データベース・システム 85－7－3
プログラミング－言語・基礎・実践－ 4－7－3
（1991. 10. 4）

永続的プログラミング言語とオブジェクト指向データベース

鶴岡 邦敏
日本電気㈱C＆Cシステム研究所
〒216 川崎市宮前区宮崎4－1－1（tsuruoka@btl.cl.nec.co.jp）

1．はじめに

次世代データベースとしてのオブジェクト指向データベースの研究開発が進展するにつれて、データベースとプログラミング言語の位置づけに関して、新たな枠組みが形成される機運がある。本稿では、オブジェクト指向データベース（OODB）を処理するために、永続的オブジェクトを扱うプログラミング言語（PPL）に必要とされる機能的拡張に関して述べる。これは、特定の言語の利用者から見てDB機能は透明であること、即ちその言語側のモデルとしてDB機能を持つことが必要、という立場による。特に、筆者らが開発中のオブジェクト指向DBMS－Odinの機能を踏まえて、OODB側から見たプログラミング言語への要請（＝拡張機能）を中心に記述する。OdinはC++の拡張言語（Odin/C++）を持つため、以下ではC++等の言語を想定する。

2．プログラミング言語への要請

（1）型とクラスの併用

以下では、同種のオブジェクトが持つ性質を型、その性質を定義したオブジェクトをクラスと考える。OODBにおいても、プログラムの誤動作を防ぎデータベースの完全性を確保する目的で、型（特に型検査）は重要である。一方以下の理由で、実行時に動くオブジェクトとして、クラスも必要である。

- クラスが保有する定義情報を実行時に参照し、それを解釈しながら動くユーティリティがある（例：ブラウザ、インタープリタ）。
- クラス変数やクラスメソッドを提供する。
- クラスの構造（型の定義体）を変更する際に、応用プログラムの修正や再コンパイルをできるだけ避ける（実行時にクラスを参照する）。

（2）ポインタ変数の回避

ポインタ変数（＊p等）は、プログラムの誤動作の原因となり易く（DBにおいてはデータ破壊を起こす危険性を持つ）、またプログラムの記述性を低下させる。このため、応用プログラムに対してはポインタ変数の不要な構文を提供すべきである。C++系のOODBではポインタ変数で永続オブジェクトを指すものが多いが、以下のようにしてポインタ変数を回避すべきと考える。

- 基本型の変数はデータの値を持つ。
- クラス型の変数はオブジェクトへの参照を持つ。
- オブジェクトは他のオブジェクトの実体を含まない。

[Odin/C++の例]
```
社員  emp;
会社  comp;
comp = emp.hired;
```

[市販OODBの例]
```
persistent shain *emp;
persistent kaisha *comp;
comp = emp -> hired;
```

（3）型構成子と実行時に生成される型

OODBでは、DB中のオブジェクト群を部分的に見たい、構造を組み替えて見たい等の目的で、実行時に型を生成する場合がある。このため、型の名前の代わりに、型構成子を指定できることが必要である。

[Odin/C++の例]
```
set{社員}  emps;
set{会社}  comps;
select{社員}  selemp;
set{select{社員}}  selemps;
join{社員, 会社}  empcomp;
set{join{社員, 会社}}  empcomps;
selemps = emps.select(...);
empcomps = emps.join(...comps...);
```

（4）型構成子を持つクラスの定義

あるクラスを定義する場合に、その型を型構成子として与えたい場合がある。例えばOdinの集合クラスSimple_Setは、要素となるオブジェクトのクラスをクラス変数の値として持つ。この時Simple_Setのサブクラスを定義するためには、要素のクラスを指定する構文が必要となる。

[Odin/C++の例]
```
persistent class 社員集合
:public Simple_Set
:as set{社員}
{ };
```

上記の例では、:as 以下で要素の型を指定している。

（5）ビュークラス

多数の利用者が共有するＯＯＤＢにおいては、オブジェクト群の構造を利用者ごとに特有の観点から見るために、ビュー機能が必要とされる。Odinでは、結合、グループ化等の集合操作の結果として一時ビュークラスが生成され、また利用者が永続ビュークラスを定義することもできる。

```
[Odin/C++の例]
 persistent view class 社員会社
 :as join{社員, 会社}
 { projected:        // 可視変数、可視メソッド
     string empname;
     string compname;
   added:            // 付加メソッド
     ...
   public:
     meta void derive();  // ビュー導出メソッド
 };
```

永続ビュークラスにより、インスタンス変数やメソッドの参照の制限（可視条件）や、メソッドによる参照・更新の意味の記述（付加メソッド）等が実現できる。

（6）トランザクション構文

ＯＯＤＢにおいては、並行制御やデータベース処理の一貫性を維持する目的で、トランザクションの概念が必要とされる。あるトランザクション内で参照／更新されたオブジェクト群は、トランザクション終了時に発行されるcommitによって他のトランザクションとの競合をチェックされる。またrollbackによって、プログラマが意識的に更新を無効にすることも可能である。この際、トランザクション内のオブジェクトを外から参照できない、commit/rollbackを発行せずにトランザクションから抜けられない、等の制約を実現する構文が望まれる。このような目的で、トランザクション文が必要となる。

```
[Odin/C++の例]
 transaction Tr(...)
 { ...
  if(...) break_trans;
   ... }
```

（7）集合検索

大量のオブジェクト群が共有される環境においては、ある利用者が当面必要なオブジェクト群を特定するために、集合検索機能が重要である。

```
[Odin/C++の例]
 selemps = emps.select([self: E, $会社: C |
   where E.hired == C and C.compname == "NEC"]);
```

（8）その他の機能

マルチメディア処理等の応用を考慮すると、可変長変数／可変長配列が必要となる（動的な変数追加も要請される）。このため、コンパイル時にインスタンス変数値のオフセットが決定できない。

3．実現上の問題点と今後の方向

現在の言語処理系は、その実現において以下のような問題点がある。将来は、ＯＯＤＢとＰＰＬとが統合されると共に、利用者に一貫したプログラム開発環境を提供すべきである。

- 型（クラス）定義情報をソース形式でファイルに保管し、応用プログラムのコンパイル時にincludeして用いる。この方式だと、例えば対話型のユーティリティを用いてクラス定義の変更を行った場合など、ＤＢ中のクラス構造とinclude file中のクラス定義の不整合が生ずる恐れがある。型／クラス定義情報はオブジェクトとしてＤＢ中に格納し、これを言語処理系が直接参照すべきである。
- メモリ管理（ヒープ領域等）が大量のまたは長大オブジェクトの処理に適していない（ヒープが無制限に拡大する、実行中にメモリ割り当てが不可の状態に陥る等）。ＯＯＤＢ側のメモリ管理との統合を図り、仮想記憶を越える量のオブジェクトを処理でき、かつ安定したメモリ割り当てを行える管理方式を実現すべきである。
- エディタ、コンパイラ、デバッガ、ローダ等が開発環境として統合されていない。またこれらのツールは、「ファイル」という物理的な実体を意識した実現をとっている。将来はＯＯＤＢを中心として、以下のようにオブジェクトを管理する（ファイルを意識しない仮想的な）開発ツールとすべきである。

  エディタ：　「ソースメソッドクラス」のオブジェクトを編集するオブジェクトエディタ

  コンパイラ：　「ソースメソッドクラス」のオブジェクトを「実行形式メソッドクラス」のオブジェクトに変換するオブジェクトトランスレータ

  ローダ：　「実行形式メソッドクラス」のオブジェクトを動的リンクするダイナミックメソッドローダ

参考文献

[KiTs90]木村、鶴岡「ｵﾌﾞｼﾞｪｸﾄ指向ﾃﾞｰﾀﾍﾞｰｽ操作言語Odin/C++について」、電子情報通信学会データ工学研究会、DE90-26、1990.12.14

[KiTs91] Kimura, Y. and Tsuruoka, K. "A View Class Mechanism for Object-Oriented Database Systems," Proc. DASFAA'91, pp.269-273, April 1991.