社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2006－SE－151（5）
2006／3／23

インターフェースの情報を用いた状態遷移図の生成

大川　敦†　　加藤大輝‡　　紫合　治‡

† 東京電機大学情報環境学研究科　　‡ 東京電機大学情報環境学部

組み込みシステムの振る舞いは，インターフェース(外部装置など)からシステムへの入力イベントに応じて，システムがインターフェースに出力イベントを返す状態遷移図で表すことができる．ここでは，システムの振る舞いを記述する前に，まず対象システムとインターフェースとのやりとり（プロトコル）を，状態遷移図を用いて詳細に定義し，続いて，定義したプロトコルの状態遷移図を利用して，システムの振る舞いを表す状態遷移図を系統的に構築する方式を提案する．さらに，本方式による設計を支援するシステムを開発し，支援システムを用いた本方式の設計の有効性を検証した．

Generating State Transition Diagram Using Interface Specification

Atushi Okawa†　　Daiki Kato‡　　Osamu Shigo‡

† Graduate School of　Information Environment, Tokyo Denki University

‡ School of Information Environment, Tokyo Denki University

The behavior of the embedded system can be expressed by state transition diagram that receives an input event from the interface (external device etc.) then sends output actions to the interface. This paper proposes a design method that defines the communication between target system and interface by protocol state transition diagram before the system behavior is described, then systematically constructs the state transition diagram of system behavior using the protocol state transition diagram. We developed the system that supports proposed design method. The effectiveness of the design method using the support system evaluated by experimental design projects is reported.

1. はじめに

近年，ユビキタスシステムの発展に伴い，組み込みシステムの重要性が高まってきている．一般にシステムの設計では，その構造と振る舞いを記述するが，組み込みシステムの場合，特に振る舞いの設計が重要になる．組み込みシステムの振る舞いの設計では，システムがイベントを受け，適切なアクションを返す規則を定める状態遷移図による設計が中心的な作業となる．これはUML[1][2]のステートチャートを用いて表記されることが多い．しかし，システムの設計の始めから正確な状態遷移図を記述することは難しいといえる．そのため，容易に状態遷移図を記述できる方式と，その適切な支援ツールが望まれる．

本論文では，リアルタイムシステム向きの設計技法であるROOM[3][4]を拡張し，対象システムのインターフェースの仕様を，そのインターフェースが表す外部装置の規則として定義し，その規則を用いて，システムの振る舞いを示す状態遷移図を系統的に構築する方式[5]を提案し，この提案方式を支援するために開発したシステムについて述べる．これにより，システムの設計にかかる時間の短縮や設計ミスによる手戻りが防げるようになり，システム開発における設計段階のコストの減少を目指す．

2. 従来の技術と問題点

ここでは従来のモデリング技術とその問題点についてUMLとROOMを例に述べる．

2.1. UML

UMLとは，Unified Modeling Language(統一モデリング言語)の略で，システムの分析や，設計，実装などを円滑に進めるためのモデルの表記法を定義したものである．UMLは現在ではモデリング手法として世界標準となりつつある．UMLは，クラス図やシーケンス図，ステートチャート図などのシステム設計において必要な設計手法をまとめたもので，様々なダイアグラムがUMLには存在し，いろいろな場面で利用される．しかし，この多様なダイアグラムがUMLのデメリットでもある．UMLを利用する際は，これらのダイアグラムの全てを利用する必要がないが，それぞれのシステム設計に必要なUMLのダイアグラムを選択する能力が求められる．

またUMLは，UML2.0[1][2]という新しい概念やexecutable UML[6]という実行向きの拡張概念が存在し，幅広く利用できるようになっている．しかし，振る舞いをUMLのステートチャート図で表現する場合，システム内

部で行われる振る舞いや状態の遷移に関しては記述しやすいが，対象システムの外部にあるオブジェクトとの関連を利用した振る舞いを容易に記述することはできない．executable UML による設計では，他のオブジェクトとのやり取りや振る舞いを詳細に記述することが可能であるが，その図はほとんどプログラムのコーディングに近いため，難易度や複雑さが増しているといえる．

2.2. ROOM

ROOM は，Selic らにより提唱されたリアルタイムシステム向きのモデリング技法であり，Real-time Object-Oriented Modeling の略である．ROOM では，システムの構造と振る舞いを統一的に記述することが出来，ポートやプロトコルといった一部の概念が UML2.0 にも導入されている．

ROOM では，システムの構造をカプセル，ポート，コネクタと呼ばれる要素で表現する．カプセルはシステムや部品を表し，信号の入出力を制限する窓口としてポートを持つ．ポートを流れる信号はプロトコルという型によって規定される．ポート間はコネクタによって結ばれ，信号はコネクタ上を行き交う．

以降は，電話交換システム，つまり，発信側の電話と受信側の電話とそのネットワークの仲介を行うコーディネータを例にして説明していく[7]．（図１参照）

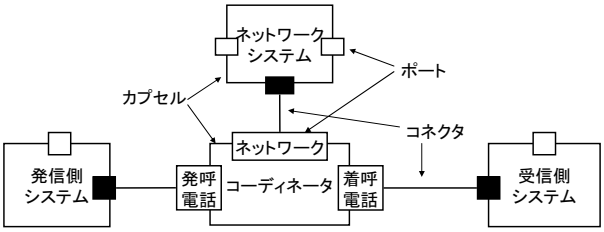

図 1. ROOM によるシステムの構造例(電話交換機)

ROOM では，カプセルに状態遷移図を記述することによって，システムの振る舞いを表現する．ここで利用される状態遷移図は UML のステートチャート図とほぼ同様の内容であり，記述方式も既存のステートチャート図と同様である．

UML は多くダイアグラムを内包しているが，ダイアグラム間の関連が明確でなく，それら多様なダイアグラムを使って設計する手順も特には示されていない．つまり，UML は様々な図表現の寄せ集め的なものであると言える．それに対して，ROOM では各種の図の関係が明確に定義されているため，ROOM の設計図はシステム全体の設計が見通しやすい．また，ROOM のダイアグラムは，それらを設計に使う順序なども示されているため設計がしやすいという利点が挙げられる．

また，前述したとおり UML のステートチャート図は，対象システムと関連のある他のオブジェクトとの信号のやり取りを理解しやすいとは言いがたい．その点 ROOM ではプロトコルによってシステム間のやり取りが示されているため，まだ理解しやすいといえる．しかし，システムの振る舞いの記述におけるプロトコルの使い方までは示されていない．

3. ROOM の拡張

ここでは，インターフェースの情報を用いた状態遷移図の生成方法について説明し，実際にその方式を用いた利用例を提示する．

3.1. ROOM の利用

既存の技術では，システムの振る舞いを表す状態遷移図をはじめから正確に記述が可能であるとは言いがたい．そこで我々は ROOM を利用し，対象システムの振る舞いの記述を，外部システムとの信号のやり取りを用いて記述する方式を提案する．

3.1.1. インターフェース情報の利用

ROOM では，外部システムとの信号のやり取りをポート，つまりそのシステムのインターフェース部分で行い，そのインターフェース部分の情報はプロトコルという概念によって定められている．我々はこのインターフェースの情報，つまりプロトコルを利用し，入力信号を状態遷移図の遷移のイベントとし，出力信号を状態遷移図の遷移のアクションとすることとし，状態遷移図の記述を行うことにする．

例えば，電話交換システムのコーディネータは３つのポートを持っており，それぞれが入力信号と出力信号の入口になっている．この３つのポートのプロトコルの情報である入出力信号を利用して，状態遷移図の記述を行う（図 2 参照）．しかし，このプロトコルの情報だけでは，状態遷移図の作成時に利用されるイベント名，アクション名が分かるだけであり，状態遷移図の作成がそれほど効率的になるとは言えない. そこで, 我々はプロトコルの拡張を行った.

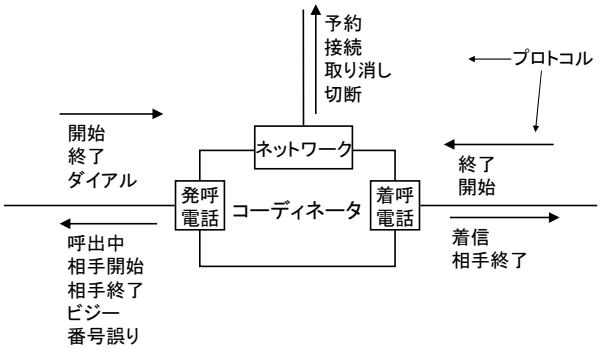

図 2. ROOM のプロトコル

3.1.2. プロトコルの拡張

外部装置に対応するインターフェースのより詳細な情報を用いるために，我々は ROOM のプロトコルの拡張を行った．従来の ROOM のプロトコルは，ポートを流れる入出力信号の集合のみを規定した（図３左参照）．ここではわかりやすいように入力信号には＜？＞マークを，出力信号には＜！＞マークを信号名の前に書いている．

我々は，プロトコルに状態遷移図を追加することによって，ポートを流れる入出力信号の妥当な順番とタイミングを規定し，インターフェースの規則として定義した[8](図３右参照).

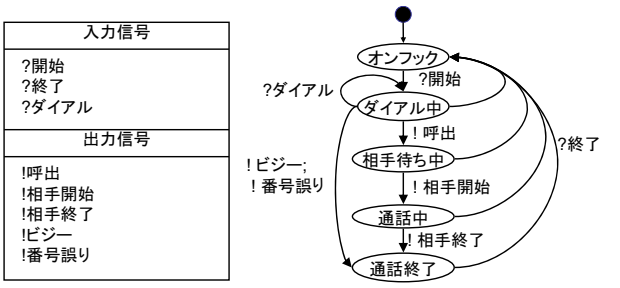

図 3. 発呼電話ポートのプロトコル状態遷移図

図 3 は，図 2 のコーディネータの発呼電話ポートのプロトコルを表したものである．プロトコルの状態は●を開始点とし●の次の状態から始まる．各状態から次の状態への遷移は入力信号を用いた入力イベントによる入力遷移と，出力信号を用いた出力イベントによる出力遷移から成る．入力遷移はその状態のときに発生しうる入力イベントを，出力遷移はその状態のときに発生させてもよい出力イベントを示す．例として，'ダイアル中'の状態からは入力遷移として"ダイアル", または"終了"の入力イベントを受け取る必要があり，出力遷移として"呼出", "ビジー", "番号誤り"の出力イベントを発生させることができることを表す．

3.2. 系統的状態遷移図の作成

プロトコルの組み合わせを用いて，系統的にシステムの状態遷移図を記述する方法について述べる．

システム（カプセル）の状態遷移図において，状態は状態名とポートの状態の組み合わせで表すものとする．ポートの状態の組み合わせが全て同じ状態は，潜在的に同じ状態ということとなり，一つの状態にまとめられる． カプセルの状態遷移図の遷移は，ポートの入力イベントを 1 つ受けることで遷移し，必要なアクションとしてポートの出力イベントを複数(0 個以上)出し，次の状態に遷移する．さらにこの時，入力イベントの直後に条件判定を加えることができるものとする．

コーディネータを例として，振る舞いとなる状態遷移図を作ってみる．まずはコーディネータのポートのプロトコルの情報が必要となる (図 4 参照)．この情報を用いてシステムの状態遷移図を作っていく．

対象システムの初期状態は，ポートの初期状態の組み合わせとなる．図 4 のプロトコルの情報より発呼電話，着呼電話，ネットワークのそれぞれの初期状態は●の次の状態なので，『発呼電話 : オンフック，着呼電話 : オンフック，ネットワーク : 予約待ち中』と表せる．ここでは，「待機中」という状態名とする．

図 4 より初期状態から受け取れる入力イベントは発呼電話からの"開始"イベントだけである．"開始"イベントを受け取ることで，発呼電話の状態は'オンフック'から'オフフック'に遷移する．カプセルの状態も『発呼電話 : オンフック，着呼電話 : オンフック，ネットワーク : 予約待ち中』から『発呼電話 : ダイアル中，着呼電話 : オンフック，ネットワーク : 予約待ち中』(状態名は「発呼電話ダイアル中」とする) に遷移する (図 5 参照)．

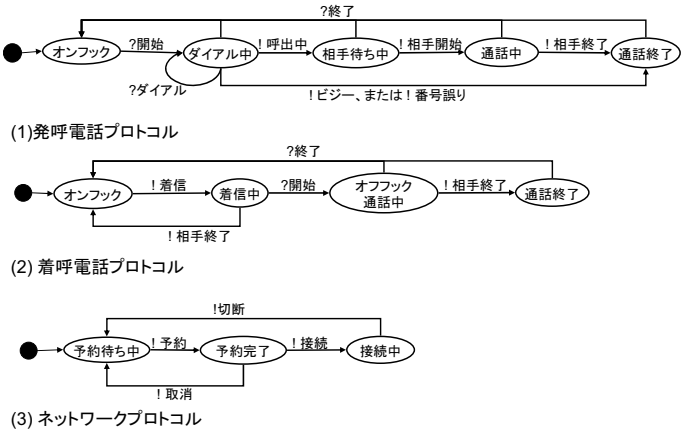

図 4. コーディネータの全ポートのプロトコル情報

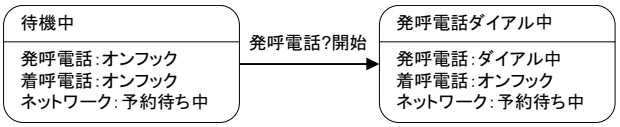

図 5. 系統的状態遷移図生成例 1

次に，システムの状態「発呼電話ダイアル中」を見てみる．発呼電話が'ダイアル中'の場合，発呼電話のプロトコルより，2 つの入力イベント，"ダイアル"と"終了"を受け取る必要がある．

"終了"イベントを受け取った場合，発呼電話は'ダイアル中'から'オンフック'の状態に戻り，発呼電話が受話器を下ろしただけでは着呼電話やネットワークは何もしないため，他のプロトコルは何もしない処理にしたいので，カプセルの状態は『発呼電話 : オンフック，着呼電話 : オンフック，ネットワーク : 予約待ち中』となる．これは，初期状態の「待機中」と同じポートの状態の組み合わせなので，コーディネータは状態「発呼電話ダイアル中」から初期状態「待機中」に戻る．

"ダイアル"イベントを受け取った場合は，ダイアルの押された数や並び，回線状態をチェックする必要があるので，条件判定を加える必要がある．条件判定の記号は◇の形で表され，条件を表す文字列は[ ]内に記述される．条件判定が[成功]だった場合は，発呼電話の状態'ダイアル中'から相手を呼び出しているとことを意味する"呼出中"イベントを，着呼電話'オンフック'からは電話がかかってきたことを知らせる"着信"イベントを，ネットワーク'予約待ち中'からは回線を予約する"予約"イベントを出す．これらの出力イベントは，図 4 のポートのプロトコルの仕様より，『発呼電話 : ダイアル中，着呼電話 : オンフック，ネットワーク : 予約待ち中』，つまり「発呼電話ダイアル中」の状態で出力可能であることがわかる．さらに発呼電話は'ダイアル中'から'相手待ち中'，着呼電話は'オンフック'から'着信中'，ネットワークは'予約待ち中'から'予約完了'へ遷移し，カプセルの状態は『発呼電話 : ダイアル中，着呼電話 : オンフック，ネットワーク : 予約待ち中』から『発呼電話 : 相手待

ち中，着呼電話 : 着信中，ネットワーク : 予約完了』へ遷移することが可能である．ここではこの状態を「着呼電話着信」する．また，条件判定が[未完]だった場合は，コーディネータは何もしないので，『発呼電話 : ダイアル中，着呼電話 : オンフック，ネットワーク : 予約待ち中』のままとなり，「発呼電話ダイアル中」と同じポートの状態の組み合わせになるので，元に戻る遷移となる．最後に，[ビジーor 番号誤り]だった場合は，発呼電話に”ビジーor 番号誤り”という出力イベントを出し，発呼電話は’ダイアル中’から’通話終了’の状態へ遷移する．また他のポートは何もしない処理としたいので，「発呼電話ダイアル中」から『発呼電話 : 通話終了，着呼電話 : オンフック，ネットワーク : 予約待ち中』へと遷移する．ここではこの状態を「発呼側終了待ち」とする（図 6 参照）

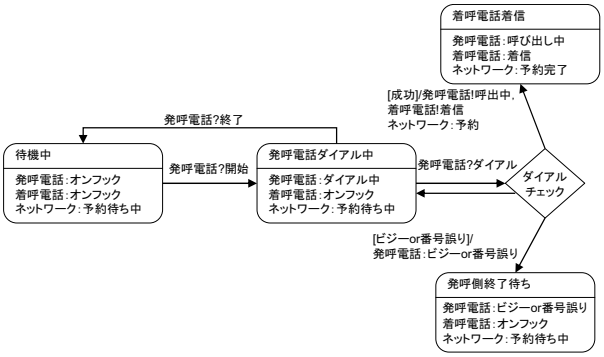

図 6. 系統的状態遷移図生成例 2

このように，インターフェースの情報（ポートのプロトコル）を用いることによって，カプセルの各状態で発生する入力イベントや，そのイベントによる遷移で実行可能なアクションの候補(出力イベントの集合)をリストアップすることができる．設計者はそこからイベントを選択する方式で，系統的にカプセルの状態遷移図を作成することができる．図 7 はこのようにして生成された電話交換システムのコーディネータの状態遷移図であり，これは，図 4 のポートのプロトコル仕様を満たしている．（図 7 参照）

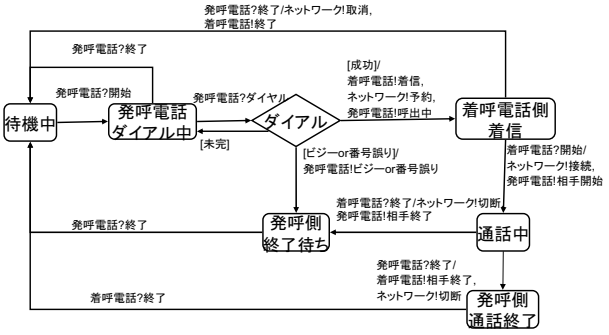

図 7. 系統的状態遷移図生成例 3

3.3. インターフェース情報を用いた振る舞いの整合検証

プロトコルを拡張し，入出力信号の順番を定める状態遷移図を作ることによって，システムの振る舞いの整合検証が行える．振る舞いの整合検証はシステムの各状態をポートの状態と比較し，入出力信号の漏れがないかをチェックする．漏れがあった場合は，状態遷移図を修正する[9]．特に，入力信号の漏れがあっては完全な状態遷移図とは言えない．この提案方式の手法を用いることで，どの状態のときにどのような信号の抜けがあるかのチェックが簡単に行える．この検証により，インターフェースの情報に整合した状態遷移図の生成が行える．

4. 支援システム

UML は，その記述をサポートするための支援システムとしては，Microsoft Office Visio[10]や Rational Rose[11]など数多く存在する．Rational Rose に関しては記述だけでなく，シミュレートも行える．支援システムの利用によるメリットは，記述ミスの減少や記述時間の短縮などの効率性の上昇が挙げられる．我々も同様に提案方式をサポートする支援システムの開発を行った．本方式をサポートする支援システムの目的は，提案方式による設計支援及び，系統的な状態遷移図の生成の支援である．

4.1. 提案方式の記述支援

提案方式による構造と振る舞いの設計を構成する要素は従来の ROOM に準拠する．構造はカプセル，ポート，コネクタの組み合わせで設計され，振る舞いは状態遷移図で設計される．支援システムは，これらの要素を用いて構造や振る舞いの設計を支援する．本システムでは，プロトコルの規定としては，入出力信号の宣言，管理を行えることに加え，3 で示したようなプロトコル状態遷移図の記述が行えるようになっている．支援システムはこれらの設計方法に特化した描画が行える．

4.1.1. プロトコルの宣言記述

新規にプロトコルを生成した場合，はじめに入出力信号の宣言を行う．プロトコルの名前と入出力信号名を入力する．信号名を決定し，入力か出力かを選択をし，追加することで，リストに登録される．次にこの入出力信号の使われる順番やタイミングを表したプロトコル状態遷移図の設計を行う．はじめに開始点と状態 1 が描画領域に配置される．状態名は状態を選択することで記述できる．状態の登録は，アイテム欄から状態の要素を選択し，描画領域に配置することで行う．遷移の登録は，アイテム欄にあるシグナル要素(矢印)を選択しそれで状態と状態を結ぶことで行う．このとき，宣言した入出力信号のリストが表示されるので，その中から信号を 1 つ選択する．信号が入力なら?信号名，出力なら!信号名で描画領域に表示される．このプロトコル状態遷移図では，入力信号は入力イベントして，出力信号は出力イベントとして利用される．遷移の矢印は，表示の見易さのために中間点を設けることができる．図 8 に，プロトコル状態遷移図作成画面の例を示す．

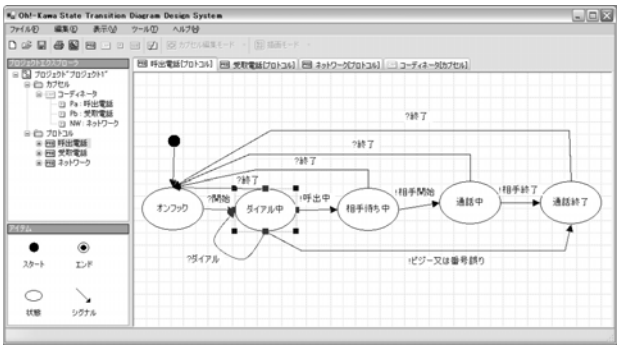

図 8. 支援システムによるプロトコルの宣言画面

4.1.2. カプセルの構造記述

システムの構造は，まずシステムを表すカプセルを生成する．カプセルを生成した場合，描画領域にカプセルを表す大きな四角ができる．カプセルを選択することで，カプセルに名前をつけることができる．

次に描画領域に生成されたカプセルにポートをつける．アイテム欄からポートの要素を選択し，それをカプセルのバウンダリ上に配置する．ポートにはプロトコルを設定する必要がある．プロトコルはすでに宣言されているプロトコルの中から選択することができる．ポートが配置されたときに宣言してあるプロトコルがリスト表示され，その中から選択する．ポートにも名前をつけることができる初期値では P1，P2 と連番で名前がつけられる．ポートを選択することで，名前の変更とプロトコルの設定ができるようになっている．

サブカプセルはカプセルの内部に記述し，ネストの関係を表す．サブカプセルもポート同様にアイテム欄から要素を選択し，描画領域に配置する．サブカプセルは，すでに生成されているカプセルのインスタンスとして扱う．そのため，サブカプセル生成時にもとのカプセルから選択することができる．サブカプセルのポートは，元のカプセルのポートの情報を参照するので，サブカプセルに対してはポートの配置ができない様になっている．サブカプセルにも名前をつけることができる．初期値では，サブカプセルのもととなるカプセル名に連番をつけたものとなる．サブカプセルを選択することで，名前の変更と元となるカプセルの変更が可能である．コネクタは描画領域内のポート同士を接続する．アイテム欄からコネクタを選択し，コネクタの両端を描画領域内の２つのポートに接続する．システムは，それぞれのポートがもつプロトコルを確認し，接続できるかどうかをチェックする．ここでは，接続されるプロトコルが共役関係になっているかどうかのチェックを行い，そうでなければ注意することに留めた．つまり，厳密な意味での接続可能性のチェックまでは行っていない．図９にカプセルの構造図作成画面の例を示す．(図９参照)

4.1.3. 振る舞いの記述支援

振る舞いの記述支援に関しては 4.2 で詳細に説明する．

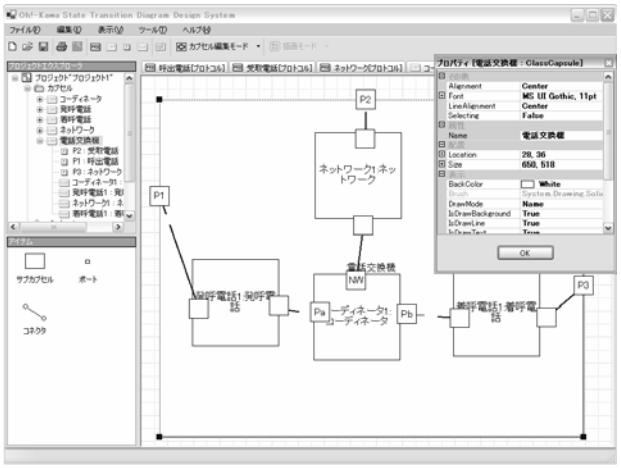

図 9. 支援システムによる構造の設計画面

4.2. 状態遷移図の系統的生成支援

提案方式は，カプセルの状態遷移図を，インターフェースの情報を参照して系統的に生成する方式である．支援システムは，プロトコル情報の参照を自動的に行うので，設計者は，支援システムが生成したアクション候補リストから必要なアクションを選択するだけで，誤りのない設計を進めることができる．

描画領域に開始点と，初期状態がはじめから表示されている．初期状態では，カプセルが持つすべてのポートの状態がそれぞれの初期状態になる．状態を選択することで，状態名の変更が可能である．次に，その状態から次の状態への遷移の記述をする．従来の通常のシステムでは，遷移の登録は遷移先の新しい状態を作り，遷移元の状態と遷移先の状態を接続する方法が使われる．本システムでは．まず遷移元となる状態を選択する．支援システムは遷移元の状態のときのポートの状態をチェックし，その状態から発生可能なイベント毎に遷移先の状態の候補を発生させる(図 10 参照)．ここで作られた遷移先の状態の候補は，まだ決定された状態ではないので実線ではなく破線で表現される．

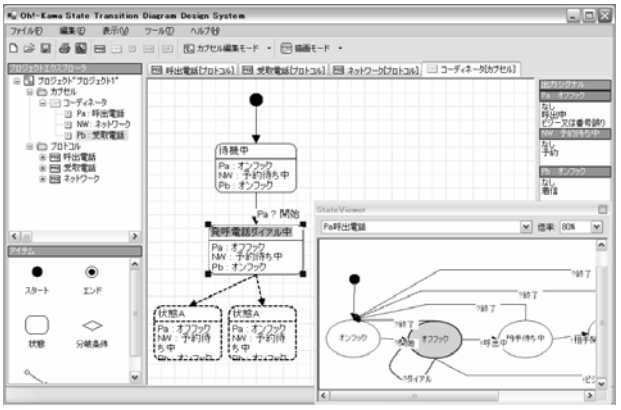

図 10.支援システムによる系統的状態遷移図作成画面

次に表示されている破線状態を選択すると，システムウィンドウの右にある出力シグナルのリストが変更される．出力シグナルのリストはカプセルが持つポートの数だけ表示され，それぞれ現在のポートの状態から出力可能なアクションのみを表示する．このアクションのリストの中から必要なものを設計者が選択し，決定することで新しい状態が完成する(図 11 参照).

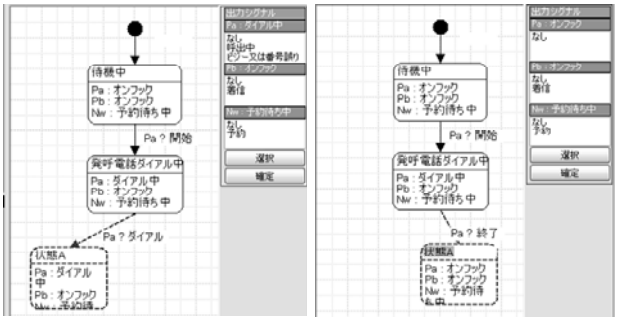

図 11.破線状態の選択と出力シグナルリストの例

新しくできた状態は，破線状態から実線の状態となり遷移先の状態となる．生成された遷移先の状態と遷移元の状態をシステムが自動的に線でつなげてくれる．生成された遷移先の状態のポートの状態の組み合わせが，すでに存在している状態のポートの状態の組み合わせと同様の場合，システムは新しく状態をつくるのではなく，遷移元の状態からすでに存在している状態に遷移を行う．

分岐により，同じイベントによる状態の遷移を複数作ることが可能である．これにより作成される状態も破線状態であり，ここで行う作業はそれぞれの状態に対して，分岐の条件と，アクションの選択を行うことである．

本方式では，状態遷移図を作る際にポートのプロトコル情報を見ることは必要不可欠である．そこで，state viewer という，アプリケーションウィンドウとは別に小型のウィンドウを用意しプロトコルの情報を見ることができるようにした．図 12 は，カプセルが「呼出電話ダイアル中」状態のときの Pa 呼出電話ポートのプロトコルを示している．ポートの現在の状態は青枠灰色で，またそこから発生しうる入力信号の遷移は赤矢印で明示される．

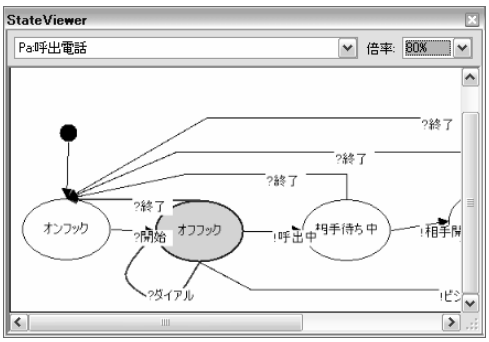

図 12.state viewer

以上のように，支援システムでは状態ごとのイベント，アクションを管理しているため，ある状態のときにその状態から遷移を起す入力イベントは自動的に選出，決定してくれる．またその状態から遷移する際のアクションの候補はシステムが選出し，設計者がその中から自由に選ぶことができる．この支援システムを用いることで，インターフェース情報を無視した誤った参照をなくすことができる．つまり，これによりインターフェースの情報に整合した状態遷移図が作成できる．また，state viewer により参照時の無駄なウィンドウ切り替えを排除でき，状態の作成や状態間の接続，同じ状態のチェックなどの必要がなくなることで，状態遷移図作成の手間が大幅に減ることが期待される．図 13 に，カプセルの状態遷移図作成の完成画面を示す.

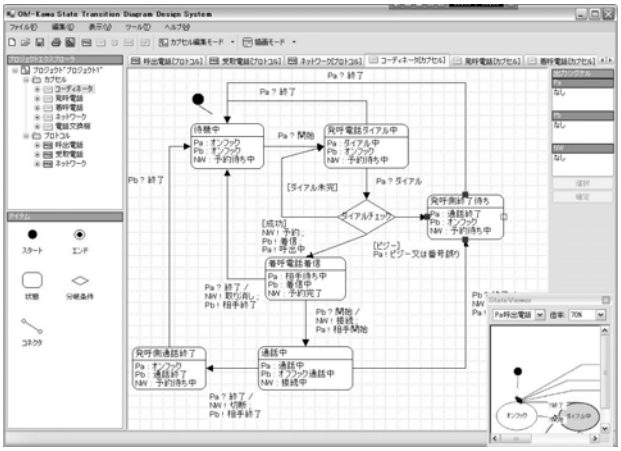

図 13.支援システムによる状態遷移図作成の完成画面

5. 評価実験

学生数人に，提案方式と従来の UML を使った設計を交互に行ってもらい，その結果の集計を行った.

5.1. 評価実験方法

学生 6 人に協力してもらい，A と B の 2 つのグループに分け，それぞれに問題を 4 つ提示する．1，2 問はインターフェースの情報を提示した問題とする．また，3，4 問はシステムの要求のみを提示した問題とする. 提示する問題は，それぞれ正常系の働きのみを行うように機能を制限した組込システムで，平均的な状態数は 6，遷移数が 10 程度の問題にした．これらの問題を A グループは 1，3 問目を UML で，2，4 問目は提案方式で設計を行ってもらう．B グループは反対に，1，3 問目を提案方式で，2，4 問目は UML で設計を行ってもらう(表 1 参照).

表 1. 実験手順

問題番号.問題内容	UML	提案方式
1.券売機(プロトコル提示)	Aグループ	Bグループ
2.洗濯機(プロトコル提示)	Bグループ	Aグループ
3.炊飯器(要求のみ)	Aグループ	Bグループ
4.ATM(要求のみ)	Bグループ	Aグループ

その後，各学生に各問題の設計を行った時の設計にかかった時間とシステムの状態遷移図を提出してもらい，アンケートなどにより本方式や支援ツールを使った設計に関する感想などを結果として出してもらう．6個の質問に対して1~5までの数字を入力してもらい，その数字を得点として集計した．この結果から，UMLによる設計と提案方式による設計の比較を行い，評価した．

なお, UMLによる設計では, Microsoft Office Visio 2003を利用した.

5.2. 結果

UMLと提案方式による評価実験の評価は，定量的なデータとして設計にかかった時間や設計図の状態数やイベント数，アンケートの集計結果を，定性的なデータとしてヒアリングによる感想や実際にできた設計図のデータ，アンケートの答案をもとに行った．設計時間に関しては，表2より各々の問題におけるUMLによる設計時間と提案方式による設計時間は, あまり変わらなかったと言える. また，各人のUMLによる1回目の設計時間の平均と提案方式による1回目の設計時間の平均では，UMLに比べ提案方式の設計時間の方が遅かったが，各人の2回目のUMLによる平均設計時間よりも，2回目の提案方式による平均設計時間の方が速かったことが表3より分かる．この結果をヒアリングの結果とあわせて考えると，提案方式の説明不足や支援システムの操作性の問題により，提案方式による1回目の設計は時間がかかったことがわかる．2回目の提案方式による設計は，提案方式と支援システムの使い方に慣れたため大幅な時間短縮ができている．各人が1回目にやった問題と2回目にやった問題が違うといった点などを考慮に入れても，かなり大幅な短縮ができていることがわかる．それに比べ，一般のUMLの描画ツールを使って設計したUMLによる設計では，1回目も2回目も時間の短縮はあまり見られず, 問題の違いによる誤差範囲とも言える.

表2.各問題の設計の平均時間

問題	UML	提案方式
券売機	65分	65分
洗濯機	77分	100分
炊飯器	62分	58分
ATM	66分	86分

表3.設計回数による設計時間

設計回数	UML	提案方式
1回目	78分	93分
2回目	71分	63分

表4は，各人の問題と設計方式毎の状態数と遷移数を集計したものであり，問題と設計方式ごとに，状態数と遷移数の最小，最大が表されている．表4より，UMLの方が提案方式よりも状態数が極端に多くなる場合があったことがわかる．これは，提案方式を用いた設計ではプロトコルの状態の組み合わせによりその状態を表すため，UMLでは別の状態として扱われる状態が，提案方式では同一と見られたためである. そのために, 提案方式では状態が減り，UMLでは状態が多く現れたと推測される．また，UMLとは違い，提案方式では各人の設計した状態遷移図の状態数，遷移数がほとんど同数になった．この結果は，問題の内容にプロトコルが提示されていようが，機能要求だけであろうが変わらなかった. 各人の状態遷移図を見てみると，提案方式では全員がよく似た状態遷移図になっていた．つまり，提案方式を用いればどんな設計者が設計してもほぼ同じ状態遷移図が作れるといえる.

表4.設計結果

	UMLによる設計				本方式による設計			
	状態数		遷移数		状態数		遷移数	
	最小	最大	最小	最大	最小	最大	最小	最大
問題1	3	3	4	5	3	4	5	6
問題2	7	20	8	21	6	6	10	10
問題3	4	5	11	18	4	5	13	14
問題4	5	6	9	13	5	6	11	13

表5はアンケートの問題1～6の集計結果である．質問1のアンケート結果はかなり票がわかれるような形になった．これは，プロトコルの入出力信号の順番やタイミングの記述が難しいという点と，ツールの操作性の問題が挙げられる．しかし，質問2のアンケート結果は，概ね好意的だったので支援システムの操作性が向上することで，作りやすさも向上するのではないかと考えられる．質問3のアンケート結果では全員から作りやすいという結果が得られた．これは支援システムが，プロトコルの情報を用いて状態や遷移の候補を自動的に作り出してくれ，設計者はそこから選択するだけで設計を進めることができるためであると言える．これにより，特にインターフェースの仕様が事前に与えられている場合は，本方式が効果的であることが確かめられた．質問4では，本方式の方が設計しやすいとの意見が多数得られた．これは，本方式ではインターフェースの情報を用いることで簡単に状態遷移図の設計が可能なためであると考えられる．質問5の設計方式別の状態遷移図の比較では，結果としてあまり変わらないという結果が得られた．これは今回のような規模の状態遷移図では，どちらの設計方式でも完成図に差がでないためであると考えられる．質問6では，もう一度本方式でやってみたいという人が多く見られた．これは，本方式の方がUMLによる設計と比べて，理解しやすかったためであると考えられる.

設計図で得られた情報では，提案方式による設計では，状態数や状態名，遷移の数などを参考にしたところ，ほとんどの人が類似した設計図を提出した．またこちらで用意した模範解答ともほぼ同じものができていた．しかしUMLに関しては，ほとんどの人が異なった状態名や状態数，遷移数の状態遷移図が提出された．ヒアリングの結果より，UMLの設計は設計者によって自由に書けてしまうために逆に記述が難しいといった意見や，必要な複数の状態を一つの状態にまとめてしまいがちであるという意見が得られた.

表 5.アンケート結果

番号	内容	5 点	4 点	3 点	2 点	1 点
1	プロトコルの状態遷移図は作りやすいか?	0 人	2 人	2 人	1 人	1 人
2	プロトコルの状態遷移図はわかりやすいか?	0 人	4 人	1 人	1 人	0 人
3	プロトコルの状態遷移図があった場合，システムの状態遷移図は作りやすいか?	4 人	2 人	0 人	0 人	0 人
4	本方式による設計とUMLによる設計とを比べて，本方式の方が設計しやすかったか?	0 人	4 人	2 人	0	0 人
5	本方式で設計した状態遷移図は，UMLで設計した状態遷移図に比べて分かりやすいか?	0 人	3 人	2 人	1 人	0 人
6	次に状態遷移図を作る機会があったら，本方式を使っても良いと思う?	0 人	4 人	2 人	0 人	0 人

提案方式に関しては，プロトコルの記述が大変であるという意見が得られた．しかし，実際のシステム開発においては外部装置とのインターフェース部分のプロトコルに関してはすでに定まっていることも多いため，一からの開発ということはないはずである．ほとんどの人からプロトコルの記述があると，状態遷移図を簡単にかけるといった意見や，他の人と大体同じ結果になるため意思の疎通が図りやすいといった意見が得られた．また，支援システムに関しては，系統的な状態遷移図の作成の機能はかなり好評だったが，それ以外の細かい操作部分に関しての不満が多かった．これらの結果より，プロトコル記述の簡易性の向上と支援システムの操作性の向上が今後の課題として挙げられる．

6. おわりに

リアルタイムシステム向けの設計技法である ROOM のプロトコルの概念を詳細化して，システムの状態遷移図を系統的に作成する方式とその支援システムについて述べた．プロトコルを詳細に表すために，インターフェースの入出力信号リストに加えて，信号の利用される順番やタイミングを規定したプロトコル状態遷移図を導入した．システムを表すカプセルは，外部とのやり取りの窓口として，いくつかのポートを持ち，各ポートはその型を表すプロトコルを持つ．これらポートのプロトコルの状態の組み合わせによって，システムの状態を表し，その状態でのプロトコルの状態より，利用可能なイベントやアクションを制限することで，システムの振る舞いを示す状態遷移図を系統的に設計できる．

この提案方式を実装した支援システムの開発を行った．システムの開発は microsoft visual studio .net 2003 で行った．支援システムでは，プロトコル状態遷移図の記述ができ，そのプロトコル状態遷移図を用いてシステムの状態遷移図の設計をガイドするようになっている．次に，この支援システムを利用して UML と提案方式による評価実験を行った．これにより提案方式とツールの有効性を確かめることができた．設計結果やアンケートの集計，ヒアリングの集計による考察により，従来の通常の UML による設計と比べて，提案方式による設計は，設計の自由度が少なく，かえって設計しやすいという結果が得られた．支援システム自体の操作性がまだよくないという意見も得られたが，カプセルの状態遷移図作成においてはかなりの効果が期待できることが分かった．

今後の課題として，現在の提案方式の記述には含めていない変数の処理や戻り値に関する表記の拡張があげられる．また，それらの記述に対応した支援システムの強化と，今回の実験で問題となった操作性の向上などが挙げられる．今回の評価実験では 6 人の学生の協力してもらったが，評価人数が少ないため十分有効な評価まではできなかったのも事実である．今後はより実際的なシステムを対象にして，本方式と支援システムの評価を進めていきたい．

参考文献

[1]OMG: UML 2.0 Infrastructure Specification, http://www.omg.org/, 2003

[2]OMG: UML 2.0 Superstructure Specification, http://www.omg.org/, 2004

[3]Bran selic and Jim Ranbaugh:Using UML for Modeling Complex Real-Time Systems,IBM report,1998.

[4]Bran selic et al.:REAL-TIME OBJECT-ORIENTED MODELING, John Wiliy and Sons, Inc.,1994.

[5] 大川敦， 紫合治 ： —33— ース情報を用いた状態遷移図の生成， 第６７回情報処理学会全国大会， 2005.

[6] スティーブ J・メラー他: Executable UML，翔泳社，2003

[7] I. Jacobson et.el. Object-Oriented Software Engineering: A Use Case Driven Approach. Addison Wesley, 1992

[8]紫合治：環境の仕様を用いたシナリオから状態マシンの生成, FOSE2004 pp173-176,2004

[9]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

[10]Microsoft: Microsoft Office Visio, http://www.microsoft.com/japan/office/visio/prodinfo/default.mspx

[11]IBM: Rational Rose, http://www-06.ibm.com/jp/software/rational/products/design/rosetech/