ICOT Technical Report: TR-638

TR-638

# オブジェクト指向データベース・プログラミング言語

横田　一正、森田　幸伯、
宮崎　収兄（沖）

April, 1991



ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

# オブジェクト指向データベース・プログラミング言語†

横田一正†† 森田幸伯††† 宮崎収兄†††

## 1. はじめに

この数年，データベース (DB) の世界にいくつかの大きな変化が起こりつつある．次世代 DB をめぐって，さまざまな新しいデータモデルが提唱される一方で，DB のいくつかの面で，プログラミング言語 (PL) や人工知能 (AI) の世界と密接な関係が築かれつつある．本稿ではその一つである，DB と PL の統合をめざすデータベース・プログラミング言語 (DBPL) を概説する．本稿では特に，新しいデータモデルの一つとしてのオブジェクト指向データベース (OODB) との関連から，オブジェクト指向 DBPL (OODBPL) に焦点を絞る．

まず次章で，DBPL の特徴を中心に検討する．それ以降の章では，その中でもオブジェクト指向 (OO) 概念を取り入れた言語に的を絞り，OO 言語，関数型言語，論理型言語の三つのプログラミング・パラダイムに基づいた主要な OODBPL を説明する．

## 2. OODBPL の背景と分類

### 2.1 DBPL の特徴

データベース言語 (DBL) と PL とはほとんど独立に発展してきた．その結果，DB を使用する応用のプログラマは DBL と PL の二つの言語を知る必要がある．さらに二つの言語の間に型の不整合があれば，その整合性は応用側の責任となっている．そこでこれら二つの言語の統合化が DBPL として叫ばれ始めている．また PL の世界でも，PL で扱う任意のデータの永続化を可能にするための永続的 (persistent) PL (PPL) が検討されている．これらに焦点を当てた国際会議 (DBPL 国際ワークショップや永続オブジェクトシステム国際ワークショップなど) もほぼ定期的に開催され，幅広い議論が展開されている[4),5),10)]．

DBPL は一言でいうと，DBL と PL を一つの言語として結合したものである．DBL での表現単位を「オブジェクト」と呼ぶと，DB システムの要件は，「オブジェクト」の要件として以下のようにまとめることができる．

1) 「オブジェクト」は実世界のデータを素直に表現する．

2) 「オブジェクト」は永続性をもちうる．

3) 「オブジェクト」は共有・同時実行 (concurrency) が可能である．

1) はデータモデルの概念に対応しており，伝統的な用語でのデータ定義言語 (DDL) とデータ操作言語 (DML) の両者と，一貫性制約の記述も含んでいる．この「オブジェクト」は，これまで使われてきた関係 (relation) や組 (tuple)，あるいは複合オブジェクト (complex object) などの DB オブジェクトと上位互換であることが望まれる．

2) の永続性とは，データの（トランザクション，プログラム，ジョブ，システム再生成などと関連した）ライフタイム（生存期間）の性質である．つまり，あるデータが一時的なものなのか，プログラムが終了しても消失しないものなのか，などの生存期間を「オブジェクト」の性質としてもたせることである．永続性をもった PL の発展過程としては

- PL と DB システムとのインタフェースの確立
- 揮発（一時）データと永続データの 2 種類の区別

† Object-Oriented Database Programming Languages by Kazumasa YOKOTA (Institute for New Generation Computer Technology), Yukihiro MORITA and Nobuyoshi MIYAZAKI (Oki Electric Industry Co., Ltd.).

†† (財)新世代コンピュータ技術開発機構第3研究室
††† 沖電気工業(株)総合システム研究所

- 任意のオブジェクトへの任意期間の永続性の付与

の三つの段階が考えられる．最初の点は，(PL とファイル・システムのインタフェースのように) 低レベルの表現への変換を必要としないこと，あるいは DB の集合の扱いを PL でサポートすること，などを意味している．2番目は現在最もよく使われている定義であり3番目は永続性の考え方をさらに発展させたものである．後の二つはまた，単なる問合せだけではなく，データの更新可能性を含意している．この永続性の概念は，共有性を実現する上でも必須のものである．

3) は 2) とは独立の概念で，言語仕様というよりむしろその実装に多く関係しているが，

DBPL＝DBL＋PL

というだけではなく，

DBPL システム
＝DBL システム＋PL システム

と考えれば十分に考慮しておくべき基本的な性質である．

### 2.2 DBPL へのアプローチ

DBPL を実現するためには

1) DBL を計算完備な言語に拡張する．

2) PL に 2.1 の「オブジェクト」の要件をもたせる．

の二つが考えられるが，DBL を PL として考えると DBL の機能が少ないせいか，前者のアプローチは非常に少ない．DBPL への多くのアプローチは後者の立場を取っている．ここで留意すべき点は，DBPL は DML と PL の統合だけではなく，DDL や一貫性制約の記述も可能にしなければならないという点である．DDL と一貫性制約は，「オブジェクト」のデータ構造のほかに，各属性の型（制約），あるいは属性間の制約をも記述する．したがってこれを記述するために，PL で扱う型と DBL で扱う型・制約を包摂する共通の型システム（制約システム）という考えに行き着く．両者間の型の不一致はインピーダンス・ミスマッチと呼ばれており，この意味で，DBPL の目的がインピーダンス・ミスマッチの解消であるといわれることもある．

また DBPL を考える際，永続性と型の直交性の概念がよく議論される．すなわち，永続性を型の性質とせず，同一の型のオブジェクト（データ）についても個々にいかなる永続性をもたせるかどうかを指定できる性質である．これは，PS-Algol[2] で導入された概念であるが，実装上の効率などの面などから，いくつかの言語では直交性を完全にはもっていない．

PL の拡張としての PPL と DBPL の区別が議論されることもあるが，最近では区別がなくなりつつあると思われる．DBPL は，永続性以外に DB 管理機能の要件などを意識していることから本稿では PPL は DBPL の一部と考えることにする．「DB を意識する」とは，

- PL と DB システムとのインタフェースの確立
- 言語が DB オブジェクトの意味論をもちうる
- 言語仕様で DB オブジェクトをサポートしている

という発展過程を考えることができる．第2点の例としては，論理型言語（演繹 DB）の最小浮動点意味論のような関係読み (relational reading)，つまり述語と関係の自然な対応関係があげられる．

本稿で扱っている OODBPL の "OO" は DBPL のプログラミング・パラダイムの狭義の "OO" に，必ずしも対応していない．この "OO" は，DB オブジェクトに関する以下のような性質を指している．

- オブジェクト識別性 (object identity)
- 複合オブジェクト
- カプセル化 (encapsulation)
- 継承 (inheritance)
- メッセージ・パッシング

これらはさまざまなプログラミング・パラダイムに埋め込むことができる．PL でのプログラミング・パラダイムには，OO 型，関数型，論理型などがあるが，これは OODBPL のプログラミング・パラダイムに継承されてくる．

以上の枠組を簡単に図示すると，図-1 のようになる．

本稿では DBPL を以上のような観点で考えているが，ほかにも多くの考えがある．たとえば Atkinson と Buneman[3] のものがよく引用される．これはどちらかというと PL 側からの基準であり，DBL の立場からは少し読み替えが必要かも知れない．また，従来 DBMS がもっていた一貫

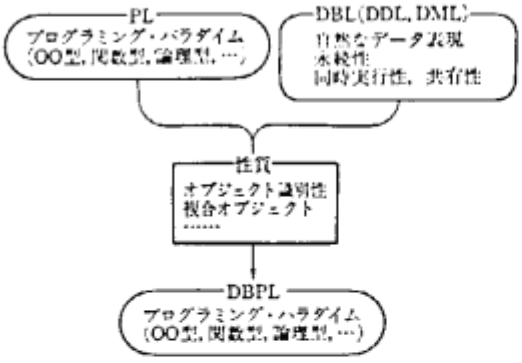


図-1 OODBPL の枠組

性制約やデータの共有の制御などを DBPL で満足すべきだという議論も行われている[7]．さらにいくつかの要件が考えられているが，現状では十分なコンセンサスは得られていない．

### 2.3 DBPL の分類

DBPL と呼ばれている（自称している）言語，あるいは DBPL 的要素をもった言語は数多くある．そこでそれら言語がどのようなプログラミング・パラダイムに属しているかで分類しよう（言語名の後の括弧内はその言語の先祖である）．

- OO 言語
  C系： O++, E, PCOB
  Smalltalk 系：OPAL
  その他： Clovers, Taxis, OOdaplex
- 関数型言語
  ML 系： Machiavelli
  Lisp 系：Orion, PCLOS, LispO$_2$
  その他： Poly (Russel), FQL (FP)
- 論理型言語
  DOOD* 系：O-logic, F-logic, IQL, *Quixote*
  複合オブジェクト系：COL, $\mathcal{LDL}$, C-logic
  その他：HiLog, ILOG
- その他言語
  Pascal/R, Modula/R, PS-Algol, Adaplex (Ada), CO$_2$ (C), BasicO$_2$ (Basic), Aldat (関係代数)

このほかにも多くの言語がある（たとえば参考文献3）を参照していただきたい）．

これらはいずれも 2.1 の条件をすべて満たしているわけではないが，永続性，型システムの拡張，および DB 機能の取り込みの観点から，表に入れている．論理型言語の先祖を書いていないのは，いずれも確定節に基づいた言語（あるいは pure Prolog）を，論理的側面，データモデル的側面，あるいは計算モデル的側面から拡張したものだからである．

最近の DBPL の研究の中で OO 概念をもつものが注目を集めている．その理由は，OODB が最近「ブーム」であるという以上に，

- データと手続きの融合がオブジェクトとして自然に実現できる
- OO 概念の包容力が大きく，柔軟性があるなどが考えられる．しかし，OO 概念の一番の問題は，その形式的基礎が不明確なことである．したがって，OODBPL としては，OO 概念に基づいて実用レベルで融合したものから，最近では，上記の要件を満たし，かつ理論的基礎をもった（主として関数型言語や論理型言語に基づいた）DBPL をめざす研究開発まで，幅広い活動が行われている．

本稿では，OODBPL に焦点を絞り，上の分類のプログラミング・パラダイムにしたがって，代表的な言語を典型的な例を使って概説する．詳細は各言語の参考文献を参照されたい．

## 3. オブジェクト指向パラダイムに基づいた OODBPL

この章では，OO パラダイムに基づく OODBPL について概説する．これに分類される言語としては OPAL[14]，O++[1]，PCOB[15] などがある．また，2.3 ではそのほかに分類した CO$_2$，BasicO$_2$ など複数の DBPL をもつ OODBMS である O$_2$[13] についても述べる．

OO パラダイムに基づく PL は Smalltalk-80[9]，C++[16] など各種がある．これらの言語は 2.2 で述べた DB オブジェクトに関するオブジェクト識別性など "OO" の性質を基本的に満足している．したがって，DBPL はこれらの言語に DBL からの要件である永続性，条件つき検索（問合せ）などの機能をもたせたものとなる．また，DB 管理や効率化のための拡張を行っている言語もある．

---

* 演繹・オブジェクト指向データベース (Deductive and Object-Oriented Database)[20]

### (1) OPAL

Smalltalk-80 を元に設計された OPAL は最初の OODBMS である GemStone の言語である．OPAL はデータ定義，データ操作，および汎用計算のための言語である．既存の関係 DB の言語と比較して，

1) OO 概念を取り入れることにより複雑なデータの表現が容易になり，新しい応用に適している．

2) PL と同様に計算完備である．

などの特徴があり，DBPL の基本的な特徴をもっている．PL としてみたとき，OPAL は構文的には Smalltalk-80 とほぼ同じであり，すべてのオブジェクトが2次記憶に格納され永続性をもつことを除けば同様に使用できる．

OPAL では問合せ記述のための拡張が行われている．たとえば，従業員オブジェクト anEmp の lastName が 'Sanders' であるという問合せは次のように記述できる[14]．

```
empSet select :
{ :anEmp | anEmp. empName. lastName =
'Sanders'}
```

オブジェクトの検索を効率良く行うためにオブジェクトに索引をつけることもできる．

また同時実行制御やオーソライゼーションの単位である segment や，媒体障害対策の単位である repository に対応したクラスがあり，ユーザは OPAL でこれらの単位を制御できる．

### (2) O++

C++ (またはC系の OOPL) を元にした DBPL には O++，E，PCOB などがある．また商用の OODBMS も Objectstore, ONTOS, Objectivity, Versant などが C++ を元にしたインタフェースを提供しており，実用上重要な言語の系統である．

C++ を基にした言語の例として O++ の概要を説明する．O++ は，AT&T Bell 研究所で研究されている ODE (Object Database and Environment) のための DBPL である．C++ に，永続性，版 (versions) 管理，制約 (constraints)，トリガ (triggers) の機能を追加している．O++ では，永続性をもつ永続オブジェクトともたない揮発 (volatile) オブジェクトが存在するが，両者は同じように操作できる．O++ は，OPAL と同様，データ定義，データ操作，および汎用計算を行う言語である．

O++ の構文は C++[15] と同様である．O++ における，オブジェクトの永続性は，クラス仕様 (型) ではなくインスタンスの性質として定義される．したがって同じクラスで，揮発オブジェクトと永続オブジェクトの両方のインスタンスをもつことができる．たとえば，stockitem のクラス仕様が定義されているとき，揮発オブジェクト stockitem は，

```
stockitem *sip ;
.....
sip = new stockitem (initial values) ;
```

と書くことによって生成される．永続オブジェクトは同様に，

```
persistent stockitem *psip ;
.....
psip = pnew stockitem (initial values) ;
```

で生成される．pnew によって stockitem のインスタンスが永続記憶 (persistent store) に生成され psip はこの永続オブジェクトへのポインタ（オブジェクト識別子）である．オブジェクトの消去は delete または pdelete で行う．永続オブジェクトへのアクセスはオブジェクトが永続記憶にあること以外は通常のオブジェクトの場合と同様に記述できる．

O++ では，問合せを効率的に行うためクラスタの概念があり，同じ型の永続オブジェクトはすべて同一のクラスタに置かれる．また，集合を扱うこともできる．問合せでは関係 DB で重要な結合 (join) と同等な記述もできる．たとえば person が

```
class person {
public :
  Name name ;
  charsex ;
  Name Profession ;
  persistent person *children <> ;
  /* *はポインタ，<>は集合を意味する．
 */
} ;
```

のように定義されているとき，子供の職業が computer science (CS) であるような人とその子供の名前を検索し出力するためには，次のように書くことができる[1]．

```
for p in person
  for c in p->children
    suchthat (c->Profession == Name ("CS"))
  printf ("% s % s\n", p->name, c->
  name);
```

また演繹 DB で議論されているような「再帰問合せ」も記述できる．たとえば，abraham のすべての子孫は次のようにして検索できる．

```
for p in person
suchthat (p->name == Name ("abraham"))
  descendants=p->children;
for d in descendants
  descendants+=d->children;
```

### (3) PCOB

問合せ記述に関しては，PCOB の試みが面白い．PCOB は，日本アイ・ビー・エムで開発されている，C に OO の概念を導入して拡張した COB に永続性をもたせた言語である[15]．永続性に関しては，DB 変数から間接的に参照されているオブジェクトが永続性をもつ．PCOB では，集合指向の問合せ処理のために，dbcollection を導入している．あるクラス c に対して dbcollecton [c] は c のインスタンスの集合を保持するクラスである．c の中で定義されたインスタンスメソッドに特定の型（たとえば，dbbool）を与えることにより，そのメソッドの値を利用した問合せを可能にするメソッドが，dbcollection [c] に自動生成される．これにより，c のオブジェクトに対する情報隠蔽を損なうことなく問合せの記述を行うことが可能となる．たとえば次のようなクラス定義があるとする[15]．

```
class Employee {
  dbbool olderThan (int age);
  };
class impl Employee {
  int myage;
  Employee boss;
  definition:
   dbbool olderThan (int age) {
    return (myage>age);
    }
  };
```

return 値が dbbool 型であるメソッドは，フィルタと呼ばれ，問合せの選択条件として使用されることが想定される．この場合，メソッド olderThan がフィルタである．Employee 型の集合を保持する dbcollection [Employee] には，olderThan に対応して select_olderThan というメソッドが自動生成され，dbcollection [Employee] のインスタンス Emps に対して以下のような問合せが可能となる．

```
oldEmps=Emps->select_olderThan(30)
```

さらに，フィルタに対応するインスタンスメソッドは，return 文の中に論理式で宣言的に記述できるので DB で開発された最適化技法を使用する機会を与えるとしている．

### (4) $O_2$

特徴のあるアプローチとして Altair で研究開発されている $O_2$ がある．ほかのシステムが単一言語によって応用 PL と DBL の統合化を図っているのに対して $O_2$ では，多言語によるアプローチをとっている．現在，$CO_2$，$BasicO_2$，$LispO_2$ の3種が報告されている[12]．

$O_2$ では，すべてをオブジェクトとして扱うのではなく，オブジェクトと複合値の両方を提供している．また，永続性については，名前付けを行うことにより永続性を指定するやり方と，指定されたクラスのすべてのインスタンスに永続性をもたせるやり方をもつ．前者の場合には，名前付けられたオブジェクトや値，およびそれらを構成するオブジェクトや値が永続性をもつことになる．たとえば，$CO_2$ では，

```
add object Eiffel_tower: Monument
```

により Monument 型のオブジェクト Eiffel_tower は，名前付けられ永続性をもつ．代入により名前付けられたオブジェクトの変更ができる．

```
Eiffel_tower=new(Monument)
```

このほか，例外属性や例外メソッド，型チェッカなどをもっている．

## 4. 関数型パラダイムに基づいた OODBPL

関数型パラダイムに基づく DBPL としては，2.3 で述べたように，大きくは Lisp 系のものと ML 系のものがあげられる．前者はどちらかというと実用レベルでの研究であるのに対して，後者は理論的な面に対する研究の色彩が強いものが多い．ここでは，両者の例として ORION[12] と Machiavelli[17] を概説する．

### （1） ORION

ORION は MCC で研究開発された OODBMS で，Common LISP を OO 概念と DB 機能を導入して拡張した言語を用いている．DB 機能としては，DB に対するアクセス，合成 (composite) オブジェクトや版の管理，DB 制御のための機能拡張がなされている．

ORION では，新しいクラスの生成を以下のように記述する[12]．

```
(make-class Classname
    :superclasses ListofSuperclasses
    :attributes ListofAttributes
    :methods ListofMethodsSpaces)
```

このクラスのインスタンスを作るには以下のようにする．

```
(make Classname :Attributel valuel
                .....
                :AttributeN valueN)
```

DB へのアクセスでは，クラスに対して Query Expression を与えて，それを満足するインスタンスを検索できる．

```
(select 'Vehicle' (>Weight 5000))
```

ここで，'Vehicle はクラスの指定で，そのインスタンスで (>Weight 5000) の条件を満足するものが検索される．同様の記述で指定クラスのインスタンスの更新，削除も行える．このほか，版管理のための記述，トランザクション管理などのための拡張が行われている[12]．

### （2） Machiavelli

DB と PL の型の不整合に対する研究として Machiavelli がある．従来の PL は，豊富な型とその操作をもつが，関係 DB で基本的な操作である自然結合や射影のような操作はなかった．逆に DBMS でも型の概念は存在していたが PL の型システムの利点（多相型や型推論）をもたなかった．すなわち両者の型の概念に不整合が存在している．

Machiavelli は関数型言語 ML[9] に DB で扱うデータ型（構造）とそれらの間の演算として自然結合や射影を扱えるように拡張した言語である．そのうえに OO 概念の導入を試みている．

ML の最も大きな特徴は，多相型を含む強い型付けと強力な型推論であるが，これは Machiavelli でさらに拡張されている．型の利点としては，静的な型解析による型エラーの検査や，オブジェクトコードの効率化などがあげられる．すべてのデータに型を指定するのはプログラマにとって煩わしい作業であるので，型の指定されていないデータに対しても型を推論する機能が重要となる．これが型推論機能である．DB オブジェクトは一般に，より複雑な構造をもっているので，これらの機能はより重要となってくる．

Machiavelli における OO 概念としては，オブジェクト識別性，抽象データ型および継承の導入を行っている．たとえば，オブジェクト識別性の ML の reference 型を用いた定式化や継承を含むクラス階層に対する型システムの適用などを試みている[17]．

型推論の例題[17]をみてみよう．DB に以下のようなデータが格納されていたとしよう．

```
{[Name="Joe", Salary=22340],
 [Name="Fred", Salary=123456],
 [Name="Helen", Salary=132000]}
```

これらのデータに対して，給与 (Salary) が 100000 以上であるものを求める関数は以下のように記述できる．

```
fun Wealthy(S)=select x.Name
                where x<-S
                with x.Salary>100000;
```

ここで，select, where, with などは，構文的糖衣で関数を用いた形に展開される．

この Wealth の関数定義には型宣言はないが，型推論の結果以下が得られる．

```
Wealthy(S) := {([("a) Name:"b, Salary:int]}
  ->{"b}
```

ここで，"a や "b は型変数であり，この場合，("a) は，それを含むレコード型に Name, Salary 以外の属性 (field) がいくつか存在してもよいことを表している．つまり，直観的には，Wealth (S) は，少なくとも "b 型の属性 Name と整数の属性 Salary をもつレコードの集合から Name 属性の型と同じ型 ("b 型) の集合への関数であると推論されている．

型を型変数を用いた多相型で表現しているため，Wealthy は，たとえば，Name 属性が，文字列型でなく，姓と名からなるレコード型である場合や，weight 属性を含んでいるような場合でも，そのレコード型の集合に対して適用できる関数として定義されている．

## 5. 論理型パラダイムに基づいた OODBPL

論理型 PL に基づく DBPL は，演繹 DB (DDB) の拡張と考えることができる．DDB は一言でいえば，論理型 PL（あるいは一階述語論理）により DB を再構成したものである．通常構文は論理型 PL と同様で確定節などを使用しているが，DB オブジェクト（関係）の意味論をもち，問合せの評価方法が異なっている．これにより DB の特徴である集合操作をもつほか，問合せの停止性なども特徴としている．さらにこのパラダイムに基づいた言語の特徴は，知識表現や知識プログラミングを意識していることが多いことである．しかし 2.1 で検討したように，DBPL の要件には，永続性や型システムを考える必要があり，いくつかの拡張が必要である．

論理型パラダイムにのっとった DBPL と考えられる主要な言語をまとめると，表-1 のようになる．

この表の中で，OO 概念を取り込んだ DOOD 言語としては，O-logic, F-logic, *Quixote* があり，本稿の OODBPL に対応していると考えられる．本章では F-logic[11] と *Quixote*[16),19)] について例を中心に概説する．

### (1) F-logic

F-logic では，オブジェクト識別子に対応する id-項 (id-term) が重要な役割をもっている．id-項は通常の一階項として定義され，この具象項が意味領域を構成する．id-項間の関係がオブジェクトとして定義されるが，これは，ラベル（これも id-項）とほかのオブジェクト（の集合）の対（属性）を，組構成子で構成したものである．

表-1 論理型言語の OODBPL

| 言語名 | 集合[a] | 複合構造 | OID[b] | 更新[c] | 型[d] | メソッド |
|---|---|---|---|---|---|---|
| $\mathcal{LOL}$ | ○ | × | × | ○ | × | × |
| ILOG | × | × | ○ | × | × | × |
| COL | ○ | ○ | × | × | ○ | × |
| IQL | ○ | ○ | ○ | × | ○ | × |
| O-logic | ○ | ○ | ○ | × | ○ | × |
| F-logic | ○ | ○ | ○ | × | ○ | ○ |
| *Quixote* | ○ | ○ | ○ | ○ | ○ | ○ |

[a] 集合を明示的に記述可能か？
[b] オブジェクト識別性をもっているか？（ただし言語によっては認識の違いがある．）
[c] DB とのインタフェースをもつだけでなく，更新も可能か？
[d] 型概念をもっているか？（ただし PL での型概念とは異なっている．）

*person* (*john*, 30) [*addr*→*chiba*,
*hobby*↠{*golf*, *tennis*},
*affil* [*name*→*abc*,
*addr*→*tokyo*]]

ラベルと値間の "→" と "↠" は，値が個体か集合かを区別している．

F-logic では型とインスタンスは相対的な概念で，共に id-項がその役割をはたしている．id-項の集合は束構造をもっており，この順序にしたがって属性の継承が行われる．

F-logic はこのほかに，メソッドの概念を論理型 PL の中に埋め込んでいる．

$X$[*children*::$Y$↠{$Z$}]⇐
*person*: $Y$[*children_set*↠{$Z$}]
*person*: $X$[*children_set*↠{$Z$}]

は，$X$ に $Y$ を引数としてもったメッセージ *children* を送ると，$X$ と $Y$ に共通の子供の集合 {$Z$} を返してくれる．ここで使われているラベル *children*:: $Y$ は，本質的には *children* ($Y$) という述語に対応しており，述語記号の多重定義 (overloading) を行っている．

### (2) *Quixote*

*Quixote* のオブジェクト識別子は，F-logic での id-項を「複合オブジェクト」に拡張している．これは，ほかの計算パラダイムと異なり，論理型パラダイムではルールによって生成される内包的オブジェクトの識別子を生成する必要性があるためである．この意味領域は（超集合として定義される）ラベルつきグラフの集合である．

オブジェクト識別子に付属する属性とオブジェクト項内の属性は同一構文で書かれるが，後者をオブジェクトにとって本質的と考えることによって，オブジェクト項の順序にしたがった例外継承が自然に表現できる．

鳥/[飛行能力＝yes]∧ペンギン⊑鳥
　ペンギン/[飛行能力＝yes]
ペンギン [飛行能力＝no]⊑ペンギン
　ペンギン [飛行能力＝no]/[飛行能力＝no]

この例で "/" の前がオブジェクト識別子であり，"/" の右がそれに付随する属性である．属性 "飛行能力＝yes" はオブジェクト "ペンギン [飛行能

力=no]" に継承されようとするが，同一ラベルの属性がオブジェクト識別子中に現れるので，ここで例外として扱われている．

もう一つの大きな特徴はモジュール概念である．モジュールは，*Quixote* のオブジェクトから構成されるルールの集合であり，モジュール識別子もオブジェクト項で表現される．モジュール "$m$" でオブジェクト "$o$" がサポートされることを "$m::o$" と書くと，モジュール間には以下のサブモジュール関係 $\sqsubseteq_s$ がある．

$$m_1 \sqsubseteq_s m_2 \wedge m_1::o \supset m_2::o.$$

モジュールはほかのモジュールの「集合演算」式によって定義することが可能で，この意味で *Quixote* はモジュール間の例外つきルール継承を可能にしている．

*Quixote* での永続オブジェクトの扱いは，DDB における外延 DB のようなインタフェースとその更新を制御する入れ子トランザクションの導入によって行っている．

## 6. おわりに

PL では，実行の効率性，意味論の明快さ，表現の簡単さなどが重視されているが，DBPL ではさらに，DB のような巨大オブジェクトの扱い，永続性の扱いなどが必要となってくる．これらを備えた，実用的でかつ形式的基礎の明確な DBPL は，本稿でみてきたようにまだ存在しないが，本稿では，それを目指しその可能性をもったいくつかの言語を紹介した．情報処理分野での DB の重要性はますます増しており，この意味で DBPL への期待も大きくなりつつある．DBPL の研究は急速に進展しており，プログラミング・パラダイムや応用分野に対応して，今後さらに多くの DBPL が登場してくると思われる．

**謝辞**　本稿の内容の多くは，1990 年度に ICOT で開催された DOOD-WG DBPL-SWG での議論によっている．田中克己（神戸大）氏，大堀淳（沖電気）氏を始めとする委員の方々に感謝する．

## 参考文献

1) Agrawal, R. and Gehani, N. H.: ODE (Object Database and Environment): The Language and the Data Model, *Proc. SIGMOD '89* (1989).
2) Atkinson, M. P., Bailey, P. J., Cockshott, W. P., Chisholm, K. J. and Morrison, R.: An Approach to Persistent Programming, *Computer*, 26 (4) (1983).
3) Atkinson, M. P. and Buneman, P.: Types and Persistence in Database Programming Languages, *ACM Comput. Surv.*, 19 (2) (1987).
4) Atkinson, M. P., Buneman, P. and Morison, R. (eds.): *Data Types and Persistence*, Springer (1988).
5) Bancilhon, F. and Buneman, P. (eds.): *Advances in Database Programming Languages*, Addison-Wesley (1990).
6) Barbedette, G.: LISP$O_2$: A Persistent Object-Oriented Lisp, *Proc. Extending Data Base Technology '90* (1990).
7) Buneman, P. and Ohori, A.: Developments in Database Programming Language, *Tutorial Texts, InfoJapan* (Oct. 1990).
8) Goldberg, A. and Robson, D.: *Smalltalk-80: The Language and its Implementation*, Addison Wesley (1983).
9) Harper, R., Milner, R. and Tofte, M.: The Definition of Standard ML (Version 2), *LFCS Report Series, ECS-LFCS-88-62*, Dept. of Computer Science, U. of Edinburgh (Aug. 1988).
10) Hull, R., Morrison, R. and Stemple, D. (eds.): *Proc. Second International Workshop on Database Programming Languages*, Morgan Kauffmann, Gleneden Beach, Oregon, 4-8 (June 1989).
11) Kifer, M., Lausen, G. and Wu, J.: Logical Foundations of Object-Oriented and Frame-Based Languages, *Technical Report, 90/14*, State U. of New York at Stony Brook (June 1990).
12) Kim, W., Ballou, N., Chou, H., Garza, J. and Banerjee, J.: Integrating an Object-Oriented Programming System with a Database System, *Proc. OOPSLA '88* (1988).
13) Lécluse, C. and Richard, P.: The $O_2$ Database Programming Language, *Proc. VLDB '89* (1989).
14) Maier, D., Stein, J., Otis, A. and Purdy, A.: Development of an Object-Oriented DBMS, *Proc. OOPSLA '86* (1986).
15) 二ッ井，上村：オブジェクト指向プログラミング言語への持続性機能の導入，*WOOC '90* (Mar. 1990).
16) Morita, Y., Haniuda, H. and Yokota, K.: Object Identity in *Quixote*, 情報処理学会データベース研究会資料，*DB-80-12* (1990).
17) Ohori, A., Buneman, P. and Breazu-Tannen, V.: Database Programming in Machiavelli—A Polymorphic Language with Static Type Inference, *Proc. ACM SIGMOD '89* (1989).
18) Stroustrup, B.: *The C++ Programming Language*, Addison-Wesley (1986).
19) Yasukawa, H. and Yokota, K.: Labeled Graph as Semantics of Objects, 情報処理学会データベース研究会資料，*DB-80-13* (1990).
20) 横田，西尾：演繹・オブジェクト指向データベース，情報処理，Vol. 31, No. 2 (Feb. 1990).

（平成3年1月7日受付）