4N-02　メタデータプログラミングのための利用者インターフェイス *

渡辺浩平 野末英生 柏川伸悟 松本渉 三浦孝夫 †
法政大学工学部電気電子工学科 ‡

概 要

メタ情報制御を利用することによって入れ子構造や複合オブジェクト、マルチメディアデータなどを統一的に支援するための利用者インターフェイスを提案する。

1　前書き

開発中のデータベースシステム Harmonizing Objects and Meta-Objects Environment(HOME) システムでは、高水準のデータベース操作のプログラムインターフェイスを提供している。特にマルチメディアデータや入れ子構造だけにとらわれないデータ格納構造とデータ操作を提供する。これを現在のシステムで実装するとするとマルチメディアデータや入れ子構造は単純値でないから、他の単純値と一括して扱えない。マルチメディアデータ、入れ子構造を扱う特殊な操作を必要としてしまう。入れ子構造や複合オブジェクト、マルチメディアデータの管理をそれぞれに固有の操作ではなく、すべてのオブジェクトを同じ操作で扱えるようにする。具体的には、関係とスキーマをプログラミング上での管理、操作を分ける。C++言語のクラス概念を用い、関係をオブジェクトとみなす。そうすることで、HOME プログラマは今扱っているデータ型を意識することなく各情報を操作できる。

2　HOME データベースシステム

HOME システムの特徴は次の 3 つである。

1. 利用者がデータベース設計ができるプログラミングインターフェイス
2. オブジェクトとメタオブジェクトを切り離さない管理
3. 具体化関数と抽象化関数

HOME のプログラミングインターフェイスは、関係代数などのデータベース操作の関数群を用意しており、利用者はこれらを利用してデータベースが設計、実装できる。HOME はメタオブジェクトの管理方法としてコアセットと呼ばれる 3 つのカタログ (**Relation Catalog**、**Attribute Catalog**、**Domain Catalog**) を利用している。**Relation Catalog** は各関係についての情報、**Attribute Catalog** は全関係中の属性情報、**Domain Catalog** はすべての値に対する領域情報を保存しており、これらのカタログにより、関係と関係スキーマの間で整合性を保たせている。スキーマ操作はカタログへの関係代数を適用することによって実現する。このことはオブジェクトとメタオブジェクトを一括して扱っていることを意味するが、利用者は簡単なスキーマ操作にも複雑な関係代数を扱わなければならない。内部ではカタログによってメタオブジェクトを管理している事によってスキーマ操作が複雑になっているインターフェイスを単純化することを考える。プログラミングレベルでのオブジェクトとメタオブジェクトの管理、操作を分けるべきであろう。メタオブジェクトの操作は利用者にはスキーマオブジェクトの操作、システム内部ではカタログに対する関係代数と見える。HOME 特有の操作拡張関係代数、すなわち具体化関数**$**、抽象化関数**^**がある。**$**は、メタオブジェクトにオブジェクトを対応させ、**^**は、オブジェクトにメタオブジェクトを対応させる。

3　利用者インターフェイス

3.1　設計方針

メタデータプログラミングとは、スキーマ問い合わせを意識した、データベースプログラミングである。これによってスキーマから関係が検索できるので、その結果について再び操作をするような場合に、継ぎ目のないプログラミングが可能となる。そのためには、関係を多数のパラメータで管理するのではなく、1 つのデータ型で管理する必要がある。また、利用者にデータ型を意識させないためにデータ型に固有の操作をなくすことも必要である。例えば $\pi \ldots \sigma \ldots (R)$ という関係代数は、

```
select("rel0", "rel1", ...);
project("rel1", "rel2", ...);
```

というように関係は文字列などのパラメータで管理され、段階的に関係代数を実行する。これに C++言語のクラスの概念を利用し、**Relation** クラスを設定する。そうすることによって関係の内部を隠し操作は単純化される。関係代数は、

```
R.Select(...).Project(...);
```

のように単純化できる。データベース (**Database**)、関係 (**Relation**)、タプル (**Tuple**)、値 (**Value**) にクラスを設定する。各スキーマ (データベーススキーマ (**DBS**)、関係スキーマ (**RELS**)、属性 (**ATTRS**)、領域 (**DOMS**)) もクラスとして設定する。

DBS ↔ **RELS** ↔ **ATTRS** ↔ **DOMS**
↕　　↕
Database ↔ **Relation** ↔ **Tuple** ↔ **Value**

上のように相互に参照させることで関係からのスキーマ問い合わせ、またその逆を容易に実現する。例えば、「ある関係に属する JPEG 型の画像をすべて抽出」など関係からスキーマに問い合わせることが可能となる。このような操作は利用者から見るとスキーマを扱っていることが明確であるが、内部ではそれはカタログにより関係として扱われる。タプルや値は、関係の要素ではなく、独立させている。関係の操作は、タプル、値と操作されて、最終的に **Value** クラスに対して行われる。そこからの動作は、メソッドのオーバーロードにより、システムが自動的に動作を振り分ける。

---

*User Interface for Meta-Data Programming

†Kohei WATANABE Hideki NOZUE Shingo KASHIKAWA Wataru MATSUMOTO Takao MIURA

‡HOSEI University Dept. of Elec. & Elec. Engr.

3.2 HOMEが提供するクラス

利用者は、データベース、スキーマ、関係の順に作成することでデータベース設計をする。具体的にはDatabase、DOMS、ATTRS、RELS、Value、Tuple、Relationの順に設計する。関係やスキーマは利用者が完全に定義できるが、データ型に対しては利用者に選択方式を提供する。HOMEはあらかじめ基本的なデータ型を用意して、利用者はそれらを使って領域を作成する。データ型はクラスとして提供され、その操作も用意する。データ型クラスは(INT, CHAR, DOUBLE, JPEG, ...)のように定義しておく。利用者が新たなデータ型とその操作を定義することはできない。

HOMEが用意する関係への操作には、Project、Selectなどの関係代数や表示Print、タプルの挿入+=、スキーマの参照Schemaがある。具体化機能は、Eval関数として提供する。メタオブジェクトに対してEvalを実行すると評価された結果を返す。スキーマへの操作は、カタログ操作で実装されている。関係スキーマRELSへメソッド+=, -=を適用することでスキーマ変更も可能にしている。

例 1 次の関係を作るときの利用者の操作を示す。

STUDENT

name	picture
watanabe	pic.jpg

```
db.Connect("test");
        //データベースを「test」という名前で作成
DOMS d0("name", CHAR); DOMS d1("image", JPEG);
        //領域を作成
ATTRS a0("name", &d0); ATTRS a1("picture", &d1);
        //属性を作成
RELS r("STUDENT");      //関係スキーマを作成
r += &a0; r += &a1;     //関係スキーマに属性を追加
Relation R(&r); Tuple t(&R);
        //関係、タプルを作成
Value v0("watanabe"); Value v1("pic.jpg);
        //値を作成
t += &v0; t += &v1;     //タプルに値を追加
R += &t;                //関係にタプルを追加
```

3.3 利用者定義の可能性

複合オブジェクト、特にマルチメディアデータの比較、表示などの実際の動作要求はHOME利用プログラマによって異なる。われわれが示すシステムにおいては、多くの型とその操作をサポートしてはいるが、その内容はごく簡単なものとする。利用者はこのデータ型のクラスをそのまま用いることもできるが、これらのサブクラスを利用者が設計することで、利用者は機能の拡張が可能となる。

例 2 利用者がJPEG型の表示操作を再定義する。

```
class MyJPEG : public JPEG {
...
    Print() { cout << fsize; }
...
};
```

ファイル名ではなく、ファイルサイズが出力されるように変更した。

name	picture
watanabe	300k

4 インターフェイスの実装

3節で提示した設計を実装するにあたって、データ型をシステムは認識し、利用者はしないという違いを解決する。システムがデータ型を認識し、処理を選ぶ内部構造を提示する。また値として複合オブジェクトを指定するときの内部動作も提示する。実装上生成される、中間ファイルの利用者に意識させない処理構造も提示する。

4.1 インターフェイスの構造

3.1で述べたように、各オブジェクトとメタオブジェクトは相互に参照し合っている。関係への操作はタプル、値と切り分けられ、最終的には値に対して行う。値はその領域を走査し、領域によって各データ型のメソッドが呼ばれる。これはメタオブジェクトをクラスとして定義していることで、利用者にはカタログ操作を意識させない、単純なインターフェイスを提供している。

例 3 3.2で挙げた例の関係についてPrint操作を行った場合に呼び出されるクラスメソッド

```
R.Print();     //Relationクラス
R[0].Print(); //Tupleクラス
R[0].val.Print();     //Valueクラス
R[0].valはCHARなので、
((CHAR) R[0].val).Print(); //CHARクラス
が最終的に呼ばれる。
```

4.2 複合オブジェクトの扱い

関係の値に具体化関数を適用した時に、結果が複合値である場合がある。このことからも複合オブジェクトを単純値と同様に扱える環境が重要である。値として関係を設定することによって、表現する。値まで走査された時点で、関係のメソッドまで戻り、再帰的に処理する。

例 4 値に関係が設定されている場合に呼ばれるメソッド

```
R.Print();          //Relationクラス
R[0].Print();       //Tupleクラス
R[0].val.Print(); //Valueクラス
R[0].valがRelationの場合、
((Relation) R[0].val).Print(); //Relationクラスに戻る
((Relation) R[0].val)[0].Print; //Tupleクラス
...
```

4.3 中間ファイルの管理

現在のHOMEシステムでは、関係代数を実行した結果を中間関係として保存している。またその際作られた関係スキーマも不要になる。この中間関係は、ほとんどの場合一度計算が終わってしまえば必要なくなってしまう。[1] 中間関係を、計算結果が画面に表示される時点で削除する。そのため中間関係にフラグを立てておく必要がある。同様に関係スキーマにもフラグを立てる。フラグはシステムが立てるので、利用者は中間ファイルに対する操作を必要としない。

5 結び

HOMEシステムに各オブジェクト、メタオブジェクトに対するクラスをを介してプログラミングレベルでオブジェクトとメタオブジェクトの設計と操作を切り離すことにより、入れ子構造やマルチメディアデータを関係の値として組み込む方式を提案した。これらの設計はプログラミングレベルであるのでわれわれのHOMEシステムの本来の特徴であるオブジェクトとメタオブジェクトを切り離さない管理、ということも内部的に実現されている。Valueクラスの実装により、利用者は型を意識することなくプログラミングが可能になった。

参考文献

[1] Miura, T., Matsumoto, W.: Harmonizing Objects and Meta Objects for Data Warehousing, proc. *IDC* (1999)

[1] 計算順序をたどる最適化のような動作では必要になる場合もある。