データベースシステム 89−14
（１９９２．７．２３）

振舞いの分析によるオブジェクトの集約化

酒井博敬† 堀内 一††

†中央大学理工学部 ††日立製作所ビジネスシステム開発センター

オブジェクトの振舞いの設計において集約化は基本的な役割を果たす．まずオブジェクトの存在依存性の視点から集約化を見直し，構成集約化と協定集約化を定義した．次にライフサイクル概念にもとずいてオブジェクトの振舞いをモデル化し，これを用いてオブジェクトの協調の基礎となる振舞い制約を定式化した．協調的に振舞う複数のクラスのオブジェクトの集まりをサブシステムという．ここではサブシステムを振舞い集約化としてとらえ，振舞い制約の分析にもとずいて，サブシステムの振舞いを組織的に設計する方法を示した．

Aggregation of Objects through Behavioral Analysis

Hirotaka Sakai† Hajime Horiuchi††

†Chuo University ††Hitachi, Ltd.

The aggregation concept plays a fumdamental role in designing behavior of objects. We defined the aggregation from a viewpoint of the existential dependency of objects, and proposed two types of the concept : the structural aggregation and the cooperative aggregation. In the framework of the behavior model that reflects the life cycles of objects, we formulated the behavioral constraints . We further defined the concept of behavioral aggregation that is a subsystem constructed from collaborating object classes. Based on the analysis of behavioral constraints, we proposed a method for designing behavior of subsystems.

1. はじめに

集約化概念はオブジェクトの振舞いの設計において基本的な役割を果たすが，両者の関係は十分議論されていない．ここでは振舞いの視点から集約化を整理し，集約化概念にもとずいた振舞い設計の体系化を試みる．

2. オブジェクトの集約化

集約化(aggregation)とは，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作り上げることである[3, 4, 5, 10]．構成要素となるオブジェクトを成分または成分オブジェクト，構成されたオブジェクトを集約オブジェクトという．たとえば自動車は駆動装置，車輪，車体を成分とする集約オブジェクトである．このとき自動車は，その成分オブジェクトの詳細を抑止してアイデンティティ[1]をもつ1つの実体として認識される．

集約化において，成分オブジェクトがすべて存在しなければ集約オブジェクトは存在し得ない．たとえば自動車は駆動装置，車輪，車体がすべてそろわなければ存在し得ない．この意味で集約オブジェクトはすべての成分オブジェクトに存在依存する．さらに成分オブジェクトの一部のアイデンティティが変われば，集約オブジェクトのアイデンティティも変わることがある．すなわちアイデンティティ依存性を制約として付加することもできる．

集約オブジェクトおよび成分オブジェクトのクラスをそれぞれ集約クラスおよび成分クラスという．集約クラスは成分オブジェクトを参照する属性をもつこともあるし，もたないこともある．たとえばクラス 自動車 は属性 使用駆動装置：駆動装置，使用車輪：set 車輪，使用車体：車体をもつであろう．また1つのオブジェクトが2つ以上の集約オブジェクトの成分となることもある．設計仕様上，エンジンの同一オブジェクトが自動車と飛行機の成分として使われるような例がこれにあたる．集約-成分の関連が再帰的であることもある．たとえばソースモジュールはいくつかのサブモジュールとしてのソースモジュールからなる．

集約オブジェクトの振舞いにおいて，成分オブジェクトを参照する属性の値は一般に安定しており，変化するのはそれ以外の属性の値である．たとえば自動車において，属性 使用駆動装置，使用車輪，使用車体の値は安定しているが，属性 所有者，走行距離，整備歴 の値は変化するであろう．

オブジェクトを形式的にそのすべての属性値の集約化と定義する考え方があるが，これには問題がある．たとえば自動車の属性 所有者，走行距離，整備歴は自動車が存在するために必須の成分であろうか．またこの定義によれば，あるオブジェクト o を構成する成分オブジェクトの成分オブジェクトはまた o の成分と考えることができる．そうすると所有者の成分として勤務先の会社や給与があるとすれば，会社や給与は自動車が存在するために必須の成分であろうか．このような不自然さは集約化を形式的に定義したことに起因する．すなわち集約化は，オブジェクトが存在するためにいかなる成分がなければならないかという意味を考えて規定されるものでなければならない．

集約化は集約オブジェクトの振舞いの様子に着目して，構成集約と協定集約（または事象集約）に分けることができる．

［構成集約］

オブジェクトがその成分からなる構造体として認識されるとき，これを構成集約オブジェクト，またそのクラスを構成集約クラスという．成分クラス 駆動装置，車体，車輪をもつクラス 自動車，あるいは成分クラス テキスト，図，表，写真をもつクラス 本 は構成集約クラスの例である．

［協定集約（事象集約）］

オブジェクトがいくつかの成分オブジェクトの協定関係によって成り立つようなとき，これを協定集約オブジェクトとよぶ．またそのクラスを協定集約クラスとよぶ．協定はある事象の発生によって成立するから事象集約ともよぶ．たとえば銀行，駅，劇場などの窓口における客へのサービスを考えよう．

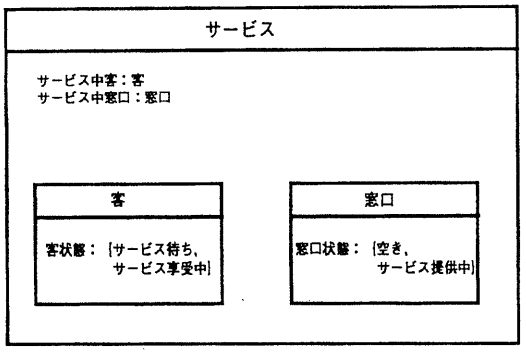

図1　協定集約クラス サービス

図1に示されるように，サービスは2つのクラス客，窓口を成分クラスとする協定集約クラスとして定義される．窓口（のオブジェクト）はいくつかあって，空いている窓口があれば到着する客（のオブジェクト）はそこでサービスを受ける．このときサービス待ちの客と空き窓口の集約オブジェクトとしてのサービス（のオブジェクト）が生成されると考えてよい．サービスは完了すると消滅するから，客と窓口の集約オブジェクトであるサービスの生成，消滅は動的に起こる．協定集約オブジェクトは，成分オブジェクトの間にそれぞれの特定の状態にもとずく協定があるとき生成され，協定がなくなれば消滅する．

協定集約オブジェクトはその成分オブジェクトに存在依存するが，成分オブジェクトは集約オブジェクトが消滅しても存在し得る．またたとえば客がフライトとともに，新たな協定集約クラス フライト予約を作るように，同一のオブジェクトが，異なる協定集約オブジェクトの成分オブジェクトとなることもある．

3．振舞いのモデル

3．1　オブジェクトのライフサイクル

オブジェクト指向設計においては，オブジェクト固有の振舞いを調べ，その基本操作をアクション（メソド）として記述しなければならない．ここではオブジェクト固有の基本操作の抽出，振舞いの共有性，および振舞いの抽象化を目標として振舞いをモデル化する．

実世界の実体と同様に，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[7, 8, 9]．

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あるいはマイルストーン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使うことにする．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起動される．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ことにする．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 を起動する事象を形式 [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 の起動はオブジェクトのある状態 u をある状態 v に変える効果をもつ．すなわちオブジェクトが状態 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，アクションは起動されない．u を t の事前状態 (pre_state)，v を t の事後状態 (post_state) という．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．**produce** はオブジェクトを生成するクラスアクションであり，ライフサイクルには１つしかない．また **consume** はオブジェクトを消滅させるインスタンスアクションで，複数個あってもよい．

ライフサイクルは状態とアクションの列によって表現される．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．

オブジェクトがある状態 u に遷移することは，オブジェクトの状態値が更新されることを意味する．一般に属性の値制約は，オブジェクトが存在しているかぎりいかなる状態にあっても成り立たつべき制約として定義される．これに対してオブジェクトが特定の状態にあるとき成立すべき値制約を，その状態に付随する値制約とよぶ．

３．２　ライフサイクルの洗練

ライフサイクルにおいて定義される状態は，分析の程度によってその精粗は一様ではない．たとえば **exist** はオブジェクトが存在することを表すもっとも粗い状態である．すなわち状態 **exist** とアクション **produce**, **consume** のみからなるライフサイクルを考えることができる．しかし **exist** においては，その状態自身の中でオブジェクトの状態値が時間とともに変わり得る．このため属性の値制約の正確な表現が期待できない．振舞いの一貫性制約をきめこまかく設計するためには状態の洗練が必要である．状態の洗練は必然的にアクションの洗練をともなう．ライフサイクルにおける状態とアクションの洗練をライフサイクル洗練という．

ライフサイクルにおいて，ある状態あるいはアクションをより細分された状態とアクションの列に分解することを，もとの状態あるいはアクションを洗練するという．またある状態あるいはアクションを洗練することをライフサイクルの洗練という．

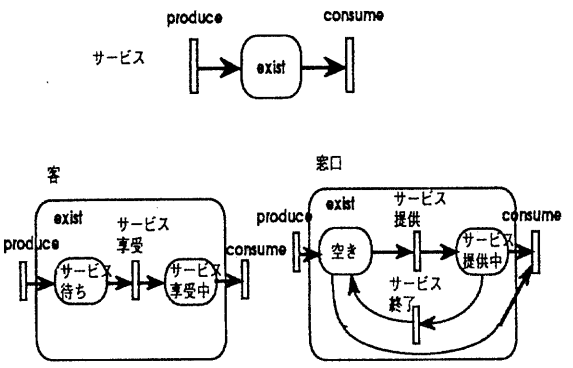

図２　ライフサイクル図の例

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．原子ライフサイクルに洗練することによって，属性の値制約をもっとも精密な形で状態に付随させることができる．また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．

ライフサイクルを図式的に表現したものをライフサイクル図という．図２は客，窓口，およびサービスのライフサイクル図を表している．角の丸い長方形が状態，たて形の細長い長方形がアクションである．図２において，窓口のオブジェクトは空きとサービス提供中への状態遷移を繰り返し，空きの状態にあるときのみ消滅することを示している．このようにライフサイクル図の中に，いくつかの状態遷移を繰り返すループが現われることがある．状態がループ中に現われる場合，同じ状態名でもその状態値は時間とともに変わり得る．

４．振舞いの制約

単一のオブジェクトが事象の発生によってある状態に遷移するには，ライフサイクル記述によって定められた事前状態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振舞うのではなく，どの

応用環境においても，同一または異なるクラスの複数のオブジェクトがメッセージを送り合って協調的に振舞うのが普通である．複数のオブジェクトが協調して振舞うとき，それらのオブジェクトの状態および状態遷移の間に何らかの関連が維持されなければならない．このオブジェクトのライフサイクル間で維持されるべき制約を，以下のように状態間制約および遷移制約として定義する．

［状態間制約］

クラスOのオブジェクトoが状態uにあることをstate(o:O, u)で表すことにし，状態述語とよぶ．状態間制約は「あるクラスOのオブジェクトoが状態uにあるとき，Oと同一または異なるクラスPのオブジェクトpが状態vになければならない」という制約で，次のように書く．

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

状態間制約によれば，オブジェクトoが状態uにあるかぎり，オブジェクトpは状態vを持続しなければならない．記号 **requires** の右には，いくつかの状態が同時に成立していることを要求する複数個の状態述語が現われてもよい．

［遷移制約］

遷移制約は「クラスOのオブジェクトoまたはクラスO自身に対してアクションtを実行するためには，事象[o:O t]または[O t]の発生時にOと同一または異なるクラスPのオブジェクトpが状態vになければならない」という制約で，次のように書く．

[o:O t] **requires** state(p:P, v)；
[O t] **requires** state(p:P, v)；

とくに事象[o:O t]または[O t]の発生時にオブジェクトpが「新たな」状態vに遷移しなければならないことを明示的に表すために，状態述語の前に記号 **new** を付けて次のように書く．

[o:O t] **requires** **new** state(p:P, v)；
[O t] **requires** **new** state(p:P, v)；

また遷移制約において，事象の発生によってオブジェクトoの状態uへの遷移がオブジェクトpの状態vへの遷移を引き起こすべきことを表すために，記号 **requires** の左に事象の代わりに状態述語を用いて次のように書く．

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

遷移制約は事象[o:O t]または[O t]の起動の時点で，あるいはオブジェクトoのuへの状態遷移の時点で，オブジェクトpは状態vにあるか，あるいは新たな状態vに遷移しなければならないことを要求するが，オブジェクトoが事後状態に遷移した後もpが状態vを持続している必要はない．記号 **requires** の右には，いくつかの状態が同時に成り立つことを要求する複数個の状態述語，あるいはいくつかの状態遷移が同時に起ることを要求する複数個の **new** の付いた状態述語が現われてもよい．

たとえばクラス客，窓口，サービスにおいて次の振舞い制約がある．

(C1) ［サービス **produce**］ **requires**
state(g:客，サービス待ち); state(c:窓口，空き)；
"客gに対する窓口cでのサービスを生成するためには，gはサービス待ち，cは空きの状態でなければならない．"

(C2) state(s:サービス，**exist**) **requires**
state(g:客，サービス享受中)；
state(c:窓口，サービス提供中)；

(C3) state(g:客，サービス享受中) **requires**
state(s:サービス，**exist**)；

(C4) state(c:窓口，サービス提供中) **requires**
state(s:サービス，**exist**)；

この例にみられるように協定集約オブジェクトとその成分オブジェクトのライフサイクルの間には，一般にいくつかの状態間制約および遷移制約がある．そもそも協定集約オブジェクトは成分オブジェクト間の協定によって存在するわけだから，その状態には成分オブジェクトの状態が反映される．すなわち成分オブジェクトの状態の協定によって集約オブジェクトの状態が成り立つことになる．

5． コントラクト

オブジェクトの属性とアクションはオブジェクト固有の性質であると同時に，オブジェクトが外部に対してもつ基本的な責務(responsibility)でもある．すなわちあるオブジェクトからその属性値を問われればこれに答え,メッセージを送られればアクションを実行する．

クラスOのオブジェクトoがクラスPのオブジェクトpにメッセージを送り，pのある責務の遂行を要求するとき，その責務に関してoをクライアント，pをサーバという．そしてクライアントの集合としてのOをクライアントクラス，サーバの集合としてのPをサーバクラスという．

クライアントクラスOのオブジェクトがサーバクラスPのオブジェクトに要求できる責務は，PがOに対して請け負った一定の責務にかぎられる．この責務の集合をサーバクラスPのクライアントクラスOに対するコントラクト[2]のといい，contract(P, O)と書く．簡単のため，属性値の問合せに関する責務はどのクラスに対するコントラクトにも含まれるものとし，以後コントラクトとしては属性値の更新に関する責務のみを問題にする．

客と窓口の集約オブジェクトとしてのサービスの振舞いは，成分オブジェクトの振舞いと協調する．振舞いの協調はスクリプトとして表現される．スクリプトは協調する一連のオブジェクトの振舞いであるが，その振舞いの主体となるクラスあるいはオブジェクトの複合的な操作として仕様化される．すなわちクラスOで定義される1つのスクリプトは，Oにおける他のスクリプトや原子アクション，および他のクラスのスクリプトや原子アクションの起動から構成される．

例として単一のサービスのオブジェクトの生成と消滅に関するスクリプトをそれぞれサービス生成，サービス消滅とし，2つのスクリプトをクラスサービスの複合操作として仕様化する．

スクリプトは振舞いの制約から導かれるが，その記述には次のように状態表現と事象表現がある．

script サービス生成；
［状態表現］
pre_state: state(g: 客，サービス待ち);
state(c: 窓口，空き);
post_state: state(s: サービス，**exist**);
state(g: 客，サービス享受中);
state(c: 窓口，サービス提供中);
［事象表現］
event [サービス **produce**]; [g: 客 サービス享受];
[c: 窓口 サービス提供]；

script サービス消滅；
［状態表現］
pre_state: state(s: サービス，**exist**);
state(g: 客，サービス享受中);
state(c: 窓口，サービス提供中);
post_state: state(g: 客，**not_exist**);
state(c: 窓口，空き);
state(s: サービス，**not_exist**);
［事象表現］
event [g: 客 **consume**]；[c: 窓口 サービス終了]；
[s: サービス **consume**]；

2つのスクリプトは成分クラス客，窓口の協調を必要とする．すなわち客と窓口はサーバとして振舞い，クライアントから要求されるスクリプトあるいはアクションを実行する．そのため客，窓口はサービスに対するコントラクトをもつことになる．このときクライアントはコントラクトに記述されたスクリプトないしアクションをサーバに依託（delegate）するという．この例では，3つのクラス間の協調は

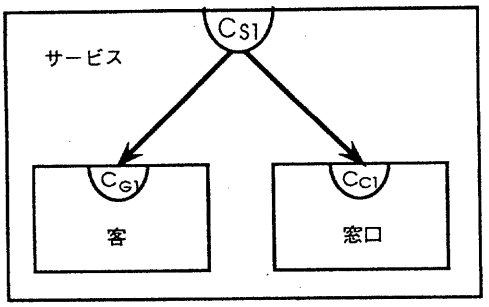

図3　コントラクト図の例

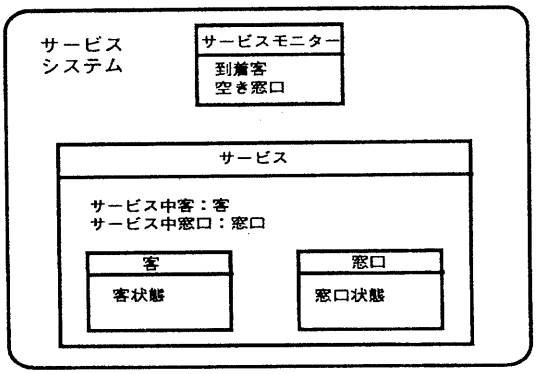

図4　サービスシステム

図3のコントラクト図によって概念的に表される．C_{S1}はサービスの責務を表すコントラクトである．サービスから客のコントラクトC_{G1}および窓口のコントラクト C_{C1} への矢印が依託を表している．ここでC_{G1}, C_{C1}はそれぞれ次のコントラクトである．

C_{G1} = **contract** (客，サービス)
= {サービス享受，**consume**}；
C_{C1} = **contract** (窓口，サービス)
= {サービス提供，サービス終了}；

6．　振舞い集約としてのサブシステム

ある業務環境において，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サブシステム，各オブジェクトをそのパートナー，パートナーの属するクラスをパートナークラスとよぶ．

サブシステムの振舞いはパートナーの協調的な振舞いの合成として実現される．サブシステムの振舞いはパートナーの振舞いがすべてそろわなければ存在しない．これはオーケストラによる演奏にたとえることができる．この意味でサブシステムはパートナーの振舞い集約 (behavioral aggregation) であるという．

振舞い集約としてのサブシステムも，他のサブシステムあるいは他のクラスに対する責務としてのコントラクトをもつ．サブシステムのコントラクトは，いわばパートナーの振舞いを集約した振舞いに関する仕様である．

サブシステムのコントラクトを設計するためには，パートナーが維持すべき振舞い制約と，パートナー間で結ばれるべきコントラクトをも明らかにしなければならない．この考察にもとずいて，サブシステムをパートナーの振舞いに関する制約記述と，サブシステムおよびパートナークラスのコントラクト記述から構成する．

サービス提供を責務とするサブシステムをサービスシステムとよぶことにする．サービスシステムには多数のサービス要求が発生する．要求を円滑に処理するためにただ1つのオブジェクトをもつクラス サービスモニター を設けることにしよう．そうすると図4に示されるように，サービスシステムは4つのパートナークラス 客，窓口，サービス，およびサービスモニター によって構成されることになる．

サービスモニター（のオブジェクト）はサービスの生成，消滅のスケジューリングを行なう制御ポイントとして振舞う．そのためにサービス待ちの客の集合を値とする属性 到着客 と，空いている窓口の集合を値とする属性 空き窓口 をもち，これをサービスモニターの状態として維持する．

サービスモニターは図5のライフサイクル図に示されるような振舞いをもつ．新モニター状態は図に示されるように2つの洗練された状態をもつ．サービス可能状態は2つの属性値がともに空でない状態で，サービス不可状態はそれ以外の状態である．これらの状態値は付随する値制約として定義される．

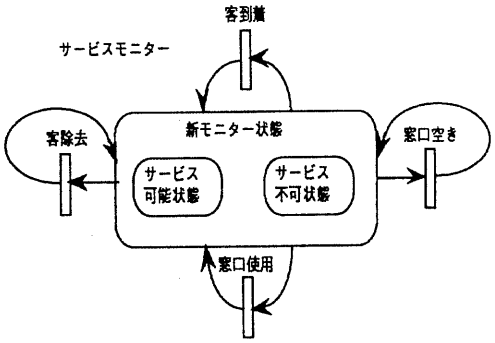

図5　サービスモニターのライフサイクル図

4つの原子アクション客到着，客除去，窓口空き，および窓口使用はいずれも新モニター状態の遷移を引き起こすアクションで，次のような操作である．

action 客到着（g: 客）；
　" 客のオブジェクトgを属性到着客の値に加える．"

action 客除去（g: 客）；
　" 客のオブジェクトgを属性到着客の値から除く．"

action 窓口空き（c: 窓口）；
　" 空きの状態になった窓口のオブジェクトcを属性空き窓口の値に加える．"

action 窓口使用（c: 窓口）；
　" サービス提供中の状態になった窓口のオブジェクトcを属性空き窓口の値から除く．"

さらにサービスシステムにおいて，サービスモニターとその他のパートナーの間に次の振舞い制約を設ける．

(C5)　[m: サービスモニター　客到着] **requires**
　　new state(g: 客，待ち状態)；
　" 客が到着すれば新たな客のオブジェクトが生成される．"

(C6) state(m: サービスモニター，サービス可能状態)
　　requires new state(s: サービス，**exist**)；
　" サービス可能状態ならば新たなサービスが生成される．"

サービスモニターの状態記述の中で，とくにサービス可能状態に関しては次のように記述する．

state(m: サービスモニター，サービス可能状態) **triggeres**
　　[サービス　サービス生成]；

記号 **triggeres** は，左に書かれた状態が右に書かれた事象を発生させるトリガーになることを表している．すなわち事後状態としてサービス可能状態になると，サービスのスクリプトサービス生成を起動することになる．これは遷移制約(C6)から導かれる．

客への窓口サービスをスケジュールするために，サービスモニターのスクリプトとして客サービスを定義しよう．このスクリプトは振舞い制約から次のように導かれる．このスクリプトの中で，クラス客に対してアクション **produce** を依託している．

script 客サービス；
[状態表現]
pre_state: state(m:サービスモニター,新モニター状態);
post_state: state(g: 客，待ち状態)；
　　state(m: サービスモニター,新モニター状態)；
[事象表現]
event　[客　**produce**]；
　　[m: サービスモニター　客到着]；

ここでサービスはそのオブジェクトの生成，消滅の際，そのことをサービスモニターに通知する必要がある．そのためにスクリプトサービス生成，サービス消滅において，それぞれサービスモニターのアクション客除去，窓口使用，および窓口空きを適切に起動するように仕様を変更する．変更された事象表現のみを次に示す．

script サービス生成；

［事象表現］

event [サービス　**produce**]; [g: 客　サービス享受];
[c: 窓口　サービス提供];
[m: サービスモニター　客除去];
[m: サービスモニター　窓口使用];

script サービス消滅；

［事象表現］

event [g: 客　**consume**]; [c: 窓口　サービス終了];
[m: サービスモニター　窓口空き];
[サービス　**consume**];

この変更によって，サービスはサービスモニターにアクション客除去，窓口使用，および窓口空きの起動を依託することになる．

サービスシステムのクライアントはユーザインタフェースというサブシステムであるとする．サービスシステムがユーザインタフェースに対して請け負う責務は客の到着を受け入れ，これにサービスを提供することである．したがってサービスシステムのクライアントに対するコントラクトは，客へのサービスというただ1つの責務に関する仕様ということになる．一般にサブシステムのコントラクトもスクリプトの集まりとして記述される．サービスシステムの場合，コントラクトはただ1つのスクリプト客サービス受け入れからなる．このスクリプトの仕様はクラスサービスモニターのスクリプト客サービスを起動することにほかならない．すなわち次のようにサービスモニターに対する依託の形式で記述される．

script　客サービス受け入れ；
event　[m: サービスモニター　客サービス]；

結局依託されたサービスモニターの振舞いは，これまで述べたようにサブシステム内のパートナーの振舞いの協調によって実現されることになる．ここでサービスシステムのパートナークラスにおけるクライアント，サーバの関係，および各サーバクラスにおけるコントラクトは図6のようなコントラクト

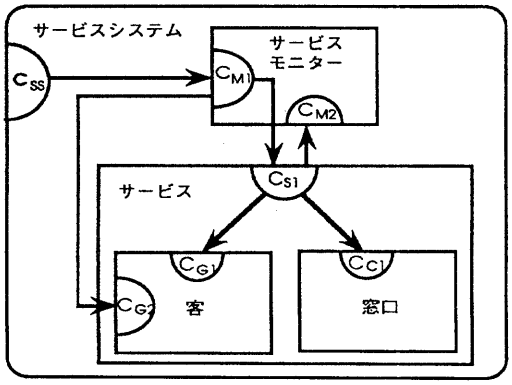

図6　サービスシステムのコントラクト図

図によって表される．C_{SS} はサブシステムのコントラクトであり，ここから C_{M1} への矢印がサービスモニターへの依託を表している．コントラクトの構成は次に示すとおりである．

C_{SS} = **contract**(サービスシステム，ユーザインタフェース) = {客サービス受け入れ}；

C_{M1} = **contract**(サービスモニター，サービスシステム) = {客サービス}；

C_{M2} = **contract**(サービスモニター，サービス)
= {客除去，窓口使用，窓口空き}；

C_{S1} = **contract**(サービス，サービスモニター)
= {サービス生成}；

C_{G1} = **contract** (客，サービス)
= {サービス享受，**consume**}；

C_{G2} = **contract**(客，サービスモニター)
= {**produce**}；

C_{C1} = **contract** (窓口，サービス)
= {サービス提供，サービス終了}；

```
subsystem サービスシステム
 partner class  サービスモニター; サービス; 客; 窓口;

 constraint
 " 振舞い制約 C1 ~ C6 の記述"

contract ( サービスシステム, ユーザインタフェース) ;
 script  客サービス受け入れ ;

contract ( サービスモニター, サービスシステム) ;
 script  客サービス ;

contract ( サービスモニター, サービス) ;
 action  客除去 ;
 action  窓口使用 ;
 action  窓口空き ;

contract ( サービス, サービスモニター) ;
 script サービス生成 ;

contract (客, サービス) ;
 action  サービス享受 ;
 action  consume ;

contract (客, サービスモニター) ;
 action  produce ;

contract (窓口, サービス) ;
 action  サービス提供 ;
 action  サービス終了 ;
```

図7　サブシステム　サービスシステム

なおサービスにおけるスクリプトサービス消滅はサービスの内部で起動されるべき内部スクリプトであり，コントラクトには現われない．図7にサブシステムサービスシステムの記述の概略をあげる．詳細は本文中に示した通りである．

7．　あとがき

存在依存性にもとずいて集約化概念の見直しを行なった．さらにライフサイクルと振舞い制約の概念を取り入れてオブジェクトの振舞いをモデル化し，振舞い集約としてのサブシステムの設計法を提唱した．

参　考　文　献

1) Atkinson, M., Bancilhon, F., DeWitt, D., Dittrich, K., Maier, D., Zdonik, S . : The Object-Oriented Database System Manifesto, Proc. 1st Int. Conf. on Deductive and Object-Oriented Databases, pp.40-57(1989).

2) 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).

3) Hull, R., King, R. : Semantic Database Modeling : Surveys, Applications, and Research Issues, ACM Comput. Surv., Vol.19, No.3, pp.201-260(1987).

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

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

6) Rebecca, J., Wirfs-Brock, Johnson, R. E. : Surveying Current Research in Object-Oriented Design, Comm. ACM, Vol.33, No.9, pp.104-124(1990).

7) Robinson, R. A. : An Entity/Event Data Modeling Method, Comput. J., Vol.22, No.3, pp.270-281(1979).

8) Sakai, H., Horiuchi, H. : A Method for Behavior Modeling in Data Oriented Approach to Systems Design, Proc. 1st Int. Conf. on DataEngineering, pp.492-499(1984).

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

10) Smith, J.M., Smith, D.C.P. : Database Abstractions : Aggregation and Generalization, ACM Trans. Database Systems, Vol.2, No.2,pp.105-133(1977).