データベースシステム　97－6
（１９９４．　３．　１８）

柔軟なオブジェクトを扱うOODBMSのための ECAルール機構の実現

田島 孝一[†]　下條 真司[‡]　原 伸一郎[*]　田中 克己[*]

[†]大阪大学 基礎工学部 情報工学科
[‡]大阪大学 大型計算機センター
[*]神戸大学 工学部 情報知能工学科

内容梗概

構造をもたないデータのインスタンスを格納し，後からスキーマを進化させることができるオブジェクト指向データベース管理システム (OODBMS) が注目されている．これらのOODBMSではスキーマをもたないインスタンスが大量に生成されるため，あらかじめ制約を記述しておくことで自動的にデータベースを管理する機構が必要である．本稿では，柔軟なオブジェクトを扱うOODBMSのプロトタイプであるObaseシステムのアーキテクチャについて述べるとともに，Obase内に実現したECAルール機構と呼ばれるデータベース管理機構について述べる．この実現をもとに，これらのOODBMSにおけるECAルール機構に関する問題点を議論する．

Implementation of an ECA rule mechanism for instance-based OODBMS

Koichi TAJIMA[†], Shinji SHIMOJO[‡],
Shinichiro HARA[*] and Katsumi TANAKA[*]

[†]Department of Information and Computer Sciences, Osaka University
[‡]Computation Center, Osaka University
[*]Department of Computer and Systems Engineering, Kobe University

abstract

There are several approaches to support incremental database schema evolution in object-oriented database management systems (OODBMS). Among these apporoaches, instance-based one doesn't assume the existence of classes. Since a lot of schemaless data instances will be created in instance-based databases, these systems need an automatic management mechanism to classify these instances by constraints. This paper discusses implementation issues regarding an ECA rule mechanism dedicated to Obase system, a prototype system of instance-based OODBMS. We also mention about the architecture of Obase system.

1 はじめに

データベースシステムの利用分野の広がりにともない, 科学技術, ハイパーメディア, 金融などアプリケーションに適したデータベースシステムに対する要求が高まっている. これらのデータは, データベースに登録する時点ではスキーマを完全に確定することができないという特徴を持つ. しかし従来のオブジェクト指向データベースは, あらかじめ完全に定義されたスキーマを用いてインスタンスを管理するため, このようなデータには対応できない [1].

OODB のスキーマ進化への対応を大別すると, クラスを前提とするアプローチとクラスを考えないアプローチの 2 つがある. あらかじめ構造が知られていないインスタンスを扱うアプリケーションは, 後者のアプローチである. このアプローチのデータモデルに, プロトタイプベースまたはインスタンスベースと呼ばれるモデルがある.

クラスを持たないインスタンスベースの OODBMS 実現の試みとして, Obase システム [4] のプロトタイプの開発が Obase プロジェクト[1]で進められている.

しかし Obase などのインスタンスベースの OODB においては, スキーマを持たないインスタンスが大量に生成される. データベース内の一貫性を保つためには, これらのインスタンスに対する更新操作を監視し, 一貫性制約を破るものは排除するなどの対応が必要である. ところが, これらの OODB では属性操作を監視する機構をインスタンスに持たせることが難しい. 一方, クラスがないインスタンスベースの OODB の応用システムを考えた場合, これらのシステムではスキーマに基づくデータ分類ができないためにインスタンスが無作為・無計画に作成される可能性がある. これらを自動的に分類するために, あらかじめ制約を記述しておくことで自動的にデータベースを管理する機構が必要である.

アクティブデータベースでは, イベント (Event), 発火条件 (Condition), アクション (Action) を記述した ECA ルールをもとに, 一貫性やデータベース操作に関する制約を記述することができる. データベース操作などに応じてイベントが発生したとき, ルールの発火条件が満たされていれば, 指定のアクションが実行される.

ここでは, Obase システムを一例に, 柔軟なオブジェクトを扱う OODBMS に必要とされる ECA ルール機構を実現する. また, これらの OODBMS における ECA ルール機構に関する問題点を議論する.

[1]Obase プロジェクトは (財) 千里国際情報事業財団, 5 大学, 8 企業からなる OODBMS に関する共同研究プロジェクトで, 商用 OODBMS の調査, 種々の応用システム開発, オブジェクトベース管理システム Obase の開発などを主目的としている.

2 Obase モデル

Obase オブジェクトデータベースモデル [4][5] は, C++ オブジェクトモデルに基づく従来の OODBMS の欠点を克服することを目標とし, 各オブジェクトを集合と組の両方の構造的構成子として表す. Obase モデルでは, 型, クラスやインスタンスオブジェクトを区別せずに統一的に扱う.

Obase オブジェクトは, オブジェクト識別子 (OID), サブオブジェクト集合, プロパティ列の 3 つの構成子から構成される. サブオブジェクト集合は OID の集合であり, 要素 OID で参照されるオブジェクトのことをサブオブジェクトと呼ぶ. また, サブオブジェクト集合を含むオブジェクトのことをスーパオブジェクトと呼ぶ. スーパ/サブオブジェクト関連が表す意味は, 従来の OODBMS でサポートされてきた a-kind-of 関連, instance-of 関連および member-of 関連を包括している. スーパ/サブオブジェクト関連全体は非巡回有向グラフを構成する. データベース内には推移的に全部のオブジェクトのスーパオブジェクトとなっている特別なオブジェクト, "ObjectWorld" が存在する. Obase オブジェクトは 0 個以上のプロパティからなるプロパティ列を持つ. プロパティはひとつの属性/属性値, ひとつのメソッドを表す. ただしプロパティとして持つメソッドは, データベース更新操作を含まない, データベース内に副作用を及ぼさないメソッドの記述である. また, 属性値には文字列, 数値, ブール値などの原始値やオブジェクト識別子のほか, 経路式や問い合わせを格納することができる.

2.1 Obase 言語

Obase 言語 [6][7] は, Obase モデルを表現したユーザインタフェース言語である. Obase 言語は Obase データベースの操作言語であり, Obase 検索文, Obase 更新文およびトランザクション文から構成される.

Obase 検索文は Obase 代数式 [7] の一部と ObaseSQL 検索文で構成される. ObaseSQL では, Obase オブジェクトの構造的な特徴である組, 集合の両構成子に対応する航行演算子を導入している. また, データモデルの特徴である柔軟な継承指定に基づくオブジェクトの多様なビューの提供を実現するために, 組, 集合両構造に沿った継承演算子を持つ. 構文は了解性のために基本的には

SQL にならうが, 自由度を高くし通常の SQL を拡張した多様な表現法を許す (例えば, FROM 句が無いような SQL 文も許す).

オブジェクトの作成・削除, オブジェクト名の変更, プロパティの追加・削除・変更, スーパ/サブオブジェクト関連の追加・削除・変更には, Obase 更新文を用いる. Obase 更新文では, 操作対象を Obase 検索文などを用いて指定することができる. このため, ひとつの更新文で複数のオブジェクトに対する操作を記述することができる. 一般に, Obase 更新文は実行結果として OID 集合を返す.

3 ObaseECA 機構の設計

ObaseECA は, Obase システムのインスタンスを監視する機構である. また, Obase 言語ではデータベース内に副作用を及ぼすメソッドをオブジェクトに持たせることが許されないため, ObaseECA の記述はオブジェクトに更新操作をともなうメソッドを定義する役割を果たす.

3.1 ObaseECA モデル

ECA ルールは ObaseECA 記述言語を用いて, 以下の形式で記述する.

"ON"	イベント名・タイミング (Event) "at" 束縛対象 (Inverse Query)
"IF"	発火条件 (Condition)
"THEN"	アクション (Action)

3.2 イベント

ObaseECA モデルではイベントとして, 外部アプリケーションで定義された信号入力, 時間・時刻, データベースの更新操作を含むことができる. データベース更新操作にはオブジェクト間のスーパ/サブ関連の変更が含まれている. また, ルールにイベントを記述するときは, Obase 更新文の名前をイベントに指定する. 例えば, deleteObject という更新操作を監視するときは, 更新文の名前 "deleteObject" をイベントに指定する.

更新操作に対応するイベントの発生タイミングを 3 種類定義する.

Before: 更新操作の呼び出し時
After: 更新操作の終了時 (コミット前)
Separate: 更新操作のコミット時

Before, After, Separate の各タイミングに発生するイベントは, 例えば更新操作が deleteObject であれば, それぞれ "-deleteObject", "deleteObject", "deleteObject+" と記述する. それぞれの役割や実現については 3.5 節で述べる.

3.2.1 監視対象の動的束縛

ルールの監視対象には, その対象が満たすべき性質を問い合わせで記述する. 従来の OODBMS を対象とするアクティブデータベースでは, ルールは特定の OID もしくはクラス (スキーマ) に静的に束縛される. しかし Obase のようなインスタンスベースの OODBMS では監視の対象を動的に束縛する必要がある.

例えば, 従来の OODBMS ではクラスにルールを束縛することでインスタンスの一貫性制約を与える. しかし Obase においてはクラス定義にあたるスキーマを前提としないため, これができない. また, ある時点でのある属性を持つインスタンスに対してルールを静的に束縛しても不完全であり, この時点でルールを束縛されなかったインスタンスに後から属性が動的に追加された場合, このインスタンスにはルールが適応されない. そこで Obase ではルールは動的に束縛されるような形態をとった.

通常の問い合わせは条件を満たすオブジェクトを選び出すために利用されるが, ここではイベントを発生したオブジェクトが束縛条件を満たすかどうかを調べるために利用されるため, 逆の意味になる. このためこれを Inverse Query と呼ぶ.

Inverse Query の構文は Obase 検索文にしたがう. このため, Inverse Query には ObaseSQL の FROM 句と同じ構文や, ObaseSQL そのものを記述することができる. イベントを発生したオブジェクトが Inverse Query の表す集合に含まれていた場合, ルールはオブジェクトに束縛される.

3.2.2 イベントのマッピング

データベース更新操作では, まったく同じ動作を複数の構文で記述することができる場合がある. 例えば OID #1 のオブジェクトを OID #2 のサブオブジェクトとするという動作は以下の 2 通りの書き方ができる.

(1) addSubObject ({#1}) to ({#2})
(2) addSuperObject ({#2}) to ({#1})

これらの更新操作が行われるとき, 同じ動作にも関わらずそれぞれ異なるイベントが発生する. ObaseECA ルール機構は, どちらのイベントが発生した場合でも同じルールで対応できるよう, イベントのマッピングを行う. 例

の場合, 更新操作 addSuperObject を addSubObject の糖衣構文とみなし, 前者に対応するイベントは後者に対応したイベントに変換する. 一方, ObaseECA 記述言語ではルールの監視対象には糖衣構文を排したデータベース更新文を指定することで冗長にならないようにしている. したがって, addSuperObject を監視するルールは addSubObject を監視するルールとして記述する.

3.3 発火条件

発火条件には, Before, After に応じて事前チェック条件, 事後チェック条件が記述できる. また Separate の条件には, 事後処理が必要な場合の条件が記述できる. ObaseECA ルールの記述では, 発火条件 IF 句の構文は, ObaseSQL の WHERE 句の構文にしたがう.

3.4 アクション

ObaseECA モデルでは, ルールのアクションには更新操作系列, ユーザトランザクションアボートの指示, ユーザへの通知のいずれかを記述する. アクションの更新操作系列の記述には, 記述要素として Obase 更新文, Obase 検索文, 予約変数, オブジェクト変数が利用できる. また, アクションには代入文が記述できる.

Obase 言語では, ObaseSQL の中で OID 集合を参照するためにオブジェクト変数を定義しており, 例えば "select $x" のように使用する. ObaseECA で定義しているオブジェクト変数は, ObaseSQL と同じ構文を持ち, 同じアクションの中では ObaseSQL とスコープを共有する. しかし ObaseECA のオブジェクト変数は, Obase 更新文の結果を代入できる点で ObaseSQL のオブジェクト変数と異なる.

また, イベントを発生してルールに束縛されたオブジェクトを参照するために予約変数を用意している. 予約変数は selfObject, superObject, subObject の 3 種類あり, データベース更新操作に応じて異なる意味を持つ. オブジェクトの生成や削除, 属性操作などの更新操作によってイベントが発生した場合, selfObject に操作対象の OID が束縛される. オブジェクト間のスーパ/サブ関連の更新の場合は, それぞれスーパオブジェクトとサブオブジェクトの OID が superObject と subObject に束縛される. この予約変数のスコープはあるルールの記述, すなわち Inverse Query, 発火条件, アクションのそれぞれに共通である.

ObaseECA ルール記述言語が制御構文として提供しているのは, 発火条件とアクションから構成される条件構文のみである. したがって, 繰り返しなどの複雑な構文を持つ処理は現在のところ記述できない.

3.5 トランザクションの扱い

ここでは, ルールの評価実行に必要な処理系列から構成されるルール用のトランザクションとユーザトランザクションの関係について述べる. Obase では, ユーザからデータベース更新操作の系列を入力されるとユーザトランザクションを開始し, この中でそれぞれの更新操作を実行する (図 1 参照). 更新操作はそれぞれ, ユーザトランザクションを親とするサブトランザクション (図中の $S_1, S_2, ..., S_n$) の中で実行する. このときそれぞれの更新操作からイベントが発生するため, ユーザから与えられた更新操作に加えて, ルールの評価実行が行われる.

ルールの評価実行には, 以下の処理が必要である.

- ルールの束縛 (Inverse Query の評価)
- 発火条件の評価
- アクションの実行

あるイベントに対して複数のルールを処理する場合, ObaseECA ではそれぞれのルールについて, 束縛, 発火条件の評価とアクションの実行を一連の動作として繰り返す. ユーザトランザクションとその結果発火するイベントの評価実行トランザクションの関係は, 前述の 3 つのイベントが発生するタイミングによって異なり, それぞれにおけるトランザクションの関係とルールの役割は以下のようになる (図 2 参照).

Before: 更新操作を行うサブトランザクションの中で, ルールの処理も一緒に行う. このとき, まずルールの評価実行を行い, 次にイベントを発生したデータベース更新操作を実行する. このタイミングのイベントを監視することで, 例えば "ある条件を満たすオブジェクトは削除してはならない" などの制約を犯す更新操作の実行を未然に防ぐことができる.

After: 更新操作を行うサブトランザクションの中で, ルールの処理も一緒に行う. このとき, まずイベントを発生したデータベース更新操作を実行し, 次にルールの評価実行を行う. このタイミングのイベントを監視するルールでは, 例えばオブジェクトを削除する更新操作の結果が何らかの制約に違反したとき, 更新操作の実行を無効にすることができる.

Separate: ルールの評価実行は, ユーザトランザクションとは独立したトランザクションとして, ユーザトランザクションのコミット後に開始する. このため, Separate イベントの発生はユーザトランザクションのコミット時まで遅延される. このタイミングのイベントを監視することで, 例えばオブジェクトを削除した結果を受けて, 後処理を実行することができる.

ユーザから与えられた更新系列を処理するときは一般に, ユーザトランザクションの他に, Separate イベントによるルールトランザクションが複数生成される. またこのとき, 更新操作を実行するサブトランザクションの中では更新操作の前後に, Before イベントと After イベントによるアクションが入れ子型トランザクションとして実行される. このため, イベントの発生とルールの評価実行が再帰的に繰り返される. このように Obase システムで取り扱うトランザクションは入れ子構造を持つ.

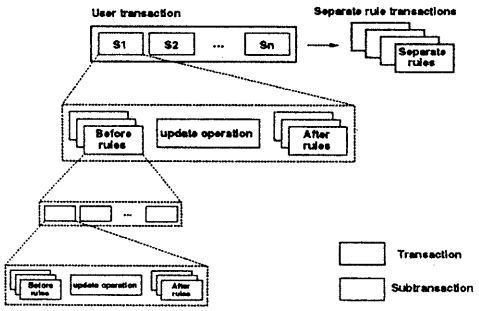

図 1: 一般的なトランザクションの構成

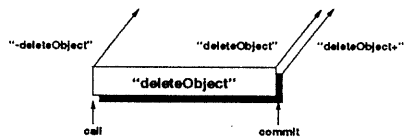

図 2: データベース更新文 "deleteObject" により発生するイベント

3.6 ECA ルールのオブジェクト化

OODBMS においてルールを実現する場合, ルールを通常のオブジェクトと同様に実現するかどうかで議論が分かれる. ECA ルールのオブジェクトを一般のオブジェクトと同様に実現した場合, 永続オブジェクトの管理やイベントの発生機構を通常のオブジェクトと共有することができる. また, 属性を使って検索の対象にできる. さらに Obase のような柔軟なオブジェクトを扱う OODBMS においては ECA ルールオブジェクトのスキーマを進化させたり, あるいは他の ECA ルールオブジェクトの属性やメソッドを継承させることができる. これを用いて, ひとつの ECA ルールを状況に対して動的に適応させながら利用することが考えられる.

ただし今回の実現では ECA ルール機構の実装と Obase システム本体の実装を並行して行ったため, ECA ルールオブジェクトは, 現状では Obase オブジェクトとは別種のオブジェクトとして実現している. また ECA ルールオブジェクトは, ファイルシステムを二次記憶として用いて管理している. 以上の理由から, ECA ルールの更新に対応したイベントは定義していない.

4 ECA ルールの例

以下では ECA ルールの有用性を示すために, ObaseECA モデルにもとづく ECA ルールの記述例を示す.

例 1: 従業員の給与に対する制約

データベースにおける一貫性制約の例として, 「従業員の給与額は, 自分の上司の給与額を越えてはならない」という制約を考える. この制約を表すルールは, ObaseECA では以下のように記述できる.

```
ON    update at employee/selfObject
IF    selfObject.manager.salary
      < selfObject.salary
THEN ABORT
```

このルールは, 従業員の集合を表すオブジェクト (*employee*) のサブオブジェクト, すなわち従業員を表すオブジェクト (selfObject) に対して行われる更新操作 (*update*) が終了した直後に発生するイベントを監視し, 更新を行うべきかをチェックする. 発火条件 (IF) における *manager* は上司集合を表すオブジェクトで, そのサブオブジェクトは上司オブジェクトのインスタンスを表す. 更新操作の結果, 上司の給与が従業員よりも低くなったとき, トランザクションをアボートする.

例 2: オブジェクトの削除に関する制約

Obase では, オブジェクト間にスーパ/サブの関係を与えることができる. ここで, 「サブオブジェクトを保持するオブジェクトは, 削除してはならない」との制約を設ける場合, この制約は以下のように記述できる.

```
ON      -deleteObject at
              ObjectWorld/selfObject
IF      exist (select $x from selfObject/$x)
THEN ABORT
```

このルールは, オブジェクトを削除する更新操作の呼びだし (*-deleteObject*) を監視し, 更新の前にチェックを行う. 発火条件 (IF) に現れる問い合わせ (select...) を評価し, サブオブジェクトを持つオブジェクトに対する削除操作であればアボートする.

例 3: トピックによる分類

Obase システムの応用例として, テレビのニュース番組の記事を分類するシステムを考える. このシステムでは, 動画映像, 音声, 原稿, 見出しなど, データベース化することを前提とせずに生成された膨大なマルチメディア情報から構成されるニュース記事が次々にデータベースに蓄積されていく. このときニュース記事が持つ情報をもとに自動的にグループに分類し, この結果を利用してユーザからの問い合わせに答える. 新しくニュース記事オブジェクトが生成されたとき, その記事をトピックにしたがって分類するルールは以下のように記述できる.

```
ON      addSubobject+ at news
IF      not exist(select  $x
                  from   group/$x
                  where $x.topic
                        = subObject.topic)
THEN $y ::= createSuperObjectOf subObject
              withName '%subObject.topic%';
        addSubObject ($y) to (group);
        update ($y).topic = '%subObject.topic%'
```

このルールは, ニュース記事の集合を表すオブジェクト *news* の "addSubObject" という更新操作のコミット直後に発生するイベントを監視し, その結果, 別の動作を起こす. 発火条件は, ニュース記事の分類を表すオブジェクト (*group*/$x) の中に, 新しく *news* に登録された記事オブジェクト (subObject) と同じトピック (*topic*) を持つものがなければ真となる. アクションは 3 つの更新操作から構成されている. まず, 新しく登録されたオブジェクト (subObject) のスーパーオブジェクトを, 生成 (*createSuperObject withName '%...%'*) する. このとき生成するオブジェクトの名前には, 登録されたオブジェクトのトピックを割り当てる. 次に, 新しく生成したスーパーオブジェクトを, 分類を表現するオブジェクト (*group*) のサブオブジェクトにする (*addSubObject*). 最後に, 新しく作られた「分類」オブジェクトの *topic* 属性を設定してアクションを終了する.

5 ObaseECA 機構の実現

ここでは Obase 全体のアーキテクチャについて述べ, ECA ルール機構の実現について詳しく述べる.

Obase システムはサーバ・クライアントアーキテクチャに基づくデータベースシステムである. Obase システムのクライアントは ECA 管理実行部, 言語処理部および外部アプリケーションから構成される (図 3 参照).

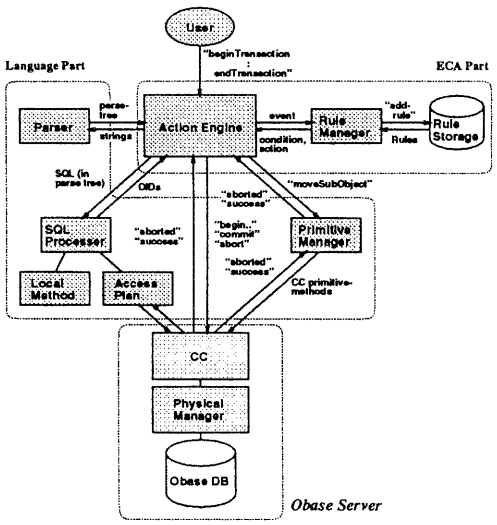

図 3: Obase システム構成図

ECA 管理実行部はアクション実行部 (Action Engine) と ECA 管理部 (Rule Manager) から構成され, ECA ルールの維持管理や評価実行, ユーザトランザクションの開始終了の管理などを行う. 言語処理部はパーザ (Parser), SQL 処理部 (SQL Processor), 更新文実行部 (Primitive Manager) から構成され, Obase 言語で記述されたユーザトランザクションの構文解析, および Obase 検索文の評価とデータベース更新文の実行を担当する. 一方, Obase

サーバ部は並行処理部 (Concurrency Control Manager) と物理層管理部 (Physical Manager) から構成される. サーバは複数のクライアントを相手に, 更新操作を管理する. Obase の並行処理部では, 制限された入れ子トランザクション機能を提供しており, ここではサブトランザクションの開始やコミットはチェックポイントの設定として機能する. つまりあるサブトランザクションをアボートすると, サブトランザクションを開始した時点の状態にロールバックできる.

実装には ParcPlace Systems 社の ObjectWorks Smalltalk を用いた. このため, ObjectWorks が動作する環境であれば, Obase システムおよび ObaseECA ルール機構を利用することができる. ここでは SPARC Station 10 を用いて実装した.

5.1 ECA ルール管理部

ECA ルール管理部ではルール作成用の GUI (Graphical User Interface) としてブラウザとエディタを提供している. ブラウザは登録してある ECA ルールを一覧するために利用する. またエディタ (図 4) を用いることで対話的にルールの作成, 変更ができる.

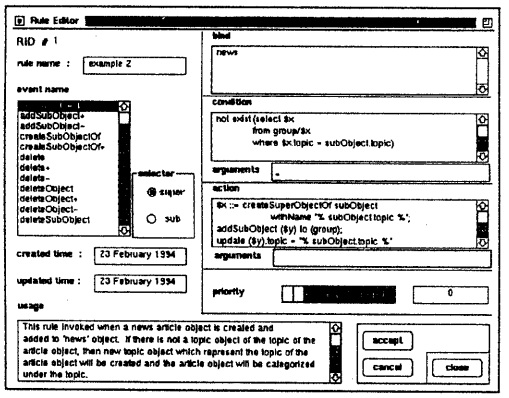

図 4: ECA ルールエディタの外観

5.2 アクション実行部

データベースの更新操作は, ルールの発火により再帰的に実行されるため, その実行環境をアクション実行部で管理する. ここでいう環境とは, 例えばルールを起動したトランザクションでのデータの値や, ルールの中で利用する変数, 特にイベントを発生したインスタンスへの参照などを指す.

アクション実行部では, 並行処理部が提供する入れ子トランザクション機能を用いて, 更新操作やルールの評価実行をサブトランザクションの中で処理する.

なお, Obase におけるデータベース更新文では, ひとつの更新文で複数のオブジェクトに対する更新操作を記述できる. アクション実行部では, 複数のオブジェクト (または多対多の組合せ) を対象とするデータベース更新文は, 単一のオブジェクト (または 1 対 1 の組合せ) を対象とする更新操作に展開して実行する. 例えば,

```
addSuperObject ({#1,#2}) to ({#3,#4})
```

は OID #1 で参照されるオブジェクトと OID #2 で参照されるオブジェクトを, それぞれ OID #3 で参照されるオブジェクトと OID #4 で参照されるオブジェクトのスーパオブジェクトにするという意味の Obase 更新文だが, これを,

```
addSuperObject ({#1}) to ({#3})
addSuperObject ({#1}) to ({#4})
addSuperObject ({#2}) to ({#3})
addSuperObject ({#2}) to ({#4})
```

という 4 つの更新操作に展開して実行する. それぞれの更新操作ごとにイベントが発生するが, これらはまったく同種のイベントであるため, ルールの発火処理をそれぞれのイベントごとに行ったのでは効率が悪い. この問題の扱いについては 6.1 節で述べる.

5.3 ObaseECA 機構の動作

5.3.1 ユーザトランザクションの開始

ユーザからの命令列は, "beginTransaction;" と "endTransaction;" で囲まれた文字列としてアクション実行部に入力される. アクション実行部はユーザインタフェースから文字列を受けとるとこれをパーザに渡し, 返値として中間表現 (パーズツリー) を受けとる.

ルールのアクションを記述した文字列を言語部のパーザに送り, その結果として中間表現を受けとる. アクションの実行は, この中間表現をもとに行う.

アクション実行部はユーザから与えられた命令列を処理するために, 並行処理部のトランザクション処理部に要求を出し, 新しくトランザクションを開始する.

5.3.2 更新操作の処理

更新操作の命令列を逐次実行する際, アクション実行部は以下の 1 から 8 の処理を繰り返す.

1. 更新文に現れる ObaseSQL を評価し, OID 集合を受けとる.
2. 複数のオブジェクトを対象とするデータベース更新操作の場合, 単一のオブジェクトを対象とするデータベース更新操作の系列に展開し, 以下の処理を繰り返す.
3. アクション実行部は Before イベントを発生し, ECA ルール管理部 に渡す.
4. Before イベントに対応したルールを受けとり評価実行する.
5. 更新文を更新文実行部に渡して実行し, 返り値 (OID) を受けとる.
6. 代入文の場合, 変数テーブルに返り値を登録する.
7. アクション実行部は After イベントを発生し, ルール管理部に渡す.
8. After イベントに対応したルールを受けとり評価実行する.

5.3.3 サブトランザクションのアボート時の対応

ObaseSQL の評価, または更新文の実行用のサブトランザクションが並行処理部のデッドロック回避などの事由によりアボートする場合がある. このとき並行処理部からは, デッドロックがサブトランザクションだけの問題か, それともユーザトランザクションのレベルからアボートしなければ解決できないかが伝えられる. サブトランザクションレベルだけの問題でアボートしたのであれば, アクション実行部は再試行する. ユーザトランザクションレベルからデッドロックを生じていた場合, ユーザトランザクションをアボートし, 異常終了する.

5.3.4 イベントの管理とルールの評価

イベントの発生はアクション実行部で管理する. イベント待ち行列の管理, および発火したイベントの処理もアクション実行部で管理する.

アクション実行部は更新文のサブトランザクションを開始する直前に Before イベントを, 終了 (コミット) 直前に After イベントを発生する. サブトランザクションのコミット直後には Separate イベントを生成するが, このイベントは一旦, 待ち行列に保存される. 待ち行列中の Separate イベントは, ユーザトランザクションのコミット後に逐次処理する. イベント発生時の処理手順は以下の通りである.

1. ルールの抽出
   ルール管理部はアクション実行部から発生したイベントを受けとると, そのイベントに対応するルールを抽出し, リストを生成する. リスト中の各ルールについて Inverse Query を評価する. イベントを発生したオブジェクトに束縛されているルールを抽出し, 各ルールの <Condition, Action> の対のリストを生成する. 生成したリストをアクション実行部に返す.

2. アボートルールの処理
   Before イベント, After イベントの処理の場合, アクション実行部は受けとったリスト中のルールからアクションの内容がイベントを発生した更新操作を強制中断するもの (アボートルール) を選び出す. それぞれの発火条件を評価し, 条件が満たされていた場合にはユーザトランザクションの更新操作を中断する.

3. ルールの評価実行
   アクション実行部は発火ルールリスト (この時点まで残されたリスト中にアボートルールは存在しない) の各ルールについて, 発火条件の評価と, 条件が満たされていた場合にはアクションの実行を, 交互に繰り返す. このとき, アクション実行の手順はユーザトランザクションの実行手順と基本的に同じである. ただし, Before, After イベントのルールは, ユーザトランザクションから派生するサブトランザクションの中で実行する. Separate イベントのルールは, ユーザトランザクションとは独立した新たなトランザクションの中で実行する.

6 残された課題

OBaseECA 機構はこれまでの関係データベースにおける ECA 機構とは異なり, インスタンスベースの OODB のためのルール機構を与える. そのため, 実現において多くの新しい課題が持ち上がった. ここでは, そのうちいくつかについて述べる.

6.1 Inverse Query の最適化

ObaseECA ルール機構においては, 束縛対象の記述と発火条件の記述には, それぞれ ObaseSQL を使用することができる. 例えば,

```
ON update at employee/selfObject
IF  selfObject.salary > 50
```

などである. このルールは *employee* オブジェクトの任意のサブオブジェクトに対して行われる, *update* というデータベース更新操作が発生するイベントを監視しており, サブオブジェクトの *salary* 属性の値が 50 を超えたときに発火する.

イベントの評価である Inverse Query を問い合わせとして実行すると, イベントの発生ごとに問い合わせを行わなければならず, 効率が悪い. たとえば, Obase では, 複数の OID が指定されたデータベース更新文は単一の OID を引数とする複数のデータベース更新文系列に展開され, 実行される. このためそれぞれの OID ごとに同じイベントが繰り返し発生され, まったく同じ問い合わせが繰り返し評価されてしまう.

このような場合, ひとまとまりの更新で発生したイベントは順序関係がないため, 評価せずにためておき, 後でまとめて問い合わせを行うことによって, 効率良くイベントを評価することができる.

Inverse Query とルールの発火条件の問い合わせから, ひとつの問い合わせを合成することができる. 例では, Inverse Query と発火条件 "selfObject.salary > 50" を,

```
select selfObject
from  employee/$x
whereselfObject.salary > 50
```

というひとつの ObaseSQL に合成できる. その結果, Inverse Query の評価を通常の ObaseSQL の評価と同様の手法で最適化できる. さらに, このようにしてできたいくつかの問い合わせどうしを組合せ, より一般的な問い合わせにすることにより, イベント監視を最適化できる可能性がある.

6.2 イベントの階層関係

Obase 更新文によって生じるイベントを検討すると, これらには is-a 関係や, part-of 関係などの関連が存在する. 例えば moveSubObject イベントは, deleteSubObject イベントと addSubObject イベントの複合イベントである. このようなイベントの階層関係については [3] に指摘されており, これを導入することで ECA ルールの記述能力を高めることができ, より幅広いアプリケーションに対応することが可能となる. このため, この問題は今後の検討を必要とする.

6.3 ルール実行の停止性の問題

ルールの再帰的な実行により, 無限ループに陥ってしまう可能性がある. 単純な例として,

```
ON    update at ({#1})
IF    exist (select ObjectWorld/$x)
THEN update ({#1})
```

が考えられる. このルールは OID #1 のオブジェクトに *update* イベントが発生したとき, OID #1 のオブジェクトに無条件でデータベース更新文 update を実行するという意味になる. ここで, 更新文 update は再び *update* というイベントを発生するため, ルールの再帰実行は無限に続いてしまう. ルールの実行が無限ループに陥らないための条件として, 再帰呼び出しを表すグラフが非巡回有向グラフを構成することが必要である. このため, 登録時にルールを解析し, 停止性や無矛盾性を調べる機構が必要である. しかし, ObaseECA ルール機構では, ルールの監視対象を動的に束縛するため, ルールのチェックをさらに難しくしている.

6.4 OID 集合のセマンティクス

Obase 言語の更新文では引数に検索文などを指定できるため, 更新文の引数に複数の OID が束縛される場合がある. また, ECA ルールにおいても束縛対象 (監視対象) に検索文が指定できるため, イベントが発生した際, ひとつのルールに複数のオブジェクトを束縛することが可能である. ここで問題になるのが, 引数に与えられた複数 OID の解釈である. このモデルによって, システムの動作が大きく異なる.

Obase では, 複数の OID に対して更新操作を行う場合, それぞれの OID に対する更新操作をトランザクションと考え, 処理する. 通常, 複数のトランザクションを実行する場合, それぞれのトランザクションは独立であり, あるトランザクションの実行は他のトランザクションのコミットやアボートには影響されない. しかし, 複数のトランザクションに相互依存性を持たせる必要が生じる場合がある.

例えば団体旅行の宿泊施設の予約を考える. 更新操作の対象は, 客を表すオブジェクトの集合とする. あるホ

テルに団体客の予約を入れたとき, このホテルには利用資格があって, 客のうち何人かは予約を断られたとする. このとき, これは団体旅行の予約なので, 一部の客についてだけ予約が行われることがあってはならない. このようなシステムを実現する場合, それぞれの旅行客の処理を行うトランザクションには相互に依存関係があり, たとえあるトランザクションの処理に成功しても, もし他に一つでもアボートしたトランザクションがあれば, 関係するすべてのトランザクションをアボートする必要がある.

このように, 複数 の OID のセマンティクスの解釈には, 以下の 2 通りが考えられる.

(解釈 1) 互いに独立したオブジェクトにたまたま同じ操作をするために, それぞれの OID を一箇所にまとめて記述しているだけ.

(解釈 2) 複数の OID はある集合を表している. つまり OID が示すオブジェクトは, なんらかの意味を持つ集合の要素であり, 互いに依存している.

解釈 1 は非常に一般的かつ単純な解釈であり, 曖昧さを持たない. 一方の解釈 2 は曖昧な部分が多いが, 一般的な解釈を拡張しており, ここであげた団体旅行の予約や, ニュース分類システムなど今後の応用システムを考える上で重要となる.

7 おわりに

本稿では柔軟なオブジェクトを扱う OODBMS に対応した ECA ルール機構について議論し, インスタンスベースの OODBMS のプロトタイプ, Obase の一部として ECA ルール機構を実現した.

問い合わせを用いて ECA ルールやメソッドの束縛を動的に行うことなどを提案したが, そのために新しい課題が生まれており, そのいくつかについては今後も引続き検討が必要である.

データベース化することを前提とせずに生成された膨大なデータを管理できるデータベースシステムの必要性は, 今後もいっそう高まるものと思われる. このようなデータベースシステムにおいて, データベース管理を自動化する ECA ルール機構の用途は多様化するものと思われる. しかし現時点では柔軟なオブジェクトを扱う OODBMS に関して解決すべき課題も多く, これらのシステムに対応した ECA ルール機構に関する問題と並行して議論を進める必要がある. 特に Inverse Query の最適化, イベントの階層化, ECA ルールの停止性, OID 集合のセマンティクスの各問題は, Obase システムの実用性や, ObaseECA ルール機構の応用性を高める上で, 特に重要である.

参考文献

[1] Stanley B. Zdonik, "Incremental Database Systems: Databases from the Ground Up", *Proceedings of ACM SIGMOD '93*, pp.408–412, 1993.

[2] S. Chakravarthy, et al, "HiPAC: A Research Project in Active, Time-Constrained Database Management — Final Technical Report", *Technical Report XAIT-89-02, XAIT Reference Number 187*, July 1989.

[3] J. ランボー, M. プラハ, W. プレメラニ, F. エディ, W. ローレンセン 著, 羽生田栄一 監訳, "オブジェクト指向方法論 OMT", プレンティスホール・トッパン刊, 1992.

[4] 千里国際情報事業財団 Obase コンソーシアム, "Obase システムの概念設計", *Obase* プロジェクト第二期 研究成果報告書, pp. 163–272, 1992.

[5] 田中 克己, 西尾 章治郎, 吉川 正俊, 下條 真司, 森下 淳也, 上善 恒雄, "Obase オブジェクトデータベースモデル", 日本ソフトウェア科学会 第 9 回オブジェクト指向計算ワークショップ, 1993.

[6] 吉川 正俊, 田中 克己, 上善 恒雄, 田中 康暁, 蛭井 潤, 堀田 光治郎, "ObaseSQL: 拡張経路式と継承演算子を持つオブジェクトベース言語", *Proceedings of Advanced Database System Symposium '93*, pp.63–71, Tokyo, Japan, December 2-3, 1993.

[7] 吉川 正俊, "構造検索機能と継承演算子を持つオブジェクトベース代数", 情報処理学会データベースシステム研究会研究報告, vol. 93–DBS–95, September 1993.

[8] 田島 孝一, "柔軟なオブジェクトを扱う OODBMS のための ECA ルール機構の実現", 平成 5 年度 大阪大学大学院 基礎工学研究科 物理系専攻 修士学位論文, February 1994.