組み込みシステムにおける仕様の自動生成

和　田　遥 †1　紫　合　治 †2

組み込みシステムの開発において，全ての例外を完全にテストすることはほとんど不可能である．そこで本稿では，プロブレムフレームの考え方を用い，要求とドメインプロパティを合成することで，出現しうる全てのイベントを考慮した状態マシンの仕様を生成する手法を提案する．ドメインプロパティは状態マシンとして，要求は今回提案する ”安全状態 ”という概念として定義する．これにより，システムができる最大限に実行可能なパスを自動生成できると考えられる．

Deriving the State Machine Specification for Embedded System

Haruka Wada †1 and Osamu Shigo †2

For embedded software system, it is almost impossible to completely test all exceptions. This paper provides a method to generate the state machine specification with all possible events from the requirement and the domain properties of problem frames. In the method, state machines are provided as the domain properties and domain state constraints with a concept of ”Safe” states are provided as the requirement. Then resultant specification of system state machine with maximum possible passes is automatically generated.

†1 東京電機大学院 情報環境学研究科
Graduate School of Information Environment, Tokyo Denki University
†2 東京電機大学 情報環境学部
School of Information Environment, Tokyo Denki University

1. はじめに

組み込みシステムの開発では，機器や装置は想定外の色々な振る舞いを起こすため，これらを全て考慮すると非正常系の処理が全体の約８割[1] を占め，分析や設計の中心的な課題となる．そのため，システムの分析・設計時に外界の機器や装置など影響を受ける全ての事象につき漏れなく把握しておくことが重要となる．そこで，問題が存在する外界の環境そのものをまず分析する手法として，環境をドメインプロパティという概念で表し考えるプロブレムフレーム[2] に着目した．プロブレムフレームではシステムを，開発の対象である「マシン」と，それを取り巻く外部環境である制御対象の「ドメイン」，ドメインに対する振る舞いの「要求」との３つを分けて考える．プロブレムフレームを用い，要求から仕様を生成する手法はいくつか提案されている[3][4][5]．そこでは，要求やドメインプロパティは述語論理や CSP[6] などにより記述されるが，現場の技術者にとってこれらの形式的な記述は簡単ではない．そのため，システムに対する要求を分かりやすい形で書く手法が求められる．

本論文では，プロブレムフレームに基づいたシステムのモデル化をサポートする手法を提案する．本手法では，要求とドメインプロパティから仕様を状態マシンとして生成する．ただし，ドメインプロパティは状態マシンとして，要求は ”安全状態 ”という概念を用いドメインの振る舞いに対する ”制約 ”として規定する．結果として得られるシステムの仕様は，その時のドメインと安全状態の規定の枠組み内でシステムが出来る最大限のパスを含む状態マシンを表す．本手法は網羅的に検査するというモデル検査[7] の基本的な概念を用いるが，モデル検査がシステムのモデル化後に適応するのに対し，提案手法はシステムのモデル化が完了する前に適応する．

2. プロブレムフレーム

Jackson はプロブレムフレームとして,「必要とされる振舞い」,「命令される振舞い」,「情報表示」,「仕掛品」,「変換」の 5 つのフレームを提案しているが，今回取り上げているフレームは「必要とされる (又は命令される) 振る舞い」フレームと呼ばれるものである．組み込みシステムの多くがこの振る舞いフレームに当てはまる[8] ため，今回はこれ以外のフレームに関しては考慮しない．

プロブレムフレームではシステムを，開発の対象である「マシン」と，それを取り巻く外部環境である制御対象の「ドメイン」，ドメインに対する振る舞いの「要求」との３つを明確に分けて考える．図 1 にプロブレムフレームのプロブレム図の例を示す．

© 2011 Information Processing Society of Japan

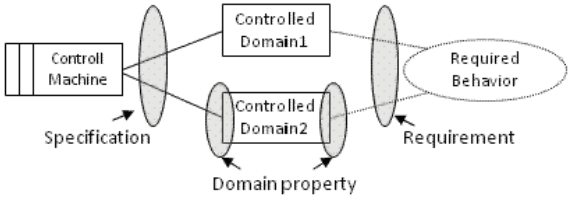

図 1　プロブレム図
Fig. 1　The Problem frame diagram.

矩形はドメイン，垂直に 2 本線の入った矩形はマシン，楕円は要求を示す．また，マシンとドメインの間に引かれた線はそれぞれのインタフェースを，要求とドメインの間に引かれた線はドメインに対する要求参照を表している．各プロブレム図は 1 つのマシンと要求を持ち，1 つ以上のドメインを持つ．マシンと外部環境である問題を区別し，また両者の関係を明らかにするために，プロブレム図のドメインの多くは物理的に存在するものを記述する．

ここで，それぞれのインタフェースと要求参照の規定を説明する．マシンによってどのようにドメインが制御されるかを記述したものを ”仕様 ”，ドメインがどのように振る舞って欲しいかを記述した要求の制約を ”要求 ”，ドメインのインタフェースと要求参照の関連の規定を ”ドメインプロパティ ”と呼ぶ．”要求”はシステムに対する顧客の要求であり，”願望法”で表現する．”ドメインプロパティ”はマシンの振る舞いに関わらない客観的事実について示すため”所与”であり，”直接法”で表現する．”仕様”はマシンの設計，つまりプログラミングを行う上でのマシンの仕様書であり，ドメインの制御に対するマシンの望ましい振る舞いの記述であり，”願望法”で表現する．問題の分析で，それが”所与”か”願望”であるか明確に区別することは重要である．

3. 仕様の生成

仕様の生成方式は以下の 6 つのステップからなる．

① ドメインプロパティを状態マシンとして定義
② 要求であるドメインの制約を安全状態として与える
③ 全てのドメインの状態の組み合わせを生成
④ ③から安全状態と入力遷移だけの状態マシンを生成
⑤ ④に必要な条件を加える
⑥ ⑤に必要な出力遷移を加える

上記のステップのうち，① ② ⑤ は手動，③ ④ ⑥はツールにより自動で行う．上記の手順のうち，① ②は元の問題の記述，⑤は要求の詳細な記述である．④である入力イベントから次の安全状態が存在しない場合や，到達出来ない安全状態が存在する場合，また⑥で出力イベントのリストが不足または冗長だった場合など，エラーとして報告される．これらのエラーを取り除き適切な仕様が得られるまで，ドメインプロパティと安全状態を訂正し，③から⑥の手順を繰り返す．全ての手順を終えて，最終的に得られる仕様 Spec = (S, Init, I, O, C, T) の要素は以下の通りである．

S : 安全状態
Init $\subseteq$ S $\times$ C : 初期状態
I : 入力イベント
O : 出力イベント
C : ガード条件
T $\subseteq$ S $\times$ I $\times$ C $\times$ O* $\times$ S : 遷移規則

得られる遷移規則 T は，安全状態間の遷移規則で表され，トリガーとなる入力イベント，条件記述，アクションとなる出力イベントのリストで構成される．

3.1　各ドメインの状態マシンの定義

各ドメイン $D_1, .., D_n$ の状態マシンを，それぞれ $D_i = (S_i, s_{i0}, I_i, O_i, T_i)(1 \leqq i \leqq n)$ とする．各要素は以下の定義とする．

S_i : 状態の有限集合
$s_{i0} \in S_i$: 初期状態
I_i : 入力イベント
O_i : 出力イベント，ただし $I_i \cap O_i = \emptyset$.
$T_i \subseteq S_i \times (I_i \cup O_i) \times S_i$: 遷移規則

遷移規則は決定性を持つものとする．すなわち，もし (s, e, s_1) と (s, e, s_2) が共に T_i に存在するなら $s_1 = s_2$ である．e が入力イベントなら，(s, e, s') を入力遷移，そうでなければ出力遷移と呼ぶ．入力遷移 (s, e, s') は， ドメインの状態が s の時，入力イベント e が起こりうることを意味し，これはマシンによって処理されなければならない．この時，ドメインの状態が s' に変わる．出力遷移 (s, e, s') は，ドメイン状態が s の時，マシンは出力イベント e をドメインに送る事が出来る事を意味する．この時，ドメインの状態が s' に変わる．

3.2　安全状態の定義

要求は，マシンの安定状態において同時に起こることが出来る各ドメインの状態の組み合わせとして定義する．ここでいう安定状態とは入力待ち状態を意味する．タイムアウトのようなタイマーイベントも入力イベントとして扱う．この同時に成立することを許された安定

© 2011 Information Processing Society of Japan

状態を”安全状態”と呼ぶ．安全状態の集合を Safe とする．

$$\mathrm{Safe} \subseteq \prod_{i=0}^{n} \mathrm{S}_i (1 \leqq i \leqq n)$$

我々は今回いくつかの組み込みシステムに置いて，状態の対応関係を規定するこの安全状態の集合を，要求の本質として比較的自然に記述が出来ることを確認した[8]．

3.3　ドメインの合成状態マシンの生成

$\mathrm{D}_i (1 \leqq i \leqq n)$ を合成した状態マシン $\mathrm{Dx} = (\mathrm{Sx}, \mathrm{sx}_0, \mathrm{Ix}, \mathrm{Ox}, \mathrm{Tx})$ を作成する．Dx の各要素は以下の定義となる．

$\mathrm{Sx} \subseteq \prod_{i=1}^{n} \mathrm{S}_i$　　：ドメインの状態の組み合わせ
$\mathrm{sx}_0 = (\mathrm{s}_{10}, .., \mathrm{s}_{n0})$ ：ドメインの初期状態の組み合わせ
$\mathrm{Ix} \subseteq \bigcup_{i=1}^{n} \mathrm{I}'_i$　　：全てのドメインの入力イベント，ただし $\mathrm{I}'_i = \{\text{”}\mathrm{D}_i?e\text{”} | e \in \mathrm{I}_i\}$
$\mathrm{Ox} \subseteq \bigcup_{i=1}^{n} \mathrm{O}'_i$　　：全てのドメインの出力イベント，ただし $\mathrm{O}'_i = \{\text{”}\mathrm{D}_i!e\text{”} | e \in \mathrm{O}_i\}$
$\mathrm{Tx} \subseteq \mathrm{Sx} \times (\mathrm{Ix} \cup \mathrm{Ox}) \times \mathrm{Sx}$：合成遷移規則

Ix の要素であるイベント”$\mathrm{D}_i?e$”は ”D_i”がマシンに対して発したイベント e であることを，また Ox の要素である”$\mathrm{D}_i!e$”はマシンから”D_i”に発されたイベント e である事を示す．イベントの前にドメインの名前をつけることで，全てのイベントが一意に決まる．

また，Tx の遷移規則を次のように定義する．

$$\mathrm{Tx} = \{((s_1, .., s_n), \mathrm{D}k@e, (s'_1, .., s'_n)) | \\ s_i = s'_i \text{ for } 1 \leqq i \leqq n \text{ and } i \neq k, \text{ and } (sk, e, sk') \in \mathrm{T}k\}$$

ただし，e が入力イベントなら@ は ？，出力イベントなら！とする．システムに多くのドメインが存在する場合 Tx は膨大な遷移規則を含む．状態の数，遷移規則の数を制限するために，初期状態 sx_0 から到達できない状態は Sx から取り除く．

3.4　安定状態と入力遷移のみからなる状態遷移を生成

安定状態間の遷移規則 Ts を含む状態マシン $\mathrm{Ds} = (\mathrm{Ss}, \mathrm{Ss}_0, \mathrm{Is}, \mathrm{Ts})$ を，合成ドメイン Dx と安全状態 Safe から生成する．Ds の要素は以下の通りとする．

$\mathrm{Ss} \subseteq \mathrm{Safe} \cup \{\mathrm{ERR}\}$：安全状態とエラー状態
$\mathrm{Ss}_0 \subseteq \mathrm{Safe}$：初期状態の候補
$\mathrm{Is} \subseteq \mathrm{Ix}$：入力イベント
$\mathrm{Ts} \subseteq \mathrm{Safe} \times \mathrm{Is} \times \mathrm{Ss}$：遷移規則

Ts の内容を記述するために，Tx の推移閉包 $\mathrm{Tx}^* \subseteq \mathrm{Sx} \times (\mathrm{Ix} \cup \mathrm{Ox})^* \times \mathrm{Sx}$ を以下のように定義する．

・任意の Sx の要素 s について $(s, \varepsilon, s) \in \mathrm{Tx}^*$（$\varepsilon$ は空列）
・$(s, e, s_1) \in \mathrm{Tx}$ かつ $(s_1, list, s_2) \in \mathrm{Tx}^*$ ならば $(s, (e.list), s_2) \in \mathrm{Tx}^*$

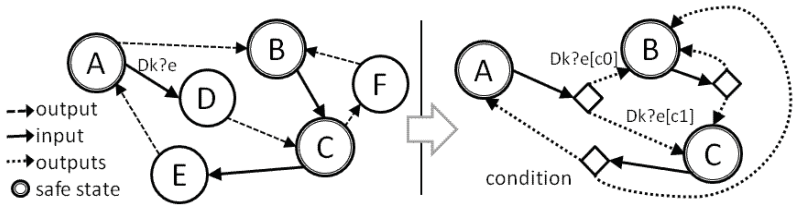

図 2　仕様生成の概念
Fig. 2　The concept of Specification generation.

Tx^* を用い，s から 0 個以上の出力遷移のみで到達可能な状態の集合 $\mathrm{SOx}(s)$ を定義する．

$$\mathrm{SOx}(s) = \{s' \in \mathrm{Sx} | \exists os \in \mathrm{Ox}^* : (s, os, s') \in \mathrm{Tx}^*\}$$

Ss_0 は，初期状態 sx_0 から出力遷移のみで到達可能な安全状態の集合である．

$$\mathrm{Ss}_0 = \mathrm{SOx}(\mathrm{sx}_0) \cap \mathrm{Safe}$$

ここで Ss_0 が空なら，新たな安全状態を Safe に追加する必要がある．この場合 $\mathrm{SOx}(\mathrm{sx}_0)$ は，Ss_0 を定義するのための候補となる．

また，Ts は以下のように生成される．Tx から安全状態で始まる全ての入力遷移 $\mathrm{Ts}_1 = \{(s_1, e, s) \in \mathrm{Tx} | s_1 \in \mathrm{Safe} \text{ and } e \in \mathrm{Ix}\}$ を選択し，$(s_1, e, s) \in \mathrm{Ts}_1$ に対して $\mathrm{SOx}(s)$ を作る．もし $\mathrm{SOx}(s) \cap \mathrm{Safe}$ が空でなければ，全ての $\mathrm{s}_2 \in \mathrm{SOx(s)} \cap \mathrm{Safe}$ に対して (s_1, e, s_2) を Ts に追加する．もし，$\mathrm{SOx(s)} \cap \mathrm{Safe}$ が空なら，(s_1, e, ERR) を Ts に追加する．ここで，SOx(s) はこのエラーを修正するための新たな安全状態の候補となる．形式的には，Ts は以下のように定義出来る．

$$\mathrm{Ts} = \{(s_1, e, s_2) | s_1 \in \mathrm{Safe}, e \in \mathrm{Ix}, \exists s \in \mathrm{Sx} : (s_1, e, s) \in \mathrm{Tx} \\ \text{もし } \mathrm{SOx}(s) \cap \mathrm{Safe} \neq \emptyset \text{ なら } s_2 \in \mathrm{SOx}(s) \cap \mathrm{Safe}, \\ \text{そうでなければ } s_2 = \mathrm{ERR}\}$$

ここで Ts 生成の概念を図 2 に示す．最初に，合成遷移 Tx から A,B,C のような入力遷移を伴う安全状態を定める．入力遷移によりそれぞれの状態は A から D，B から C，C から E へと変化する．ここで，A からの遷移に注目すると，D から出力遷移のみで到達可能な次の安全状態 B,C となり (A, $\mathrm{D}_k?e$, B)，(A, $\mathrm{D}_k?e$, C) を得ることが出来る．

ここで，もし Ts にエラーが含まれていた場合，エラー発見と同時に生成される新しい安定状態の候補の中から，安定状態の追加定義が必要となる．この過程はエラーがなくなるまで繰り返し行う．

3.5　条件の付加

ドメインの状態マシン Di と合成遷移 Dx は共に決定性を持つが，3.4 章で得られた Ds

© 2011 Information Processing Society of Japan

は決定性を持たないため Ds に決定性を持たせるために遷移規則 Ts にガード条件を付加する．ガード条件を持つ遷移は以下の 3 つのタイプに分類できる．

① 有効でない遷移
② 出現しうる遷移 (ただし今は必要ない).
③ 必要な遷移

もし初期状態 Ss_0 が複数含まれる場合，Ss_0 の各状態にもガード条件を付け加える．

①，②に関しては，”false”という条件をつけ遷移を削除し，③に関しては適切なガード条件を指定する．同じ手順を初期状態 Ss_0 の集合に対しても行う．このようにガード条件を付加することで，不要な遷移を除いた決定性を持つ状態マシンを得ることが出来る．ここで，簡便のため (s_1, e, c, s_2) を $(s_1, e[c], s_2)$ と表記する．また，e は $e[true]$ を意味する．

3.6　必要な出力イベントの生成

最後に，3.5 で得られた遷移規則に必要な出力イベントのリストをアクションとして追加する．$(s_1, e[c], s_2) \in$Ts に対して，$(s_1, e, s) \in$Tx ならば $s_2 \in$OSx(s)，つまり s_2 は s から出力遷移のみで到達可能である．$(s_1, e[c], s_2) \in$Ts の遷移に伴うアクションは s から $s2$ に遷移するのに必要な出力イベントのリスト $os \in$ Ox*である．os が空列の場合は入力イベントだけで次の安定状態に到達することを意味する．出力イベントのリストは合成遷移から簡単に求めることができ，最終的な仕様は，$(s_1, e[c], os, s_2)$ として得られる．

4.　電話交換システムの例

3 で記述した手法を，簡単化した電話交換システムの例を用いて説明する．電話交換システムは発呼電話 (Caller)，着呼電話 (Callee)，回線 (Network)，課金装置 (Billing) の 4 つのドメインを持つ．ここではオブジェクト指向スクリプト言語 Ruby を用いた．

4.1　各ドメインの状態マシンの定義

ここで，発呼，着呼，回線，課金ドメインの状態マシンを定義する．

(1) 発呼電話ドメイン：D_1

D_1 は，*offHook, onHook, dial* の 3 つの入力イベント，*call, start, finish* の 3 つの出力イベントを持つ．電話が *offHook* された時，状態は初期状態 *idle* から *dialing* へと変化する．状態が *dialling* の時，発呼電話は全ての電話番号が入力されるまで *dial* を繰り返す．*dial* の入力が終了すると，マシンはダイアルの終了を通知する *call* イベントを送り，状態を *calling* へと変更する．*calling* 状態時，着呼電話が *offHook* した場合，マシンは会話の開始を通知するために *start* イベントを送り，状態を *conv* へと変化させる．会話が終了し，着呼電話が *onHook* した場合，マシンは *finish* イベントを送り状態を *waitFin* へと変化させる．逆に， 会話終了時に発呼電話が *onHook* した場合，マシンは発呼電話から *onHook* イベントを受け取り，状態を *idle* へと変化させる．

Ruby を用い D_1 の状態マシンの遷移規則を記述すると以下のように表される．

```
d1tran = [ ["idle","offHook","dialing"], ["dialing","dial","dialing"],
           ["dialing","call","calling"], ["calling","start","conv"],
           ["conv","finish","waitFin"], ["waitFin","onHook","idle"],
           ["dialing","onHook", "idle"], ["calling","onHook","idle"],
           ["conv","onHook","idle"] ]
```

(2) 着呼電話ドメイン：D_2

D_1 と同様に，D_2 の状態マシンの遷移規則を記述すると以下のように表される．ただし，*offHook, onHook* は入力イベント，*call, finish* は出力イベントである．

```
d2tran = [ ["idle","call","calling"],["calling","offHook","conv"],
           ["conv","finish","waitFin"],["waitFin","onHook","idle"],
           ["calling","finish","idle"],["conv","onHook","idle"] ]
```

(3) 回線ドメイン：D_3

D_3 の状態マシンの遷移規則を記述すると以下のように表される．ただし，*reserve, connect, disconn, cancel* は全て出力イベントである．

```
d3tran = [ ["idle","reserve","resv"], ["resv","connect","conn"],
           ["conn","disconn","idle"], ["resv","cancel", "idle"] ]
```

(4) 課金装置ドメイン：D_4

課金装置ドメインは *on, off* の出力イベントのみのシンプルな状態マシンを持つ．D_4 の状態マシンの遷移規則を記述すると以下のように表される．

```
d4tran = [ ["idle","on","charge"], ["charge","off","idle"] ]
```

4.2　安定状態の定義

電話交換システムの要求を定義する際には，各ドメインの状態の許される組み合わせを定める必要がある．例えばシステムが通話中になるためには，システムの状態は初期状態 *Idle* から，*Dialing, Calling, Talking* と変わっていく．具体的には，*Calling* の時，発呼電話と着呼電話ドメインは呼び出し音が鳴っている状態 (*calling*)，回線は予約済み (*reservation*)，課金装置は *idle* 状態である必要がある．この様なシステムの要件において必要なドメインの状態の組み合わせを，”安全状態”と呼ぶ．

電話交換システムにおけるすべての安全状態を表 1 に示す．この安全状態という概念を用いることで，システムがどう振る舞って欲しいかという要求の本質な部分を表すことが可能であると考えている．

© 2011 Information Processing Society of Japan

表 1　電話交換システムにおける安全状態
Table 1　Safe states for telephone switching system.

	Idle	Dialing	Calling	Talking	WaitFin_Caller	WaitFin_Callee
Caller	idle	dialing	calling	conv	waitFin	idle
Callee	idle	idle	calling	conv	idle	waitFin
Network	idle	idle	resv	conn	idle	idle
Billing	idle	idle	idle	charge	idle	idle

4.3　合成状態マシンの生成

3.3 で述べた定義に基づき，D_1, D_2, D_3, D_4 から自動的に合成状態マシン Dx を生成する．合成された遷移規則 Tx の要素数は 676 個となる．

4.4　安定状態と入力遷移のみからなる状態マシンを生成

3.4 で記述した Ds の定義に沿って，安全状態と入力イベントのみからなる状態マシン Ds を生成する．生成された 13 個の遷移規則を以下に示す．

```
[["Idle", "Caller?offHook[c0]", "Dialing"]
 ["Idle", "Caller?offHook[c1]", "WaitFin_Caller"]
 ["Idle", "Caller?offHook[c2]", "Calling"]
 ["WaitFin_Caller", "Caller?onHook", "Idle"]
 ["Calling", "Caller?onHook", "Idle"]
 ["Calling", "Callee?offHook", "Talking"]
 ["Talking", "Caller?onHook", "WaitFin_Callee"]
 ["Talking", "Callee?onHook", "WaitFin_Caller"]
 ["WaitFin_Callee", "Callee?onHook", "Idle"]
 ["Dialing", "Caller?dial[c3]", "Dialing"]
 ["Dialing", "Caller?dial[c4]", "WaitFin_Caller"]
 ["Dialing", "Caller?dial[c5]", "Calling"]
 ["Dialing", "Caller?onHook", "Idle"]]
```

上記の状態名は表 1 に準ずる．同じ安全状態・入力イベントで始まる複数の遷移規則に決定性を持たせるためにガード条件 c_0,..,c_5 を付加した．もし，入力遷移 $(s_1, e, s_2) \in$ Tx において (ただし s_1 は安全状態)，s_2 から出力遷移のみで到達できる次の安全状態がなかった場合，ツールにより (s_1, e, ERR) が報告される．

ここでエラーの一例として，*WaitFin_Callee* [idle, waitFin, idle, idle] が安全状態として定義されていない場合を考える．この場合，発呼電話が *onhook* した時，*Talking* からの次の安全状態が存在しないため [Talking,"Caller?onHook",ERR] がエラーとして報告される．*ERR* には，以下のように安全状態となり得る状態の候補が報告される．

```
[idle,conv,conn,charge]
- - -
[idle,conv,idle,idle]
[idle,waitFin,conn,charg]
- - -
[idle,waitFin,idle,idle]
```

この中から，新しい安全状態として [idle,waitFin,idle,idle] を選択することになる．また，もし上記手順により得られた遷移規則に必要な遷移が存在しない場合，各ドメインの状態マシンの定義を見直す必要がある．

4.5　ガード条件の記述

最後にガード条件を適切な意味に書き加える．3.5 章で説明したように遷移には 3 つのタイプがある．①については当然削除の対象となる．②については，今回考えるシステムのバージョンに含むか否かを決める必要がある．その後，すべての条件に適切な説明を記述する．電話交換システムの例では，以下の 6 つのガード条件が存在する．

```
["Idle", "Caller?offHook[c0]", "Dialing"]
["Idle", "Caller?offHook[c1]", "WaitFin_Caller"]
["Idle", "Caller?offHook[c2]", "Calling"]
["Dialing", "Caller?dial[c3]", "Dialing"]
["Dialing", "Caller?dial[c4]", "WaitFin_Caller"]
["Dialing", "Caller?dial[c5]", "Calling"]
```

Idle から *Caller?offHook* イベントによる遷移は，*Dialing, WaitFInCaller, Calling* への 3 つがありそれぞれ c_0, c_1, c_2 の条件が付けられている．ここで，それぞれの条件の内容について考える．c_0 は通常遷移である．c_1 は *Dialing, Calling* への遷移が拒否されており，加入者へのサービスの提供を禁止している場合や回線が混雑している場合が考えられる．c_2 は，*Dialing* を経由することなく発呼電話と着呼電話が繋がり呼び出し中となったことから，直通電話であることが考えられる．今回考えるシステムでは直通電話の機能を実装しないこととし，c_2 の遷移を削除した．その結果，*Caller?offHook* の条件は以下の通りである．

c0：normal(not blocking),
c1：Blocking,
c2：false (direct call but not implemented).

Dialing から *Caller?dial* イベントによる遷移も，*Dialing, WaitFin_Caller, Calling* の 3 つがあり，それぞれ c_3, c_4, c_5 の条件が付けられている．c_3 はダイアルナンバーの桁が不完全であり，更にダイアル入力が必要なことを表す．c_4 はダイアルナンバーが正しくない，または着呼電話が通話中などでビジーであることが考えられる．c_5 はダイアル入力が正し

© 2011 Information Processing Society of Japan

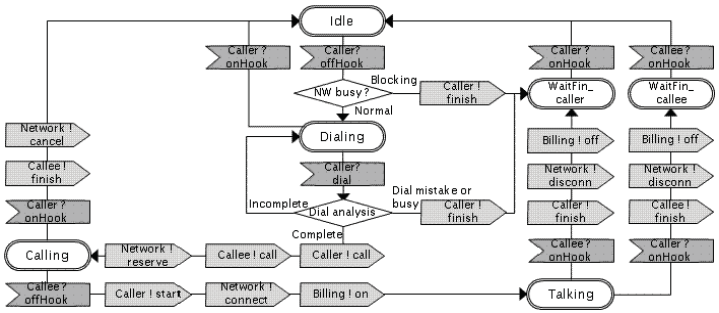

図 3　電話交換システムの仕様
Fig. 3　Specification state transition diagram for telephone switching system.

く完了し，発呼電話，着呼電話共に *Calling* となる通常遷移である．これらの条件は全て③のなけらばならない遷移である．結果，*Caller?dial* の条件は以下の通りである．

c3：Incomplete
c4：Dial mistake or busy
c5：Complete

4.6　必要な出力イベントの生成

最後に，3.6 章で記述した出力イベントのリストを生成し，追加する．図 3 にこのようにして得られた電話交換システムの最終的な遷移の仕様を示す．

5. おわりに

組み込みシステムの仕様設計において，プロブレムフレームの概念を用い，要求とドメインプロパティからマシンの仕様を得る手法を提案した．また，簡単化した電話交換システムの例を用いてその有用性を確かめた．得られた仕様は，全ての必要かつ起こり得る遷移で構成され，ドメインが出来る ”最大限の仕様 ”であると言える．もし，得られた仕様に必要な機能が存在しない場合には，外部環境であるドメインプロパティを変更する必要がある．

電話交換システムの例では，発呼電話や着呼電話のような入力イベントを多く持つドメインが問題を特定しやすくしたために，ガード条件の数が相対的に少なく，結果として ①有効でない遷移が現れなかった．on/off スイッチのみの様な，より一般的なドメインが多く使われる例については，条件①が多く現れることを確認している．それでも，本手法は組み込みシステムの仕様の導出に有用であると考えられる．

元の問題が複雑でドメインの数が多い場合，この手法のままでは状態爆発の問題を避けては通れない．元の問題があまりに大きい場合に対しては，プロブレムフレームの基本概念に則り，問題をシンプルなサブプロブレムへと分解し考えることで状態爆発を回避できる．それでも依然として残るドメインの数の問題については，現在 4.3 の合成マシンの生成の手順を見直し改良を行っている．また，仕様の導出だけでなく，完成したシステムに対してレビューを行うツールとして用いることも考えられる．モデル化したシステムを検証することで，現在の構成が最適であるかどうかをレビューし，最適化を行うことも可能である．

少し複雑な例として，ドメインの数 8 個，安全状態 14 個の ATM について検証を行ったところ，①が 93 個，②が 24 個，③が 23 個という結果になった．②の例として，カード挿入後に直ぐにカードを排出する遷移 (エラーカード等)，カード挿入後にパスワードの入力なしに認証が始まる遷移 (特別なカードではパスワードが必要ない認証もあり得る)，パスワード入力後認証せずにカードを排出する遷移 (パスワードが 0000 等の認証するまでもなくエラーと分かる場合) があった．これらは，今回は使用しなかったがいずれも出現し得る遷移であると考えられる．今後は，より実用的な組み込みシステムの例で検証を行い，ツールの有用性を評価していきたい．

参　考　文　献

1) 三瀬敏郎他: 組込みソフトウェア仕様抽出のための非正常系分析マトリクス, **情報処理学会ソフトウェア工学研究会報告**, pp.113–120 (2004).
2) Jackson, M.: *Problem Frames: Analysing* **&** *Structuring Software Development Problems*, Addison-Wesley Publishing Company (2001).
3) Seater, R. and Jackson, D.: Problem Frame Transformations : Deriving Specifications from Requirements, *2nd International workshop on Applications and Advances in Problem Frames* (2006).
4) Z.Li, J. and L.Rapanotti: From Requirements to Specifications: a Formal Approach, *2nd International workshop on Applications and Advances in Problem Frames* (2006).
5) RobinLaney, LeonorBarroca, M. J. B.N.: Composing requirements using problem frames, *12th IEEE International Requirements Engineering Conference, 2004.*, pp. 122 – 131 (2004).
6) Hoare, C.: *Communicating Sequential Processes*, Prentice Hall (1985).
7) 中島震: *SPIN* **モデル検査入門**, オーム社 (2010).
8) 紫合治: ジャクソン法（JSP）による状態遷移設計, **情報処理学会論文誌**, Vol.50, No.12, pp.3030–3040 (2009).

© 2011 Information Processing Society of Japan