データベースシステム 111－5
（1997. 1. 21）

長期トランザクションにおけるチェックイン・チェックアウトの自動化と短期トランザクションの実行順序の保障方式

鬼塚 真, 磯部 成二

NTT 情報通信研究所

設計業務や分析業務では手戻りが頻繁に起きやすいため，ある時点での一貫性のあるデータをデータベース管理システムが自動的に保持することが要求される．また業務処理の手順が決定している場合は，この手順をデータベースで管理することによってデータの一貫性を保障することも要求される．前者の要求を実現するため，本稿では共用データベース上に，オブジェクトの状態とオブジェクト間に関する静的制約を定義することによって，チェックイン・チェックアウトを自動化する方式を提案する．一方後者の要求を実現するため，事象の事前条件・事後条件をデータベース上に定義することによって，長期トランザクションを構成する短期トランザクションの実行順序を保障する方式を提案する．

Management of Short Transactions Execution Order and Automatic Check in/out in Long Transactions

Makoto ONIZUKA, Seiji ISOBE
email:{onizuka,isobe}@dq.isl.ntt.co.jp

NTT Information and Communication Systems Laboratories

We present two mechanisms of long transaction for engineering database management systems, one is to automate check in and check out control and the other is to insure the execution order of short transactions that compose a long transaction. The former mechanism automates the control of transferring objects from group database to private database and vice versa by the static constraints of objects. The latter mechanism insures the short transactions execution order by the dynamic constraints which manage pre-condition and post-condition of each event in short transactions.

1 まえがき

近年，CADシステムやソフトウェア開発などでの計算機利用の高度化に伴い，設計業務のためのデータベースが重要視されてきている．設計業務では従来のOLTP業務とは異なり業務が長期間に渡るため，設計業務のためのデータベース管理システム(以降 DBMS) は長期トランザクションの機能が重要である．しかし，現状の長期トランザクションの機能には以下の問題がある．

・**チェックイン・チェックアウトの自動化が不十分** 長期トランザクションを実現するには，共用データベース(group database)と個人データベース(private database)を用意し，それらの間でデータを受け渡す(チェックイン・チェックアウト)方法が主流[1][2]である．このタイプの長期トランザクションでは，チェックイン・チェックアウトのタイミングが重要であり，一定の制約を課した状態でデータを共用データベースへチェックインすることが要求される．しかし，従来の DBMS ではチェックイン・チェックアウトのタイミングをアプリケーションまたはユーザで管理していたため，アプリケーションの実装が複雑になってしまったり，またデータの正しさがユーザの操作に任されてしまっていたという問題があった．

例えば，データベースの一部を変更するため，データを共用データベースから個人データベースへチェックアウトし，このチェックアウトしたデータを一部変更したと想定する．この変更により，もし変更したデータと他のデータとの間にある制約が満たされなくなった場合，このデータをチェックインすべきでないが，現在の DBMS ではデータがチェックインされるべきか否かを判定できない．

・**短期トランザクションの実行順序の保障が不十分** OLTP用の DBMS ではトランザクションによって，トランザクションを構成する個々の処理の実行順序の保障・データのロック・原子性(atomicity)を実現していた．一方，設計業務のための DBMS では，長期トランザクションという概念を導入することによって，本来トランザクションであるものを長期トランザクションとして扱い，この長期トランザクションをより小さい単位に分割したものを短期トランザクションとして扱っている．この結果，短期トランザクションによって個々の処理の実行順序の保障・データのロック・原子性を実現し，一方長期トランザクションによってデータの長期ロック(永続ロック)と長期の原子性を実現している．しかし，長期トランザクションは個々の短期トランザクションの実行順序を保障していない．なぜならば，長期トランザクションが短期トランザクションに分割されたため，各短期トランザクションの実行順序の情報が失われてしまうからである．

例えばデータベース設計業務では，最初にテーブル名・カラム名を標準化し，次に第一正規化，第三正規化する，などのように各処理の実行順序が決まっているが，DBMS 側でこの情報を持たないため各処理の実行順序を保障できない．この結果，各処理の実行順序はアプリケーション側で実装するか，またはユーザの操作に任されてしまっていたという問題があった．

上記の2つの問題を解決するため，本稿では以下の方針に基づく方式を提案する．

・**静的制約によるチェックイン・チェックアウトの自動化** オブジェクトの状態や複数のオブジェクト間の関係にどのような静的制約があるかを定義し，この定義を用いて DBMS がデータの整合性を自動的に判定し，チェックイン・チェックアウトを自動制御する．

・**動的制約による短期トランザクションの実行順序保障** 事前条件・事後条件を用いて表現したオブジェクトの状態遷移を動的制約として定義し，この定義を用いて DBMS が 短期トランザクション間の実行順序を決定することにより，長期トランザクションを構成する短期トランザクションの実行順序を保障する．

本稿の構成を示す．2で関連研究について述べる．3で本稿での議論に必要なクラスとオブジェクト等について定義する．4でチェックイン・チェックアウトを自動化するために導入した静的制約について定義し，この定義に基づくチェックイン・チェックアウトの自動化規則について述べる．5で長期トランザクションを構成する短期トランザクションの実行順序を保障するために導入した動的制約について定義し，この定義に基づく短期トランザクションの実行順序の保障方式について述べる．6で本手法の有効性についてまとめる．

2 関連研究

長期トランザクションに関連する研究として，共用データベース上に領域一貫性制約を導入したデータ管理モデルに関する研究 [3] が挙げられる．この研究では，共用データベースが制約を満たさない状態になることを許した上で，複数の設計者が異なるデータを更新する際に，これらのデータ間の制約を満たす制御モデルについて論じている．例えばデータ A, B にまたがる制約が定義されており，ある設計者が A を更新してこの制約が満たされなくなった場合は，制約を満たすような子トランザクションを生成して別の設計者にこのトランザクションを委任する．一方，本手法は共用データベースは常に定義された制約を満たすことを目的する．

動的制約に関連する研究については，OMT[4] などの状態遷移図を用いる方法や Eiffel[5] の事前条件・事後条件を用いる方法，また Eiffel の事前条件・事後条件をデータベースの動的制約表現に応用した研究 [6] などが挙げられる．本手法は Eiffel の事前条件・事後条件を応用して，長期トランザクションを構成する短期トランザクションの実行順序を保障するものである．

また，長期トランザクションにおけるチェックイン・チェックアウトの自動化と動的制約の両方に関連する研究として，ECA ルールを用いてデータベース自身が自立的に一貫性を保障するアクティブデータベースに関する研究 [7][8] などが挙げられる．ECA ルールを用いることによって，データベースが常に正しい状態，つまりチェックインされた状態であることを保障できると考えられるが，設計業務においては全ての業務をルール化するのは困難である．これに対し本手法は，データベースに正しい状態 (チェックインされた状態) と正しくない状態 (チェックアウトされた状態) の両方があることを許した上で，できるだけデータベースを正しい状態に維持することを目的とする．その実現手段の 1 つが静的制約によるチェックイン・チェックアウトの自動化であり，もう 1 つが動的制約によるトランザクションの実行順序の保障である．

3 Preliminaries

この章では，後の章での議論に必要なクラス，オブジェクト，外延，関連について定義する．後の章では継承を必要としないため，ここでは省略する．

クラス クラス (*class*) はオブジェクトの型であり，名称 (*className*)，属性 (*attribute*) の集合，メソッド (*method*) の集合とから構成される．また属性は，名称 (*name*) と型 (*type*) からなる．

$$class = \langle className, classID, Set(attribute), Set(method) \rangle$$
$$attribute = \langle name, type \rangle$$

但し，$Set(v)$ は v の集合を意味し，$(Set(v))[n]$ (但し$n \geq 0$) という操作によって n 番めの要素を参照することとする．

オブジェクト オブジェクト (*object*) はその状態を表現する属性値 (*value*) の集まりからなる．オブジェクトの属性値は *object.attribute* として，またオブジェクトのメソッドの返却値は *object.method* として参照する．

$$object = \langle objectID, classID, Set(value) \rangle$$

但し，*object* の $Set(value)$ の $n(n \geq 0)$ 番めの *value* と，$classID$ により決定できる *class* の $Set(attribute)$ の n 番めの *attribute* の *type* に対し，$dom(value) = type$ である．

外延 オブジェクトは生成の元となったクラスの外延 (*ext*) に属する．

$$ext : class \rightarrow Set(object)$$

但し，$dom(object) = class$ である．

例えば，クラス A の外延は $ext(A)$ として表現される．

関連 関連 (*relation*) は関連元のクラスから関連先のクラスへ定義され，具体的には関連元のクラスの外延に属するオブジェクトから関連先のクラスの外延に属するオブジェクト群への参照である．この参照を用いてオブジェクトを検索する関数としてトラバース (*traverse*) があり，同一の関数で逆方向の検索も可能であるとする．また，関連のタイプとして 1 対 1，1 対多，多対 1，多対多の 4 種類がある．

$relation = \langle class, class, reltype \rangle$
$traverse : relation \times object \rightarrow Set(object)$
但し，$object$ は $relation$ の定義で指定された 2 つのクラスのいずれかの外延に属する．
$reltype = \{OnetoOne, OnetoMany, ManytoOne, ManytoMany\}$

例えば，クラス A から クラス B へ定義された 1 対多の関連 rel は，$rel = \langle A, B, OnetoMany \rangle$ と表現される．また，クラス A の外延に属するオブジェクト a から関連 rel を用いてトラバースして得られるオブジェクト群は $traverse(rel, a)$ と表現される．

4 静的制約

この章では 1で述べた第一の問題である，チェックイン・チェックアウトの自動化が不十分であるということを解決するために導入した静的制約の定義と，それを利用したチェックイン・チェックアウトの自動化規則について述べる．

4.1 定義

静的制約は 1 クラスに閉じたオブジェクト内制約と，異なるクラスにまたがるオブジェクト間制約の 2 種類がある．それぞれについて以下で定義する．

4.1.1 オブジェクト内制約

オブジェクト内制約 ($internalConstraint$) は，クラスと制約関数 ($internalFunction$) からなる．オブジェクト内制約とはオブジェクト自分自身の状態またはメソッドの返却値を制約するものであり，一方の制約関数は制約が満たされているか否かを表現する真偽値 ($boolean$) を返却する．

$internalConstraint = \langle class, internalFunction \rangle$
$inernalFunction :\rightarrow boolean$

図 1 のクラス $CompositePart$ を用いてオブジェクト内制約の例を説明する．クラス $CompositePart$ は，更新日付を表す属性 $updatedate$ と現在の処理の進行状況を表す属性 $status$ を持つ．もし，任意のオブジェクト $comp1$ (但し $comp1 \in ext(CompositePart)$) の属性 $updatedate$ の値が 1997/1/23 より大きくなければならないという制約がある場合は，オブジェクト内制約 $con0$ がある．但し $con0$ は以下のように定義される．

$con0 = \langle CompositePart, func0 \rangle$
$func0 = (comp1.updatedate >= \text{“1997/1/23”})$

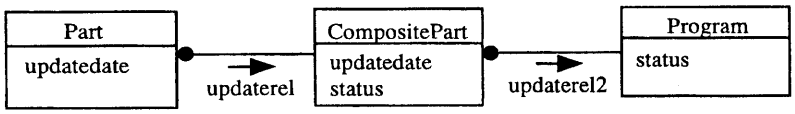

図 1: プログラム構成のオブジェクトモデル

4.1.2 オブジェクト間制約

オブジェクト間制約 ($externalConstraint$) は，関連と制約関数 ($externalFunction$) からなる．オブジェクト間制約とは関連元のオブジェクトを入力として関連先のオブジェクト群の状態またはメソッドの返却値を制約するものであり，一方の制約関数は制約が満たされているか否かを表現する真偽値 ($boolean$) を返却する．

$externalConstraint = \langle relation, externalFunction \rangle$
$externalFunction :\rightarrow boolean$

図1全体を用いてオブジェクト間制約の例を説明する．図1では，部品クラス $Part$ と集約クラス $CompositePart$ とプログラムクラス $Program$ があり，
$Part$ から $CompositePart$ へ多対1の関連 $updaterel = \langle Part, CompositePart, ManytoOne \rangle$ と，
$Compositepart$ から $Program$ へ多対1の関連 $updaterel2 = \langle CompositePart, Program, ManytoOne \rangle$ とがあることを示している．

もし，任意のオブジェクト $comp1$ (但し $comp1 \in ext(CompositePart)$) の属性 $updatedate$ の値が，$updaterel$ を用いてトラバースして得られる $traverse(updaterel, comp1)$ の個々のオブジェクトの属性 $updatedate$ の値の最大値でなければならないという制約がある場合は，オブジェクト間制約 $con1$ がある．但し $con1$ とは，

$$con1 = \langle updaterel, func1 \rangle$$
$$func1 = (comp1.updatedate == max((traverse(updaterel, comp1))[0].updatedate, (traverse(updaterel, comp1))[1].updatedate, ..))$$

である．

4.2 チェックイン・チェックアウトの自動化規則

チェックイン・チェックアウトの自動化規則の方針は，静的制約が満たされているデータは共用データベースに格納し，静的制約が満たされていないデータは個人データベースに格納するということであり，共用データベースと個人データベース間のデータのやりとりを，この静的制約に基づき自動化するというものである．この方針に基づくチェックイン・チェックアウトの自動化規則を以下に示す．

チェックアウト規則 あるトランザクション tr により共有データベース上のオブジェクト a ($a \in ext(A)$) の状態が更新されたと仮定する．

tr の実行後は $inFunc = false$ となる $con0 = \langle A, inFunc \rangle$ なるオブジェクト内制約が存在するか，
または tr の実行後は $exFunc = false$ となる $con1 = \langle rel, exFunc \rangle$ (但し $rel = \langle A, B, reltype \rangle$, B は任意のクラス，$reltype$ は任意の関連のタイプ) なるオブジェクト間制約が存在する場合，
もし a がチェックアウトされていないならば a をチェックアウトする．

チェックイン規則 あるトランザクション tr により個人データベース上のオブジェクト b ($b \in ext(B)$) の状態が更新されたと仮定する．

$con0 = \langle B, inFunc \rangle$ なる全てのオブジェクト内制約に対し，tr の実行後は $inFunc = true$ であり，
且つ $con1 = \langle rel1, exFunc1 \rangle$ (但し $rel1 = \langle A, B, reltype \rangle$, A は任意のクラス，$reltype$ は任意の関連のタイプ) なる全てのオブジェクト間制約に対し，tr の実行後は $exFunc1 = true$ であり，
且つ $con2 = \langle rel2, exFunc2 \rangle$ (但し $rel = \langle B, C, reltype \rangle$, C は任意のクラス，$reltype$ は任意の関連のタイプ) なるオブジェクト間制約が存在しないならば，b をチェックインする．

4.1 で説明した，オブジェクト間制約 $con1$，$con2$ を用いて，チェックイン・チェックアウトの動作を説明する (図2)．図2の意味は以下の通りである．

時刻 $t1$ は，オブジェクト $part1$，$part2$，$part3$ (但し $part1, part2, part3 \in ext(Part)$)，オブジェクト $comp1$，$comp2$ (但し $comp1, comp2 \in ext(CompositePart)$)，オブジェクト $prog$ (但し $prog \in ext(Program)$)，が存在し，オブジェクト間制約 $con1$，$con2$ は満たされている．

時刻 $t2$ は，時刻 $t1$ の $part1$ に更新操作を施した結果得られる状態を示しており，この更新操作の結果 $part1$ と $comp1$ の間の制約 $con1$ が満たされなくなったため，チェックアウト規則が適用されて $part1$ は個人データベースへチェックアウトされ，共通データベースでは更新前の $part1$ を格納する．

時刻 $t3$ は時刻 $t2$ の $comp1$ に更新操作を施した結果得られる状態を示しており，この更新操作の結果 $comp1$ と $prog$ の間の制約 $con2$ が満たされなくなったため，チェックアウト規則が適用されて $comp1$ は個人データベースへチェックアウトされ，共通データベースでは更新前の $comp1$ を格納する．更に個人データベース上の $part1$ と $comp1$ 間の制約 $con1$ は満たされるようになったとする．

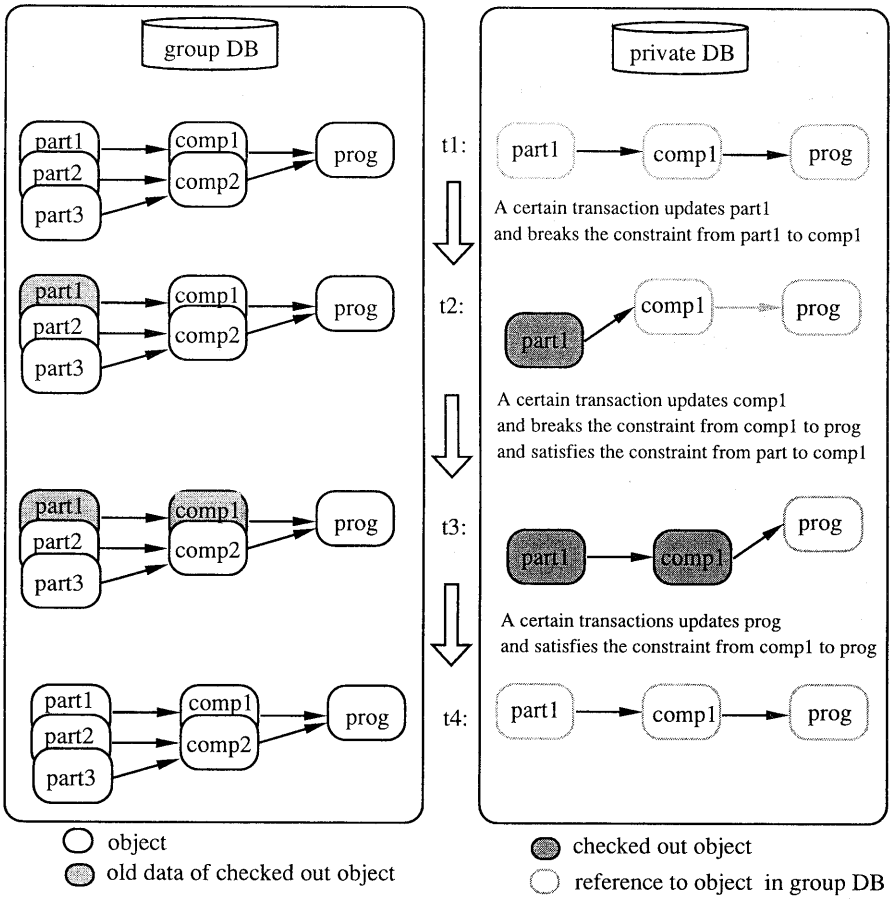

図 2: チェックイン・チェックアウトの動作例

時刻 $t4$ は時刻 $t3$ の $prog$ に更新操作を施した結果得られる状態を示しており，この更新操作の結果 $comp1$ と $prog$ 間の制約 $con2$ が満たされるようになったため，チェックイン規則が適用されて $comp1$ は共用データベースへチェックインされる．$comp1$ がチェックインされた結果，続けてチェックイン規則が適用されて $part1$ がチェックインされる．

5 動的制約

この章では 1で述べた第二の問題である，短期トランザクションの実行順序の保障が不十分であるということを解決するために導入した動的制約の定義と，それを利用した短期トランザクションの実行順序の保障方式について述べる．

5.1 定義

動的制約 文献 [4] での概念である事前・事後条件を応用し，動的制約を次のように定義する．

> あるクラス C の外延に属する任意のオブジェクトが特定の状態 S_0 にあり，その時に特定の事象 E が特定の入力 E_{in} を伴い発生した場合，そのオブジェクトは事象の特定の出力 E_{out} を伴い新たな状態 S_1 へ推移するとする．この事象の起きる前の条件 (対象オブジェクトの状態 S_0 と事象に対

する入力 E_{in} に関する条件) をその事象の事前条件と呼び，この条件が成り立たない場合は事象を実行しない．また上記の事象が終了する時に保証しなければならない条件 (対象オブジェクトの状態 S_1 と事象の出力 E_{out} に関する条件) をその事象の事後条件と呼び，この条件が成り立たない場合は事象の起きる以前の状態 S_0 にオブジェクトを戻す．これら事前条件・事後条件を合わせて，対象オブジェクトの動的制約と呼び，以下のように表現する．

$C : S_0 \cap E_{in} \xrightarrow{E} S_1 \cap E_{out}$

但しオブジェクトが存在しない場合，状態としては何も記述しない．

5.2 動的制約による短期トランザクションの実行順序の保障

動的制約を用いることによって，任意のクラスの外延に属するオブジェクトの1状態遷移を表現できる．この結果，動的制約の集合を用いることによって全てのオブジェクトの状態遷移を構成することが可能となる．この動的制約をデータベースで管理することは，全てのオブジェクトの状態遷移をデータベースで管理することになるため，1で述べた第二の問題 – 長期トランザクションは短期トランザクションという単位に分割されてしまっているため，各短期トランザクション間を接続する情報が失われてしまう – を解消できる．

具体例を図3に示す．この図3の意味は以下の通りである．長期トランザクション ltr があり，これは3つの短期トランザクション $tr1$，$tr2$，$tr3$ を順次実行することによって構成されている．また a，b，c はオブジェクトでありそれぞれ特定の相異なるクラス A，B，C の外延に属するものとする．

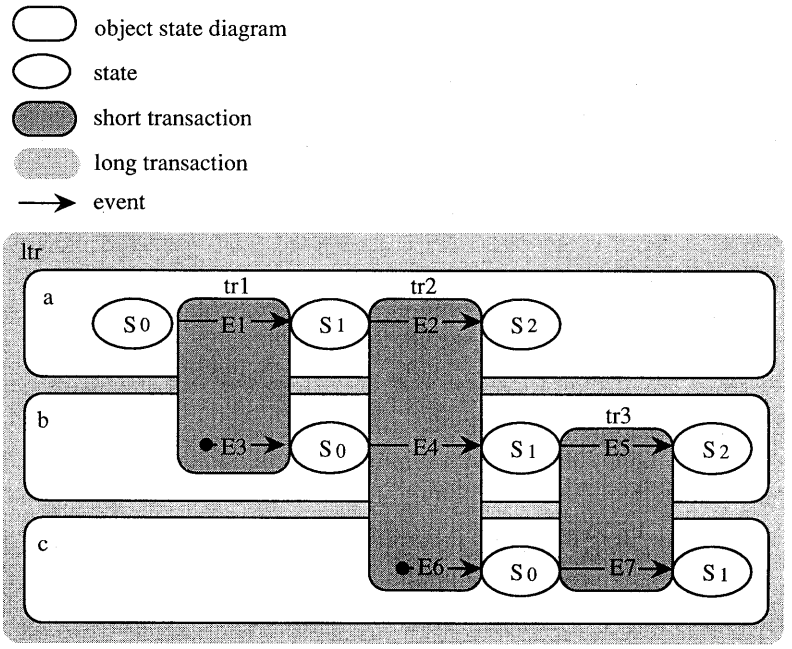

図 3: 長期トランザクションの例

$tr1$ は事象 $E1$，$E3$ から構成されていて，$E1$ は a の状態が S_0 の時に実行され，その結果 a の状態を S_1 にし，また $E3$ は実行されると b を生成してその状態を S_0 にする．

$tr2$ は事象 $E2$，$E4$，$E6$ から構成されていて，$E2$ は a の状態が S_1 の時に実行され，その結果 a の状態を S_2 にし，また $E4$ は b の状態が S_0 の時に実行され，その結果 b の状態を S_1 にし，また $E6$ は実行されると c を生成してその状態を S_0 にする．

$tr3$ は事象 $E5$，$E7$ から構成されていて，$E5$ は b の状態が S_1 の時に実行され，その結果 b の状態を S_2 にし，また $E7$ は c の状態が S_0 の時に実行され，その結果 c の状態を S_1 にする．

図3における動的制約を下記に示す.

A:$S_0 \stackrel{E1}{\rightarrow} S_1$, A:$S_1 \stackrel{E2}{\rightarrow} S_2$,
B: $\stackrel{E3}{\rightarrow} S_0$, B:$S_0 \stackrel{E4}{\rightarrow} S_1$, B:$S_1 \stackrel{E5}{\rightarrow} S_2$,
C: $\stackrel{E6}{\rightarrow} S_0$, C:$S_0 \stackrel{E7}{\rightarrow} S_1$

上記の動的制約から，事象の間には以下の処理順序 (→ で表現することとする) が規定される.

$E1 \rightarrow E2$, $E3 \rightarrow E4$, $E4 \rightarrow E5$, $E6 \rightarrow E7$

一方，図3における短期トランザクションと事象の関係は下記のように表現できる.

$tr1 = \{E1,E3\}$, $tr2 = \{E2,E4,E6\}$, $tr3 = \{E5,E7\}$

但し各式の左辺は短期トランザクションであり，右辺はそのトランザクションが起動する事象群である.

上述の事象の間の処理順序の関係と，短期トランザクションと事象の関係とから，短期トランザクション trn $(1 \leq n \leq 3)$ の間には以下の処理順序 (→ で表現することとする) が規定される.

$tr1 \rightarrow tr2$, $tr2 \rightarrow tr3$

このように各短期トランザクション $tr1$, $tr2$, $tr3$ は実行順序が制約されるので，$tr1$, $tr2$, $tr3$ から構成される長期トランザクション ltr の短期トランザクションの実行順序を保障できる.

6 むすび

本稿では設計系データベース管理システムにおける2つの方式，1つはチェックイン・チェックアウトの自動化であり，もう1つは長期トランザクションを構成する短期トランザクションの処理順序の保障方式，を提案した．前者は，オブジェクトの状態と複数のオブジェクトの関係とに対する静的な制約を用いて，チェックイン・チェックアウトを自動化するものであり，従来のDBMSの長期トランザクション管理機構と比較してアプリケーションの生産性の向上が期待できる．一方後者は事象の事前条件・事後条件である動的制約を用いて，長期トランザクションを構成する短期トランザクションの処理順序を保障する方式であり，従来のDBMSと比較してより信頼性の高いデータベースを保持することが容易になると考えられる．今後は，これらの方式を実装して有効性について定量的に評価する予定である.

References

[1] S. Ahmed, Albert Wong, Duvvuru Sriram and Robert Logcher, "Object-oriented database management systems for engineering:a comparison," JOOP, pp.27-44, June 1992.

[2] Versant Object Technology, "VERSANT object database management system release 3.0 C++/VERSANT usage guide,", April 1994.

[3] 春本 要，八幡 孝，西尾 章治郎，"協調作業支援のためのデータ管理モデル，" 信学論 (D-I)，Vol.J79-D-I，no.5，pp.271-279，May 1996.

[4] J.Rumbaugh, M.Blaha, W.Premerlani, F.Eddy and W.Lorensen, "Object-oriented modeling and design," Prentice-Hall, New York, 1991.

[5] B. Meyer, "Eiffel: the language," Prentice-Hall, New York, 1992.

[6] H.Martin, M.Abida and B.Defude, "Consistency checking in object oriented databases: a behavioral approach," Proc. of the 2nd Int. Conf. on Information and Knowledge Management, pp.53-68, 1992.

[7] U.Dayal, "Active database systems, " Proc. of the 3rd Int. Conf. on Data and Knowledge Bases, Jerusalem, Isral, June 1988.

[8] 石川 博，酒井 洋一，"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，" 情処学データベースシステム研報，89-10，July 1992.