2007－SE－157 (11)
2007／9／28

状態指向の状態遷移表

紫合 治‡

状態遷移表は，状態遷移の漏れのチェックに適しており，また特別のツールがなくても簡単に使えるので，実際のシステムの設計に広く使われている．しかしシステムが複雑になると，状態数やイベント数が増加し，状態遷移表も大きくなり，全ての項目のチェックは難しくなる．ここでは，状態遷移の漏れのチェックの手間を減少させるために，Kripke 構造を応用した状態指向の状態遷移表を提案する．これは，システムの環境(コンテキスト)の状態を，Kripke 構造の原子命題に対応させることにより，その状態で起こりうるイベントの集合を明示的に示すものである．

State Oriented State Transition Table

Osamu Shigo‡

State transition table is widely applied for reliable large-scale system design without special drawing tools. State transition table is more suitable than state transition diagrams to check the lack of transition design. However, for complex system, state transition table contains a large number of transition items, and it is almost impossible to check all transition items. To reduce the number of possible transition items in the table, we have introduced the state oriented concept, which is similar to the Kripke structure, to the state transition table. Here, states of the environment of the system, or the states of context, are specified as atomic propositions in Kripke structures, which determine the set of possible events at the system state.

1. はじめに

ユビキタス社会の進展で信頼性の高い組込みソフトウェアの需要が高まりつつある．多くの組込みソフトウェアでは，実時間処理等の複雑な設計を適切の表現するために，システムの動的な振舞いの設計が重要になる．このため，UML[1]や SDL[2]の状態遷移図などが広く使われている．

システムが複雑になると，状態数やイベント数が増加し，1つの図に記述するのは難しくなってくる．そこで，図ではなく状態とイベントのマトリクスからなる状態遷移表が，実際の現場ではよく使われている．状態遷移表は状態遷移図の場合のような特別なツールを使わなくてもエクセル等の表ツールで簡単に作成することができる．また起こりうる状態遷移の漏れは，マトリクスの項目をつぶさに調べることでチェックできる．このため，誤りのない詳細な状態遷移の設計では，図よりも表の方が適している場合も多い[3]．

しかし，状態遷移表が大きくなると，遷移項目(状態とイベントの組合せ)の数が膨大になり，全てをチェックすることは難しくなってくる．一般に，ある状態では特定のイベントだけが発生しうるため，本来チェックすべき項目は制限されている．各状態で起こりうるイベントが事前に明示的になっていれば，チェックの手間を大幅に減少することができる．

ここでは従来の状態遷移表に状態指向の概念を導入して，本来チェックすべき項目を自動的に検出する方式を提案する．状態指向の状態遷移表は，モデル検査[4]の基盤としてよく知られている Kripke 構造のように，各状態が「どうなっているか(to be)」をシステムの環境(コンテキスト)にある装置や他システムの状態によって規定する．これにより，各状態で起こりうるイベント集合が，環境の仕様(プロトコル)から自動的に求めることができる．なお，従来の状態機械のように，状態遷移に伴って「何をするか(to do)」を規定したものを遷移指向の状態遷移という．

以下に，2 節で，遷移指向と状態指向の状態遷移について説明し，3 節で，状態指向の状態遷移表の基本機能について，また 4 節でその拡張機能について述べる．5 節では，簡単な電話交換機の設計例を状態指

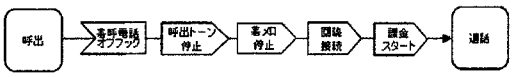
‡東京電機大学 情報環境学部
School of information Environment, Tokyo Denki University

向の状態遷移表で示す. 6節で関連研究を議論し, 最後に7節で, まとめと今後の課題について述べる.

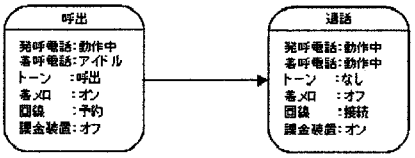
2. 遷移指向と状態指向

UML の状態マシン[1]や SDL の状態遷移図[2]では, 状態遷移に伴って「何をするか(to do)」をアクションとして記述する. これは状態の中に記述することもできるが, その場合も状態への(からの)遷移に伴うアクション記述である. つまり, 主な機能は, 遷移時に何をするかを記述することによって表現される. ここではこれを「遷移指向」という. 一方, モデル検査等で最近注目されている Kripke 構造[4]では, 状態の遷移には何も指定しないで, 状態そのものが「どうなっているか(to be)」を原子命題の集合によって記述する. これは, システムの状態遷移で, 遷移に伴って何をするか(アクション)を記述するのでなく, 各状態で環境がどうなっているかを記述し, 状態遷移でシステムが環境をどう変えるかを環境の状態の変化によって示す. このように状態がどうなっているかを記述する方式を「状態指向」という[5].

例として, 電話交換で呼出状態から通話状態への遷移を考える. 従来の遷移指向では, この遷移は, 着呼電話の"オフフック"イベントによって起こり, 遷移に伴うアクションとして, 発呼電話の呼出トーンと着呼電話の着メロを止め, 回線を接続し, 課金を開始することを記述する(図 1(1)). 一方, 状態指向では, 呼出では「発呼電話は動作中, 着呼電話はアイドル, 呼び出しトーンと着メロが鳴っており, 回線は予約, 課金はオフ」の状態, 通話では「発呼電話も着呼電話も動作中, トーンや着メロは停止, 回線は接続, 課金はオン」の状態であることを記述する(図1(2)).



(1) 遷移指向の状態遷移



(2) 状態指向の状態遷移

図1. 遷移指向と状態指向

このように, 遷移指向は「手続き的」, 状態指向は「宣言的」な記述になり, どちらが分かりやすいかは場合によって異なる. 一般には, 概要的な記述には宣言的な状態指向が, 詳細な記述には手続き的な遷移指向が適していることが多い. さらに, 完全な状態指向ではなく, 状態指向をベースとして, それに遷移指向的な記述もオプション的に付加していく方式が現実的であると思われる.

3. 状態指向状態遷移表の基本機能

3.1. プロトコルの規定

最初に設計対象コンポーネント C の環境(コンテキスト)を規定する. 環境は, C と直接信号のやり取りをする装置や他コンポーネントによって定められる. これらの装置や他コンポーネントとのやり取りの窓口を「ポート」と呼ぶ. つまり, C はポートを通して環境と信号のやり取りをする.

信号は, C が環境に対して働きかける「出力信号」と, 環境が C に対して働きかける「入力信号」に分けられる. ポートに対するこれらの信号の規定を「プロトコル」と呼ぶ. プロトコルは, 入力信号の集合, 出力信号の集合, これらの信号の妥当な出現順を定める状態遷移表によって規定される.

設計対象コンポーネントの例として ATM 制御を考える. 以下に, ATM 制御のパスワードエラーやカードのアカウントエラーに対する処理のみを抜き出して考える[6]. コンポーネントのポートとしては, カード挿入装置につながるカードポート, パスワード入力につながるパスワードポート, 銀行の認証につながる認証ポートなどがある(図 2). 実際には, この他に引出, 預入, 払込等の処理に対するポート(出金口, 入金口, 金額入力, 銀行データベース等)があるが, 簡単のためここでは省略する.

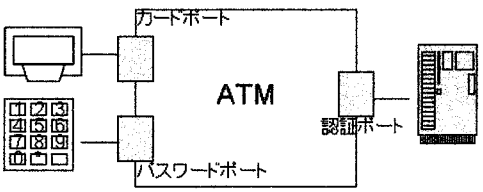


図 2. 設計対象コンポーネントとポート

表1にこれらのポートのプロトコル状態遷移表を示す. (1)はカードポートのプロトコルを規定したもので, 3

表1. ポートのプロトコルを示す状態遷移表

(1) カードポートのプロトコル

No	状態名	入力信号			出力信号	
		カード挿入	キャンセル	カード取出	カード要求	カード排出
0	アイドル				1	
1	カード挿入待ち	2				
2	カード挿入中		3			4
3	キャンセル発生					4
4	カード取出待ち			0		

(2) パスワードポートのプロトコル

No	状態名	入力	出力	
		パスワード入力	パスワード要求	中断
0	アイドル		1	
1	パスワード入力中	0		0

(3) 認証ポートのプロトコル

No	状態名	入力信号		出力信号	
		パスワードエラー	アカウントエラー	認証要求	キャンセル
0	アイドル			1	
1	認証中	0	0		0

つの入力信号(カード挿入, カード取出, キャンセル)と2つの出力信号(カード要求, カード排出)をもつ. 初期状態(No=0)のアイドルでは, ポートにカード要求信号のみを出力でき, その結果, カード挿入待ち状態になる. カード挿入待ち状態では, カード挿入信号が入力され, カード挿入中状態になる. カード挿入中では, キャンセルが入力される可能性があり, その時はキャンセル発生状態になる. また, キャンセルが発生する前に, コンポーネントからカード排出信号を出力して, カード取出待ち状態に移ることもできる. 同様に, (2)にパスワードポート, (3)に認証ポートのプロトコルを示す.

ポートのプロトコルは, そのポートにつながる装置の仕様を規定している[7]. この規定をみたす限り, どんな装置(別コンポーネント)でも接続できる. 複数のポートが同じプロトコルを持つことができ, ポートとプロトコルは変数と型に対応していると考えられる.

3.2. 振舞い状態遷移表

コンポーネントが, その環境とどのように信号のやり取りを行うかを示したものが, コンポーネントの「振舞い状態遷移表」である. これは, 環境の変化(通常信号の入力を伴う)に対して, 環境の状態をどう変えていくかを規定したものであり, コンポーネントの機能そのものと考えることができる. 振舞い状態遷移表は, 環境の規定(ポートのプロトコル規定)に基づいて構築される.

振舞い状態遷移表は, 基本的には, コンポーネントの状態を行, 環境からの入力を列とした状態遷移表である. さらに, コンポーネントの状態に対して, そのときの各ポートの状態を明示しておく. これによって, コンポーネントの状態は, 環境の状態によって説明され, 「状態指向」の機能を持つことができる.

表2に振舞い状態遷移表の構成例を示す. 通常の状態遷移表に加えて, 状態には各ポートの状態を指定する. また, 入力信号は, ポート毎にまとめて記述される. 表2の入力信号は, 表1のポートのプロトコルの入力信号を書き写したものである. 初期状態は No=0 の状態(カード待ち)とする. 初期状態に対するポートの状態は, ポートの初期状態か, またはポートの初期状態から出力遷移のみで遷移できる状態が指定できる. 表2では, カードは, 表1(1)より, カードのアイドル状態(初期状態)からカード要求を出力して(つまり, 出力遷移で)カード挿入待ち状態に遷移した結果をコンポーネントの初期状態にしている. これは, コンポーネントの初期処理として, カードにカード要求を出すことが含ま

表2 状態指向の振舞い状態遷移表

状態					入力信号					
No	状態名	ポートの状態			カード			パスワード	認証	
		カード	パスワード	認証	カード挿入	キャンセル	カード取出	パスワード入力	パスワードエラー	アカウントエラー
0	カード待ち	カード挿入待ち	アイドル	アイドル	1					
1	パスワード待ち	カード挿入中	パスワード入力中	アイドル		3		2		
2	認証中	カード挿入中	アイドル	認証中		3			1	3
3	カード取出待ち	カード取出待ち	アイドル	アイドル			0			

れることを示す．コンポーネントの初期状態では，カードはカード挿入待ち，パスワードと認証はアイドルであるため，表1のプロトコル規定より，可能な入力はカード挿入のみであることがわかる．このように，コンポーネントの状態に対するポートの状態を記述することで，その状態で発生しうる入力が決められる．表2の遷移表で，灰色のセルはポート状態と表 1 のプロトコルから求められた可能な遷移の項目を示す．設計者はこの項目についてのみ遷移を考えればよく，これによって，遷移の漏れのチェックの自動化が可能になる[8]．

表 2 で，カード待ちでカード挿入を入力するとパスワード待ちに遷移する．パスワード待ちでは，カードはカード挿入中，パスワードはパスワード待ち，認証はアイドルになっている．カード待ちからパスワード待ちへの遷移での各ポートの状態遷移は，カードはカード挿入待ちからカード挿入信号を入力しカード挿入中に，パスワードはアイドルからパスワード要求を出力しパスワード入力中に，認証はアイドルのままになる．このように，コンポーネントの状態遷移による各ポートの状態遷移は，そのプロトコル規定に従わなければならない．

4. 拡張機能

4.1. アクション規定

3．2の振舞い状態遷移表にはアクションについての記述が省略されていた．通常の状態遷移表では，各遷移に付随してアクションの指定がなされるが，我々の状態遷移表では，各状態にポートの状態を記述することによって，遷移に付随するアクションは，各ポートの状態変化を起すのに必要な出力としてプロトコル規定で暗黙に示されているので，省略可能としている．例えば，表 2 のカード待ちからパスワード待ちへの遷移で必要なアクションは，パスワード要求の出力であることが，表 1 のプロトコル規定から求めることができる．

しかし，この方式では複数のポートに対する出力が必要な場合の出力の順序までは求められない．さらにポート出力以外のアクションが必要なこともある．そこで，アクションを明示的に表現することができるように，振舞い状態遷移表にアクション指定を付加し(表3)，アクション番号に対応するアクション表(表 4)を記述する機能

表3．アクション指定付の状態指向状態遷移表

状態					入力信号					
No	状態名	ポートの状態			カード			パスワード	認証	
		カード	パスワード	認証	カード挿入	キャンセル	カード取出	パスワード入力	パスワードエラー	アカウントエラー
0	カード待ち	カード挿入待ち	アイドル	アイドル	A1					
					1					
1	パスワード待ち	カード挿入中	パスワード待ち	アイドル		A3		A2		
						3		2		
2	認証中	カード挿入中	アイドル	認証中		A4			A5	A6
						3			1	3
3	カード取出待ち	カード取出待ち	アイドル	アイドル			A7			
							0			

表4．アクション表

アクション番号	概要	カード		パスワード		認証		その他	
		アクション	順	アクション	順	アクション	順	アクション	順
A1	パスワードを要求			パスワード要求					
A2	認証を要求					認証要求			
A3	パスワードキャンセル	カード排出	2	中断	1				
A4	認証のキャンセル	カード排出	2			キャンセル	1		
A5	パスワードエラー処理			パスワード要求	2			パスワードエラーのメッセージ表示	1
A6	アカウントエラー処理	カード排出	2					アカウントエラーのメッセージ表示	1
A7	カード取出後の処理	カード要求							

をオプションとして追加できるようにする. 表4のアクション表で, アクション番号とそれに対応する各ポートのアクションの部分(灰色部分)は, ポートの状態の変化とそのプロトコル規定より自動生成することができる. これによって, 必要な処理の漏れを防ぐことができる.

4.2. 条件分岐

入力に対して条件分岐をさせる場合, 振舞い状態遷移表の入力信号部分に条件を書くことができる. 表5に条件分岐の例を示す. ここでは, 表2のパスワードエラーに対して, 条件として3回未満であればパスワード入力を繰返し, 3回以上なら終了処理(カード排出)に移る処理を規定している. なお, 条件分岐に対しても, 3. 2の状態における可能な入力の決定や, 4. 1のアクションの自動抽出については同様に行うことができる.

4.3. 状態の階層

UML の状態遷移図では, 状態は階層をもつ. 状態を階層化する場合, ある種の共通点を持ついくつかの状態をまとめて, それらを包含する上位の状態を作ることになるが, いかにして適切な共通点を見つけるかが理解しやすい階層化にとって重要になる. 3. 2の振舞

表5. 条件の記述

状態			入力			
				認証		
No	状態名	...	...	パスワードエラー		アカウントエラー
				3回未満	3回以上	
1	パスワード待ち					
2	認証中			1	3	3
3	カード取出待ち					

い状態遷移表では, 共通点としてポートの状態が等しいという性質を使うと, 適切な階層化を発見できることが多い. 例えば, 表2で, カードポートの状態がカード挿入中である状態(パスワード待ちと認証中)をまとめて, 上位階層の状態を作ると, キャンセルの処理は上位階層の状態からの遷移としてまとめることができる. 表6に状態の階層をもつ状態遷移表を示す. 表で, 上位の状態でポート状態に記述がある場合は, それに含まれる全ての状態で同じポート状態をもつことを示す.

なお, この図の場合は, No.1の状態に含まれる下位の状態は2つしかないが, 支払, 預入, 振込等の処理を含む最終的な状態遷移表に対しては, その処理に

表6. 状態の階層表現

状態					入力信号					
					カード			パスワード	認証	
No	状態名	ポートの状態			カード挿入	キャンセル	カード取出	パスワード入力	パスワードエラー	アカウントエラー
		カード	パスワード	認証						
0	カード待ち	カード挿入待ち	アイドル	アイドル	1					
1	カード挿入中	カード挿入中				2				
1-0	パスワード待ち		パスワード待ち	アイドル				1-1		
1-1	認証中		アイドル	認証中					1-0	2
2	カード取出待ち	カード取出待ち	アイドル	アイドル			0			

表7. 状態の階層に対するアクション指定

No	状態	ポート状態		カード			
		カード	...	カード挿入	キャンセル	カード取出	...
0	カード待ち	カード挿入待ち		A1			
				1			
1	カード挿入中	カード挿入中			2		
1-0	パスワード待ち				A3		
1-1	認証中				A4		
2	カード取出待ち	カード取出待ち				A7	
						0	

必要な状態のほとんどがカード挿入中に含まれる.

階層をもつ状態遷移表では, 1 つの上位状態からの遷移に伴うアクションが, 下位状態によって異なることがある. 表 7 に階層をもつ状態遷移表でのアクション記述例を示す. このように, 階層を持つ状態遷移表では, アクションは下位の状態にそれぞれ記述する.

4.4. 実時間処理機能

実時間処理については, タイマーポートを設けることによって明示的に記述することができる. 簡単なタイマーのプロトコル例を表 8 に示す. 但し, ここでは 1 つのコンポーネント内でのタイマー記述であるため, 複数の平行処理コンポーネント間のタイミング処理については今後の問題となる.

表 8 タイマーのプロトコル

No	状態名	入力	出力	
		タイムアウト	開始	中断
0	アイドル		1	
1	カウント中	0		0

5. 電話交換機の例

簡単な電話交換機の例[9]を使って, 状態指向の状態遷移表の設計例を示す. この電話交換機の環境としては, 発呼電話, 着呼電話, 回線, 課金装置からなるものとする. このうち, 入力信号としては, 発呼電話の開始(オフフック), 終了(オンフック), ダイアル, 着呼電話の開始, 終了がある. 表 9 に発呼電話, 着呼電話, 回線, 課金装置のポートのプロトコルを示す. 表 10 に振舞い状態遷移表を示す.

表 9. 電話交換機のポートのプロトコル

(1) 発呼電話のプロトコル

No	状態名	入力			出力			
		開始	終了	ダイアル	ダイアル完了	ビジー	相手開始	相手終了
0	アイドル	1						
1	ダイアル中		0	1	2	4		
2	呼出		0				3	
3	通話		0					4
4	ビジー		0					

(2) 着呼電話のプロトコル

No	状態名	入力		出力	
		開始	終了	着信	相手終了
0	アイドル			1	
1	着信中	2			0
2	通話		0		3
3	終了待ち		0		

(3) 回線のプロトコル

No	状態名	出力			
		予約	取消	接続	切断
0	オフ	1			
1	予約済		0	3	
2	接続中				0

(4) 課金装置のプロトコル

No	状態名	出力	
		オン	オフ
0	オフ	1	
1	課金中		0

表 10. 電話交換機の振舞い状態遷移表

状態						入力						
No	状態名	ポートの状態				発呼電話					着呼電話	
		発呼電話	着呼電話	回線	課金装置	開始	終了	ダイアル			開始	終了
								未完	OK	ビジー		
0	アイドル	アイドル	アイドル	オフ	オフ	1						
1	ダイアル中	ダイアル中	アイドル	オフ	オフ		0	1	2	5		
2	呼出中	呼出	着信中	予約済	オフ		0				3	
3	通話中	通話	通話	接続中	課金中		4					5
4	発呼側終了	アイドル	終了待ち	オフ	オフ	3						0
5	着呼側終了	ビジー	アイドル	オフ	オフ		0					

6. 関連研究と議論

(1) モデル検査

モデル検査[4]では，Kripke 構造は設計結果の解析のために利用するのに対して，我々の方式では，設計記述として直接 Kripke 構造を利用する．つまり，Kripke 構造を，分析者や設計者が考えていることを直接表現するために利用している．さらに，Kripke構造の原子命題として，環境に含まれる装置の状態を使うことで，装置の状態変化の規定(プロトコル)が Kripke 構造の状態変化の制約となり，これをモデル検査時でなく分析・設計時に利用している．

(2) SDL の Pictorial Elements

初期の SDL[10]の Pictorial Elements(図 3)の概念は，状態指向の SDL を定義していたが，その扱いがコメント的であったため，現在の SDL[2]では除かれている．我々の状態指向の状態遷移表は，初期の SDL の状態指向の概念を形式化し，状態遷移表に表したものであると考えることができる．ただ，SDL の Pictorial Elements は，設計に絵を入れるために，簡単にツールなしでは書くのは難しい．状態指向の状態遷移表は，エクセルのようなツールで手軽に書くことができ，より実用的であるといえる．

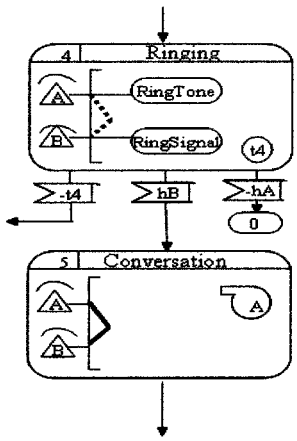


図 3. 初期の SDL の Pictorial Elements

(3) OCL 制約言語記述

状態指向と関連して，Whittle 等[6]は状態ベクトルとそれに対する OCL(Object 制約言語)記述で，状態を規定する方法を提案している．ただ，その場合，状態ベクトル(変数の集合)の選択の基準はなく，その OCL 記述も十分に理解しやすいとはいえない．我々の方式は，この状態ベクトルとして環境に含まれる装置(コンテキスト)の状態を使うことを示唆している．これによって，設計コンポーネントの環境の理解が進み，システムの見通しがよくなるといえる．

(4) ポート間の関連

ここでは，ポート毎のプロトコルは規定したが，ポート間の制約については何も書いていない．しかし，実際には，分析段階で分かるポート間関連もあり，それを明記することにより，より制約された設計が可能になることも多い．例えば，洗濯機制御で，水位センターのポートを考える．水位センサー単独のプロトコルは，入力として満水イベントと空水イベントがあるとする．この場合，例えば給水栓ポートと排水栓ポートがあって，給水栓が開，排水栓が閉の状態が続くとそのうち満水イベントが発生し，満水後，給水栓が閉，排水栓が開の状態が続くとそのうち空水イベントが発生する．これは，給水栓，排水栓，水位センサーのポートが外部で洗濯槽の水という物理実体を通して関連しあっているためである．この場合，外部の物理実体をモデル化[11]し，その振舞い状態遷移をこれらのポートのプロトコルを利用して定義すれば，3つのポートのプロトコルと物理実体の状態遷移を合成(状態マシン合成)[7]することにより，ポート間の関連を示す新たなプロトコルが得られる(図 4)．

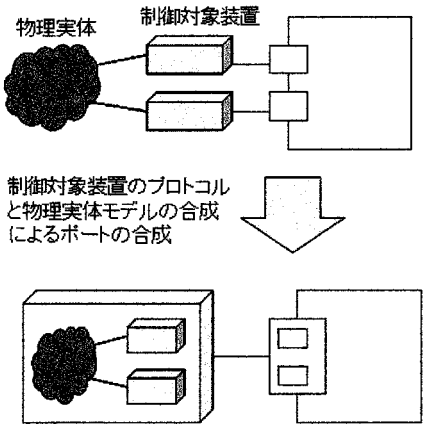


図 4. 物理実体モデルによるプロトコルの合成

7. おわりに

ここでは，状態遷移表の状態に環境(コンテキスト)の状態を記述することにより，その状態で発生しうるイベントや，遷移に伴う必要なアクションを，環境の仕様(プロトコル)から自動的に求めることができる方式を述べた．状態に，環境の状態を指定する方式は，モデル検査の基盤としてよく知られている Kripke 構造を応用したものと捉えることができる．つまり，Kripke 構造の原子命題として，環境の要素の状態(ポート状態)を適用していることになる．これによって，UML の状態マシンのように遷移に伴って「何をするか(to do)(手続き的)」を指定するのではなく，各状態が「どうなっているか(to be)(宣言的)」を指定する「状態指向」の状態遷移表を提案した．

我々は，以前いくつかの簡単な例題(洗濯機，自動販売機，ATM，簡単な電話交換機等)について，学生による設計実験を行い，ここで提案した状態指向の方式が，「理解しやすい，誤りのない」設計に効果的であることを確認することができた[9]．但し，そこでは状態遷移表ではなく状態遷移図を用いた．ここで述べた状態遷移表についても，同様の効果があると考えている．今後は，より複雑な実際的なシステムの設計に本方式を適用して，評価することを検討している．

ここでのべた状態遷移表は，まだ基本的な機能しか含んでおらず，今後は UML の状態マシンの履歴状態(H, H*)や SDL の入力信号セーブ等も表現できるように拡張していくつもりである．さらに，状態遷移表作成エディタや，状態遷移表のシミュレータ・プログラム生成等のツールの開発も進めていく所存である．

参考文献

[1] James Rumbaugh 他著，石塚圭樹訳：UML リファレンスマニュアル，ピアソンエデュケーション，2002.

[2] Ferenc Belina 他著，若原恭訳：仕様記述言語 SDL，カットシステム， 1996.

[3] 山岡貴哲他：状態遷移表に基づいたテストシナリオ生成の実用的手法とその適用，組込みソフトウェアシンポジウム 2004.

[4] Edmund M. Clarke,Jr., Orna Grumberg and Doron A. Peled : Model Checking, The MIT Press, 1999.

[5] 紫合治：「状態指向のステートチャート」，ソフトウェア工学の基礎ワークショップ(FOSE05).

[6] Jon Whittle, Johann Schumann : Generating Statechart Designs From Scenarios, 22nd International Conference on Software Engineering, 2000.

[7] Luca de Alfaro, Thomas A. Henzinger : Interface Automata, ACM SIGSOFT FSE01, 2001.

[8] 紫合治：「組込みソフトウェアにおける非正常処理の抽出」，組込みソフトウェアシンポジウム 2005.

[9] 大川敦，加藤大輝，紫合治：「インタフェース情報を用いた状態遷移図の生成」，情報処理学会 ソフトウェア工学 研究報告 No.151

[10] CCITT: Recommendations Z.101 to Z.104, "Functional Specification and Description Language(SDL)," Yellow Book, Volume VI=Fascicle VI.7, Geneve 1981.

[11] 紫合治：「組込みソフト設計のための物理実体(間接オブジェクト)の分析，情報処理学会 SIGSE ウィンターワークショップ 2006