データベースシステム 106－8
（1996. 1. 24）

ECAアーキテクチャを支援するためのSQLサーバの拡張

小島 功

電総研情報ベース研究室
kojima@etl.go.jp

ECAアーキテクチャで代表される、アクティブデータベースをSQLを基礎とした関係データベースシステムを用いて構築する手法について述べる。基本的な考え方は、ECAアーキテクチャにおける条件／動作記述にSQLを用いることと、プロセス代数ベースのイベント記述とそれらの間の並行制御記述を組み合わせることである。この言語ECA-SQLにより、

- 述語による条件記述表現と、集合操作によるデータベース処理
- 代数表現による複合的なイベント記述／処理
- 結合モードと並列処理の支援。また、並列処理の振る舞いの検証

などが可能となる。既存のSQLサーバは、システムの1要素として用いることができる。次に、これを用いた応用として、アクティブデータベースをWebのバックエンドとして用い、ネットワーク上のハイパーテキストの管理に適用することを考える。

- ECAリンクによる柔軟性の高いリンク記述
- Webなどの環境でのマルチメディア／ネットワーク制御

について述べ、この応用に対する適合性を議論する。

Extending SQL Server to Support ECA Architecture

Isao Kojima

Information Base Section
Electrotechnical Laboratory

This paper presents the organization of the active database system supporting ECA architecture using exising SQL relational databases. Basic approach underlying the system is 1) to support SQL-like conditional descriptions and set operations, 2) to support algebraic expression for databases events and their compound representations, and 3) to support concurrent operations between SQL processes and their coupling modes. ECA-SQL is provided as an extension of the SQL language.
Hypertext application of the ECA active databases is presented. For instance, it is useful to use adctive databases as a backend of hypertext databases. Using ECA based rule expression, it is possible to provide 1) flexible link expressions between text data and 2) flexible control structre by using parallelism of active databases.

1 まえがき

アクティブデータベースシステム [1] は、規則表現とその自動的な実行、波及的な更新処理、規則の並列実行あるいは、複雑なイベントの扱いと言った様々な機能的な特徴を持ち、データベースの応用分野の拡大と共に近年研究の進められている分野の一つである。
このようなアクティブデータベースは、多くが ECA (event-condition-action) アーキテクチャと呼ばれる枠組みによって扱われる。これは、データベースに登録されるアクティブな規則を、1）その起動されるタイミング (Event)、2）その時点でデータベース上で成立する条件 (Condtion)、3）条件成立後に実行されるデータベース処理 (Action) の3つの要素により扱い、これらの記述による規則によって、全体として高度なデータベース処理を行なおうとするものである。
このようなデータベースシステムは、従来のシステムと異なる要素が多いので、データモデル、記述言語からシステム構成に至るまで、多くの新しい機能の導入を必要とする。従って、その実現については、様々な問題が考えられる。
一方で、関係データベースシステムやオブジェクト指向など、既存のデータベースシステムも広く普及しており、これらの機能を用いることも重要である。本稿では、既存の関係データベースシステムとその言語の特徴を可能な限り生かした、アクティブデータベースシステムの実現手法について議論する。

2 プロセス代数による ECA アーキテクチャの支援

ECA アーキテクチャを支援するアクティブデータベースは、既に多くの実験システムが提案 [2] されている。さて、ECA を基礎とするデータモデルに対しては、次のような要求が考えられる。

1. 従来のデータベースモデルで扱いにくい、振る舞いの扱いが重要。
2. 並行処理による更新やその波及によるデータベースの状態変化の扱いが不可欠。
3. 規則のデバッギングや処理の検証が重要。

結局、モデルは振る舞いとその並行処理を扱え、かつ何らかの方法で処理の検証や結果の正当性の判定が可能であることが重要である。これに対して、著者らの提案するプロセス代数によってモデル化するアプローチ [3] は、次のような特徴がある。

1. プロセスによる、自然な並行動作の記述。
2. プロセス間の通信による、更新や制御の記述とその波及。
3. 言語としてのプログラミング能力が高い。
4. 等価性や停止性の判定が可能。

従って、ECA アーキテクチャをプロセス代数の枠組みで扱えば、その検証実行能力を使って規則の並行動作や更新による状態変化を扱うことができ、実際にデータベース上で動く規則を書く利用者を支援できる。

3 プロセス代数による、アクティブデータベース記述言語 ECA-SQL

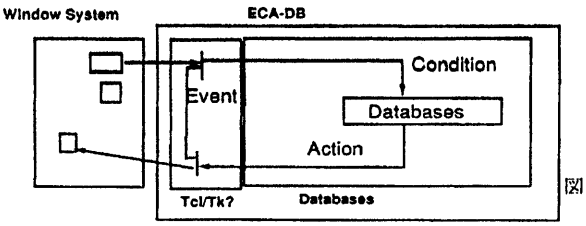
ここでは、このようなプロセス代数に基づいた、アクティブデータベースの記述言語 ECA-SQL を述べる。基本的には、SQL に基づく条件／動作記述と、プロセスによる並行記述と複合事象記述を持つ。単純には、

```
Label:(event);((SQL-exp) | (SQL-exp) / (SQL-exp
```

のように、ECA の各要素がプロセス代数を元とした結合演算子で結ばれた形式を持つ。
ECA の各要素にあてはめて考えると、

- event は更新などのデータベースイベント＋ウィンドウなどの外部イベント（例えば、あるテキストがアクセスされたとか）を基礎イベントとし、プロセス代数演算子で組み合わされた複合表現を許す。
- condition は、SQL の条件記述＋組み込み手続き (select from TextDB where 以下で条件を書き、データベース上で成り立つ条件のチェックをする。一般の処理は組み込み関数的に扱い、例えば、where PROGRAM(select from ...) ＞ 5 とか言う形でプログラムを呼び出す。() 内はパラメタ。
- action も同様にデーターベースの処理文例えば、Update DB where .... の形式。ないしは、プログラム実行 PROGRAM () のような形式を持つ。

ＥＣＡによるハイパーテキスト
処理メカニズム



Event: Click / Select / Update
Condition: Select button from GraphicDB
where
Action: Select Text from TextDB where
.....

図 1: ECA メカニズムの実現 (1)

概念的には図 1のようになる。
大きな特徴として、ECA の各要素に対して結合モードと伴うトランザクション処理との対応関係を記述できることである。例えば、データベースである条件が成立したとして、それに続く動作の実行を別のトランザクションであれば、その動作の結果に条件評価は影響されない。つまり、一貫性の保存／処理単位を利用者がある程度プログラミングできる。
例えば、C - A が独立 (de-coupled) モードだったとすると、ECA-SQL の C、A の記述

```
(select * where ..) | (update text set ...)
```

は、並行実行可能な演算子 | によって結ばれる。他にも、CCS から得られた演算子のマクロとして記述される。
Coupling(C1,C2,M) は、T1c のような処理を行なうルール C1,C2 との接続関係を定義するもので、トランザクションの起動イベント st1,st2 と受けとるコミットイベント d1,d2 の順序関係によって定まる。
また、C1;C2 はこの順序で処理をする演算、C1 | C2 は並行に実行するような定義である（記述の区別のため、図中では 2 重線 C1 | | C2 にしてある）。
結合モードは、コミットとサブトランザクションの起動 (st1,st2) との順序関係から、例えば、次図のように定義できる。
基本的な特徴は、

$$Coupling(C_1, C_2, M) \stackrel{\text{def}}{=} (st_1.C_1[d_1/done]|| st_2.C_2[d_2/done]||M)\backslash\{st_1, st_2, d_1, d_2\}$$

$$Rule(event, CA) \stackrel{\text{def}}{=} (event.\overline{st}.0|CA)\backslash\{st\}$$

$$C_1; C_2 \stackrel{\text{def}}{=} (C_1[d/done]||d.C_2)\backslash\{d\}$$

$$C_1 \parallel C_2 \stackrel{\text{def}}{=} (C_1[d/done]||C_2[d_2/done]|| d_1.d_2.done)\backslash\{d_1, d_2\}$$

図 2: ECA-SQL の演算子の CCS によるマクロ記述

$$DC \stackrel{\text{def}}{=} st.\overline{st_1}.\overline{st_2}.d_1.\overline{com}.d_2.\overline{done}.0$$

$$IM \stackrel{\text{def}}{=} st.\overline{st_1}.d_1.\overline{st_2}.d_2.\overline{com}.\overline{done}.0$$

$$DF \stackrel{\text{def}}{=} st.\overline{st_1}.d_1.\overline{com}.allcom.\overline{st_2}.d_2.\overline{done}.0$$

図 3: カップリング・モードの記述

- SQL 表現による条件、動作の述語表現と、集合的処理。
- プロセス代数によるイベントの記述と、その組合せ表現。

である。さらに、条件、動作で実行される SQL 処理を基本的なプロセスの単位として扱うと、プロセス代数で示されている等価性／合流性といった理論を適用することができる。つまり、イベントと個々の SQL プロセスとの関係はプロセス代数そのもので扱えるので、プロセス代数における一貫性や停止性とデータベース処理における一貫性や停止性との関連を扱うことで、有益な性質や検証条件を導くことができる。

4 ECA-SQL システム

次に、ECA-SQL を可能な限り既存のサーバを用いて実現する方法について議論する。
ECA-SQL とは基本的には、SQL 表現がプロセス代数演算子でつながれた記述であるから、個々の SQL プロセスは結合モードで制御できるように、別々の処理であっても良いわけである。条件、動作などの SQL 処理は並行に動くトランザクションであっても良い。つまり、この言語は基本的には SQL のフロントエンドとして実現可能である。
また、プロセス代数によって検証する場合は SQL を一つのプロセスとしてシンボリックに扱うこととし、

処理すればよいことになる。例えば、

```
T1:(update V);
; (select from V+ union R where not in V-))
(この場合Cはなし)
```

は、基本的には実際のaction()以下はSQLの処理系に実行させることができる。ここで、次のような点に注意したい。

- 結合のための代数オペレータとはCCSなどの、比較的強い検証能力を持つ代数に展開可能なマクロの範囲に限る。ここでは、先に上げたようにCCSのマクロに限る。
- 実際にはプロセスの間にはcommitなり真理値なり制御用のデータが間に流れるが、基本的にはシンボリックに扱える範囲になっている。

さて、

- データベースシステム：検索／更新などデータベース処理を行なう基本ルーチンを提供しているパッケージを仮定する。同じデータに対してSQLの呼びだしが可能ならば、なおいうことがない。
- 並列制御／操作系：実際に複数のプロセスなりプログラムを、ある代数モデルに従って実行するもの。ここではCCSを用いる。
- 検証系：Concurrency Workbenchのような、ある検証が可能なプログラム。ここではエジンバラのConcurrency Workbenchを用いる。

で、ECA-SQLシステムとは、これらの要素を相互につなぐ／呼び出すフロントエンドとして実現されている。(図5)
具体的には、既存のデータベースの2つのインターフェイスを利用して、以下のような手順で処理を行なうことになる。(図4)

- ファイルレベルのインターフェイスを使うイベント検知系(event detector): イベントはデータベースを監視していないと検知できない。同様に、イベント記述自身がSQLの範疇から外れるので、これを検知する系／記述は作らなければいけない。ウィンドウシステムのイベントもここに通知される。また、これは常時動き、イベントをモニタする。
- SQLサーバを使った条件評価／実行部(condition evaluator/action executor) 実質的にはルールのうち条件／動作部を実行する。これはSQLサーバに送られて、呼びだしと結果のデータの受けとりを結合モードのタイミングで行なう。Actionがルール（すなわちマクロ化されて登録されたC-SQL文）の場合は、さらにルールが評価される。プログラムの場合には呼び出してデータを渡して終了する。
- 規則管理フロントエンド(Rule Manager) ルールが入力されたら構文を解析して、必要な事象をevent検知系にぶら下げる。
- トランザクション管理部(Transaction Manager) イベントが起こったらルールを引っ張ってきて、これをトランザクションとする。親があったら、この下にサブトランザクションとしてインスタンスに付加する。トランザクションの木構造と階層関係はここで保持される。結合モードの存在から、規則処理とトランザクション処理が1対1対応にならないので、このような管理が別に必要となる。

さて、文献[1]の例をECA-SQLを使って記述してみる。

```
T1:(update rate);
 (select * where new!=old )
|(select value from new-value)
T2:(update rate);
  (select * where currency="$" and new < 100);
  (buy(1000))
T3:(buy());
  (select * where account < 0)
   /(select warning(X) where account<0)
buy(X) = update account set account=account-X
```

で考えると、仮にT 2を入力したとして、

- update rate というイベントがテーブルに登録される。
- 条件部分はeventを含めたAssertion （の機能があれば）としてDBに登録する。
- （で、何かの拍子に実行されたら）、()内の表現をSQLに送って評価し、
- その結果により即buy（これは別のマクロとして登録してあるとして）を起動して、トランザクションの木構造の下に（immediateだから）追加し、処理系に制御を渡す。

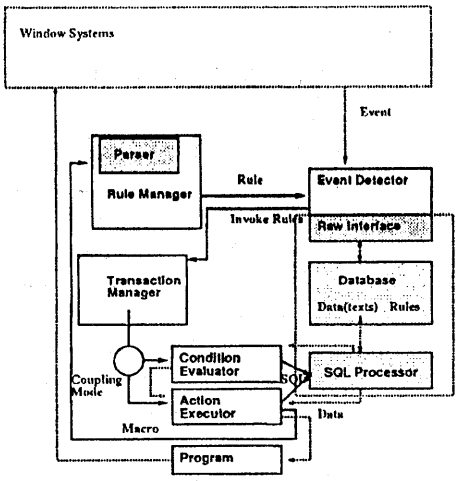


図 4: システムの構成

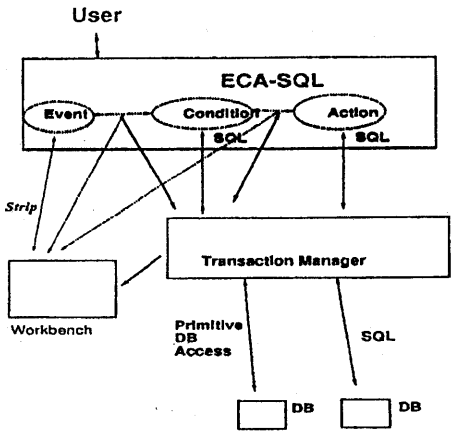


図 5: 結合モードの利用

これは一方で、

```
T1=e;C1|A1
T2=e;C2;A2
T3=A2;C3/A3
```

と制御的には等価だから、例えばこれだけの情報を言語から除いて、CCS なりのシミュレータに与えることで、検証に用いることができるわけである。
処理の各段階でデータベースアクセスがあって全体の処理が重くなるが、実際には、event とそれによって起こるデータベースの変更／検索は全体の小さな部分だけに起こるので、差分的な処理手法をとることで、かなり解消できる。

5 ECA-SQL による、ハイパーテキストの管理

さて、このようなアクティブデータベースの応用として、WWW のようなハイパーテキスト管理の応用に用いることを考える。
普通のテキストデータベースでは、更新にともなうリンクの管理が難しい。例えば、データベース操作の対象はレコード単位なので、例えばレコードの変更があった時に、それに付随するリンクへの変更は処理が難しい。

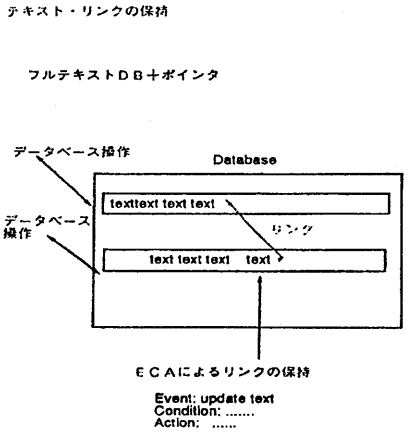


図 6: テキスト管理

そこで、ECA によるリンク保持機構を提供する。例えば、ECA によるリンクの保持規則は、

- event テキストの更新
- condition リンクのローカルな接続性のチェック
- action 上に失敗したら、エラーの表示 and/or リンクの変更

のようになる（図 6）。当然これは波及的に実行されるから、

- event リンクの変更
- condition リンクの記述（抽象的なものを考えているから）により検索されるデータの検査
- action 上に失敗したら、エラーの表示 and/or リンクの変更／データの変更

といった処理を呼び出すことになる。実際には、この機構は（非常に重いが高機能の）リンクの記述そのものであって、

- event click/select → click/select（をデータベースへの検索として解釈した時の、データベース event の一般）（複合事象）
- condition なし → データベース処理とその結果
- action ファイル名（で該当ファイルをアクセス）→データベース処理とその結果

で、これが結合モードでつながれた状態を指す。（図 7）。つまり、ECA によるリンク記述メカニズムを使うと、単なるファイルをたどる以上の機能を規則によって記述することができる。
また、リンク自身が ECA オブジェクトとして、それらが呼ばれて処理される環境を考えるとすると、当然、それらの実行管理というかハンドルをシステムが考えることになる。つまり、リンク管理の最適化、抽象化、組織化の余地が生じ、例えばあるリンクが選ばれたら、ある最適化基準によって、その代わりに別のリンクをアクセスするといったことができる（図 9）。
WWW のようなネットワーク上でのハイパーテキスト環境とデータベースを組み合わせた研究はいくつか行なわれているが、ここで示すアプローチは、次のような特徴を有する。

1. ECA 規則を使った、柔軟性の高いリンク記述: 例えば、ボタンのクリックをイベント、該当するデータのアクセスを動作とすると、WWW における動作は ECA アーキテクチャによりかなり高機能化できる。例えば、複合的なイベント記述や、WWW のサーバ側の条件に応じたデータベース処理、あるいは CGI に近いような高度のプログラム処理を動作で記述する等である。このように、通常のデータベースシステムとの組合せに比べ、はるかに高機能でかつ親和性の高い環境が構築できる可能性がある。

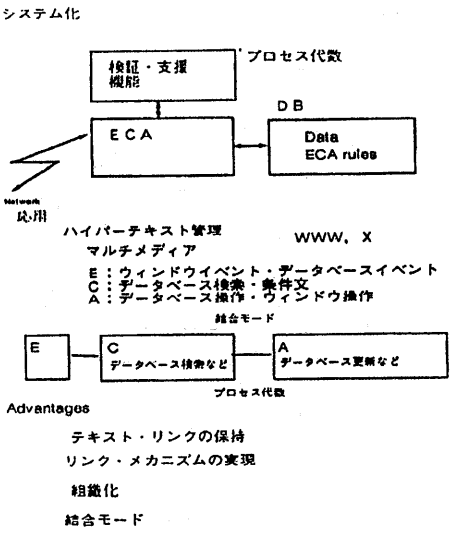


図 7: システムの概要

2. マルチメディア、ネットワーク環境への適合性: 例えばテキストデータベースのキャッシュの制御や、マルチメディアデータの出力／表示などは、データベースの状態変化に対する自動的な更新や、同期を含んだ並列処理になる。ECA によるデータベースの処理記述は、このような機能についてもデータベースシステム上での統合的な記述を可能にする。

結局、高機能なリンクを実現することで、上のような問題に対して統一的な記述ができると言う点が特徴である。また、プロセスモデルに基づいているので、この上で例えば concurrency workbench などと接続でき、検証の実験などが可能である（図 8）。現在、

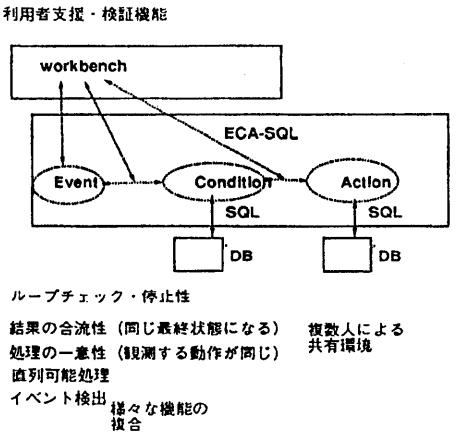


図 8: workbench との接続

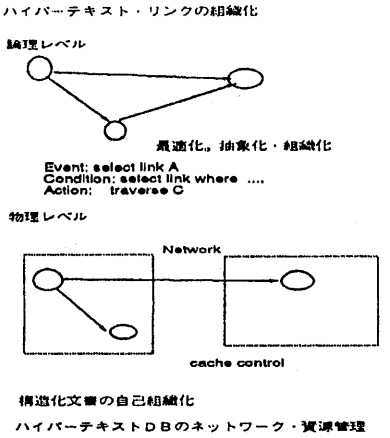


図 9: ECA リンクの実現 (2)

- 複合 event の記述／受理の方法
- ECA の接続関係による並列処理の検証 （つまり、個々の E,C,A を一つのプロセスと言うか処理の単位と仮定して、結合モードとか、条件評価の結果などそれらの間の制御情報をプロセス間でやりとりするようか環境）

が可能になっている。
また、結合モードを使ってマルチメディア／ネットワークに適応することもできる（図 10）。例を上げる。

- 遅延 (deffered) モードを使った利用者制御。

  例えば、ベースのテキストだけをアクセスできた時点で制御をとにかく返し、インラインイメージは遅延されたプロセスとして処理させるような場合が記述できる。

  deffered の条件は親のプロセスのコミット時までということなので、例えば全部のイメージが来るまでに別のページをアクセスすると、それは親の commit であるから、必然的に遅延されたプロセスも commit/abort せざるをえない。つまり、次のテキストをアクセスしてからイメージが返ってくるといったことを避けられる。

- 独立 (decoupled) モードを使った並列処理

  これは、処理がトランザクションとして完全に独立な場合で、例えばポインタの先読みとかの場合に用いることができる。

- 即時 (immediate) モードを使ったアクセス制御。

  この場合、E-C の評価の結果によって即時に処理ができるので、条件の結果によってアクセスすべきテキストの場所を変えるなどといった処理が記述できる。例えば、キャッシュが存在すれば、そちらから読むと言った記述である。

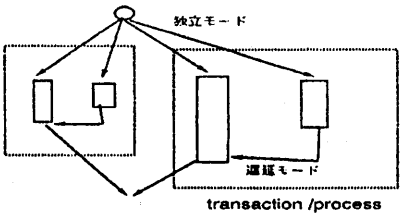
基本的には、ECA-SQL の結合モードの導入により、並行処理とトランザクションの概念が整理でき、この手の細かい処理記述が可能になる点が特徴である。

6 現状と議論

現在は、実験用の SQL サーバ [6] と、エジンバラの Concurrency Workbench[3] を用いた基本システムを実現中であり、基本的なイベントの受け渡しと、ここで得たような一部の並行制御を含む規則処理が可能となっている。ただし、SQL システムの細部を一部拡張したので、引続き postgres95 などの汎用の SQL サーバを用いたシステム実現を行なう予定である。

結合モードの利用

1）マルチメディア。並列処理



2）ネットワーク・アクセス

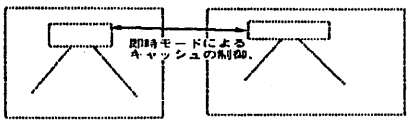


図 10: 結合モードの利用

参考文献

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

[2] 小島：“アクティブデータベースシステム” 電総研研究速報 TR95-31,1995.

[3] 小島他：” プロセス代数による、アクティブデータベースの支援について” 情報処理学会研究報告 1993.05

[4] RIDE94', Research Isahues in Data Engineering , 1994.02

[5] IEEE Bulletin of Data Engineering 1992 Vol15,No1-4

[6] I.Kojima et.al."Design and Implementation of an Object-oricnted Database System Using an Extensible Database Toolkit",ICSCP2,1992.