データベースシステム 93－3
システムソフトウェアと
オペレーティング・システム 59－3
（1993．5．27）

プロセス代数によるアクティブデータベースの扱いについて

小島 功　　磯部 祥尚　　大蒔 和仁

電総研情報ベース研究室

{kojima,isobe,ohmaki}@etl.go.jp

要 旨

本稿では、アクティブデータベースの諸問題に対する、プロセス代数の有効性について議論する。アクティブデータベースでは、イベントや更新の波及など、システムの動的な挙動などの扱いが不可欠である。また、ルール／トランザクション処理の一貫性の検証や、システムによって異なる機能の比較などのできるモデル化の道具が必要である。プロセス代数は並行処理の扱いや様々なクラスの等価性判定が可能と言った性質があり、処理の記述の検証に有効と考えられる。また、これに基づいた記述はプログラミング言語として実行可能であるので、モデルの記述と実行のメカニズムを同じ記述言語を用いて扱うことができるといった特徴がある。

キーワード：アクティブデータベース、プロセス代数、等価性、一貫性検証

Description of Active Database Systems Using Process Algebras

Isao KOJIMA　　Yoshinao ISOBE　　Kazuhito OHMAKI

Information Base Section, Electrotechincal Laboratory

Abstract

This paper presents the approach for active databases using process algebras. Data models for active databases are required to support dynamic aspects of databases such as events and update propagations. Process algebras like CCS are suitable for these application. Several equalities defined on the algrebra can be used for the consistensy checking on active databases. Also, process algebra will be useful as programming language for describing the mechanism of active databases.

Keywords: Active databases, Process Algebra, Equality/Confluence, Consistency checking

1 まえがき

アクティブデータベース [2, 3] は、データベース様々な状況をモニタし、それに従って動的にデータベース処理を実行することのできるＤＢＭＳであり、近年のデータベース応用の高度化に伴って注目されている研究分野である。このような機能はエキスパートシステムやルールベースなどを含む様々な分野で扱われているあるが、アクティブデータベースでは頻繁な更新やそれに伴うトランザクション処理の機能などに特徴がある。

アクティブデータベースを使うことで、大規模なデータベース上でのルール処理や、高度の自動化、リアルタイム性の高い処理といった応用が支援でき、その重要性が高いと考えられている。また、従来のＤＢＭＳで支援されてきた一貫性の保持やビューの計算などの機能についても、より統合的なアーキテクチャを提供できるので、様々な実験システムが研究開発されている。

このようなアクティブデータベースではイベントやトリガなどデータベースの動的な側面を扱うことが不可欠であり、次のような問題を始めとする従来のシステムで扱われていなかった様々な課題 [1] がある。

- 利用者の登録したルールなどが、データベース上で問題なく動くかかどうか検証する必要がある [4]。例えば、正しく動くための基準として、

  1. ルール集合が停止するかどうかの判定
  2. 最終的な状態が一意かどうかの判定
  3. 観測される動作が一意かどうかの判定

  といった判定を行なわなければならない。このような検証を容易にするための形式的な手法が必要である。また、ルールの記述言語、モデルはこのような手法に適した形式である方が望ましい。

- アクティブデータベースでは、ボトムアップ的に実現された機能も多く、システムによってその記述力などが大きく異なる。このため、様々なシステムの比較基準となるような参照モデルとしての、形式的な枠組みが必要となってきている [5]。

本稿では、このようなアクティブデータベースの問題を扱うための形式的な枠組みとして、プロセス代数に基づいた記述を用いることを提案する。ルールの記述やトランザクション処理の記述に用いられ、実行のメカニズムを提供すると共に、その正当性、一貫性の検証の道具としても用いられる点が特徴である。2章でアクティブデータベースの簡単な紹介とその問題点、3章でプロセス代数アプローチの基本的な考え方を示す。4章でそれぞれの実例を示しその有用性を議論し、5章で言語の方向を示す。

2 アクティブデータベースシステムの研究と課題

2.1 アクティブデータベースの概要

アクティブデータベースとは、一般にはシステムに対する様々な状況の変化をイベントとして、それに対応する動作をトリガするような機能を持つデータベースシステムである。システムは状況をモニタする機 能と対応する動作の呼び出し機構を持っている。処理の記述は図1に示すように、ECA アーキテクチャとして知られる、

1. あるルールの評価を行なうタイミングを決定するイベント event
2. ルールのうち評価すべき部分である条件 condition
3. 評価が満たされた場合に実行される動作 action

の要素から構成されており、次のような実験システムを始めとする様々な研究が行なわれている。

- **HiPAC :** HiPAC は Event-Condition-Action の３要素や、カップリングモードを明示的に記述する点に特徴があり、多様な種類の操作が可能になっている。ルール自身はデータベースの対象として操作が可能であり、タイプ付けされている [2, 3]。

- **Starburst Rule System :** when(event)-if(condition)-then(action) タイプのルールを登録することができる。ルール自身はオブジェクトではない。ルールの precedes, follows 句によって、並行処理可能なルールの実行順序を明示的に指定できる点に特徴がある [6, 10]。

- **Postgres Rule System :** Postgres では PostQuel 文による条件記述と、それを満足することによる PostQuel 文の起動によるとい

1)

```
Event :   update   exchange-rate
Condition:     new value != old value
Action :     display new-value
```

2)

```
Event : update exhange-rate
Condition:  currency ="$" &
                new value  < 116
Action :    buy $10000  from  account
```

3)

```
Event:    buy $
Condition :   balance < 0
Action :    if balance < 0 then report warning
```

図 1: アクティブデータベースの処理記述

う、C-A ルール的な処理系 [8] と、その実現の経験から、On 文のように明示的にイベントを記述できるよう機能が改良された処理系 [9] がある。

- **Ariel** : Starburst 同様、実行の priority の指定ができるルール記述 ARL を持つ [13]。データの挿入の波及的な影響は、更新のタイプによって定まっている。
- **Ode** : C++ ベースのデータベースプログラミング言語 O++ にルール処理を加えたもの。トランザクション、トリガ、イベントの概念を持つ [16, 17]。
- **大規模なプロダクションルールシステム** : KEEなど、プロダクションルールシステムをそのまま 2 次記憶上のデータを支援するように拡張したものが知られている。多くは、述語左辺のマッチングによる右辺の実行を行なうようなものであり、述語はデータベースに格納されている [11]。システムの性格上 ECA アーキテクチャはあまり意識されていない。

他にも Exodus Rule System[12], Alert システム [14]、ADAM システム [15] などが知られている。

2.2 アクティブデータベースへの要求

これらの研究における、さまざまな問題点をまとめると、次のようになる [1]。

- **データモデルに対する要求**

  アクティブデータベースのためのデータモデル／記述言語には、次のような要求が考えられる。

  1. イベントの扱い : [17, 21]
     ルールを起動するイベントの発生とその処理を記述できる必要がある。イベントは従来のデータベースではあまり扱われなかった概念であり、これは関係モデル、オブジェクト指向といったデータモデルの種類に関わらず同様の問題となっている。
  2. 更新、状態遷移の扱い :
     一般に更新によるデータベースの状態変化の扱いは、重要な研究の対象として知られており、多くの研究が行なわれている [18, 19, 20]。アクティブデータベースでは、ルール処理による更新とそれに伴うデータベースの状態変化の波及的な扱いが不可欠であり、このような機能と統合できる記述言語／モデルである必要がある。

- **システム・アーキテクチャに対する要求**

  アクティブデータベースシステムは、波及する更新を含んだデータベース処理や実行の各段階でイベントの評価を行なうために、従来のシステムとは異なった次のような要求がある [4, 13, 23]。

  1. 実行モデルの支援 :
     イベントの検出やルールの評価の手順といった、従来のＤＢＭＳで支援されていない実行メカニズムの効率的な実現の必要がある。
  2. スケジューリング :
     集合的なルール処理は高機能なトランザクション処理でもあるので、並列性の高いデータベース処理を実現するためのスケジューリング手法が必要である。
  3. 正当性の検証 :
     利用者が自由にルールを記述できるので、トランザクションの実行が正しい結果を導くかどうかの検証可能なメカニズムが必要である。

3 アクティブデータベースへのプロセス代数的アプローチ

CCS、π -calculus といったプロセス代数 [24, 25] は、並行プロセスの動作の記述などのために用いられる記述体系であり、アクティブデータベースの記述の道具として考えると、次のような特徴がある。

1. **並列性や状態遷移を自然に扱える。**

   例えば更新による状態遷移は論理を用いても記述できるが、プロセス代数によるアプローチは、論理によるものに比べて、状態の遷移を素直に扱えるといった特徴がある。また、ここで示したイベントの扱いにも適合しやすいと考えられる。

2. **様々なクラスの等価性の判定、決定性の判定ができる。**

   利用者が個々のルールを登録、プログラムしていく過程では、一貫性の破壊や矛盾する操作、停止しない操作などが生じうる。実際にはこれらはトランザクション処理同様の直列可能性などの概念によって検出しているが、プロセス代数は先に述べたような性質の検査の判定を自然に行なえることができる。

3. **プログラミングの道具でもある。**

   グラフや状態遷移図といった従来の直列可能性判定のための道具と異なり、代数は計算能力が高いので、これ自身を用いてルール自身やそれを処理するスケジューラの処理を記述することができる点は大きな特徴である。一方、この記述の検証も可能である。

ここではまずＣＣＳを例として、以上のような性質を確認する。

3.1 CCS とその性質

CCS[24, 1] は、図 2のような構文と意味を持っている。

さて、この上でいくつかの等価変換のための規則が存在しており、これを使った様々な等価性がいくつか示されている。例えば、図 3のような、生産者／消費者の問題を状態遷移図による仕様とペトリネットによる実現を考えるとする。それぞれに従ったこの２つの CCS 記述は代数的に等価であることが示されている。

呼び方	構文
inaction	**0**
prefix	***a.E***
summation	***E1 + E2***
composition	***E1 | E2***
restriction (L	***E \ L***
relabelling	***E[f]***

名前	導出規則
Act	$\vdash \alpha.E - \alpha \rightarrow E$
Sum	(i) $E_1 - \alpha \rightarrow E_1' \vdash E_1 + E_2 - \alpha \rightarrow E_1' + E_2$ (ii) $E_2 - \alpha \rightarrow E_2' \vdash E_1 + E_2 - \alpha \rightarrow E_1 + E_2'$
Comm	(i) $E_1 - \alpha \rightarrow E_1' \vdash E_1 \mid E_2 - \alpha \rightarrow E_1' \mid E_2$ (ii) $E_2 - \alpha \rightarrow E_2' \vdash E_1 \mid E_2 - \alpha \rightarrow E_1 \mid E_2'$ (iii) $E_1 - l \rightarrow E_1' \wedge E_2 - \bar{l} \rightarrow E_2'$ $\vdash E_1 \mid E_2 - \tau \rightarrow E_1' \mid E_2'$
Res	$E - \alpha \rightarrow E' \wedge (\alpha, \bar{\alpha} \notin L)$ $\vdash E \setminus L - \alpha \rightarrow E' \setminus L$
Rel	$E - \alpha \rightarrow E' \vdash E[f] - f(\alpha) \rightarrow E'[f]$
Con	$P - \alpha \rightarrow P' \wedge (A \stackrel{\text{def}}{=} P) \vdash A - \alpha \rightarrow A'$

図 2: CCS の構文と意味

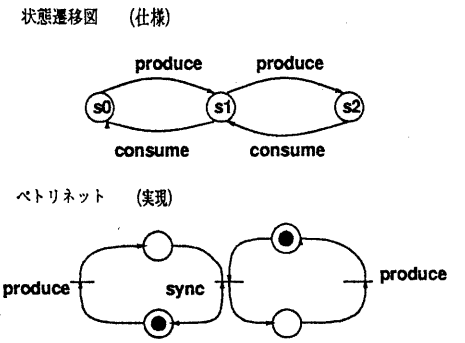

図 3: 生産者、消費者の問題

(1)

$$S0 \stackrel{\text{def}}{=} a.S1$$
$$S1 \stackrel{\text{def}}{=} a.S2 + \overline{b}.S0$$
$$S2 \stackrel{\text{def}}{=} \overline{b}.S2$$

(2)

$$C \stackrel{\text{def}}{=} a.C'$$
$$C' \stackrel{\text{def}}{=} \overline{b}.C$$

$$PC\text{-}SYSTEM \stackrel{\text{def}}{=} (C[sync/b]|C[sync/a])\backslash sync$$
$$PC - SYSTEM[produce/a, consume/b]$$

4 CCSによる、アクティブデータベースの扱い

さて、このような記述を使って、1）プロトコルとしてのアクティブデータベース、2）スケジューラとしてのアクティブデータベースの、それぞれの記述にこのようなプロセス代数が適用できるかどうか考える。

つまり、1）個々のルールを記述した場合に、それらの組合せや接続によって一貫した動作が規定できるかどうかと、2）システムの動作を記述してスケジューラとして正しい動作を実現することが可能かどうか、について考察する。CCS自身で処理の全てを記述すると少し冗長なので、以降の例は完全な記述となっていない。

4.1 カップリング・モードの扱い

プロトコル的な表現の例として、ルール処理のカップリングモードの記述を行なう。カップリングモードとは、ECAルールのEvent,Condition,Action各部分の評価とその順序関係を2つのトランザクションの接続関係として与えるものである。

例えば条件の評価と続く動作の実行との関係を2つのトランザクションの関係と考え、この組み合わせ方にいくつかの選択枝を与えて利用者が指定できるようにし、順序関係を規定している。HiPACのカップリング・モード[3]では、次のような種類が提案されている。

1. **immediate** 呼びだし先のトランザクションをサブトランザクションとして親トランザクションを中断して先に行なう。
2. **deffered** 親トランザクションのコミット寸前にサブトランザクションを実行する。

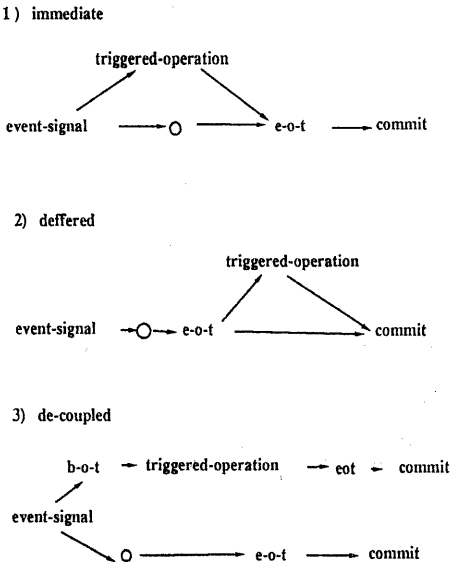

図 4: カップリング・モード

3. **de-coupled** 呼び出すトランザクションを全く別のトランザクションとして実行する。

これにより、例えば更新に伴う値の変更関係とルール処理の順序その他を指定することが可能となる。図で表すとトランザクションの接合関係は図4のようになる。

さて、これをCCS的な代数表現で書くと、図5（接続関係以外は適当な簡略化／省略化を含む）のようになる。

実際のルールをこのようなエージェントを用いて記述すると、動作可能な式が得られる。さて、カップリングモードは接合したトランザクションの間で一種の順序関係を規定するが、CCSの記述でもこれは保存されていると考えられる。

また、2 (immediate) や3 (deffered) によって結合されるエージェントの場合次のような性質がある。

1. 内部的な同期に用いられている commit が制約 (restrict) されていて、外部から操作できない。このことは、commitをはさんだルール処理における中間状態が他から見えないことを示している。ルール処理における中間状態をどう扱うかについては、いくつかのアプローチがある[5]が、このような制約を使うことで隠蔽したり、制御できる。

```
Agent:
   A0 def= event. A1
   A1 def= operation. ok . commit.A0 + op.nok.abort.A0 ...
   .....

Controller:
   1) de-coupled
      Cdc def= A [update/event, op1, ..... ] | A[update/event.op2...
           .....
   2) immediate
      Cim def= (A[commit1/event,op1,...] | A[update/event,op2,.
               ...., commit1 ]) \ {commit1,commit2} ...
   3) deffered
      Cdf def= ( A[update/event,  op1, ..., commit1]
             |  A[commit1/event, op2, ..., commit2])
             \ {commit1,commit2}
      .....
```

図 5: カップリング・モードの省略化した記述の一部

2. 同じくこの制約により、規則に従って展開しても同期の関係は保存されている。これは、操作をどう実行しても一貫性が保たれることと対応している。

従って、この上で適切な変換を行なえば、一貫性を保持した実行スケジュールを出力することも可能である。

4.2 スケジューラ

一般にアクティブデータベースのシステムは、図6のようなアーキテクチャになっている。例えば、図の構成要素間での処理は、次のように行なわれている。

1. イベントがEvent Detectorにより検知される。外部イベントでなく、データベース処理によるイベントなら、その操作は一時的に待たされる。

2. Rule Manager がそのイベントによって起動されるべきルールを決める。各ルールに対して条件の評価と実行に必要なデータを Object Manager から呼びだす。

3. 例えば condition の評価が immediate モードを持つ ルールに対しては、Rule Manager は Transaction Managerに対して condition 評価のためのサブトランザクションの生成を呼びだし、Condition Evaluator に評価のためのイベントシグナルを送る。

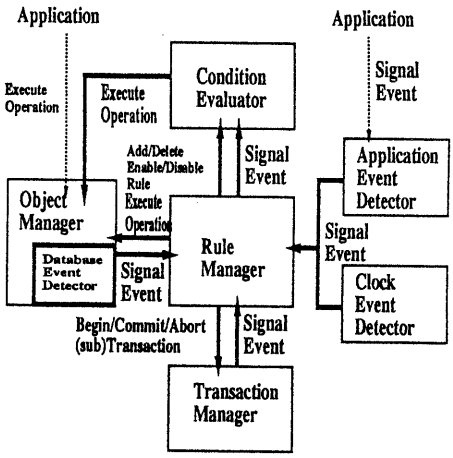

図 6: アクティブデータベースの処理

4. Condition Evaluator は評価された条件を満足したルールのリストを Rule Manager に戻す。 そのためにサブトランザクションの一環として Object Manager に対してデータベース問い合わせを行ない、評価結果を Rule Manager に戻した後サブトランザクションをコミットする。

5. Rule Manager は条件が満たされた immediate ルール集合のうち、実行すべきものを決め、サブトランザクションを生成した後、Action を呼び出す。

さて、このような Rule Manager,Condition Evaluator などの動作を記述できれば、それは実行可能なシステムの記述とできる。一部の例を図7に示す。先の場合と異なり、ルールを処理するモジュールをエージェントとし、その間でイベントが伝達されている。また、ルール処理の非決定的な並行実行も記述できる。

ここで、先に書いたような個々のルールとその接続関係の記述と、ここでのスケジューラの視点からの記述が、等価であることを判定できれば、システムとしての検証に使うことができると期待できる。

(Transaction Manager)

$TM \stackrel{def}{=}$ event(list).TM'

$TM' \stackrel{def}{=}$ $\overline{commit}$(list).TM+$\overline{abort}$(list)TM

(Condition Evaluator) 同じ形式とする

(Rule Manager)

$RM \stackrel{def}{=}$(TM[trig/commit,] | TM [trig/event])\ trig

アボートの分を省く

RM [transaction-signal/event, condition-signal/commit]

図 7: スケジューラの要素の接続の記述の一部

4.3 正当性のための基準

アクティブデータベースでは、ルール処理の実行については、影響のある結果については決定的で、かつ一貫であることが必要である。このための条件の一つとして confluence がある [4]。これはルール集合が停止するという条件の元で状態が一意であるための条件を示している。

さて、CCS においても同じ用語を使う概念として合流性 confluence が存在する。定義は省くが、この性質について、次のようなことが知られている。

- 合流性を保存する結合子が定義できる。つまり、ある種類の結合子のみを使って定義されるエージェントは、必ず合流的であると証明されている。
- 合流的であるシステムの等価性はトレース等価と呼ばれる条件を満たすことによって証明できる。つまり、等価性の判定条件がより単純になっている。

さて、あるアクティブデータベース [4] における confluence の判定は、ルールの実行によって状態が変化するような状態遷移グラフを考え、その状態が同じになることを検出している。このような遷移関係ははプロセス代数の confluence を使って容易に記述できる。また、ルール実行にともなう状態遷移も前節までに述べたように扱うことができる。従って、

- 処理の記述に代数的表現をそのまま用いることができるので、判定が容易になる。
- 予め知られた合流を満たすための結合子を含んだ記述言語が定義可能であり、より安全な記述体系を提供できる。

といった性質が期待できる。

```
specification PC-SYSTEM[produce,consume]:    noexit
  behavior
    hide sync in
      Cell[produce,sync]|[sync]|Cell[sync,consume]
      where
      process Cell[input,output]: noexit:=
              input;output;Cell[input,output]
      endproc
endspec
```

図 8: LOTOS による問題のペトリネット記述

5 アクティブデータベースのための言語

先に述べた場合は、基本的な代数のメカニズムとしては利用／検証可能であるが、実際に応用を記述する場合にはデータ型の能力やプログラミングの能力の点では問題がある。そこで、プロセス代数に基づいた、アクティブデータベースの記述言語の可能性について考える。

ここでは同様のアプローチの例としてプロセス代数に基づいた言語 LOTOS[26] を示す。これはプロトコル記述用の仕様記述言語であり、次のような特色がある。

- 抽象データ型記述を持つ。
- CCS に基づいた並行プロセス記述を持つ

つまり、記述対象のうち、静的な性質を持つ部分については、抽象データ型的な記述を利用でき、動的な性質を持つ部分については、並行プロセス記述を利用できる。また、プログラム言語でもあるので、様々なスタイルで実行可能な処理を記述しつつ、検証できると言う利点がある。例えば、先の producer/consumer の問題に関して、ペトリネットと状態遷移図の記述をそれぞれ図 8, および図 9に書くことができる。

ここで、同じ問題に対して 2 つの仕様に基づいた記述の等価性を判定できることを利用して、プロトコルの仕様と実現の等価性を判定できる。先に CCS を使って記述したアクティブデータベースのルール記述であるモデルと、その実行処理方式であるトランザクション処理のメカニズム／スケジューラについても、それぞれの検証が可能であることが期待できる。

一方、抽象データ型を持つプログラミング言語であることは、このような言語を使って、利用者がそのまま処理の記述、実行できる道具でもあるので、検証、実行のために特別な道具を必要としない。また、

```
specification PC-SYSTEM[produce,consume]:  noexit
  type State is boolean
    sort State
    op   s0,s1,s2;-> State
    op -eq-:State ,State -> Bool
    eqns   s0 eq s0 = true;  s0 eq s1 = false;
                    *
  endtype
  behavior   FSM[produce,consume](s0)
    where
      process FSM[produce,consume](s:State):  noexit:=
        [s eq s0] -> produce; FSM[produce,consume](s1)
        [] [s eq s1] -> consume; FSM[produce,consume](s0)
         [] [produce;FSM[produce, consume](s2)
        [][s eq s2] -> consume;FSM [produce,consume](s1)

       endproc
  endspec
```

図 9: LOTOS による問題の状態遷移図記述

型表現を利用したデータベースモデルとしての利用も可能である。つまり、CCS などのプロセス代数に基づいたアクティブデータベースの記述言語は、記述性と検証性の 2 点で有益であることが分かる。

また、このような言語の提供するモデルを一般のデータモデルと比較すると、従来のデータモデルの手法は型の定義とその階層に重きをおいていると考えることができる。オブジェクト指向データベースでも、メソッドプログラムなどのふるまいのモデル化の試みが行なわれているが、記述、検証の道具としては有効なものが知られていない。プロセス代数による枠組みはこの問題に対して有効であり、このような性質を持つデータベースモデルは有益であると考えられる。

6 あとがき

ここでは、アクティブデータベースの問題点について、プロセス代数によるアプローチの特徴をまとめた。プロセス代数に基づいたプログラム言語やデータベースプログラミング言語を用いれば、アクティブデータベースのメカニズムの記述はそのまま実行可能な処理系である点は他のアプローチに比べて大きな特徴になる。また、同時にここで述べたような等価性判定などに基づく検証を行なえるので、このようなアプローチは有益であり、ここで述べた性質を満たす検証手続きの確立と記述言語が必要である。

謝辞 日頃熱心に御討論頂く研究室の皆様に深謝いたします。

参考文献

[1] , 大蒔他: "ブルーバブル情報ベースシステムに関する調査報告", 電総研調査報告第２２３号、 1993.03.

[2] U.Dayal: "Active Database Systems," 3rd International Conference on Data and Knowledge Bases, pp.150-169, 1988.

[3] D.McCarthy and U.Dayal: "The Architecture of an Active Database System," ACM SIGMOD Conference, pp.215-224, 1989.

[4] A.Aiken et.al: "Behavior of Database Production Rules: Termination, Confluence and Observable Determinism," ACM SIGMOD Conference, pp59-68, 1992.

[5] R.Hull and D.Jacobs: "Language Constructs for Programming Active Database," VLDB Conference, pp.455-467, 1991.

[6] S.Ceri and J. Widom: "Deriving Production Rules for Constraint Maintenance," VLDB Conference, pp.566-577, 1990.

[7] U.Dayal: "Active Database Management Systems," 7th IEEE Conference on Data Engineering, Tutorial, 1991.

[8] M.Stonebraker et al.: "The Postgres Rule Manager," IEEE Trans. on Software Engineering., Vol.14, No.7, pp.908-921, 1988.

[9] M.Stonebraker et al.: "The Implementation of Postgres," IEEE Trans. on Knowledge and Data Engineering, Vol.2, No.1, pp.125-142, 1990.

[10] J.Widom and S.Finkelstein: "Set-Oriented Production Rules in Relational Database Systems," ACM SIGMOD Conference, pp.259-270, 1990.

[11] C.Maindeville and E.Simon: "A Production Rule Approach to Deductive Databases," 4th IEEE Conference on Data Engineering, pp.234-241, 1988.

[12] U.Chakravathy and S.Nelson: "Making an Object-Oriented Database Active," 3rd Conference on Extending Data Base Technology(EDBT), Lecture Notes 416, pp.393-406, 1988.

[13] E.Hanson: "Rule Condition Testing and Action Execution in Ariel," ACM SIGMOD Conference, pp.49-58, 1992.

[14] U.Schreifer et al.: "Alert: An Architecture for Transforming a Passive DBMS into an Active DBMS," VLDB Conference, pp.469-478, 1991.

[15] O.Dias et al.: "Rule Management in Object Oriented Databases: A Uniform Approach," VLDB Conference pp.317-326, 1991.

[16] N.Gehani and H.Jagadish: "Ode as an Active Databases: Constraints and Triggers," VLDB Conference, pp.327-336, 1991.

[17] N.Gehani et al.: "Event Specification in an Active Object-Oriented Database," ACM SIGMOD Conference, pp.81-90, 1992.

[18] R.Reiter: "On Formalizing Database Updates: Preliminary Report," Conference on Extending Data Base Technology(EDBT), Lecture Notes 580, pp.10-20, 1992.

[19] D.Jacobs and R.Hull: "Database Programming with Delayed Updates," Workshop on Data Base Programming Languages, pp.416-428, 1991.

[20] R.Fagin et al.: "Updating Logical Databases," ACM Principles of Database Systems Conference, 1983.

[21] C.Beeri and T. Milo: "A Model for Active Object-Oriented Database," VLDB Conference, pp.337-349, 1991.

[22] E.Teniente and A.Olive: "The Events Method for View Updating in Deductive Databases," Conference on Extending Data Base Technology(EDBT), Lecture Notes 580, pp.245-260, 1992.

[23] U.Dayal et al.: "A Transaction Model for Long-Running Activities," VLDB Conference, pp.337-349, 1991.

[24] R.Milner,"Communication and Concurrency", Prentice-Hall 1989.

[25] R.Milner et al.,"A Calculus of Mobile Process", Report of LFCS, Univ.of Edinburgh. 1989.

[26] 大蒔、二木: " 図書館の問題とエレベータの問題の LOTOS による仕様記述," 情報処理学会研究会報告, SE64-12,1989.