データベース・システム 85－6
プログラミング－言語・基礎・実践－ 4－6
（1991. 10. 4）

持続的オブジェクトの計算モデルについて

箕原 辰夫　　所 真理雄

慶應義塾大学大学院計算機科学専攻

我々は、**持続的情報システム**と呼ばれる、データベースシステムとプログラミング言語が融合された総合的なシステムを構築しようとしている。そのような総合的なシステムにおいては、プログラミング言語で書かれるような通常の計算の他に、型検査や検索処理などの付加的な計算が行なわれている。我々は、持続的情報システムを構築するために持続的オブジェクトを用いる。しかし、すべての付加的な計算機構を内包するような複雑な持続的オブジェクトは、もはや要求されてはいない。そのような複雑なオブジェクトの代わりに、ユーザが望む計算機構を持つようなオブジェクトを定義できるようにすることが求められている。この論文では、持続的オブジェクトの計算モデルについて提案する。このモデルでは、**連接**と呼ばれる、計算機構が付加されたリンクから持続的オブジェクトが構築されている。

A COMPUTATIONAL MODEL FOR PERSISTENT OBJECTS

Tatsuo Minohara　　Mario Tokoro

Department of Computer Science, Keio University

3-14-1, Hiyoshi, Kohoku-ku, Yokohama, 223, Japan

We are constructing a total system called *persistent information system*, which integrates database management systems and programming language systems. In such an integrated system, various auxiliary computations, such as type checking, constraint solving, exception handling, and query processing, are required to be expressed systematically, in addition to ordinary computations expressed by programming languages. A complicated object which includes all auxiliary computations is no longer needed. Instead of the complicated object, the computation mechanisms of objects should be defined by the user semantically. In this paper, we proposes a computational model. In the model, an object is constructed from multiple computational links called *associations*. Therefore, an object has a different computational mechanism.

1 導入

持続的情報を扱うアプリケーションプログラムは、様々な問題に直面している。まず、言語の問題として、通常のプログラミング言語は、検索言語ほどには大量のデータを扱うほど強力な機構によってサポートされていない。また、検索言語はプログラム言語ほどには柔軟にデータを制御することはできないからである。我々は、プログラミング言語と検索言語を融合させることを考えている。その他に、持続的情報は、それ自体が様々な特性を持っている。我々は、次のように持続的情報をこの論文の中で定義する。

> **持続的情報** (Persistent Information) とは、活性で、長い間に渡って、複数のアプリケーションプログラムによって並行にあるいは時間をおいて共有される情報である。

持続的情報は、受動的なデータとその解釈の両方を意味している。1つの操作がある情報の解釈に適用された結果として、内部で計算が行なわれる。そのため、持続的情報は活性であると考えることができる。例えば、オブジェクト指向データベース [Mo86] の中に格納されている値とメソッドは、それぞれ受動的なデータとその解釈とを表現している。

このような持続的情報を柔軟にかつ効率よく処理するために、データベースプログラミング言語が提案されてきた [AB87, BB90]。我々は、この論文で、持続的情報およびプログラミング言語と検索言語を融合させた言語のための1つのモデルを提案する。

このモデルを提案した目的は、意味データモデル [HK87, PM88] の利点とオブジェクト指向パラダイムの利点とを融合することにあった。概して言えば、実体間の関連の意味というものは、意味データモデルにおいては明示的に表現することができる。しかしながら、意味データモデルにおいて実体と関連は、データベースのスキーマを定義するために用いられてきたので、個々の実体間の特別な関係というものを記述することはできなかった。それ以上に、実体が持つ手続き的な性質は、意味データモデルにおいては表現することができなかった。一方、オブジェクト指向パラダイムにおけるオブジェクトは、個々に他のオブジェクトを指すことができた。また、オブジェクトの中に手続き的な性質を持つこともできた。しかしながら、オブジェクトが持つポインタの意味というものは、明示的には記述されず、システムによっても管理されてはいない。ポインタの意味、つまり何故それを指しているかということは、システムによって明示的に独立に管理されいなければならない。ここで、「管理」という言葉の意味には、例外や制約の管理も含まれる。

もう1つの目的は、計算機構を明確に表現することであった。それは、持続的オブジェクトを持つシステムのユーザにとって、計算機構そのものが使用のための有効な情報となるからである。計算機構を表現するために、メタレベルのアーキテクチャにおいては、主に次のような2つの実現方法があった [MT90a]。

- Macs[Mae87] の方法では、オブジェトの解釈は、メタレベルにある1つのオブジェクトによって記述された。
- Muse オペレーティングシステムでは、オブジェクトの解釈は、メタレベルにある複数のオブジェクトによって、そのオブジェクトへの環境を提供するという形で実現されている [YTT89]。

多様な計算のためのメタレベルアーキテクチャを構成することが計算機構を明確にすることで初めて可能となるのである。

この論文では、持続的オブジェクトのための新しい計算モデルを提案する。第2章では、このモデルの既念をその背景と共に紹介する。第3章では、このモデルの主要な構成要素となっている連接の構造とその分類、および実行形態について説明する。第3章以降では、このモデルを記述するための記法を導入する

が、これはモデルの説明のためのものであって、この論文の主題ではないことに注意されたい。第4章では、持続性を名前付けの機構によって定義し、持続的オブジェクトを連接から構成する。第5章では、我々のモデルと今までのデータモデルとを比較する。第6章で、将来の研究方針について触れながら、この論文の結論を呈示する。

2 概念

我々は、異なる計算機構を持つオブジェクトを表現するためのモデルを提案する。このモデルの原型は、[MT90b] にある。。このモデルは、持続的オブジェクトだけではなく、一般のオブジェクトモデルとしても新しい概念を含んでいる。ここでは、それらの概念をその足跡と共に簡単に呈示する。

このモデルでは、1つのオブジェクトを、複数の存在を表現するための実体 (entities) と、オブジェクトの機能あるいは実体間の意味的な関連を表現するための連接 (associations) とに分解する。ここで、1つの実体がそのオブジェクト自身を示すものとなっている。実体と様々な連接を組み合わせることによって、異なる意味・機能を持つオブジェクトを構成することができる。

実体の広義の意味では、すべての持続的情報の中に含まれているものは、実体である。例えば、名前や連接もまた実体である。しかし、狭義の意味では、名前や連接を除いたものを実体と呼ぶ。この狭義の実体は、それに対して結びつけられている連接によって解釈される。例えば、値 23 が持続的情報の中に表現されていたとしても、それ自身は何の解釈も示していない。それが整数の1つであると解釈されるのは、値 23 が持つ連接が整数の性質を示している場合である。連接の種類によっては、それは機械のインストラクションであったり、文字であったりする。原始的な値でさえ、連接がなければ、その解釈をすることができない。名前と連接は、狭義の意味では実体から外されていると定義したが、それは名前や連接自体は特別な実体であり、これらの解釈はモデルの中で既に予め定められているからである。

我々はまた、このモデルにおいて、オブジェクトの計算機構というものも明示的に表現することを考えた。しかし、計算機構を示すメタレベルアーキテクチャには、いろいろなものがあり、複数のアーキテクチャを1つのシステムの中で表現することが必要であった。これらの機構を意味的に記述し、同一のシステムの中で複数のメタレベルアーキテクチャを構成することができるようにするために、我々はメタレベルさえも次のように連接によって構成するようにした；通常レベルにあるオブジェクトとメタレベルにあるオブジェクトを結ぶ連接が前者のオブジェクトの計算機構とする。もし、そのオブジェクトに別の計算機構が必要となれば、メタレベルの他のオブジェクトに結び付けるように連接を変更すればよい。このように、通常のレベルとメタレベルのオブジェクトを結ぶための多種の連接を用意することによって、メタレベルのアーキテクチャの構成を柔軟に変えることもでき、また同時に複数持つこともできる。このようにして、計算機構の意味的な表現も連接を使うことによって可能とした。

そこで、計算システムにおける様々な活動を概念的に整理された形で観測するために、このモデルでは次のような3つの側面を呈示することとした。これらは、あくまでも側面であって、分類ではない。つまり、同一の計算活動が3つ側面から観測されることを意味している。

- **実体** (Entity) この側面は、個々の存在および識別性を表現している。
- **連接** (Association) この側面は、実体間の関連性や実体の機能を表現している。
- **メタ** (Meta) この側面は、実体の持つ計算機構とメタレベルのアーキテクチャの構成を表現している。

3 連接

連接は実体間の関係を表現するために用いられる。実体がグラフ上でのノードとして表すとすると、連接はその有向辺として表現することができる。連接は、また写像と考えることができ、写像の元となる実体から写像の先となる実体を結ぶものである。このような連接に対して、我々は次のような機能を求めた。

- 関係の意味の明示的な表現、
- ユーザが定義できる計算機構

つまり、連接は意味関係と関数の両方の機能を有するものとなっている。この節では、そのような機能を実現するための連接の内部構造と計算機構を説明する。次に、意味関係と関数のそれぞれの機能を表現するための部分連接と全連接を説明する。最後に連接を記述するための記法を呈示する。

3.1 連接の構造

連接は、次のような2つの構成物より成り立っている。

- **射** (Morphism): 1つの射は1つの関数を表現する。
- **個** (Individual): 1つの個は1つの有向リンクを表現する。

1つの連接は、射の集合と個の集合を持っている。ただし、後で述べるように全連接の場合は、射の集合のみから成り立っている。**連接射** (Association Morphism) とは、連接の射の集合のことを指す。**個別連接** (Individual Association) とは、1つの個のことを指す（これから特に必要がない限り、個のことを個別連接と呼んでいく）。それぞれの個別連接は、連接射によって制御される。1つの個別連接は、2つの実体を結合しているが、その元なる実体は**射の元** (source)、その先として個別連接によって結合されている実体は**射の先** (destination) と呼ばれる。よって、個別連接は3つ組 (a, s, d) で表現される。ここで、a は連接射を、s は射の元を、d は射の先を示している。

3.2 連接射とトリガー

連接射は、射の集合から成り、それが連接のセマンティックスを表現している。何かの操作が、個別連接に適用された場合は、それに対応する射が連接射の中から起動される。我々のモデルでデフォルトで定義されている射は次のような3つのものがある。

- *Creation*: 個別連接を生成する際に用いられる。
- *Evaluation*: 射の元から射の先に航行するのに個別連接が使われた際に用いられる。
- *Deletion*: 個別連接が削除される際に用いられる。

ただし、このモデルを実現しているシステムがごみ収集機能のような自動的な実体削除機能を用いているときは、削除のための射は必要ではない。例えば、ある実体からある実体へ1つの個別連接が張ってあり、その上を航行することを考えよう。これは、例えばハイパーテキストでリンクを航行する時や、あるオブジェクトの属性をそのオブジェクトからポインタを介して調べに行く時に対応する。この航行の際は、射の元となる実体が与えられ、評価用の射が起動される。その射が一連の仕事を終えた後、射の先である実体が最終的に評価される。

制約などの機構を表現するための実体など、通常の計算過程では記述できない異なる計算機構を持つオブジェクトを構築する際には、ユーザが様々な計算の局面で必要となる射を起動させるように表現できることが必要である。**トリガー** (Trigger) は、そのために用いられるものである。トリガーは、そのトリガーが

発火するための条件と発火した後呼ばれる射の名前の２つの記述から構成される。また、射が起動される時に渡されるべきパラメータを射の名前の記述と同時に指定することができる。先に述べたデフォルトの射も、該当する条件と受け渡されるべきパラメータがモデル内に仮定されている。トリガーは、既存のメタレベルアーキテクチャにおいて、メタオブジェクトが起動されるためのトラップとして考えることができ、実際のアーキテクチャにおいても効率的な実装をすることが可能である。トリガーと連接射をモデルの中に用意したことは、ユーザにインタープリタの機能を解放したことに等しい。

3.3 全連接と部分連接

連接には、**全連接** (Total Associations) と**部分連接** (Partial Associations) の２つ種類がある。全連接は関数を示すのに対して、部分連接は有限集合の中の実体間の関係を示す。全連接は、連接射のみから構成されている。全連接の場合、普通の関数と同様に射の元として受け渡された実体は、何らかの射の元なる実体に射影される。この意味で、全連接は全射とも考えることができる。全連接は、可算無限集合である定義域を同じく可算無限集合である値域に射影することができる。個別連接が表現するような個々のリンクは、連接射によって計算されるために個別連接が必要とないのである。簡単な全連接の場合、連接射は評価用の射だけから構成される。また、１つの射を１つの全連接と見なすことも可能である。

部分連接は、連接射と個別連接の両方を必要とする。部分連接の定義域と値域は持続的情報システムの中に存在する実体から成る有限集合でなければならない。すべての定義域に属する実体は値域内のある実体に射影される。定義域内、値域内のすべての実体は、少なくとも１つの個別連接によって結合されていなければならない。もし、射の元となる実体に対して、該当する個別連接が定義されていなければ、射の先となる実体は、その連接によっては求まらない。このように、個別連接は有向関係を示している。

全連接の評価は、関数の評価と等しい。射の先となる実体は、射の元となる実体に対して評価用の射が適用され、生成されるかあるいは既存の実体の中から選択される。前にも述べたように、部分連接の場合は、評価は個別連接上の航行に等しい。評価用の射が評価の最中に起動されるが、最終的には個別連接で結びつけられている射の先が評価される。

3.4 連接の記法

ここでは、連接の記法について例を用いて説明する。記法についての文法などは、付録を参照されたい。この記法は、連接を具体的に記述するための道具であって、我々のモデルの主題ではない。全連接は、先に述べたように連接射のみから構成され、関数と等しいものと考えることができる。よって、全連接の記法は関数の記法と似ている。下記の例において、連接演算子$\ggg$の左側にあるのが、連接への仮引数である。もし、それが省略されたら、引数が必要ない連接である。局所変数は、宣言演算子 | によって導入されている。仮引数も局所変数である。例えば、最初の例では局所変数 x が定義されており、仮引数に用いられている。この連接は、与えられた引数の後継を求めるものである。２番目の例では、連接は、$isAscend$ と名付けられている。この連接は、与えられた３つ引数が大きさが降順になっていれば真を返す。

$$(x \mid x \ggg succ(x)),$$
$$isAscend = (x,y,z \mid <x,y,z> \ggg and(isGreaterThan(x,y), isGreaterThan(y,z)))$$

評価 (evaluation) の記法は、連接を評価するために用いられる。括弧の中の式のリストは、連接への実引数つまり射の元である。前にも述べたように、全連接の場合は、その評価は関数の評価と等しい。部分連接の場合は、評価は射の元から射の先への航行として扱われる。射の元は、全連接と同様に引数として与えられる。よって、部分連接の場合も最終的には射の先が評価結果として返されてくる。もし、引数として

与えられた実体に対応する射が結びつけられていないときは、未定義を表す特別の実体 Nil が評価結果として返される。次の最初の方の例では、先ほどの後継を返す連接に引数として5が与えられている。また、2番目の例では $isAscend$ に3つの引数が渡されている。引数の連なりは、1つの組であると解釈する。3番目の例は、部分連接の評価の例であるが、$point1$ という名の実体に、$line$ と名付けられた連接（後で例として出てくる）の個別連接が結びつけられていれば、その射の先が返されてくる。

$$(x \mid x \ggg succ(x))(5),$$
$$isAscend(30,27,21),$$
$$line(point1)$$

個 (individual) の記法は、部分連接の個別連接を定義するために用いられる。次の例において、ある個別連接が定義されている。その連接の連接射の名前は $attribute$ であり、$entity1$ を $entity2$ に結びつけている。これは、意味としては、$entity2$ が $entity1$ の属性であるということを定義している。もし、$entity1, entity2$ のいずれかが省略された場合は、代わりにその環境を示す実体がその個別連接の射の元あるいは射の先として仮定される。

$$attribute[entity1 \ggg entity2]$$

環境で定義されている部分連接に加えて、ユーザは新しい部分連接を定義することが可能である。先に、部分連接の機能として連接射を定義すると述べ、その機能は個別連接の射を航行する時に評価されると述べた。これ以外に、部分連接の機能を表現するためには、個別連接を生成する時の手続き、制約など幾つかの手続き的な特性を表現する必要がある（最低限、航行と生成のための手続きは表現する必要はある）。これらの手続きな特性は、各々別個に射として表現され、部分連接の機能は、それらから成る1つのN-組として表される。以下の例においては、$line$ という名前の新しい部分連接が定義されている。これは、2つの射から成る組として定義されており、生成のための射においては、リンクが定義され、そのリンクには属性として、両端の点から計算された線の重心、線の長さを持つ。もう1つは、航行（評価）に使われる射である。その下の例は、連接 $line$ の個別連接を定義したものである。

$$\begin{aligned} line =< & creation = (a,x,y \mid < x,y > \ggg a =< link\_define(x,y,`line), \\ & \quad position := attribute[a \ggg div(add(x.`point(), y.`point()), 2)], \\ & \quad length := attribute[a \ggg absolute(minus(x.`point(), y.`point()))] >), \\ & evaluation = (x \mid x \ggg link\_navigate(x,`line)) > \\ a\_line = & line[point1 \ggg point2] \end{aligned}$$

最後に組の記法を導入する。組は、部分連接によって実現されている。1つの組は、それを示す実体と組の各列を示す実体および、それらの間をつなぐ個別連接によって実現されている。この記法では、直接、N-組を表現することができて、$< a_1, \ldots a_n >$ で表現される。各列は、名前が与えられていれば名前で、そうでなければ、順番を示す数で選択することができる。この記法は、既に上で使われているし、持続的オブジェクトを表現するのにも使われる。

4 持続的オブジェクトの構築

この節では、持続的オブジェクトの構築を、持続性を定義することと、オブジェクトを構築することによって説明する。

4.1 持続性と名前

持続的情報は、**持続空間** (persistent space) と呼ばれるシステムを表す単一の空間によって管理される [MT90b]。すべての活動は、持続空間の中で生起され処理される。実体の生存時間は、持続空間におい

て相対的なものである。例えば、応用プログラム内の大域変数は、持続空間の中でそのプログラムのプロセスの開始と共に作られ、その終了と同時に削除される。ある実体が持続的ならば、応用のプログラムのセッションの後でも生き残る。例えば、応用プログラムそれ自体のような持続的実体は、プロセスの終了時には削除されない。名前付けは、このモデルで実体を持続的にするための唯一の手段である [MT90b]。

名前付けは、名前定義あるいは名前代入のいずれかによって行なわれる。名前代入は $n = x$ と表される。これで、名前 n は実体 x に束縛され、以後、n で x を識別することができる。一方、名前代入は $n := e$ あるいは $n \leftarrow e$ と表現される。これは、式 e の評価の結果を示す実体を名前 n に束縛することを示している。名前定義が評価をしないのに対して、名前代入は、評価を行なうことに注意されたい。例えば、次の例では、*orange* は、連接 *attribute* そのものを示すのに対して、*grape* は実体 y を示す（個別連接の評価では、射の先が評価される）。

$$orange = attribute[x \ggg y],$$
$$grape := attribute[x \ggg y]$$

名前付けは、*name* と *bind* という 2 つの部分連接によって実現されている。ある実体 x が、環境 e において n という名前付けがされているとき、2 つの個別連接 $bind[n \ggg x]$ と $name[e \ggg n]$ が定義されている。環境というのは、ある実体が名前付けされている状態あるいは構造を意味しており、1 つの実体で表現されている。この 2 つの部分連接を用いて、**持続性の定義**を次のように行なうことができる；ある環境が持続的であれば、その環境下で名前付けされている実体は持続的である。根となる環境は、持続的情報システムそれ自身であるとする。もし、実体がどのような持続的な環境下においても識別されないならば、その実体はゴミ収集機構によって消去されるだろう。

4.2 オブジェクトの構築

1 つのオブジェクトは、実体と複数の連接とを組み合わせることによって構成される。基本的には、オブジェクトは他のアプローチと同様に、N-組で表現されるが、我々はオブジェクトのセマンティックスを表現するために、次のような部分連接を用意した。

- *attribute*: オブジェクトと、属性を示す実体とを結合する。
- *method*: オブジェクトと、メソッドを示す射とを結合する。
- *object*: 環境と、オブジェクトを示す実体とを結合する。

情報隠蔽のような、標準的なオブジェクトに要求されるような機能は、これらの部分連接の連接射の中に記述されている。これは、我々のモデルがオブジェクトのモデルを構築するための機構自体も表現できることの例証である。下の例は、あるオブジェクトを定義するためのスクリプトである。1 つのオブジェクトは、幾つかの連接から成る N-組をシステムに、*object* という名前の連接を用いて結び付けることによって表現されている。このオブジェクトには、*Tokyo_tower* という大域的な名前が付けられているので、持続的になっている。このオブジェクトは、3 つの属性と 1 つのメソッドによって、東京タワーを表現している。なお、*address* と名付けられた属性は、別のオブジェクトとなっていて、2 つの属性から構成される。

```
Tokyo_tower :=
    object[⋙<
        name := attribute[Tokyo_tower ⋙" Tokyo_tower"],
        address := attribute[Tokyo_tower ⋙
            object[⋙<
                city := attribute[address ⋙" Tokyo"],
                nation := attribute[address ⋙" Japan"]
            >]],          height := attribute[Tokyo_tower ⋙ meter(333)],
```

$$illumination := method[Tokyo\_tower \ggg (\ggg displayColorbox(red, div(height, 2)), displayColorbox(white, div(height, 2)))] >]$$

5 関連データモデル

我々のモデルは、先進的な意味データモデル [HK87, PM88] やオブジェクト指向データモデルを参照しながら構築された。ここで、我々のモデルとそれらの意味データモデルあるいは、オブジェクト指向データベースにおけるデータモデルとを比較することにする。

我々のモデルは、初期の意味データモデルである実体ー関連モデル (E-R モデル) [Ull88] と同じであると思えるかも知れない。しかしながら、E-R モデルは、値から構成されるデータのスキーマを記述するために提案されたものである。それゆえ、個々の実体ではなく、スキーマレベルでのモデル化しか可能ではない。我々のモデルでは、コードを含む活性な情報や特定の個々の実体間での関係の意味も連接を使うことによって表現することができる。

Iris[F*87] や関数データモデル [Shi81] もまた、我々のモデルの基礎となっており、E-R モデルと同様、我々のモデルと非常に似通っている。しかしこれらのモデルも、我々のモデルが個々の実体に基づいているのに対して、原則的にはスキーマレベルのモデル化しかできなかった。また個別連接は、これらのモデルのデータベース関数に非常に近いものである。しかし、我々のモデルでは例えば個別連接の評価の際に連接射の評価用の射が用いられるのに対して、これらのモデルではデータベース関数は単なる実体間の射影のためにしか用いられていない。Iris では、外部関数を使って、我々のモデルの連接射のようなものを実現することは可能であるが、残念なことに、外部関数は既に Iris のデータモデルの外側の話しになっている。

ORION[WKL86, B*87] のデータモデルは、我々が最も重要視したもので、本質的に我々のモデルと非常に近いものとなっている。例えば、我々のモデルの *attribute* や *method* のような連接に関しては、ORION のデータモデルでも同様のリンクが定義されている。オブジェクトは、実体とそれらのリンクを組み合わせることによって表現される。しかしながら、ORION ではリンクの種類がシステムによって固定されていた。新しい機構を持ったリンクは、ユーザによって定義することはできなかった。

最後に、O_2[L*88] のデータモデルでは、個々のオブジェクトを中心としたモデル化が行なわれている。我々も個々の実体に注目してきた。このようなモデル化の仕方は、これからのデータモデルの将来の方向を示唆しているように思われる。しかしながら、O_2では、我々のモデルとは違って、リンクの概念は明示的には現れてこない。

6 結論

我々の持続的オブジェクトのためのモデルでは、通常のオブジェクトよりも更に細かな原始的な実体レベルからオブジェクトを構築することができる。オブジェクトは、複数の実体および連接から構成され、その中の１つの実体はそのオブジェクト自身を表現するのに用いられた。連接は、連接射と個別連接（部分連接の場合）から構成され、連接射がその連接の計算機構を表現しており、それは計算の各局面で呼び出される。例えば、連接が示す写像を評価する場合、連接射の中の評価用の射が、その途中で起動される。このような、独自の計算機構を持つ連接と実体を組み合わせることにより、複雑な計算機構を持つオブジェクトを表現することが可能となった。

この計算モデル上での動的な視点 (View) の構築および型仕様の表現に関する議論は、[MT91] で行なわれている。我々は、この計算モデルに対してより形式的な表現を与え、その上で計算可能性を議論することを計画している。この議論は、関数計算モデルとの比較が中心となる可能性が高い。その成果は、我々の次の論文として表出させる予定である。

謝辞

本田耕平氏は、我々の連接の概念が非常に本質的であり、この概念が既存のデータモデルとは別の方向にいく可能性を秘めたものであると最初に認識した。ICOT 次世代データベースワーキンググループのメンバーとの議論は、我々の研究を向上させるのに助けとなった。特に、田中克巳先生、吉川正俊先生から有益な示唆を戴いた。嶋田貴夫氏には、引用文献の整理をして戴いた。吉田展子氏には、論文の最終的なチェックをして戴いた。

参考文献

[AB87] M. P. Atkinson and P. Buneman. "Types and Persistence in Database Programming Languages". *ACM Computing Surveys*, 19(2), 1987.

[ABD*89] M. Atkinson, F. Bancilhon, D. DeWitt, K.Dittrich, D. Maier, and S. Zdonik. "The Object-Oriented Database System Manifesto". In *Proceedings of DOOD*, 1989.

[B*87] J. Banerjee et al. "Data Model Issues for Object-Oriented Applications,". *ACM TOOIS*, 5(1), 1987.

[BB90] F. Bancilhon and P. Buneman, editors. *"Advances in Database Programming Languages"*. ACM Press, 1990.

[Cod70] E. F. Codd. "A relational model of data for large shared data banks". *Comm. ACM*, 13, 1970.

[Dat86] C. J. Date. *"An Introduction to Database Systems"*. Volume 1,2, Addison-Wesley, 4 edition, 1986.

[F*87] D. H. Fishman et al. "Iris: An Object-Oriented Database Management System". *ACM TOOIS*, 5(1), 1987.

[GR83] A. Goldberg and D. Robson. *"Smalltalk-80: The language and its implementation"*. Addison-Wesley, 1983.

[HK87] R. Hull and R. King. "Semantic Database Modeling: Survey, Applications, and Research Issues". *ACM Computing Surveys*, 19(3), 1987.

[KMP84] G. Kahn, D.B. MacQueen, and G. Plotkin, editors. *Semantics of Data Types*, chapter "Semantics of Data Types". Volume 173, Springer-Verlag, 1984.

[L*88] C. Lecluse et al. "O_2, an Object-Oriented Data Model". In *Proceedings of ACM SIGMOD*, 1988.

[Mae87] P. Maes. "Concepts and Experiments in Computational Reflection". In *Proceedings of ACM OOPSLA*, 1987.

[Mo86] D. Maier and ohters. "Development of an Object-Oriented DBMS". In *Proceedings of ACM OOPSLA*, 1986.

[MT90a] T. Minohara and M. Tokoro. "Multiple meta-objects support an object,". In *ECOOP / OOPSLA '90 Workshop on Reflection and Metalevel Architectures*, 1990.

[MT90b] T. Minohara and M. Tokoro. "MyAO: A Model for Expressing Persistent Objects". In *Workshop Object Oriented Computing*, 1990.

[MT91] T. Minohara and M. Tokoro. "Providing Dynamic Type Abstractions and Specifications for Persistent Information,". In *DOOD '91 to be appeared (LNCS)*, 1991.

[PM88] J. Peckham and F. Maryanski. "Semantic Data Models". *ACM Computing Surveys*, 20(3), 1988.

[Shi81] D. Shipman. "The functional data model and the data language DAPLEX". *ACM TODS*, 6(1), 1981.

[Ull88] J. D. Ullman. *"Principles of Database and Knowledge Base Systems"*. Volume 1, Computer Science Press, 1988.

[WKL86] D. Woelk, W. Kim, and W. Luther. "An Object-Oriented Approach to Multimedia Databases". In *Proceedings of ACM SIGMOD*, 1986.

[YTT89] Y. Yokote, F. Teraoka, and M. Tokoro. "A Reflective Architecture for the Object-Oriented Distributed Operating System". In *Proceedings of ECOOP-89*, 1989.

付録. 言語構文

メタ記号

次のようなメタ記号が言語の記法の構文を定義するのに用いられる。

::=	この記号の左側にある非終端記号は、この記号の右側の記法で置き換えられる。
|	選択
[]	省略可能
{}	括弧の中の式は、0回以上の繰り返しが可能

識別子と定数

識別子には2つの種類がある。一方は、識別子変数と呼ばれており、もう一方は、識別子定数と呼ばれている。両者の違いは、前者にはある実体が代入され、それを表示するに対し、後者は名前そのものを表示するために使われる。普通は、識別子変数は局所変数として使われ、識別子定数は実体の集まりの中から実体を選ぶの用いられる。

sample	識別子変数
'sample	識別子定数

現在3つの種類の定数がある：整数と、文字と、文字列である。文字の場合は、#が文字に先行し、文字列の場合は二重引用符"が用いられる。

$+123, -256$	整数定数
$\#c, \#\%, \#X$	文字定数
"aStringforexample"	文字列定数

式

記述は、式の列として構成される。式の記法は、以下の通りである。

Expression ::= *Definition* | *Evaluation*
| *Individual* | *Association*
| *Qualification* | *Construction*
| *Variants* | *Identifier*
| *Constant* | *Identification*
| *Specification* | *Assignment*
| '('*Expression*')'

Expression_List ::= *Expression*{,*Expression*}

Assignment ::= *Expression*' :=' *Expression*
Association ::= '('[*Declaration*][*Expression*] ⋙ *Expression*')'
Construction ::= ' <' *Description*' >'
Declaration ::= *Expression_List*'—'
Definition ::= *Expression*' =' *Expression*
Description ::= [[*Declration*]*Expression_List*]
Evaluation ::= *Expression*'('[*Expression*], *Expression*')'
Identification ::= *Expression*'.'*Expression*
| *Expression*' !' *Expression*
Individual ::= *expression*'['[*Expression*]' ⋙' *Expression*']'
Specification ::= *Expression*' :' *Expression*
Qualification ::= '{'*Description*'}'
Variants ::= '['*Description*']'