データベース・システム 72- 7
(1989. 7. 20)

# オブジェクトベースプログラミング言語 Jasmine/C

# The Object-Base Programming Language Jasmine/C

牧之内顕文　石川 博　鈴木文雄
MAKINOUCHI Akifumi　ISHIKAWA Hiroshi　SUZUKI Fumio
富士通株式会社
Fujitsu Limited

## 1. 始めに

Jasmine はマルチメディア知識ベースの開発・利用・運用を効率よくかつ容易に行うためのシステムである。Jasmine/C はJasmine が管理するオブジェクトを定義し、操作するためのオブジェクトベースプログラミング言語である。本稿ではこの言語の設計思想と機能について報告する。

Jasmine はオブジェクト指向モデルに基づいた知識ベースシステムであり、設計目標は次の３つである。

（１）人工知能分野の応用のための大量の知識を格納・操作・管理できる。

（２）データベース分野の高度応用に必要となる複雑な構造のデータも扱える。

（３）何れの分野でも必須であるイメージ、グラフィックスなどマルチメディアを格納・操作・管理できる。

ＡＩ分野での知識表現に影響を与えた考えの一つにフレーム[13,22]がある。フレームはデータ構造であり、デーモンと呼ばれる手続きを付加できることが特徴である。多くの知識表現言語[6]がこれに基づき開発されてきた。Loops[19]は代表的な例である。それはフレームを基にして、データ、手続き、ルールをオブジェクト指向パラダイム[18]で組織化した知識プログラミング言語である。他のいわゆるＡＩシェル言語の多くはこの路線に概ね沿っている。

これら知識表現言語は知識の表現では強力であるが大量の知識を効率よく管理する機能に欠ける。あるとしてもそれはよくて外部データベースシステムへのアクセス機能であり、知識表現の枠組みの中に統一的に組み込まれていない。一方、データベース分野では関係モデルがモデルの簡明さと強力な操作言語によって評価を固めた。しかし、関係モデル[1]は

（１）エンジニアリングやオフィス環境で見出される複雑な構造のデータを表現できない、

（２）データの動的側面を表現できない、

（３）モデルのデータタイプ（レコード）とプログラミング言語で扱うデータタイプの間にギャップがあるなどの欠点が指摘されている。

（１）と（３）は互いに関連している。ＣＡＤやＯＩＳのような応用では複雑なデータ構造を表現するのに言語が提供するタイプ機能を使う。そこでは既存のタイプを使って新しいデータのタイプを定義し[10]、その意味を明確にする。しかし、データの意味はタイプ定義だけでは充分捉えられない。プログラマはデータの振る舞いや解釈をそのタイプ固有の操作によって表現する。これら操作は言語の手続きあるいは関数によって記述される。

これら欠点を克服する試みには大きく２つの方向がある。一つはSMALLTALK[9]で代表されるオブジェクト指向言語の概念を借りてそれをデータベースに適用する方向。他は関係モデルの拡張である。前者にはGemStone[5]、ORION[3,23]などがある。後者はPOSTGRES[15]が有名である。GemStoneやORION はSMALLTALK の直接の子供である。そこではオブジェクト、クラス、インスタンス変数、クラス階層、継承、メソッドと言った概念がほぼそのまま生かされる。オブジェクトの属性は変数で表され、その値はメソッドを通してのみ操作される。

ＩRIS[7]は関数データモデル[16]を基にして、それに分類(classification)、汎化(generalization)、集約(aggregation)[17]と動的抽象化(behavioral abstraction)を導入したシステムである。関数はテーブルとしての蓄積関数(stored function)、問い合わせコマンドから合成される導出関数(derived function)が実現されている。又、Ｃ言語のような汎用プログラミング言語で書かれた操作を外部関数として実現する計画がある。インタフェースはＣ言語サブルーチン群、オブジェクトＳＱＬ、ＬＩＳＰインタフェースがある。

POSTGRESは関係モデルに手続きタイプ[20]を導入した。実行可能なテーブル操作コマンドが手続きタイプのフィールドに格納される。それは他のタイプと同様操作されうる。実行されると、テーブルを返す。このアプローチは関係モデルの単純さを保持しながら複雑な構造を表現しようとしたものであるが処理の単位はレコードであり上記（３）の欠点を克服してはいない。

Jasmine は関数データモデルをオブジェクト指向パラダイムで拡張したモデルに基づいて開発されたシステムである。拡張した点は

（１）関数の定義域、値域をオブジェクトの集合とし、それをクラスとする。

（２）クラス上に定義された関数はそのクラスの属性と呼ぶ。属性には静的属性と動的属性がある。後者は属性値を動的に（手続きにより）計算する。動的属性はＣ言語を拡張したJasmine/C の任意の関数を書くことが可能である。

（３）クラスは上位／下位関係で階層化され、属性の継承がなされる。

（４）属性にはデーモンが定義される。

（５）Jasmine の親言語は言語Ｃをベースにした Jasmine/C である。そこではクラスはタイプの一つである。

Jasmine は関数データモデルに基づいているのでインスタンス変数という概念はない。この点がGemStoneやORION 等と異なる。Irisとは基本モデルが同じである。しかし、Jasmine の動的属性はオブジェクト指向モデルのメソッドの概念に近い。具体的には上記（２）、（４）、（５）でIrisと異なる。

一方、Jasmine/C はオブジェクト指向言語の一種とみなせる。Ｃを基盤とした言語にはＣ++、Objective C などがある。それらとJasmine/C が異なる点は後者が永続的オブジェクトをサポートしている点である。

本稿ではJasmine/C の永続的オブジェクトの操作機能を中心に、具体例をもとに説明する。その説明に必要な限りにおいてJasmine モデルをまず概説する。

## ２．Jasmine モデル

Jasmine が扱うオブジェクトに２種類ある。一つは自己参照型（self-reference）であり、他は参照型(reference) である。前者にはＣ言語で扱うデータタイプのほとんどが含まれる。それらのデータは自分自身がオブジェクト識別子（OID)である。参照型は自身とは異なる識別子を持つ。自己参照型と参照型オブジェクトは存在の仕方が違う。参照型オブジェクトは識別子が異なれば異なる存在である。しかし自己参照型は同一のオブジェクトが複数個存在しうる。例えば、整数の１０はオブジェクトベース中の複数の場所に格納されるがそれらは皆同じ１０である。

オブジェクトの集合をクラスと呼ぶ。クラス間に関数（数学的な意味で）を定義できる。関数の定義域であるクラスを定義域クラス、値域のクラスを値域クラスと呼ぶ。例えば、部品オブジェクトの集合であるクラス Part を考えよう。PartNoはクラスPartを定義域とし、Integer クラスを値域とする関数である。整数 10 が部品 'obj001' の部品番号であれば 'PartNo(obj001) = 10 ' の関係が成り立つ。

Jasmine では関数が返す値に多値と未定義値を許す。多値はオブジェクトの集まりである。本稿では記号 '｛・・・｝' で表現する。未定義値は 'nil'で表し、値が未定義であることを示す。空集合 '｛｝' は「無い」ことを意味し、未定義値とは区別される。

関数の定義方法には２種類ある。一つは定義域と値域の両クラス間に対応を与える方法である。これを数え上げによる定義と呼ぶ。２番目は関数を手続き的に定義する方法である。Jasmine ではJasmine/C の関数（プログラミング言語の意味で）で定義される。これを手続き定義と呼ぶ。今、ある関数の定義域と値域の両クラスが参照型オブジェクトのクラスであり、関数が数え上げ定義によるものならばその逆関数が定義される。

クラスが関数の定義域である時、その関数はそのクラスに定義されているといい、そのクラスの属性と呼ぶ。又、そのクラスに属するオブジェクトの属性でもある。オブジェクトの属性は値をもち、その値は対応する関数をオブジェクトに適用して得られる値である。属性はその定義方法により、静的（又は数え上げ）属性と動的（又は手続き）属性と呼ぶ。

クラス間にIS-A関係を定義する。この関係は通常の上位下位関係である。 Jasmine ではすべてのオブジェクトは次の２つの条件を満たす。

１．全てのオブジェクトは少なくとも一つのクラスに属する。

２．あるクラスに属するオブジェクトはそのすべての上位クラスにも属する。

オブジェクトはそれが属する一番下位のクラスのインスタンスであると言う。

オブジェクトがインスタンスとして属するクラスのことをオブジェクトのタイプと呼ぶ。オブジェクトの属性のタイプは、それを定義する関数が返す値（オブジェクト）のタイプと定義する。図１の例ではオブジェクト obj001 のタイプはPartであり、その属性 PartNo のタイプはInteger である。

図１はオブジェクトベースの例である。この例は文献2)で使用された例とほぼ同じである。この例を用いたのは同文献で紹介されているデータベースプログラミング言語との比較に便利だからである。又、同文献は応用プログラム例としてデータベース定義（タスク１）、簡単な条件検索（タスク２）、複合部品の総コスト（重量）の計算（タスク３）、新複合部品の登録（タスク４）を取り上げている。本稿でもほぼ同様な

例題を取り上げる。

## 3．オブジェクトプログラミング言語 Jasmine/C

Jasmine/C はマルチメディア知識ベースシステム Jasmine のオブジェクト操作言語であり、又手続き属性定義言語でもある。プログラミング言語Cをベースにして、Cを完全に含んでいる。即ち、Cがサポートするデータタイプのほかに Jasmine がサポートする永続的オブジェクトのタイプをも扱える。その中には Image, Graphics などのマルチメディアタイプも含まれる。

### 3．1 クラス創成

クラスの創成は instantiate-class 文で行う。図2に例示した。本稿では書式上クラス（タイプ）名や属性名は大文字で始めている。クラス Part, Supplier, Compositionの上位クラスはシステム定義クラス Compositeである。BasePart、CompositePart のそれは Part である。従って属性の継承により、基本部品，複合部品は属性 Pno, Pname も持つ。前節で述べた属性は 定義時にはproperty, instance-method, set-method, class-method の4種類に分かれる。propertyは静的属性を定義するのに用いる。他は動的属性のためにある。3種類の動的属性の違いは定義関数が受け取る特別なパラメタ変数'self'がとる値の違いである。この変数はinstance-method では単一オブジェクトを、set-methodではオブジェクトの集合を、class-methodではクラス（特別のオブジェクトである）を指す。複合部品の属性 Cost と Mass は動的属性として定義してある。これらはそれぞれ構成部品のコストと重量（基本部品と複合部品で定義が異なる）の総和である。基本部品のコスト（重量）はそのもののコスト（重量）である。一方、複合部品のそれはその構成部品のコスト（重量）の総和に組立コスト(付加重量 )を加えたものである。

動的属性はデータベースの属性とオブジェクト指向言語のメソッドの2面性を兼ね備えている。属性としては静的属性と全く同じ扱いを受ける。一方、オブジェクトの動作を表現（例えば 'Display' )するのに使える。値を返さないときは 'void' タイプである。

### 3．2 オブジェクト変数とタイプ

C言語の変数がタイプをもつようにJasmine/C のオブジェクト変数もタイプを持つ。変数の宣言は‘クラス名 変数名;'で行われる。例えば、‘Part x;'は変数 xのタイプが Part であることを宣言するがその意味は「変数x はクラス Part に属する任意のひとつのオブジェクトを代表する」である。これはC言語の例えば 'Int y ' が「y はInt で規定される（恐らくは）有限の集合に属する任意の一つの値を代表する」ことに似ている。しかし、クラスにはIS-A関係で規定される階層がありそれによって「属する」ことの意味が規定される。Partの下位クラスのBasePartとCompositePart(図1）に属するオブジェクトは又Partにも属する。そして3 つのクラスのインスタンスオブジェクトのタイプは各々互いに異なる。この意味で変数 xはPart、BasePart、CompositePart タイプのユニオンである。

Jasmine/C では多値変数を宣言できる。‘ Part x multi;'は変数x はタイプがPartであるオブジェクトの集合を代表することを意味する。この多値変数はオブジェクトベースの検索が基本的に集合型条件検索であることから必要である。

### 3．3 オブジェクトの同定

Jasmine/C プログラムでオブジェクトを同定する方法には外延的同定と内包的同定の二種類ある。前者はオブジェクト同定子を書くことによって行う。例えば、'10' は Integer型オブジェクト10を意味する。後者は条件によって、それを満足させるオブジェクト（の集合）を同定する。その構文は‘対象オブジェクト集合〔where 条件式〕’である（‘〔 〕’は無くてもいいことを意味する）。これは対象オブジェクト集合の内条件式を満たすオブジェクト（の集合）を同定する。条件式が書かれないときは‘ where true' と同義である。 例えば、'Part'は，Partクラスに属するすべてのオブジェクトを同定する。'Part where Pno == 15'は部品番号が15である部品を求める。しかし求めた部品は基本部品なのか複合部品なのかあるいはそのどちらでもない単なる部品（即ちクラスPartのインスタンスオブジェクト）なのかは分からない。もし基本部品のみを求めたいのなら'BasePart where Pno == 15'とするか'Part where Pno == 15 and Class == <BasePart>'とする。'Class' はすべてのオブジェクトが持つシステム定義の属性でインスタンスオブジェクトの属するクラスのOID を返す。又'<クラス名>'はクラスのOID を意味する外延的同定表現である。

さて、「コストが100 以上の基本部品」を求めるプログラムは‘ BasePart x multi; x = BasePart where Cost >= 100; ’ である。これは文献2)のタスク2に当たる。比較すべき値が外から与えられるのであれば、'BasePart x multi; Int y; 〔比較値を入力しy に代入する〕 x = BasePart where Cost >= y;'とすればよい。

### 3．4 オブジェクトの属性連鎖

オブジェクトが同定されるとそのオブジェクトの属性値としてのオブジェクトを同定できる。それは点記法による。例えば、価格が１００以上の基本部品の部品番号は 'BasePart.Pno where Cost >= 100' で求まる。又は上記タスクプログラムの変数x を使って、'x.Pno' と書ける。さらにそれら部品の納入業者を求めるには別の属性連鎖を使い 'x.-Supply.Sname' とすればよい。ただし '-Supply' は Supply の逆属性である。

変数を条件式の中で使うことも可能である。例えば

業者「富士」によって納入されている部品でコストが100 以上のものを求めるのは上と同じ変数を使い‘ x whrer x.-Supply.Sname == “富士” ’とする。

3．5　動的属性

動的属性の利用形態は基本的には静的属性と同じである。例えば、コストが1000以上の複合部品を求めるのは‘CompositePart where Cost() >= 1000’とすればよい。括弧はＣ言語の関数の構文に合わせて導入した。動的属性を属性連鎖の途中に使うことも自由である。

動的属性はクラス定義中にJasmine/C 言語の関数として書かれる。図３は複合部品のコストの定義である（文献2)のタスク３に対応）。 Scan タイプの変数scanはオブジェクトの集合から単一オブジェクトを取り出すために必要である。Jasmine/C では集合中の個々の要素オブジェクトをＣ言語が用意するデータ処理機能で処理するためにこの機構が要る。Scanはシステム定義の一時的クラスであり、プログラムセッション中保持される。OpenScan, Nextは自己参照型と参照型オブジェクトの各々の最上位クラスに定義された動的属性である。OpenScanの値域クラスはScan、Nextのそれは定義域クラス自身である。Next属性はパラメタで与えられたスキャンオブジェクトが指すオブジェクトを返すと同時にそのスキャンオブジェクトが次のオブジェクトを指すようにすることである（特に指定しなければオブジェクトの集合はシステムで定めれた順序で並べられている）。

この動的属性の定義では、構成部品である複合部品のコストを求めるのに同じ属性を使用する。即ち、この定義は再帰的である。このような再帰的属性により再帰的問い合わせも非常に簡単に書ける。[12]

4．　オブジェクト操作と一貫性制御

インスタンスオブジェクトの創成は対応するクラスのシステム定義動的属性Instantiate によって行う。この属性は新しいオブジェクトを作り、その識別子を返す。図４に例を示す。このプログラム構成は文献2)中に掲載されたAdaplex による例に似ている。これはAdaplex とJasmine が共に関数型データモデルを基盤としていることによる。

Instantiate は前述のCost属性とは性質の異なる動的属性である。前者はクラスに送られるメッセージを受け、後者はインスタンスへのメッセージを受けるメソッドに当たる。この意味でJasmine では前者をクラスメソッドと呼び、インスタンスメソッドと区別する（しかし、メソッドが定義されるのは共にクラスである）。‘CompositePart.Instantiate()’ ではクラスCompositePart のオブジェクト識別子が第一パラメタとして関数Instantiate に与えられる。関数Instantiate 内では特別な変数selfがこれを受け、selfが指すクラスにインスタンスオブジェクトを創成し、そのオブジェクト（識別子）を返す。

‘newpart.ComposedOf += { newcomp} ’ は集合値属性に値を追加する操作の構文甘味剤である。意味は多値属性ComposedOfの値であるComposition オブジェクトの集合にnewcomp に格納されているオブジェクトを加える(Append)ことである。

図３や図４に示されたJasmine/C プログラムの書法上の特徴は永続的オブジェクトの扱い方の記法がＣ言語の構造体の記法と極めて類似していることである。しかも、構造体と違って、関数を属性として使うことが可能である。この両者間の類似はプログラマの負担を軽減する。なお、‘end_transaction’はトランザクションの終わりを意味し、これが実行されてオブジェクトベース中のオブジェクトの変更が実施される。

一方、タスク４は複合部品の登録業務であるからCompositePart クラスの動的属性として定義するのが最も直接的にその意味を表現しているだろう。しかし、注意すべきはタスク４はCompositePart に属するオブジェクトの属性ではないことである。むしろ、クラスオブジェクトとしてのCompositePart の属性であると考えられる。この種のクラスの動的属性はクラスメソッドで実現できる。task4 をNewCompositePartと名前を変え、それをクラスCompositePart の動的属性として定義すれば、‘CompositePart.NewCompositePart()’でタスク４を実行できる。

5．　一貫性制御

一貫性制御はデーモンによって行う。デーモンはフレームの付加手続きと呼ばれるもので、スロットの値を監視する。Jasmine では値の更新、追加、削除（後の２つは集合値の操作）等がある。また、値が未定義の場合にその値が求められた時に起動されるif_neededデーモンも書ける。本節では、Parts オブジェクトベースを例に取りJasmine における一貫性制御機能の一部を説明する。

部品の属性 Pno（図１参照）は関係データベースで実現するとすればキー属性である。OID でその替わりをさせることが可能だが、利用者間のコミュニケーションのためには一意的番号は必要であろう。そのためここでは Pnoを導入してある。この Pnoについては「すべての部品は部品番号を持ち、かつそれは一意である」という制約が考えられる。この制約の前半は属性 Pnoの定義属性 ‘mandatory’を指定すれば充足される（図５参照）。この指定はInstantiate 時に Pnoの属性値を指定することを強制する。一方、後者の制約はconstraintデーモンにより実現できる。constraintデーモンはそれが書かれた属性の値の初期設定や変更時に起動され、trueかfalse を返す。もしfalse が返されると初期設定や変更は実施されない。

Jasmine では下位のクラスに属さないオブジェクトの存在を許す。例えばCompositePart でもなく、

BasePartでもない部品をPartクラスのインスタンスオブジェクトとして創成することは可能である。しかし、「部品はすべて複合部品か基本部品であるべきだ」と言う制約は自然である。これはPartクラスのインスタンスオブジェクト創成を禁止することにより実現できる。そのためには、Instantiate メソッドと同じ名前の継承不可クラスメソッドをPartクラスに定義し、そのメソッドでオブジェクト創成要求を無視すればよい。このInstantiate メソッドは継承不可なのでPartの下位クラスは継承しない（継承されるのは継承可のInstantiate である）。

新しい複合部品の登録にはタスク４で行うことは既に述べた。この時、構成部品は既存の部品でなければならない。このような制約はタスク４中に記述してある。また、タスク４はクラスCompositePart の動的属性NewCompositePartとして実現するのが望ましいことは既に論じた。複合部品登録に於ける意味的制約を表現するのにはこの立場に立つ方がより相応しい。

Jasmine ではフレームに許されているようなデーモンを書くことが出来る。従って、フレームベースの知識表現言語で可能なことはすべて可能である。本節では最後に能動データベース[12]の観点から典型的なデーモンの応用例を示す。

今、Parts オブジェクトベースに在庫管理機能をもたせたいとする。そのためにはPartクラスに静的属性Stock を追加する。この属性は部品の在庫量を示すものとする。この在庫量がある量以下になると、もしそれが基本部品ならば業者に発注し、もし複合部品ならば工場に製造を指示するものとする。そのためには属性在庫量の変化を見守り、適当な時期に起動されるデーモンを属性Stock に付与すればよい。図６に属性の定義例を示す。この属性はBasePart, CompositePart の両クラスに継承される。勿論それぞれのクラスにStock 属性を定義してよい。その場合には当該オブジェクトのクラス判定は不要になるので定義は簡単になる。発注伝票発行や製造指示伝票発行は又それぞれのクラスが定義され、そのクラスに付与された動的属性を呼び出すことにより行う。但し、本稿ではそれらについてはこれ以上詳説しない。

## ６．結論

Jasmine/C はＣ言語に基づいた、永続的オブジェクトを扱うオブジェクトベースプログラミング言語である。この言語を使えばプログラマーは従来プログラミング言語に統合されていなかった「外部記憶」上の大量データを内部記憶領域上に展開されるデータと全く同様に扱うことができる。このことはプログラミングの概念ばかりでなく、その環境をも大きく変えるであろう。

Jasmine/C の永続的オブジェクトはJasmine で管理される。Jasmine は関数データモデルに基づくオブジェクトベースシステムである。その特長は

１）数値、文字列、イメージ、グラフィックスばかりでなく手続き（プログラム）も管理する。

２）Ｃ言語を含むオブジェクトベースプログラミング言語Jasmine/C 。

３）Jasmine/C で定義される動的属性。

４）オブジェクトの条件による集合論的同定

である。

Jasmine とJasmine/C とは相互依存関係にあり、一方を切り離すことは出来ない。これがオブジェクト指向パラダイムによるオブジェクトベースの大きな特徴である。

Jasmine/C は現在プリプロセッサ方式で開発されている。従って、Jasmine/C 言語特有の文には'$' が付けられ、Ｃ言語文と区別される。しかし、本稿では見易さのためにその記号は省いた（近い将来 '$' 無しのJasmine/C を開発する予定である）。

Jasmine にはインタラクティブ言語Jasmine/I が有り、端末からの対話的利用も可能である。Jasmine/I を使って、動的属性の定義、コンパイル、実行が可能である。これによりオブジェクトベースの構築が容易になる。

Jasmine はSUN ワークステーションのUNIXのもとC言語で開発された。現在、本システムはβテストを受けている。

謝辞

本稿で述べたJasmine モデルとJasmine/C 言語の仕様は主に筆者達の考案になるものであるが、システムの開発には他に泉田、山根、青島、成田、小桜の諸氏が参加した。開発中種々の仕様改良がなされた。これら貢献に対して記して感謝の意を表したい。又、研究開発の機会と発表の機会を与えてくれた情報処理研究部門長棚橋氏に感謝する。なお本研究の一部は通産省工業技術院大型プロジェクト「電子計算機相互運用データベースシステムの研究開発」の一環として行われた。

参考文献

1) Astraham, M. et al. System R: A relational approach to database management. ACM Trans. Database System. Vol.1, No.2, June 1976, pp.97-137

2) Atkinson, M.P. and Buneman, O.P. Types and persistence in database programming languages, ACM Computing Surveys, Vol.19, No.2, June 1987, pp.105-190

3) Banejee, J. et all. Data model issues for object-oriented applications. ACM Trans. on Office Information Systems. April 1987

4) Bloom, T. and Zdonik, S.B. Issues in the designof object-oriented database programming

languages. Proc. of OOPSLA'87, October 1987, pp. 441-451
5) Copeland, G. and Maier, D. Making Smalltalk a database system. Proc. of 1984 ACM-SIGMOD International Conference on Management of Data, June 1984, pp. 316-325
6) Fikes, R. and Kehler, T. The role of frame-based representation in reasoning. C. ACM, Vol. 28, No. 9, 1985, pp. 904-920
7) Fishman, D. H. et al. Iris: An object-oriented database management system. ACM Trans. on Office Information Systems, Vol. 5, No. 1, January 1987, pp. 48-69
8) 二村良彦 他 特集：新しいプログラミング・パラダイムによる共通問題の設計. 情報処理学会 第26巻第5号, 1985年
9) Goldberg, A. and Robson, D. SMALLTALK-80 The language and its implementation. Addison-Wesley, Reading, Mass., 1983
10) Kernigham, B. W. and Richi, D. M. The C programming language. Printice-Hall, London, 1978
11) 牧之内顕文 他 関係データベースシステムを中核とした計画管理情報システム. 情報処理学会論文誌第25巻、第1号、昭和59年
12) Morgentern, M. Active databases as a paradigm for enhanced computing environments. Proc. of the 9th International Conference on VLDB, 1983, pp. 34-42
13) Minsky, M. A framework for representing knowledge. In the psychology of computer vision, McGraw-Hill by Winston, P. H. (Ed.)
14) 西尾章治郎、楠見雄規 演繹データベースにおける再帰的問い合わせ評価法. 情報処理 Vol. 29, No. 3, 1988年, pp. 240-255
15) Rowe, L. A. and Stonebraker, M. R. The POSTGRES data model. Proc. of the 13th International Conference on VLDB, 1987, pp. 83-95
16) Shipman, D. W. The functional data model and the data language DAPLEX. ACM Trans. on Database Systems, Vol. 6, No. 1, March 1981, pp. 140-173
17) Smith, J. M. and Smith, D. C. P. Database abstraction: Aggregation and generalization. ACM Trans. on Database Systems, Vol. 2, No. 2, June 1977, pp. 105-133
18) Stefik, M. and Bobrow, D. G. Object-oriented programming: Themes and variations. The AI Magagine, Vol. 6, No. 4, 1986, pp. 40-62
19) Stefik, M., Bobrow, D. G., and Kahn, K. M. Integrating access-oriented programming into a maltiparadigm environment. IEEE Software, Vol. 3, No. 1, January 1986, pp. 10-18
20) Stonebraker, M., Anton, J., and Hanson, E. Extending a database system with procedure. ACM Trans. on Database Systems, Vol. 12, No. 3, September 1987, pp. 350-376
21) Stroustrup, B. An overview of C . Sigplan Notice, Vol. 21, No. 10, October 1986, pp. 7-18
22) Winston, P. H. and Horn, B. K. P. LISP. Addison-Wesley, Reading, Mass., 1981
23) Woelk, D. and Kim, W. Multimedia information management in an object-oriented database system. Proc. of the 13th International Conference on VLDB, 1987, pp. 319-329
24) 横田一正 新しいプログラミングパラダイム14：レコードプログラミング bit 12, 1988, 共立出版, pp. 83-92

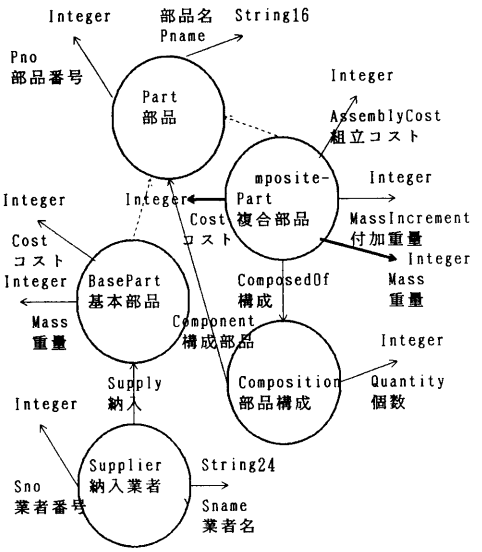


図1. Parts(部品) オブジェクトベースの構成
丸はクラスを示す。但し、Integer, stringは自己参照型クラスで丸を省いた。細い実線の矢印は静的属性を、太いのは動的属性を表す。

```
instantiate_class Part {
   kb Parts
      comment "部品オブジェクトベース"
   super Composite
      comment "上位クラスは Composite"
   property Int Pno
      comment "部品番号"
      String16 Pname
        comment "部品名　最大長は16バイト" } ;
instantiate_class BasePart {
   kb Parts
   super Part
      comment "上位クラスは Part"
   property Int Cost
      comment "購入コスト"
      Int Mass
        comment "重量"  } ;
instantiate_class CompositePart  {
   kb Parts
   super Part
      comment "上位クラスは Composite"
   property Int AsseblyCost
      comment "組立コスト"
      Int MassIncrement
        comment "付加重量"
   instance_method Int Cost
      { ...総コストを求める手続き... }
      comment "図３に内容掲載"
      Int Mass  { ...総重量を求める手続き... }
        comment "総コストを求める手続きと
           同類なので省略" } } ;
```

図２．オブジェクトベース Partsのクラス創成。
クラスSupplier, Composition についての
記述は省略。但し、それらクラスの上位
クラスはComposite である。

```
Int Cost()
  {Int cost, subcost;
   Composition cmpts multi,
               cmpt;
   Scan scan;
   Part part;
   cost=0; /*Costを0 で初期化*/

   cmpts=self.ComposedOf;
     /*当該複合部品の部品構成集合を求める*/
   scan=cmpts.Openscan();
     /*scanは最初の部品構成を指す*/
   while(scan)  {
     /*全ての構成部品のコストを求め加える*/
     cmpt=cmpts.Next();
      /*scan が指す部品構成を取り出しscanを進める*/
     part=cmpt.Component;
      /* 当該部品構成中の構成部品を取り出す*/
     if (part.Class == <BasePart> )
        /* 構成部品は基本部品か？*/
       subcost=part.Cost*cmpt.Quantity;
         /*コスト= 基本部品コスト* 個数*/
     else
       subcost=part.Cost()*cmpt.Quantity;
         /*コスト= 複合部品コスト* 個数*/
     cost=cost+subcost;
        /* 今求めた構成部品のコストを加える*/
     }
  cost+=self.AssemblyCost;
      /*組立コストもコストに含める*/
  return(cost); }
```

図３．クラスCompositePart の動的属性Costの定義。
再帰呼び出しにより構成部品の総コストの
計算が容易に出来る。

```
void task4() {
 Int partnum, quant, asscost, incrmass;
 String16 partname;
 Part newpart, cmpntpart
 Composition newcomp;
   ┌ ユーザとの対話部分 ──────────┐
   │新しい複合部品の部品番号、部品名 │
   │組立コスト、付加重量を入力させ  │
   │それらを各々 partnum, partname,  │
   │asscost, incrmass に格納する     │
   └──────────────────┘

 begin_transaction;
  newpart=CompositePart.
          Instantiate(Pno partnum);
    /*クラスCompositePart に
      与えられた部品番号を持つ
      オブジェクトを創成し
      newpartpに返す*/
  newpart.AssemblyCost=asscost;
    /*当該複合部品の組立コスト、*/
  newpart.IncrementalMass=incrmass;
    /*付加重量を代入  */
  newpart.Pname=partname;
  partnum=0;
    /* 0  は部品番号として有りえない数 */
   ┌ ユーザとの対話部分 ──────────┐
   │当該複合部品を構成する構成部品  │
   │の部品番号、個数をそれぞれpartnum,│
   │ quantに入力させる              │
   └──────────────────┘

  while ( partnum !== 0)
  /*partnum が0になった時、
    もう構成部品はない*/
   {  cmpntpart=Part where Pno == partnum;
      if (cmpntpart==Void)
      {        /* 既存部品でない*/

       ┌──────────────┐
       │誤り処理。メッセージを出力し  │
       │エラー出口に飛ぶ-             │
       └──────────────┘
                                    }
    newcomp=Composition.Instantiate();
    /*クラスComposition にインスタンス*/
    /*オブジェクトを創成する */
    newcomp.Component=cmpntpart;
    newcomp.Quantity=quant;
    newpart.ComposedOf+= { newcomp} ;
     ┌ ユーザとの対話部分 ──────────┐
     │当該複合部品を構成する構成部品の │
     │部品番号、個数をそれぞれpartnum,  │
     │ quantに入力させる              │
     └──────────────┘

   }
 end_transaction;}
```

図４．タスク４のプログラムtask4 。
誤り処理などは簡単化してある。

```
        .
        .
        .
property Int Pno mandatry
   constraint  {
     Part parts multi;
     parts=Part where Part.Pno == value;
     if (parts ==  {} )/* 空集合 {} は
                          Voidと同義*/
      return(TRUE);
     else return(FALSE); }

        .
        .
        .
```

図5．Partの属性Pno に関する制約。Pno の値は必ず創成時に値を与えなくてはならない。又、既存の値であってはならない。

```
        .
        .
        .
property Int Stock
  if__updated  {
        /* updatedデーモンの定義*/
    if (self.Class == <BasePart>
       /*基本部品か*/ → Stock
       &&  self.~~storage~~<=20 )
         /*在庫量が20以下か*/
      {
          発注伝票発行
                                  }
    else if (self.Class == <CompositePart>
       /*複合部品か*/      → Stock
            &&  self.~~storage~~ <= 30 )
            /* 在庫量は30以下か*/
          {
            製造指示伝票発行
                                  }
    }
```

図6．クラス Part の Stock属性。その値（在庫量）が一定値以下になると適当なアクションがとられる。