セットメンバアノテーションに基づく版管理機構の提案

宮脇忠光† 北川博之††

近年，CADシステム，ソフトウェア開発等のエンジニアリング分野におけるデータベースシステムの利用拡大に伴い，版（バージョン）管理の重要性が認識されている．しかし，版管理に対する要求は管理対象となるオブジェクトの種類や設計環境によって多様であるため，それぞれの要求に基づいた版管理を行うことが必要である．このためには，版管理の方式によって異なった種類の情報や手続きを，版管理の対象である各版を表すオブジェクトに付与するための何らかの仕組みが要求される．本研究は，その1つのアプローチとしてセットメンバアノテーションの概念を提案し，それをベースとした版管理機構の実現について述べる．セットメンバアノテーションとは，オブジェクトがある集合の要素となった時点で，集合の要素としてそのオブジェクトが持つべき付加的な情報や手続きを動的にアノテーションオブジェクトとして付与する機構である．本研究では，ある設計対象に対する版の集まりである版集合の要素としてオブジェクトが登録された時点で，ユーザ要求に基づいた版管理を行う上で必要な各種の情報や手続きをアノテーションオブジェクトとして付与することにより，版管理を実現する．本論では，セットメンバアノテーションの概念をオブジェクト指向データモデルに導入し，具体的エンジニアリングデータベースの例として，ソフトウェア開発データベースを取り上げ，その版管理機構の実現を示す．

A Version Management Scheme Based on the Set-Member Annotation

Tadamitsu Miyawaki† and Hiroyuki Kitagawa††

In support of engineering database applications such as CAD and software development, version management is an indispensable feature required for database systems. However, application requirements for version management vary depending on target object types and design environments. To cope with such a varity of requirements, some scheme is required to associate requirements-oriented attributes and procedures with objects which are targets of version management. This paper proposes the concept of set-member annotation and shows its application to version management as an approach to this issue. The set-member annotation means that an object is dynamically tagged with some supplementary attributes and procedures when the object becomes a member of a set. In our scheme, when an object is registered into a version set as a version, the version object is tagged with an annotation object which has attributes and procedures required from version management policy. We show an object-oriented data model incorporating the set-member annotation and demonstrate its use in a software development database system providing three types of version management functions.

1. はじめに

近年のエンジニアリング分野におけるデータベース利用の活発化に伴い，データベースにおける版（バージョン）管理の重要性が認識されている[1),2)]．

版管理が必要とされるデータベースアプリケーションの一例としてソフトウェア開発が挙げられる．ソフトウェア開発作業は通常試行錯誤を繰り返しながら，いくつもの段階を経ることによって進められ，その過程は版として管理される．開発を進めた結果，ある一定の水準に達したオブジェクトはリリースされる．また，ソフトウェア開発では外部から導入したソフトウェアシステムやライブラリを部品として利用することも多い．これら外部から導入したオブジェクトにも通常各種の版が存在する．

このようなソフトウェア開発を支援するためには，管理対象のオブジェクトの性質に応じて，異なる版管理の方式をきめ細かに提供することが要求される．例えば，開発途中のオブジェクトを対象とする版管理においては，各版（開発版オブジェクト）の版番号，登録日時，「どの版に変更を加えて導出された版であ

† 富士ゼロックス株式会社
FUJI XEROX CO., LTD.
†† 筑波大学電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba

るか」という版間の導出関係等の情報を管理すると共に，初版を登録する手続きや既に登録済みの版から新たな版を導出する手続き等を提供することが必要である．また，リリースされたオブジェクトを対象とする版管理においては，各版（リリース版オブジェクト）のリリース番号，リリース日時，リリース管理者，「どの版の機能を継承した版であるか」という版間の機能的継承関係等のリリース版オブジェクト特有の情報を管理すると共に，一定のチェックを行った上でオブジェクトをリリース版オブジェクトとして登録する手続き等の提供が必要である．さらに，外部から導入したソフトウェアシステムやライブラリ等の外部オブジェクトは直接的に開発の対象ではないため，導出関係や機能的継承関係のような版（外部版オブジェクト）間の関係を明示的に管理する必要は通常無いが，導入日時や導入担当者等のような外部版オブジェクト特有の情報の管理と，外部版オブジェクトとしてオブジェクトを登録する手続き等の提供が必要である．

一般に，既にリリースされたオブジェクトをさらに改良するための開発が行われる場合も多い．この場合，リリース版オブジェクトから新たな開発版オブジェクトを導出することになる．従って，その場合対象となる導出元のオブジェクトに対しては，上で述べたようなリリース版オブジェクトとしての版管理のための情報や手続きのみならず，導出関係等の開発版オブジェクトとしての版管理のための情報や手続きをも付与する必要がある．すなわち，リリースされたオブジェクトはリリース版オブジェクトに対する版管理方式のみならず，開発版オブジェクトに対する版管理方式の適用対象にもすることが要求される．

さらにまた，一般に開発環境では，テスト的に生成したオブジェクトが結果的には開発の中で利用されるといった場合も多い．このため，当初版管理の対象として想定していなかったオブジェクトを後から版管理の対象にしたいという状況も珍しくはない．

以上のような要求は，ソフトウェア開発に限ったものではなく，設計データベースにおける版管理においてしばしば発生するものである．従って，データベースシステムの提供する版管理機構は，少なくとも以下のような要求条件を満足するものであることが望まれる．

1) 版管理の方式を対象オブジェクトごとにきめ細かく設定できること．このためには，管理対象のオブジェクトに対して，それぞれの版管理方式に応じて必要とされる異なったデータ要素や手続きを付与することが必要である．
2) 同一のオブジェクトを複数の版管理方式で管理できること．
3) オブジェクトを生成後に版管理の対象にできること．

これまでにも，リレーショナルデータモデル，関数型データモデル，オブジェクト指向データモデル等を基に，様々な版管理に関する研究が行われてきた[3)~10)]．特に，オブジェクト指向データベース管理システム（OODBMS）においては版管理は重要な基本機能として認識されており，いくつかのシステムでは版管理機構を提供している．しかし，それらのほとんどはデータベース中のすべてのオブジェクトに画一的な版管理方式を提供しているため，上で述べたような版管理の要求に十分に対応できない．

これに対し，近年，特に上記 1)の要求条件を満たす版管理機構の実現を目指した研究も幾つか行われている[11)~16)]．オブジェクト指向データモデルに基づくアプローチ[13)~15)]では，クラスを定義する際にそのクラスのオブジェクト固有の版管理方式を決定し，それを実現するのに必要なデータ要素や手続きをクラス定義に含めることにより，上記 1)への対応を図っている．これらのアプローチでは，クラスごとに版管理の有無やその方式がデータベース設計時に決まり，各オブジェクトが版管理の対象となるか否か，またその方式がそのクラスに応じてオブジェクト生成時に決まるという基本的な制約がある．従って，これらのアプローチでは一般に以下のような問題がある．

（1）データベース設計時に各クラスのインスタンスの版管理方式をあらかじめ決定しておくことが必要である．また，これにより同一種類の対象でも版管理方式が異なると別のサブクラスを定義する必要が生じ，データベーススキーマが複雑化する．
（2）オブジェクトの生成時にそのクラスに応じて適用可能な版管理方式が決定されるため，オブジェクトの生成後にそのクラスで想定していない他の版管理方式を適用することができない．従って，これらのアプローチで要求条件 2)に対応しようとすると，そのいずれかのインスタンスに対して複数の版管理方式が適用され得るクラスには，あらかじめ適用可能性のある版管理方式で必要なすべての版管理用のデータ要素や手続きを定義しておく必要がある．これは，実際にはごく一部のインスタンスでしか利用

されないデータ要素や手続きを含んだ，冗長で複雑なクラス定義を生じる．

(3) 同様に，オブジェクトの生成時にそのクラスに応じて版管理されるかどうかが決定されるため，版管理を想定していないクラスのオブジェクトを後から版管理の対象とすることはできない．従って，これらのアプローチで要求条件 3)に対応しようとすると，そのいずれかのインスタンスが版管理され得るクラスには，あらかじめ版管理用のデータ要素や手続きを定義しておく必要がある．これらは，版管理されないインスタンスにとっては不要なものであり，(2)と同様冗長なクラス定義を生じさせることとなる．

本論文では，これらの問題に対するアプローチとしてセットメンバアノテーション (set-member annotation) の概念を提案し，それを利用することにより上記 1)～3) の要求条件を満たす版管理機構が実現可能であることを示す．アノテーションの概念は，アクセス指向プログラミングの分野で，オブジェクトのデータ要素に対して用いられている例がある[17]．本研究で提案するセットメンバアノテーションの概念はこれとは異なり，集合中の各メンバオブジェクトに付加的な情報を表すオブジェクト（アノテーションオブジェクト）を付与するものである．これにより，既存のオブジェクトをある版管理方式の下で管理する時点で，その版管理で必要とされる異なったデータ要素や手続きをアノテーションオブジェクトとして動的に付与することを可能とする．

以下，第2章では本研究のベースとなるデータモデルに関する基本概念を述べる．第3章では本研究で提案するセットメンバアノテーションの概念について説明する．第4章ではセットメンバアノテーションに基づく版管理機構とその実現方法について述べる．第5章ではオブジェクト指向データベース管理システム ONTOS における実装と，ソフトウェア開発データベースに対する適用例について述べる．第6章では本版管理機構の特長と問題点を整理する．最後の第7章では結論と今後の課題を述べる．

2. 基本概念

本研究では，ODMG オブジェクトモデル (ODMG-93 Rel. 1.0)[18] に基づいたオブジェクト指向データモデルをベースとして議論を進める．ODMG オブジェクトモデルは，主要 OODBMS ベンダにより構成される Object Database Management Group (ODMG) により提案されたもので，現在のところ，標準的オブジェクト指向データモデルの1つとして見なされつつある．本章では，以下の議論を展開する上で必要なデータモデルに関する基本概念について述べる．

```
class System : public Pobject{
private:
// データメンバ
char*              systemName; // システム名
Set<Program>*      program; // プログラムの集合への参照
                   …
public:
// メンバ関数
System(char* systemName); // コンストラクタ
Set<Program>*      Program(); // プログラムの集合を返す．
int                AddProgram(Program* theProgram); // プログラムを追加する．
                   …
};
```

図 1 クラス定義
Fig. 1 Class definition.

(1) オブジェクトとクラス

実世界の実体をオブジェクトとして表す．一般に，クラスはそのインスタンスであるオブジェクトの状態を記述するためのデータメンバと，振る舞いを記述したメンバ関数を規定する．オブジェクトの状態は，データメンバが持つ値により定義される．クラス定義の例を**図1**に示す．ここでの記法は，ODMG オブジェクトモデルの C++ ODL (Object Definition Language) にほぼ準じている．図1の System クラスは，データメンバとしてシステム名を表す systemName，そのシステムを構成するプログラムの集合を表す program 等を持つ．Set〈Program〉は，Program クラスのオブジェクトの集合を表す．(Set クラスについては以下でより詳しく述べる.) また，System クラスのインスタンスを生成し初期化するためのコンストラクタ System を持つ．またさらに，データメンバの読み書きのためのメンバ関数 Program，AddProgram 等を持つ．C++ 言語と同様，クラス定義は private と public の二つの部分からなり，一般にはどちらの部分にもデータメンバ，メンバ関数，コンストラクタを宣言可能である．private 部のデータメンバやメンバ関数は，外部から隠蔽されアクセスでき

ない．public 部は外部とのインタフェースとなる部分である．System クラスでは，データメンバ systemName, program 等を private 部で，コンストラクタ System, メンバ関数 Program, AddProgram 等を public 部で宣言している．クラス間には C++ 言語と同様なスーパークラス/サブクラスの関係があり，それらの間には継承関係がある．System クラスは以下で述べる Pobject クラスのサブクラスであり，Pobject クラスのデータメンバとメンバ関数を継承する．

（2） 基本クラス

ODMG オブジェクトモデルでは，種々の基本クラスがあらかじめ規定されている．ここでは，以下の議論において必要な2つの基本クラスについて述べる．

1) Pobject クラス

データベーススキーマ中のすべてのクラスのスーパークラスであり，オブジェクトを永続化可能にするための基本機能を提供するクラスである．このクラスのサブクラスのインスタンスであるオブジェクトは永続オブジェクトとなり得る．

2) Set クラス

オブジェクトの集合をそのインスタンスとするクラスである．このクラスのインスタンスである集合オブジェクトの生成時には，集合のメンバとなるオブジェクトのクラスを指定する．Set クラスは集合のメンバとしてのオブジェクトの登録，削除を行う以下のメンバ関数を提供する．

1. void insert_element (Pobject* theElement) ;
   引数 theElement で指定したオブジェクトを集合のメンバとして登録する．
2. void remove_element (Pobject* theElement) ;
   引数 theElement に指定したオブジェクトを集合のメンバから削除する．

このほか，基本的な集合操作を行うため，メンバの数を返すメンバ関数，指定したオブジェクトがメンバであるかどうかを判定するメンバ関数等を提供するが，以下の議論には関係しないため詳しい説明は省略する．

3. セットメンバアノテーション

本章では，セットメンバアノテーションの概念について説明する．また，第2章で述べた基本データモデルの枠組み中におけるセットメンバアノテーションの実現のために新たに導入する，ASet クラスについて述べる．

3.1 セットメンバアノテーションの概念

セットメンバアノテーションとは，あるオブジェクトが集合のメンバとなった時点において，オブジェクトに動的にあるデータメンバとメンバ関数を持つ1つのアノテーションオブジェクト (annotation object) を生成し付与するものである（**図2**）．このような機能を持つ集合をアノテーション付き集合 (set with annotation) と呼ぶ．メンバオブジェクトとアノテーションオブジェクトの対応付けはアノテーション付き集合によりなされる．あるオブジェクトが複数のアノテーション付き集合に属する場合には，それぞれ別のアノテーションオブジェクトが生成され付与される．あるメンバオブジェクトがアノテーション付き集合のメンバでなくなった場合には，自動的にアノテーションオブジェクトは消去される．アノテーション付き集合のメンバとすることにより，オブジェクトに動的にアノテーションオブジェクトの持つデータメンバとメンバ関数を付与することができる．

3.2 データモデルへの導入

基本データモデル中にセットメンバアノテーションの概念を導入するため，アノテーション付き集合の基本的性質を規定した ASet クラスを導入する．アノテーション付き集合は一般の集合としての性質も併せ持つため，ASet クラスは Set クラスのサブクラスとする．ASet クラスのインスタンスがアノテーション付き集合である．また，メンバオブジェクトに付与するアノテーションオブジェクトのクラスをアノテーションクラス (annotation class) と呼ぶ．

ASet クラスのオブジェクトは，メンバとしてあるオブジェクトが登録された際，アノテーションオブジェクトを生成しメンバオブジェクトとアノテーション

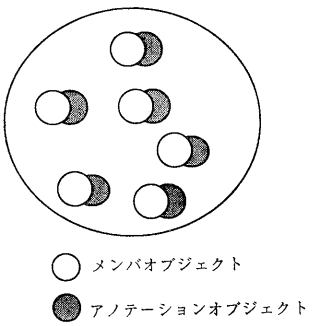


図2 セットメンバアノテーション
Fig. 2 Set-Member annotation.

オブジェクトの対応付けを行う．ASet クラスでは，Set クラスから継承するメンバオブジェクトの登録，削除を行うメンバ関数 insert_element，remove_element をアノテーションオブジェクトの処理を行うよう再定義する．また，メンバオブジェクト，アノテーションオブジェクト，ASet クラスのオブジェクトの対応関係をたどるための新たな関数を追加定義する．それらを以下に示す．

（1） コンストラクタ

ASet(Type* theElemType, Type* theAnnType)；ASet クラスのオブジェクトを生成する．引数 theElemType はメンバとなるオブジェクトのクラスを指定する．引数 theAnnType はアノテーションクラスを指定する．

（2） 再定義するメンバ関数

1. void insert_element (Pobject* theElement)；
   引数 theElement で指定したオブジェクトをメンバとして登録する．この時，対応するアノテーションオブジェクトが生成付与される．
2. void remove_element (Pobject* theElement)；
   引数 theElement で指定したオブジェクトをメンバから削除する．この時，対応するアノテーションオブジェクトが削除される．

（3） 追加するメンバ関数

1. Pobject* Annotation (Pobject* theMember)；
   引数 theMember で指定したメンバオブジェクトに対応するアノテーションオブジェクトを返す．
2. Pobject* Member (Pobject* theAnnotation)；
   引数 theAnnotation で指定したアノテーションオブジェクトと対応するメンバオブジェクトを返す．

（4） 追加する関数

1. Pobject* GetASetFromMember (Pobject* theMember)；
   引数 theMember で指定したオブジェクトがメンバとして含まれる ASet クラスのオブジェクトを返す．複数の場合は ASet クラスのオブジェクトをメンバとする Set クラスのオブジェクトを返す．
2. Pobject* GetASetFromAnnotation (Pobject* theAnnotation)；
   引数 theAnnotation で指定したアノテーションオブジェクトと対応するメンバオブジェクトを含む ASet クラスのオブジェクトを返す．

4. セットメンバアノテーションに基づく版管理機構

本章では，セットメンバアノテーションに基づく版管理の実現法と実現例を述べ，本版管理機構が第1章で述べた要求条件 1)～3) を満たすことを示す．

4.1 実 現 法

本稿で提案する版管理機構では，第1章で述べたような各版管理方式に応じて以下の2種類のクラスを定義することにより，版管理を実現する．

（1） 版集合クラス

版管理においては，ある1つの対象に対応して作成された一群の版の集まりを版集合（version set）として管理する必要がある．版集合クラスはこれら版集合を表現するオブジェクトをそのインスタンスとするクラスであり，ASet クラスのサブクラスとして定義する．これにより，版集合の要素となる各版オブジェクトに対して，版管理用の情報をアノテーションオブジェクトとして付与することが可能となる．版集合クラスには，版集合または版の操作用のメンバ関数を定義する．

（2） アノテーションクラス

版集合のメンバオブジェクトとして管理対象となる各版オブジェクトに付与する，アノテーションオブジェクトを規定するクラスである．本クラスには，版管理用のデータメンバやメンバ関数を定義する．アノテーションオブジェクトを永続化するため，Pobject クラスのサブクラスとして定義する．

4.2 実 現 例

セットメンバアノテーションに基づく版管理を 4.1 節の実現法に従って実現する．例として，第1章で述べた開発版オブジェクトとリリース版オブジェクトを対象とする2つの版管理方式を考える．

（1） 開発版オブジェクトの版管理

開発版オブジェクトの版管理においては，互いに導出関係にある版の集まりを開発版集合として管理し，その中の版を識別するために各版に版番号を付与する．また，各版の登録日時や版の登録日時の時間的前後関係を管理する．さらに，既存の版から新たな版を導出する手続きとテスト的に生成されたオブジェクトを初版として登録するための手続きを提供する．

このような版管理を実現するために，**図3** の版集合クラス DesignVerSet とアノテーションクラス DesignVersion を定義する．

```
class DesignVerSet : public ASet {
private:
 DesignVersion*     mostRecent;  // 最も最近登録された版への参照
public:
 DesignVerSet(Type* theMemType);  // コンストラクタ
 Pobject*           FirstVersion(Pobject* theVersion);  // 初版を登録する.
 Pobject*           Derive(Pobject* theBaseVersion);  // 版を導出する.
 int                VersionNo(Pobject* theVersion);  // 指定した版の版番号を返す.
 Pobject*           Version(int theVersionNo);  // 指定した版番号の版を返す.
 Time*              RegistTime(Pobject* theVersion);  // 指定した版の登録日時を返す.
 Pobject*           MostRecent(Time* theTime);  // 指定した時刻での最新版を返す.
 Pobject*           Parent(Pobject* theVersion);  // 親の版を返す.
 Set<Pobject>*      Child(Pobject* theVersion, Time* theTime);  // 子供の版を返す.
 Pobject*           Pred(Pobject* theVersion);  // 時間的に直前に登録された版を返す.
 Pobject*           Succ(Pobject* theVersion, Time* theTime);  // 指定した時刻で時間的に直後
                                                               //に登録された版を返す.
                    ...
};

class DesignVersion : public Pobject {
private:
 int                    versionNo;  // 版番号
 Time*                  registTime;  // 登録日時
 DesignVersion*         parent;  // 親の版への参照
 Set<DesignVersion>*    child;  // 子供の版への参照
 DesignVersion*         pred  // 時間的に直前に登録された版への参照
 DesignVersion*         succ;  // 時間的に直後に登録された版への参照
                    ...
public:
 DesignVersion();  // コンストラクタ
 int                    GetVersionNo();  // 版番号を返す.
 void                   SetVersionNo(int theVersionNo);  // 版番号をセット.
                    ...
};
```

図 3 DesignVerSet クラスと DesignVersion クラス
Fig. 3 DesignVerSet class and DesignVersion class.

DesignVerSet クラスのインスタンスが開発版集合である．本クラスは，開発版オブジェクトの版管理に特有の手続きをメンバ関数として提供する．主なメンバ関数について述べると，版オブジェクトの登録用メンバ関数として，指定したオブジェクトを初版として登録する FirstVersion と，既存の版から新たな版を導出する Derive を提供する．また参照用メンバ関数として，指定した版の版番号を返す VersionNo，版番号から該当する版を返す Version，登録日時を返す RegistTime，指定した時刻における最新の版を返す MostRecent を提供する．またさらに，導出関係に基づくある版の親の版を返す Parent，指定した時刻における子供の版を返す Child，ある版の時間的に直前に登録された版を返す Pred，直後の版を返す Succ 等を定義する．これらのメンバ関数は，ASet クラスから継承されるメンバ関数と，各版のアノテーションオブジェクトに適用可能な DesignVersion クラスのメンバ関数とを組み合わせて使用することにより実現される．

DesignVersion クラスは，開発版オブジェクトに固有の付加情報を記録するためのアノテーションクラスである．このクラスには，版番号，登録日時，導出関係，時間的に直前・直後の版の情報を保持するためのデータメンバと，これらデータメンバを読み書きするためのメンバ関数を定義する．

（2） リリース版オブジェクトの版管理

リリース版オブジェクトの版管理に関しては，外部ユーザがリリース版集合中で各リリース版オブジェク

```
class ReleaseVerSet : public ASet {
private:
 ReleaseVersion*  mostRecent;  // 最も最近リリースされた版への参照
public :
 ReleaseVerSet(Type* theMemType);  // コンストラクタ
 Pobject*         Register(Pobject* theVersion, Pobject* theBaseVersion);  // リリース版として登録する.
 int              ReleaseNo(Pobject* theVersion);  // 指定した版のリリース版番号を返す.
 Pobject*         Release(int theReleaseNo);  // 指定したリリース版番号の版を返す.
 Time*            ReleaseTime(Pobject* theVersion);  // 指定した版のリリース日時を返す.
 Designer*        ReleaseManager(Pobject* theVersion);  // リリース管理者を返す.
 Pobject*         MostRecent(Time* theTime);  // 指定した時刻での最新リリース版を返す.
 Pobject*         ParentRelease(Pobject* theVersion);  // 親のリリース版を返す.
 Set<Pobject>*    ChildRelease(Pobject* theVersion, Time* theTime);  // 子供のリリース版を返す.
                  ...
};

class ReleaseVersion : public Pobject {
private:
 int                      relaeseNo;  // リリース版番号
 Time*                    releaseTime;  // リリース日時
 Designer *               releaseManager;  // リリース管理者
 ReleaseVersion*          parentRelease;  // 親のリリース版への参照
 Set<ReleaseVersion>*     childRelease;  // 子供のリリース版への参照
                ...
public:
 ReleaseVersion();  // コンストラクタ
 int                      GetReleaseNo();  // リリース版番号を返す.
 void                     SetReleaseNo(int theReleaseNo);  // リリース版番号をセット.
                ...
};
```

図 4 ReleaseVerSet クラスと ReleaseVersion クラス
Fig. 4 ReleaseVerSet class and ReleaseVersion class.

トを識別するためのリリース版番号を付与する．また，リリース版間での機能的継承関係，リリース日時，リリース管理者といった情報を管理する．さらに，各種管理情報に対する参照用メンバ関数と，開発途中のオブジェクトが一定の水準に達したかどうかをチェックした後にそれをリリース版オブジェクトとして登録するためのメンバ関数を定義する．

このようなリリース版オブジェクトの版管理を実現するために，**図 4** の版集合クラス ReleaseVerSet とアノテーションクラス ReleaseVersion を定義する．

ReleaseVerSet クラスのインスタンスがリリース版集合である．本クラスは，リリース版オブジェクトの版管理に特有の手続きをメンバ関数として提供する．登録用メンバ関数としては，指定したオブジェクトがある水準に達しているかをチェックした後に，それをリリース版オブジェクトとしてリリース版集合に登録する Register を定義する．引数 theVersion は登録対象のオブジェクトを指定し，theBaseVersion はどのリリース版オブジェクトを機能的継承関係における親のリリース版とするのかを指定する．また，参照用メンバ関数としてやリリース版番号を返す ReleaseNo，リリース版番号から版を返す Release，版のリリース管理者を返す ReleaseManager，機能的継承関係における親のリリース版を返す ParentRelease，子供の版を返す ChildRelease 等を定義する．

ReleaseVersion クラスは，各リリース版オブジェクト固有の付加情報を記録するためのアノテーションクラスである．このクラスには，リリース版番号，リリース日時，リリース管理者，機能的継承関係等の情報を保持するためのデータメンバと，その読み書きのためのメンバ関数を定義する．

4.3 版 操 作

本節では，第 2 章で示した System クラスのオブジェクトを 4.2 節の版管理方式で管理した場合を考

```
System* theDSys1;
DesignVerSet* theDVS;
        ...
// Systemクラスのインスタンスを生成
theSys1 = new System();

// 開発版集合theDVSを生成
theDVS = new DesignVerSet(System);
        ...
// theSys1を初版として登録
theDVS->FirstVersion(theSys1);
```

図 5 初版の登録の手順
Fig. 5 Procedure for registration of the first version.

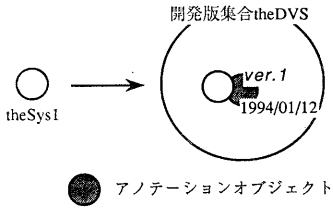


図 6 初版の登録
Fig. 6 Registration of the first version.

```
System* theSys2;
        ...
// 新たな版をtheSys1から導出
theSys2 = theDVS->Derive(theSys1);
```

図 7 版の導出の手順
Fig. 7 Procedure for derivation of a new version.

え，基本的版操作例を示す．これにより，セットメンバアノテーションに基づく版管理機構が，第1章で述べた各要求条件を満たすことを示す．

1) 初版の登録

System クラスのインスタンスを生成し，このオブジェクトをその生成後に開発版オブジェクトに対する版管理の対象とする（**図5**）．まず，System クラスのインスタンス theSys1 を生成する．次に，DesignVerSet クラスのインスタンスとして開発版集合 theDVS を生成する．最後に，メンバ関数 FirstVersion を theDVS に適用して theSys1 を初版として登録する．この時**図6**に示すように，theSys1 に対応する DesignVersion クラスのアノテーションオブジェクトが生成されると共に，DesignVersion クラスのメンバ関数が FirstVersion 内で内部的に呼び出され，

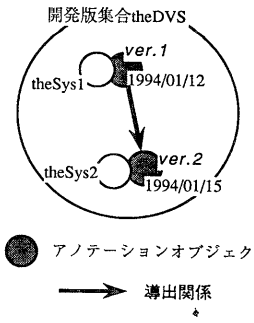


図 8 版の導出
Fig. 8 Derivation of a new version.

```
ReleaseVerSet* theRVS;

// リリース版集合theRVSを生成
theRVS = new ReleaseVerSet(System);
        ...
// theSys3をリリース
theRVS->Register(theSys3, NULL);
```

図 9 版のリリースの手順
Fig. 9 Procedure for release of a version.

アノテーションオブジェクトに初版の版番号，登録日時等が登録される．

2) 版の導出

初版 theSys1 から新たな版を導出するために，DesignVerSet クラスのメンバ関数 Derive を呼び出す（**図7**）．この時**図8**に示すように，Derive 内で新たな開発版オブジェクトとして System クラスのインスタンス theSys2 が生成され，theSys2 に対応するアノテーションオブジェクトが生成される．このアノテーションオブジェクトに導出された版の版番号，登録日時，導出関係等が登録される．この操作を繰り返すことにより，次々と開発版オブジェクトが導出されていく．

3) 版の初リリース

開発作業を進めた結果リリースする水準に達した開発版オブジェクト theSys3 をリリースする（**図9**）．まず，リリース版オブジェクトに対する版管理の対象にするため，ReleaseVerSet クラスのインスタンスとしてリリース版集合 theRVS を生成する．次に，リリース版オブジェクトとして開発版オブジェクト theSys3 を登録するために，ReleaseVerSet クラスのメンバ関数 Register を呼び出す．初リリースであり，機能を継承されるリリース版オブジェクトがまだ無いため，メンバ関数 Register の第2引数（機能を継承される

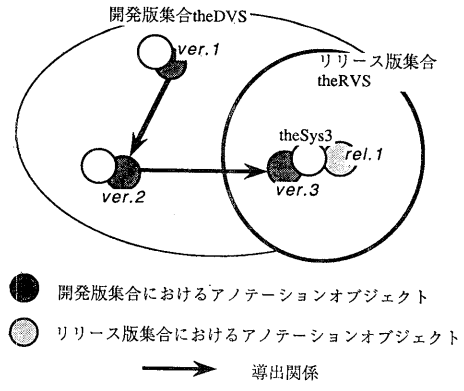


図 10 版の初リリース
Fig. 10 First release of a version.

版を指定する引数）には NULL を入れる．この時**図 10** に示すように，リリース版オブジェクトとしての情報を保持するため，ReleaseVersion クラスのアノテーションオブジェクトが theSys3 に対して生成付与される．このアノテーションオブジェクトにリリース版番号，リリース日時等の情報が登録される．

4) 版のリリース

操作3)でリリースした版にさらに改良を加えるために，2)と同様の操作により theSys3 から新たに開発版オブジェクトを導出し，開発作業を進めたものとする．その結果，リリースする水準に達した開発版オブジェクト theSys7 を，theSys3 の機能を継承するリリース版オブジェクトクとして登録する．このためには，3)の操作同様 ReleaseVerSet クラスのメンバ関数 Register を呼び出す．ただし，theSys7 が theSys3 の機能を継承することを登録するために，メンバ関数 Register の第2引数には theSys3 を指定する．この時**図 11** に示すように，リリース版オブジェクトとしての情報を保持するためのアノテーションオブジェクトが theSys7 に対して生成付与され，このアノテーションオブジェクトにリリース版番号，リリース日時，機能的継承関係等の情報が登録される．

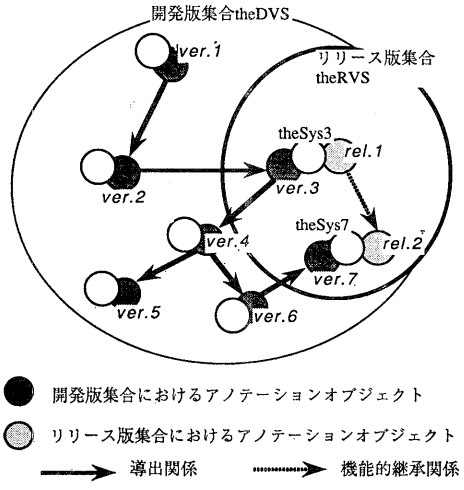


図 11 版のリリース
Fig. 11 Release of a version.

上記の版操作例では，4.2 節で述べた版集合クラスとアノテーションクラスを用いて，開発版オブジェクトに対する版管理とリリース版オブジェクトに対する版管理という異なる2つの版管理方式を提供している．また同様の方法により，さらに別の版管理方式をシステム内に導入することも可能である．また，上記操作 1) では，System クラスのオブジェクト theSys1 をその生成後に，開発版集合 theDVS のメンバとして登録し，開発版オブジェクトに対する版管理の対象とすることができた．さらに，上記操作 3) 4)では，開発版オブジェクトとして管理されていた System クラスのオブジェクトを，リリース版集合 theRVS のメンバとして登録し，開発版オブジェクトに対する版管理の対象のみならずリリース版オブジェクトに対する版管理の対象とすることができた．

このように，セットメンバアノテーションに基づく版管理機構では，第1章で述べた要求条件 1)〜3) を満たすことが可能である．

5. 実装と適用例

本研究では，セットメンバアノテーションの機能を持つ ASet クラスをオブジェクト指向データベース管理システム ONTOS[19] 上で実装し，ソフトウェア開発データベースを対象としたセットメンバアノテーションに基づく版管理機構の実装を行った．実装には，ONTOS DB 2.2 が提供するクラスライブラリを使用した．ONTOS では，第2章で述べた Pobject クラス，Set クラスに対応するクラスとして，Object クラス，Set クラスが存在する．また，ASet クラスの実装には ONTOS の提供する Dictionary クラスの機能を利用した．

5.1 ASet クラスの実装

ASet クラスは Set クラスのサブクラスとして実装する．ASet クラスを実装する上では，下記の3種類

のオブジェクト間の対応関係を管理する必要がある．

(1) メンバオブジェクトとアノテーションオブジェクト

(2) メンバオブジェクトと ASet オブジェクト

(3) アノテーションオブジェクトと ASet オブジェクト

上記対応関係を管理するために，本実装では ONTOS が提供する Dictionary クラスを使用した．Dictionary オブジェクトは，複数のオブジェクトを登録管理するための基本機能を提供する．Dictionary オブジェクトへのオブジェクトの登録時にはキーオブジェクトを与える．このキーオブジェクトを指定することにより，該当する登録オブジェクトを高速に検索することができる．キーオブジェクトは，どのクラスのオブジェクトでも良く，また同じキーオブジェクトに対応する登録オブジェクトが，同一の Dictionary オブジェクトに複数登録されていても良い．

本実装では，5種類の Dictionary オブジェクトを使用している．それら Dictionary オブジェクト，ASet オブジェクト，メンバオブジェクト，アノテーションオブジェクトの関係を**図12**に示す．図中で，各 Dictionary オブジェクトは2列の表として記述されている．表中左列はキーオブジェクト，右列は対応する登録オブジェクトを表している．以下に，各 Dictionary オブジェクトの役割を説明する．

1. ASetToMem オブジェクト

ASet オブジェクトから対応する MemToAnn オブジェクトを検索するのに用いる．

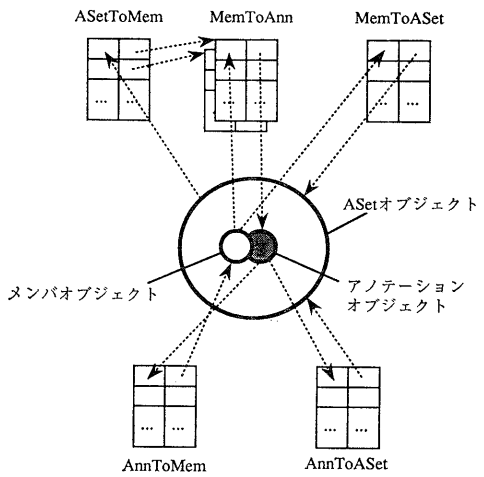


図 12 Dictionary オブジェクトの利用
Fig. 12 Use of Dictionary objects.

2. MemToAnn オブジェクト

各 ASet オブジェクトに対応して1つ存在し，その ASet オブジェクトのメンバオブジェクトから対応するアノテーションオブジェクトを検索するのに用いる．

3. MemToASet オブジェクト

メンバオブジェクトからそれをメンバとして含む ASet オブジェクトを検索するのに用いる．

4. AnnToMem オブジェクト

アノテーションオブジェクトからメンバオブジェクトを検索するのに用いる．

5. AnnToASet オブジェクト

アノテーションオブジェクトから ASet オブジェクトを検索するのに用いる．

一般に，あるオブジェクトをメンバとして含む ASet オブジェクトは複数存在し得るため，ASet オブジェクトごとにそのメンバオブジェクトと対応するアノテーションオブジェクトを管理する必要がある．従って，MemToAnn は各 ASet オブジェクトごとに1つ用意される．他の Dictionary オブジェクトは，データベース生成時に生成され，データベース中で常に1ずつ存在する．

ASet クラスのメンバ関数 insert_element (Pobject* theElement) が実行された時，これら Dictionary オブジェクトへの登録は以下の手順で行われる．

(1) メンバオブジェクトとして挿入される引数 theElement のオブジェクトに対応したアノテーションオブジェクトを生成する．

(2) メンバ関数の適用対象である ASet オブジェクト自身をキーオブジェクトとして，ASetToMem オブジェクトを用いて MemToAnn オブジェクトを検索する．

(3) 引数 theElement で指定されたオブジェクトをキーオブジェクトとして，(1)で生成したアノテーションオブジェクトを MemToAnn オブジェクトに登録する．

(4) MemToASet オブジェクトに，引数 theElement で指定されたオブジェクトをキーオブジェクトとして ASet オブジェクト自身を登録する．

(5) AnnToMem オブジェクトに，アノテーションオブジェクトをキーオブジェクトとして，引数 theElement で指定されたオブジェクトを登録する．

(6) AnnToASet オブジェクトに，アノテーションオブジェクトをキーオブジェクトとして ASet オブ

ジェクト自身を登録する．

ASet クラスに付随する他の関数も，内部的にこれらの Dictionary オブジェクトを利用して実装されている．

5.2 ソフトウェア開発データベースにおける版管理の実装

本研究では，ソフトウェア開発データベースを対象として本版管理機構を実装した．このデータベース中には，System クラス，Program クラス等のソフトウェア部品を管理するための各種クラスがある．これらのクラスのオブジェクトを版管理するために，第1章で述べた開発版オブジェクト，リリース版オブジェクト，外部版オブジェクトを対象とする3種類の版管理方式を実装している．実装には C++ 言語を使用した．実装システムにおける各クラスの定義は第2章で示したクラス記述とほぼ同様であり，また版管理のための版集合クラスやアノテーションクラスのクラス定義は，4.2 節で示したものとほぼ同様である．メンバ関数の実装例として DesignVerSet クラスのメンバ関数 VersionNo のコードを**図13**に示す．VersionNo は，ASet クラスから継承したメンバ関数 Annotation とアノテーションクラス DesignVersion のメンバ関数 GetVersionNo を内部的に呼び出すことにより実装されている．

```
int DesignVerSet::VersionNo(Object* theVersion)
{// このメンバ関数は、指定した版の版番号を返す。

  // 指定した版と対応するアノテーションオブジェクトを得る。
  Object *theAnn = (DesignVerion*)Annotation(theVersion);

  // アノテーションオブジェクトがあれば、版番号を返す。
  // 無ければ 0 を返す。
  if (!theAnn)
    return 0;
  else
    return theAnn->GetVersionNo() ;
}
```

図 13 メンバ関数 VersionNo のコード
Fig. 13 Code of the function VersionNo.

本システムでは，版操作を含む各種のデータベース操作のために ONTOS SQL をベースとしたインタプリタを実装している．**図14**にそのインタプリタを使用した版操作の実行例を示す．これは，本ソフトウェア開発データベース中の System クラスのオブジェクトを対象として，4.3節の版操作4)を実行し，その後検索を行っている例である．図中の(a)では，リリース版番号1のリリース版オブジェクトをさらに改良するために，新たな開発版オブジェクトを導出している．ここでは，対象となるリリース版集合および開発版集合の名前を “System-DVS” および “System-RVS” としている．where 節の条件は，これらの名前により開発版集合 (theDVS) およびリリース版集合

```
OSQL> select theDVS.VersionNo(theDVS.Derive(rv1))
      from   DesignVerSet theDVS, ReleaseVerSet theRVS, System rv1
      where  theDVS.Name() = "System-DVS" and                          ---(a)
             theRVS.Name() = "System-RVS" and
             theRVS.ReleaseNo(rv1) = 1;

---------------------
|4                  |
---------------------

              ...

OSQL> select theRVS.ReleaseNo(theRVS.Register(dv7, rv1))
      from   ReleaseVerSet theRVS, DesignVerSet theDVS, System dv7, System rv1
      where  theRVS.Name() = "System-RVS" and                          ---(b)
             theDVS.Name() = "System-DVS" and
             theDVS.VersionNo(dv7) = 7  and
             theRVS.ReleaseNo(rv1) = 1;

---------------------
|2                  |
---------------------

OSQL> select theDVS.VersionNo(theDVS.Parent(dv4))
      from   DesignVerSet theDVS, System dv4
      where  theDVS.Name() = "System-DVS" and                          ---(c)
             theDVS.VersionNo(dv4) = 4;

---------------------
|3                  |
---------------------
```

図 14 版操作の実行例
Fig. 14 Session example of version manipulation.

(theRVS) を，そしてリリース版番号1により導出元となるリリース版オブジェクト (rv 1) を指定している．select 節で DesignVerSet クラスのメンバ関数 Derive を呼び出しこのメンバ関数の実行時に生じる副作用を利用することにより，新たな開発版オブジェクトを導出している．(a)の処理結果としては，新たに導出した開発版オブジェクトの版番号を出力している．図中の(b)では，(a)で導出した版を元にさらに作業を進めて改良を加えた結果，一定の水準に達した版番号7の開発版オブジェクト (dv 7) をリリース版番号1のリリース版オブジェクト (rv 1) の機能を継承するリリース版オブジェクトとして，リリース版集合 (theRVS) に登録している．結果として出力されているのは，登録したオブジェクトのリリース版番号である．これらの実行後，開発版集合とリリース版集合の状態は図 11 のようになっている．(c)では，開発版集合において版番号4の開発版オブジェクト (dv 4) の親の開発版オブジェクトを検索し，その版番号を出力している．結果は，版番号3であることを示している．

6. 特長のまとめと問題点

4.3 節で示したように，本研究において提案したセットメンバアノテーションに基づく版管理機構は，第1章の要求条件 1)〜3)を満足するものである．

要求条件 1) に関しては，ASet クラスのサブクラスのオブジェクトとして版集合を実現し，各版オブジェクトにアノテーションオブジェクトを生成することにより，管理対象に応じた版管理方式に必要なデータ要素や手続きを付与することを可能としている．要求条件 2) に関しては，開発版オブジェクトを同時にリリース版集合に登録することが可能であり，この例のように複数の版管理方式の下に同一オブジェクトを管理可能である．要求条件 3) に関しては，既存のオブジェクトを後から版集合中に登録し版管理の対象とすることが可能である．

また本研究での実装を通じ，セットメンバアノテーションに基づく版管理機構が，既存のオブジェクト指向データベース管理システムの枠組みの中で実現可能であることを示した．

一方，上記の検討を通じて，セットメンバアノテーションに基づく版管理機構は，以下のような問題点を持つことも明らかになった．

(1) 版集合の特定に伴うデータ操作記述の複雑化

オブジェクトごとに異なる版管理方式の対象とすることが許されるため，あるオブジェクトに版操作を行う場合には，実行時に版集合の特定がまず必要となる．これは，セットメンバアノテーションに基づく版管理機構の持つ自由度の結果であり，ある程度やむを得ないこととは言え，データ操作記述の複雑化をもたらす傾向がある．

(2) 版の削除

本版管理機構では，ASet オブジェクトの持つ基本機能により，メンバオブジェクトを版集合から削除すると対応するアノテーションオブジェクトも自動的に削除される．そのため，版集合から削除されたメンバオブジェクトに関する履歴情報も対応するアノテーションオブジェクトの削除と同時に消えてしまう．従って，例えば版の実体は削除したいが，その導出関係等のアノテーションオブジェクトの持つ履歴情報は残したいというような要求に対応することが難しい．

7. おわりに

本稿では，セットメンバアノテーションに基づく版管理機構を提案した．さらに，本版管理機構の実現について述べ，ソフトウェア開発データベースを具体的適用対象としたその実装例を示した．これらを通して，管理対象に応じたきめ細かな版管理方式が必要とされる設計環境における本版管理機構の有効性と問題点を明らかにした．また，標準的なオブジェクト指向データモデルの枠組みを大きく変更することなく，本版管理機構は導入可能であることを具体的実装を通して示した．

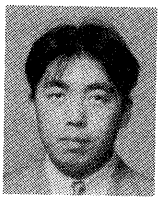
今後の研究課題としては，第6章で述べた問題点の解決法あるいは緩和法が挙げられる．また，ソフトウェア開発以外の他のエンジニアリング応用を含めた，大量の設計データを対象とした本版管理機構の適用評価および性能評価が必要である．さらに，種々の応用要求に対応した版集合クラスとアノテーションクラスの定義を容易とするため，版管理用の標準的版集合クラスやアノテーションクラスの整備やそのライブラリ化について検討する必要がある．

謝辞 本稿に対して貴重なコメントをくださった査読者の方々に感謝いたします．なお，本研究の一部は筑波大学学内プロジェクトの研究助成による．

参考文献

1) Atkinson, M. et al.: The Object-Oriented Database System Manifesto, *Deductive and Object-Oriented Databases*, Kim, W., Nicolas, J. M. and Nishio, S. (eds.), pp. 223-240, North-Holland (1990).
2) Katz, R. H.: *Information Management for Engineering Design*, Springer-Verlag (1985).
3) Snodgrass, R.: The Temporal Query Language TQuel, *ACM Trans. Database Syst.*, Vol. 12, No. 2, pp. 247-298 (1987).
4) Katz, R. H.: Toward a Unified Framework for Version Modeling in Engineering Databases, *ACM Comput. Surv.*, Vol. 22, No. 4, pp. 375-408 (1990).
5) Ahmed, R. and Navathe, S. B.: Version Management of Composite Objects in CAD Databases, *Proc. ACM SIGMOD*, Denver, pp. 218-227 (1991).
6) Chou, H. T. and Kim, W.: Versions and Change Notification in an Object-Oriented Database System, *Proc. 25th ACM/IEEE Design Automation Conf.*, pp. 275-281 (1988).
7) Chou, H. T. and Kim, W.: A Unifying Framework for Version Control in a CAD Environment, *Proc. 12th Int. Conf. on VLDB*, Kyoto, Japan, pp. 336-344 (1986).
8) Beech, D. and Mahbod, B.: Generalized Version Control in an Object-Oriented Database, *Proc. 4th Int. Conf. on Data Eng.*, Los Angeles, pp. 14-22 (1988).
9) Agrawal, R., Buroff, S., Gehani, N. and Shasha, D.: Object Versioning in Ode, *Proc. 7th Int. Conf. on Data Eng.*, Kobe, Japan, pp. 446-455 (1991).
10) 北川博之, 田中 肇, 大保信夫, 鈴木 功: 履歴データ型を用いたバージョン管理データモデルの提案, 情報処理学会論文誌, Vol. 34, No. 5, pp. 1031-1044 (1993).
11) Klahold, P., Schlageter, G. and Wilkes, W.: A General Model for Version Management in Databases, *Proc. 12th VLDB*, Kyoto, Japan, pp. 319-327 (1986).
12) Dittrich, K. R. and Lorie, R. A.: Version Support for Engineering Database Systems, *IEEE Trans. Softw. Eng.*, Vol. 14, No. 4, pp. 429-437 (1988).
13) Bjoernerstedt, A. and Hulten, C.: Version Control in an Object-Oriented Architecture, *Object-Oriented Concepes, Databases, and Applications*, Kim, W. and Lochovsky, F. H. (eds.), pp. 451-485, Addison-Wesley (1989).
14) Sciore, E.: Multidimensional Versioning for Object-Oriented Databases, *Proc. 2nd Int. Conf. DOOD*, Munich, Germany, pp. 355-370 (1991).
15) Sciore, E.: Using Annotation to Support Multiple Kinds of Versioning in an Objct-Oriented Database System, *ACM Trans. Database Syst.*, Vol. 16, No. 3, pp. 417-438 (1991).
16) Wuu, G. T. J. and Dayal, U.: A Uniform Model for Temporal Object-Oriented Databases, *Pcoc. 8th Int. Conf. on Data Eng.*, Tempe, pp. 584-593 (1992).
17) Stefik, M., Bobrow, D. and Kahn, K.: Integrating Access-Oriented Programming into a Multiparadigm Environment, *IEEE Softw.*, Vol. 3, No. 1, pp. 10-18 (1986).
18) Cattell, R. G. G. (ed.): The Object Database Standard: ODMG-93, Morgan Kaufmann (1994).
19) ONTOS DB 2.2 Developer's Guide, Ontos, Inc. (1992).

(平成5年4月26日受付)
(平成6年6月20日採録)



宮脇 忠光 (正会員)

1968年生. 1991年筑波大学第三学群情報学類卒業. 1993年同大学院修士課程理工学研究科修了. 同年富士ゼロックス(株)入社. 現在, 同社ドキュメントシステム開発センター・サービスコア開発部勤務.

北川 博之 (正会員)

1955年生. 1978年東京大学理学部物理学科卒業. 1980年同大学院理学系研究科修士課程修了. 日本電気(株)勤務を経て, 1988年筑波大学電子・情報工学系講師. 現在, 同学系助教授. 理学博士 (東京大学). データベースシステム構成法, 時間データ管理, エンジニアリングデータ管理, ソフトウェア開発支援システムなどに興味を持つ. 著書「The Unnormalized Relational Data Model」(共著, Springer-Verlag). 電子情報通信学会, 日本ソフトウェア科学会, ACM, IEEE-CS 各会員.