オブジェクトの『所有者』問題とC++による実装例

3F−3

上原隆平、吉本雅彦†、黒澤貴弘†、柴山茂樹†
† キヤノン(株) 情報システム研究所

1 はじめに

近年、オブジェクト指向データベース(OODB)が注目を集めてきた。OODBは従来の関係型データベース(RDB)に比べ、データベース中の複雑な構造を持ったデータを、プログラム中で直接操作できるという特徴を持っている。具体的にはOODBでは、オブジェクト単位でデータをアクセスすることができる。

本稿ではこうしたOODBシステムでのアクセス管理の問題をとりあげる。ファイルシステム、RDB、OODBについてのアクセス管理機構を表にまとめると、表1のようになる。

すなわち、従来のファイルシステムでは、複雑な構造を持ったデータを直接扱うことはできず、ユーザが操作する対象も、OSがアクセス管理を行なう対象も、単純な構造からなるファイルが単位である。

RDBシステムでは、データは関係という単純な表の形で表現される。ユーザは関係の中のタプル単位でデータを操作することができ、またアクセス管理も、VIEWを使用すれば、タプル(レコード)単位で行なうことが可能である[1]。

OODBシステムでは、オブジェクト、すなわち複雑な構造を持ったデータを、そのままデータベースの中に蓄積することができる。しかしながらOODBシステムが提供するアクセス管理は、データベース単位であり、オブジェクト単位でのアクセス管理を行なう機構は、現段階では提供されていない。

オブジェクト指向プログラミングでは、意味のあるまとまりをオブジェクトにマッピングすることが多い。これを受けて、近年のOODB技術や、ネットワーク環境での分散オブジェクト技術は、ユーザがデータベースやネットワークなどを意識することなく、オブジェクト単位での操作を可能にする方向にむかっている。こうした背景を考えると、アクセス管理に関しても、オブジェクト単位で自然に記述できることが望ましい。しかしながら現在のOODBシステムでは、こうした管理機構が提供されているものはない。

本稿では、オブジェクトに「所有者」という概念を持ち込むときに生じる問題を扱う。こうした問題は、後で示すように、一般にオブジェクトに新たな属性を導入し、その属性に対して意味のある操作を定義しようとするときなどに、典型的に生じる問題である。ここではこうした問題をオブジェクトの所有者問題と呼び、これについて論じる。

2 オブジェクトの所有者問題

オブジェクト単位での所有者の概念を導入するためには、個々のオブジェクトに対して、所有者情報と、管理情報とを追加して、オブジェクトに対してアクセスがあった場合、これらの情報に従ってオブジェクトのふるまいを変更すればいい。

	ファイルシステム	RDB	OODB
最小のデータベースの単位	ファイル	Relation	データベース
アクセス単位	ファイル	Relation/tuple	データベース
操作単位	ファイル	tuple	オブジェクト
操作単位のアクセス管理	○	○	×

表1: ファイル、RDB、OODBの比較

具体的には、オブジェクト間でメッセージの授受が行なわれるときに、正当なユーザには期待されている動作をし、不当なユーザには、エラーを出力しなければならない。例えばそのオブジェクトが所有者以外の変更を許さない、という属性を持っていた場合には、不当な変更要求に対しては、エラーや例外発生などを実行する必要があり、しかもこうしたふるまいは複数のクラスにわたって一貫して行なう必要がある。

つまり所有者属性を追加するためには以下のような要件を満たす必要がある。

網羅性: すべてのメッセージを検査する。

系統性: 複数のクラスにまたがって、一貫した処理を行なう。

本稿では、メッセージに対して上に述べた網羅的かつ系統的な要件を必要とする問題を、総称してオブジェクトの所有者問題と呼ぶことにする。

所有者問題を解決するためには、以下のような方法が考えられる。

方法(i): オブジェクトのアクティベイション時に検査する。

方法(ii): オブジェクト指向言語の継承機能などを利用して、すべてのメッセージの授受部分を検査し、その結果に応じて管理を行なう。

方法(i)は有効な方法であるが、OSのメモリ管理部分に手を加えるなど、低レベルでの対処が必要となることが多く、一般にはこうした方法はとれない場合が多い。

A Solution to an "Ownership" Problem in Object-Oriented Systems
Ryuhei UEHARA, Masahiko YOSHIMOTO†, Takahiro KUROSAWA†, Shigeki SHIBAYAMA†
† CANON Inc.

現在のオブジェクト指向言語の中には、方法 (ii) で所有者問題に対処できる言語もある。具体的には、ルートクラスに管理機構を組み込んでしまったり、before daemon の機構を利用して、オブジェクトをアクセスする際に、すべてのメッセージ授受に先だって、管理機構を起動することが可能な場合がある。

しかしながら、現在非常にユーザの多いオブジェクト指向言語である C++においては、方法 (ii) で所有者問題を解決することは、非常に困難である。

なぜなら、所有者属性を C++の継承機能を用いて実現しようとすると、網羅的かつ系統的という特徴のため、複雑なクラスや、多数のクラスに対しても、すべてのメソッドをもれなく変更する必要があるためである。

つまり C++での継承機能は、あるクラスのふるまいをそのまま引き継ぐ場合には有効であるが、所有者属性のように、ふるまいを網羅的かつ系統的にモディファイして引き継ぐ場合には、あまり有効ではない。

以下本稿では、C++において上記以外の方法を用いたオブジェクトの所有者問題の解決方法を示し、実際にこの方法に基づいて所有者クラスを作成するための、Registrant template について紹介する。

3 Registrant template

本章では、C++の template 機能 [2] を利用した所有者問題の解決方法を示す。なお、今回実現した Registrant template の所有者属性や管理方法は、UNIX のファイルシステムの管理方法と、ほぼ同様のものにした[1]。

Registrant template は、引数であるクラス T に対して、内部変数としてクラス T のインスタンスへのポインタと、アクセス管理を行なうための管理情報とを持っていて、内部関数として、クラス T のインスタンスをアクセスするための関数と、管理情報を操作するための関数を持っている。具体的な構造を以下にソースの形で示す。

```
template <class T> class Registrant{
  T* data;          // オブジェクトへのポインタ
  int permission;   // 管理情報
  int u_id, g_id;    // ユーザ ID, グループ ID
public:
  Registrant(T& d, int p, int uid, int gid );
    // ポインタ, 管理情報, user id, group id
  Registrant(Registrant&);
  Registrant();
  operator T&();
  operator T();
    // 以下、管理情報を操作する関数は略
};
```

クラス X に所有者属性を追加したクラス、Registrant<X> のインスタンス x を生成するには、以下のように宣言する。

```
Registrant <X> x;
```

クラス Registrant<X>のインスタンス x は、クラス X へのキャスト演算子を用いることによって、以下のように、通常のクラス X のインスタンスと同様の操作をすることが可能である。

```
X xx;
xx = (X)x;          // x の参照
(X&)x = xx;         // x への代入
((X)x).message(); // クラス X のメッセージ
```

キャスト演算子が適用される時点で、管理情報がチェックされ、不当なアクセスであれば、例外が発生して例外処理ルーチンが起動される。こうした管理情報は、以下のように変更・参照する。

```
x.set(0666);              // 属性の設定
int permit = x.get();  // 属性の参照
```

正しい権利がなければ、管理情報の変更・参照ができないのは、言うまでもない。

4 まとめ

本稿では、オブジェクトの所有者問題をとりあげた。現在もっとも普及が進んでいる C++では、この問題を継承機能を使用して解決することは、非常に困難であり、本稿では template 機能を使用した解決方法を示した。

網羅的かつ系統的という要件を満たす、所有者問題の例としては、メッセージのプロトコルが異なる複数のオブジェクトの共有システム同士を結合する場合が考えられる。この例において、結合されたシステム上で、プロトコルの差異を意識せずにメッセージをやりとりするためには、今回同様、網羅的かつ系統的なメッセージ変換が必要となる。

分散環境において、複数のユーザがオブジェクト単位での共有を行なう場合、同様の問題は他にも考えられ、今後こうした問題の重要度はますます増していくものと考えられる。そうした意味では、このような問題を解決する手段は、本来、より基本的な部分で与えられているべきであろう。

また [3] で指摘されている、OODB におけるオブジェクトの蒸発問題についても、前記の方法 (i) と同様の方法で解決できることが指摘されている。

これらの事柄を考え合わせると、今後の OODB システムでは、ユーザがオブジェクトをデータベースから取得してくる時点で、何らかの形でオブジェクトへの操作を可能にするようなメカニズムが必要になってくると考えられる。

[1] 実行パーミションには意味はない。こうした管理方法が必ずしも妥当であるかどうかは検討を要するが、本稿では取り上げない。

参考文献

[1] C.J.Date. *Database Systems.* Addison-Wesley, 5th edition, 1991.

[2] Bjarne Stroustrup. *The C++ Programming Language.* Addison-Wesley, 2nd edition, 1991.

[3] 黒澤貴弘, 上原隆平, 吉本雅彦, 柴山茂樹. データベース・アプリケーションにおけるオブジェクトの 蒸発問題の検討. WOOC' 92 予稿, 3 1992.