プログラミング 14−16
(1997. 8. 22)

GUI制御部の記述と実現の一手法

山本 亮, 岡野 浩三, 東野 輝夫, 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 ソフトウェア科学分野

概要

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) プログラムにおいて, GUIの制御部はより複雑になる傾向にある. そのような制御部の設計法のひとつとして, まず, 各GUI部品 (オブジェクト) とイベントの表 (オブジェクト属性表), さらに, 内部処理の集合を決定し, GUIの動作を記述したシナリオ群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の動作仕様群を自動導出するという方法を提案する. 本手法の利点は, (a) シナリオが優先度を持つこと, (b) 時間制約を記述できること, (c) 全体仕様と各オブジェクトごとの動作仕様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ムによる実装が容易になることである.

Synthesis of a timed automaton from scenarios which describe GUI's behavior and its division into timed automata for GUI objects

Akira Yamamoto, Kozo Okano, Teruo Higashino, Kenichi Taniguchi
Division of Software Science, Department of Informatics and Mathematical Science,
Graduate School of Engineering Science, Osaka University

Abstract

Recently, controllers of Graphical User Interface (GUI) programs become rather complex. We propose one method for designing such GUI controllers. In our method, first, a whole specification (abstract level specification) of a GUI's controller is derived automatically as a timed automaton from scenarios which describe the GUI's partial behavior, using a relation table of events and objects (corresponding to widgets such as buttons, scroll bar and so on) and a name list of primitive functions used in the program. The timed automaton is automatically divided into a set of concurrent timed automata (lower level specification), each of which describes behavior of the GUI's object. Advantages of our method are the following. (a) Each scenario has a priority. The priority is used to derive a rather complex whole specification from simple scenarios. (b) Timing restrictions such as time-out and wait and so on, can be described. (c) The method derives two levels of specifications. The abstract level specification helps us to understand the whole behavior of the GUI program. The lower level specification helps us to implement the program in the object oriented programming style.

1 まえがき

グラフィカルユーザインターフェース (GUI) は Seeheim モデルにおいて, プログラムの外観部, 制御部, 内部関数部の三つから構成される [1]. 近年の大規模な GUI プログラムにおいては制御部が複雑になる傾向にある. このような制御部を正確に記述するための方法として, インタラクティブなプログラムのシナリオ記述群からそのプログラムの制御フローを導出する研究がいくつかなされている [3][4]. 例えば, 文献 [4] では銀行の ATM のシナリオ記述群から, タイムオートマトンモデル [5] で記述された制御部の動作仕様を自動導出する方法を提案している. この方法では時間制約を課したシナリオ [2] からタイムオートマトンを導出しており, 時間制約を本質的に持つ GUI の制御部の記述に向いていると思われる. しかしながら, これらの導出方法を GUI プログラムに適用するには以下のような問題がある. 一つは, オブジェクトの概念があまり考慮に入れられていないことである. GUI プログラムを実装する際には, GUI の各部品をオブジェクトととらえ, 各オブジェクトごとの動作内容をプログラムする. 従来手法ではひとつのタイムオートマトンが導出されるだけであり, ここからオブジェクト指向で実装するには距離がある. 二つ目の問題点は, シナリオの記述のしやすさの問題である. 一般にシナリオ記述者が時間制約の記述のバランスをとりながら, 例外動作を含めシステム全体の動作内容すべてをシナリオとして記述することは非常に困難であろう.

そこで, 本稿では以下の導出方法を提案する.

システム設計者は, イベント, 内部処理, GUI 部品を定めたのち, システムの一般的な動作を複数のシナリオとして記述する. 各シナリオは, システムの代表的な振る舞いを時間制約込みで「どの状況でどの GUI 部品がどのイベントに対してどの基本処理を行うか」という動作の列として記述される. 様々な状況で起こりえるが, 主要処理からはずれる例外的な動作は, 優先度の高い別のシナリオとして記述する. これらのシナリオ間に矛盾する内容が書かれていた場合は, 優先度の高い記述の内容が優先されるとする. このようにして与えられたシナリオ群から, GUI の全体動作記述を自動導出する. 導出の際, 同じ優先度の矛盾が発見されれば, 設計者にその旨を通知する. 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を見ることにより, システム全体の動作内容を正確に把握することが可能になる.

本提案法では, 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. 各動作仕様は, お互いに内部通信イベントを用いて, 部分的に同期することにより入力のオートマトンを模倣するタイムオートマトンである. 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, 各オートマトンを具体的に実装することを容易にしている.

以降, 2 章では, シナリオの定義と記述例, および, オブジェクト属性表について述べる. 3 章では, タイムオートマトンを用いた全体仕様の定義, 及び, 各オブジェクトと大域的状態遷移の動作仕様の定義と導出例について述べる. 4 章では, 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. 5 章はあとがきである.

2 シナリオ

2.1 シナリオの定義

シナリオとは, 対象となるシステムや環境の, ある制限された状態における部分的な振る舞いを記述したものである [2]. 本稿では, シナリオは形式的に以下のような構文で定義される.

⟨ シナリオ ⟩	::=	⟨ 優先度 ⟩⟨ 前提条件 ⟩⟨ 文 ⟩$^+$
		| ⟨ 前提条件 ⟩⟨ 文 ⟩$^+$ | ⟨ 文 ⟩$^+$
⟨ 優先度 ⟩	::=	整数
⟨ 前提条件 ⟩	::=	オブジェクトまたは大域的な状態に関する述語
⟨ 文 ⟩	::=	⟨ 時間制約 ⟩⟨ 動作 ⟩$^+$⟨ 後置条件 ⟩⟨ 例外 ⟩
		| ⟨ 時間制約 ⟩⟨ 動作 ⟩$^+$⟨ 後置条件 ⟩
		| ⟨ 動作 ⟩$^+$⟨ 後置条件 ⟩ | ⟨ 動作 ⟩$^+$
⟨ 動作 ⟩	::=	⟨ オブジェクト ⟩⟨ イベント ⟩⟨ 関数 ⟩
⟨ 例外 ⟩	::=	⟨ 動作 ⟩$^+$
⟨ 時間制約 ⟩	::=	時間に関する述語
⟨ 後置条件 ⟩	::=	前提条件に等しい

シナリオは優先度, 前提条件 (ともに省略可) と文の繰り返しからなる. 優先度は, 0 以上の任意の整数で, 大きな数ほど優先される. 前提条件は, そのシナリオがどのような状態での動作を表すものかを示すもので, オブジェクトまたは大域的な状態に関する述語である. 優先度が省略された場合, デフォルト値 (例えば 5) が適用され, 前提条件に "*" を用いた場合は, すべての状態に対してのシナリオとみなされる.

文は, 時間制約, 動作の繰り返し, 後置条件からなる. 時間制約と後置条件は省略可能である. 時間制

約は時間に関する述語であり, After, Wait と Until がある. After, Wait が指定された場合, 続く動作は, 直前の動作の終了から指定された時間が経過した直後に実行されなければならない. ただし, Wait では待機中に処理がブロックされるのに対し, After ではブロックされず, 指定時間経過後に指定した動作が実行される. つまり, After は非同期に発生するタイマーイベントによって呼び出される動作と考える. このとき, After そのものがイベントとなるため, After によって呼び出される動作には特にオブジェクトを指定せず, "–" を使用する. また, Until が指定された場合は, 直前の動作の終了から指定された時間内に動作が終了しなければならないことを表す. さらに, この場合にのみ, 指定時間内に処理が行われなかったときの動作を例外処理として指定しなければならない.

動作はオブジェクトとそれに対するイベント, そしてそのとき実行される関数の対からなる. すべてのオブジェクトやすべてのイベントを指定するときは "*" を使用することができる.

後置条件は, 前提条件と同じく, オブジェクトまたは大域的な状態に関する述語であるが, 動作の繰り返しの実行後は, これを満たさなければならないという意味を表す.

ここで, 例としてリモートビデオコントローラーについてのシナリオの一部を表す.

- シナリオ 1
  When Out_of_Control
  ⟨Apply, Push, apply_control⟩
  ⟨Apply, Permit,
  change_label_permitted, Under_Control⟩
  ⟨After(60)⟩
  ⟨–, –,change_label_rejected, Out_of_Control⟩
- シナリオ 2
  When Out_of_Control
  ⟨Apply, Push, apply_control⟩
  ⟨Apply, Rejected, reinit, Out_of_Control⟩
- シナリオ 3
  When Under_Control
  ⟨Pause, Push, pause_video, Paused⟩
- シナリオ 4
  When Paused
  ⟨Pause, Push, replay_video, Under_Control⟩
- シナリオ 5
  When *
  ⟨Quit, Push, quit_app, Quit⟩

リモートビデオコントローラーは, 遠隔地のビデオカメラの操作権を入手すると, 60 秒間操作できるというアプリケーションである.

シナリオ 1 は次の内容を表している. 操作権がない (Out_of_Control が真) のとき, Apply ボタンを押す (Apply ボタンに Push イベントが発生) と, apply_control を実行する. 操作権が得られる (Apply ボタンに Permit イベントが発生) と, ラベルの表示を変更 (change_label_permitted を実行) して操作権が得られた状態に移る (Under_Control を真にする). そして 60 秒後に (After(60) が真となるとき) ラベル表示を変更 (change_label_rejected を実行) し, 操作権が取り消される (Out_of_Control が真となる).

同様に, シナリオ 3 と 4 は, Pause ボタンが押されるとビデオを Pause し, 再び押されると解除することを表している.

2.2 オブジェクト属性表

オブジェクト属性表とは, 対象となるアプリケーションを構成する各オブジェクトについて, クラス, 名前, 発生するイベントを与えるものである.

ここで, 2.1 のリモートビデオコントローラーについてのオブジェクト属性表を表す.

Class	Name	Event
Display	Disp	None
Button	Up	Push
	Down	Push
	Right	Push
	Left	Push
	Pause	Push
	Apply	Push, Permit, Reject
	Quit	Push
Label	Label	None

3 タイムオートマトン

タイムオートマトン (timed automata)[5] により, 全体仕様, 及び, 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, 時間制約付きの状態遷移が記述できる. 状態遷移表 (timed transition table) は次の 5 項組 $\langle \Sigma, S, S_o, C, E \rangle$ 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 $\langle s, s', \sigma, \lambda, \delta \rangle \in E$ は次の意味をもつ. 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 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 ($\leq$) 及びそれらの論理積, 論理否定で構成される.

タイムオートマトン $\mathcal{A}$によって認識される timed word (σ, τ) は入力記号系列 $\sigma = \sigma_1, \cdots, \sigma_n$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 $r\,(\overline{s}, \overline{v})$ 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:正の実数) 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 $v_0(x) = 0$
- すべての $i > 0$ について，$\langle s_{i-1}, s_i, \sigma_i, \lambda_i, \delta \rangle$ 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．

ある run r の部分系列

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

を r の partial run として定義する．

本導出法では，全体仕様のタイムオートマトンの入力記号と，シナリオの構成要素を以下の様に対応づけする．

シナリオに表れるオブジェクトの集合を $\mathcal{O}$，イベントの集合を $\mathcal{E}$，基本内部関数の集合を $\mathcal{P}$ とする．ここで特に，$\mathcal{P} \cap \mathcal{E} = \emptyset$ とする．

$\Sigma = \{\langle o, a \rangle \mid o \in O, a \in \mathcal{P} \cup \mathcal{E}\}$．ここでの意図は，シナリオにおけるイベントや内部関数を区別せず，これらと，オブジェクトの組をタイムオートマトンの入力記号と見なすことである．この対応のもとでタイムオートマトンの動作の解釈を行う．

動作仕様群は，このようなタイムオートマトンの $n+1$ 項組 (ここで n はオブジェクトの数) として表現される．動作仕様群としてのタイムオートマトンの $n+1$ 項組に対して，以下の制約を置く．

(1). 各オブジェクト i に対応するタイムオートマトン $\mathcal{A}_i$の入力記号は $\langle i, a \rangle$ のみとする．ここで，a はオブジェクト i にオブジェクト属性表において関連付けられたイベントや内部動作か，通信イベントである．

(2). 時間制約を処理するタイムオートマトン $\mathcal{A}_c$の入力記号は $\langle N, a \rangle$ とする．ここで，N はすべてのオブジェクトと相異なる記号，a は通信イベントとする．

上述のような組のタイムオートマトンの動作を以下のように定義する．初期状態は各タイムオートマトンの初期状態組とする．通信イベントはコンカレントに発生するものとする．すなわち，ある通信イベントに対して，それを受理できるオートマトンすべてがそれを受理する．このときオートマトンの組の時間制約は，受理できるオートマトンすべての制約の論理積とする．

4 導出問題，アルゴリズム

4.1 シナリオ群からのタイムオートマトンの導出

4.1.1 問題定義

シナリオ群から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：シナリオの集合，オブジェクト属性表，内部処理の集合

出力：ある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以下の性質を満たす．まず，入力されたすべてのシナリオそれぞれに対して，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．また，partial run がオートマトンで受理されないシナリオが存在する場合は，そのシナリオにオーバーラップする，より優先度の高いシナリオが存在し，それを生成できる partial run が受理されなければならない．

4.1.2 アルゴリズム

オートマトンを生成するアルゴリズムは，文献 [4] に従う．文献 [4] のアルゴリズムは，後置条件の組合せによって状態分割を行い，シナリオ中の動作を遷移のラベルとしてオートマトンを合成する．また，オブジェクト属性表に記述されていないオブジェクト，イベントを使用した場合や，内部関数のリスト中に存在しない関数を用いた場合はユーザーにその旨を通知する．

時間制約に関しては以下のように合成を行う．

Wait 時間制約のついた遷移を作成するだけでよい (図 1-(a))．

Until 時間内にすべきすべての動作の遷移先から，例外処理への時間制約の付いた遷移を作成する (図 1-(b))．

After それに関するクロックを初期化する遷移を除いてその状態から到達可能なすべての状態に，指定した動作を行う遷移を付加することでオートマトンを構成する (図 1-(c)(d))．

本稿では, 到達可能な状態中に再びAfterの開始状態が表れることがないという仮定を置く. また, Afterで指定した動作が終了した結果, 後置条件が変化する場合, 付加する遷移の遷移先は既存の状態, もしくは新規の状態となる (図1-(c)). 変化しない場合は, 自己ループが付加される (図1-(d)). さらに, 他のシナリオがAfterの合成より後に合成された場合, 到達可能な状態が変化する場合があるので, シナリオを合成するたびに到達可能な状態をチェックし, 異なっていれば再構成する.

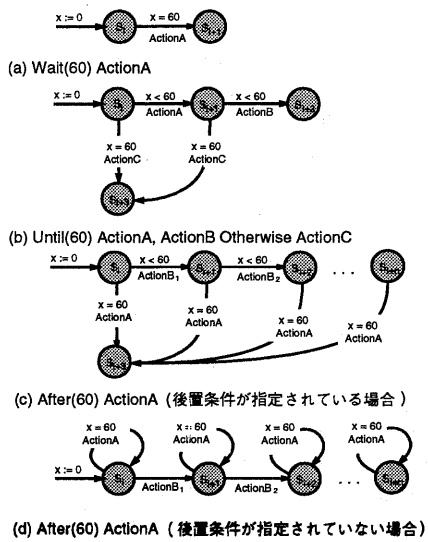

図 1: 時間制約に関する合成例

また, 文献 [4] は優先度を考慮していない. そのため, 優先度に関する部分は以下の方針に基づいてオートマトンを作成する. ここで, 矛盾とは, ある状態において, 同一オブジェクトに同一イベントが発生した際, 異なる動作が存在することを意味する.

(1). あるシナリオScAに対するオートマトンを合成する際, ScA中の動作がそれまでに合成されたオートマトン中の動作と矛盾していなければ, 優先度を考慮せずオートマトンを作成する.
(2). あるシナリオA中の動作が, より優先度の低いシナリオScB中の動作と矛盾していれば, ScB中の動作でScAと矛盾する遷移を削除し, 新たにScA中の動作に対応する遷移を生成する.
(3). 同じ優先度のシナリオ間に矛盾が存在する場合は, ユーザーに通知し, 合成を中止する.

2.1, 2.2で例として示したリモートビデオコントローラーのシナリオ群とオブジェクト属性表, さらに内部処理の集合を入力とし, このアルゴリズムを用いて導出した例を図2に表す. ここで, Afterに関する遷移は省略形を用いて記述している. 図の点線で表されている遷移は, その遷移が出ている状態から到達可能な状態 (Afterに関するクロックを初期化する遷移を除く) から点線の遷移先への遷移が存在することを表している. 例えば, シナリオ1は図2中の $S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow S_0$ の遷移に, シナリオ2は $S_0 \rightarrow S_1 \rightarrow S_0$ の遷移に対応している.

このアルゴリズムを用いて合成されたオートマトンは, 入力のシナリオと, 優先度を含めて, 等価なものであるのは構成の方法からほぼ明らかである. また, 時間制約に関しては, 図1の構成法に則して考えれば明らかであろう.

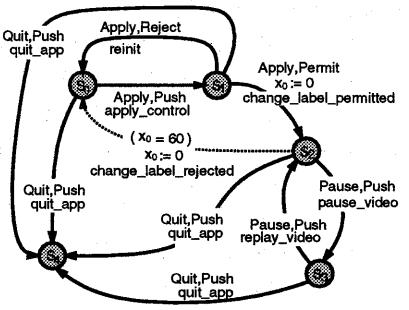

図 2: 例シナリオから導出されたタイムオートマトン

4.2 タイムオートマトンの分解

4.2.1 問題定義

分解問題を以下のように定義する.
入力：4.1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章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と認識する入力系列が等しく, 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とする.

4.2.2 アルゴリズム

まず, 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. 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. ここで, 入力オートマトンにおいて, 異なるオブジェク

ト O_i, O_jの部分系列が状態 S_kで連結されているとする. このとき, O_iのオートマトン中の S_kに, 通信イベント C_iによる遷移と, その遷移先 S_αを付加する. 既に S_αが存在する場合は, それを用いる. そして, O_jのオートマトン中の S'_k(入力オートマトンにおいては S_k と同一) に, 状態 S_βと C_i(S_kから S_αへの遷移の入力に等しい) を入力とする S_βから S'_kへの遷移を付加する. S_βが既に存在する場合は, 新たな状態を付加しない. あるオブジェクト O のオートマトンにおいて, 上述の S_α, S_βにあたる状態が存在し, 入力オートマトンにおいて S_αを導く S_kと, S_βから導かれる S'_kが到達可能な関係にあれば, S_αと S_βを一つの状態 S_γにまとめる. 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は, 入力オートマトン中のすべての時間制約に関し, 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ー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ほとんど変化しないため, 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. 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 2 のオートマトンを分解した結果を図 3 ～ 6 に表す.

この例では, Apply ボタンは Push イベントで S_1へ遷移, Permit イベントで change_label_permitted を実行, 通信イベント ⟨Com1⟩ を発生させ[1], S_2へ遷移する. そして時間制約を処理するオートマトンからタイマーイベントを表す通信イベント ⟨Time⟩ を受取り[1], S_0へと戻る. Pause ボタンは, Apply ボタンが ⟨Com1⟩ を発生[1]させると, S_1へと遷移する. そして, Push イベントが発生するたびに, S_1, S_2を往復する. 時間制約を処理するオートマトンで ⟨Time⟩

[1]オートマトンの動作としては入力として解釈する

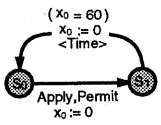

図 3: 時間制約を処理するオートマトン

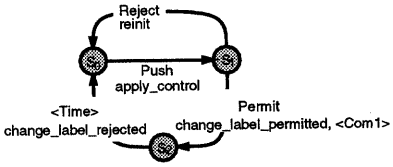

図 4: Apply ボタンのオートマトン

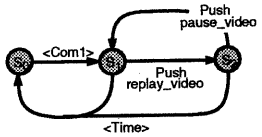

図 5: Pause ボタンのオートマトン

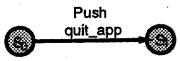

図 6: Quit ボタンのオートマトン

が発生すると, S_0へと戻る.

5 あとがき

本稿では, 優先順位のついたシナリオ群から, GUI の各部品ごとの動作仕様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. 実現環境によっては, 各ウインドウを独立したオブジェクトとみなすこともあり, この場合は時間制約を処理するオートマトンを用いない動作仕様を導出すべきであろう. これは 4 章後半の導出法を変更することにより, 容易に実現できると考える. 実用例題で本手法の有効性を調べることが今後の課題のひとつである.

参考文献

[1] Dan R. Olsen. Jr : "User Interface Managiment System", Morgan Kaufmann Publishers, Inc., 1992.

[2] Kevin M. Benner, Martin S. Feather, W Lewis Johnson, and Lorna A. Zorman : "Utilizing Scenarios in the Software Development Process", Informatioin System Development Process, pp.117-134, 1993.

[3] Kai Koskimies and Erkki Mäkinen : "Automatic Synthesis of State Machines from Trace Diagrams", Software-Practice and Experience, Vol.24, No.7, pp.643-658, 1994.

[4] Stèphane Somè and Rachida Dssouli : "Toward and Automation of Requirements Engineering using Scenarios", Journal of Computing and Information, Vol.2, No.1, pp.1070-1092, 1996.

[5] Rajeev Alur and David Dill : "A theory of timed automata", Theoretical Computer Science, pp.45-73, 1994.