データベースシステム 106－13
（１９９６．１．２５）

排他制御機構の拡張可能化についての一考察

大野 友寛　　宝珍 輝尚　　都司 達夫

福井大学 工学部 情報工学科
〒910　福井市文京３丁目９－１

本論文では，排他制御機構を拡張可能とするための基本方式について述べる．ここでは，先進的なデータベース応用分野のための排他制御法のみに重点をおくのではなく，２相ロック法，時刻印順序法や楽観的制御法といった従来の方法をも含めて拡張可能とすることを目標とする．まず，従来の排他制御法から先進的なデータベース応用分野のための排他制御法まで，幅広い範囲の排他制御法を，利用する情報，演算の種類，演算のパターン，ならびに，アクションにより記述することを試みる．この結果をもとにして，拡張可能な排他制御機構の実現について検討する．その結果，データに関する情報，トランザクションに関する情報，システムに関する情報，ならびに，演算の種別を個別に管理し，演算のパターンを含めたアクションを利用者が遷移図により記述する方式を採用する．

On the Extendible Concurrency Control Mechanism

Tomohiro OHNO　　Teruhisa HOCHIN　　Tatsuo TSUJI

Dept. of Information Science, Faculty of Eng., Fukui University
3-9-1, Bunkyo, Fukui-shi, Fukui 910 Japan

This paper describes the extendible concurrency control mechanism. The purpose of this research is that the concurrency control mechanism is made extendible for conventional methods, such as the two phase lock, the time-stamp ordering, and the optimistic methods, as well as those for the advanced database applications. First, various concurrency control methods are surveyed. Next, it is tried to describe these methods through the information used, operation types, patterns of operations, and actions. This observation brings us an extendible concurrency control mechanism. For this mechanism, informations on data, transactions, and system, and operation types are managed. Actions including the controls of operation patterns are described through transition diagrams.

1 はじめに

CAD/CAM，グループウェア，ソフトウェア開発環境といった先進的なデータベース応用分野においては，ロングトランザクションの考慮，利用者制御のサポート，協調的な協同作業のサポートといった要求がある．これらの要求を満足させるために，様々な排他制御法が提案されている [1, 3, 4, 5]．また，先進的なデータベース応用分野におけるトランザクションは階層性を持つことが多く，親子のトランザクション間の関係で様々なトランザクション管理方式が考えられる．このような多種のトランザクション管理法のサポートに関する研究も行われている [6, 7, 8, 9]．

一方，データベースの適用分野の急速な広がりによりデータベース管理システム (DBMS) の拡張性が強く求められており，DBMS のあらゆる機能を拡張可能にする研究が行われてきている [10, 11, 12]．ここでは，多種のトランザクション管理法のサポートに重点をおいて研究が行われている．

著者らが開発中の拡張可能 DBMS COMMON [13, 14] においても DBMS のあらゆる機能を拡張可能とすることを目標としている．COMMON では，DBMS もデータベース応用プログラムであるととらえ，DBMS を実現するための様々な情報を一種のデータベースとして管理し，それを使用することで DBMS を実現しようとしている．個々の情報を管理する (小さな)DBMS が従来のモジュールに相当する．従って，各モジュールでは，個々の質の異なる情報を扱い，操作は基本的に SELECT, INSERT, DELETE といった基本演算を通して行う．このような設計指針で構築中の COMMON であるが，排他制御機構やトランザクション管理機構に関しては未検討であった．

そこで，本論文では，COMMONにおいて排他制御機構を拡張可能とするための基本検討を行う．ここでは，意味に基づく排他制御 [3] やトランザクショングループ [4] といった先進的なデータベース応用分野のための排他制御法のみに重点をおくのではなく，2 相ロック法，時刻印順序法や楽観的制御法といった従来の方法をも含めて排他制御機構を拡張可能とすることを目標とする．そこで，まず，従来の排他制御法から先進的なデータベース応用分野のための排他制御法まで，幅広い範囲の排他制御法を対象に整理を行い，様々な排他制御法をいくつかの項目の組み合わせで記述することを試みる．ここでは，利用する情報，演算の種類，演算のパターン，ならびに，アクションによる記述を試みる．次に，この結果に基づいて，拡張可能な排他制御機構の実現について検討する．本論文では，簡単のために，従来の排他制御法を対象に検討を行う．そして，開発の現状について述べる．

以下，2. で，様々な排他制御法について概説する．そして，3. で排他制御機構の拡張可能化の基本方針について述べる．4. では開発の現状について述べ，5. でまとめを行う．

2 排他制御法について

代表的な排他制御法について概説する．

2 相ロック法

データにアクセスする際にロックを設定し，データを独占的に操作する方法である．施錠を行う第一の相（成長相）と解錠を行う第二の相（縮退相）がある．

時刻印順序法

トランザクションには単調増加する一意な値 (タイムスタンプ (TS)) が与えられる．また，データは read-TS, write-TS を持つ．データを read した時には read-TS，write した時には write-TS をトランザクションの TS に更新する．遅く開始したトランザクションに先行された時点でアボートする．

多版時刻印順序法

前記の時刻印順序法を拡張した方法である．データは < write-TS, 値 > の集合と read-TS の集合を持つ．トランザクション T_i がデータ x を読むと，T_i の TS を x の read-TS に追加する．トランザクション T_i がデータ x に書き込む場合は，T_i の TS が x の最新の版の TS よりも小さいとき T_i はアボートされ，大きいとき T_i の TS を持つ版を作成する．

楽観的制御法

アクセスの競合はないものとし，read, write した履歴を蓄えながら実行し，トランザク

ション終了時に競合が発生していないか確認する方法である．競合が発生した場合，トランザクションはアボートされ再実行される．

利他的 (altruistick) ロック法

2相ロック法の拡張で，いつ資源が不要になるかという情報を用いて並行性を上げる方法である．すなわち，不要なロックを解除することによりロック待ちを少なくする．今後アクセスする予定のデータ集合を正のアクセス情報とし，過去にアクセスしたが今後アクセスしないデータ集合をwakeとする．不要になったロックはその場で解除され，データ項目がwakeに追加される．あるトランザクションT_iのアクセスするデータ集合が他のトランザクションT_jのwake中にある場合は，それらが完全にT_jのwake中に含まれるか，または，T_jが終了するまでwaitする必要がある．T_iのデータ集合が全てT_jのwake中にあれば，T_jの終了を待たずにT_iをコミットできる．

Snapshot validation

楽観的制御における処理の衝突を，危険な衝突と危険でない衝突とに分けることで，トランザクションの再実行のリスクを減らす方法である．危険な衝突とは，再開始が必要な衝突である．衝突が危険か否かを判断するために，トランザクションはデータをアクセスした時刻と他のトランザクションの終了した時刻を記述したread setを持つ．評価相での競合の検査において，他のトランザクションの終了前にreadしたデータがあるか検査し，あれば危険な衝突があったことが分かる．

トランザクションの意味を用いる方法

トランザクションの意味的な情報を利用することにより，処理の衝突を緩和する方法である．すなわち，2つのトランザクションが意味的に可換であれば実行順序を変更できるとし，処理の衝突を緩和する方法である．

チェックイン・チェックアウト法

1度に1ファイルしかアクセスできないようにして版管理を行い，自分が作成した版には排他的にアクセスする．複数ユーザは，同時に同一の実体を並行的な版によって更新できる．版の競合はユーザが手動で解決する．

会話的なトランザクション法

DBは，Public DBとPrivate DBから構成される．Public DBは全利用者で共有され，Private DBにおいて利用者はロングトランザクション(LT)を開始する．Public DB中のデータが必要な場合は，ショートトランザクション(ST)によりチェックアウトを試みる．もし衝突を起こさなければ，そのSTはLTの期間，そのデータにロックをかける．データが他のトランザクションによりチェックアウトされているならば，STは利用者にアクセス不可の旨を通知しアボートする．

トランザクショングループ法

意味的な情報を使用するトランザクション管理方法の一つである．トランザクションの意味を用いる方法では，文字通りトランザクションの意味的な情報を用いたが，トランザクショングループ(TG)法ではデータの意味的な情報を利用する．トランザクションは入れ子関係を持つ．この入れ子関係を表す木の2ノードの差し込みは最小共通先祖によって定義される．これは，意味的なパターンを定義することである．意味的なパターンとは，オブジェクト内／間の一貫性を保持できる演算列で，抽象データ型とTGにより定義される．

3 排他制御機構の拡張可能化について

3.1 排他制御法の整理

2で述べた排他制御法を，利用する情報，演算の種類と演算のパターン，アクションの観点からまとめて，各々，表1，表2，表3に示す．

利用する情報は，大きく，データに関する情報，トランザクションに関する情報，システムに関する情報に分けて示した．演算では，全ての方法に存在するトランザクションの開始・終了に関する演算は記述していない．また，アクションでは，排他制御機構を中心に記述しており，トランザクションの開始・終了に関するアクションは特に必要のある場合を除いて記述していない．

表1　利用する情報

排他制御法	データ			トランザクション	システム／サーバ
	データ実体	DB	DBの一部		
2相ロック法	状態*1	–	–	–	–
時刻印順序法	read-TS, write-TS, { (TS, 値) }	–	–	TS	–
多版時刻印順序法	{read-TS}, { (write-TS, 値) }	–	–	TS	–
楽観的制御法	–	–	–	TS, Tr.read, Tr.write	Tr.write
利他的ロック法	状態*1	–	–	wake, 正のアクセス情報	–
Snapshot validation	–	–	–	TS, read-set	read-set
Semantic atomicity	状態*2	–	–	type	可換集合
Checkin/checkout	–	状態*3	–	–	–
Conversational Tr.	–	–	状態*3	–	–
Tr. group	直前までの演算パターン	–	–	–	type毎のconflict-table

*1: lock, unlock 等　*2: 可換な Tr の type の集合　*3: checkout, checkin

表2　演算の種類と演算のパターン

	演算	演算のパターン
2相ロック法	lock, unlock 等	$\{lock\text{各種}\}^{*}unlock^{*}$
時刻印順序法	read, write	$\{read, write\}^{*}$
多版時刻印順序法	read, write	$\{read, write\}^{*}$
楽観的制御法	read, write	第1相 ($\{read, write\}^{*}$) + 第2相 (確認)
利他的ロック法	lock, unlock	$\{lock, unlock\}^{*}$
Snapshot validation	read, write	第1相 ($\{read, write\}^{*}$) + 第2相 (確認)
Semantic atomicity	–	–
Checkin/checkout	checkout, checkin	checkout checkin
Conversational Tr.	checkout, checkin	checkout checkin
Tr. group	–	意味的なパターン

表3　アクション

	アクション
2相ロック法	状態 (X) = unlock & Op(T_i) = lock → OK, 状態 (X) = lock by T_i 状態 (X) = lock by T_j & Op(T_i) = lock → T_i.wait 状態 (X) = lock by T_i & Op(T_i) = unlock → OK, 状態 (X) = unlock
時刻印順序法	Op(Tr) = read & TS(Tr) ≥ {read-TS(X), write-TS(X)} & 暫定値なし → OK, read-TS(X) = TS(Tr) Op(Tr) = read & TS(Tr) ≥ {read-TS(X), write-TS(X)} & 暫定値あり & TS(Tr) ≥ TS(暫定値) → Tr.wait Op(Tr) = write & TS(Tr) ≥ {read-TS(X), write-TS(X), TS(X の暫定値)} → OK, write-TS(X) = TS(Tr) TS(Tr) < TS(X) → Tr.abort
多版時刻印順序法	Op(Tr) = read & 暫定値なし → OK, read-TS(X) に TS(Tr) を追加 Op(Tr) = read & 暫定値あり → read-TS(X) に TS(Tr) を追加し、 最大の write-TS(X) を持つ暫定値を read Op(Tr) = write & TS(Tr) ≥ {read-TS(X)} → OK, write-TS(X) に TS(Tr) を追加 Op(Tr) = write & TS(Tr) < {read-TS(X)} → Tr.abort
楽観的制御法	第1相：Op(T_i) = read → OK, read レコードに X を追加 Op(T_i) = write → OK, write レコードに X を追加 第2相：T_i.{read,write} レコード ∈ T_iの第1相中に第1相が終了した他 Tr の write レコード → T_i.abort T_i.{read,write} レコード ∈ T_iの第2相と第2相が同時実行されている他 Tr の write レコード → T_i.abort
利他的ロック法	状態 (X) = lock by T_i & Op(T_i) = unlock → OK, 状態 (X) = unlock 状態 (X) = lock by T_j & Op(T_i) = lock → T_i.wait 状態 (X) = unlock & Op(T_i) = lock & X ∉ 全ての wake → OK, 状態 (X) = lock by T_i 状態 (X) = unlock & Op(T_i) = lock & X ∈ T_jの wake & T_iのアクセスした全てのデータがT_jの wake 中 → OK, 状態 (X) = lock by T_i 状態 (X) = unlock & Op(T_i) = lock & X ∈ T_jの wake & T_iのアクセスした一部のデータがT_jの wake 中 → T_i.wait 状態 (X) = unlock & Op(T_i) = lock & X ∉ T_jの wake & T_iのアクセスした全てのデータがT_jの wake 中 → T_i.wait
Snapshot validation	第1相：OK, read-set に X を追加 T_iと同時実行されているT_jが終了 → T_iの read-set に T_jを追加 第2相：T_i.read-set ∈ {T_i.read-set 中の現在地点以前に終了情報のない Tr の read-set 中の write} → T_i.abort
Semantic atomicity	Type(T_j) ∈ CS(Type(T_i)) and Type(T_i) ∈ CS(Type(T_j)) → T_iと T_jは可換
Checkin/checkout	(checkout 時に版を生成) 状態 (X) = checkin & Op(Tr) = checkout → OK, 状態 (X) = checkout 状態 (X) = checkout & Op(Tr) = checkin → OK, 状態 (X) = checkin 状態 (X) = checkout & Op(Tr) = checkout → Tr.abort
Conversational Tr.	同上
Tr. group	直前までの演算パターン (X)・演算 ∈ 意味的なパターン → OK

3.2 基本方針

3.1で述べたように，(i) データに関する情報，(ii) トランザクションに関する情報，(iii) システムに関する情報，(iv) 演算の種別，(v) 演算のパターン，ならびに，(vi) アクションによって排他制御機構は記述できると考えられる．すなわち，(i)〜(vi) をユーザ定義可能とすることで，排他制御機構をカスタム化できると考えられる．

そこで，(i)〜(vi) を各々管理することとし，これらをもとに排他制御機構を実現することとする．この中で，(i)〜(iv) は単なるデータであり管理上特に問題はないが，(v) と (vi) はどのように記述し管理するかが問題となる．これらの記述方法によって，排他制御機構の記述に問題が生じたり，著しく性能が劣化する恐れがあるからである．そこで，次にこれらの記述方法について考察する．まず，(vi) アクションについて考察し，次に (v) 演算のパターンについて考察する．

3.3 アクションの記述方法

3.1ではアクションをルールによって記述したが，アクセスの競合 (すなわち，データ項目の状態とアクセス方法との関係) を表すものはルールばかりではない．そこで，ここでは幾つかの記述方法について述べ，これらを比較する．

(1) ルール

全て操作を if-then 形式で記述し，最初に合致したルールに従って処理を行う方法である．アクションの並べ換えや組換えといった操作をしやすいという利点があるが，うまく最適化しないと処理が遅い，どのルールにも合致しない，もしくは，複数のルールに合致する可能性があるという欠点がある．

(2) テーブル

従来の conflict table の方法に基づく方法で，それぞれの条件をテーブルの各要素として表す方法である．テーブルであるので，ダイレクトに条件を決定できる，常に何らかの結果を得られる事が保証されるという利点があるが，僅かな情報のために膨大なテーブルを保持しなければならない場合がある，不要な条件まで判別する必要がある，条件が複数ある場合において最初の条件ですでに abort されるような場合でも全ての条件を調べる必要がある，包含関係や大小関係をそのままでは記述することができないといった欠点がある．

表 4　アクション記述法の比較

アクション記述法	記述容易性	高速性	空間効率	動的な対応	正確さ
(1) ルール	○	△	○	○	△
(2) テーブル	×	○	×	×	○
(3) テーブル + ルール	△	△	×	×	○
(4) 遷移図	○	△	○	△	○

(3) テーブル + ルール

包含関係や大小関係のようなテーブルで記述できない条件を，ルールによって判別し，テーブルにより判断する方法である．改善された欠点以外は (2) と同じ得失がある．

(4) 遷移図

条件の値により，次に調べる条件へと分岐していく方法である．不要な条件を評価する必要がないという利点がある．

以上述べた (1)〜(4) の方法を，記述容易性，高速性，空間効率，動的な対応，正確さという観点から比較した結果を表 4 に示す．

表 4 より，(1) と (4) は同程度の良さであるが，正確さが優れ，利用者がアクションの実行順序を決定できる (4) を採用することとした．

3.4 演算パターンの記述方法

演算パターン (表 2) を見て分かることは，相 (フェーズ) を記述できるようにしておけば良いということである．このためには，各相で許される演算と相間の遷移の条件を記述する必要がある．これらを記述して厳格に相を管理する方法も考えられるが，ここでは，(1) 相間の遷移は状態遷移であって状態遷移図で書けること，ならびに，(2) 相内で許される演算はアクション中に現れることから，相はアクション記述に含めることとする．従って，演算のパターンを含めたアクションを利用者 (DBMS インプリメンタ) が記述することになる．また，相の情報はトランザクションの情報として管理する．

4 実装

現在，2 相ロック法，時刻印順序法，ならびに，楽観的制御法を対象として実装を行っている．現

在のトランザクション管理と排他制御関係の処理概要を，データフロー図風の記法で図1に示す．図の円は処理を表し，円間の矢印は遷移を表す．矢印につけられたラベルは遷移で渡されるデータである．2つの平行線はデータストアを表す．

"Tr 開始"では，トランザクション ID(TID) の取得，および，トランザクション (Tr) の登録を行う．"Tr の次の処理を実行"では，指定された Tr の次の処理命令を"排他制御"(CC) に入力する．"CC の結果を反映"では，Tr が行う予定のデータ操作を"Tr の実行予定リスト"に追加する．"Tr 終了"では，Tr の実行予定の操作を DB に反映する．"排他制御"では，入力された処理が実行できるかを登録された方法に従って判断する．この方法は，DBMS インプリメンタが遷移を行う関数を作成することで実現する．ここでは，与えられた演算を CC で認識する演算に変換する処理も行う．

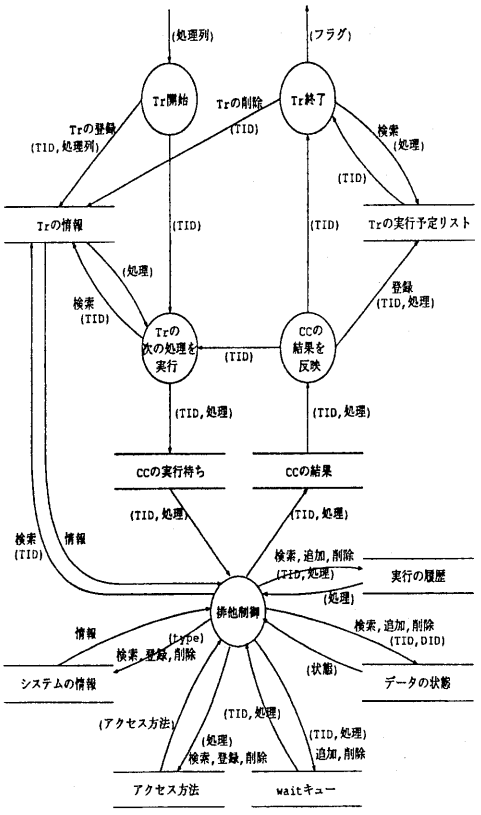

図1　処理の概要

また，"Tr の情報"では，TID と 命令の関係付けをしている．これは，3.2で述べた (ii) トランザクションに関する情報 である．"Tr の実行予定リスト"は，Tr が行う予定の (実行が確定した) データ操作のリストであり，リカバリのための情報である．"データの状態"では，CC で制御情報として用いるデータの状態を管理する．これは，3.2で述べた (i) データに関する情報 である．"実行の履歴"は，Tr が行ったデータ操作のリストである．また，"システムの情報"では，システム／サーバの定義上の情報を持つ．これら2つは，3.2で述べた (iii) システムに関する情報 に相当する．"アクセス方法"では，使用可能なアクセス方法やその外部表現から内部表現への変換情報を持つ．これは，3.2で述べた (iv) の演算の種別 に相当する．"実行の履歴"，"データの状態"や"wait キュー"等は，排他制御法によっては使用しないこともある．

以下に，各法の実現の概要を示す．

(1) 2 相ロック法

2 相ロック法のアクションを記述する遷移図の例を図2に示す．ここでは，アクセス方法 (演算)，データの状態と相を利用してアクションを記述している．"排他制御"ではこの遷移図をもとに処理を行う．ここでは，トランザクションの終了処理を"排他制御"で行っている．

(2) 時刻印順序法

時刻印順序法では，トランザクションの時刻印を参照して処理を行う．遷移図は表3のアクションをもとに (1) と同様にして作成することができる．

(3) 楽観的制御法

"終了"以外のアクセス方法では，トランザクションの read 履歴や write 履歴にアクセスしたデータ

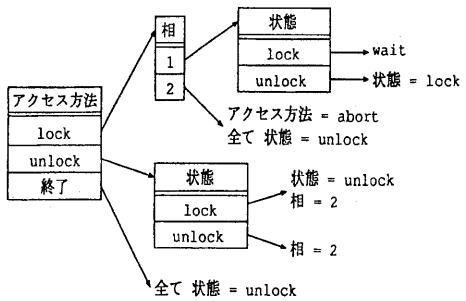

図2　2相ロック法のための遷移図

を追加するのみである．評価相における競合の検査も表3の第2相のルールをそのまま遷移図にすることで，"排他制御"で行うことができる．

アクセス方法が"終了"の場合の処理はトランザクション終了処理であり，本来トランザクション管理が行うべき処理であるが，トランザクション管理機構の詳細が未定であるため，ここでは"排他制御"のみで処理する場合を示した．これ以外に，"Tr終了"で行う方法や"Trの次の処理を実行"と"排他制御"で処理を分担する方法が考えられる．これらを含めて拡張可能なトランザクション管理機構を検討する必要がある．

5 おわりに

本論文では，2相ロック法，時刻印順序法や楽観的制御法といった従来の方法をも含めて排他制御機構を拡張可能とするための基本方式について述べた．まず，従来の排他制御法から先進的なデータベース応用分野のための排他制御法まで，幅広い範囲の排他制御法を対象に整理を行い，利用する情報，演算の種類，演算のパターン，ならびに，アクションによる記述を試みた．次に，この結果をもとにして，拡張可能な排他制御機構の実現について検討した．その結果，データに関する情報，トランザクションに関する情報，システムに関する情報，ならびに，演算の種別を個別に管理し，演算のパターンを含めたアクションを利用者が遷移図により記述する方式とした．また，開発の現状についても述べた．

今後は，排他制御対象の粒度の多様性への対処，トランザクション間の親子関係の多様性を考慮した拡張可能なトランザクション管理機構の検討，拡張可能化なリカバリ機構の検討等が課題である．

参考文献

[1] Barghouti, N., and Kaiser, G. E. : "Concurrency Control in Advanced Database Applications," ACM Computing Surveys, Vol. 23, No. 3, pp.269–317 (Sept. 1991).

[2] 滝沢 誠："データベースシステム入門技術解説"，ソフト・リサーチ・センター (1991).

[3] Garcia-Morina, H. : "Using Semantic Knowledge for Transaction Processing in a Distributed Database," ACM TODS, Vol. 8, No. 2, pp. 186-213 (1983).

[4] Skarra, A. H. and Zdonik, S. B. : "Concurrency Control and Object-Orented Databases," in Object-Oriented Concepts, Databases, and Applications (W. Kim, F. H. Lochovsky eds.), pp.395-421 (1989).

[5] 谷口 伸一，西尾 章治郎，久保 信也："オブジェクト指向データベースにおける競合保存直列可能スケジューラの有効性"，信学論 D-I, Vol. J77-D-I, No. 7, pp. 514-524 (1994).

[6] Walter, B : "Nested transaction with multiple commit points: An approach to the structuring of advanced database applications," Proc. of 10th VLDB, pp. 161–171 (1984).

[7] Chrysanthis, P. K., and Ramamrithan K. : "ACTA: A Framework for Specifying and Reasoning about Transaction Structure and Behavior," Proc. of ACM SIGMOD'90, pp.194–203 (1990).

[8] Chrysanthis, P. K., and Ramamrithan K. : "Synthesis of Extended Transaction Models Using ACTA," ACM Trans. on Database Systems, Vol. 19, No. 3, pp. 450–491 (1994).

[9] Georgakopoulos, D. *et al*: "Specification and Management of Extended Transactions in a Programmable Transaction Environment," Proc. of the 10th Int'l Conf. on DATA ENGINERRING, pp.462-473 (1994).

[10] Dittrich K. R.: "Database Technology Research at the University of Zurich: Using and Engineering Object-oriented, Active, and Heterogeneous DBMS," 信学技報 DE91–60, Vol. 91, No. 538 (1992).

[11] Geppert, A. and Dittrich, K. R. : "Constructing the Next 100 Database Management Systems: Like the Handyman or Like the engineer?," ACM SIGMOD RECORD, Vol. 23, No. 1 (March 1994).

[12] Wells, D. L. *et al*: "Architecture of an Open Object-Oriented Database Management System," COMPUTER, Vol. 25, No.10, pp.74–82 (Oct. 1992).

[13] Hochin, T. and Inoue, U. : "An Extensible DBMS Composed of Specific DBMSs," Proc. of International Symposium on Next Generation Database Systems and Their Applications, pp.180–187 (1993).

[14] 宝珍輝尚: "拡張可能 DBMS のためのデータ独立な部品間インタフェースの実現"，信学論D–I, Vol. J75–D–I, No. 11, pp.1070–1078 (1992).