マルチタイプオブジェクトに対する被アクセス側による型選択制御法の実現

佐藤秀樹[†] 有次正義[††]

INADAは拡張C++永続プログラミング言語であり，ODMG標準に準拠する．INADAでは，永続オブジェクトに対して動的に型の獲得・喪失を可能とするマルチタイプオブジェクト機能が提供される．この機能を使えば，実世界の実体が持つロール/アスペクトの集合はモデル化可能である．マルチタイプオブジェクトに対するアクセスは，そのオブジェクトが持つ型集合の中から特定の型の選択を必要とする．これまで，この選択はマルチタイプオブジェクトをアクセスするオブジェクトに委ねられていた．しかし，実世界の実体は相対する実体に応じて自らのロール/アスペクトを決定できるという柔軟性を有する．このため，マルチタイプオブジェクトが自らをアクセスするオブジェクトに応じて自らが持つ型集合の中から特定の型を選択する被アクセス側による型選択制御（*AccessEE Controlled Type Selection*; *AEE*）法が求められている．本論文では，この要求に答えるために実装された，INADAのマルチタイプオブジェクトに対する*AEE*法を提示する．この*AEE*法は，(1) ODMG標準に準拠する永続プログラミング言語上の実現，(2) 強く型付けされた言語INADA上の実現，(3) 型選択のための知識は固定的でなく変更可能であること，(4) INADAの言語仕様と処理系を変更することなく簡易に実装されていることを特徴としている．

Implementation of Accessee Controlled Type Selection for Multiple-type Objects

Hideki Sato[†] and Masayoshi Aritsugi[††]

INADA is an enhanced C++ persistent programming language and compliant with ODMG standard. INADA supports *multiple-type object* facility which enables any persistent objects to obtain any type at any time the type is needed, and to lose any unnecessary types dynamically. Using the facility, we can model a set of roles/aspects which a real-world entity possesses. Access to a multiple-type object needs to select one from among its own types. The selection is conventionally left to an object accessing a multiple-type object. A real-world entity is, however, flexible enough to decide its role/aspect depending on a meeting entity whom it exchanges messages with. Accordingly, *AccessEE Controlled Type Selection* (*AEE*) method is highly required, in which a multiple-type object selects one from among its own types depending on an object accessing it. This paper presents the *AEE* method implemented for multiple-type objects of INADA. The characteristics of the method are as follows: (1) implementation on a persistent programming language compliant with ODMG standard, (2) implementation on INADA, a strongly typed language, (3) dynamically changable knowledge for type selction, and (4) simple implementation without any modification to the language specification and the processing system of INADA.

1. はじめに

永続プログラミング言語とオブジェクトデータベース[1)]の利用により，オブジェクトを永続化可能，すなわちそれらを作成するプロセスの終了時点を超えて生存可能にできる．永続/データベース・プログラミング言語では，揮発オブジェクト，すなわち一時的オブジェクトと同様に，永続オブジェクトを作成・操作できる．永続オブジェクトは2次記憶上に割り当てられ，明示的に削除されるまで存在する．実世界の実体は，永続オブジェクトによりモデル化可能である．

実世界の実体は，それが果たす複数の役割（ロール）を持っている．また，実体は多面的な性質を有しており，各々の側面（アスペクト）を通して認識される．さらに，実体が持つロール/アスペクトの集合は，時とともに変化していく．オブジェクトデータベースは

† 大同工業大学情報学部
School of Informatics, Daido Institute of Technology
†† 群馬大学工学部
Faculty of Engineering, Gunma University

表 1 マルチタイプオブジェクト操作のための関数
Table 1 A list of functions for manipulating a multiple-type object.

機能項目	メソッド関数/オペレータ関数	機能の説明
型の付加	template <class T> void d_Ref<T>:: transforms (const d_Ref_Any& base_object)	永続オブジェクト base_object に，この参照が示すオブジェクトを新たな型として付加する．
型の選択	template <class T> void* d_Ref<T>:: as (const char* typename) const	永続オブジェクトより，型名 typename が指定する型への参照を返す．
型の検査	template <class T> int d_Ref<T>:: hastype (const char* typename) const	永続オブジェクトが，型名 typename が指定する型を持っていれば 1 を返す．持っていなければ，0 を返す．
型の削除	void d_Ref_Any::operator delete_of (d_Ref_Any& object, const char* typename)	永続オブジェクト object より，型名 typename が指定する型を削除する．
オブジェクトの削除	void d_Object::operator delete (void*)	永続オブジェクトを削除する．

(*) d_Ref, d_Ref_Any, d_Object は，各々，特定型の永続ポインタクラス，任意型の永続ポインタクラス，永続オブジェクトクラスである[1)]．

実体に対する高いモデル化能力を備えているものの，こうしたロール/アスペクトに対するモデル化能力を提供していない．この欠点を指摘した研究は，その解決のために独自のオブジェクトデータモデルを提案してきた[2)~7)]．著者らも，これらの研究で報告された機能に類似のマルチタイプオブジェクト機能を提案した[8)~10)]．この機能は ODMG 標準[1)]に準拠する，拡張 C++永続プログラミング言語 INADA[8)~10)]上に実装されている．INADA のマルチタイプオブジェクトは，動的に型の獲得・喪失を行う．さらに，マルチタイプオブジェクトは複数の型に対しても唯一のオブジェクト識別子を有する．

従来のオブジェクトデータモデル[1)]では，オブジェクトは唯一の型，すなわちオブジェクトの作成時に割り当てられた型を持ち☆，この型を通してアクセスされる．これに対して，INADA のマルチタイプオブジェクトは複数の型を持ち，この型集合の中から選択された特定の型を通してアクセスされる．これまで，この選択はマルチタイプオブジェクトをアクセスするオブジェクトに委ねられていた．しかし，実世界の実体は相対する実体に応じて，自らのロール/アスペクトを決定するという柔軟性を有する．このため，マルチタイプオブジェクトが自らをアクセスするオブジェクトに応じて自らが持つ型集合の中から特定の型を選択する被アクセス側による型選択制御（*AccessEE*☆☆ *Controlled Type Selection*; 以降では *AEE* と記す）法が求められている．*AEE* 法は，クラスバージョン機構[11)]において提案された．しかし，クラスバージョン機構はオブジェクトデータモデルを拡張した独自のモデルに基づいており，INADA のマルチタイプオブジェクトに適用することはできない．

本論文では，INADA のマルチタイプオブジェクトに対して実現された，被アクセス側による型選択制御 *AEE* 法の実装を提示する．この *AEE* 法は，(1) ODMG 標準に準拠する永続プログラミング言語上の実現，(2) 強く型付けされた言語 INADA 上の実現，(3) 型選択のための知識は固定的でなく変更可能であること，(4) INADA の言語仕様と処理系を変更することなく簡易に実装されていることを特徴としている．

本論文の構成は，以下のとおりである．2 章では，オブジェクトの型選択制御法について述べる．3 章では，マルチタイプオブジェクトに対する *AEE* 法の実装を議論する．4 章では，本研究の関連研究に言及する．最後に，5 章では本論文をまとめる．

2. 型選択制御法

本章では，まずマルチタイプオブジェクトに対する型選択制御法について述べる．次に，*AEE* 法の先行研究を示す．

2.1 マルチタイプオブジェクトの型選択制御法

INADA はオブジェクト識別性を保持しつつ，永続オブジェクトが持つ型集合を動的に変更するマルチタイプオブジェクト機能[8)~10)]を備える．**表 1** は，マルチタイプオブジェクトを操作するための関数群を示す．これらの関数は，永続オブジェクトに対する型の付加（transforms），永続オブジェクトからの型の選択（as），永続オブジェクトが特定の型を持つか否かの検査（hastype），永続オブジェクトからの型の削除（delete_of），永続オブジェクト自体の削除（delete）といった機能を提供する．

INADA の永続オブジェクトは，任意の時点において複数の型を持ちうる．このため，オブジェクトのア

☆ オブジェクトは暗黙的にそれが作成された型の上位型も持つが，ここでは上位型はオブジェクトの型としての数には含めていないことを断っておく．

☆☆ 本論文では，accessee は他オブジェクトによりアクセスされる被アクセス側のオブジェクトを意味する．また，accessor は他オブジェクトをアクセスするアクセス側のオブジェクトを意味する．

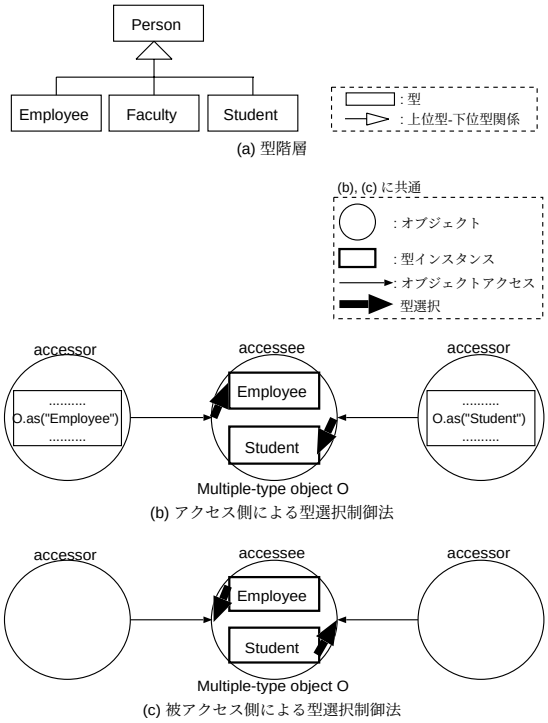


図 1 マルチタイプオブジェクトに対するアクセスのための型選択制御法
Fig. 1 Type selction methods for accessing a multiple-type object.

クセスに際しては，複数の型の中から特定の型を選択することが必要となる．**図 1** は，マルチタイプオブジェクトに対する型選択制御法を示す．図 1 (a) は，図 1 (b)，(c) において参照される型階層を表す．

図 1 (b) では，accessor は型を選択するためにメソッド `as` を起動し，選択された型を通してマルチタイプオブジェクトをアクセスする．本論文では，この種の型選択法をアクセス側による型選択制御（*Access OR Controlled Type Selection*；以降では *AOR* と記す）法と呼ぶ．*AOR* 法では，マルチタイプオブジェクトが持つ型集合の中から特定の型を選択するための知識は，それに対するアクセス側である accessor の集合に分散されている．

図 2 は，*AOR* 法の型選択知識を実装する INADA プログラムコードの例を示す．変数 `s` は，`Employee` 型，`Faculty` 型，`Student` 型のいずれかを有するオブジェクトを指す OID の集合を参照する．`Employee` 型，`Faculty` 型，`Student` 型の各々はそれぞれの型に応じて異なるメッセージを出力する多態的（polymorphic）なメソッド `print()` を備えていることが仮定されている．このプログラムコードの実行により，accessor はアクセスされるマルチタイプオブジェクトの特定の型を選択できる．

```
d_Ref<d_Set<d_Ref<Person>>> s;
d_Ref<Person> p;
d_Iterator<d_Ref<Person>> it=s->create_iterator();
while(it.next(p)){
    if (p.hastype("Employee")){
        ((d_Ref<Employee>)p.as("Employee"))->print();
    }
    else if (p.hastype("Faculty")){
        ((d_Ref<Faculty>)p.as("Faculty"))->print();
    }
    else{
        ((d_Ref<Student>)p.as("Student"))->print();
    }
}
```

図 2 アクセス側による型選択制御を実現する INADA プログラムコードの例
Fig. 2 An example of INADA program codes for accessor controlled type selection.

表 2 実世界の実体が持つ型選択知識の例
Table 2 Examples of type selection knowledge of a real-world entity.

例	型選択知識の内容
例 1	ある人が従業員と対面していることとする．前者が後者の上司であるならば，前者は後者に対して管理者として応じる．
例 2	ある人が学生と対面していることとする．前者が後者の担当者であるならば，前者は後者に対して指導教員として応じる．

図 1 (c) は，マルチタイプオブジェクトの型選択に関する *AEE* 法を示している．図では，accessee 自身が自らの型集合の中から特定の型を選択する．このため，マルチタイプオブジェクト自身が型集合の中から特定の型を選択するための知識を有する．実世界の実体は，**表 2** の例のように，相対する実体に応じて自身のロール/アスペクトを決定する．型選択知識は，これらの内容を記述する．*AEE* 法の場合，accesssor は accessee をアクセスするために選択される型を知らない．代わって，accessor は accessee の一般的な型である `Person` 型を指定する．このため，accessor は `Person` 型の属性，関連，メソッド（仮想メソッド[1)]を含む）を通して accessee をアクセスできる．

2.2 クラスバージョン機構

クラスバージョン機構[11)]は，オブジェクトの多面性表現を実現するため提案された．この機構により，相対するオブジェクトや状況に応じたオブジェクトの振舞いが可能となる．**図 3** は，クラスバージョン機構の構成を示す．バージョンは実体の各々の側面に対応しており，間連する属性とメソッドを備える．クラスは，バージョン集合からなる．バージョン集合はバージョン階層を構成し，この階層に基づきバージョン記述の

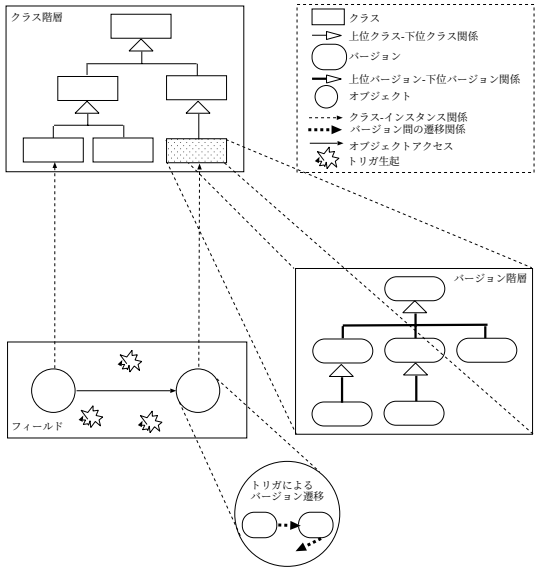


図 3 クラスバージョン機構の構成
Fig. 3 Class version mechanism.

継承が行われる．クラスバージョン機構では，クラスのインスタンスであるオブジェクトはフィールドに登録される．トリガがフィールド内のオブジェクトを監視しており，オブジェクトの間にトリガが規定する関係条件が成立すると，該当するオブジェクトのバージョンが遷移する．トリガは，(1) フィールド内のオブジェクトにバージョン遷移が起きたときに，(2) フィールド内のオブジェクト間でメッセージがやりとりされたときに，(3) フィールドの情報が変化したときに，評価される．オブジェクトアクセス時には，被アクセス側のオブジェクトのその時点におけるバージョンが持つ属性あるいはメソッドがアクセスのために使われる．

バージョンとマルチタイプオブジェクトの型とを対応付ければ，クラスバージョン機構におけるオブジェクトアクセス時のバージョン選択の制御は *AEE*法によるものといえる．しかし，以下の理由より，INADA のマルチタイプオブジェクトに対する *AEE*法の実現のために，クラスバージョン機構を用いることはできない．

- クラスバージョン機構はフィールドとトリガの両概念に基づいており，オブジェクトデータモデルを拡張した独自のモデルに基づいている．このため，クラスバージョン機構をオブジェクトデータベースシステムに適用することはできない．
- クラスバージョン機構のバージョン集合は動的に変更できないため，オブジェクト作成前に設計されなければならない．しかし，永続オブジェクトは 2 次記憶上に長期間存在するため，将来も見通して永続オブジェクトのバージョン集合を設計することは困難である．
- クラスバージョン機構はクラス内のバージョン階層に沿ったバージョン継承（一般バージョン継承）のほかに，上位クラスのバージョン階層からのバージョン継承（クラス越境バージョン継承，一般継承，同バージョンリンク）を備えている．このようなバージョン継承の制御は複雑多様であり，クラスバージョン機構は大規模システムの構築には適していない．
- クラスバージョン機構は弱く型付けされた言語 CLOS [12] 上に実装されており，型の不整合による問題をコンパイル時に発見できない．

3. マルチタイプオブジェクトに対する *AEE*法の実装

本章では，マルチタイプオブジェクトに対する *AEE*法の実装を議論する．まず，*AEE*法実現のための設計方針を示す．次に，*AEE*法で用いる型選択知識のルール表現を検討する．引き続き，ルール集合の実装について述べる．最後に，ルール集合の処理操作を提示する．

3.1 設計方針

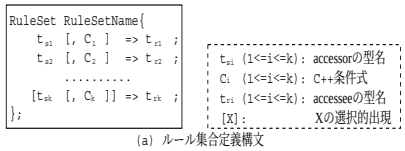
以下は，マルチタイプオブジェクトに対する *AEE*法の実装に課せられた設計方針である．

- 従来の INADA の言語仕様と処理系との互換性を保持する．このため，*AEE*法の実装では，INADA の言語仕様と処理系を変更しない．
- *AEE*法と *AOR*法は競合することなく，共存可能とする．すなわち，同一のマルチタイプオブジェクトに対して，プログラムのある箇所では *AOR*法により，別の個所では *AEE*法によりアクセスが可能とする．
- 永続オブジェクトは 2 次記憶上に長期間存在するため，将来も見通して *AEE*法の型選択知識を設計することは困難である．このため，型選択知識は固定的ではなく，動的に変更可能とする．

3.2 型選択知識に対するルール表現

以下は，*AEE*法における型選択知識の表現に対する設計方針である．

- 型選択に関する知識表現のためにルール形式を用いる．
- accessor の型は，ルールの前件部で指定される（表 2 の例 1 の「従業員」，例 2 の「学生」）．
- accessor の制限条件は，ルールの前件部で指定さ



(a) ルール集合定義構文

```
RuleSet rule_set01{
    Employee, ((Employee*)ACCESSOR)->boss()==ACCESSEE        =>  Manager;
    Student,  ((Student*)ACCESSOR)->supervisor()==ACCESSEE   =>  Faculty;
                                                             =>  Person;
};
```

(b) ルール集合定義例

図 4 ルール集合の定義
Fig. 4 Definition of a rule set.

れる（表 2 の例 1 の「前者は後者の上司である」，例 2 の「前者は後者の担当者である」）．

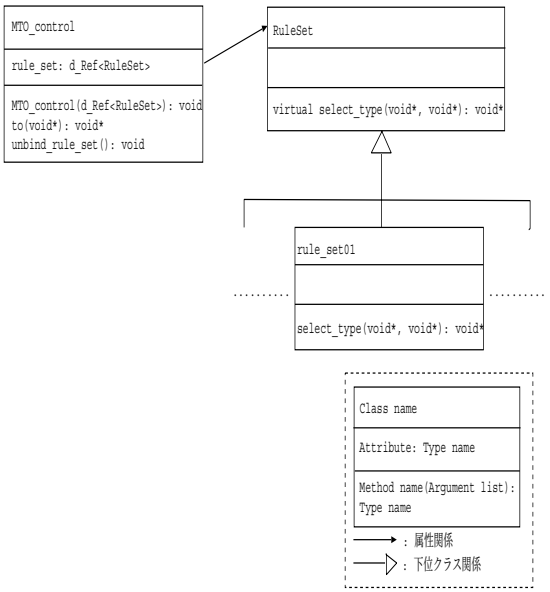
- 選択される accessee の型は，ルールの後件部に指定される（表 2 の例 1 の「前者は後者に対して管理者として応じる」，例 2 の「前者は後者に対して指導教員として応じる」）．

図 4 (a) は，上記の設計方針に従うルール集合定義構文を示す．“$t_{si}[, C_i] => t_{ri}$” $(1 \leq i \leq k)$ はルール集合の i 番目のルールであり，t_{si} は accessor の型を，C_i は accessor に関する選択的な制限条件を，t_{ri} は accessee の型を指定する．変数 `ACCESSEE` と変数 `ACCESSOR` は，C_i を構成するために使われる．前者は accessor の型を示し，後者は accessee の型を示す．ルールは，accessor の型が t_{si} であり，t_{si} が制限条件 C_i を満足し，かつマルチタイプオブジェクトが型 t_{ri} を保持するならば，マルチタイプオブジェクトは型 t_{ri} を通してアクセスされることを述べている．ルール集合の各ルールは，優先度の降順に整列している．複数のルールがその前件部を満足するならば，最も優先度の高いルールが適用されるために選択される．最後尾のルールでは，t_{sk} あるいは C_k が省略可能であり，この場合，accessor に制限は課せられない．図 4 (b) は，表 2 の例 1 と例 2 に対するルール集合の定義例を示す．

3.3 ルール集合の実装

*AEE*法実現のための設計方針として，マルチタイプオブジェクトに対する型選択知識は固定的ではなく，変更可能とすることが課せられている．このため，永続オブジェクトに対して動的に型の獲得・喪失を可能とするマルチタイプオブジェクト機能自体を，次のように *AEE*法のルール集合の実装に利用する．

(1) マルチタイプオブジェクトが保持する型集合の中から特定の型を選択するためのシステム定義型 `MTO_control` を用意する．
(2) マルチタイプオブジェクトに対して，`MTO_control` 型を付加する．
(3) マルチタイプオブジェクトがアクセスされるとき，`MTO_control` 型のメソッドが起動され，マルチタイプオブジェクトが保持する型集合の中から特定の型が選択される．
(4) `MTO_control` 型は，不必要になったならば，付加されたマルチタイプオブジェクトから削除される．
(5) 型選択を行うメソッドは，ルール集合の定義に基づき生成される．



(a) 型選択に関するクラス図

```
MTO_control::MTO_control(d_Ref<RuleSet> rs){
    this->rule_set=rs;
}

void* MTO_control::to(void* ACCESSOR){
    return this->rule_set->select_type(ACCESSOR, (void*)this);
}

void MTO_control::unbind_rule_set(){
    delete this->rule_set;
}
```

(b) MTO_controlクラスの実装

図 5 型選択のためのクラス
Fig. 5 Classes for type selection.

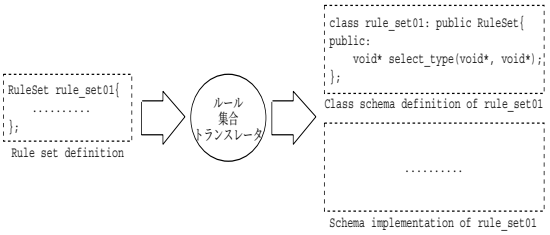
図 5 (a) は，型選択に関係するクラス図を示す．システム定義クラス `MTO_control` は，型選択を制御する型を提供する．`MTO_control` 型は，マルチタイプオブジェクトに付加される．`RuleSet` クラスは，型集合の中から特定の型をどのように選択するかを実際に定義するクラス（例. 図 5 (a) の `rule_set01`）の上位クラスである．`RuleSet` クラスは，`MTO_control` クラスの `rule_set` 属性のドメイン型である．`RuleSet` クラスの

select_type メソッドは，accessor を第 1 引数に，accessee を第 2 引数にとる．これは，accessor が accessee をアクセスするために用いる accessee の型を選択する．select_type メソッドは多態的（polymorphic）であるので，実際に実行されるメソッドは RuleSet クラスの下位クラスに定義される．RuleSet クラスの各下位クラスは，後述の対応するルール集合の定義に基づき生成される．

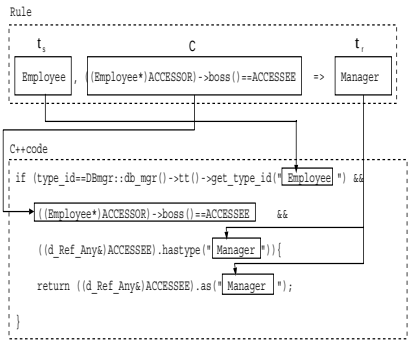
図 5 (b) は，MTO_control クラスのメソッドの実装を示す．MTO_control クラスのコンストラクタメソッドは，rule_set 属性に引数として与えられる RuleSet クラスのインスタンスの参照値を代入する．メソッド to は，RuleSet 型に型選択を依頼する．メソッド unbind_rule_set は，rule_set 属性が参照する RuleSet 型を削除する．

図 6 は，ルール集合定義の，RuleSet クラスの対応する下位クラスに関する定義への変換を示す．図 6 (a) のルール集合トランスレータはルール集合定義を入力し，RuleSet クラスの下位クラスの定義を出力する．図 6 (b) は，変換単位としての単一のルールと対応する C++プログラムコードとの間の対応を示す．accessor の型（ルールの t_s），accessor の制限条件（ルールの C），選択される accessee の型（ルールの t_r）といった情報が，生成される対応する C++プログラムコードの可変部分に埋め込まれる．C++プログラムコードの第 1 行目の type_id 変数には，accessor の型の番号が割り当てられていることとする．第 1 行目で，メソッド tt() は型に関する情報を保持する表のポインタアドレスを返し，メソッド get_type_id(s) は文字列 s が指定する型の番号を返す．したがって，第 1 行目の連言項は accessor の型が t_s が指定する型に一致することを意味している．第 2 行目の連言項は，accessor の制限条件そのものである．第 3 行目の連言項は，accessee が t_r が指定する型を保持することを意味している．1〜3 行目の各々の条件が同時に満足されるならば，ルールは適用可能となる．この場合，accessee の型 t_r が選択され，accessor に返される（4 行目参照）．

図 6 (c) は，ルール集合 rule_set01 に対して生成された select_type メソッドの C++プログラムコードを示す．select_type メソッドの第 1 引数の変数名は ACCESSOR であり，第 2 引数の変数名は ACCESSEE であり，各々の値はその名前が出現するメソッド本体中の全箇所で参照可能である．第 2 行目で，メソッド get_type_id() の起動により，type_id 変数には accessor の型の番号が割り当てられる．ルール集合中の各々のルールに対応する C++プログラムコードは，ルール集合中のルールと同じ順序で出力される．前述のように，ルールの順序は優先度を反映している．最後尾の箇所には，適用可能なルールがない場合に備えて，“return NULL;” 文が置かれる（図 6 (c) の 16 行目参照）．



(a) ルール集合トランスレータ



(b) ルールと生成される INADA プログラムコードとの対応

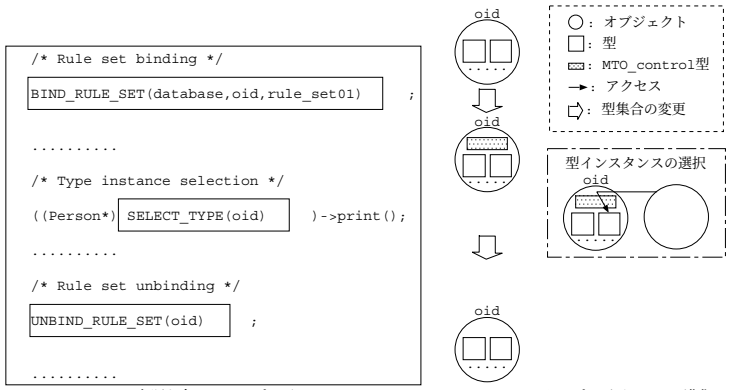
```
line-no.
1       void* rule_set01::select_type(void* ACCESSOR, void* ACCESSEE){
2           int type_id=((d_Ref_Any&)ACCESSOR).get_type_id();
3           if (type_id=DBmgr::db_mgr()->tt()->get_type_id("Employee") &&
4               ((Employee*)ACCESSOR)->boss()==ACCESSEE &&
5               ((d_Ref_Any&)ACCESSEE).hastype("Manager")){
6               return ((d_Ref_Any&)ACCESSEE).as("Manager");
7           }
8           if (type_id=DBmgr::db_mgr()->tt()->get_type_id("Student") &&
9               ((Student*)ACCESSOR)->supervisor()==ACCESSEE &&
10              ((d_Ref_Any&)ACCESSEE).hastype("Faculty")){
11              return ((d_Ref_Any&)ACCESSEE).as("Faculty");
12          }
13          if (type_id=DBmgr::db_mgr()->tt()->get_type_id("Person")){
14              return ((d_Ref_Any&)ACCESSEE).as("Person");
15          }
16          return NULL;
17      }
```

(c) rule_set01 のスキーマ実装

図 6 ルール集合定義の RuleSet 下位クラス定義への変換
Fig. 6 Translation of a rule set definition into a subclass definition of RuleSet.

3.4 ルール集合の処理操作

BIND_RULE_SET マクロ，UNBIND_RULE_SET マクロ，SELECT_TYPE マクロが，ルール集合の処理操作のために定義される．**図 7** は，このマクロ処理を示す．図 7 (a) は，マクロ参照を含む INADA プログラムの断片を示す．まず，BIND_RULE_SET マクロは，ルール集合をマルチタイプオブジェクトに付加するために使われる．



(a) マクロ参照を含むINADAプログラムコード　(b) マルチタイプオブジェクトの進化

```
/* Rule set binding */
((d_Ref<MTO_control>)new(database,"MTO_control")MTO_control(
 (d_Ref<RuleSet>)new(database,"rule_set01")rule_set01()).transforms(oid)) ;
.....
/* Type instance selection */
((Person*) ((MTO_control*)oid.as("MTO_control")).to((void*)this) )->print();
.....
/* Rule set unbinding */
((MTO_control*)oid.as("MTO_control")).unbind_rule_set(),
 delete_of(oid,"MTO_control") ;
.....
```

(c) 前処理後のINADAプログラム断片

図 7 マルチタイプオブジェクトに対するルール集合操作
Fig. 7 Rule set manipulation over a multiple-type object.

このマクロの第 1 引数は，型 d_Database [1)] のオブジェクトに対する参照である．第 2 引数は，マルチタイプオブジェクトの OID である．第 3 引数は，付加されるルール集合の名前である．次に，SELECT_TYPE マクロは，マルチタイプオブジェクトをアクセスするために使われる．このマクロの第 1 引数は，マルチタイプオブジェクトの OID である．最後に UNBIND_RULE_SET マクロは，マルチタイプオブジェクトから不必要となったルール集合を取り除くために使われる．図 7 (b) は，マルチタイプオブジェクトに対する操作履歴の例を図示する．

図 7 (c) は，マクロ参照処理後の INADA プログラムの断片を示す．図 7 (a) の各マクロ参照は，図 7 (c) では INADA(C++) プログラムコードで置き換えられている．BIND_RULE_SET マクロは RuleSet クラスの rule_set01 下位クラスのインスタンスを作成し，このインスタンスを参照する MTO_control インスタンスを作成し，オブジェクト識別子 oid が参照するマルチタイプオブジェクトに MTO_control インスタンスを付加する．SELECT_TYPE マクロは，マルチタイプオブジェクトの MTO_control 型のメソッド to を実行する．擬似変数 this をメソッド to の引数とすることにより，accessor が受け渡される．UNBIND_RULE_SET マクロは，RuleSet クラスの下位クラスのインスタンスに対するマルチタイプオブジェクトの MTO_control 型からの参照関係を取り消す．その後，MTO_control 型が削除される．

4. 関 連 研 究

本章では，本論文で述べたマルチタイプオブジェクトに対する *AEE* 法の関連研究に言及する．なお，*AEE* 法に基づくクラスバージョン機構[11)] に関しては，2.2 節を参照されたい．

オブジェクトデータベースにおけるビュー[13)~18)] は，オブジェクトに対する複数の異なる見方を提供する．ビューは，基底オブジェクトまたは他のビューに基づく仮想型として定義される．この結果，ビューはオブジェクトが複数の型を持つことを可能とする．ビューを通したオブジェクトに対するアクセスは，accessor によるビュー指定，すなわち *AOR* 法☆により行われる．

☆ 本節では，*AOR* 法は広義の意味でとらえられていることを断っ

Iris[2]は，実体の振舞い進化に対するモデル化機能を提供する．オブジェクトは一生の間に型の獲得・喪失を行うが，オブジェクト識別性は維持される．しかし，オブジェクトは任意の時点ではどの文脈においても確定的に振る舞う．したがって，オブジェクトをアクセスするために，型選択は必要ない．

Sciore[3]は，オブジェクト階層において親オブジェクトに振舞いを委譲する継承機構を提案する．継承はオブジェクトごとに異なるが，振舞いはオブジェクト階層の構造に依存する．このため，オブジェクトをアクセスするために，型選択は必要ない．

Clover[4]は，複数のロールを持つオブジェクトをモデル化できる．型 T のオブジェクトが型 T' のインスタンスであれば，T' に特有なメソッドとデータを有する．このモデルは型チェック用の演算子に加えて，型階層を上昇/下降するための演算子を備えている．オブジェクトの振舞いは型に厳密に依存し，オブジェクトに対するアクセスは型指定による *AOR*法による．

Aspect[5]はオブジェクトが複数アスペクトを保持可能とし，かつオブジェクト識別性を損なうことなく，オブジェクトが一生の間に新しい型で拡張可能とする．この実装は強く型付けされた言語に基づいており，各々の型は固有なメソッドとデータを備えている．オブジェクトに対するアクセスは，型指定による *AOR*法による．

Fibonacci[6]は，複数ロールを持つオブジェクトのモデル化機構を備える，強く型付けされた言語である．オブジェクトの振舞いは，オブジェクトアクセスのために指定されるロールに依存する．ロール指定は accessor により行われるため，オブジェクトに対するアクセスは *AOR*法による．

Gottlob ら[7]は，複数ロールにより拡張可能なオブジェクトを提案する．クラス階層は，ロール階層により補完される．オブジェクトに対するアクセスは accessor によるロール指定，すなわち *AOR*法による．この実装は，Gemstone[19]の Smalltalk[20]ベースのデータベース・プログラミング言語 OPAL を利用する．INADA とは異なり，OPAL は弱く型付けされた言語である．

多態性（polymorphism）[1]は，正確に型を指定することなく，オブジェクトのメソッドを起動する機能である．多態性は遅延束縛機構により，従来型オブジェクトが持つ単一の型インスタンスに基づきメソッドを選択する．一方，本論文で述べた *AEE*法はルール評価機構により，マルチタイプオブジェクトが持つ複数の型インスタンスの中から特定の型を選択する．

5. おわりに

本論文は，INADA のマルチタイプオブジェクトに対する *AEE*法の実装について述べた．以下は，この実装の骨子である．

(1) accessee が自らの型集合の中から特定の型を選択するための知識の表現にルール形式を用いている．

(2) システム定義型がマルチタイプオブジェクトに付加される．この型は，ルール集合を使って自身の型集合の中から特定の型を選択するメソッドを備えている．

(3) マルチタイプオブジェクトに対するアクセス時，上記のメソッドが accessor に応じて特定の型を選択する．選択された型を通して，アクセスが行われる．

以下は，実装された *AEE*法の特徴である．

- *AEE*法は，ODMG 標準[1]に準拠する永続プログラミング言語 INADA 上に実装されている．
- *AEE*法は，強く型付けされた言語 INADA の長所を損なうことはない．
- 型選択のための知識は固定的ではなく，変更可能である．
- *AEE*法と *AOR*法は競合することなく，共存可能である．
- *AEE*法は INADA の言語仕様と処理系を変更することなく，簡易に実装されている．

著者らは今後の検討課題として，問合せ言語 OQL[1]上での *AEE*法の利用を考えている．また，マルチタイプオブジェクトに関する一貫性制約管理は，別の検討課題である．さらに，著者らは *AEE*法がマルチタイプオブジェクトに対するアクセス制御機構ならびにスキーマ進化[21]処理機構として潜在的に利用可能であると考えており，これらの検討も予定している．

謝辞 本論文を改善するうえで査読者から，有益なご意見をいただきましたことに対し深謝いたします．

ておく．すなわち，accessor はビュー名，ロール名，型名を指定して accessee をアクセスできる．

参考文献

1) Cattel, R.G.G. and Barry, D.K. (Eds.): *The Object Data Standard: ODMG3.0*, Morgan Kaufmann (2000).

2) Fishman, D.H., Beech, D., Cate, H.P., Chow, E.C., Connors, T., Davis, J.W., Derrett, N., Hoch, C.G., Kent, W., Lyngback, P., Mahbod, B., Neimat, M.A., Ryan, T.A. and Shan, M.C.:

Iris: An Object-Oriented Database Management System, *ACM Trans. Office Information Systems*, Vol.5, pp.48–69 (1987).
3) Sciore, E.: Object Specialization, *ACM Trans. Office Information Systems*, Vol.7, pp.103–122 (1989).
4) Steing, L.A. and Zdonik, S.B.: Clovers: The Dynamic Behavior of Type and Instances, Brown University, Technical Report, No.CS-89-42 (1989).
5) Richardson, J. and Schwardz, P.: Aspects: Extending Objects to Support Multiple, Independent Roles, *Proc. ACM Int. Conf. on Management of Data*, pp.298–307 (1991).
6) Albano, A., Bergamini, R., Ghelli, G. and Orsini, R.: An Object Data Model with Roles, *Proc. Int. Conf. on Very Large Data Bases*, pp.39–51 (1993).
7) Gottlob, G., Schrefl, M. and Rock, B.: Extending Object-Oriented Systems with Roles, *ACM Trans. Office Information Systems*, Vol.14, No.3, pp.268–296 (1996).
8) Aritsugi, M. and Makinouchi, A.: Design and Implementation of Multiple Type Objects in a Persistent Programming Language, *Proc. 19th Int. Computer Software and Applications Conf.*, pp.70–76 (1995).
9) Aritsugi, M.: Studies on Facilities for Persistent Programming Languages and Their Implementations, Doctoral dissertation, Kyushu University (1995).
10) Aritsugi, M. and Makinouchi, A.: Multiple-type Objects in an Enhanced C++ Persistent Programming Language, *Software-Practice and Experience*, Vol.30, pp.151–174 (2000).
11) 志村秀人，上田賀一：オブジェクトの多面性表現のためのクラスバージョンの導入，情報処理学会研究報告ソフトウェア工学，Vol.101, pp.25–32 (1994).
12) Keene, S.E. and Gerson, D.: *Object-Oriented Programming in Common Lisp: A Programmer's Guid to CLOS*, Addison-Wesley (1989).
13) Tanaka, T., Yoshikawa, M. and Ishihara, K.: Schema Virtualization in Object-Oriented Databases, *Proc. Int. Conf. on Data Engineering*, pp.23–30 (1988).
14) Mamou, J.C. and Medeiros, C.: Interactive Manipualtion of Object-Oriented Views, *Proc. Int. Conf. on Data Engineering*, pp.60–69 (1991).
15) Abiteboul, S. and Bonner, A.: Objects and Views, *Proc. ACM Int. Conf. on Management of Data*, pp.238–247 (1991).
16) Scholl, M., Laasch, C. and Tresch, M.: Updatable Views in Object-Oriented Databases, *Proc. Int. Conf. on Deductive and Object-Oriented Databases*, pp.189–207 (1991).
17) Rundensteiner, E.: Multiview: A Methodology for Supporting Multiple Views in Object-Oriented Databases, *Proc. Int. Conf. on Very Large Data Bases*, pp.187–198 (1992).
18) Amano, H., Aritsugi, M. and Makinouchi, A.: The Multiple-type Mechanism and View Functions of an Object-Oriented Persistent Programming Language INADA, *ADVANCES IN COMPUTING TECHNIQUES — Algorithms, Databases and Parallel Processing*, Imai, H., Wong, W.F. and Loe, K.F. (Eds.), pp.123–135, World Scientific (1995).
19) Butterworth, P., Otis, A. and Stein, J.: The Gemstone Object Database Management System, *Comm. ACM*, Vol.34, pp.64–77 (1991).
20) Goldberg, A. and Robson, D.: *Smalltalk-80 The Language and Its Implementation*, Addison-Wesley (1983).
21) Zaniolo, C., Ceri, S., Faloutsos, C., Snodgrass, R., Subrahmanian, V. and Zicari, V.: *Advanced Database Systems*, Morgan Kaufmann (1997).

(平成 14 年 12 月 27 日受付)
(平成 15 年 4 月 2 日採録)

(担当編集委員 石川 博)

佐藤 秀樹(正会員)

1975 年名古屋大学工学部電気工学科卒業．同年,(株) 富士通研究所入社．1989 年日本電装 (株)，1998 年愛知学泉大学を経て，2002 年より大同工業大学情報学部教授．この間，データベースシステムの研究開発に従事．博士(工学)．電子情報通信学会，ACM，IEEE Computer Society，日本データベース学会各会員．

有次 正義(正会員)

1991 年九州大学工学部情報工学科卒業．1996 年九州大学大学院博士後期課程修了．同年，群馬大学．現在，同大学工学部情報工学科助教授．データベースシステム，分散並列データ処理等に興味を持つ．博士 (工学)．電子情報通信学会，ACM，IEEE Computer Society，日本データベース学会各会員．