ソフトウェア工学 103－19
（1995．3．10）

ライフサイクル図を用いたオブジェクトの協調的な振舞いの設計

牧野 憲義　酒井 博敬
中央大学 理工学部 管理工学科

概要

本論文は，オブジェクトの協調的な振舞いに関する概念設計手法について述べている．システム中に存在するオブジェクトは，他の複数のオブジェクトと連絡を取り合い，協調しながら目的を達成する．この集団的な振舞いは，個々のオブジェクトの振舞いと共に振舞い設計プロセスにおける重要な問題である．本稿では，振舞い設計の代表的なアプローチであるライフサイクル図を基にした協調表現，およびオブジェクト間の関連を考慮したオブジェクトの責務の配分法を提案する．これらによって，システムの複雑な状態の厳密で明確な表現および分析プロセスやカプセル化を反映した協調設計を実現できると考える．

キーワード：オブジェクト指向開発，ライフサイクル図，振舞い設計，オブジェクトの協調，概念設計．

A Method for Collaborating Objects Behaviors Modeling by Life Cycle Diagrams

Noriyoshi Makino　and　Hirotaka Sakai
Department of Industrial and Systems Engineering,
Faculty of Science and Engineering, Chuo University

Abstract

This paper discusses a conceptual design of object behavior. Object can not behave alone in any system. Object achieves its tasks by communicating and collaborating with many other objects. A behavior of object is concerned with its construction, so an effective method for designing not only behaviors of each object but dynamic interactions of many objects behaviors is needed. We propose a model of collaboration among objects with a diagrammatic representation technique. In particular, we introduce a collaboration diagram of object behavior using a concept of object life cycle which is a conventional expression of behavior. Furthermore, we describe a method of collaboration modeling and delegation or distribution of responsibilities of objects.

key words: object-oriented software development, life cycle diagrams, behavior modeling, collaboration of objects, conceptual design.

1 はじめに

オブジェクト指向システム開発プロセスの中心的課題の一つに，オブジェクトの振舞い（behavior）に関する問題がある．振舞いは，オブジェクトの機能，反応，実行の動的な側面を説明する概念であり，オブジェクトの構造とも密接に関係している．動的な変化を伴うオブジェクト間の関連（relationship）もまた，振舞いに表現される．一般にシステムの機能はシステム中に存在するオブジェクトの集団的な振舞いによって実現される．オブジェクトは，他の複数のオブジェクトと連絡を取り合い，協調しながら目的を達成する．したがって，個々のオブジェクトの振舞いと共に，その間の相互作用を組織的に開発する方法が求められる．またその開発プロセスには，システムが提供する機能を協調するオブジェクトによって表現，管理することが要求される．これまでのところ，個々のオブジェクトの振舞いに関する議論はなされているものの，それらを統合するモデルやシステムの振舞いをオブジェクトの振舞いに変換する手法は，まだ十分確立されていない．

本論文は，オブジェクトの協調的な振舞いに関する概念設計手法として，以下の項目について述べている．

(1) 協調するオブジェクトの様子を明確に表すために，システムの振舞いをオブジェクトの振舞い（ここではライフサイクル）を用いてモデル化する．モデルの表記にはペトリネット [5] を用いる．この種の既存モデル [2] [4] では，ペトリネットを単なる表記上のモデルとしてのみ拡張して使用しているため，グラフの持つ動的な性質を保持しておらず，状態遷移のトレース等に問題があり，厳密さを欠いていた．本モデルはその点を改善しており，ライフサイクル図を保持したまま，協調的な振舞いにおける競合性や並行性を正確に表現する．

(2) 協調はシステムに与えられた機能ごとに発生する．そこで，これらの機能を実現する一連の協調を設計する手段として，"代表パートナーによるリング型協調" を提案する．この方法では，特に分析プロセスにおけるオブジェクトの関連の分類を設計プロセスに反映させる目的で，オブジェクトの関連の種類を利用する．これによって無秩序な設計による関連を減らし，ラビオリコードの発生を防ぐ．ここでは特に，協調をオブジェクト間の直接通信として設計する場合，すなわちオブジェクト間の関連が属性（attribute）として実現される場合を示す．

(3) オブジェクトの振舞いをより明確に表現するために，クラスの振舞いの表記を与える．

(4) 一般にライフサイクル図は，各々が異なる抽象度によって構築されているため，モデル間の相互作用の設計にそのまま利用することが難しいとの指摘がある．そこで，ライフサイクル図間の抽象度の差異の問題を考慮しながら，モデルを洗練する方法について述べる．

論文の構成は，まず2章で本稿に必要な用語と概念を整理し，3章でモデルの形式化と説明を行う．さらに4章では協調設計プロセスとして，協調方法の設計とモデルの洗練方法について述べる．

2 特性に関する整理

オブジェクト間の関連は，その特徴や種類によって，分類される．本稿では特に，集約，グループ化，役割および一般関連（association）を積極的に設計に用いる．そのとき集約，グループ化，役割に対しては，以下の性質または制約を仮定している．

1）集約（aggregation）

いくつかのオブジェクトを集めてこれを成分とする高位のオブジェクトをつくることを集約化という．構成要素となるオブジェクトを成分オブジェクト，成分から構成されたオブジェクトを集約オブジェクトという．またすべての成分を備えた状態を集約オブジェクトの完全体という．

e がオブジェクトの集合 O の集約であるとき，(1) $\forall o \in O$ は e の属性として実現される．(2) e は，空でない $O' \subseteq O$ に存在依存する．(3) e のアイデンティティは O に依存しない．オブジェクトのアイデンティティはそれを与える特性集合によって得られるものとする．

特に，成分を値に持つある属性 p について，その属性値が単純でなく（multi-valued property），単一クラス G のオブジェクトの部分集合であるとき，e を G のグループ化（grouping）という．

2）役割（role）

役割は，"システム中，一つのデータは一カ所に記録する" という考えに立ち，実世界をシステム中に実現するための分析手段である．現在，役割関連を直接支援するプログラミング言語はないので，汎化／特化関連の一種として分析されることが多いが，本稿では役割関連を一つの特別な関連として扱う．いま，オブジェクト b がオブジェクト a の役割として定義されているとき，(1) a は b の属性として実現される．役割にあるオブジェクトを属性によって直接参照することでデータの一貫性を維持する．(2) b は a に存在依存する．(3) a は b に存在独立である．

さらに集約，グループ化，役割および基本データ型

特性（数，文字，記号，配列等を値とする属性）の総称として内部特性という用語を使う．我々は，内部特性の有する諸性質がオブジェクトにとって特に重要であると考え，それらはオブジェクトの外部から隠蔽されるべきものであるとした．また一般関連は，属性として実現される上記以外の関連を指すものとする．

3　振舞いモデル

オブジェクトの振舞いを記述するには，事象に反応してオブジェクトが何を行うのかを明確に述べなければならない．本稿では振舞いを状態とアクションによって表現する．アクションは，実世界の出来事や信号に対して，オブジェクトがどう反応するかを抽象化したものである．また状態は，アクションが適用される際の条件，すなわち特性値の集合を表す．あるアクション a，b に順序がある時，a の発火後から b の発火までの間を a の事後状態，または b の事前状態という．

3．1　オブジェクトの振舞い

すべてのオブジェクトが最終的にクラスとして実現されることから，クラスごとにオブジェクトの振舞いをそのライフサイクル（ＬＣ）としてモデル化する方法は効果的であり，それをペトリネットで表現する研究も，いくつかの例が報告されている [1] [4]．モデル化のためのＬＣスキーマ（life cycle schema）およびその洗練規則（refinement rules）は，酒井 [1] で示されているものを用いることにするが，ＬＣ図を簡略化するため，ＬＣスキーマにおいては以下の制限を与える．

P，T をそれぞれオブジェクトの状態，アクションの有限集合とし，P，T の要素を点，有向辺集合 A を辺とする２部グラフを $B=(P\cup T, A)$ とする．また $|P|$ は P の要素数，$u\in P\cup T$ に対して，

$$pre\ (u)=\{v \mid v\in P\cup T, (v,u)\in A\},$$
$$post\ (u)=\{v \mid v\in P\cup T, (u,v)\in A\}$$

とする．ここで，(u, v) は点 u から点 v への有向辺を表す．

このとき，$\forall t\in T$ に対して，$|pre\ (t)|=1$ かつ $|post\ (t)|=1$ であるような B を振舞いグラフ（ＬＣスキーマ）という．

クラス ＥのＬＣにＬＣスキーマを適用してペトリネットグラフで表現したものがＥのＬＣ図である．状態をプレースに，アクションをトランジションに割り当てる．特に，$|P|=1$ を満たす振舞いグラフを最抽象度振舞いグラフという．

図１は，クラス「ホテル客」のＬＣ図の例である．initial，final句はそれぞれ生成，消滅可能な状態を表現する．生成，消滅アクションを表現する場合は，状態 "not exist" 等を用いてモデル化する．

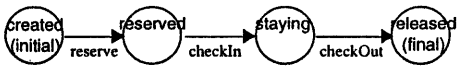


図１：「ホテル客」オブジェクトのLC図

3．2　協調するオブジェクトの振舞い

ある時点 t におけるシステムの状態は，その中で振舞うすべてのオブジェクトの t における状態の直積によって得られる．したがってシステムの状態表現はこのことを反映していなければならない．そこでシステムの振舞いをオブジェクトのＬＣを組み合わせることで表現する．これをオブジェクトの協調グラフといい，次のように定義する．

P，T をそれぞれオブジェクトの状態，アクションの有限集合とし，P，T の要素を点，有向辺集合 A を辺とする２部グラフを $\beta=(P\cup T, A)$，クラス E_i（$i=1,\cdots,n$）の振舞いグラフを $B_i=(P_i\cup T_i, A_i)$ とする．またすべての P_i と素な状態の有限集合を J とし，その要素をトリガー状態と呼ぶ．

このとき，$P=(P_1\cup\cdots\cup P_n)\cup J$，$T=T_1\cup\cdots\cup T_n$，$A=(A_1\cup\cdots\cup A_n)\cup D$，$D\subset(T\times J)\cup(J\times T)$ および $\forall u\in J$ に対して，$|pre\ (u)|=|post\ (u)|=1$，$pre\ (u)\subseteq T_i$，$post\ (u)\subseteq T_k$，$i\neq k$ であるような β を協調グラフという．

クラス $E_1,\cdots,E_n$ のＬＣ図を用いて協調グラフをペトリネット上に表現したものが，クラス $E_1,\cdots,E_n$ による振舞いの協調図である．

システム中で振る舞うオブジェクト間の協調は，それらの間のメッセージパッシングとして現れる．そこで，この様子をオブジェクトの持つ状態とは別の状態（トリガー状態）を置くことで表現する．図２はクラスＡとクラスＢによる振舞いの協調図の例である．協調図においてトランジションを発火させるものは，他のオブジェクトで起こったアクションか，あるいはシステム外部からの入力である．トリガー状態は，"あるクラスのオブジェクトが，別のあるクラスのオブジェクトへメッセージを発した状態" を表現する．これによってペトリネットとしての性質を失うことなく，オブジェクトの協調をモデル化できる．各々のＬＣ図は，協調図のサブネットとなる．

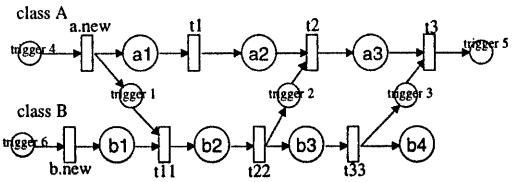
この図では，システムの振舞いをトークンによってトレースすることができる[1]．トリガー状態からの入

[1]ライフサイクル図上のトークンはインスタンスオブジェクトと見なして分析できるが，トリガー状態上のトークンの意味はこれと異なる．またすべてのトークンはアイデンティティを持つ．

力有向辺をもつトランジションの発火は，トリガー状態プレース上にトークンがあるときのみ発火する．これにより，あらゆる事象に対してシステムが全体として適切な反応を行うかどうかを検証できる．

本モデルは，ライフサイクル図をそのまま利用するため，構築や理解が容易であり，かつ厳密な設計が可能である．ライフサイクル図間の矛盾の検出と調整，個々のライフサイクル図のさらなる洗練を可能にする．また協調図では，トリガー状態に関する一連のアクション群がスクリプト [1] を表現する．スクリプト (script) とは，サブシステムの各機能ごとに，それを実現するためにパートナーがどのように協調的に振る舞うべきかを記述したものをいう．ここでサブシステムは，業務環境において与えられた責務を果たすために協調するオブジェクトの集まりをいう．

協調図を構築する目的は，異なるクラスのオブジェクト間の関連を詳細にモデル化することにある[2]．



I (ts1) = |a.new| .　　O (ts1) = |t11| .
I (ts2) = |t22| .　　O (ts2) = |t2| .
I (ts3) = |t33| .　　O (ts3) = |t3| .
I (ts5) = |t3| .　　O (ts4) = |a.new| .
ts : trigger　　O (ts6) = |b.new| .

＊Iは拡張入力関数，Oは拡張出力関数を表す[5].

図2：振舞いの協調図

4　構築の手順と方法

4.1　協調の設計

オブジェクトの振舞いを決定する手段として，あるオブジェクトが協調するパートナーや外部にもたらす機能を，そのオブジェクトの責務 (responsibility) と捉え，それを配分することでオブジェクト間のコントラクト (contract) を設計しようとする考え方がある [3]．そこではサブシステムに与えられる責務を各オブジェクトにどのように配分するかに設計の焦点が絞られる．この問題を解く手段の一つに，"集約階層パスに沿った委託" [1] がある．この方法では，オブジェクトはその直接の集約成分に関してのみ責務を負うとして，サブシステムに含まれるオブジェクトに集約階層を想定し，それ以外の責務を順次，自分の成分に委託する．この方法は，実際に集約関連が存在する下では特に有効である．

一般にオブジェクト間の通信を直接行う方法は，集約 (グループ化)，役割，一般関連，汎化／特化および類型化がある．これらは振舞いのスクリプトごとに単一，あるいは互いに組み合わさって現れる．したがって，これらすべてについて有効な設計法が求められる．ここでは特に，集約，役割および一般関連における設計について述べる．

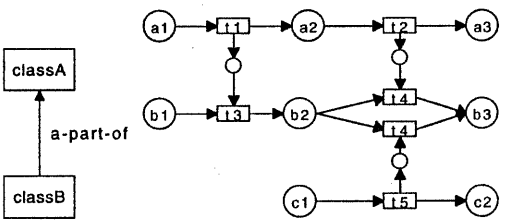


図3：集約階層と協調の例

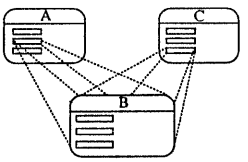


図4：属性による参照の様子

4.1.1　内部特性の設計

インスタンスオブジェクトと直接通信する一般的な方法は，そのオブジェクトを属性とすることである．

いま，一つの例を考える．クラスA，B，Cに属するオブジェクトがそれぞれ次のライフサイクルを持ち，またAをBの集約として分析したとする (図3左)．

A : $a1 \rightarrow a2/t1 \rightarrow a3/t2$.
B : $b1 \rightarrow b2/t3 \rightarrow b3/t4$.
C : $c1 \rightarrow c2/t5$.

便宜上，オブジェクトの状態遷移を

"*クラス名：事前状態→事後状態／アクション名*"

と表した．

協調の一つに，$t2$, $t5$がそれぞれ$t4$を発火することがあるとする．これをそのまま設計すると図3右のようになる．しかしこの協調図は，bがaとcの両方の属性として設計されるべきことを示している．しかし，成分オブジェクトへの参照を自由に許すと，集約オブジェクトは自分の属性に対して外部からの変更に無防備になる (図4)．またこのような設計は，成分となるオブジェクトクラス内部の変更をも引き起こす可能性を持つ．これはカプセル化の概念に反する．したがって，そのすべてを無秩序に属性とするのは効果的では

[2] したがって，同一クラスに属するオブジェクト間の相互作用は，うまく表現できないことがある (例えば，互いに重なり合うウィンドウオブジェクトの様子)．

ない.

そこで我々はカプセル化を実現するために，このようなオブジェクトに対する一般関連を否定することにした.以下に内部特性の設計に関する制約を定義する.

＜内部特性制約＞

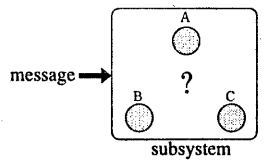
クラスEのオブジェクトeがクラスFのオブジェクトfの内部特性に定義されるとする．このときeは任意のオブジェクトの一般関連に含まれてはならない[3].

この制約により内部特性への参照は，参照を許されたオブジェクトのみに限られるため，すべてのメッセージパッシングはそれらを経由して実現される．したがって，これを設計するための手続きは，"集約階層パスに沿った委託" の原理と同じになる．そこで内部特性に関する直接成分への委託を，ここでは" 関連階層パスに沿った逐次委託" として次のように拡張して利用する．

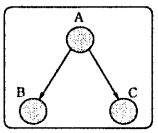
関連が集約のときは，" 集約階層パスに沿った委託" を行う．また関連が役割のときは，役割階層に沿って委託する．このときのパスは集約とは逆にたどる．すなわち役割関連にあるオブジェクトは役割階層の直接のスーパクラスにのみその責務を負うとして，順次その他の責務をスーパクラスへ委託していく．

4．1．2　一般関連の設計

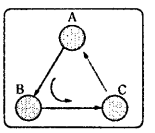
一般に，オブジェクトの協調を設計する作業は，最終的にスター型（図5ｂ）か，リング型（図5ｃ）かの選択になる．図5は最も単純な場合の例である．オブジェクトA，B，Cを含むサブシステムが，A，B，Cすべての振舞いを要求するような責務を負うとき，開発者にはそれをどのオブジェクトに最初に送り，その後パートナーへどのようにメッセージを送るかの問題が生じる．



(a)：A，B，Cを含むサブシステム



(b)：スター型



(c)：リング型

図5：協調方法の選択

スター型の協調では，あるオブジェクトが制御の中心になって複数のパートナーに直接メッセージを送る．この協調ではスターの中心となるオブジェクトの責務が突出して大きくなり易く，部品としての振舞いは期待できない．反面それ以外のオブジェクトの独立性は高い．一方，リング型では協調の始点になるオブジェクトからそのパートナーへ順次制御を渡していく．最後のパートナーが自分の責務を果たすと，自動的に始点のオブジェクトへ制御が戻る．この協調では，各オブジェクトはほぼ等しく責務を分担できる．しかし推移的な関連にあるオブジェクトが多くなり，リングが大きくなるにつれて個々の独立性は失われていく．

我々は，階層的な関連にないオブジェクト同士を対等であるとし，対等な立場にあるオブジェクトは，できるだけ等しく責務を負うべきであるとの考察のもとに，" 代表パートナーによるリング型協調" を考案した．これは，等しい責務の配分とカプセル化を実現する目的で，一般関連をリング型に，内部特性をスター型に設計する一つの方法である．以下にその手続きを示す．

(1) 対象とするスクリプトに関するオブジェクトの集合をWとする．またどのオブジェクトの内部特性にも属さないオブジェクトの集合をW_1，$W_2 = W - W_1$として，W_1を部分集合w_j（$j = 1, \cdots, n \mid n \leq |W|$）に分割する．

(2) 各w_jの要素から代表パートナーを選ぶ．代表パートナーは，サブシステムやスクリプトにおいて一番始めにメッセージを受け取るオブジェクトである．

(3) W_1の代表パートナーを始点にして各w_jの代表パートナー同士を順に結ぶ．

(4) 各w_jをW_1としてオブジェクトがなくなるまで(1)～(3)を繰り返す．

(5) 最後にW_2の要素を" 関連階層パスに沿った逐次委託" に従って協調させる．

(1)において，スクリプトを分割する手続きがあるが，これはオブジェクトモデルを考慮して行う．

リング型を利用した協調設計の簡単な例として，サブシステム「ホテル予約」におけるスクリプト「チェックイン」の設計をあげる．スクリプトの仕様は酒井 [1] にならって，協調に加わるオブジェクトの状態遷移によって記述することにする．

スクリプト「チェックイン」の仕様は図6の通り．このスクリプトの代表パートナーは，クラス *Customer*（客）のオブジェクトである．また，クラス *Hotel*（ホテル）のオブジェクトはクラス *CustomerData*（顧客データ）とクラス *HReservation*（ホテル用予約）オブジェクトの集約として，クラス *Agent*（旅行代理店）の

[3] eは任意のオブジェクトの内部特性に定義されてよく，またeの一般関連は任意である．

オブジェクトはクラス *AReservation*（代理店用予約）オブジェクトの集約として与えられるとする（図7）．

図8に結果例を示す．まずどのオブジェクトの内部特性にも属さないオブジェクトの集合を求める．|*Customer, Hotel, Agent*| が選ばれる．そこでこの3つをリング型に協調させる．この例ではオブジェクトの数が少ないため，特にスクリプトの分割は行わずに進める．ここでは代表パートナー *Customer* から，*Hotel, Agent* の順に協調するよう設計した．トリガー状態（*ts1*）からトリガー状態（*ts2*）へと制御が流れていき，最後は *Customer* に戻る．次に各集約オブジェクトに対して，その直接成分の振舞いに関する責務を負わせ，集約階層パスに沿って逐次委託を行う（*ts3*～*ts5*）．

script "check In"
Customer : exist → exist/ arriveHotel.
Hotel : exist → exist/ cArrive → exist/ checkIn.
Agent : exist → exist/ receive$.
CustomerData : not-exist → exist/ new.
HReservation : created → finalState/ consume.
AReservation : created → finalState/ consume.

図6：スクリプト「チェックイン」

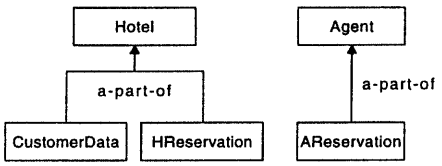


図7：集約階層図

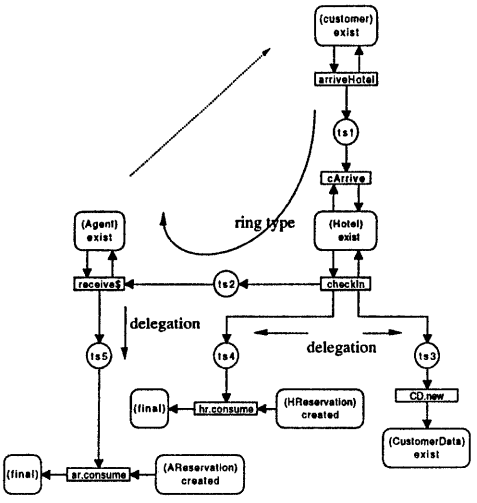


図8：" 代表パートナーによるリング型協調" の様子

4．2　協調図の洗練方法

協調図を利用した振舞い設計の手順として，協調図を洗練するための幾つかの方法を示す．

4．2．1　最抽象度グラフによる構築

ペトリネットグラフはその構造上複雑化しやすい．そこで始めは最抽象度グラフを用いて協調図を構築する．最抽象度グラフによる協調の設計後，各ＬＣ図を洗練したものに順次置き換えていくことでオブジェクト間の協調を設計できる．ＬＣ図はさまざまな抽象度で構築されている．この方法によって各モデルの抽象度も統一される．

4．2．2　クラスの振舞い分析

クラスオブジェクトの働きを明確にすることは，重要である．ＬＣ図は，インスタンスオブジェクトのＬＣをクラス単位で表しているにすぎない．クラスオブジェクトとそのインスタンスオブジェクトは異なるオブジェクトであり，したがってその振舞いも異なる．またクラスの振舞いは，けっしてそのインスタンスを生成するだけではない．今までのところ，クラスオブジェクトとインスタンスオブジェクトとの関連表現を明確にサポートしている手法はない．両者の振舞いを明確に表現できる記述が必要である．そこで，クラスの振舞いをその表記法と共に以下に与える．

1）状態

クラスには最抽象度グラフのみを与える．クラスとインスタンスの関連を表現するとはいえ，複雑化しやすいペトリネット上で，ＬＣ図の数を2倍にするのは現実的でない．そこでグラフ上ではインスタンスのＬＣ図（最抽象度グラフ）に重ねて表現する．ただしインスタンスの状態が洗練されたとしてもクラスの状態は洗練されない．またトークンを用いる場合，常に一つ存在することになるが表示しない．

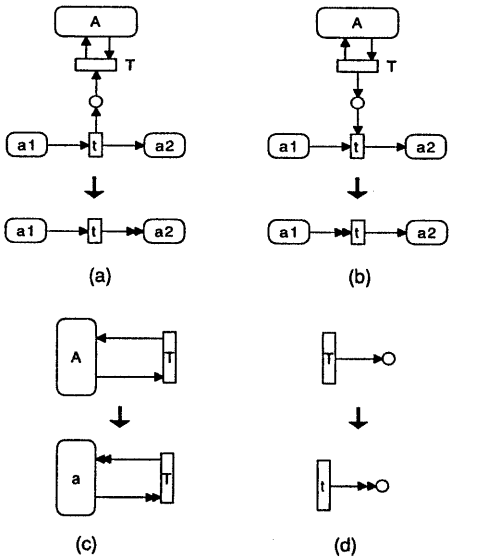
2）アクション

クラス動作に関わるアクションは，有向辺を二重にすることで表現する．クラスアクションとして次の4つの記述を定義する．

(1) インスタンスオブジェクトから，それが属するクラスオブジェクトへのメッセージパッシングをともなうアクションでは，トランジションの出力有向辺を二重にする（図9 a）．
(2) クラスオブジェクトからそのインスタンスオブジェクトへのメッセージパッシングをともなうアクションでは，トランジションの入力有向辺を二重にする（図9 b）．
(3) クラスオブジェクトのみ関わるアクションは，トランジションに対する有向辺を入出力ともに二重

にする（図9 c）.

(4) クラスオブジェクトがそのインスタンス以外の対象にアクションを発生するときは，トリガー状態への出力有向辺を二重にする（図9 d）.



＊大文字の状態名を持つグラフはクラスのライフサイクルを，小文字の状態名を持つグラフはインスタンスのライフサイクルを，最も小さいブレースはトリガー状態を表す.

図9：クラスの振舞い表現

以上の表記は省略表現であり，グラフの拡張ではない．したがって，ペトリネットの構造に対する情報，および動的な性質は失われない．クラス記述は例えば以下のような手続きによって元の構造を導出できる.

（例） 図9 aの省略記法によるペトリネットの構造をCとして，形式表現すれば以下のようになる.（Pはペトリネット上におけるプレースの有限集合，Tはトランジションの有限集合，Iは入力関数，Oは出力関数を表す）.

$C=(P, T, I, O)$． $P=\{a1, a2\}$．

$T=\{t\}$． $I(t)=\{a1\}$． $O(t)=\{a2\}$．

二重の有向辺があることからクラスアクションがあることがわかり，$I(T)=\{A\}$，$O(T)=\{A\}$，の関数を加える（クラスは状態を一つしか持たない）．またP，Tにそれぞれ$\{A\}$，$\{T\}$が加わる．二重の有向辺が出力側にあることから，メッセージの方向がインスタンスからクラスであることが定まる．したがって，アクションtの出力関数，アクションTの入力関数およびPにそれぞれトリガー状態tsがなくてはならない．結局その構造Cに対して，

$P=\{a1, a2, A, p\}$． $T=\{t, T\}$．

$I(t)=\{a1\}$． $I(T)=\{A, ts\}$．

$O(t)=\{a2, ts\}$． $O(T)=\{A\}$．

を得る．これは，省略前のグラフの構造に一致する.

4．2．3 スキーマの洗練

最抽象度グラフによって構築された協調図は，さらに詳細な設計のために洗練される．以下に3つの洗練方法を示す.

1）全体を洗練する

各ＬＣ図の抽象度をある程度保ちつつ，協調図上のすべて（または部分）のＬＣ図を徐々に洗練していくことで協調図を洗練する方法である．この方法は単純でありながら，より詳細な全体像を表現する有効な手段である．規模のそれほど大きくないシステムにおいては，より厳密で，明確な振舞いモデルを提供できる.

2）一つのＬＣ図のみを洗練する

設計対象とするクラスを一つ選び，そのＬＣ図をそれより洗練されたものに置き換えることで協調図をＬＣ図ごとに洗練する．この方法は，図の発散を防ぎつつ協調図を洗練する手段となる．また協調図において，あるアクションの発火がオブジェクト内部のものとして設計されるならば，その遷移は縮約 [5] する[4]．図10は，図2の例において$t1$をオブジェクト内部の遷移とみなして縮約した時の様子を示す．協調に最小限必要な状態とアクションによって振舞いを表現することで，効果的な抽象度を構築する．縮約は，汎化やフレームワークの設計にも利用できる.

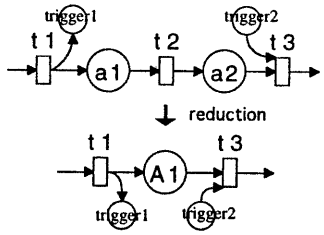


図10：ペトリネットの縮約（図2の例）

3）一つのＬＣを複数に洗練する

この方法は，データベース開発におけるいわゆる3層スキーマの考え方に基づいている．基本的な方針は2）と同様であるが，ここではＬＣ図を複数に洗練する．一般にＬＣの洗練作業は，分析したときの視点によって結果が異なる．そのオブジェクト本来の性質に基づいた洗練を行うことが理想であるが，このような洗練を行えることは少ない．また，ある責務を負って振る舞う協調図中で，このようなＬＣ図がうまく適合するとは限らない．そこで，次の2種類の洗練をＬＣ図に適用する.

[4]協調図においてアクションは自然発火しない.

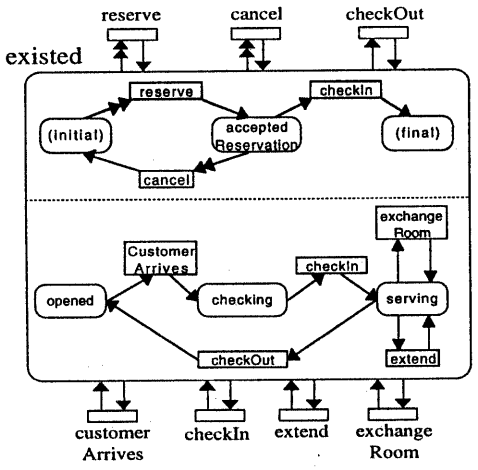


図11：「ホテル」オブジェクトの外部スキーマ洗練

概念スキーマ洗練：協調図構築前に求めた各オブジェクトのＬＣは，そのオブジェクト自体の性質に基づいている．これは，そのオブジェクトをシステム中に存在する実体として抽象化したときの考察に基づいたものであり，最も一般性の高い視点から導かれたものと考えることができる．そこで，このような洗練をＬＣの概念スキーマ洗練（Ｃ洗練）ということにする．オブジェクトのＬＣというときはＣ洗練によるものを指す．

外部スキーマ洗練：協調図に表現されるオブジェクトは，そのパートナーに対して何らかの責務を負う．通常，これらの責務すべてが，もとのＬＣ中に表現されているとは限らない．またその振舞いに新たな順序が見つかる場合も想定できる．実際，協調するパートナーが増えてくると，一つのＬＣでそのオブジェクトのすべてのパートナーに対する振舞いを表現するような理想的な洗練をすることはかなり難しい作業になる．Ｃ洗練自体何らかの一視点から行われているといえる．オブジェクトに機能や役割に対する何らかの見通しも持たないでそのＬＣを洗練することはできない．したがって，開発者がオブジェクトに対して異なった機能を頭の中に描くことで，全く異なったモデルを作り出すことになる．今，協調図によって協調すべきパートナーが決まるとすれば，パートナーそれぞれの視点からＬＣを洗練することが可能である．このようにある視点からＬＣを洗練することをＬＣの外部スキーマ洗練（Ｅ洗練）と呼ぶ．

複数のパートナーから責務を負うオブジェクトの協調を詳細に設計するために，Ｅ洗練を行うことで対応できる．洗練したＬＣは洗練前の図に重ねて表記する．図11にＥ洗練を適用した例を示す．「ホテル」オブジェクトのＬＣを「旅行代理店」オブジェクトから見た視点（上）と「ホテル客」オブジェクトから見た視点（下）によって，２通りに洗練した様子を最抽象度グラフに重ねて表している．協調の概略であるメッセージの送受信の様子と協調パートナーは，最抽象度グラフを用いた協調図によって示される．オブジェクト内部の状態遷移は，パートナーの視点ごとに示すことで，複数の遷移系列を容易に表現できる．またＬＣの一部分のみ複数に洗練することで対応できることもできる．この方法は，大規模なシステムにおいて適用できる．

5　おわりに

本稿では，オブジェクト指向システム開発プロセスにおけるオブジェクトの協調的な振舞いを設計する方法を提案した．ライフサイクルの組み合わせによる協調表現を用いて，オブジェクトの責務の配分法，およびライフサイクル図間の抽象度の差異を考慮した設計方法を示した．これらによって，協調的な振舞いをより厳密に設計でき，カプセル化を実現した再利用性の高いシステム構築に貢献できると考える．また本稿で示した手法は，すべての抽象度に対して適用できると考えられる．すなわちサブシステムをひとつのオブジェクトとみなすことで，より大きなサブシステムの状態を複数のサブシステムの状態の統合によって表現するのである．このとき最も抽象的なモデルは，システムとその外部との相互作用を示す．

今回示した各設計法の確立とその評価，汎化階層・類型化における責務の配分，インスタンス間またはサブシステム間の協調表現などは，現在研究中の課題である．

参考文献

[1] Sakai H.,"A Method for Contract Design and Delegation in Object Behavior Modeling.",IEICE TRANSACTIONS on Information and Systems.", Vol.E76-D,No6, June 1993.

[2] 酒井博敬、堀内一,"オブジェクト指向入門",オーム社,1989.

[3] R.Wirfs-Brock et al.,"Designing Object-Oriented Software", Prentice Hall,1990.

[4] Gerti Kappel and Michael Schrefl,"A Behavior-Integrated Entity-Relationship Approach for the Design of Object-Oriented Databases", In C.Batini,ed.,Entity-Relationship Approach : A Bridge to the User : Proc.Seventh International Conference on Entity-Relationship Approach.Rome.North-Holland,1988.

[5] 村田忠夫,"ペトリネットの解析と応用",近代科学社,1992.