データベースシステム 89−10
（１９９２．７．２３）

オブジェクト指向データベースにおける制約管理について

石川博　　酒井洋一
富士通研究所　富士通大分ソフトウエアラボラトリ

設計業務支援等のデータベースの高度応用のためにオブジェクト指向データベースシステムが開発されている。こうした高度応用では，従来の応用に比べてアクティブな機能が要求される。即ち，設計対象が持つ複雑な構造に加えて構成要素間の制約条件を記述する必要がある。その際，単一の属性に関する制約と複数のオブジェクトにまたがる制約の両方を表現する必要がある。この目標のために，完全性制約の維持，値の変更伝播のためのイベント／トリガ機能，値充足（生成）のための制約ルールからなる制約管理機能を考える。本論文では，制約管理機能を持つアクティブデータベースの機能と実現を，オブジェクト指向データベースを基にして述べる。

ON CONSTRAINT MANAGEMENT IN AN OBJECT-ORIENTED DATABASE SYSTEM

Hiroshi Ishikawa
Fujitsu Laboratories Ltd.
1015 Kamikodanaka, Nakahara-Ku, Kawasaki 211, Japan
Yoichi Sakai
Fujitsu Oita Software Laboratories Limited
17-58 Higashikasugamachi, Oita-Shi, Oita 870, Japan

Object-oriented databases have been developed for advanced applications such as CAD and office document management. Such applications require more active functionality than conventional applications. Complex structures and constraints of design objects must be represented. Constraints include those on single attributes and those between multiple attributes. To this end, we must provide constraint management facilities to object-oriented databases. These facilities include integrity constraint check, event/trigger mechanism for update propagation, and rule mechanism for constraint satisfaction. In this paper, we discuss the functionality and implementation of an active object-oriented database system which realizes the constraint management facilities.

1. はじめに

我々は，設計等のエンジニアリング業務支援やオフィスにおける構造化文書の作成管理等のデータベースの高度応用のためにオブジェクト指向データベースシステムJasmine〔AOSH90〕〔ISHI91c〕を研究開発してきた。こうした高度応用では，従来の応用にくらべてよりアクティブな機能が要求される。即ち，設計対象が持つ複雑な構造や関連を表現するだけでなく，設計仕様としての制約や図形的な制約のように構成要素間の制約条件を扱うことが必要になる。さらに制約は単一のオブジェクトに還元できる場合と複数のオブジェクトに本質的にまたがる場合があり，その両方を表現できなければならない。こうした目標を達成するための新しいデータベース技術として制約管理機能を考える。

制約管理機能としては，完全性制約の維持，値の変更伝播のためのイベント／トリガ機能の他に値充足のための制約ルール等が含まれる。これらの機能の全部またはその一部を持つデータベースを総じてアクティブデータベース〔MORG83〕と呼ぶことにする。イベント／トリガ機能の研究として，HiPAC〔MCCA89〕やSAMOS〔GATZ91〕があり，リレーショナルデータベースの拡張としてのルール機能の研究としてPostgres〔STONE90〕やStarburst〔LOHM91〕等がある。またオブジェクト指向データベースの導出属性や完全性制約管理の研究として，それぞれCactis〔HUDS89〕とODE〔AGRA89〕がある。我々もオブジェクト指向データベースシステムJasmineをエンジニアリング業務支援に対して適用した研究システムHyperCAD〔ISHI91a〕〔ISHI91b〕を報告してきた。またオフィス文書作成支援のための制約管理機能としては，文書部品間の同等性を含めた意味的関係を利用する方式〔TANA92〕を報告してきた。

ここではエンジニアリング応用での経験をもとに，より一般的な枠組みを提案していく。提案する枠組みが解決しようとしている問題点としては以下のものがる。まず従来のルールの方式では制約条件を満たす値を生成する能力が弱い。第二にトリガ機能とルール機能の両方を単一のパラダイムで表現すべきか，複数のパラダイムを与えて表現すべきかという問題がある。第三にアクティブな機能の意味を明確にしているか，特にデータベースのトランザクションや一貫性の概念と関連付けてのべられているかという問題がある。最後に機能の効率的実現について，特にトリガ／ルールの評価方法を述べる必要がある。ここに提案する研究はオブジェクト指向データベースはもちろん，トリガ機能，ルール機能，そしてnested transaction〔GARC87〕の研究と深くかかわっている。我々はオブジェクト指向データベースJasmineを使ってアクティブデータベースJasmine/A（Jasmine Active Database）を実現した。オブジェクト指向データベースの基本機能を提供するJasmineと区別するためにこう称する。この論文ではJasmine/Aの制約管理機能とその実現について述べる。機能としては完全性制約の維持，イベント／トリガ機能，制約ルールまでを扱い，それらの機能をマルチパラダイムで提供する。その意味付けをトランザクションと一貫性の概念で説明する。処理は個々のオブジェクトに対するアクセスか，集合論的なアクセスかに応じて，ページバッファ上かオブジェクトバッファ上で効率的に処理する。さらにユーザが制約評価の方式を制御することができる。ちなみに設計等の協調的作業の支援のための要素技術として異種データベースの研究〔ISHI92〕を行なっており，それをJasmine/H（Jasmine Heterogeneous Database）と呼ぶ。Jasmineを拡張して分散環境における複数データベースのスキーマ統合を行なうためのビュー機能を実現した。ただしここではくわしくは述べない。

本論文は以下のように構成されている。第2章では解決すべき課題と関連する他の研究について述べる。続く第3章ではアクティブオブジェクト指向データベースJasmine/Aの制約管理機能とそのセマンティクスを説明する。第4章でその効率的実現について説明する。第5章で今後の課題について論ずる。

2. 課題と他の研究

2.1 課題

この研究によって解決しようとしている問題点をまとめると以下のようになる。第一にオブジェクトの属性に関する制約を管理することである。制約は1つの属性に閉じることもあれば，同一オブジェクトの他の属性や，他のオブジェクトの属性にまたがる場合があるので，特に複数の属性にまたがる制約を扱える必要がある。制約機能としては，属性値に関する制約条件のチェックや維持，属性値の変更の伝播だけでなく，制約を満足するようなオブジェクトの属性値を生成する機能が必要になる。ユーザに対してはこれらの異なる目的のそれぞれに相応しい複数のパラダイムを提供する必要がある。定型的なことはシステムにまかせ，ユーザは応用独自の制約を指定できる必要がある。さらに制約管理機能の意味をできるだけ，データベースのトランザクションや一貫性の概念を用いて説明できる必要がある。また効率的な実現方式を与えると同時に，最適化に使える情報をシステムが全てわかるわけではないので，システムの限界を補って，ユーザが制約の評価方式を制御できることが望ましい。この論文では以上のような観点からJasmine/Aについて述べる。

2.2 他の研究

Jasmine/Aと関連する他の研究について簡単にふれる。まずオブジェクト指向データベースにアクティブな概念を統合しようとする試みとしてHiPAC〔MCCA89〕とSAMOS〔GATZ91〕がある。HiPACはイベント（E），コンディション（C），アクション（A）パラダイムに基づく。イベントとしてはプリミティブイベントの他に時間イベントや複合イベントをユーザは定義することができる。実行トランザクションのカップリングモードとしてECとCAカップリングの2モードと，評価モードとしてimmediate／deferred／decoupledの3モードを組み合わせて利用できる。トリガ機能としてのルールが中心であり，値の生成のための制約ルールの機能はない。SAMOSもHiPAC同様に，ECAパラダイムに基づき，複合イベントや時間イベントを定義できる。さらに2つのカップリングモードと3つの評価モードを指定できる。トリガ機能中心であり，値の生成を行なうルールはない。

リレーショナルデータベースまたはその拡張に対するルール機能の追加に関する研究としてPostgres〔STON90〕とStarburst〔LOHM91〕がある。Postgresは拡張リレーショナルデータベースであり，ルール機能を提供する。Jasmine/Aとは異なり，完全性制約チェックや変更伝播の他にビュー機能も含め，単一のパラダイムアプローチを取る。ただし値生成のためのルールはない。ルールの評価方式として前向きと後ろ向きの2つを持ち，システムが最適化の際に選択する。実現方式としては，タプルレベルと問い合わせレベルの2通りがあり，それぞれJasmine/Aのオブジェクトバッファとページバッファの評価方式に対応する。複合イベントは定義できない。Starburstも拡張リレーショナルデータベースであり，プロダクションルールを提供する。挿入，削除等のイベントに対してコンディションとアクション部には，拡張SQLによる問い合わせとデータベースコマンドをそれぞれ記述できる。Alertと呼ばれるサブシステムでビューを定義することができ，これもルールの一種と考える。複合イベントと値生成のためのルールはない。

オブジェクト指向データベースそのものに完全性制約とトリガ機能を備えたシステムにODE〔AGRA89〕がある。Cactis〔HUDS89〕もアクティブなオブジェクト指向データベースを目指している。メソッド（それをルールと考える）定義における依存関係に基づくルールの評価方式を提案する。ODEもCactisもともに複合イベントと値生成のためのルールはない。

3． 制約管理機能

3.1 完全性制約とトリガ機構

ここではJasmine/Aの提供する完全性制約とトリガ機構について説明する。それらはすべてデモン〔ISHI91c〕としてユーザに提供される。ここでは制約をイベント／コンディション／アクションとしてモデル化する。Jasmine/Aではまずシステム定義のデモンとしてmandatory, multipleデモンがある。以下に構文を示す。

```
PropertyName    mandatory
                multiple
```

mandatoryはインスタンス生成時（即ちinstantiateイベント発生時）に，もしそれが指定された属性に値が挿入されると（即ちinsertイベントが発生すれば）正常終了する。さもなければエラーを引き起こす。またその属性の値が削除されると（即ちdeleteイベントが発生すると），エラーが起きる。もちろん値の置き換え（即ちreplaceイベント）は可能である。multipleが指定された属性（即ち多値属性）に対してinsertイベントが発生すれば，値が追加されるが，multipleが指定されていない属性（即ち単一値属性）に値の追加はできない。もちろん値がない場合と置き換えはできる。mandatory, multipleデモンの２つともイベント／コンディション／アクションともにシステム定義である。

次にconstraintデモンについて説明する。その構文は以下のようになる。

```
PropertyName        constraint  {condition}
```

これはinsert／replaceイベントが発生した時にユーザが指定したコンディションが成り立てばコミットし，さもなければアボートする。この場合，イベントならびにアクションはシステム定義である。

さらに属性には次の４つのデモンが指定できる。

```
PropertyName    if-needed     {if-needed demon}
                if-added      {if-added demon}
                if-deleted    {if-deleted demon}
                if-replaced   {if-replaced demon}
```

if-neededデモンは属性の値が参照されると（即ちreferイベントが発生したときに），起動される。if-added, if-deleted, if-replacedデモンは，それぞれinsert, delete, replaceイベント発生時に起動される。イベントはいずれの場合もシステム定義である。各デモンは以下のように，ユーザ定義のコンディションとアクションをモジュール化したものである。

```
if condition then action
else if condition then action
...
```

デモンの内部ではSelfとValueというシステム定義変数が利用できる。Selfには当該イベントが発生したインスタンスが束縛される。Valueには挿入された値か，削除された値か，置き換えられる新しい値が束縛される。置換される前の値はOldValueに束縛される。

属性にだけでなくメソッドにもデモンを付加することができる。

before-demon　method　after-demon

メソッドそのものの起動はユーザ定義イベントの発生になる。beforeデモンは，ユーザ定義イベントが発生したら，その処理の前に起動される。afterデモンはユーザ定義イベントに伴う処理の後に起動される。before／afterデモンは前述した属性に対するデモンと同様にユーザ定義のコンディションとアクションをモジュール化したものである。通常beforeデモンでは，処理の前提条件のチェックや達成を主な仕事とし，afterデモンは，処理の影響を他へ波及させる。この場合イベント／コンディション／アクションのすべてがユーザ定義である。

以上のべてきたデモンによる完全性制約やトリガ機構について注意すべき点を説明する。以上のデモンはすべてクラスの中に指定され，個々のインスタンスに対して適用される。もちろん集合論的な検索や挿入，削除，置換に対しても起動される。つまり集合指向でもある。１つのイベントに対して適用できるルール（コンディション／アクション）が複数存在する場合がある。それらはif then elseの構文によってif文でコンテクストを参照したり優先順位を指定することで実現する。複合イベントに対してはイベントオブジェクトを定義し，そこで複合イベントの生起，コンディション，アクションを管理する。イベント／コンディション／アクションのカップリングモードについてもイベントオブジェクトとあわせてユーザが定義する。オブジェクトに対する操作はシステム定義であれユーザ定義であれ必ずメソッドを通して行なわれる。前者には属性の参照，put，delete，replace等のシステム定義メソッドが含まれ，後者にはユーザ定義のメソッドが含まれる。メソッド起動がイベント発生に対応するので，システムはイベントの発生を認識することができる。

デモンの記述力を確認するために他システムで提案されている機能をどのように表現するかについて説明する。まずPostgres〔STON90〕の例を考える（ルールの番号は現文献のそれと一致する）。

Salary　if-replaced { if Self.Name = "Joe" then EMP.replace ("Salary", Value) where EMP.Name = "Sam"}

これはJoeの給料が変更されたら，Samの給料にも同じ値を反映させるルール（ルール１）である。

Salary　if-needed { if Self.Name = "Joe" then EMP.replace ("Salary", Value) where EMP.Name = "Bill"}

このルール（ルール２）は，Joeの給料がアクセスされる度にBillの給料がJoeの給料に等しくされる。

Salary　if-needed { if Self.Dept = "shoe" and user() = "Joe" then Value = Null }

Joeはshoe departmentの従業員の給料を見ることは許されない（ルール３，ルール４）。ルールに対する例外を定義するルール６とルールそのものを扱うルール７は，イベントオブジェクトを導入することで扱

えるが，ここでは述べない。

Salary if-needed { <AUDIT>. instantiate (Access: user(), Object: Self.name, Value: Value) }

このルール（ルール7）は給料に誰かがアクセスすると監査のための情報（security audit）を登録する。

原論文にあるルール8はTOY_EMPというリレーション（クラスに相当）を定義している。これはJasmine/Aよりむしろ，別の拡張であるJasmine/Hのビュー機能を用いて実行する。Jasmine/AとJasmine/H〔ISHI92〕は排他的なものではなく，相補的である。即ち

```
TOY_EMP
  OriginalClass EMP
  Property *
  Method *
  Selection Dept = "toy"
```

ここで＊は元のクラスの全属性，全メソッドを指定したことになる。このようにJasmineでは制約管理は，それぞれの機能に相応しいパラダイムがあると考えて，マルチパラダイムでアプローチする。これらのデモンはクラス階層をとおして遺伝されるし，ポリモルフィズムもある。ユーザがデモンの起動をサプレスすることもできる。

次にDateのHypothetical Integrity Language〔DATE90〕の例を見て見よう（例の番号は原文献のそれに対応する）。

（例1）Status constriant { Value > 0 }

（例2）Year constraint { 0 < Value and Value < 99 }

```
Month constraint { 1<= Value <= 12}
Day Costriant {Self.Month in (1,3,5,7,8,10,12) and Value > 0 and Value <32
  or Self.Month in (4,6,9,11) and Value > 0 and Value < 31
  or Self.Month = 2 and mod(Self.Year, 4) = 0 and Value > 0 and Value < 30
  or Value < 29 }
```

（例3）Status if-replaced { if Value <= OldValue then Self.replace ("Status", OldValue)

これはイベントの効果を補償（相殺）するようなアクションを指定している。以下のようにも指定できる。

```
Status constraint { Value > OldValue}
```

（例4）Status if-added { if S.avg() <= 25 then Self.delete("Status", Value)

これも補償するアクションの例である。

（例5）City constraint { Value = "London" and Self.Part.Name = "P2"}

（例6）primary keyはOIDで表わす。

（例7）foreign keyはインスタンスにアクセスする際にオブジェクトバッファで不正なアクセスをチェックする（参照妥当性のチェック）。

このように例6と例7についてはシステム定義のデモン（イベント／コンディション／アクション）を使ってエレガントに指定する。決まり切ったことは，システムにやらせ応用依存のことはユーザに記述さ

せるという立場を，我々はとる。

3.2 制約ルール

ここではエンジニアリング分野で有効性を確認することのできた設計ゴール〔ISHI91a〕を一般化した制約ルールについて，その機能を説明する。前述したデモンが値の変更の伝播を主たる目的としていたのに対して，この制約ルールは値の生成を主たる目的とする。制約ルールは第一級のオブジェクトである。構文を以下に示す。

```
Name Parameter
Parameters Parameter, ...
GenerateMethod generatemethod
Conditions ConditionClause ...
IncreaseMethod increasemethod
DecreaseMethod decreasemethod

generatemethod ::= generate(init, cond, dif) ¦ calculate (exp) ¦ retrieve(db, cond, order) ¦ ask()
ConditionClause ::= Condition advice Action or Action ...
inceasemethod ::= GenerateIncr ¦ RetrieveIncr ¦ AskIncr ¦ obj.increase ¦ obj.decrease
decreasemethod ::= GenerateDecr ¦ RetrieveDecr ¦ AskDecr ¦ obj.increase ¦ obj.decrease
```

Nameはルール名であり，このルールが決定するパラメータ（属性）名を指定する。Parametersはこのルールが依存するパラメータを示す。複数指定できる。GenerateMethodとしては制約条件を満足する候補値を生成する手段を指定する。それには値の生成，計算，データベース検索，ユーザ入力などの手段を持つ。Conditionsには複数の条件節（ConditionClause）が指定できる。条件節は制約条件とそれが満たされなかった際に起動されるアクションが複数指定できる。アクションには自分自身もしくは他の制約ルールの起動が指定できる。先頭からはじめてアクションが成功すれば，それ以後のアクションは実行されない。アクションはルールのバックトラックの指定に対応する。条件節が複数指定されている場合はすべてのコンディションが満たされたときにこのルールは成功する。さもなければこのルールはアボートされる。ルールの起動の順序はルールの依存関係に基づき，ルールのスケジューラが決定する。イベントはルール起動そのもの（generate，increase，decrease）であり，条件節にコンディションとアクションが対応する。イベントはシステム定義であるが，コンディションとアクションがユーザ定義である。

例を使って説明する。

```
Name            PistonHeadThickness
Parameters      ExplosionPower, CylinderDiameter, PistonDiameter
GenerateMethod  generate(init: 3.7, condition: Value < 3.9, dif: 0.01)
Condisions      (1)Value>0.06*CylinderDaimeter advice self.increase
                (2)Value<0.065*CylinderDiameter advice self.decrease
                (3)pow(PistonDiameter,2)*ExplosionPower/pow(value,2)*4<80.0
                   advice self.icrease¦PistonDiameter.decrease
```

この例ではGenerateMethod は値の生成である。コンディションは３つ指定されている。

3.3　セマンティクス

ここでは完全性制約とトリガ機能のためのデモン及び制約ルールの意味をトランザクションの概念で説明する。トランザクションはデータベースの一貫性のある状態から次の一貫性のある状態への遷移を作り出す。デモンについては以下のように表わされる。

C event condition action C'

ここにCとC'は一貫性のあるデータベースの状態を示す。constraintデモンやmultiple, mandatoryのようなシステム定義デモンに相当するものはコンディション（constraintデモンのみユーザ定義）が成立すれば，CからC'へ遷移即ち，トランザクションはコミットされる。コンディションが満たされないとトランザクションはアボートされる。すなわちCのままである。if-needed, if-added, if-deleted, if-replacedデモンの場合は，コンディションが成立するとアクションが実行され，CからC'へ遷移する。特に後３者の場合，イベントの効果を補償するようなアクションを実行し，CからC'へ遷移する場合がある。その場合C'がCに意味的に等価になる。

制約ルールについても同様であり，コンディションを満足すればトランザクションをコミットしてイベントの状態を反映させるし，コンディションが満たされないとそのトランザクションをアボートして（正確にはイベントの効果を補償するようなアクションを実行して），別のルールを起動する。もちろん１つのルールがアボートされるのであってルール全体がアボートされるのではない。

ここでいうトランザクションはいわゆるshort transactionであり，それに伴う一貫性は応用独立の一貫性（あるいは局所的一貫性）である。そのトランザクションの列もしくは木構造からいわゆるlong transactionあるいはnested transactionが構成される。このトランザクションに伴う一貫性は応用依存の一貫性（大域的な一貫性）である。応用はこの応用依存の一貫性を達成することが目的である。nested transactionにおいて，それ以上応用依存の一貫性を達成できる見込がなくなった時に，応用はnotify, alertを起動することになる。すべてのルールのイベントが成立したら，即ちandイベントで応用依存の一貫性は達成される。デモンの場合は局所性に注目しており，制約ルールについては全体性に注目している。いうなればデモンは微分的ルールに対応し，制約ルールは積分的ルールに対応する。最後に注意点を述べておこう。さきの式でイベント／コンディション／アクションは別々のトランザクションである場合がある。よってCからC'への遷移の間にも応用独立の一貫性状態がありうる。それがトランザクションのカップリングモードに対応し，評価モードとの組み合わせが考えられる〔GATZ91〕〔MCCA89〕。

4．実現

4.1　システムアーキテクチャ

基本システムであるJasmineはオブジェクト管理とデータ管理の２つのサブシステムからなる。データ管理では非正規化テーブルとそのアクセスメソッドを示す。アクセスメソッドとしては順アクセス，B-tree, hashを提供する。インデックスもB-treeまたはhashテーブルを用いて実現できる。ジョインはnested loopとインデックスを利用したtid-joinのほかにhash joinを提供する。インタフェースとしてselect, join, insert, union, difference, intersection, nest, unnest等を持つ。このレイヤではSQLおよびその最適化機能を

提供しない。条件や検索／操作は，それぞれ前述のインタフェースのパラメータとしての述語関数と操作関数として指定する。それはデータ管理で提供するページバッファ上での操作に変換される。述語関数がtrueの時に限り，操作関数は実行される。さらにトランザクション管理機能を提供する。オブジェクト管理ではオブジェクトの定義操作のインタフェースを与える。集合論的な問い合わせ・操作機能並びにプログラミング機能を提供する。問い合わせではメソッドを起動できるし，メソッドを問い合わせで定義できる。問い合わせの結果を集合オブジェクト変数にセットしてスキャンしながら操作もできる。オブジェクト管理で問い合わせの最適化を行なう。メソッド，オブジェクト式，クラス階層を含む問い合わせを行なう。さらにメモリ内のオブジェクトの効率的アクセスのためにオブジェクトバッファを提供し，ページバッファから必要に応じてオブジェクトをキャッシュし，プログラムからのアクセスを速くする。オブジェクトバッファは，内部ハッシュテーブルであり，そのエントリにはメモリ内オブジェクト表現，OID，TIDを持つ。

4.2 完全性制約とトリガ機構

ここではシステム定義／ユーザ定義デモンの実現について述べる。multipleの場合次のようなコードがinsertの述語関数の中に埋め込まれる。即ちmultipleの場合はtrueを返す。もしなければ（単一値の場合は），値がない時に限りtrueをかえすようにする。mandatoryの場合は，挿入はinstantiate時のみ可能である。削除はできない。参照と置換はできる。そこでinstantiate時とdelete時にそれぞれシステムでチェックする。値によらずにチェックできる。constraintについてはinsertとreplaceの述語関数にコンパイルされる。述語関数がtrueのときのみ操作関数は実行される。

if-needed，if-added，if-deleted，if-replacedの場合，コンディションとアクションはそれぞれ，select，insert，delete，replaceの述語関数と操作関数にコンパイルされる。述語関数と操作関数はページバッファ上でデータにアクセスすることができ，ページバッファと応用プログラム間での不要なデータアクセスを減らして効率的な処理を許す。以上のべた実現方式は集合的なデモンの起動の場合である。単一の場合はオブジェクトバッファ上でコンディションとアクションは評価される。このように評価も使い方にあわせて行なう。

トリガの評価順序にはforward chainingとbackward chainingの２つがある〔STON90〕。２つのオブジェクトの属性AとBがあり，AがBによって決まるとする。通常，更新はBしかできない。Aの参照がたまでBの更新が頻繁であるなら，Aにif-neededデモンを使う。もしAの参照が頻繁でBの更新が少なければ，Bのほうにif-added，if-deleted，if-replacedデモンを指定する。前者がbackward chainingに後者がforward chainingに相当する。Jasmine/Aではシステムでなくユーザが評価順序を制御する方式をとる。なぜなら最適な使い方をユーザがよく知っていると考えるからである。もしBが更新がないならAにif-neededデモンを使う。もしAが更新可能であれば，Aのほうにif-added，if-deleted，if-replacedデモンを指定する。そのときはアクションにBを更新する操作を指定する。併せて通常はBのデモンの起動をサプレスする。

4.3 制約ルール

ルールはオブジェクトの属性を決定していく。その決定順序を属性即ちルール間の依存関係に基づいて決定していく。そのグラフを依存ネットワークという。制約充足はこの依存ネットワークに従って，ルートを上にして上から下へ通常は進む。１つのルールが成功すると次のルールが起動される。一つのルールはshort transactionであり，局所的一貫性を達成する。ルール全体でlong transactionを形成し，大域的一貫性を達成する。推論過程でのバックトラックはshort transactionのundoを行ない，別のshort transaction（ルー

ルのadviceで指定される）を起動する。依存ネットワークについてはループの検出を行なう。ルールは一般性のためにgenerate and testを用いる。実行時にループ検出も行なう。

5．おわりに

この論文では，アクティブデータベースについて，オブジェクト指向データベースのコンテクストでその制約管理機能を中心に，意味と実現を述べてきた。ここでは今後の課題について触れる。制約に関しては機能の汎用性を向上させるために制約言語の一般化，特に充足手段の多様化として他のオブジェクト指向データベースやリレーショナルデータベースとの連携も必要になるであろう。その意味でオブジェクト指向データベースを中心とした異種データベースの研究が重要になる。

参考文献

〔AGRA89〕 Agrawal, R. et al.: ODE: The language and the data model, Proc. the 1989 ACM-SIGMOD conference (1989).

〔AOSH90〕 Aoshima, M. et al.: The C-based Database Programming Language Jasmine/C, Proc. the 16th International Conference on Very Large Data Bases (1990).

〔DATE90〕 Date, J.C. : An Introduction to Database Systems, vol. 1, Addison-Wesley, 1990.

〔GARC87〕 Garcia-Monlia, H. et al.: SAGAS, Proc. the 1987 ACM-SIGMOD conference (1987).

〔GATZ91〕 Gatziu, S. et al. : Integrating Active Concepts into an Object-Oriented Database System , Proc. the 3rd int. Workshop on Database Programming Languages (1991).

〔HUDS89〕 Hudson, S. et al: Cactis: A Self-Adaptive, Concurrent Implementation of An Object-Oriented Database Management System, ACM Trans. Database Syst. , vol 14., no.3(1989).

〔ISHI91a〕 Ishikawa, H. et al. : An Object-Oriented Database : System and Applications, Proc. the IEEE Pacific Rim Conference on Comm., Computers, and Signal Processing (1991).

〔ISHI91b〕 石川，泉田，川戸：エンジニアリング業務支援とオブジェクト指向データベース，情報処理学会，情報処理，vol. 32, no. 5 (1991).

〔ISHI91c〕 Ishikawa, H. et al. : The Model, Language, and Implementation of An Object-Oriented Multimedia Knowledge Base Manegement System, accepted for publication in ACM Trans. Database Syst. (1991).

〔ISHI92〕 Ishikawa, H. et al. : An Object-Oriented Database System and its View Mechanism for Schema Integration, Proc. the 2nd Far-East Workshop on Future Database Systems(1992).

〔LOHM91〕 Lohman, G. et al.: Extensions to Starburst: Objects, Types, Functions, and Rules, Comm. ACM, vol.34, no.10 (1991).

〔MCCA89〕 McCarthy, D. et al.: The Architecture of An Active, Object-Oriented Database System, Proc. the 1989 ACM-SIGMOD conference (1989).

〔MORG83〕 Morgenstern, M.: Active databases as a paradigm for enhanced computing environments, Proc. the 9th VLDB conference(1983).

〔STON90〕 Stonebraker, M. et al.: On Rules, procedures, caching and views in database systems, Proc. the 1990 ACM-SIGMOD conference (1990).

〔TANA92〕 田辺他：定型業務支援システムの検討，人工知能学会全国大会(1992).