者または開発システム側で AO の CO への割り当てが行われる. 図 1 では, CD プレーヤーは CO である 5 つのボタン (Start, Stop, Pause, Forward, Backward) , その集まりである SubContainer とトップウィンドである TopContainer から構成されている. たとえば, TopContainer はフレーム, SubContainer はパネルとなる.

また, State Model の各クラスと Presentation Model の各 SubContainer を関連付けることで, モーダル制御やフォーカス制御を実現する. 図 1 では, CDPlayer クラスで定義された play, pause, stop の各値を用いて, SubContainer の AO, つまり, 5 つのボタンのモーダル制御を実現する.

2.2 動作部の記述

本稿では, 動作部を $B = \langle B_{pre}, B_I \rangle$ を要素とする集合として記述する.

- B_{pre}:動作系列 B_I の前提状態. ここで前提状態とは, GUI アプリケーションが B_I 実行前に満たすべき状態名である.
- B_I:ユーザと GUI アプリケーションのインタラクションのリストであり, $B_I = [I_1, \cdots, I_n]$ で構成される.
  $I_i = \langle S_i, T_i, EB_i, P_i \rangle$ は, 以下の要素により構成される.
  - S_i:GUI アプリケーションのタスクの実行を引き起こすオブジェクトか, またはタスクで用いる変数に関する制約式である.
  - $T_i = [t_{i1}, \cdots, t_{im}]$:$S_i$ が成立するときに GUI アプリケーションが順次実行するタスクのリストである.
  - $EB_i = \langle to_i, ET_i \rangle$:一定時間経過しても T_i が実行されなかったときに, そのタイムアウト後に実行される GUI アプリケーションのタスク名のリスト ET_i とそのタイムアウト時間 (to_i) である. I_i 中に S_i, T_i の記述がともに無ければ, B_{pre} 遷移後のタイムアウト時間 (to_i) とタイムアウト後に実行される GUI アプリケーションのタスク名のリスト (ET_i) を表すものとする. なお, ET_i 実行後の遷移先は次にのべる P_i とする.
  - P_i:後置状態. ここで後置状態とは, T_i 実行後に GUI アプリケーションが満たさなければならない状態名である.

CD プレーヤーの動作部の一部を図 2 に表わす. 図 2 は次の内容を表している.

```
\begin{behavior}
@*
 :Button("play"),StartPlayback: >play
 :Button("stop"),StopPlayback: >stop
@play
 :Button("pause"),PausePlayback: >pause
 :Button("forward"),CheckTrack:
 :(trackNum < maxNum),SkipForwardTrack
                      ;StartPlayback: >play
 :(trackNum = maxNum),StartPlayback: >play
 :Button("backward"),CheckTrack:
 :(trackNum > minNum),SkipBackwardTrack
                      ;StartPlayback: >play
 :(trackNum = minNum),StartPlayback: >play
@pause
 :!(60sec),StartPlayback: >play
 :Button("pause"),StartPlayback: >play
 :Button("forward"),CheckTrack:
 :(trackNum < maxNum),SkipForwardTrack
                      ;StartPlayback: >play
 :(trackNum = maxNum),StartPlayback: >play
 :Button("backward"),CheckTrack:
 :(trackNum > minNum),SkipBackwardTrack
                      ;StartPlayback: >play
 :(trackNum = minNum),StartPlayback: >play
@stop
 :Button("pause"),StartPlayback: >play
 :Button("forward"),CheckTrack:
 :(trackNum < maxNum),SkipForwardTrack
                      ;StartPlayback: >play
 :(trackNum = maxNum),StartPlayback: >play
 :Button("backward"),CheckTrack:
 :(trackNum > minNum),SkipBackwardTrack
                      ;StartPlayback: >play
 :(trackNum = minNum),StartPlayback: >play
\end{behavior}
```

図 2: CD プレーヤー動作部

pause ボタンが押されると, PausePlayback を実行し, トラックの再生を一時停止する pause 状態に遷移する. また, forward ボタンが押されると, CheckTrack を実行してトラック番号を点検する. トラック番号が CD トラックの最大値よりも小さければ, SkipForwardTrack を実行して頭出しを行い, そして, StartPlayback を実行し, トラックの再生を開始する play 状態に遷移する. 一方, トラック番号が CD トラックの最大値であれば, 頭出しを行わず, StartPlayback を実行し, トラックの再生を開始する play 状態に遷移する. backward ボタンに関しても同様である.

pause 状態においては, 状態遷移後, 60 秒間どのボタンも押されなければ, StartPlayback を実行し, トラックの再生を開始する play 状態に遷移する.

モーダル制御やフォーカス制御に関しては, State Model で記述されたクラスの各状態に対して, 関連付けられた SubContainer のうち, どのオブジェクトが実行可能であるかを動作部とともに記述する.

ここで, 上述のオーディオプレーヤーのモーダル制御を表す. 図3は次の内容を表している. 全ての状態にお

```
\begin{modal control}
@*       :Button("forward");Button("backward")
@play    :Button("pause");Button("stop")
@pause   :*
@stop    :Button("play")
\end{modal control}
```

図 3: モーダル制御

いて, forward ボタンと backward ボタンが有効である. play 状態において, pause ボタンと stop ボタンが有効である. pause 状態において, 全てのボタンが有効である. stop 状態において, play ボタンが有効である.

3 導出問題とアルゴリズム

3.1 タイムオートマトン

Alur のタイムオートマトン (timed automaton) [1] により, 全体仕様, および, 時間制約処理部と各オブジェクトの動作仕様を記述する. タイムオートマトンは, 任意の状態遷移においてリセット可能な有限個のクロックを用いて, 時間制約付の状態遷移が記述できる. 状態遷移表 (timed transition table) は次の5項目 $\langle \Sigma, S, S_0, C, E \rangle$ で表現される. ここで, Σ は有限アルファベット, S は状態集合, $S_0 \subset S$ は初期状態集合, C は有限個のクロック, $E \subseteq S \times S \times \Sigma \times 2^C \times \Phi(C)$ は状態遷移の集合である. 状態遷移 $\langle s, s', \sigma, \lambda, \delta \rangle \in E$ は次の意味を持つ. クロックに関する時間制約 δ を満たす時刻に, 入力記号 σ に対して, 状態 s から状態 s' に遷移し, λ で指定されたクロックがリセットされる. 時間制約はクロックと定数の大小比較 ($\leq$) およびそれらの論理結合で構成される.

タイムオートマトン A によって認識される timed word (σ, τ) は入力記号列 $\sigma = \sigma_1, \cdots, \sigma_n$ と時間系列 $\tau = \tau_1, \cdots, \tau_n$ からなり, 時間 τ_i に σ_i が入力されることを意味する. また, timed word (σ, τ) における状態遷移表の run r $(\bar{s}, \bar{v})$ は次の無限系列として定義される.

$$r : \langle s_0, v_0 \rangle \overset{(\sigma_1, \tau_1)}{\rightarrow} \langle s_1, v_1 \rangle \overset{(\sigma_2, \tau_2)}{\rightarrow} \langle s_2, v_2 \rangle \cdots$$

ここで, すべての $i \geq 0$ について, $s_i \in S, v_i \in [C \rightarrow R]$($R$: 正の整数) であり, 次の性質を満たす.

- $s_0 \in S_0$ かつ すべての $x \in C$ について $v_0 = 0$.
- すべての $i > 0$ について, $\langle s_{i-1}, s_i, \sigma_i, \lambda_i, \delta_i \rangle$ という辺が存在する. ここで, $(v_{i-1} + \tau_i - \tau_{i-1})$ は δ_i を満たし, v_i は λ_i で指定されていれば 0 にリセットされ, そうでなければ $(v_{i-1} + \tau_i - \tau_{i-1})$ に等しい.

また, ある run r の部分系列

$$r : \langle s_i, v_i \rangle \overset{(\sigma_{i+1}, \tau_{i+1})}{\rightarrow} \langle s_{i+1}, v_{i+1} \rangle \cdots \langle s_{i+n}, v_{i+n} \rangle$$

を r の partial run として定義する.

3.2 全体動作仕様の導出

3.2.1 問題定義

コンポーネント部と動作部の記述から全体動作仕様であるタイムオートマトンの導出問題を次のように定義する.

入力: GUI 制御部のコンポーネント部の記述, 動作部の記述

出力: 一つのタイムオートマトン

ここで, 出力されたタイムオートマトンは, 入力された動作部の記述それぞれに対して, それが生成できる partial run を, 出力タイムオートマトンで受理できなくてはならない.

3.2.2 アルゴリズム

導出アルゴリズムについては, 文献 [4] にもとづく. また, 前述した制約を満たすもののうち, なるべく状態数が少ないタイムオートマトンを出力する.

- **Input:** $R = \langle B_{pre}, B_I \rangle$, $A_{i-1} = \langle \Sigma_{i-1}, S_{i-1}, S0_{i-1}, C_{i-1}, E_{i-1} \rangle$
- **Output:** $A_i = \langle \Sigma_i, S_i, S0_i, C_i, E_i \rangle$

$A_i := A_{i-1}$;
Let s_p be a state characterized by B_pre;
If (no state in S_i is identical to s_p)
 $S_i := S_i \cup \{s_p\}$;
For each ($B_i (= \langle S_i, T_i, EB_i, P_i \rangle)$ in B_I) {
 Let OP_i be a set of operations in S_i and T_i;
 If ($P_i \neq \emptyset$) {
 Let s_k be a state characterized by P_i;
 If (no state in S_i is identical to s_k)
  $S_i := S_i \cup \{s_k\}$;
 }
 Else {
 Let NC_i be conditions obtained by considering OP_i from s_p
 If there is no state s_k characterized by NC_i
  creat s_k;
  $S_i := S_i \cup \{s_k\}$;
 }
 If ($EB_i \neq \emptyset$) {
 Let c_i be a clock constraint corresponding to to_i;
 If ($ET_i \neq \emptyset$)
  add expiry transitions corresponding to ET_i;
 }
 add transition $\langle s_p, s_k, OP_i, \lambda, \gamma \rangle$,
 with λ set of clocks variables reseted and $\gamma = c_i$;
 If ($P_i \neq \emptyset$)
  $s_p := s_k$;
}

図 4: 全体動作仕様の導出アルゴリズム

図4は次の内容を表している.

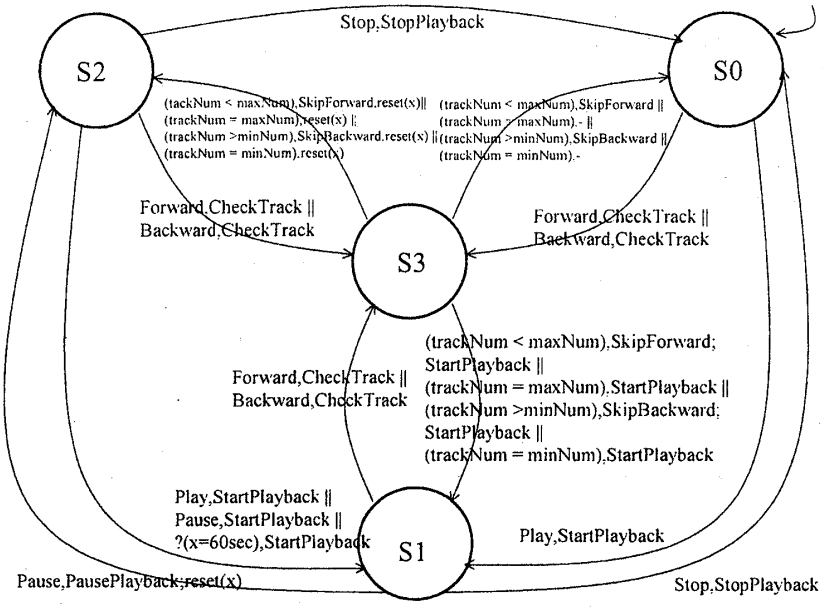

図 5: 図 2 の動作部から導出されたタイムオートマトン

各動作系列のの前提状態 B_{pre} を partial run の初期状態として, GUI アプリケーションのタスクとそれを引き起こしたオブジェクトまたは条件式の組を遷移のラベルとして, タイムオートマトンを合成する. 後置状態 P_i がないものに関しては, 状態遷移先を内部処理で特徴づけられる状態とし, その状態を次の動作の初期状態としてタイムオートマトンを合成する. また, タイムアウトに関しては, タイムアウト時間とタイムアウト後に実行されるタスクの組を遷移のラベルとして, タイムオートマトンを合成する.
タイムオートマトンの合成は, $A_0 = \emptyset$ から開始し, すべての動作部を入力として終了する. 図 1, 図 2, および図 3 で例として示した CD プレーヤーのコンポーネント部と動作部を入力とし, 図 4 のアルゴリズムを用いて導出したタイムオートマトンを図 5 に表す.

このアルゴリズムを用いて合成されたタイムオートマトンは, 入力の動作系列と等価なものであるのは構成の方法から明らかである. このタイムオートマトンを用いることで, 動作部で記述した GUI アプリケーションの部分的な振る舞いだけでなく, GUI アプリケーション全体としての動作内容を容易に把握することが可能となると考えられる. また, 文献 [5] では, タイムオートマトンで記述された実時間システムに対するモデルチェッキングを行うツールである UPPAAL[6] を用いて, 実時間に依存する Audio/Video プロトコルに対する形式記述法および検証法が紹介されている. また, 文献 [7] では, モータバイクのギアチェンジなどを行うギアコントローラに対する形式記述法および検証方が紹介されている. 本手法で合成されたタイムオートマトンに対しても, モデルチェッキングの手法を用いることで, GUI 制御部に対して安全性などの自動検証が可能であると考えられる.

3.3 動作仕様群の導出

3.3.1 問題定義

動作仕様群の導出問題を以下のように定義する.
入力: 3.2 で導出したタイムオートマトン
出力: 各オブジェクトごとのタイムオートマトンと時間制約を処理するタイムオートマトンの組

ここで, 出力のタイムオートマトンの動作を 3.1 で定義したものとし, このタイムオートマトンの組中の通信イベントを ϵ 遷移としたとき, 入力のタイムオートマトンを認識する入力系列と等しく, 時間制約はすべて時間制約を処理するタイムオートマトンに記述するものする.

3.3.2 アルゴリズム

図 6 は次の内容を表している. 入力タイムオートマトンから同一オブジェクトに対する遷移の射影をとる (replace). その遷移系列が各オブジェクトのオートマトンの構成要素となる. なお, 各オブジェクトのオートマトンはいわゆる ϵ 遷移を含んでいることがある. ここで, オブジェクト O_i のオートマトンの遷移 $\langle s_\alpha, s_\beta, T_{i_k} \rangle$ ついて, 入力オートマトンにおいて, 異なるオブジェクト O_j の部分系列と状態 s_α で連結されているとする. このとき, O_i のオートマトンに状態 s_γ から状態

- **Input:** $A = \langle \Sigma, S, S0, C, E \rangle$
- **Output:** $A_i = \langle \Sigma_i, S_i, S0_i, E_i \rangle$,
  $A_T = \langle \Sigma_T, S_T, S0_T, C_T, E_T \rangle$

Let a transition be $\langle s_\alpha, s_\beta, T_{i_k} \rangle$ in A_i
characterized by $\langle s_\alpha, s_\beta, OP_{i_k} \rangle$,
OP_{i_k} be $\langle OB_i, T_{i_k} \rangle$ in A;
Let a set of other transitions in A_i be OT_i;
replace(A_i, OT_i, ϵ);
For each (transition $\langle s_\alpha, s_\beta, T_{i_k} \rangle$ in A_i) {
If ((transition $\langle s_\gamma, s_\alpha, OP_{j_l} \rangle$ in A) $\neq \emptyset$) {
add transition $\langle s_\gamma, s_\alpha, ?M_{\gamma\alpha} \rangle$ to A_i,
Let $\langle ?M_{\gamma\alpha} \rangle$ be the message M_a reception event;
replace the transiton $\langle s_\gamma, s_\alpha, T_{j_l} \rangle$ in A_j
to the transition $\langle s_\gamma, s_\alpha, T_{j_l}; !M_{\gamma\alpha} \rangle$,
Let $\langle !M_{\gamma\alpha} \rangle$ be the message $M_{\gamma\alpha}$ transmission event;
}
}
For each (A_i)
simplify A_i;

図 6: 動作仕様群の導出アルゴリズム

s_α への遷移を示すメッセージ $M_{\gamma\alpha}$ の受信による遷移 $\langle s_\gamma, s_\alpha, ?M_{\gamma\alpha} \rangle$ を付加する．また，O_j のオートマトンの遷移 $\langle s_\gamma, s_\alpha, T_{j_l} \rangle$ をメッセージ $M_{\gamma\alpha}$ の送信イベントを付加した遷移 $\langle s_\gamma, s_\alpha, T_{j_l}; !M_{\gamma\alpha} \rangle$ に置き換える．そして，通信イベントが付加された各オブジェクトのオートマトンの簡約化を行う (Simplify)．

また，時間制約に関しては，クロックをリセットする動作を，時間制約を処理するオートマトンへの通信イベントとし，時間制約条件を時間制約を処理するオートマトンからの通信イベントに置き換える．時間制約を処理するオートマトンに関し，入力オートマトン中で指定されたイベントでクロックをリセットし，指定時間が経過するとタイマイベントを発生させ，初期状態に戻るオートマトンとなる．

アルゴリズムの構成法から，時間制約を処理するオートマトンが時間制約を集中的に管理し，時間制約に関する構造は変化しないため，入力オートマトンの時間制約は保存される．また，各オブジェクトのオートマトンを通信イベント部分で他のオブジェクトのオートマトンと結合させることにより，入力オートマトンを実現できることから，等価性は満たされる．入力オートマトンがデッドロックを持たなければ，出力オートマトンがデッドロックを持たないことも，アルゴリズムの構成法から同様に保証される．

このアルゴリズムを用いて図5のオートマトンを分解した結果を図7から図11に表す．この例では，Play ボタンにユーザからのイベントが発生すると，StartPlayback を実行，通信イベント m01 を発生させ，S1 に遷移する．S1 では，Pause ボタンが m12 を発生させると S2 へ遷移し，また，Stop ボタンが m10 を発生させると，S0 へと戻る．S2 では，ユーザからのイベントが発生すると，StartPlaack を実行，m21 を発生させ，S1 に遷移し，また，Stop ボタンが m20 を発生させると，S0 へ戻る．

このアルゴリズムにより，時間制約を一つのオートマトンに集約することで，以後，各オブジェクトのオートマトンを具体的に実装することが容易となる．つまり，各オブジェクトごとにユーザからのイベントが発生したときに，各状態に応じたタスクを行うように実装すればよい．

4 議論

4.1 関連研究

GUI 開発環境として，Model-Based User Interface Development Environments (MB-UIDEs)[3, 8] がある．これは，ユーザインターフェースの異なる側面を component model として記述し，またそのモデル間の関係を記述することでユーザインターフェースの設計を行うものである．ユーザインターフェースを抽象的に記述することで，ユーザインターフェースの設計や実装が容易となり，その工程も自動化できるなどの利点がある．しかし，文献 [3] の GUI の仕様では，GUI アプリケーションの全体動作仕様が明確でなく，したがって，安全性などの検証に関して考慮された仕様記述ではないと考えられる．また，詳細な実行制御の記述も困難であると考えられる．検証も含めた GUI 開発環境として GUISE[9] がある．これは，仕様の設計からプロトタイプの作成までの部分に焦点を当て，PGD モデル [10] に基づいて，GUI の各状態を自動検証することによるテスト量の削減，形式的仕様からプロトタイプの自動生成を行うことによるプログラミング量の削減ができるという利点がある．しかし，クラス構造，コンポーネント化など，GUI に特有なオブジェクト指向モデルを考慮した仕様であるかは明確でなく，時間制約に関する記述および検証も行われていない．

その他様々な GUI 開発環境および GUI 制御部の形式的仕様記述に関する研究が行われているが [11, 12]，いずれも時間制約に関する記述および，GUI アプリケーションの全体動作に対する検証は行われていない．

4.2 GUI 仕様記述の改良

文献 [9] では，本手法で用いている有限状態機械に基づくモデルの問題点として，状態爆発の問題を取り上げている．つまり，GUI が大規模である場合，構成される有限状態機械も非常に多くの状態を含むものになり．また，GUI では一時に受付けることのできる入力の種類が多く，各状態からの遷移数が多くなる．そのため，単純に GUI の形式的仕様を記述するとその記述量が膨大となり，また，検証を行う際には，すべての状態と遷移をしらみ潰しに調べるという方法では，いわゆる状態爆発が起こり，妥当な時間で検証を行えないことになる．

しかし，本手法では，コンポーネント部を階層化構造で記述することで状態爆発の問題を解決できると思われる．すなわち，コンポーネント部で，似た機能をもたオブジェクトに関しては，同じ親を持つように記述し，また，似た機能をもつ内部処理に関しても，同じ親を持つように記述する．そして，動作部で子に関する動作を

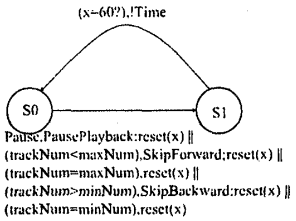

図 7: 時間制約を処理するオートマトン

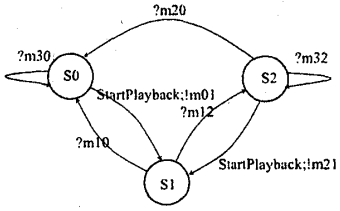

図 8: Play ボタンのオートマトン

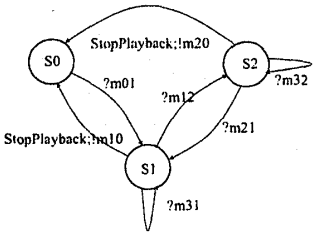

図 9: Stop ボタンのオートマトン

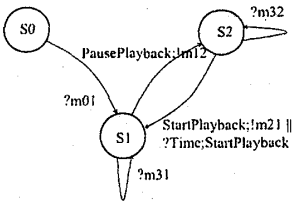

図 10: Pause ボタンのオートマトン

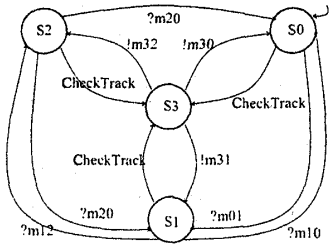

図 11: Foward ボタンのオートマトン

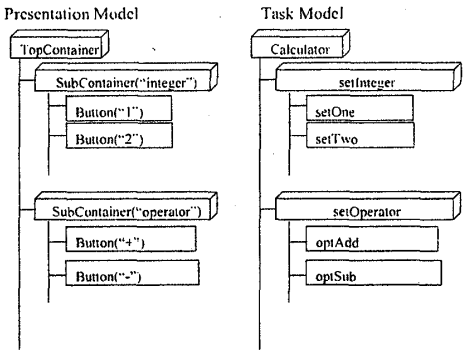

図 12: 電卓の仕様記述の一部

記述するのではなく, である親に関する記述を行う. 例として, 電卓を考える. 図 12 は電卓のコンポーネント部である. 動作部においては, 各ボタンごとに実行される各内部処理を記述するのではなく, 親オブジェクトごとに実行される親 task を記述する. このように, 親単位で動作部を記述することで, GUI 制御部の動作を抽象化した形で記述することが可能となり, 導出されたタイムオートマトンが含む状態数と各状態からの遷移数も減少でき, 状態爆発も避けることができる. そして, 抽象化された動作部に対して安全性などが確認されれば, その各子に関する動作の安全性なども確認することができると考えられる. 図 12 の例では, 各テンキーボタンを押すことで対応した値がセットされること, 各演算ボタンを押すことで対応した演算が行うことに対して安全性などが確認できる.

また, GUI アプリケーションにおいて, 一つのウィンドウに含まれるオブジェクトが非常に多いと, 利用者から見ると非常に分かりにくいアプリケーションであると考えることができるため, 質の良い GUI アプリケーションを各ウィンドウごとに状態機械で記述すると, 状態機械に含まれる状態数, 遷移数ともに状態爆発を引き起こす程の記述量にはならないと考えられる. よって, 本手法において, GUI アプリケーションを構成する各ウィンドウ単位での仕様記述を行い, その各仕様記述ごとにタイムオートマトンを導出すれば, 状態爆発は避けられることができる.

4.3 時間制約処理の実現法

時間制約処理のうち, タイムアウト処理や周期的な動作処理などに関する処理の実現方法の一つとして, スレッドを用いる方法が考えられる. スレッドを用いることにより, 時間制約処理に関するプログラムが簡潔になる. 例として, CD プレーヤーの各状態 (再生・一時停止・停止) をアニメーションで表示することを考える. つまり, 再生状態では, 画像を描き変える処理を実行し, 一定時間スレッドを停止させる. 時間経過後, 再び画像を描き変える処理を実行する. この一連の処理を実行することでアニメーションを表示する. 一時停止状態で

```
\begin{behavior}
@*
 :Button("play"),StartPlayback
                ;Animation.start: >play
 :Button("stop"),StopPlayback
                ;Anime.stop
                ;DrawStopState: >play
@play
 :Button("pause"),PausePlayBack
                 ;Anime.stop: >pause
@pause
 :Button("pause"),StartPlayback
                 ;Anime.start: >play
\end{behavior}

\begin{thread}
Anime:sleep(0.01sec),DrawPlayState:
\end{thread}
```

図 13: アニメーションを表示する動作部の一部

はスレッドを停止させ, 再生状態になると再び画像を描き変える処理を実行する. 停止状態では, スレッドを停止させ, 停止状態を表す画像を表示する.
図 13 はアニメーションの表示の処理を追加した CD プレーヤーの動作部の一部である. 動作部には, 内部処理の一部として実行を開始再開・または停止するスレッドを記述する. 図 13 では, play ボタンが押されると Anime スレッドを実行すること, stop ボタンが押されるとスレッドを停止することが記述されている. そしてスレッドに関する処理については, 各スレッドごとの動作を記述する. 図 13 では, Anime スレッドでは, DrawPlayState を 0.01sec ごとに実行することを記述している.

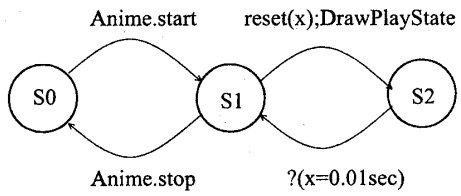

図 14: スレッド部を記述したタイムオートマトン

このように記述されたスレッドに対して, 図 14 のようなタイムオートマトンで記述されたモデルを合成することができる. よってスレッドを用いて時間制約を実行する場合でも安全性などの自動検証は可能であると考えられる.

5 あとがき

本稿では, GUI 制御部の仕様記述から, GUI の各部品ごとの動作仕様群であるタイムオートマトンを生成する方法について述べた. 本手法により, 全体動作仕様の把握が容易となること, オブジェクト指向プログラミングによる実装が容易となること, 安全性などの自動検証が可能となると考えられる.
今後の課題としては, 時間制約の他に, アプリケーションの内部処理がインターフェースに反映される仕様を含めた GUI 制御部の記述の考案や, 実用例題で本手法の有効性を調べるために, 本稿で示した仕様記述から GUI アプリケーションを生成するシステムの実装を行うことがあげられる.

参考文献

[1] Rajeev Alur and David Dill : "A theory of timed automata," In *Theoretical Computer Science*, pp. 45–73, 1994.

[2] 山本亮, 岡野浩三, 東野輝夫, 谷口健一 : "GUI 制御部の記述と実現の一手法", 情報処理学会研究報告, Vol. 97, No. 78, 97–PRO–14–16, pp. 91–96, 1997.

[3] Paulo Pinheiro da Silva, Tony Griffiths, and Norman W. Paton : "Generating User Interface Code in a Model Based User Interface Development Environment," In *Advanced Visual Interfaces*, pp. 155–160, 2000.

[4] Stephane Some and Rachida Dssouli : "Torward and Automation of Requirements Engineering using Scenarios," *Computing and Information*, Vol. 2, No. 1, pp. 1070–1092, 1996.

[5] Klaus Havelund, Arne Skou, Kim G. Larsen, and Kristian Lund : "Formal Modeling and Analysis of an Audio/Video Protocol: An Industrial Case Study Using UPPAAL," In *Proc. of the 18th IEEE Real-Time Systems Symposium*, pp. 14–24, 1997.

[6] Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi : "UPPAAL in 1995," In *Proc. of the 2th Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, No. 1055, pp. 431–434, 1996.

[7] Magnus Lindahl, Paul Pettersson, and Wang Yi : "Formal Design and Analysis of a Gear-Box Controller," In *Proc. of the 4th Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, No. 1384 in Lecture Notes in Computer Science, pp. 281–297, 1998.

[8] Paulo Pinheiro da Silva : "User Interface Declarative Models and Development Environments: A Survey," In *Design Specification, and Verification of Interactive Systems*, pp. 207–226, 2000.

[9] 尾崎寛和, 渋谷雄, 辻野嘉宏 : "形式的仕様に基づく GUI 開発環境の試作", 情報処理学会研究報告, Vol. 2001, No. 3, 2001–HI–92–8, pp. 55–62, 2001.

[10] 辻野嘉宏 : "GUI ダイアログのための検証法について", 電子情報通信学会論文誌, Vol. J82–D–I, No. 10, pp. 1286–1294, 1999.

[11] Angle Puerta and Jacob Eisenstein : "Interactively Mapping Task Models to Interfaces in MOBI-D," In *Design Specification, and Verification of Interactive Systems*, pp. 261–273, 1998.

[12] Jean–Pierre Jacquot and Dominique Quesnot : "Early Specification of User-Interface: Toward a Formal Approach," In *Software Engineering*, pp. 150–160, 1998.