ソフトウェア工学 127－5
（2000．5．26）

リアクティブシステムの発展的構成法の提案

尾崎弘幸† 片山卓也†

近年，開発されるソフトウェアは大規模化，複雑化している．故に，段階を踏むことなくシステムを開発することは不可能である．システム開発における典型的なモデルの段階的記述法には，トップダウン的記述とボトムアップ的記述がある．しかし，どちらの記述法も，システム全体を見通すことと，システム全体の挙動を確認することを両立することが難しい．そこで，我々は，各段階間の関係を数学的に定義し，抽象的な振舞いから具体的な振舞いへの段階的な記述を行なう発展的構成法を提案する．本構成法はこの数学的な関係に基づいているため，段階の前後の整合性を保つことと，ある段階の振舞いを前段階へ抽象化することが可能である．このため，抽象度の異なるオブジェクト間の協調動作を，抽象度をそろえることで可能とする．また，複雑な振舞いをより単純な振舞いへ抽象化可能であるため，可読性が向上する．

An Evolutional Development Method for Reactive Systems

HIROYUKI OZAKI† and TAKUYA KATAMAYA†

Recently, software is becoming larger and more complex and it is important to develop systems by incremental development methodology. Mainly, there are two types of incremental description for modeling systems. One is called top-down description, the other bottom-up description. In top-down description, although it is possible to describe system behavior by looking overall system, system behavior cannot be executed until the last stage system is obtained. In bottom-up description, inversely, although individual system behavior can be executed at middle stage, system behavior cannot be executed as a whole, and it is difficult to describe system behavior by looking overall system. We define a relation between intermediate stages mathematically and then we propose an development method for evolving the behavior from abstract to concrete level incrementally. Since the method is based on the relation, it is possible to preserve consistency between stages and to execute the system behavior in its abstract level. It is possible to execute the system behavior as a whole by adjusting abstraction levels even if its individual components are not in the same level.

1. はじめに

近年，開発されるシステムは大規模化，複雑化している．大規模で，複雑なシステムの分析/設計にOMT[1]などのオブジェクト指向方法論が広く使われるようになってきている．分析/設計段階では，明確にされていないシステムの要求から明確な仕様を作ることが目的である．小規模で単純なシステムの開発ならば，段階を踏むことなく分析/ 設計を行なうことが可能かもしれないが，大規模で複雑なシステムの仕様を段階を踏まずに構成することは一般的に困難である．

段階的な記述方法は多々あるが，典型的な記述方法は，トップダウン的記述とボトムアップ的記述である．トップダウン的記述は全体を見通して記述するため，初期の分析/設計段階において，有効である．しかし，途中段階で動作させることが困難である．逆に，ボトムアップ的記述は部分的にでも動作可能であるので，実装段階において，有効である．しかし，全体を見通すことが困難であるため，明確な仕様を必要とする．

段階的な記述において，実際にモデルを動作させることは重要である．記述と動作の確認（テスト）のサイクルを短くすることで，早期に誤りを発見しやすい．実装段階だけでなく，分析/設計段階においても，同様に記述と実行のサイクルを短くしたい．しかし，分析/設計段階では，システム全体を見通しながら段階的に記述する必要がある．これまでの典型的な段階的記述方法では，この2つを両立することができず，結

† 北陸先端科学技術大学院大学
Japan Advanced Institute of Science and Technology

果的に設計者自身の経験や勘に頼っていることが多い．

高品質のシステムを構築するために，システムの振舞いを正確に記述する必要がある．システムの振舞いは仕様の一つであり，オブジェクト指向方法論では，システムの振舞いを記述するモデルとしてStatechart[2]が広く用いられている．Statechartは状態遷移図に，状態の階層性と並行性の拡張を行なったモデルである．

状態遷移図は，一般的な，プログラミング言語と異なり，計算機が意味を解釈できなくても，イベントを発生させることで，シンタックス的に動作させることが可能である．故に，トップダウン的記述を行なったとしても，個々のオブジェクトにおいては動作可能であるが，

(1) 段階の前後における振舞いの整合性が取りにくい
(2) 完成度の異なるオブジェクト間で協調動作が困難

という問題点がある．

そこで，本研究では，ある段階の状態遷移図を前段階に抽象化可能となる関係を数学的に定義する．この関係を保ちつつ，段階的に記述を行なうことで，

(1) 段階の前後における整合性を保つことが可能
(2) 抽象実行可能となり，完成度の異なるオブジェクト間の協調動作が可能
(3) 複雑な状態遷移図の抽象化が可能となり，可読性が向上

となる．本研究は，関数型言語によるソフトウェアの段階的構成法 (ISDR法[3]) を状態遷移図に応用した研究である．

本稿の構成は以下の通りである．まず，2章で，各段階の間の関係を数学的に定義するために，本稿で用いる状態遷移図を形式的に定義する．3章では，各段階の間の関係を数学的に定義する．4章では，各段階の間の関係を保ちながら段階的に状態遷移図を構成するための規則を定義する．4章で定義した規則の適用例を5章で述べる．6章では，抽象実行について述べる．

2. 準　　備

リアクティブシステムとは何らかの刺激を受けることで反応を返すシステムのことである．このようなシステムは計算機上のサーバー，組み込みシステム，銀行のATMなど広く用いられている．リアクティブシステムの振舞いを記述する際に用いられる状態遷移図は，大きく分けると，Mealy型とMoore型である．これらの表現能力は等価であるが，StatechartはMealy型状態遷移図を拡張したモデルである．故に，ここではMealy型状態遷移図を用いる．この状態遷移図の特徴はある状態において遷移可能なイベントを受けることで，イベントを出力し，次の状態に遷移するということである．ここで，受け取るイベントを**入力イベント**，出力するイベントを**出力イベント**と呼ぶ．

本稿では状態遷移図の非決定性を排除するためにλ遷移を排除する．非決定性は設計の段階においてかなり強力であるが，その設計を元に実装する場合，決定的になるように変換する必要がある．決定性有限オートマトンと非決定性有限オートマトンの表現能力が等価であることは周知であるが，設計者と実装者の間にズレが生じてしまう．そこで，本研究では，最終的に実装段階までを考慮して，決定性状態状態遷移図を用いる．

決定的に状態遷移を引き起こすためには，必ず入力イベントが必要である．しかしながら，出力イベントは必ずしも必要であるとは限らない．つまり，何もイベントを出力しない状態遷移が存在するかもしれない．そこで，何もイベントを出力しない(つまり，出力イベントが存在しない)ことを特別なイベントϵとする．以下に状態遷移図Mを形式的に定義する．

定義1　状態遷移図 M

$\mathbf{S}$を状態集合，$i \in \mathbf{S}$を初期状態，$\mathbf{E}$をイベント集合，$\mathbf{T} \subseteq \mathbf{S} \times \mathbf{E} \times \mathbf{E}_\epsilon \times \mathbf{S}$を遷移規則とする．この時，これらの4つ組 $(\mathbf{S}, i, \mathbf{E}, \mathbf{T})$ を**状態遷移図** M と定義する．ここで，$\mathbf{E}_\epsilon$ は $\mathbf{E} \cup \{\epsilon\}$ である．また，この状態遷移図 M は決定性であるため，任意の状態遷移 $(s_1, e_1, \overline{e}_1, s_1')$，$(s_2, e_2, \overline{e}_2, s_2') \in \mathbf{T}$ について，

$$(s_1 = s_2 \wedge e_1 = e_2) \Rightarrow (s_1' = s_2' \wedge \overline{e}_1 = \overline{e}_2)$$

という条件が成立しなければならない．□

上で定義した状態遷移図の状態遷移は単一の入力イベントによって遷移する．この状態機械を外部から観測すると，入力イベントや出力イベントを時系列上で並べた列でのみ振舞いを観測可能である．ここで，イベントを時系列で並べた列のことを**イベント列**と呼ぶ．イベント集合$\mathbf{E}$から考えられる全てのイベント列の集合を$\mathbf{E}^+$とし，以下に定義する．

定義2　イベント列集合 $\mathbf{E}^+$

$\mathbf{E}$をイベント集合とする．また，$\cdot$を$\mathbf{E}$上の連結演算子とする．イベント集合$\mathbf{E}$と連結演算子$\cdot$によって生成される全てのイベント列全体の集合を$\mathbf{E}^+$と定義する．またイベント集合$\mathbf{E}_\epsilon$と連結演算子$\cdot$によって生成される全てのイベント列全体の集合を$\mathbf{E}_\epsilon^+$と定義する．ここで，$e \in \mathbf{E}$について$\epsilon \cdot e = e \cdot \epsilon = e$とする．そして，誤解が生じない限り，イベント列の連結演算子$\cdot$を省略する．また，イベントeが0回以上

繰り返し出現する時，e^* と記述する．

□

状態遷移はイベント列を用いることで，連結可能となる．つまり，状態遷移 $(s_1, e_1, \bar{e}_1, s_1')$，$(s_2, e_2, \bar{e}_2, s_2') \in \mathbf{T}$ はイベント列を用いることで，$(s_1, e_1e_2, \bar{e}_1\bar{e}_2, s_2')$ と記述できる．このようなイベント列を用いた遷移規則 $\mathbf{T}^+ \subseteq \mathbf{S} \times \mathbf{E}^+ \times \mathbf{E}_\epsilon^+ \times \mathbf{S}$ を以下に定義する．

定義 3 イベント列による遷移規則 $\mathbf{T}^+$

任意の $s_1, s_1', s_2, s_2' \in \mathbf{S}$，$e_1, e_2 \in \mathbf{E}^+$，$\bar{e}_1, \bar{e}_2 \in \mathbf{E}_\epsilon^+$ について，

(1) $\forall (s_1, e_1, \bar{e}_1, s_1') \in \mathbf{T}.(s_2, e_2, \bar{e}_2, s_2') \in \mathbf{T}^+$

(2) $(s_1, e_1, \bar{e}_1, s_1'), (s_2, e_2, \bar{e}_2, s_2') \in \mathbf{T}.$
$(s_1' = s_2) \Rightarrow (s_1, e_1e_2, \bar{e}_1\bar{e}_2, s_2')$

と定義する．

□

直感的に，イベント列を用いた遷移規則 $\mathbf{T}^+$ は任意の状態から到達可能な状態への状態遷移全体の集合である．

3. 発展的構成法

本発展的構成法は，抽象的な振舞いからより具体的な振舞いへ段階的に記述を進めることで，ある段階の振舞いを前段階の振舞いとして抽象化可能となる構成法である．

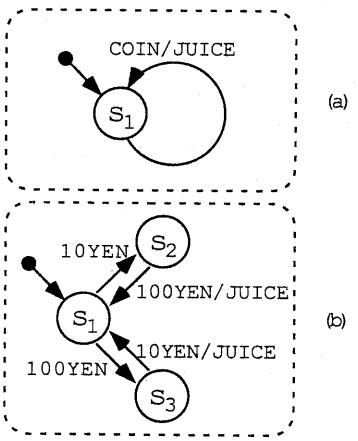

図 1 自動販売機の例

図 1 の (a) は，

a1) 初期状態で，コインを入れるとジュースを出して，初期状態へ戻る

という自動販売機，(b) は，

b1) 初期状態で，10 円を入れて，100 円を入れるとジュースを出して，初期状態へ戻る

b2) 初期状態で，100 円を入れて，10 円を入れるとジュースを出して，初期状態へ戻る

という自動販売機の振舞いを表わしている．ここで，

(1) 「コインを入れる」ということが「10 円を入れる」か「100 円を入れる」ということである

(2) b1) と b2) の抽象的な意味が a1) で，かつ，a1) の意味に b1) と b2) 以外の意味が含まれていない

とすると，(a) と (b) は，抽象度は異なるが，振舞いの意味的には同等である．故に抽象化が可能となる．常に抽象化が可能となるように状態遷移図を段階的に構成することで，システム全体としての振舞いの意味が，抽象度は異なるが，どの段階においても同等となる．本構成法は，状態遷移図を意味的には同等であるが，より具体的にすることを繰り返すことで，最終的な状態遷移図を得る手法である．

状態遷移図が前の段階に抽象化可能であるためには，

(1) 抽象化不可能な状態遷移の存在してはならない

(2) 全ての状態遷移を抽象化することで，前の段階の全ての状態遷移に抽象化される

という条件が必要となる．状態遷移図 M_2 が状態遷移図 M_1 に抽象化可能である時，M_2 は M_1 と**発展関係**にあると呼ぶ．

抽象的な振舞いを，矛盾なく欠かさず具体的な振舞いで表わすことを**詳細化**と呼ぶ．オブジェクトの内部としては，状態遷移が振舞いを表わし，外界から観察した場合は，イベント列の入出力が振舞いを表わす．詳細化された振舞いは，抽象度は異なるが，詳細化前の振舞いと意味的に同等である．つまり，(a) から (b) を得た時，(b) が b1) か b2) の振舞いをするということは (a) が a1) の振舞いをするということを詳細化したことになる．また，b1) や b2) は a1) の振舞いの一つであり，a1) から b1) や b2) を得ることを**具体化**と呼ぶ．

振舞いを外界から観測できるのはイベントのみである．外界から観察して振舞いを構成するものはイベントのみであり，振舞いを詳細化するためにはイベントが具体的にならなくてはならない．「10 円入れる」や「100 円入れる」というイベントの意味は「コインを入れる」というイベントの意味の一部である．イベント COIN から 10YEN や 100YEN を切り出すことをイベントの**具象化**と呼ぶ．

3.1 イベントの具象化

イベントの具象化とは，抽象的な意味を持つイベントの一部の意味をイベントとして具体的にすることである．イベント A をイベント B に具象化した時，イベント A を抽象的なイベント，イベント B を具体的なイ

ベントと呼ぶ．

ここで，イベント e_2 がイベント e_1 の具象化であることを，

$$e_1 \leftarrow_e e_2$$

と記述し，これを**具象化関係**と呼ぶ．つまり，図1の例では，COIN $\leftarrow_e$ 100YEN や COIN $\leftarrow_e$ 10YEN と記述する．

イベントの具象化において，抽象的なイベントと具体的なイベントの間に意味の矛盾が生じてはならない．イベント ϵ は，'何もイベントを出力しない'という意味の特別なイベントである．たとえ抽象的であったとしても，出力されるイベントをイベント ϵ に具象化することや，イベント ϵ を具象化することで出力するイベントを得ることは意味的に矛盾が生じる．故に，このような具象化を行なうことはできない．また，抽象的なイベントをより抽象的なイベントに具象化するは意味的に間違っている．例えば，MONEY(お金)を COIN(コイン) と NOTE(お札) に具象化し，COIN を MONEY に具象化することである．

以下にイベントの具象化関係 $\leftarrow_e \subseteq \mathbf{E}_\epsilon \times \mathbf{E}_\epsilon$ を定義する．

定義 4　イベントの具象化関係 $\leftarrow_e$

(1)　イベント集合 $\mathbf{E}_e$ の任意の要素 e について，$e \leftarrow_e e$ である．

(2)　イベント集合 $\mathbf{E}$ の任意の要素 e_1, e_2 について，e_2 が e_1 の具象化であるならば，$e_1 \leftarrow_e e_2$ である．

(3)　イベント集合 $\mathbf{E}_\epsilon$ の任意の要素 e_1, e_2, e_3 について，$e_1 \leftarrow_e e_2$ かつ $e_2 \leftarrow_e e_3$ ならば，$e_1 \leftarrow_e e_3$ である．

(4)　イベント集合 $\mathbf{E}$ の任意の要素 e_1, e_2 について，$e_1 \leftarrow_e e_2$ かつ $e_2 \leftarrow_e e_1$ ならば，$e_1 = e_2$ である．　□

3.2　イベントの具体化

イベントの具体化とは状態遷移図を外界から観測した時の，抽象的なイベントの振舞いの一つをより具体的にすることである．状態遷移図は外界からイベントを受け取ることで，状態が変化する．抽象的なイベントの振舞いの一つをより具体的にするということは，抽象度は異なるが，状態の変化に関して，同等でなければならない．

図1の例では，イベント COIN を受け取ることによる状態の変化とイベント列 100YEN · 10YEN を受け取ることによる状態の変化は同等である．イベント列を受け取ることによる状態の変化の度合いを状態の**変化量**と呼び，イベント列 e による状態の変化量を $|e|$ と表わす．また，$|e_1 e_2| = |e_1| + |e_2|$ と定義する．つまり，|COIN| = |100YEN · 10YEN| であるが，|COIN| ≠ |100YEN| である．

イベントの具体化とは状態の変化量が同等であるように，抽象的なイベントの振舞いを具体的なイベントを用いて切り出すことである．イベント列 100YEN·10YEN はイベント COIN の具体化であるが，これを

$$\text{COIN} \hookleftarrow_e \text{100YEN} \cdot \text{10YEN}$$

と記述し，これをイベントの**具体化関係**と呼ぶ．以下にイベントの具体化関係 $\hookleftarrow_e \subseteq \mathbf{E}_\epsilon \times \mathbf{E}_\epsilon^+$ を定義する．

定義 5　イベントの具体化関係 $\hookleftarrow_e$

$\mathbf{E}_\epsilon$ をイベント集合，$\mathbf{E}_\epsilon^+$ をイベント列集合とする．

(1)　任意の $e \in \mathbf{E}_\epsilon$ について $e \hookleftarrow_e e$ である．

(2)　任意の $e \in \mathbf{E}_\epsilon$，$e_1 e_2 \cdots e_n \in \mathbf{E}_\epsilon^+$ について，$|e| = |e_1 e_2 \cdots e_n|$ かつ，$e \leftarrow e_i (1 \leq i \leq n)$ ならば，$e \hookleftarrow_e e_1 e_2 \cdots e_n$ である．

と定義する．　□

3.3　イベントの詳細化

イベントの詳細化とは，状態遷移図を外界から観測した場合に，抽象的なイベントの振舞いを矛盾なく欠かさず具体的な振舞いで表わすことである．つまり，抽象的なイベントの具体化されたイベント列を全て記述し，それらのイベント列の振舞いが抽象的なイベントの振舞いを全て表わしているとき，全ての具体化されたイベント列は抽象的なイベントの詳細化である．

図1の例で,「コインを入れる」ことと「110円入れる」ということが意味的に同じで,「コイン」の具象化が「10円」と「100円」しか存在しないとするならば，COIN を 100YEN · 10YEN と 10YEN · 100YEN の2つを具体化することが詳細化することである．つまり，イベント e をイベント e の具体化であるイベント列の集合にすることがイベントの詳細化である．この集合を ES とした時，ES がイベント e の詳細化であることを，

$$e \prec_e ES$$

と記述し，イベントの**詳細化関係**と呼ぶ．上の自動販売機の例では

$$\text{COIN} \prec_e \{\text{100YEN} \cdot \text{10YEN}, \text{10YEN} \cdot \text{100YEN}\}$$

となる．以下にイベントの詳細化関係 $\prec_e \subseteq \mathbf{E}_\epsilon \times \mathcal{P}(\mathbf{E}_\epsilon^+)$ を定義する．

定義 6　イベントの詳細化関係 $\prec_e$

$\mathbf{E}_\epsilon$ をイベント集合，$\mathbf{E}_\epsilon^+$ をイベント列集合とする．任意のイベント $e \in \mathbf{E}_\epsilon$，任意のイベント列集合 $ES \in \mathcal{P}(\mathbf{E}_\epsilon^+)$ について，任意のイベント列 $es \in ES$ について，$e \hookleftarrow_e es$ かつ，任意のイベント列 $\bar{e} \in \mathbf{E}_\epsilon^+$ について，$\bar{e} \notin ES$ ならば，$e \not\hookleftarrow_e \bar{e}$ である時，$e \prec_e ES$

であると定義する．□

3.4 状態遷移の具体化

イベントの具体化を用いて，状態遷移をより具体的な状態遷移にすることを状態遷移の具体化と呼ぶ．ここで，状態遷移 t_2 が状態遷移 t_1 の具体化である時，

$$t_1 \hookleftarrow_t t_2$$

と記述し，これを状態遷移の具体化関係と呼ぶ．

状態遷移 t_1 と t_2 が具体化関係にあるためには，入出力イベントの具体化関係ではなく，それぞれの事前状態，事後状態に関して関係を定義する必要がある．この関係を状態の具象化関係と呼び，

$$s_1 \hookleftarrow_s s_2$$

と記述する．以下にまず状態の具象化関係 $\hookleftarrow_s \subseteq \mathbf{S} \times \mathbf{S}$ を定義し，次に状態遷移の具体化関係 $\hookleftarrow_t \subseteq \mathbf{T} \times \mathbf{T}^+$ を定義する．

定義 7　状態の具象化関係 $\hookleftarrow_s$

$\mathbf{S}$, $\mathbf{S}'$ を状態集合とする．任意の状態 $s \in \mathbf{S}$，任意の状態 $s' \in \mathbf{S}'$ について，同じ状態であるならば，$s \hookleftarrow_s s'$ であると定義する．□

定義 8　状態遷移の具体化関係 $\hookleftarrow_t$

$\mathbf{T}$ を遷移規則，$\mathbf{T}^+$ をイベント列による遷移規則とする．$(s_1, e, \overline{e}, s_1') \in \mathbf{T}$, $(s_2, es, \overline{es}, s_2') \in \mathbf{T}^+$ について，$e \hookleftarrow_e es$ かつ $\overline{e} \hookleftarrow_e \overline{es}$ かつ $s_1 \hookleftarrow_s s_2$ かつ $s_1' \hookleftarrow s_2'$ であるならば，$(s, e, \overline{e}, s') \hookleftarrow_t (s, es, \overline{es}, s')$ であると定義する．□

3.5 状態遷移の詳細化

イベントの詳細化を用いて状態遷移をより具体的な状態遷移集合にすることを状態遷移の詳細化と呼ぶ．ここで，状態遷移の集合 T が状態遷移 t の詳細化であることを，

$$t \prec_t T$$

と記述し，これを状態遷移の詳細化関係と呼ぶ．以下に状態遷移の詳細化関係 $\prec_t \subseteq \mathbf{T} \times \mathcal{P}(\mathbf{T})$ を定義する．

定義 9　状態遷移の詳細化関係 $\prec_t$

$\mathbf{T}$ を遷移規則，$\mathbf{T}^+$ をイベント列による遷移規則とする．任意の状態遷移集合 $T \subseteq \mathbf{T}^+$ について，

$$ES = \{es \mid (s, es, \overline{es}, s') \in T\}$$
$$\overline{ES} = \{\overline{es} \mid (s, es, \overline{es}, s') \in T\}$$
$$ST = \{s \mid (s, es, \overline{es}, s') \in T\}$$
$$ST' = \{s' \mid (s, es, \overline{es}, s') \in T\}$$

とする．ここで，$(s, e, \overline{e}, s') \in \mathbf{T}$ について，$e \prec_e ES$ かつ $\overline{e} \prec_e \overline{ES}$ かつ任意の状態 $st \in ST$ について $s \hookleftarrow_s st$ かつ任意の状態 $st' \in ST$ について $s' \hookleftarrow_s st'$ であるならば，$(s, e, \overline{e}, s') \prec_t T$ であると定義する．□

3.6 状態遷移図の発展関係

状態遷移図 M_1，M_2 において，M_2 が M_1 と発展関係にあるためには，

(1)　M_1 のどの状態遷移にも詳細化された状態遷移が M_2 に含まれている

(2)　M_2 のどの状態遷移も M_1 の状態遷移を詳細化した状態遷移の一部である

という条件を満たさなければならない．なぜならば，(1) では M_2 が M_1 の全ての振舞いの意味を持っていることを保証し，(2) では M_2 が M_1 にはない振舞いを追加していないということを保証するからである．この発展関係を形式的に定義するために必要となる「状態遷移 t は状態遷移 t' の一部である」という関係 (含有関係) をまず先に定義し，次に発展関係を定義する．

$\mathbf{T}$ を遷移規則とする．$\mathbf{T}^+$ をイベント列による遷移規則とする．状態遷移 $(s_1, e_1, \overline{e_1}, s_2) \in \mathbf{T}$ は $(s_1, e_1e_2, \overline{e_1e_2}, s_3) \in \mathbf{T}^+$ の一部である．この時，

$$(s_1, e_1, \overline{e_1}, s_2) \lhd (s_1, e_1e_2, \overline{e_1e_2}, s_3)$$

と表わす．以下に状態遷移の含有関係 $\lhd \subseteq \mathbf{T} \times \mathbf{T}^+$ を定義する．

定義 10　状態遷移の含有関係 $\lhd$

$\mathbf{T}$ を遷移規則，$\mathbf{T}^+$ をイベント列による遷移規則とする．

(1)　任意の状態遷移 $t \in \mathbf{T}$ について，$t \lhd t$ である．

(2)　任意の状態遷移 $(s, e, \overline{e}, s') \in \mathbf{T}$，任意の状態遷移 $(s_1, e_1, \overline{e_1}, s_2), (s_2, e_2, \overline{e_2}, s_3) \in \mathbf{T}^+$ について，$(s, e, \overline{e}, s') \lhd (s_1, e_1, \overline{e_1}, s_2)$ ならば，$(s, e, \overline{e}, s') \lhd (s_1, e_1e_2, \overline{e_1e_2}, s_3)$ である．

(3)　任意の状態遷移 $(s, e, \overline{e}, s') \in \mathbf{T}$，任意の状態遷移 $(s_1, e_1, \overline{e_1}, s_2), (s_2, e_2, \overline{e_2}, s_3) \in \mathbf{T}^+$ について，$(s, e, \overline{e}, s') \lhd (s_2, e_2, \overline{e_2}, s_3)$ ならば，$(s, e, \overline{e}, s') \lhd (s_1, e_1e_2, \overline{e_1e_2}, s_3)$ である．□

定義 11　状態機械の発展関係 $\sqsubseteq$

状態遷移図 M_1，M_2 において，M_2 が M_1 と発展関係にあるとは以下の 2 つの条件を満たす場合であると定義し，$M_1 \sqsubseteq M_2$ と記述する．

(1)　任意の状態遷移 $t \in \mathbf{T}_1$ について，$t \prec_t Tr$ となる $Tr \subseteq \mathbf{T}^+$ が存在する．

(2)　任意の状態遷移 $t \in \mathbf{T}_2$ について，$t \lhd tr$ となる $tr \in \mathbf{T}_2^+$ が存在し，かつ，$t' \hookleftarrow_t tr$ となる状態遷移 $t' \in \mathbf{T}_1$ が存在する．□

4. 詳細化規則

段階的な記述の中で，どこが次の段階なのかを決定することは困難である．発展関係にあるオブジェクトは抽象実行が可能であるが，ある意味連続的な段階的記述においては，必ずしも抽象実行が可能ではない．オブジェクトが常に抽象実行可能であるためには，あ

る段階の状態遷移図 M_2 について，常に $M_1 \sqsubseteq M_2$ となる M_1 が存在していなければならない．つまり，どの段階においても常に抽象実行可能であるためには，アトミックな操作で状態遷移図 M_1 を M_2 に発展させる必要がある．故に，このアトミックな操作を規則として定義する．この規則を**詳細化規則**と呼ぶ．詳細化規則を定義する前に，定義に用いる補助関数 $rw_t : \mathbf{T}\times\mathbf{S}\times\mathbf{S} \to \mathbf{T}$ と $rw_T : \mathcal{P}(\mathbf{T})\times\mathbf{S}\times\mathbf{S} \to \mathcal{P}(\mathbf{T})$ を定義する．補助関数 rw_t は状態遷移の状態名の書き換えのために用いる関数であり，補助関数 rw_T は状態遷移集合の要素の状態名を書き換えるための関数である．

定義 12　補助関数 rw_t

$\mathbf{S}$ を状態集合，$\mathbf{T}$ を遷移規則とする．任意の状態 $s, s' \in \mathbf{S}$，任意の状態遷移 $t \in \mathbf{T}$ について，状態遷移の状態名を書き換える補助関数 rw_t を

$$rw_t(t,s,s') = \begin{cases} (s',e,\overline{e},st) & \text{if } t = (s,e,\overline{e},st) \\ (st,e,\overline{e},s') & \text{if } t = (st,e,\overline{e},s) \\ (s',e,\overline{e},s') & \text{if } t = (s,e,\overline{e},s) \\ t & \text{otherwise} \end{cases}$$

と定義する．□

定義 13　補助関数 rw_T

$\mathbf{S}$ を状態集合，$\mathbf{T}$ を遷移規則とする．任意の状態 $s, s' \in \mathbf{S}$，任意の状態遷移集合 $TR \subseteq \mathbf{T}$ について，状態遷移集合の要素の状態名を書き換える補助関数 rw_T を

$$rw_T(TR,s,s') = \{rw_t(t,s,s') \mid t \in TR\}$$

と定義する．□

以下に，段階を進めるための詳細化規則 R1 と R2 を定義する．詳細化規則は

$$(\text{一般形})\ \frac{(M,\prec_t)}{(M',\prec'_t)}\ [c]$$

という形で定義する．これは，条件 c を満たす時，M を M' に書き換え，$prec_t$ を $prec'_t$ に書き換える規則である．

定義 14　詳細化規則 1

$$(\text{R1:})\ \frac{(M,\prec_t)}{(M',\prec'_t)}\ [c]$$

ここで，

$M = (\mathbf{S} \cup \{s_1, s_2\}, i, \mathbf{E}, \mathbf{T} \cup \{t\})$,
(ただし，$s_1 = s_2$ でもよい)
$t = (s_1, e, \overline{e}, s_2)$,
$M' = (\mathbf{S} \cup \{s_1, s_2, s_3\}, i, \mathbf{E}, \mathbf{T} \cup \{t_1, t_2\})$,
$t_1 = (s_1, e_1, \overline{e_1}, s_3)$,
$t_2 = (s_3, e_2, \overline{e_2}, s_2)$,
(ただし，e_1 が繰り返しならば，$s_1 = s_3$ であり，
e_2 が繰り返しならば，$s_2 = s_3$ となる)
$\prec'_t = \prec_t \cup \{(t, \{(s_1, e_1e_2, \overline{e_1e_2}, s_3)\})\}$
$c = s_3 \notin \mathbf{S} \wedge (s_1, e_1), (s_3, e_2) \notin \mathbf{A}$
$\mathbf{A} = \{(st, ev) \mid (st, ev, \overline{ev}, st') \in \mathbf{T} \cup \{t\}\}$
と定義する．□

定義 15　詳細化規則 2

$$(\text{R2:})\ \frac{(M,\prec_t)}{(M',\prec_t)}\ [c]$$

ここで，
$M = (\mathbf{S} \cup \{s_1, s_2\}, i, \mathbf{E}, \mathbf{T} \cup \{t\})$
(ただし，$s_1 = s_2$ でもよい)
$t = (s_1, e, \overline{e}, s_2)$
$M' = (\mathbf{S} \cup \{s_1, s_3, s_4\}, i, \mathbf{E}, \mathbf{T}' \cup \{t_1, t_2\})$
$t_1 = (s_1, e_1, \overline{e_1}, s_3)$
$t_2 = (s_1, e_2, \overline{e_2}, s_4)$
$\mathbf{T}' = \mathbf{T} - \mathbf{T}^- \cup \mathbf{T}_1^+ \cup \mathbf{T}_2^+$
$\mathbf{T}^- = \{(s_2, ev, \overline{ev}, st) \mid \forall ev, \overline{ev}, st.(s_2, ev, \overline{ev}, st) \in \mathbf{T}\}$
$\mathbf{T}_1^+ = \{(s_3, ev, \overline{ev}, st) \mid \forall ev, \overline{ev}, st.(s_2, ev, \overline{ev}, st) \in \mathbf{T}\}$
$\mathbf{T}_2^+ = \{(s_4, ev, \overline{ev}, st) \mid \forall ev, \overline{ev}, st.(s_2, ev, \overline{ev}, st) \in \mathbf{T}\}$
$\prec'_t = \{(rw_t(tr), rw_T(TR)) \mid (tr, TR) \in \prec_t\} \cup \{(t, \{t_1, t_2\})\}$
$c = s_3, s_4 \notin \mathbf{S} \wedge (s_1, e_1), (s_1, e_2) \notin \mathbf{A}$
$\mathbf{A} = \{(st, ev) \mid (st, ev, \overline{ev}, st') \in \mathbf{T} \cup \{t\}\}$
と定義する．□

状態遷移図において，最も抽象的なモデルは，単一の状態遷移からなる状態遷移図である．また，初期状態から到達不可能な状態が存在しない状態遷移図である必要がある．

単一の状態遷移からなる遷移規則は 2 種類存在し，事前状態と事後先状態が同一である状態遷移と，異なる状態遷移である．前者の状態遷移図はリアクティブシステムのように永遠に動作し続ける振舞いの最も抽象的な状態遷移図であり，後者は UNIX のコマンドのように入力を受け取ることで出力を出し，終了する振舞いの最も抽象的な状態遷移図である．本発展的構成法は，これらのモデルのどちらかを最も抽象的なモデルとするかの選択から始まる．

定義 16　最も抽象的な状態遷移図

最も抽象的な状態遷移図は単一の状態遷移からなる状態遷移図で，かつ，初期状態から到達不可能な状態の

ない状態遷移図であると定義する．すなわち，
$(\{s_1, s_2\}, s_1, \{e_{in}, e_{out}\}, \{(s_1, e_{in}, e_{out}, s_2)\})$
$(\{s_1\}, s_1, \{e_{in}, e_{out}\}, \{(s_1, e_{in}, e_{out}, s_1)\})$
のどちらかであると定義する．． □

5. 例　　題

ここでは，詳細化規則を用いて，銀行の ATM の振舞いを発展的に記述する例を挙げる．詳細化規則はイベントを 2 つのイベントに具象化することしかできない．つまり，イベントを 5 つのイベントに具象化する場合は，4 回詳細化規則を適用する必要がある．本発展的構成法の流れは以下の通りである．

(1) まず，開発するシステムの形 (リアクティブシステムなのかそうでないのか) から最も抽象的な状態遷移図を選択する．
(2) イベントを 2 つのイベントに具象化する．
(3) イベントの振舞いを具象化されたイベントを用いて詳細化する．
(4) イベントの詳細化に基づき，適切な詳細化規則を選択し，適用することで状態遷移図を発展させる．
(5) (2) から (4) を最終的な状態遷移図が得られるまで，繰り返す．

段階 0: 銀行の ATM は何らかの入力を受け取る (**input**) と，お札を出す (**notes**) リアクティブシステムである．そこで，銀行の ATM の最も抽象的な振舞いを図 2 とする (1).

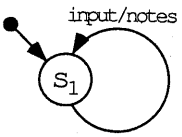

図 2 段階 0 の ATM

段階 1: 銀行の ATM への入力を具体的にする．詳細化規則では，一度にイベントを 2 つにしか具象化できない．よって，銀行の ATM への入力は「キャッシュカードを入力する」ことと「残りの入力をする」ことであるとし，イベント **input** を **card_in**(キャッシュカードを入力) と **other_in**$_1$(残りの入力) に具象化する (2). そして，イベント **input** による振舞いを {**card_in** · **other_in**$_1$} に詳細化する (3). そして，詳細化規則 R1 を用いて，$(s_1, \mathbf{input}, \mathbf{notes}, s_1)$ を $\{(s_1, \mathbf{card\_in} \cdot \mathbf{other\_in}_1, \mathbf{notes}, s_1)\}$ に詳細化し，図 2 の状態遷移図を図 3 に発展させる (4).

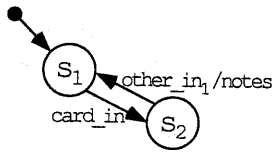

図 3 段階 1 の ATM

段階 2: 次に，イベント **other_in**$_1$ に着目する．銀行の ATM への入力のうち，残りの入力は「暗証番号の入力をする」ことと「さらに残りの入力をする」ことであるとし，イベント **other_in**$_1$ を **passwd**(暗証番号入力) と **other_in**$_2$(さらに残りを入力) に具象化する (2). この具象化を用いて，イベント **other_in**$_1$ を $\{\mathbf{passwd} \cdot \mathbf{other\_in}_2\}$ に詳細化する (3). そして，詳細化規則 R1 を用いて，$(s_2, \mathbf{other\_in}_1, \mathbf{notes}, s_1)$ を $\{(s_2, \mathbf{passwd} \cdot \mathbf{other\_in}_2, \mathbf{notes}, s_1)\}$ に詳細化し，図 3 の状態遷移図を図 4 に発展させる (4).

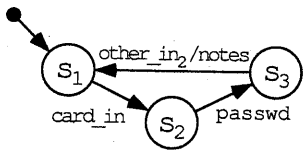

図 4 段階 2 の ATM

段階 3: そして，イベント **other_in**$_2$ に着目する．さらに残りの入力を「金額を入力する」ことと「いままでの入力が正しいかどうかの確認を入力する」こととし，イベント **other_in**$_2$ を **amount**(金額入力) と **confirm**(確認入力) に具象化する (2). この具象化を用いて，イベント **other_in**$_2$ を {**amount**·**confirm**} に詳細化する (3). そして，詳細化規則 R1 を用いて，$(s_3, \mathbf{other\_in}_2, \mathbf{notes}, s_1)$ を $(s_3, \mathbf{amount} \cdot \mathbf{confirm}, \mathbf{notes}, s_1)$ に詳細化し，図 4 の状態遷移を図 5 に発展させる (4).

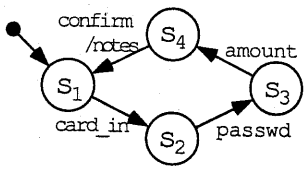

図 5 段階 3 の ATM

段階 4: さらに，イベント **confirm** に着目する．今までの入力が正しいかどうかの入力を「正しい

と入力する」ことと「正しくないと入力する」こととし，イベント confirm を ok と ng に具象化する (2)．イベント confirm を $\{\mathtt{ng}^* \cdot \mathtt{ok}\}$ に詳細化する (3)．そして，詳細化規則 R1 を用いて，$(s_4, \mathtt{confirm}, \mathtt{notes}, s_1)$ を $\{(s_4, \mathtt{ng}^* \cdot \mathtt{ok}, \mathtt{notes}, s_1)\}$ に詳細化し，図 5 を図 6 に発展させる (4)．

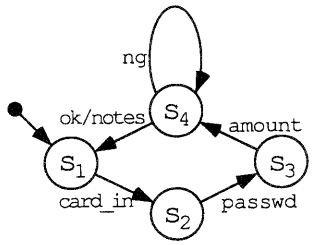

図 6　段階 4 の ATM

6. 抽象実行

オブジェクト間の通信はイベントを用いることで行なわれる．通信が成立するためにはイベントの識別子が同じでなければならない．図 7 はオブジェクト O_1 とオブジェクト O_2 が同一のイベントを用いているため，通信可能である．

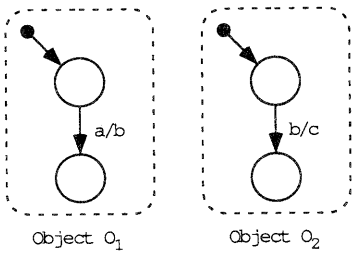

図 7　通信可能な例

しかし，オブジェクト O_2 のイベント b を詳細化することで，オブジェクトがより具体的になり，O_2' となると，オブジェクト O_1 とは通信が不可能となる (図 8)．

このように，sender と receiver の間で通信不可能なケースは以下の 2 つが考えられる．

(1)　sender が receiver よりも具体的すぎる．

(2)　receiver が sender よりも具体的すぎる．

この問題は具体的すぎるオブジェクトの抽象度をあげることで解決できる．つまり，図 8 の場合，オブジェクト O_2' を図 7 のオブジェクト O_2 に戻すことでオブジェクト O_1 と通信可能となる．このことから，システムを構成するオブジェクトの抽象度を全てそろえることで，抽象的ではあるが，システム全体を実行可能となる．

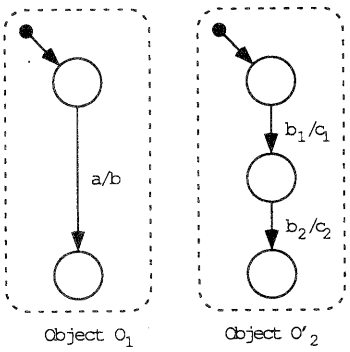

図 8　通信不可能な例

7. おわりに

本稿では，抽象的な状態遷移図から具体的な状態遷移図への発展的な構成法を提案した．この構成法は，数学的に定義された発展関係を保ちながら，段階的に振舞いを記述する方法である．発展関係が存在することで，ある段階の振舞いを前段階へ抽象化可能である．故に，段階の前後の振舞いの整合性を保証できる．また，抽象度の異なるオブジェクト間の通信を抽象実行メカニズムを用いることで可能となる．よって，抽象的ではあるが，システム全体の実行が可能となることを示した．

今後は，この詳細化規則がどのくらいの記述能力があるのか，ある程度，規模の大きなソフトウェアが記述できるのかについて研究を進めていく予定である．

参考文献

1) J.Rumbaugh, M.Blaha, M.Premerlani, F.Eddy and W.Lorensen: Object-Oriented Modeling and Design, Prentice Hall, 1991.

2) D.Harel, A.Pnueli, J.P.Schmid and R.Sherman: On the Formal Semantics of Statecharts, Proc of 2nd IEEE Symposium on Logic in Computer Science, 54-64, 1987.

3) Nobukazu Yoshioka, Masato Suzuki, Takuya Katayama: Incremental Software Development Method based on Abstract Interpretation, Proceedings of IWSSD-10, IEEE, 1998.