データベースシステム 99−32
（1994. 7. 22）

拡張可能なデータベース管理システムのための枠組

渡辺美樹　　早田宏
富士ゼロックス(株) システム・コミュニケーション研究所

オブジェクト指向データベース管理システム OODBMS では，同一のアプリケーションによる様々な種類のオブジェクトの操作を効率良く処理できることが求められている．この要求に対して，我々は，オブジェクトやそれに対する操作の種類の変化に応じて OODBMS の振舞いを動的に変更することが効果的であると考えた．本論文では，我々が研究を進めている OODBMS *Earth* において動的にシステムの動作を変更するための枠組 *OEF* (Object Extension Framework) を提案する．

Earth はオブジェクト指向の手法で設計されており，システムは複数のオブジェクトにより構成されている．*OEF* はシステムを構成するオブジェクトを，参照の対象となる実体オブジェクト (identity object) とそのオブジェクトに対する機能の振舞いを規定する振舞いオブジェクト (behavior object) に分離し，実体オブジェクトに振舞オブジェクトを追加，結合する方法を提供する．これにより，応用プログラムは適切な振舞いを適切な方法で利用することが可能となる．

さらに，本論文では，拡張の自由度を，システムまたはオブジェクトの振舞いの設計空間とその上の操作，操作のタイミングにより定め，*OEF* の拡張性が高いことを示す．

A Framework for Behavior Extension of Objects in DBMSs

Yoshiki Watanabe　　Hiroshi Hayata
Systems & Communications Lab., FUJI XEROX Co., Ltd.
KSP/R&D Business Park Bldg.
3-2-1, Sakado, Takatsu-ku Kawasaki-shi, Kanagawa-ken, 213 Japan

Dynamic extensibility of OODBMSs is one of the key factors to attain multi-purpose object storage platforms; it allows applications to operate various kind of objects efficiently. On an OODBMS *Earth* which is designed in an object-oriented approach, its dynamic extensibility is achieved with using dynamic extensibility of its constructing objects. This paper proposes a framework (*OEF*: Object Extension Framework) to design the constructing objects.

In *OEF*, a constructing object is divided into an identity object and behavior objects, and five binding ways are provided. *OEF* allows application programmers to add and select the behavior which is suitable to their applications, in the suitable way.

To evaluate extensibility of *OEF*, we also define metrics of extensibility of OODBMSs. The metrics are defined based on a design space of behavior of systems or objects, three operations on the design space, and four timings on which the operations can be applied. The metrics show that *OEF* allows dynamic selection of behavior, which is not allowed on existing extensible OODBMSs.

1 研究の背景と目的

オブジェクト指向データベース管理システム (OODBMS) は, 文書データベース, 画像データベースなど, 格納するデータの構造, 大きさの変化が多岐に渡る場合に用いられている.

変化に富むデータを効率良く管理する手段として, クラスタリング, トランザクションなど DBMS が提供する様々な機能の振舞い[1]を, データの数, 大きさの違いおよびそれらの変化に応じて, 応用プログラムの制御の元で動的に変更することが有効であると考えられる.

本研究の目的は, 上述の仮説を検証する手段として, 我々が開発を進めている *Earth*[5, 6] のクラスタリング機能の振舞いを動的に変更できる OODBMS の構成方法 (枠組) を提供することにある.

2 既存の DBMS の拡張性における問題

拡張可能な DBMS の拡張方法は, 以下の 2 種類に分けられる.

一つは, 単一の完成されたシステムにルールを与えることで質問処理の最適化処理などを変更する拡張方法である. この方法では, 変更できる部分や変更内容がルールの適用部分, 記述能力で限定されるため, DBMS の提供する種々の機能を拡張することは困難である.

もう一つの種類の拡張方法は, モジュールの組合せにより DBMS の振舞いを選択できるようにするものである. これらの拡張方法では, モジュールのインタフェースを明確に定義し, モジュール間, 応用プログラムとモジュールの接続を変更することで DBMS の振舞いを変更できる. このような拡張方法を提供する DBMS はツールキット・ベースの DBMS と呼ばれる [3].

従来のツールキット・ベースの DBMS では, 応用プログラムと DBMS のモジュールの接続は, 応用プログラムの構築時, または起動時に決定される. そのため応用プログラムの実行時に DBMS の振舞いを変更できない.

例えば, *Earth* はツールキット・ベースの DBMS であり, オブジェクト指向の手法で設計されている. この場合, DBMS の振舞いはオブジェクトに結びつけられたメソッドにより定められ, モジュール間, 応用プログラムとモジュールの間の機能の接続は, 特定のオブジェクトへの参照によって決定される. このようなシステムにおいて, 応用プログラムの制御の元で動的に DBMS の振舞いを変更することは, オブジェクトへの参照を保ちながらメソッドを変更するであり従来のツールキット・ベースの構成では実現できなかった.

3 解決の方針

オブジェクトへの参照を保ちつつメソッドを変更するには, オブジェクトの一意性の性質からメソッドを分離することが必要である.

よって, 我々は, DBMS を構成するオブジェクトを一意性を表す実体オブジェクト (identity object) と振舞いを表す振舞オブジェクト (behavior object) に分離し, その結合を応用プログラムから制御するための枠組 (*OEF*: Object Extention Framework) を定める.

4 *OEF* の提案

本節では, 動的な振舞いの拡張性を達成する枠組 *OEF* を提案し, その拡張性を示す. 説明のための例として, データベースに登録される集約オブジェクトを管理する機能を提供するオブジェクトを用いる.

集約オブジェクトはオブジェクトの集まりを表すオブジェクトである. 現在の *Earth* の集約オブジェクトのクラスは, 動的な拡張を考慮した設計にはなっていない. そこで, 以下ではこのクラスを再設計した例 **Collection Handler**

[1] アルゴリズムやアルゴリズムに対するパラメタなどにより規定される処理内容を, ここでは振舞いと呼ぶ.

を用いて *OEF* を説明する.

Collection Handler のインスタンスは応用プログラムが操作する1個の集約オブジェクトにつき1個だけ存在し, 応用プログラムは **Collection Handler** の機能を呼び出すことで, 対象となる集約オブジェクトの登録を DBMS に依頼する. ここでは, **Collection Handler** が以下の7個の機能を提供するものとする.

create() 集約オブジェクトをデータベースへ登録する. 登録時に論理的な識別子 unique id を生成し物理的な識別子 location id との組をオブジェクト・テーブルに登録する場合と, unique id を生成しない場合があり, その選択は **Collection Handler** の生成時に決定される.

destroy() 集約オブジェクトを削除する. 削除される集約オブジェクトに unique id が生成されている場合にはオブジェクト・テーブルからも削除する.

put(int num, void* obj) num 番目の要素として obj を追加する.

get(int num) num 番目の要素を取り出す.

remove(int num) num 番目の要素を削除する. 要素の追加, 取り出し, 削除の機能は, 要素を記憶するためのデータ構造に依存する. データ構造としては, 固定長の配列, 可変長の配列, リストがある. 使用するデータ構造は動的に変更できる.

load(int num) num 番目の要素をディスクから読み出す.

store(int num) num 番目の要素をディスクへ書き込む.
読み書きは, 要素単位または要素が属するページ単位に行うことができる. どの方法で読み書きするかは, 応用プログラムの処理に応じて動的に変更できる.

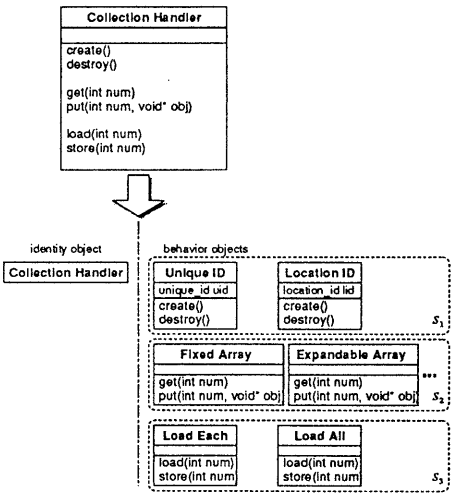

図 1　一意性と振舞いの分割

注) 表記は OMT [7] の表記法に従っている.

4.1　*OEF* におけるオブジェクトの構成

上述の **Collection Handler** の設計に *OEF* を適用する. **Collection Handler** の提供する機能は, 依存関係により大きく三つのグループに分けられる.

$$S_1 = \{\mathsf{create}(), \mathsf{destroy}()\}$$
$$S_2 = \{\mathsf{put}(), \mathsf{get}(), \mathsf{remove}()\}$$
$$S_3 = \{\mathsf{load}(), \mathsf{store}()\}$$

例えば, S_1 は, create() が unique id を生成する場合には, delete() は, オブジェクト・テーブルを更新する削除を行わなければならない.

前述の説明にあるように, 各グループ S_n には複数の実現方法が用意されている. よって, これらの実現は **Collection Handler** とは別個の複数のオブジェクトに分割して定義する (図1). 実現が定義されているオブジェクトを振舞いオブジェクト (behavior object) と呼ぶ. **Collection Handler** は, 実現の選択に関わらず, 常に応用プログラムから参照されるため, その一意性が保証されていなければならない. よって, このオブジェクトを実体オブジェクト (identity object) と呼ぶ.

図 1 では, 各グループ S_n に対して複数の

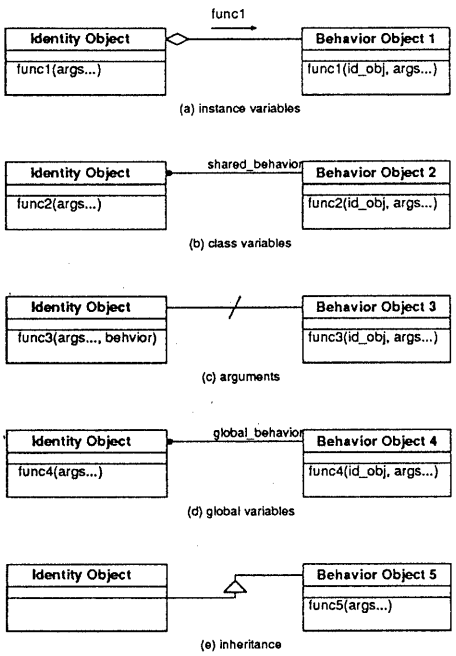

図 2　一意性と振舞いの結合

振舞いオブジェクトが定義されている．S_1 としては，unique id を生成する **Unique ID** と unique id を生成しない (location id のみで参照する) **Location ID** が定義されている．また，S_2 としては，データ構造として固定長の配列を用いる **Fixed Array**，可変長の配列を用いる **Expandable Array** などが定義されている．S_3 としては，ディスクの読み書きを要素単位に行う **Load Each** とすべての要素を同時に読み書きする **Load All** が定義されている．

Collection Handler の機能を使用する応用プログラムは，実体オブジェクトである **Collection Handler** と適宜選んだ振舞オブジェクトを結合して利用する．結合の方法として，以下の 5 種類がある (図 2 参照)．

(a) インスタンス変数　インスタンス変数を用いて，全体 - 部分の関係で結合する．実体オブジェクトに対するメソッドの起動はインスタンス変数により結合された振舞いオブジェクトに委譲される．実体オブジェクトごとに異なる振舞いを設定できる．また，実体オブジェクトに対するメソッドの呼び出しにとって透過に振舞いを変更できる．

(b) クラス変数　同じクラスのインスタンスに共通な変数であるクラス変数により結合する．同じクラスの実体オブジェクトに対して同じ振舞いを指定，変更できる．

(c) 引数　実体オブジェクトに対するメソッドの引数として振舞いオブジェクトを指定する．同じ実体に対して，メソッドの起動ごとに異なる振舞いを指定できる．

(d) 大域変数　DBMS 全体から参照できる大域的な変数により結合する．DBMS を構成しているすべてのオブジェクトの振舞いを同時に指定，変更できる．

(e) 継承　実体オブジェクトと振舞オブジェクトのサブクラスとして新たなクラスを定義する．オブジェクトの生成時にどのサブクラスを用いるかで振舞いが決定する．

これらの結合方法の使い分け方は，次節に示す拡張性の尺度を用いて 6.2 節に示す．

5　振舞いの拡張性の尺度

OEF の拡張性を示すために，振舞いの拡張の自由度を設計空間 D と設計空間における 3 種類の操作 X，X を実行できる 4 種類のタイミング T で定義する．

5.1　設計空間 D

DBMS は，トランザクション機能，記憶管理機能，通信機能など，幾つかの相互に独立した機能により成り立っている．これらの機能は，さらにそれを構成するための相互に独立したサブ機能によって構成することができる．各機能には，幾つかの実現方法の選択肢がある．よって，DBMS の設計空間は，それを構成している各機能を軸とし，各機能の選択肢を軸上の点とした空間として表すことができる．ある特定の選択肢により実現されている DBMS は，設計空間

中の一つの点として表される．また，各機能についても，それを構成しているサブ機能を用いて，同様に設計空間を定めることができる．

以下では，設計空間を D，設計空間を構成している軸を機能軸とよび F で表す．また，機能軸 F 上の点を実現と呼び f で表す．

5.2 拡張操作 X

DBMS に対する拡張操作 X は，設計空間 D を用いて，以下のように定義される．

$b' = sel(b, F_i, f_j)$ **(実現選択)** 振舞い b を，機能軸 F_i の一つの実現 f_j を用いた振舞い b' に変更する．

$D' = ins(D, F_i, f_j)$ **(実現追加)** 機能軸 F_i に実現 f_j を追加し設計空間 D を拡大する．

$D' = proj(D, F_i)$ **(機能追加)** 機能軸 F_i を追加し設計空間 D を拡大する．

X, D を用い，「システムの振舞いの拡張」を「X を D に適用すること」と定義する．

5.3 タイミング T

拡張操作 X を実行できるタイミング T には以下の 4 種類がある．

T_0 **(拡張不可能)** どのようなタイミングでも拡張操作が実行できない．

T_1 **(静的)** DBMS または応用プログラムをコンパイル，リンクする際に拡張できる．

T_2 **(プログラム動的)** 応用プログラムがデータの操作を開始する前に拡張できる．

T_3 **(完全動的)** 応用プログラムがデータの操作を開始した後であっても，応用プログラム自身により拡張操作が実行できる．

5.4 尺度

上記の 3 種類の拡張操作 X と 4 種類のタイミング T により，既存の拡張可能な DBMS の拡張性を表現することができる．たとえば，ツールキットベースの DBMS の拡張は，モジュールの組合せの変更，モジュールの追加によって行う．モジュールの組合せの変更は拡張操作における実現選択にあたり，モジュールの追加は，実現追加，機能追加に相当する．

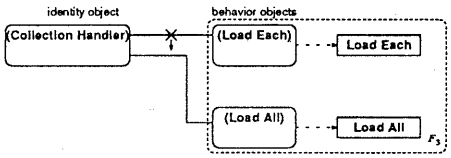

図 3 実現選択 (sel)

3 種類の拡張操作と 4 種類のタイミングによって，様々な DBMS の拡張性を表現することで，DBMS の提供する拡張能力を同じ基準で評価することが可能となる．

6 *OEF* における拡張の自由度

OEF によって達成される拡張の自由度を，前節に示した拡張性の尺度で評価する．

6.1 拡張操作

4.1 節に示した *OEF* における振舞オブジェクトのグループ S_n は拡張の尺度における設計空間の F_n に対応し，振舞オブジェクトは設計空間における機能 f_i に対応する．よって，各拡張操作は 4 節の **Collection Handler** の例で以下のように実施できる．

実現選択 図 3 のように，機能軸 F_3 の実現を定めている振舞オブジェクト **(Load Each)** を別の振舞オブジェクト **(Load All)** に置き換える．

実現追加 図 4 に示されているように，機能軸 F_3 の実現を定める新しいオブジェクト **(Load Part)** を定義する．

機能追加 図 5 の **Collection Handler** に対し，インデクスのための新しい機能軸 F_4 を定め，その実現を定める新しいオブジェクト **Hash** を定義する．

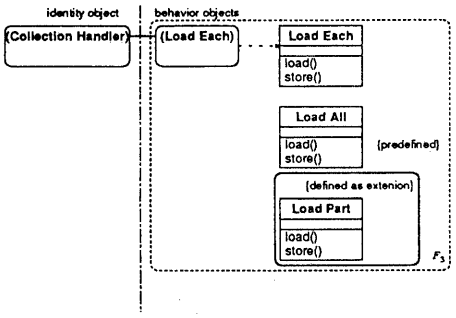

図 4　実現追加 (ins)

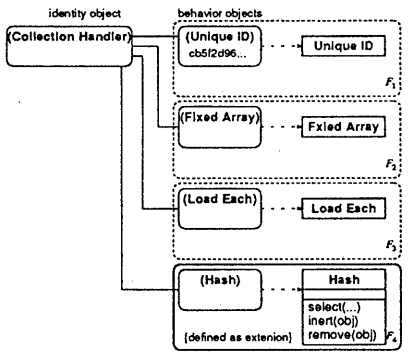

図 5　機能追加 ($proj$)

6.2　拡張のタイミング

4 節に示した実体オブジェクトと振舞オブジェクトを結合する 5 種類の方法を拡張のタイミング T によって評価する．

実体オブジェクトと振舞オブジェクトをインスタンス変数を用いて結合する場合，sel は図 3 に示されるように，インスタンス変数により参照される振舞いオブジェクトを別の振舞いオブジェクトに置き換えることで実施される．インスタンス変数の値の変更は，プログラムの実行時に可能であるから，拡張の自由度は T_3 である．また，ins を行うには，図 4 に示されているように，新たなクラス定義の追加が必要であり，$proj$ を行うには，図 5 に示されているように，新たなクラス定義の追加と，実体オブジェクトのクラス定義の変更が必要である．これらが実施できるタイミングは実装言語の能力に依存する．例えば，C++ ではクラス定義の追加はプログラムの作成時にのみ行える[2]が，Smalltalk-80, CLOS などクラスをオブジェクトとして操作できる言語では，アプリケーション・プログラムによりクラス定義の追加を行うことができる．

クラス変数で結合する場合の拡張の自由度はインスタンス変数を用いる場合と等しい．

引数によって結合する場合，sel はインスタンス変数による結合と同様に T_3 の自由度で可能である．また，ins を行うには新たなクラス定義の追加が必要であり，$proj$ を行うには新たなクラス定義を追加し，実体オブジェクトのクラス定義を変更し，実体オブジェクトを引数として振舞いオブジェクトの機能を呼び出すようなメソッドを追加する必要がある．

大域変数を用いて結合する場合，sel の自由度はインスタンス変数，クラス変数の場合と同様に T_3 である．ins, $proj$ の自由度は引数を用いた場合とほぼ同じであるが，大域変数の追加が必要である．また，実体オブジェクトにメソッドの追加を行う代わりに，大域的な手続きを用意し，その手続きの中で振舞オブジェクトのメソッドの起動を記述することも可能である．

継承を用いて結合する場合，sel を行うには継承関係を変更する必要がある．また，ins を行うには新たなクラス定義の追加が，$proj$ を行うには新たなクラス定義の追加と継承関係の追加が必要となる．

以上をまとめたものを 表 1 に示す．このように OEF の 5 種類の結合方法の拡張の自由度は実装言語によって異なる．例えば，C++ では，表 1 の (1) から (5) のタイミングは，すべて T_1 となる．よって，C++ では，sel が T_2, T_3 である必要のないものを継承で結合し，それ以外のものについては，必要に応じて，インスタンス変数，クラス変数，大域変数，引数による結合を選択的に用いる．

図 6 に 4 節で用いた例 **Collection Handler** に振舞オブジェクトを結合した例を示す．この

[2]UNIX などのダイナミックリンクを用いれば，プログラムの実行時にクラス定義を追加できる．

表 1　拡張操作のタイミング

結合方法	拡張の自由度		
	sel	*ins*	*proj*
(a) インスタンス変数	T_3	(1)	(2)
(b) クラス変数	T_3	(1)	(2)
(c) 引数	T_3	(1)	(2)
(d) 大域変数	T_3	(1)	(3)
(e) 継承	(4)	(1)	(5)

(1) ～ (5) は実装言語に依存して以下のようにタイミングが異なる．

(1) クラス定義の追加のタイミング

(2) クラス定義の追加と変更 (インスタンス変数，クラス変数の追加) のタイミング

(3) クラス定義の追加と変更 (メソッドの追加) のタイミング

(4) 継承関係の変更のタイミング

(5) クラス定義の追加と継承関係の追加のタイミング

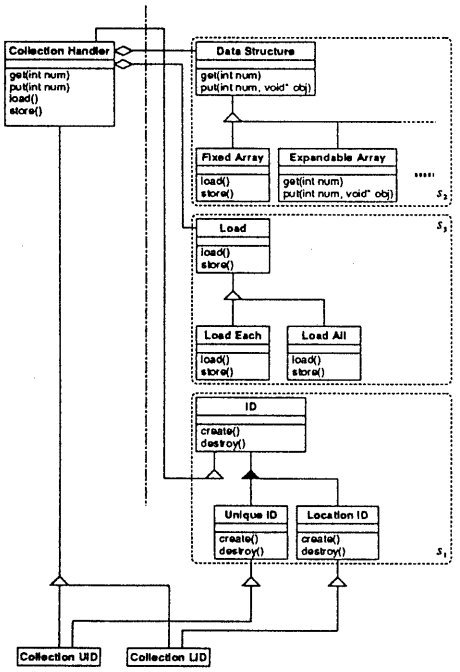

図 6　C++ での実例

例では **Data Structure**, **Load** をインスタンス変数を用いて結合している．また **ID** は静的に決定するものとして継承により実現している．

7　関連研究

7.1　拡張可能な DBMS の拡張性の比較

DBMS に拡張性を与える試みには，大きく分けて，ルールなどの記述によるカスタマイズ可能システム，モジュールの組合せにより拡張を可能とするツールキット・システムがある [3]．各々の手法について具体的な研究を紹介し，それらの拡張性を 5 節の尺度を用いて評価する．

カスタマイズ可能システム　Starburst [4] は，DBMS にルールを与えることで質問処理の最適化処理などを変更できる．実現の選択はルールを選択的に適用することで動的に可能である．実現の追加はルールの追加により行うため，動的に行うことは困難である．ルールにより変更できる処理が限定されているため，機能軸の追加はできない．

ツールキット・システム　*Earth*, EXODUS[2], GENESIS[1], Open OODB[8] など，定められたインタフェースに基づいて作成されたモジュールの組合せによってシステムを拡張可能とするアプローチでは，*sel* はモジュールの選択，*ins* は代替モジュールの追加，*proj* はインタフェースの変更によって行う．これらは T_1 でのみ可能である．

また，[8] では開放性を，拡張可能な開放性，プログラム可能な開放性，モジュールによる開放性に分けている．これらは，それぞれ *sel*, *ins*, *proj* に相当すると考えられる．しかし，Open OODB のアーキテクチャでそれらをどのように達成するかは示されていない．

7.2　関連研究と *OEF* の比較

図 7 に，上述の関連研究における各 DBMS の拡張性と *OEF* により得られる拡張性を，5 節に示した尺度で比較した結果を示す．*OEF* の拡張性は，*Earth* を *OEF* に基づいて C++ で実現したものとして示している[3]．

[3] ダイナミックリンクの機能を用いれば，*OEF* の *ins* は T_3 で可能である．

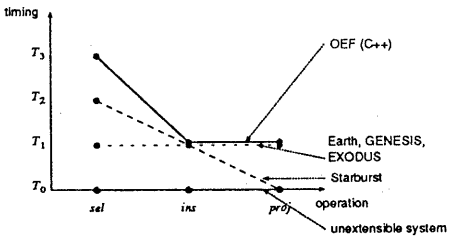

図 7　*OEF* (C++) の拡張性

この図より，*OEF* を用いて実現した *Earth* における *sel* が T_3 で可能であり，他のシステムにおける拡張性よりも高いことがわかる．

8　結論

DBMS を構成するオブジェクトの振舞いの拡張のための枠組 *OEF* を提案し，それを C++ で実装する方法を示すことにより，応用プログラムの制御の元で DBMS の振舞いを動的に変更する機能が実現可能であることを示した．

また，*OEF* の拡張性を設計空間と設計空間に対する操作により評価し，既存の拡張可能な DBMS と比較して，動的な振舞いの変更において優位性を持つことを示した．

9　今後の研究課題

OEF と拡張の尺度の妥当性を示すために，*OEF* を利用して *Earth* のクラスタリング機能を実現する．妥当性を示すための評価項目としては，クラスタリングの振舞いを動的に変更することで得られる処理効率の向上，オブジェクトを分割するによるメモリ効率の悪化，仮想関数を使用するための処理効率の低下が考えられる．

メモリ効率，処理効率が低下する場合には，拡張の自由度と実行効率のトレードオフを考慮し，振舞オブジェクトとして機能をまとめる必要がある．その際の基準を明確にすることが必要であろう．

また，拡張の操作におけるシステムの動作の一貫性の保持の仕組みを与えることも今後の課題である．

謝辞

富士ゼロックス (株) システム・コミュニケーション研究所の篠岡主幹，小部係長には *OEF* および拡張性の尺度の考え方にコメントをいただきました．ここに感謝の意を表します．

参考文献

[1] D. S. Batory, J. R. Barnett, J. F. Garza, K. P. Smith, K. Tsukuda, B. C. Twichell, and T. E. Wise. GENESIS: An Extensible Database Management System. *IEEE Trans. Softw. Eng.*, Vol. 14, No. 11, pp. 1711–1730, 1988.

[2] Michael J. Carey, David J. DeWitt, Goetz Graefe, Dvaid M. Haight, Joel E. Richardson, Daniel T. Schuh, Eugene J. Shekita, and Scott L. Vandenberg. The EXODUS Extensible DBMS Project: An Overview. In [9], pp. 474–499. Morgan Kaufmann, 1990.

[3] Andreas Geppert and Klaus R. Dittrich. Constructing the Next 100 Database Management Systems: Like the Handyman or Like the Engineer? *SIGMOD RECORD*, Vol. 23, No. 1, pp. 27–33, 1994.

[4] Laura M. Haas, Walter Chang, Guy M. Lohman, John McPherson, Paul F. Wilms, George Lapis, Bruce Lindsay, Hamid Pirahesh, Michael J. Carey, and Eugene Shekita. Starburst Mid-Flight: As the Dust Clears. *IEEE Trans. Knowledge and Data Eng.*, Vol. 2, No. 1, pp. 143–160, 1990.

[5] 早田宏. A Modular OODBMS Earth. 第四回データエンジニアリングフォーラム, pp. 39–45. ASTEM RI, 1993.

[6] 早田宏, 佐藤直人, 小部正人. OODBMS Earth における Core の設計と実装. 情報処理学会データベース研究会資料 *92-DBS-89*, pp. 59–68, 1992.

[7] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen. *Object-Oriented Modeling and Design*. Prentice-Hall, 1991.

[8] David L. Wells, Jose A. Blakeley, and Craig W. Thompson. Architecture of an Open Object-Oriented Database Mangement System. *IEEE Computer*, pp. 74–81, October 1992.

[9] Stanley B. Zdonik and David Maier, editors. *Readings in Object-Oriented Database Systems*. Morgan Kaufmann, 1990.