データベースシステム 94−17
（1993．7．22）

オブジェクトの協調の抽象化について

酒井博敬† 堀内 一††
†中央大学理工学部 ††（株）日立製作所ビジネスシステム開発センタ

オブジェクト指向による情報システムにおいて，サブシステムの業務はオブジェクトの協調によって実現される．ここではまず単純化したライフサイクルモデルおよびライフサイクル間制約によってオブジェクトの協調を表現し，業務の仕様であるスクリプトを体系的に導く．オブジェクトの再利用性は多様な業務環境に現われるオブジェクトの類似性を抽出することによって実現される．類似性の抽出においては，単一のクラスの性質のみでなく，サブシステムを構成するパートナー，パートナーの役割，およびパートナーの協調を対象として考察しなければならない．これはサブシステムが受けもつ業務の類似性を考えることにほかならない．抽象化されたサブシステムの例として資源受給サブシステムをとりあげ，ライフサイクル間制約を用いた抽象化スクリプトの表現について論じる．

On Abstraction of Object Collaboration

Hirotaka Sakai† Hajime Horiuchi††
†Chuo University ††Hitachi, Ltd.

In object-oriented design of information systems, functions of a subsystem are realized by collaboration of participating objects called partners. Using the life cycle model and the notion of inter-life cycle constraints, we model collaboration of objects, and systematically derive specifications of functions called scripts. The basis for reusability of objects is to extract abstract characteristics of similar objects that appear in various applications. In extracting similarities, we should analyze, not only characteristics of individual classes, but also partners that constitute the subsystem, their roles, and their collaboration. That is to analyze similarities of functions of subsystems. We discuss a representation of abstract scripts using the inter-life constraints in the resource providing subsystem example.

1．はじめに

抽象サブシステム（またはフレームワーク）はオブジェクトの分析，設計，ソフトウエアの再利用を実現するための基本手段である．抽象サブシステムは多様なサブシステムを構成するオブジェクト，そこにおけるオブジェクトの役割，およびオブジェクトの協調の類似性を抽出し，仕様化することによって得られる．ここではオブジェクトのライフサイクルとライフサイクル間制約からオブジェクトの協調に関する仕様を導き，抽象化する方法を論じる．

2．ライフサイクルの表現

［オブジェクトのライフサイクル］

実世界の実体がそうであるようにに，オブジェクトは与えられた環境において時間とともに変化する．すなわちオブジェクトはクラスのインスタンスとして生成され，消滅するまでの間，さまざまな状態遷移を遂げる．この生成，状態遷移，消滅の系列をオブジェクトのライフサイクルという[1, 2]．

状態 (state) はオブジェクトのもつすべての属性値の集合である．状態はオブジェクトがそのライフサイクルにおけるある時点に到達した段階に対応付けて解釈できるので，段階を表す名前を状態名として用いる．すなわち状態名はある段階におけるオブジェクトの属性値の集合に対して付けられた名称である．この属性値の集合を状態値とよぶ．状態値は一般に時間とともに変化するが，オブジェクトのアイデンティティは時間不変である．

オブジェクトには固有のアクションが定義されているが，その内容はオブジェクトの属性に対する一連の操作からなる．この操作によって状態値そのものは変化しないものと，状態値が変化するものとがある．前者を問合せアクション，後者を更新アクションとよぶ．ライフサイクルを考えるとき，アクションという言葉はとくに更新アクションに限定して用いる．アクションはオブジェクトにメッセージを送ることによって活性化される．オブジェクトがメッセージを受け取ることを事象 (event) の発生，あるいは単に事象とよぶ．またオブジェクト o（またはクラス O）に対してアクション t を活性化する事象を形式 [o:O t]（またはクラスに対しては [O t]）で表す．

アクション t の活性化はオブジェクトの状態を変える効果をもつ．すなわちオブジェクトがある状態 u にあるとき事象 [o:O t] が発生すると，オブジェクトはある状態 v に遷移する．もしオブジェクトが u 以外の状態にあれば，事象 [o:O t] は無効となり，アクションは活性化されない．u を t の事前状態 (pre_state)，v を t の事後状態 (post_state) という．アクション t の活性化に対して，オブジェクト o の属性値に関する事前条件および事後条件を付加することができる．1つのアクションは，オブジェクトを生成および消滅させるアクション **produce** と **consume** を除いて1つの事前状態と1つの事後状態をもつ．アクション t は状態 u にあるオブジェクトの性質とみなし得るという意味で，"t は u のアクションである"，あるいは "t は u にカプセル化される"という．状態 u からある状態への遷移は u にカプセル化されたアクションによってのみ可能である．

事前，事後状態という言葉は2つの状態の隣接関係を表わすのにも用いる．すなわちu, vをそれぞれ事前，事後状態とするアクションtがあるとき，u, vは隣接するといい，vをuの事後状態，uをvの事前状態という．オブジェクトoの状態の集合LCにおいて，LCが単一の状態からなるか，またはLCの任意の要素uがLCのある要素vと隣接関係にあるとき，LCをoのライフサイクルという．とくにLCが，オブジェクトが存在することを意味する単一の状態**exist**からなるとき，これを単純ライフサイクルという．同一クラスのどのオブジェクトも，定められたライフサイクルにしたがって状態遷移を遂げる．したがってライフサイクルはクラス対応に定義されたものと考えることができる．

［ライフサイクルの洗練］

状態をより細分された状態に分解することを状態の洗練という．状態の洗練は必然的にその状態のアクションの分解をともなう[3]．すなわち状態uが状態$u_1, ..., u_n$に洗練されると，uのアクションtは一般に$u_1, ..., u_n$のアクション$t_1, ..., t_n$に分解される．あるライフサイクルの状態を洗練することをライフサイクルの洗練という．単純ライフサイクルはいくつかの局所ライフサイクルに洗練される．

あるクラスにおいて，もはやこれ以上洗練されないライフサイクルを原子ライフサイクルという．またその状態およびアクションをそれぞれ原子状態，原子アクションとよぶ．原子アクションはクラスのオブジェクト固有の基本操作を表すものである．

ライサイクルを有向グラフで表現したものをライフサイクル図という．ライフサイクル図では状態を角の丸い長方形で表わし，アクションを事前状態から事後状態への有向辺で表わす．共通の事前，事後状態をもつ複数のアクションに対しては，共通の事前，事後状態を結ぶ複数の有向辺が存在する．

3．ライフサイクル間制約

単一のオブジェクトが状態uから状態vへ遷移するためには，u, vはライフサイクルによって定められた隣接関係になければならない．この意味でライフサイクルは単一オブジェクトの振舞いに関する制約を与えている．

オブジェクトは孤立して存在するのではなく，同じまたは異なるクラスのオブジェクトと協調して振舞う．協調するオブジェクトの振舞いに関する制約として次の3形式のライフサイクル間制約を定義する．各制約形式において，o:Oはクラス Oのオブジェクト oを表わし，state(o:O, u)はオブジェクトoが状態uにあることを表わす．state(o:O, u)を状態述語という[4, 5]．

［状態依存制約］

state(o:O, u) **requires** state(p:P, v);

" oが状態uにあれば，pは状態vにある．"

［順序制約］

state(o:O, u) **precedes** state(p:P, v);

" oの状態uへの遷移は，pの状態vへの遷移に先行する．"

［同期化制約］

state(o:O, u) **synchronizes with** state(p:P, v);

" o が状態 u にあることと，p が状態 v にあることとは同期する．"

状態依存制約は，状態を考慮に入れたオブジェクト間の存在依存に関する制約である．これに対して，順序制約と同期化制約は状態遷移の生起に関する制約である．

3つの制約において，状態 u あるいは v への遷移を引き起こすアクション t あるいは r を明示的に指定する必要があるときは，状態述語 state(o:O, u) あるいは state(p:P, v) の代わりに表現 state(o:O, u) **thru** t あるいは state(p:P,v) **thru** r を用いることにする．

4．業務のオブジェクト化

［集約オブジェクト］

集約化の概念についてはいくつかの定義がある[6, 7]．ここではライフサイクル間制約の概念を用いて集約オブジェクトを定義する．O, O_1, ..., O_n をクラス，LC を O のライフサイクルとする．オブジェクト o:O が LC のある状態 u およびオブジェクト o_i:O_i の（あるライフサイクルにおける）状態 u_i ($i = k, ..., m; \{k, ..., m\} \subset \{1, ..., n\}$) に対して，状態依存制約 state(o:O, u) **requires** state(o_i:O_i, u_i) にしたがうとき，o は u において o_k, ..., o_m を成分オブジェクトとする集約オブジェクトであるという．また u を成分状態 u_k, ..., u_m をもつ集約状態といい，state(o:O, u) = aggr(state(o_k:O_k, u_k), ..., state(o_m:O_m, u_m))，または簡単に $u = aggr(u_k, ..., u_m)$ と書く．

任意の o_i:O_i が o の少なくとも1つの状態における成分オブジェクトであるとき，o は o_1, ..., o_n を成分オブジェクトとする集約オブジェクトであるといい，$o = aggr(o_1, ..., o_n)$ と書く．

いくつかのオブジェクトがメッセージを送り合い，メッセージを受けたオブジェクトがそれに応じて振舞うとき，オブジェクト同士は協調するという．協調によって集約オブジェクトが生成される．業務は協調し合うオブジェクトによって生成される集約オブジェクトとしてモデル化される．

一般に，ある業務を実現するために協調するオブジェクトおよびその協調によって生ずる集約オブジェクトの集合をサブシステム，サブシステムの要素をパートナーとよぶ[8, 9]．

業務ルールはパートナーのライフサイクルおよびライフサイクル間制約として表現される．すなわち業務の進行にともなって業務状態の遷移が発生するが，これは業務を表わす集約オブジェクトのライフサイクルとして表わされる．また業務の状態遷移を引き起こすために必要な成分オブジェクトの協調は，集約オブジェクトと成分オブジェクト間の状態依存制約，および成分オブジェクト間の状態遷移に関する順序制約と同期化制約から導かれる．

たとえばレストランは客，ウエイトレス，コック，およびこれらの集約オブジェクトとしての料理サービスをパートナーにもつサブシステムとしてモデル化される．図1はレストランにおける各パートナーのライフサイクルを表わしたものである．点線の矢印が状態依存制約を表わす．

図1において料理サービスのルールはそのライフサイクルにおける状態遷移 手配中 → 引渡し済 → 勘定済 によって表現されている．

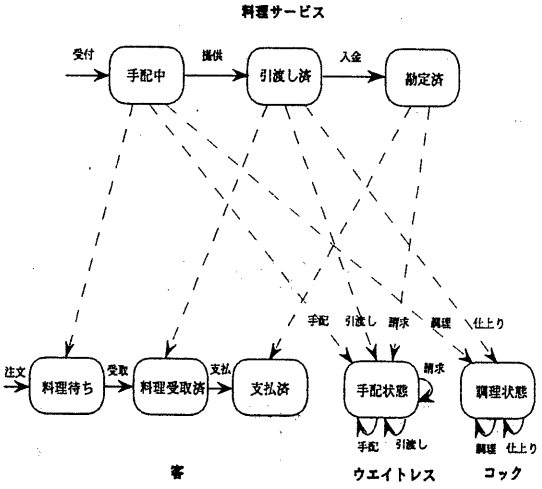


図1　レストランにおけるライフサイクル

また料理サービスの各状態が成分オブジェクトの状態に依存する様子は状態依存制約によって表わされている．次は集約状態 手配中 に関する状態依存制約，および客，ウエイトレス，コックの 状態間の順序制約と同期化制約の記述である．

state(s:料理サービス，手配中) **requires** state(g:客，料理待ち)；
state(s:料理サービス，手配中) **requires** state(w:ウエイトレス，手配状態) **thru** 手配；
state(s:料理サービス，手配中) **requires** state(c:コック，調理状態) **thru** 調理；
state(w:ウエイトレス，手配状態) **thru** 手配 **precedes** state(c:コック，調理状態) **thru** 調理；
state(g:客，料理受取済) **synchronizes with** state(c:コック，調理状態) **thru** 仕上り；
state(w:ウエイトレス，手配状態) **thru** 請求 **precedes** state(g:客，支払済)；

［サブシステムにおけるスクリプトの表現］

オブジェクトの協調はスクリプトとして仕様化される．スクリプトはサブシステムにおける業務対応に定義される．

業務を表す集約オブジェクトにおける集約状態 $u=aggr\ (u_1, .., u_n)$ は，関与するパートナーにおいて状態述語 $state\ (o_i:O_i, u_i)$ $(i=1, ..., n)$, $state(o:O, u)$ が成立することによって実現される．各状態への遷移の順序を表す状態遷移列 $w_1/ ... /w_k$ をスクリプトの状態表現という．状態遷移列は次のようにして定められる．

（1）各要素 w_i は，状態依存制約によって定まる集約状態の要素 $u, u_1, ..., u_n$，あるいは要素の組 $<u_i, u_j>$ のいずれかである．

（2）順序制約 $state(o_i:O_i, u_i)$ **precedes** $state(o_j:O_j, u_j)$ があれば，u_i を含む要素 w_f は u_j を含む要素 w_g の前に現われる．順序制約がなければ w_f, w_g の順序は任意である．

（3）同期化制約 $state(o_i:O_i, u_i)$ **synchronizes with** $state(o_j:O_j, u_j)$ に対して，u_i, u_j は状態の組 $w_h=<u_i, u_j>$ として現われる．

スクリプトは集約状態を実現するための仕様として，集約状態に対応付けて定義される．この意味でスクリプトの状態表現としての状態遷移列 $w_1/ ... /w_k$ は集約状態にカプセル化される．一方スクリプトはパートナーに分配されたアクションを活性化する事象列として表現することもできる．これをスクリプトの事象表現という．状態遷移列 $w_1/ ... /w_k$ を実現するスクリプトの事象表現は一意的ではない．状態遷移列

にしたがった遷移を引き起こす次のようないくつかの事象列の基本形を用意することによって，スクリプトを状態表現から事象表現へ容易に変換することができる．

（1）$w_1=u_1$, $w_2=u_2$ に対する w_1/w_2 を実現する事象表現

たとえば次のように分配されたアクションを定義する．ただし T_i' , T_j' はそれぞれオブジェクト $o_i:O_i$, $o_j:O_j$ において u_i, u_j への状態遷移を引き起こすアクションである．

（a）$o_i:O_i$, $o_j:O_j$ と異なるオブジェクト $o_k:O_k$ において次のアクションを定義する．

action T_k : $[o_i:O_i \quad T_i']$; $[o_j:O_j \quad T_j']$;

T_k のように，他のオブジェクトの状態遷移を引き起こすアクションを複合アクションとよぶ．

（b）オブジェクト $o_i:O_i$ のアクション T_i を次のように定義する．

action T_i : [**self** T_i']; $[o_j:O_j \quad T_j']$;

（c）オブジェクト $o_j:O_j$ のアクション T_j を次のように定義する．

action T_j : [**self** T_j']; $[o_i:O_i \quad T_i']$;

（2）$w_1=<u_i, u_j>$ を実現する事象表現

$o_i:O_i$, $o_j:O_j$ にそれぞれ次のアクション T_i, T_j を定義する．

action T_i : $[o_k:O_k \quad T_k]$;

action T_j : $[o_k:O_k \quad T_k]$;

ただし T_k は $o_i:O_i$, $o_j:O_j$ の状態遷移の同期化を調整するオブジェクト $o_k:O_k$ のアクションで，次の操作内容をもつ．すなわち $o_i:O_i$, $o_j:O_j$ がそれぞれ状態 u_i, u_j へ遷移可能ならばこの2つの遷移を引き起こす．$o_i:O_i$, $o_j:O_j$ の少なくとも一方が u_i, あるいは u_j へ遷移できない場合は，$o_i:O_i$ あるいは $o_j:O_j$ からの要求を待ち状態にしておき，$o_i:O_i$, $o_j:O_j$ がそれぞれ u_i, u_j へ遷移可能になったときこの2つの遷移を引き起こす．

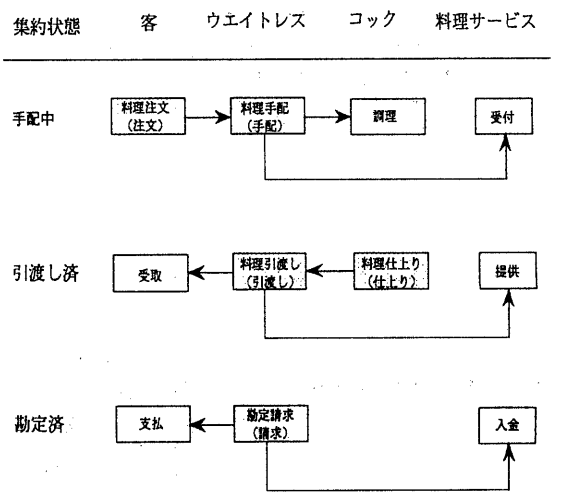


図2　レストランにおけるスクリプトの事象表現

次は料理サービスの3つの集約状態のスクリプトの状態表現である．

手配中　料理待ち／手配状態／調理状態／手配中；

引渡し済　手配状態／<料理受取済，調理状態>／引渡し済；

勘定済　手配状態／支払済／勘定済；

図2はこの状態表現を事象表現に変換したものである．色塗りのボックスが複合アクション，白抜きのボックスが原子アクションである．活性化の順序はボックスを結ぶ矢印で表わされている．アクション名の下のかっこ内は，このアクションの内部で活性化される原子アクションである．

5．抽象サブシステムの設計

オブジェクトの再利用性は多様な業務環境に現われるオブジェクトの類似性を抽出し，体系化することによって実現される．類似性の抽出は単一のクラスの性質のみでなく，サブシステムの性質に関しても適用される．サブシステムにおいては次のものが類似性抽出の対象となる．

（1）サブシステムのパートナー．
（2）パートナーのもつ役割．
（3）パートナーの協調．

（1）の類似性の抽出は汎化概念による抽象クラスを考えることである．（2），（3）の類似性の抽出はパートナーの振舞いおよび振舞いの関連性の類似性を考えることである．これはサブシステムが受けもつ業務の類似性を考えることにほかならない．このことは業務ルールを表現する集約オブジェクトのライフサイクルおよびパートナーのライフサイクル間制約の類似性を抽出し，仕様化することに置き換えられる．（1）～（3）の類似性を抽出することによって得られる抽象クラスの集合を抽象サブシステムあるいはフレームワークという．

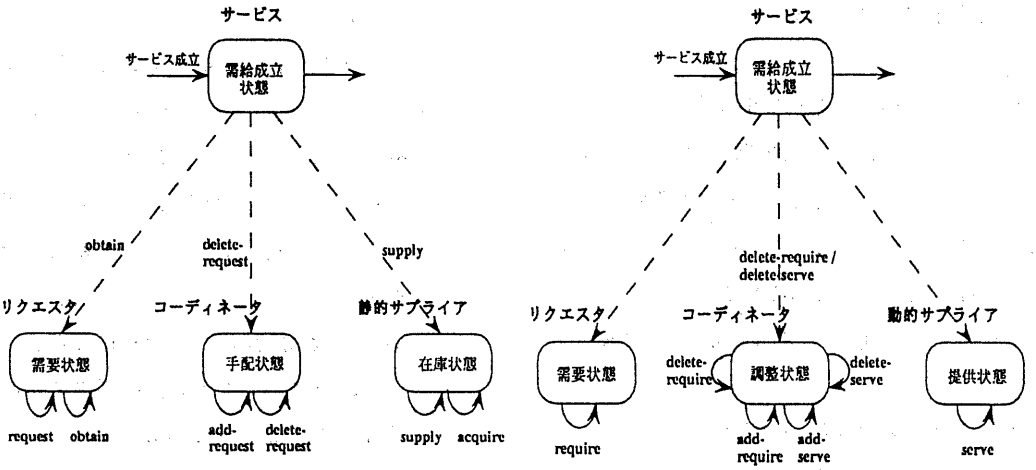
抽象サブシステムの例として，レストラン，販売管理，ホテルあるいは航空機の予約管理，銀行，劇場，洗車場などの業務環境の類似性を抽出した資源受給サブシステムを考える．個々の業務環境は資源受給サブシステムのクラスのサブクラスの集合（具象サブシステムという）として実現される．資源受給サブシステムは次のパートナーをもつ抽象サブシステムである．

（1）リクエスタ：資源を要求する役割をもつオブジェクトのクラス．
（2）サプライア：資源を供給する役割をもつオブジェクトのクラス．便宜上サプライアを2つのサブクラス静的サプライアと動的サプライア に特化する．前者は商品，ホテル，航空機のような受動的な資源を扱うオブジェクトであり，後者は銀行あるいは劇場の窓口，洗車場のような，それ自身がリクエスタを要求する能動的なオブジェクトである．
（3）コーディネータ：リクエスタとサプライアの受給要求を仲介し，調整する役割をもつオブジェクトのクラス．
（4）サービス：資源受給の業務を表す集約オブジェクトのクラス．リクエスタ，サプライア，コーディネータの協調によって生成される．

業務のルールは集約クラス サービス のライフサイクルによって表されるが，ここではその中の受給成立状態を考える．この状態はリクエスタが資源を取得し，かつサプライアが資源を供給した状態を意味する．静的サプライア，動的サプライアのそれぞれの場合における受給成立状態のスクリプトは以下のように設計される．

［静的サプライアにおける受給成立状態］

パートナーのライフサイクルとライフサイクル間制約を図3（a）に示す．リクエスタは需要状態を表す属性（たとえば要求資源，取得資源，取得量）を維持し，この状態の遷移を引き起こすアクション request, obtain をもつ．静的サプライアは在庫状態を表す属性（たとえば資



（a）静的サプライアの場合　　（b）動的サプライアの場合

図3　受給成立状態における状態依存制約

源種別と在庫量）を維持し，この状態の遷移を引き起こすアクション supply, acquire をもつ．コーディネータの手配状態は供給待ちのリクエスタの要求を維持し，この状態の遷移を引き起こすアクション add-request, delete-request をもつ．

受給成立状態に関するパートナーのライフサイクル間制約は次のように記述される．

state(s:サービス，受給成立状態) **requires**
　state(r:リクエスタ，需要状態) **thru** obtain;
state(s:サービス，受給成立状態) **requires**
　state(c:コーディネータ，手配状態) **thru**
　　delete-request;
state(s:サービス，受給成立状態) **requires**
　state(p:静的サプライア，在庫状態) **thru** supply;
state(r:リクエスタ，需要状態) **thru** obtain
　synchronizes with
　state(p:静的サプライア，在庫状態) **thru** supply;

この制約から状態遷移列 手配状態／＜需要状態，在庫状態＞／受給成立状態 を導き，受給成立状態のスクリプトの状態表現として定義する．

［動的サプライアにおける受給成立状態］

図3（b）に示されるように，リクエスタは需要状態を表す属性（たとえば享受内容）を維持し，この状態の遷移を引き起こすアクション require をもつ．動的サプライアはサービス提供状態を表す属性（たとえば提供内容）を維持し，この状態の遷移を引き起こすアクション serve をもつ．コーディネータの調整状態は，リクエスタのサービス享受待ちあるいは動的サプライアのサービス提供待ちを表す属性（たとえばリクエスタ，動的サプライアの待ち行列）を維持し，この状態の遷移を引き起こすアクション add-require, delete-require, add-serve, delete-serve をもつ．

受給成立状態に関するパートナーのライフサイクル間制約は次にように記述される．

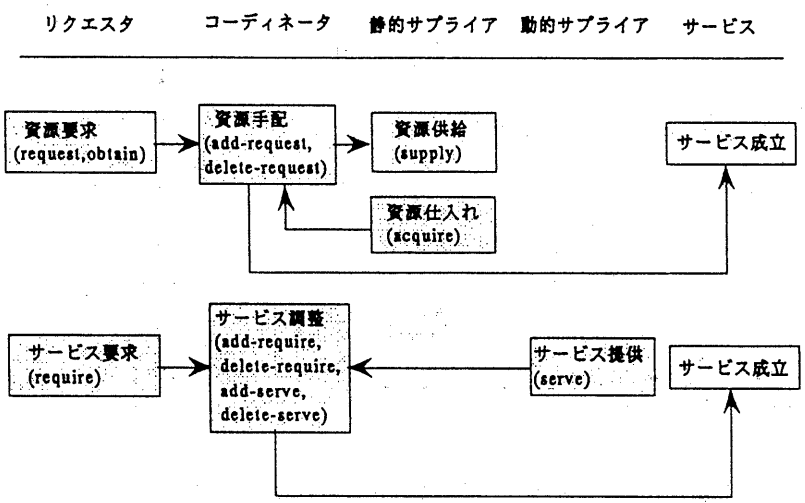


図4　受給成立状態のスクリプトの事象表現

state(s:サービス，受給成立状態) **requires**
state(r:リクエスタ，需要状態) **thru** require;
state(s:サービス，受給成立状態) **requires**
state(c:コーディネータ，調整状態) **thru**
delete-require **or** delete-serve;
state(s:サービス，受給成立状態) **requires**
state(p:動的サプライア，提供状態) **thru** serve;
state(r:リクエスタ，需要状態) **thru** require
synchronizes with
state(p:動的サプライア，提供状態) **thru** serve;

この制約から状態遷移列 調整状態／＜需要状態，提供状態＞／受給成立状態 を導き，受給成立状態のスクリプトの状態表現として定義する．

以上の手順にしたがって定義されたスクリプトの状態表現は何通りかの事象表現に変換される．図4に事象表現の例を示す．

6．サブシステム概念について

システム分割の概念としてのサブシステムの定義，およびその境界を定める基準ないし方法はまだ明確でない．サブシステムがオブジェクトの協調に依存するという事実が，その最適な境界の設定を困難にしている．

サブシステムはパートナークラスに存在依存するという意味で，サブシステム S はパートナークラス $P_1, \ldots, P_n$ の集約オブジェクトと考えることができる．すなわちサブシステムは次の状態依存制約によって定義される．

state(S:Σ, **exist**) **requires** state(P_i:Π_i, **exist**)
(i=1, ..., n)

ただしΣはSをインスタンスにもつクラス，Π_i は P_i をインスタンスにもつメタクラスである．また **exist** はそれぞれ S, P_i が存在することを意味する状態である．

サブシステム自身の振舞い，サブシステムとパートナークラスの継承階層との関連，サブシステムとパートナークラスの振舞いの関連，またサブシステムの識別アルゴリズムなどの問題はさらに解明されるべき領域である．

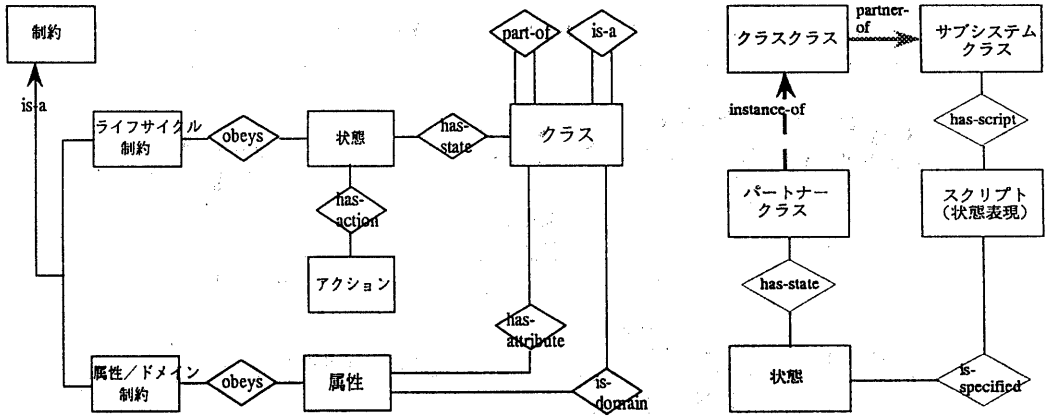


図５　オブジェクトモデルにおけるコンストラクタの関連

抽象サブシステムすなわちフレームワークは，オブジェクトの再利用性を実現する基本手段である．抽象サブシステムの設計において，具象サブシステムのどのような性質を抽象化すべきかも明確でない．この問題の解決のためには，考察領域におけるビジネスルールの表現あるいはその制約形式へのモデル化，オブジェクトの振舞いのより精密な表現，また制約や振舞いの抽象化概念などを含む，より豊かな意味表現力をもつオブジェクトモデルの再構築が必要である．図５は考えるべき要素とその関連の一部を試みに描いたものである．

７．あとがき

この提案は抽象サブシステム開発技法の基本構想である．さらに６にあげたような問題について考察を進めたい．

参考文献

(1) Sakai,H.: An Object Behavior Modeling Method, Proc. 1st Int. Conf. on Database and Expert Systems Applications, pp.42-48(1990).

(2) Sakai,H.: An Object Behavior Modeling Augmented with Modeling Integrity Constraints, Proc. 2nd Int. Symposium on Database Systems for Advanced Applications, pp.174-182(1991).

(3) Schrefl,M.: Behavior Modeling by Stepwise Refining Behavior Diagrams, Proc. 9th Int. Conf. on Entity-Relationship Approach, pp.119-134 (1990).

(4) 酒井博敬: オブジェクトの振舞いに関するコントラクトの設計について，情報処理学会論文誌３３巻８号，pp.1052-1063(1992).

(5) Sakai,H.: A Method for Contract Design and Delegation in Object Behavior Modeling, 電子情報通信学会 IEICE Trans. on Information and Systems, Vol.E76-D,No.6(1993).

(6) King, R.: My Cat is Object-Oriented, in Kim, W., Lochovsky, F.H. eds., Object-Oriented Concepts, Databases, and Applications, pp.23-30, ACM Press(1989).

(7) Mattos, N.M.: Abstraction Concepts: The Basis for Data and Knowledge Modeling, Proc. 7th Int. Conf. on Entity-Relationship Approach, pp.473-492(1988).

(8) Helm,R., Holland,I.M., Gangopadhyay,D.: Contracts : Specifying Behavioral Components in Object_Oriented Systems, Proc. Object_Oriented Programming : Systems,Languages,and Applications '90, pp.169-180(1990).

(9) Wirfs-Brock,R., Wilkerson,B., Wiener,L.: Designing Object-Oriented Software, Prentice-Hall(1990).