

プロトコル状態マシンを用いた振舞い状態マシンの設計

紫合 治

東京電機大学 情報環境学部

組込みシステムの振舞いは、外部とのインタフェースから入ってくる信号に対して、外部にどの様に反応するか（どの様な信号を出すか）の規定によって定められる。このため、インタフェース仕様がシステムの設計にとって重要になる。本論文では、コンポーネントのインタフェースとなるポート(信号の出入口)に対して、信号の出入の仕方をプロトコル状態マシンで規定することにより、その規定とコンパティブルな振舞いをもつ状態マシンを系統的に構築する手法と、それに基づく状態マシン設計支援システムについて述べる。

Constructing Behavioral State Machine Using Protocol State Machines

Osamu Shigo

School of Information Environment, Tokyo Denki University

A behavior of an embedded system is designed by a state machine, which receives input events from externals through the interface, and sends output actions to external through the interface. Thus the interface specification becomes major concerns in the system design. This paper describes a design method that defines the interface of a component by protocol state machines, and then systematically constructs a behavioral state machine of the component using the interface protocol state machines. The design support system developed to realize the method, which provides enough guidance for designing behavioral state machine by using protocol specification, is also described.

1. はじめに

近年ユビキタスシステムの発展に伴い、組込みソフトウェア開発の需要が増大している。組込みソフトウェアの設計を支援するため、UML2.0[1]に基づく設計支援ツールが数多く市販されている[2][3]。UML はソフトウェアのいろいろな側面を表現するための多くの図式をもち、それぞれの図式は理解できても、異なる図式間の関連についての意味づけはあいまいなことが多い。例えば UML では、プロトコル状態マシンと振舞い状態マシンという2つの状態マシンを含むが、その関連は厳密には定義されていない。一般に、振舞い状態マシンはコンポーネントの動作・振舞いを定めたもので、従来の UML のステートチャートと同様に、それを記述することによって UML ツールで動作シミュレーションができたり、ソースプログラムを自動生成したりできる。しかし、プロトコル状態マシンには通常これらの支援はなく、単なるコメント記述扱いでしかないことが多い。

UML2.0 のベースの一つである ROOM[4][5]では、プロトコル状態マシンはポートを通る入出力信号の妥当な順序を規定するものととらえる。ここで、ポートは物理的な実体ではなく、関連する信号の入出力の論理的な窓口である。コンポーネントは、ポートのプロトコルの規定に従って、ポートから信号を入力し、ポートへ信号を出力しなければならない。このプロトコルに従っている限り、ポートの先にどんなコンポーネントを接続してもよい。つまり、プロトコル規定

により、コンポーネントは関連する他のコンポーネントとは完全に独立に設計できる。

コンポーネントの振舞いがポートのプロトコル規定を満たすためには、ポートがある信号を受信できるときはいつも、コンポーネントはその信号を受信できなければならないし、コンポーネントはポートにその時点で送信できる信号のみ送信できる。このように、コンポーネントの振舞いがポートのプロトコル規定を満たしているとき、コンポーネントはそのポートのプロトコルとコンパティブルであるという。

コンポーネントの全てのポートのプロトコルに対して、コンポーネントの振舞いはコンパティブルでなければならない。ここでは、ポートのプロトコル状態マシンの規定を使いながら、それにコンパティブルなコンポーネントの振舞い状態マシンを構築する方式について述べる。ここでは、ポートのプロトコルはコンポーネントの振舞いを設計する前に定められるものとするが、これは、コンポーネントの内部設計の前にそのインタフェースが決められるということであり、妥当な前提と考える。

以下に、2 節でコンポーネント、ポート、プロトコル状態マシン、振舞い状態マシンなどの関連をまとめた基本モデルを、3 節でコンポーネントとポートのコンパティビリティの定義と、コンパティブルな振舞い状態マシンを構築する手法について述べる。4 節で、この方式に沿って開発した設計支援システムを説明する。5 節で関連研究について述べ、最後に今後の問題とまとめを述べる。

2. 基本モデル

ここでは、ROOM[4]モデルにプロトコル状態マシンを追加したモデルを考える。まず、ROOM 同様、システムの構造はコンポーネント階層で表現される。コンポーネント間はポートを通した信号のやり取りによって通信する。各ポートは、そこを通る信号に対する規定としてプロトコル状態マシンをもつ。コンポーネントは、その動的な振舞いを示すために振舞い状態マシンをもつ。

2.1. システム構造

システム構造は、ROOM のアクター階層と同様のコンポーネント階層で表す。システム全体は、1つのコンポーネントクラスで表せる。コンポーネントクラスは、その外部との信号のやり取りの窓口として、いくつかのポートをもつ。ポートは、概念的にはコンポーネントと外部との境界上に置かれる。コンポーネントクラスは、その内部にいくつかのサブコンポーネント(コンポーネントクラスのインスタンス)をもつことができ、コンポーネント階層を作る。サブコンポーネントのポートは、兄弟サブコンポーネントのポートか、親コンポーネントのポートとコネクタで接続できる。1つのポートは、1つのポートとしか接続できないものとする。あるポートを複数のポートと接続したい場合は、直接接続しないで、ポートの接続のみを行うサブコンポーネントを介して接続する。4節の図1にシステム構造の例を示す。

2.2. プロトコル状態マシン

各ポートは、その型としてプロトコルをもつ。プロトコルはポートを通る入出力信号の妥当な出現順を定めるプロトコル状態マシンによって規定される。

【定義1】プロトコル状態マシンは以下の4つ組

$P = (S_p, s_{p0}, M_p, \delta_p)$ である。ここで、

S_p : 状態の有限集合

$s_{p0} \in S_p$: 初期状態

M_p : 信号の有限集合

$\delta_p \subseteq S_p \times IO \times M_p \times S_p$: 遷移規則の集合。

ここで、 $IO = \{?, !\}$ で、 $?$ は入力を、 $!$ は出力を示す。

つまり、 $(s, ?, m, s') \in \delta_p$ なら、状態が s のとき、信号 m が入力される可能性があり、入力されれば状態が s' になることを、 $(s, !, m, s') \in \delta_p$ なら、状態が s のとき、信号 m を出力でき、出力すれば状態が s' になる。 ■

プロトコル状態マシン P は決定的とする。つまり、もし $(s, d, m, s_1) \in \delta_p$ か $(s, d, m, s_2) \in \delta_p$ なら、 $s_1 = s_2$ でなければならない。これにより、ポートを入出力

する信号の列からポートの状態が一意に定まる。

$s \in S_p$ につき $I_p(s) = \{m \in M_p \mid (s, ?, m, s') \in \delta_p\}$ (状態 s で p からの入力待ちをすべき信号の集合)、 $O_p(s) = \{m \in M_p \mid (s, !, m, s') \in \delta_p\}$ (状態 s で p へ出力可能な信号の集合)とする。コンポーネント C がプロトコル P のポート p を持ち、その状態が s のとき、 C は $I_p(s)$ の全ての信号を p から受信する用意をしなければならず、また C は $O_p(s)$ に含まれる信号のみを p に送信することができる。

プロトコル状態マシンはポートの振舞いを規定しているわけではなく、ポートに信号を入出力するときの使い方を規定している。例えば、振舞いを規定している状態マシンでは、ある状態から入力遷移と出力遷移の両方がある場合は許さない(意味が不明)が、プロトコル状態マシンでは、その状態で入力されるべき信号と出力できる信号の集合を規定するので、入力遷移と出力遷移が同時にあってもよい。このような、信号入出力の使い方の規定として、Interface Automata[6]があり、プロトコル状態マシンはInterface Automaton ととらえることができる。また、ここでのプロトコル状態マシンは、UML2.0 のプロトコル状態マシンの定義とは異なる。例えば、UML2.0 のプロトコル状態マシンは入力信号に対してのみ規定し、さらに遷移に事前/事後条件を記述できる。ここでのプロトコル状態マシンは、ROOM での規定に近く、出力信号についても規定し、事前/事後条件は含まない。

プロトコルは、そのポートに接続することのできる全ての外部コンポーネントを抽象化したものととらえることができる。この抽象化によって、各コンポーネントは外部のコンポーネントを知ることなく、その機能を理解することができる。

2.3. 振舞い状態マシン

コンポーネントの振舞いは、振舞い状態マシンによって規定する。コンポーネント C がもつポートの集合を P 、各ポート $p \in P$ のプロトコル状態マシンを $T(p)$ とする。このとき、 C の振舞い状態マシン B は以下のように定義する。

【定義2】ポート集合 P をもつコンポーネント C の振舞い状態マシンは4つ組 $B = (S_b, s_{b0}, M_b, \delta_b)$ である。

S_b : 状態の有限集合

$s_{b0} \in S_b$: 初期状態

$M_b \subseteq P \times IO \times M_{ps}$: ポートの信号入出力集合

ここで、 $M_{ps} = \bigcup_{p \in P} M_{T(p)}$: P のすべてのポートの信号の集合。ポート p から信号 m の入力 $(p, ?, m) \in M_b$ を $p?m$ 、ポート p への信

号 m の出力 $(p, !, m) \in M_b$ を $p!m$ と書く。

$\delta_b \subseteq S_b \times I_b \cup \{\varepsilon\} \times O_b^* \times S_b$: 遷移規則

ここで, $I_b = \{(p, d, m) \in M_b \mid d = ?\}$ (信号入力),

$O_b = \{(p, d, m) \in M_b \mid d = !\}$ (信号出力),

O_b^* は O_b の 0 以上の列の集合。 ■

振舞い状態マシンは決定的とする。つまり, $(s, i, m_1, s_1) \in \delta_b$ かつ $(s, i, m_2, s_2) \in \delta_b$ なら, $m_1 = m_2$ かつ $s_1 = s_2$ でなければならない(遷移にガード条件を追加することによって, 異なる遷移が同じ信号入力を持つように拡張することも可能であるが, ここでは簡単のため決定的とした)。また, 初期遷移のみ ε 遷移(信号入力がない遷移)を許すが, その場合, 初期遷移は ε 遷移のみとする。 ε 遷移をもつ初期状態以外では, 全ての状態でポートからの信号の入力を待ち, 信号入力のみが遷移を起す。このような状態を安定状態と呼ぶ(SDL[7]の状態と同様)。

ここでの振舞い状態マシンは, UML2.0 の振舞い状態マシンを制限したものと考えることができる。UML の振舞い状態マシンと比べて, 状態内でのアクション記述やガード条件がなく, さらにアクションとして, ポートへの信号出力の列しか許さない。従って, ここでの振舞い状態マシンは, あくまでもコンポーネントの振舞いの概要設計を記述したものであり, 信号の分析や変換処理, 計算処理などは含まない。しかし組込みソフトウェアは, 外部からのイベントに対し, 外部にどの様に反応するかを中心に規定するものであり, アクションとして信号出力しか許さなくても十分なことも多い。少なくとも, 概要設計の段階では, 外部との信号入出力の規定に限った状態マシンが重要になる。なお, UML の状態マシンと違い, ここでは状態の階層は扱わない。これはあくまでも簡単化のためであり, 我々のモデルに状態の階層を入れること自体は難しくはないと考える。

3. コンパティビリティ

コンポーネントの振舞いは, その全てのポートのprotocols規定と矛盾しないことが必要である。ここでは, コンポーネント C の振舞い状態マシン B が, ポート p のprotocolと矛盾しないとき, B は p と「コンパティブル」とであると呼ぶ。コンパティブルの定義は, ROOM ではポートのprotocol間の規定(一方のポートの出力信号は他方のポートの入力信号に含まれる, つまり送信した信号は必ず受信されるという規定[4])になっているが, ここではこれを, コンポーネントの振舞いとポートのprotocolの間の性質として定義する。

3.1. ポートとコンポーネントの振舞いのコンパティビリティ

コンポーネント C はポートの集合 P をもち, C の振舞

い状態マシンを $B = (S_b, s_{b0}, M_b, \delta_b)$ とする。ポート $p \in P$ のprotocolを $T(p) = (S_p, s_{p0}, M_p, \delta_p)$ とする。このとき, B が p とコンパティブルである条件を述べる。その前にいくつかの準備をする。

B の状態 $s \in S_b$ に対し, s で待つ信号入力の集合を $I_b(s) = \{p?m \in I_b \mid (s, p?m, act, s') \in \delta_b\}$ と定義する。ここで, $act \in O_b^*$ である。

B の信号入出力 $(p, d, m) \in M_b$ とポート $q \in P$ に対し, $p = q$ なら $(p, d, m)/q = (d, m)$, $p \neq q$ なら $(p, d, m)/q = \varepsilon$ (空列)とし, この $/q$ を M_b^* の要素(信号入出力の列)に対して定義する。つまり, $m_{bs} \in M_b^*$ に対し, m_{bs}/p は m_{bs} からポート p の信号入出力だけを抜き出した列になる。

振舞い状態 $s_b \in S_b$ とポート $p \in P$ に対して, $s_b(p) \subseteq S_p$ を s_b 状態に対応する p の状態(の集合)とし, 次のように定義する。

1. $s_{b0}(p) = \{s_{p0}\}$
2. $(s_b, i, act, s_b') \in \delta_b$ かつ $s_p \in s_b(p)$ かつ $(s_p, i, act/p, s_p') \in \delta_p^*$ なら $s_p' \in s_b'(p)$ 。
但し $\delta_p^* \subseteq S_p \times (IO \times M_p)^* \times S_p$ は δ_p の推移閉包。

上の定義より, $(s_b, i, act, s_b') \in \delta_b$ で $s_p \in s_b(p)$ のとき, もし $i, act/p = \varepsilon$, つまり振舞いの遷移に p の入出力信号が現れない場合, $s_p \in s_b'(p)$ となる ($(s_p, \varepsilon, s_p) \in \delta_p^*$ だから)。

【定義3】コンポーネント C の振舞い状態マシン $B = (S_b, s_{b0}, M_b, \delta_b)$ が, そのポート p のprotocol状態マシン $P = (S_p, s_{p0}, M_p, \delta_p)$ とコンパティブルであるとは, 次の2つの条件を満たす場合を言う。

- ①入力条件: 任意の $s_b \in S_b$ に対して, $s_p \in s_b(p)$ で $m \in I_p(s_p)$ なら, $p?m \in I_b(s_b)$ 。

ポートが状態 s_p で信号 m を受信する可能性があるとき(つまり $m \in I_p(s_p)$), s_p に対応するポート p の状態としてもつコンポーネントの状態 s_b (つまり $s_p \in s_b(p)$)は, p から m を受信した場合の遷移を持たねばならない(つまり $p?m \in I_b(s_b)$)。

②出力条件: 任意の遷移 $(s_b, i, act, s_b') \in \delta_b$ に対し,

$$s_p, s_p' \in S_p \text{ が存在し } (s_p, i, act / p, s_p') \in \delta_p^*.$$

もしコンポーネントが状態 s_b のとき入力 i を受けて act を出力し s_b' 状態に遷移するなら, i, act のポート p の信号入出力 $i, act / p$ は, s_b に対応するポート p の状態 s_p から s_b' に対応するポート p の状態 s_p' に至る遷移系列で受理されなければならない. ■

全てのポートについて振舞い状態マシンがコンパティブルになれば, コンポーネントの任意の到達可能な状態 s_b において, $s_b(p)$ は空でない. すべての $p \in P$ につき $s_b(p)$ が 1 個の要素しか持たない場合, 状態 s_b は厳密であるという. すべての状態が厳密であるとき, 振舞い状態マシンは厳密であるという. 通常, ポートの状態が変われば, それに対する処理の仕方も変ることになるので, コンポーネントの状態も変ることが多く, 振舞い状態マシンが厳密である(つまり, ポートの状態が異なればコンポーネントの状態も異なる)という制約は妥当であると考え. 以下では厳密な振舞い状態マシンのみを扱う.

3.2. ポートとコンパティブルな振舞い状態マシンの構築

コンポーネントの全てのポートに対して, プロトコル状態マシンが与えられた場合, それをもとに, それらとコンパティブルな振舞い状態マシンを系統的に構築することができる. コンポーネント C のポートの集合を P , $p \in P$ のプロトコル状態マシンを $T(p) = (S_p, s_{p0}, M_p, \delta_p)$ として, C の振舞い状態マシン $B = (S_b, s_{b0}, M_b, \delta_b)$ は次のような手順で構築できる.

1. $M_b = \{(p, d, m) \mid p \in P \text{ で } (d, m) \text{ は } \delta_p \text{ の遷移に現れる信号入出力}\}$ として, M_b を作る.
2. $S_b = \{s_{b0}\}$ とする. ここで, s_{b0} は初期状態. また, 全ての $p \in P$ につき $s_{b0}(p) = s_{p0}$ とする (B は厳密なので, $s_{b0}(p)$ の要素数は 1 になる).
3. 必要なら, ε 遷移, $(s_{b0}, \varepsilon, act_0, s_{b0}')$ を初期遷移に加える. ここで, act_0 や s_{b0}' の追加の仕方はステップ 5 と同様である. その結果として, s_{b0}' を S_b に追加する.
4. S_b のなかの各状態 s_b (ε 遷移がある場合は初期状態はのぞく) につき, 可能な信号受信の集合 $I_b(s_b) = \{p?m \in M_b \mid p \in P, m \in I_p(s_b(p))\}$ (s_b に対応するポートの状態 $s_b(p)$ でのポートからの

信号入力(の集合)を作る.

5. $s_b \in S_b$ と $p?m \in I_b(s_b)$ につき, 以下の方法で act と s_i を求め $(s_b, p?m, act, s_i) \in \delta_b$ を作る.
 - (a) $act = \varepsilon$, $s_i(q) = s_b(q) (q \in P, q \neq p)$ とする. また, $p?m$ の受信による p の状態の変化を s_p' (つまり $(s_b(p), ?, m, s_p') \in \delta_p$) とすると, $s_i(p) = s_p'$ とする. これによって, s_i は s_b から $p?m$ を入力した直後の状態になる.
 - (b) アクションとして利用できる信号出力の候補集合 $ACTS = \{q!m \mid q \in P, m \in O_q(s_i(q))\}$ (s_i の状態で各ポートに送信できる信号出力の集合) を求める.
 - (c) $q!m \in ACTS$ を必要に応じて選び, それを act に追加し, $(s_i(q), !, m, s_q') \in \delta_q$ として $s_i(q) = s_q'$ とする.
 - (d) 上の (b) と (c) を繰返し, アクションとして必要な信号出力の列を作る.
 - (e) もし S_b の中に s_i と同じポート状態をもつもの, つまり, 全ての $p \in P$ につき $s(p) = s_i(p)$ となる $s \in S_b$ が存在するときは, $(s_b, p?m, act, s)$ を δ_b に追加, そうでないときは $(s_b, p?m, act, s_i)$ を δ_b に追加し, さらに s_i を S_b に追加する.
6. 全ての状態 $s_b \in S_b$ の全ての信号入力 $I_b(s_b)$ が遷移で使われるようになるまで, 上の 4 から 5 を繰返す.

上のステップで, 設計者は信号出力を候補集合の中から選択することを繰り返すだけで振舞い状態マシンが構築でき, その他は全て自動化できることになる.

4. 状態マシン設計支援システム

3. 2 で述べたように, プロトコル状態マシンを使って, それらとコンパティブルなコンポーネントの振舞い状態マシンを系統的に構築することができる. 我々はこの方式の設計を支援するシステムを開発した. ここでは, システムの機能の説明のために, よく知られた ATM の例[8]を使う.

4.1. ATM の例とシステム構造

簡単のため, 文献[8]と同様に, お金の引き出し, 預金, 振込み等の ATM の主な機能は省いて, 例外的な利用(パスワード誤りやキャンセルなど)についてのみ取り扱う. ATM のシステム構成図の例を図 1 に示す. システムは, ATM 制御(atmController), カード読取り(cardRD), パスワード入力(passwdRD), カード認証(銀行)(bank), メッセージ

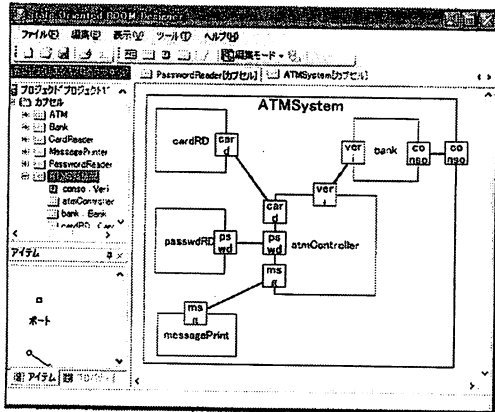


図1 ATMのシステム構成図

ジ表示(*messagePrint*)からなる。ATM 制御は、カード読取りをつなぐポート *card* (プロトコルは *Card*)、パスワード入力をつなぐポート *pswd* (プロトコルは *Pswd*)、カード認証をつなぐポート *veri* (プロトコルは *Veri*)、メッセージ表示をつなぐポート *msg* (プロトコルは *Msg*) をもつ。システム構成図は、図1の左下のアイテムリストからコンポーネント、ポート、コネクタを選択して、編集ウィンドに貼り付けながら作成していく。このとき、サブコンポーネントの登録で、そのコンポーネントクラスの名前を登録済みリストから選ぶことにより、そのサブコンポーネントのポートが自動的に現れる。

4.2. プロトコル状態マシン

プロトコル状態マシンの定義では、まずポートを通る入力信号と出力信号をリストアップし、それを使った状態マシンを描画していく。プロトコル状態マシンの編集ウィンド(図2)には、初期値として開始記号と初期状態が現れる。状態遷移を登録するには、左下のアイテムリストから状態(State)や遷移(Signal)を選んで編集画面に貼り付ける。このとき、状態に対しては状態名を入力し、遷移に対しては、入出力信号リストから遷移信号を選択する。

ATM システムのプロトコル状態マシンを図3に示す。カード読取りに対するプロトコル *Card* では、まずカード挿入を要請するために *displayMainScreen* 信号を送信でき、それによってカード挿入 *insertCard* 信号の受信を待つ。カード挿入後は、カード読取りからキャンセル信号 *cancel* を受信する可能性があり、またはキャンセルの前にカード排出信号 *ejectCard* を出すこともできる。もし、キャンセルなら、カード排出信号を出すしかできない。カード排出後は、カード取出しを要請する信号 *requestTakeCard* を出し、利用者がカードを取り出したとき、信号 *takeCard* を受信してもとの *idle* 状態に至る。*Card* 以外のプロトコル状態マシンは比較的簡単である。以上の信号名は、文献[8]で使われている信号名とほぼ同じである。ただし、文献[8]の信号に加えて、パスワード入力とカード認証のプロトコルで処理中に

Card からキャンセルが指示された場合、それを知らせる信号として *cancelPswd* と *cancelVerify* を追加した。これらの信号は、キャンセル時にパスワード入力やカード認証をアイドル状態に戻すために必要である。

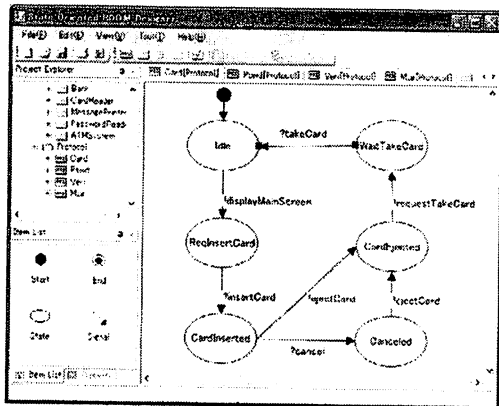


図2 プロトコル状態マシンの設計例

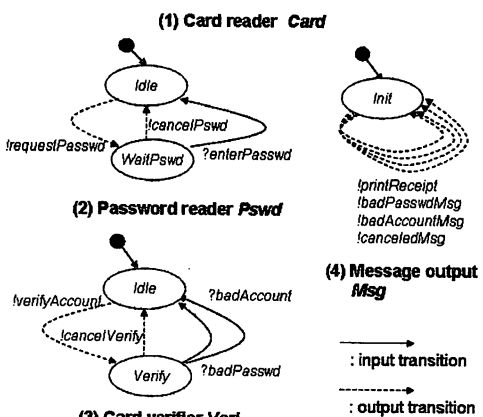
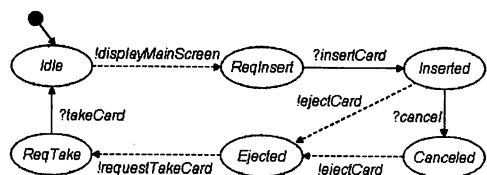


図3 ATMの各ポートのプロトコル状態マシン

4.3. 振舞い状態マシン

システム構成図とプロトコル状態マシン図を入力した後、それらを使って振舞い状態マシンを系統的に構築することができる。既存の状態マシン設計ツールでは、状態を生成し、状態から状態への遷移矢印を生成し、これらに遷移の

情報(起動イベント、ガード条件、アクション記述など)や状態の情報(入口アクション、出口アクション等)を記入していく方式が多い。我々のシステムでは、利用者が状態をクリックすると、システムが自動的にその状態から出る遷移の候補を出し、そこから遷移を選ぶと、システムはそこで利用できるアクションの候補をリストアップするので、その中からアクションを選んでいくことによって状態マシンの設計を進める。このように、システムによる自動ガイドに沿って状態マシンを構築することによって、得られる振舞い状態マシンは関連するプロトコル状態マシンとコンパティブルであることが保障される。

(1) 初期状態と ε 遷移

振舞い状態マシンの設計を開始すると、初めに開始マークと初期状態が自動的に現れる。 ε 遷移を登録するには、まず初期状態をクリックして、その状態で利用可能なアクションである信号出力の一覧を画面右のアクション候補リストを表示させる。アクション候補リストは、ポート毎に、初期状態で出力できる信号のリスト、つまり、ポート p については $O_p(s_{p0})$ からなる。ATM の例では、図 3 のプロトコル状態マシンより、 $card$ ポートは $\{ displayMainScreen \}$ 、 $pswd$ ポートは $\{ requestPassword \}$ 、 $veri$ ポートは $\{ verifyAccount \}$ 、 msg ポートは $\{ canceledMsg, badAccountMsg, badPasswdMsg, PrintReceipt \}$ が候補リストに入る。ここでは ε 遷移として、ATM 利用者にカード挿入を促すために $card$ ポートに $displayMainScreen$ 信号を出すことが必要である。そこで、アクション候補から、 $card$ の $displayMainScreen$ を選択すると、図 4 に示すような ε 初期遷移が生成される。但し、状態名はカード要求 ($ReqCard$) にしてある。これより、

$$(s_{b0}, \varepsilon, card!displayMainScreen, ReqCard) \in \delta_b$$

なる遷移が生成される。なお、図 4 で、状態には各ポートの対応状態が明示される。図 3 のプロトコル状態マシンより、 $ReqCard$ 状態では、各ポートの対応状態は、 $card$ は $ReqInsert$ 、 $pswd$ は $Idle$ 、 $veri$ は $Idle$ 、 msg は $Init$ になる。

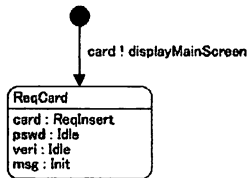


図 4 初期 ε 遷移

(2) 新たな状態への遷移

次に初期遷移以外の遷移の設計に進む。ある状態を選んで状態をクリックすると、その状態から始まる遷移の候補が入力信号付で自動的に生成される。つまり、クリックされた状態 s_b に対して、その状態の入力信号リスト $I_b(s_b)$

の要素 $p?m$ を入力信号とした遷移が生成される。例えば、 $ReqCard$ 状態の場合、 $I_b(ReqCard) = \{ card?insertCard \}$ (図 3 と図 4 から、 $ReqCard$ 状態からの入力は $card$ の $insertCard$ しかない) だから、図 5 のように $card?insertCard$ を入力信号とした遷移候補が生成される。

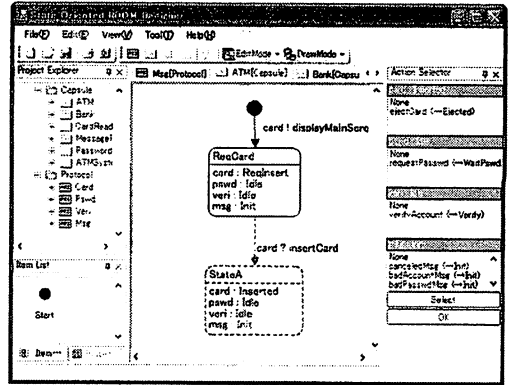


図 5 遷移候補とアクション候補の生成例

図 5 の新しい状態 ($StateA$) に対するポート状態は、 $ReqCard$ と比べて、 $card$ ポートが $Inserted$ になっているが、これは、図 3 より $card$ が $ReqInsert$ のときに $insertCard$ が入力されると $Inserted$ に状態遷移するからである。この状態遷移を完成するには、まず遷移候補の遷移先状態 ($StateA$) をクリックし、そのときのポート毎のアクション候補を画面右に表示させ、そこから必要なアクションを選択していき、最後に OK ボタンを押す。 $StateA$ 状態でのアクション候補は、 $card$ ポートは $ejectCard$ になり ($p=card$ のとき $Op(Inserted) = \{ ejectCard \}$ だから)、それ以外のポートについては以前と同じになる。ここでは、カード挿入後の利用者にパスワード入力を依頼するために、 $pswd$ ポートの $requestPasswd$ 信号をアクションに選び、 $StateA$ を $ReqPasswd$ (パスワード要求) という名前に変えて OK ボタンを押す。これによって、 $(ReqCard, card?insertCard, pswd!requestPasswd, ReqPasswd) \in \delta_b$ なる遷移ができる。 $ReqPasswd$ 状態に対するポート状態は、 $card$ ポートは $Inserted$ 、 $pswd$ ポートは $WaitPswd$ 、その他は以前の状態 ($ReqCard$) のままとなる。

$ReqPasswd$ 状態からの遷移を設計するために、 $ReqPasswd$ 状態をクリックすると、 $card$ ポートからの $cancel$ 信号と $pswd$ ポートからの $enterPasswd$ 信号の 2 つの入力に対する遷移候補が現れる ($I_b(ReqPasswd) = \{ card?cancel, pswd?enterPasswd \}$ だから)。パスワードの入力 ($enterPasswd$) に対しては、カード認証に移るために、 $veri$ ポートに $verifyAccount$ 信号を送るアクションを選択し $Verify$ 状態に移る遷移、 $(ReqPasswd, pswd?enterPasswd, veri!verifyAccount, Verify) \in \delta_b$ を作る。ここで、 $Verify$ 状態に対応するポートの状態は、 $card$ が $Inserted$ 、 $pswd$ が

Idle, veri が Verify, msg が Init になる。さらに, card ポートからの cancel 信号に対しては, カードを排出してカード取出し要求メッセージを出し, パスワード入力を中断する。これもこれまで同様, アクション候補からの選択を繰り返すことである。ここでは, アクションとして, *pswd!cancelPswd*, *msg!canceledMsg*, *card!ejectCard*, *card!requestTakeCard* を出す。結果の状態名を *ReqTakeCard* として, 新たな遷移ができる。なお, 上記のアクションで, card の *requestTakeCard* は *ejectCard* を出した後で候補として現れる。この結果の *ReqTakeCard* 状態に対応するポート状態は, card が *ReqTake*, pswd が *Idle*, veri が *Idle*, msg が *Init* となる。このようにして, 図 6 に示す遷移が設計できる。

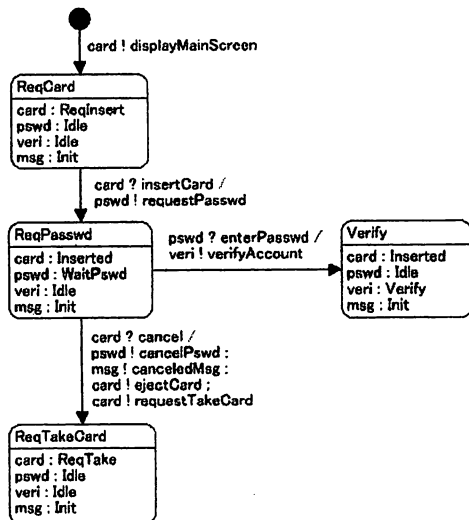


図 6 振舞い遷移の設計例

(3) 既存状態への遷移

ReqTakeCard 状態からの遷移候補は, card ポートからの *takeCard* 信号の入力による遷移に限られる ($Ib(ReqTakeCard) = \{ card?takeCard \}$ より)。そこで, この遷移のアクションとして, 元に戻ってカード挿入待ちにするために card の *displayMainScreen* を選ぶと, 結果の状態に対応するポート状態は, card が *ReqInsert*, pswd と veri が *Idle* で msg が *Init* となり, *ReqCard* と同じになる。そこで, この遷移は *ReqTakeCard* から既存の *ReqCard* への遷移と自動的に認識される。つまり, $(ReqTakeCard, card?takeCard, card!displayMainScreen, ReqCard) \in \delta b$ となる遷移が生成される。

このようにして, 全ての状態の全ての遷移候補に対して, アクションを選択しながら設定することによって, 最終的には図 7 に示すような振舞い状態マシンが構築できる。この状態マシンは, 図 3 に示すプロトコル状態マシンとコンパティブルであることが保障される。

図 7 の状態マシンは文献[8]の結果と比べて次の点で異なっている。

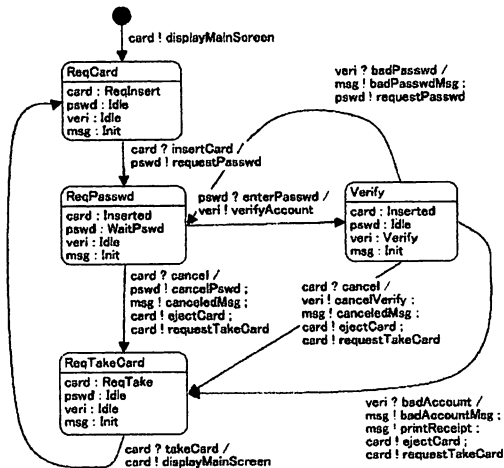


図 7 完成した ATM 振舞い状態マシン

- ・我々の状態マシンでは初期遷移以外はすべて入力信号をもつが, 文献[8]では入力信号を持たない遷移も許している。ただし, これらは相互に変換することも可能である。
- ・我々の状態マシンには, 文献[8]に加えて, キャンセルをパスワード入力やカード認証に知らせるための信号(*cancelPswd*と*cancelVerify*)をもつ。
- ・彼等の状態マシンは階層を使っているが, 我々の状態マシンは階層は使わない。

文献[8]の結果と比べて, 見かけは少し異なり, 特にプロトコルの扱いが全く違うが(文献[8]では状態マシンではなく OCL(オブジェクト制約言語)を使用), 双方の結果はほぼ同じ効果をもつ状態マシンになっている。

5. 関連研究

本論文での中心となるアイデアの一つは, コンポーネントの状態に対して, そのインタフェースとなる各ポートの状態を関連させることである。状態マシンの状態に関連外部装置の状態を対応させる方式として, 初期の SDL[9]の状態絵(pictorial elements)がある。SDL では交換ソフトの状態遷移図の状態の中に, 送受話器の off-hook や on-hook 状態, トーン発信, 着信ベル送出, タイマーの on, off などの絵を含むことにより, 各状態で制御対象装置がどうなっているかを分かりやすく表せる。しかし, SDL の状態絵はコメント扱いであり, 形式的な意味づけはなく, このため現在の SDL の版[7]では除かれている。我々のモデルは SDL

の状態絵を、ポートのプロトコル状態マシンの状態を表すものとして形式化したものととらえることもできる。

より形式的には、我々のモデルは、状態が命題集合をもつ Kripke 構造ととらえることができる。ここでは、各ポートの状態が Kripke 構造の命題に対応する。状態遷移に伴う命題の変化を、ポートにつながる装置の状態の変化ととらえ、その変化を起すためのイベントを遷移の入力やアクションに対応させたものが我々のモデルといえる。Kripke 構造をベースにしたモデル検査[10]では、出来上がった状態マシンに対して妥当性を確かめる方法であるが、ここでのアイデアは、ポートのプロトコル状態マシンとコンパティブルなことを妥当性ととらえ、はじめから妥当な状態マシンを構築する方法を提供することといえる。

文献[7]で Whittle 等はシナリオから状態マシンを生成する手法として、状態変数とそれに対する制約言語 (OCL) による記述を提案している。この場合、状態変数の選定や制約記述の妥当性が問題になるが、彼等の方式ではそれに対する考察は特にはなされていない。我々のモデルでは、彼等の状態変数と制約記述がポートの状態とプロトコル状態マシンに対応する。組込みソフトウェアの通常の設計者にとっては、ポートやプロトコル状態マシンの概念は比較的自然而あり、Whittle 等の状態変数や制約言語記述より理解しやすく記述しやすいと思う。

6. おわりに

システムの振舞い設計の方法として、ポートのプロトコル状態マシンをシステムのインタフェース仕様ととらえ、その仕様に基づいてコンパティブルな振舞い状態マシンを構築するための設計モデルと、そのモデルに基づく状態マシン設計支援システムについて述べた。

ここでは、簡単のため状態マシンの階層は入れなかったが、我々のモデルに状態の階層を入れることはすこしの拡張で可能になる。この場合、上位レベルの状態に対応するポートの状態は複数個を許すことになる。さらに、ポートの状態は、コンポーネントの状態階層を作る上で、適切なガイドを与える。例えば、図7で、*card*ポートが *Inserted* 状態 (カード挿入中) である振舞い状態、*ReqPasswd* と *Verify* をグループ化して、*Operation* という上位状態を作ることが考えられる。*Operation* から *card?cancel* 入力によって *ReqTakeCard* 状態に遷移する状態遷移を1つ記述することによって、内部のそれぞれの状態からの *cancel* による遷移を書かなくてもよい。

我々はモデルと支援システムの評価のために、簡単な組込みシステム (自販機、洗濯機、炊飯器、簡単な電話交

換機等) の振舞い設計に応用し、その有効性を確かめることができた[11]。特に、ポートのプロトコル仕様が分かっている場合は、本方式が有効であった。ここで構築した振舞い状態マシンには、一般的なアクション記述や変数記述等ではなく、詳細設計やコード生成にまでもっていくには概要すぎる。しかし、組込みシステムの場合の設計としては、ここで述べた詳細レベルで十分有効なことが多かった。

今後、より実際的な規模の組込みシステムの設計に応用して、ここでのモデルと設計支援システムの実用性、有効性、問題点等について評価する予定である。

参考文献

- [1] OMG : UML 2.0 Superstructure Specification, <http://www.omg.org/>, 2004.
- [2] IBM : Rational Rose, <http://www-306.ibm.com/software/awdtools/developer/rosexde/>
- [3] Microsoft : Office Visio, <http://www.microsoft.com/office/visio/prodinfo/default.mspx>
- [4] Brian Selic, et. al : Real-time Object-oriented Modeling, John Wiley and Sons, Inc. 1994.
- [5] Brian Selic, James Rumbaugh : Using UML for Modeling Complex Real-time Systems, White paper, IBM, 1998.
- [6] Luca de Alfaro, Thomas A. Henzinger : Interface Automata, *ACM SIGSOFT FSE01*, 2001.
- [7] SDL Forum Society : SDL , <http://www.sdlforum.org/SDL/>
- [8] Jon Whittle, Johann Schumann : Generating Statechart Designs From Scenarios, *22nd International Conference on Software Engineering*, 2000.
- [9] CCITT: Recommendations Z.101 to Z.104, "Functional Specification and Description Language (SDL)," Yellow Book, Volume VI = Fascicle VI.7, Geneve 1981.
- [10] Edmund M. Clarke, Jr., Orna Grumberg and Doron A. Peled : Model Checking, The MIT Press, 1999.
- [11] 大川敦, 加藤大輝, 紫合治 : インターフェースの情報を用いた状態遷移図の生成. 情処研報, SE-151 (2006).