データベースシステム 99−33
（1994. 7. 22）

ASN.1データベースプログラミング言語

春本 要† 塚本昌彦‡ 西尾章治郎†

†大阪大学工学部情報システム工学科 ‡シャープ（株）技術本部情報技術研究所

抽象構文記法1 (ASN.1) に基づくデータベースシステムにおけるデータベースプログラミング言語について論ずる．ASN.1 を利用する通信アプリケーションは，一般にCやC++などの汎用プログラミング言語を用いて開発されている．この手法をデータベースを利用する通信アプリケーションの開発に適用した場合，その開発効率などにおいて問題が生じる．本稿ではこれらの問題点を解決するために，型推論および型検査の利用によって複雑なデータ構造の扱いを容易にするデータベースプログラミング言語 ASN.1/PL を提案する．

A Database Programming Language for ASN.1

Kaname HARUMOTO† Masahiko TSUKAMOTO‡ Shojiro NISHIO†

†Department of Information Systems Engineering, Faculty of Engineering, Osaka University
‡Information Technology Research Laboratories, SHARP Corporation

This paper describes a database programming language for the systems based on Abstract Syntax Notation One (ASN.1). Applications described by ASN.1 for communication systems have been usually developed by using general purpose programming languages such as C and C++. If this method is applied to develop applications whose data are stored in database systems, we have several deficiencies in the software development such as an inefficiency problem. For resolving these deficiencies, we propose in this paper a database programming language ASN.1/PL, in which type checking and inference are employed for application programmers to handle easily complicated data structures.

1 はじめに

抽象構文記法 1 (Abstract Syntax Notation One, ASN.1) は，国際標準化機構 (International Organization for Standardization, ISO) によって定められた国際標準 [1, 2] である．ASN.1 は，OSI 基本参照モデルにおけるアプリケーション層間で交換されるデータの構造の抽象的な表現形式（抽象構文）を定義するためのデータ構造記述言語，および，抽象構文で表現されたデータをプレゼンテーション層間で交換されるデータの表現形式（転送構文）にコード化するための基本符号化規則からなる[1]．これらは，ネットワークで結合された異種計算機システム間でのデータの交換を曖昧さなく行なうことを目的に規定されたものである．

ASN.1 は，OSI アプリケーション層プロトコルだけでなく，TCP/IP におけるネットワーク管理プロトコルである SNMP (Simple Network Management Protocol) で利用されている．さらに近年，アプリケーションが交換するデータ構造の記述に ASN.1 を用いることがさまざまな分野で検討されている．例えば，マルチメディア通信の分野では，MHEG (Multimedia and Hypermedia Coding Expert Group)[5] や ODA の拡張である Hyper ODA などが扱うマルチメディア情報のデータ構造が ASN.1 を用いて定義されようとしている．また，複雑な遺伝子構造を扱うゲノムデータベースでの利用も考えられている [15]．

ASN.1 の利用分野が拡大する一方，このような通信アプリケーションではしばしば何らかのデータ蓄積を伴うことがあり，現在までに OSI 管理情報ベース (MIB)，OSI ディレクトリのためのディレクトリ情報ベース (DIB)，ODA 文書データベースなどの構築に関する研究が報告されている [8, 9, 10, 11, 13, 14]．これらの構築例では，関係データベース管理システムやオブジェクト指向データベース管理システムといった汎用データベース管理システムを用いて実現したり，あるいは，特定のアプリケーション専用に最適化したデータベースシステムを構築したりしている．また，アプリケーション開発のために C や C++などの汎用プログラミング言語を用いている．

筆者らはこのようなアプリケーション開発方法ではいくつかの問題が生じることを指摘し，それを解決するために ASN.1 データベースシステム ASN.1/DB，および，データベースプログラミング言語 ASN.1/PL を提案した [6]．本稿では，ASN.1/PL の言語設計について詳しく論じる．

以下，2 章では ASN.1/PL の設計において考慮すべき問題点を明らかにする．3 章では，設計した ASN.1/PL の言語仕様について述べる．4 章では，ASN.1/PL の処理系の実装について述べる．最後に 5 章で本稿のまとめと今後の課題について述べる．

2 ASN.1/PL の設計

2.1 ASN.1 データの表現形式

ASN.1 を使用するアプリケーションの実装には，一般に，C や C++などの汎用プログラミング言語が利用されている．そのために，転送構文で受信した ASN.1 データは，抽象構文に対応するその言語のデータ構造に復号化して処理を行なうことになる．このようなアプリケーション開発を支援するために，ASN.1 コンパイラと呼ばれるツールが利用されている．ASN.1 コンパイラは，ASN.1 のデータ構造から対応する汎用プログラミング言語のデータ構造への変換や，符号化関数，復号化関数の生成を行なう．

この方法でアプリケーション開発を行なった場合，アプリケーション開発者は ASN.1 コンパイラによって変換された汎用プログラミング言語のデータ構造を随時参照しながらプログラミングを行なわなければならない．変換されたデータ構造は一般に複雑であるため，この問題は開発効率を下げる原因になる．

また，OSI ディレクトリにおけるディレクトリ情報ベース (DIB) や MHS でメッセージを格納するメッセージストアのように ASN.1 のデータ構造をもつデータをデータベースに蓄積する必要がある場合，アプリケーション開発はさらに複雑に

[1] 単に ASN.1 といった場合，データ構造記述言語だけを指す場合が多い．

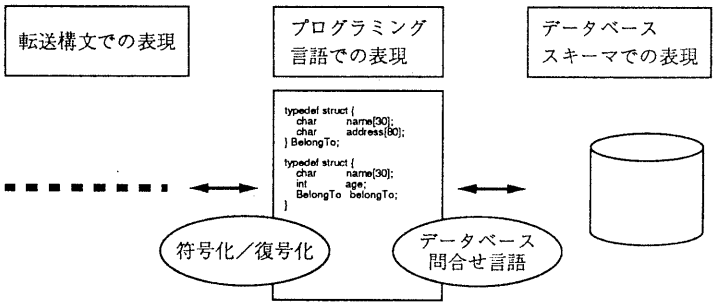

図 1: ASN.1 データの三種類の表現形式

なる．例えば汎用データベース管理システムを利用した場合，一般に ASN.1 のデータ構造を何らかの形で表現するスキーマに変換しなければならない．したがって，プログラム内では，転送構文での表現，プログラミング言語内での表現，データベース内での表現という三種類のデータの表現形式が混在することになり，アプリケーション開発者はこれらの表現形式の変換を管理しなければならない（図 1）[2]．

さらに，データの表現形式がプログラミング言語とデータベースとで異なるという，プログラミング言語とデータベースとのインタフェースに関するインピーダンスミスマッチが問題となる．一般にデータベースへのアクセスはデータベースシステムが提供する問合せ言語を通じてなされるため，アプリケーション開発者はプログラミング言語だけでなくこの問合せ言語も熟知していなければならない．また，通常，このインタフェースは問合せ文字列を引数とするライブラリ関数という形で提供されるが，これらの関数は汎用的に作られているために返り値の型はコンパイル時に決定できず，問合せの構文誤りも発見できない．もし返り値の型がプログラムで意図した型と異なっていたり，問合せに構文誤りがあったりすると，コンパイルされたアプリケーションの実行時にエラーが発生することになる．これもまた，アプリケーション開発の効率を下げる原因になり，さらに，アプリケーションの信頼性も低下する．

以上に挙げたいくつかの問題点は，すべて ASN.1 データのプログラムレベルでの表現形式が統一されていないことに起因する．したがって，これらを解決するためには，ASN.1 データを統一的に扱えるような枠組が必要であると考えられる．ASN.1/PL ではこの点を重視し，プログラミングにおいてはデータの表現形式の変換を意識せずに抽象構文のレベルで ASN.1 データの操作の記述を可能にすることを第一に考えて設計を行なった．

2.2 多相的プログラミング

プログラミング言語において，多相性をもつ関数が有効であることは様々な場面で実証されている．例えばオブジェクト指向プログラミング言語においては，同名メソッドがそれを受けたオブジェクトのクラスによって異なる動作をとるというメソッドの多相性が利用されている．データベースプログラミングにおいては，対象となるデータの構造が複雑になるほど多相性の概念が特に有効かつ重要であり，また，本質的であることが示されている [3]．ODA など多くのアプリケーションにおいて，ASN.1 で記述されたデータ構造は非常に複雑になっていることから，ASN.1/PL に多相性の概念を採り入れるのは有用であると考える．

この考えに基づく言語として，多相性をもつ関

[2] オブジェクト指向データベース管理システムを利用する場合は，クラス定義がそのままスキーマ定義となるために三種類の表現形式とはならないが，それでも二種類の表現形式が存在する．

数型言語である ML をデータベース操作に適するように拡張した汎用データベースプログラミング言語 Machiavelli が提案されている [4, 12]. また, Machiavelli は ML からの拡張において型推論アルゴリズムもデータベース操作に適するように拡張しており, これにより前述のインピーダンスミスマッチの問題も解決している. ASN.1/PL の設計では, この言語の考え方を採り入れている.

3 ASN.1/PL の言語仕様

前章では, ASN.1 データベースシステムにおけるデータベースプログラミング言語 ASN.1/PL の概略について述べた. この章では, ASN.1/PL の言語仕様についてより詳しく述べる.

3.1 構文要素

関数型言語 ASN.1/PL の各構文要素について説明する.

型: ASN.1/PL では, ASN.1 が提供している組み込み型, および, ASN.1 モジュール内で定義されたユーザ定義型をデータ型として扱う. また, プログラム内で新たな型を生成することもできる. 型の指定は ASN.1 の型名を記述する.

値: ASN.1/PL での値はすべて ASN.1 の型をもつ. プログラム内での値の記述は, ASN.1 の値記法で記述する.

変数: 変数は, ある値を参照するために使用されるものであり, 参照している値の型をもつ. 変数の型はその宣言時に決定され, それと異なる型の値を代入することはできない.

変数宣言および変数が参照している値の更新は, それぞれ次のように記述する.

宣言: val *var_name* [: *type*] = *expression*
更新: *var_name* := *expression*

変数の型 *type* は, 右辺の式から推論できるため省略することができ, 指定された場合には型の整合性が検査される. 変数は, 一般の汎用言語と異なり, すべてある値への参照を表す. したがって, 変数が永続的な値を参照している場合には, その変数の値の更新はデータベース内に格納されている値を更新することになる.

また, 特別に ASN.1 の型名を変数として記述することができる. これはデータベースに格納されているその型の値の集合 (SET OF 型をとる) を参照するために用いる.

関数: 関数は, ASN.1/PL の主体となるものであり, アプリケーション開発者は関数の組み合わせとしてプログラムを記述する.

関数宣言および関数適用は, それぞれ次のように記述する.

宣言: fun *fun_name* ([*arg1* [: *type1*], ...]) [: *type*]
= *expression*
適用: *fun_name* ([*arg1*, ...])

引数の型および返り値の型は, 右辺の式から推論可能な場合は省略することができる. これによって, 多相性をもつ関数の宣言が可能になる. 例えば, 関数 nameof が

```
fun nameof(x) = x.name
```

と宣言されている場合, この関数は name を要素としてもつ任意の SEQUENCE 型, SET 型, CHOICE 型に対して適用可能である.

式: 値, 変数, 関数適用は式である. また, 様々な計算を以下に挙げる式で表現する. すべての式は, 何らかの型をもつ.

(a) **演算子** すべての型に対して, その値の等価性を判定する演算子を使用できる. また, ASN.1 の単純型に対しては基本的な演算子を使用できる. SEQUENCE OF 型と SET OF 型をもつ式に対しては, ある値がその集合値に含まれるかどうかを判定する in 演算子を使用できる. CHOICE 型の値に対しては, その選択を判定する isa 演算子を使用できる.

(b) **航行式** ASN.1 の型は複雑な入れ子構造をもつことがある. この構造をたどるためには, ドット記法を用いて次のように記述する.

expression.field_name

これは，*expression* で表される SEQUENCE 型，SET 型，CHOICE 型の値から，その構造内の *field_name* で表される要素を取り出す．

(c) **条件式** 条件式として，次の構文で示される if 式と case 式が使用できる．

- if *expression1* then *expression2*
  else *expression3*
- case *expression1* of
  [*rel_op*] *expression2* => *expression3*,
  ⋮
  [else => *expression4*]
  end

case 式における *rel_op* は in や isa を含む比較演算子であり，省略された場合には "=" として扱う．

(d) **反復式** ASN.1/PL は次の二種類の反復式を提供する．これらの反復式は特別な型 unit をもつ．

- while *expression1* do *expression2*
- foreach (*l_var*, *expression1*) *expression2*

while 式は，BOOLEAN 型をもつ *expression1* が TRUE である間，*expression2* の計算を行なう．foreach 式は，SEQUENCE OF 型あるいは SET OF 型をもつ *expression1* の中から要素を一つずつ取り出して *l_var* がそれを参照し，*expression2* の評価を行なう．

(e) **構造化問合せ式** SQL の宣言的な問合せ記述をとり入れ，ASN.1/PL では構造化問合せ式を次のように記述する．

select *expression* [, ...]
from *var_name* <- *expression* [, ...]
where *expression*

from 節の *expression* は SEQUENCE OF 型あるいは SET OF 型をもつ式であり，問合せの対象となる集合を指定する．where 節の *expression* は BOOLEAN 型をもつ式であり，この式の値が TRUE となる要素が選択される．select 節の *expression* では，from 節に記述された変数名を含む任意の式を記述することができる．

3.2 システム関数

ASN.1/PL は，ユーザとのデータの入出力を行なう関数や，ネットワークを介したデータの送受信を行なう関数などの基本的な関数を，システム関数として提供する．

4 ASN.1/PL 処理系

この章では，ASN.1/PL の処理系について述べる．ASN.1/PL で記述されたプログラムの処理は二段階からなる．まず，プログラム全体に対して型の整合性検査を行なう．これは，データを蓄積するために実際に用いるデータベースシステムに依存しない部分である．次に，実行可能な形式にプログラムの変換を行なう．この変換処理は，使用するデータベースシステムに依存する．そこで，プログラムの変換処理については，UNIX 上の C++ ベースのオブジェクト指向データベース管理システム (OODBMS) である ObjectStore を用いて実装した ASN.1 データベースシステム [7] (図 2) での ASN.1/PL 処理系について述べる．

4.1 型検査機構

データベースプログラミングにおけるコンパイル時の静的な型検査の重要性は 2.1節で述べた．ASN.1/PL ではデータベースシステムの扱う型とプログラミング言語が扱う型がともに ASN.1 の型であるため，十分な静的型検査を行なうことが可能である．

一方，ASN.1/PL の変数宣言や関数宣言においては，型の指定を省略することができるため，式から型を決定する機構が必要である．この機構のために，言語 Machiavelli の型推論アルゴリズム [4] を利用できることを以下に示す．

4.1.1 Machiavelli の型と式

Machiavelli で扱うことができる型は，データベースシステムが扱う基本型に加え，様々な型を

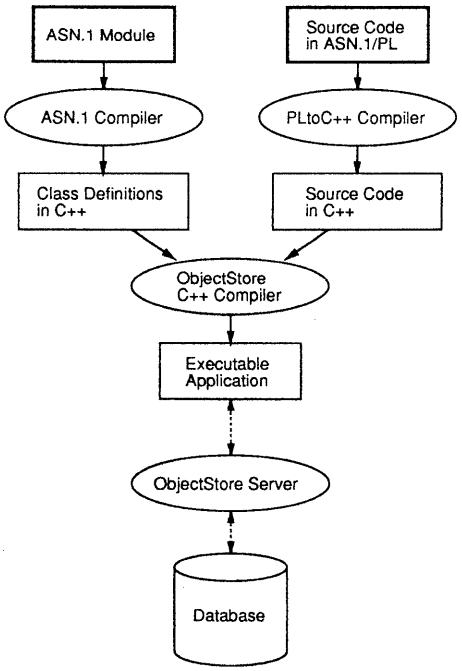

図 2: OODBMS を用いた ASN.1 データベースシステムの実装

扱うことができる．以下に，その一部を挙げる．

- $[\, l_1 : \tau_1, ..., l_n : \tau_n \,]$: フィールド $l_1, ..., l_n$（それぞれの型は$\tau_1, ..., \tau_n$）からなるレコード型．
- $< l_1 : \tau_1, ..., l_n : \tau_n >$: フィールド $l_1, ..., l_n$（それぞれの型は$\tau_1, ..., \tau_n$）からなる選択型．
- $\{\tau\}$: 型τの集合型

これらの型をもつ値に対して，以下に挙げるような式を扱うことができる．

- $[\, l_1 = e_1, ..., l_n = e_n \,]$: フィールド $l_1, ..., l_n$からなるレコード．
- $< l = e >$: 選択型からの一つの選択．
- $\{e_1, e_2, ...\}$: ある型の値の集合．
- $e.l$: レコード e からのフィールド lの選択．

Machiavelli ではこのほかにも様々な式を扱うことができ，それらに対して型付けを行なう型推論アルゴリズムが定義されている．

4.1.2 Machiavelli の型推論の利用可能性

ASN.1/PL で扱う ASN.1 の型は，Machiavelli で扱うことができる型に次のように対応づけることができる．

- 単純型: 基本型
- SEQUENCE 型，SET 型: レコード型
- SEQUENCE OF 型，SET OF 型: 集合型
- CHOICE 型: 選択型

また，変数，関数，航行式，構造化問合せ式などを組み合わせた ASN.1/PL の任意の式は，Machiavelli で扱うことができる式の表現に対応づけることができる．したがって，Machiavelli がもつ型推論アルゴリズムを ASN.1/PL の型推論機構で利用することが可能である．

4.2 符号化と復号化

ASN.1/PL でのプログラミングにおいては，データの表現形式の変換を意識せずにすむように設計を行なった．しかし，符号化された ASN.1 データはそのまま計算機内部で処理するには都合が悪いため，処理の内部においては符号化および復号化を何らかの形で行なわなければならない．

例として，符号化された ASN.1 データを受信した場合を考える．通常，受信した ASN.1 データは即時完全に復号化され，プログラミング言語で処理できる構造に変換される．しかし，受信した ASN.1 データのうち全部をプログラムで処理しなければならない場合以外には，完全な復号化を行なわない方が時間の面で効率がよい．また，ODA 文書などで受信した文書データの一部を変更して送信する場合などでは，部分的な復号化を行なうことによって，値が変更された部分のみ符号化し直せばよいことになり，効率がよい．

部分的に復号化できるようにすることは，アプリケーションを組み合わせて利用する場合にも有効である．例えば，OSI ディレクトリ，OSI 管理など ROSE を使用するアプリケーションや，内

部がいくつかのサブモジュールに分割されている FAIS (Factory Automation Interconnection System; MiniMAP 原案) などにおいては，エラー処理などの都合上，データの送受信の際に個々のプロトコルマシンは関連する部分のみを符号化，復号化するように実装することが望ましい．ODA や EDI (Electric Data Interchange) の H手順など MHS 上でデータ交換を行なう場合も，符号化および復号化は独立に行なう必要がある．

ASN.1/PL はシステム関数としてデータの送受信を行なう関数を提供するが，符号化や復号化の処理は，ASN.1/PL 処理系が自動的に最適化を行なう．例えば，データ受信関数の呼び出しに対しては，受信したデータ構造のトップレベルのみを復号化して返すコードを出力する．入れ子構造をもつデータの内部に関しては，航行式等によってそれが最初にアクセスされた時点で復号化を行なうようなコードを出力する．また，データ送信関数の呼び出しに対しては，符号化を自動的に行なってからデータを送信するようなコードを出力する．

4.3 プログラムの変換

OODBMS では，それがサポートするプログラミング言語で記述されたプログラムしか扱うことができない．したがって図 2の実装では，ASN.1/PL 処理系は ASN.1/PL で記述されたプログラムを C++のプログラムに変換することになる．

ASN.1/PL から C++への変換は，符号化や復号化の処理も考慮して次のように行なう．

- 変数宣言: 変数はすべて参照型であるので，ポインタ型の変数宣言に変換する．
- 関数宣言: 基本的には一つの関数定義に変換する．C++では多相的な関数を一つの関数定義で表現できないため，多相的な関数は引数と返り値の型の可能な組合せについて一つずつ関数定義に変換する．
- 関数適用: 関数呼び出しに変換する．
- 演算子: C++が提供する演算子，あるいは，ASN.1 コンパイラがクラスメソッドとして出力する演算子用メソッドの呼び出しに変換する．
- 航行式: $e.l$は，式 e のフィールド lが復号化されているかどうかをフラグによって判断し，復号化されていればその値を返し，復号化されていなければ復号化を行なってからその値を返すようなコードに変換する．
- 条件式: if 式，case 式ともに C++の if 文に変換する．
- 反復式: for 文に変換する．
- 構造化問合せ式: from 節で指定された要素を for 文で一つずつ取り出しながら where 節で指定された条件を評価し，それが真である場合には select 節で指定された式を評価して結果の集合に加えるようなコードに変換する．

5 おわりに

本稿では，ASN.1 データベースシステムのためのデータベースプログラミング言語 ASN.1/PL の設計，および，その処理系の実装について述べた．汎用プログラミング言語を用いた従来の開発手法の問題点を，ASN.1/PL は次のように解決した．

- プログラムレベルでのデータの表現形式を統一したことにより，プログラミング言語とデータベースシステムの間のインピーダンスミスマッチを解消した．これによって，プログラムのコンパイル時に十分な静的型検査を行なうことが可能になり，開発されたアプリケーションの信頼性を向上させることができる．
- 型多相性の概念を導入したことにより，ASN.1 で定義された複雑なデータ構造を扱うプログラムの記述が容易になった．
- 符号化や復号化の操作をプログラム中に明示的に記述せずにシステムが自動的に最適化することによって，これらの処理を意識しないアプリケーション開発が可能になった．

今後は，いくつかの点で ASN.1/PL を拡張する予定である．まず，通信アプリケーションでは，そ

の異機種分散環境での継続的な動作を保証するために，あらゆる種類のエラーに対して適切な動作を行なえるようプログラムを記述できなければならないが，そのために，ASN.1/PL の例外処理の記述能力を強化する必要がある．また，ASN.1/PL を用いて実際にアプリケーションを開発し，その有効性について検証するとともに，必要な機能を抽出し追加することを考えている．

参考文献

[1] ISO 8824, Information Processing Systems — Open Systems Interconnection — Specification of Abstract Syntax Notation One (ASN.1), 1987.

[2] ISO 8825, Information Processing Systems — Open Systems Interconnection — Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1), 1987.

[3] Atkinson, M.P. and Buneman, O.P., "Types and Persistence in Database Programming Languages", *ACM Computing Surveys*, Vol. 19, No. 2, pp. 105–190, 1987.

[4] Buneman, O.P. and Ohori, A., "Polymorphism and Type Inference in Database Programming", *ACM Transactions on Database Systems* (to appear).

[5] 半田晶彦, 山岸靖明, 中川 透, "マルチメディア／ハイパーメディア符号化標準 (MHEG) の動向", 電子情報通信学会技術研究報告 DE92-42, pp. 25–31, 1992.

[6] 春本 要, 仲秋 朗, 塚本昌彦, 西尾章治郎, 宮原秀夫, "抽象構文記法 1 (ASN.1) に基づくデータベースシステム", 情報処理学会第 97 回データベースシステム研究会報告, pp. 41–50, 1994.

[7] 仲秋 朗, 春本 要, 齋藤 淳, 塚本昌彦, 西尾章治郎, "OODBMS を用いた ASN.1 データベースの実現", 情報処理学会第 65 回マルチメディア通信と分散処理研究会報告, pp. 151–156, 1994.

[8] 中川路哲男, 勝山光太郎, 宮内直人, 玉田 純, 水野忠則, "OSI ディレクトリシステムにおける DIB（ディレクトリ情報ベース）のオブジェクト指向アプローチによる実現", 情報処理学会論文誌, Vol. 32, No. 3, pp. 304–313, 1991.

[9] 西山 智, 堀内浩規, 横田英俊, 小花貞夫, 鈴木健二, "OSI 管理情報ベース (MIB) 用データベースの設計と実装", 情報処理学会第 95 回データベースシステム研究会報告, pp. 59–66, 1993.

[10] 西山 智, 小花貞夫, 堀内浩規, 鈴木健二, "拡張可能 DBMS 構築技法に基づく高速 OSI ディレクトリ用 DBMS の設計と評価", 情報処理学会論文誌, Vol. 34, No. 6, pp. 1486–1496, 1993.

[11] 小花貞夫, 西山 智, 鈴木健二, "リレーショナルアプローチによる OSI ディレクトリの DIB（ディレクトリ情報ベース）の実装と評価", 情報処理学会論文誌, Vol. 32, No. 11, pp. 1488–1497, 1991.

[12] Ohori, A., Buneman, O.P., and Breazu-Tannen, V., "Database Programming in Machiavelli: a Polymorphic Language with Static Type Inference", in *Proceedings of ACM-SIGMOD '89*, pp. 46–57, 1989.

[13] 増永良文, "マルチメディア文書の標準データベース格納構造の考察", 情報処理学会第 93 回データベースシステム研究会報告, pp. 83–92, 1993.

[14] 空 一弘, 岸本康成, 渡辺一成, 窪田光裕, "RDBMS による OSI ディレクトリの実現", 情報処理学会第 95 回データベースシステム研究会報告, pp. 85–94, 1993.

[15] 高木利久, "遺伝子データベースの現状と研究開発動向", in *Proceedings of Advanced Database System Symposium '92*, pp. 27–33, 1992.