データベースシステム 120－8
（2000．1．24）

一貫性情報を用いたマルチバージョン並行処理制御

徐 海燕　　古川 哲也　　史 一華
福岡工業大学　九州大学　福岡工業短期大学

ワークフローによって表される作業手順に関してデータベースが一貫するための統一的な判断基準が、協調型データベースで果たす役割について検討する。データ間の一貫性情報を利用できれば、処理単位やデータベースのための一貫性管理要求を形式的に定義できる。本論文では、作業手順によって表される一貫性基準に対して、一貫性管理の問題を2つのデータ項目間の問題に帰着でき、実用的な方法で一貫性管理ができることを示す。マルチバージョンに対しても、同様な一貫性管理を行える。一方、協同作業を行う場合には、利用者は一貫する古いバージョンよりも最新バージョンを利用したい場合が多く、関連するデータに対する操作による影響も考慮すべきである。一貫性情報の利用はこのような協同作業の立場からの要求を支援するためにも役立つことを示す。

Multiversion Concurrency Control Utilizing Consistency Knowledge

Haiyan Xu　　Tetusya Furukawa　　Yihua Shi
Fukuoka Institute of Tech.　Kyushu University　Fukuoka Junior College of Tech.

In this paper, we discuss the effect on cooperative databases by consistency information such as workflows represented by graphs. Using the consistency information, both the consistency management requirements from transactions or database systems can be defined formally. The consistency information represented by graphs, furthermore, makes the consistency management efficiently. Even if multi-versions are recorded, the consistency information lets users utilize old versions explicitly. The consistency information is also useful to satisfy the both requirements of using the latest versions and avoiding rollback of a whole transaction.

1. はじめに

原子性、整合性、隔離性、耐久性というACID特性を満たす処理単位は、OLTPなどの応用分野で重要な役割を果たしてきた[3]。一方で、データベースの応用分野の拡大に伴って、処理単位を協同作業支援に用いる必要性が高まってきている[2),5)]。それを実現するためには、新しい応用分野の特徴に対処できなければならない。本論文では、それらの特徴に基づいた並行処理制御について議論する。

複数の利用者が協同作業を行う際に、データベースに要求される機能として、一貫性の管理がある。具体的には、処理単位の立場とデータベースの立場からの次のような一貫性管理が必要となる。

- 検索結果の一貫性：各々の処理単位は、一貫性が取れているデータを検索する。
- 処理単位の一貫性：各々の処理単位が単独実行される場合、データベースの一貫性は保持される。
- スケジュールの一貫性：複数個の処理単位が並行実行される場合、データベースの一貫性は保持される。

知的設計作業では、複数のバージョンを記憶・管理・利用できなければならない。すなわち、利用者が必要に応じて各種のバージョンを利用しても、上記の一貫性を管理する必要がある。

また、協同作業を支援するために、処理単位の後退復帰によって操作結果が紛失される事態は、極力避けなければならない。さらに、最新の作業結果を参照できるなど、協同作業に対する制限は行わず、関連するデータに対する操作によって生じる互いへの影響はできるだけ早い段階で検出し、利用者に知らせることが望まれる。

従来は、処理単位の一貫性は利用者の責任において、スケジュールの一貫性と検索結果の一貫性は直列可能性基準に基づく並行処理制御方式によって保証されてきた[3]。しかし、このような並行処理制御

では、協調作業支援のための要求を満足することができない[2]。この問題は広く認識されており、様々な研究がなされている[2),5),9)]。しかし、どの方式も上記の一貫性管理、マルチバージョン管理、協同作業支援という3つの要求をすべて満たすことはできない。

一方、オフィスでの協同作業や高度なビジネスプロセスをモデル化し、実行または制御するための有効な技術として、ワークフロー管理システムが注目されており[4)]、商用、工業、経営管理、ローン計算等の分野で実用的なシステムが開発されている。

本論文では、ワークフローというグラフ形式で表される作業手順に注目し、作業手順によって定義される状態の変化規則を、データベースが一貫するための統一的な判断基準と見なす。そして、一貫性情報の利用が、協調型データベースで果たす役割について検討する。

一貫性情報を利用できれば、まず、一貫性管理要求を形式的に定義できる。そして、グラフで表される一貫性情報に対して、実用的な方法で一貫性管理問題を実現できることを示す。さらに、複数のバージョンを記憶する場合でも、同様な一貫性管理を行えることを示す。一方、協同作業において、利用者は一貫する古いバージョンを利用するよりも最新のバージョンを利用したい場合が多く、それによって生じる処理単位の後退復帰という長時間に渡る作業結果の紛失は極力避けなければならない。また、関連するデータに対する作業による影響も視野に入れなければならない。作業手順という一貫性情報の利用は、このような協同作業の立場からの要求に対応するためにも役立つことを示す。

本論文は、次のように構成される。2章では、ワークフローに対する一貫性情報を、複数のバージョンを記憶するデータベースに適用した場合の基本的事項について記述する。3章と4章は、それぞれワークフローに対する一貫性情報が利用できた場合に、どのように一貫性管理と協同作業支援に役立つかについて検討する。5章は従来の研究との比較であり、6章は全体のまとめである。

2. 一貫性情報

本章では、ワークフローで表されるデータベースが一貫するための統一的な基準を形式的に定義し、それのマルチバージョンデータベースの一貫性管理における役割についてまとめる[10)]。

定義1 ワークフローは、作業手順を示す有向非巡回グラフ $WF(D,E)$ である。節点 $x \in D$は作業過程で作成、変更、利用されるデータ項目である。枝 $(x,y) \in E$はデータ項目 yに関する作業は、データ項目 x に関する作業が完成された後で行わなければならないことを意味する。 □

ワークフローのモデル化にはいくつかの方法が存在するが、本論文では、データに焦点を当てる方法を使用している。また、議論を単純化するため、$WF(D,E)$ は連結グラフと仮定する。

例1 ソフトウェア開発において、作業は仕様、ソースプログラム、目的プログラム、テストという順に行われるとする。それを表すワークフローは、図1となる。 □

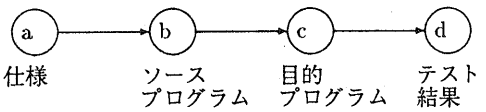

図1 ワークフロー $WF_1(D_1,E_1)$

ある作業を完成するためには、利用者は繰り返して結果を修正することがあり、一部の古いバージョンは継続的に利用されることもある。このため、マルチバージョンデータベースを用いてデータ項目の各々のバージョンを記憶する。データ項目 x のバージョンは x_k, x_j,$\cdots$というように記述する。ここで、k,jといった添字はそのバージョンを変更した処理単位のインデクスである。

すべての作業は一回で成功するとは限らない。例えば、目的プログラムを生成するためのコンパイル操作は失敗に終ることがあり、目的プログラムが仕様で定義される通りに動くとも限らない。このような作業情報を記憶するために、各バージョン x_jは、実体部分 $e(x_j)$ と状態部分 $st(x_j)$ から構成される。$e(x_j)$ の値は、ソースプログラムや目的プログラムといったバージョンのデータ部分である。$st(x_j)$ の値は、$e(x_j)$ の状態を表す真理値である。$e(x_j)$ が完成したデータの場合は $st(x_j)$ の値は真であり、それ以外の場合は偽である。

定義2 ワークフロー $WF(D,E)$ に従って作成された結果を記憶するマルチバージョンデータベース $MDB(MV,MR)$ は、バージョン集合 MVと関連集合 MRから構成される。$x_i \in MV$はインデクス番号 iである処理単位によって作成されたデータ項目 x のバージョンである。関連 (x_i,y_j) は、(x,y) という枝が $WF(D,E)$ にあり、バージョン y_jがバージョン x_iに従って作成されていることを意味する。 □

例2 ワークフロー $WF_1(D_1,E_1)$ に従う作業が、図2で示されたようにそれぞれ処理単位 T_1、T_2、T_3に

よって行われていたとする。

作業結果を記憶するMDB_1は次のようになる。

$$MV_1 = \{a_1, b_2, b_3, c_2, c_3, d_1, d_3\}$$

$$MR_1 = \left\{ \begin{array}{l} (a_1, b_2), (a_1, b_3), (b_2, c_2), \\ (b_3, c_3), (c_2, d_1), (c_3, d_3) \end{array} \right\}.$$

また、$st(d_1)$の値のみが偽であり、その他のバージョンの状態は全部真であるとする。 □

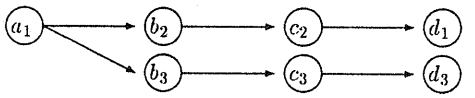

図2 マルチバージョンデータベース $MDB_1(MV_1, MR_1)$

ワークフロー$WF(D,E)$に従って作業が進んでいれば、ワークフローインスタンスが構成される。

定義3 ワークフロー$WF(D,E)$に従う$MDB(MV,MR)$におけるワークフローインスタンス(V,R)は、バージョンを節点とする有向グラフである。節点集合Vは、Dの異なるデータ項目に属するバージョンからなり、かつ次の条件を満たす。

- $y_j \in V$に対して$(x,y) \in E$ならば、$(x_i, y_j) \in MR$である$x_i \in V$が存在し、かつx_iの状態は真である。
- $(x,y) \in E$である$x_i \in V$に対して$\nexists y_j \in V$ならば、MRには(x_i, y_k)というような関連は存在しない。

すべてのワークフローインスタンスの集合を$I_{WF}(MDB)$で表す。 □

例3 $MDB_1(MV_1, MR_1)$と$WF_1(D_1, E_1)$に対するワークフローインスタンス$I_{WF_1}(MDB_1)$は、図3となる。 □

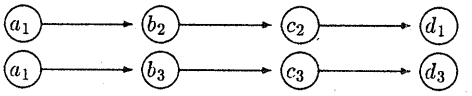

図3 WF_1のワークフローインスタンス$I_{WF_1}(MDB_1)$

各ワークフローインスタンスにおいて、$(x,y) \in E$に対して、yのバージョンはxの1つのバージョンに対応する。逆に、例3の$(a,b) \in E_1$のようにxのバージョンは複数個のyのバージョンに対応することがある。このため、MRにおいてxとyのバージョン間の関連は1対多になる。すなわち、yのバージョンを決めれば、対応するxのバージョンは一意に決められるが、その逆は成り立つとは限らない。

ワークフローインスタンスから、ワークフローに従って作成されるバージョン集合という情報が得られる。この情報を利用できれば、マルチバージョンデータベースの一貫性を定義できる[10]。

定義4 マルチバージョンデータベース$MDB(MV, MR)$がワークフロー$WF(D,E)$に対して一貫しているとは、MVのすべてのバージョンが$I_{WF}(MDB)$のいずれかのインスタンスに属することをいう。同様に同一のインスタンスに属するバージョン（の部分）集合は$WF(D,E)$に対して一貫しているという。 □

例えば、$MDB_1(MV_1, MR_1)$において、$\{a_3, b_2, c_2, d_1\}$は同一のインスタンスに属していないので、一貫していないことになる。

3. 一貫性管理

処理単位は、変更操作と検索操作の系列から構成される。本章では、それぞれ変更操作の一貫性問題と検索操作の一貫性問題について検討する。

3.1 変更結果の一貫性

変更結果の一貫性問題は、各々の処理単位が単独実行される場合にデータベースの一貫性を保持できるかどうかという処理単位の一貫性問題と、複数の処理単位が並行に実行される場合にデータベースの一貫性を保持できるかどうかというスケジュールの一貫性問題に分けられる。

作業手順というグラフによって表される一貫性情報に対して、処理単位の一貫性は、各々の変更操作が作業手順に従って行われているかどうかという問題に帰することができる。

定義5 ワークフロー$WF(D,E)$に対して一貫している$MDB(MV,MR)$において、$x_i \in MV$に対する$WF(D,E)$に従う変更操作は、次の条件を満たすものである。

(1) 枝$(y,x) \in E$が存在すれば、x_iの作業で利用されるバージョンy_jはすでに存在し、かつその状態は真である。

(2) $e(x_i)$に対する作業の終了後、状態$o(x_i)$およびx_iに関わる関連も同時に記録される。 □

利用者の変更操作は、条件(1)を満たしたものの実行のみを許可し、条件(2)はトリガーで満足することによって、ワークフローに従う変更操作とすることができる。

例4 図4は、$WF_1(D_1, E_1)$に従う変更操作からなる例2におけるT_1とT_2の並行実行である。

T_2はT_1が作成したa_1を参照しており、T_2はT_1が作成したc_2を参照しているので、この並行実行は直列可能ではない。 □

補題1 $WF(D,E)$に対して一貫しているマルチバージョンデータベース$MDB(MV,MR)$において、

T_1	値	T_2	値
$W_1(a)$	a_1		
		$R_2(a)$	a_1
		$W_2(b)$	b_2
		$W_2(c)$	c_2
$R_1(c)$	c_2		
$W_1(d)$	d_1		

図4　非直列可能実行

$x_i \in MV$に対する$WF(D,E)$に従う変更操作からなる処理単位の単独実行および並行実行は、マルチバージョンデータベースの一貫性を保持する。 □

証明： ワークフローに従う変更操作の条件は、ワークフローインスタンスを作成するための必要十分条件である。従って、処理単位の各々の変更操作がワークフローに従って作業すれば、処理単位の一貫性を保証できる。

一方、マルチバージョンデータベースにおいて、各々のワークフローに従う処理単位の並行実行も、ワークフローインスタンスを作成している。すなわち、マルチバージョンデータベースの一貫性を保持している。 □

補題1より、例4の並行実行はスケジュールの一貫性を満たす。従って、例4より、マルチバージョンデータベースの一貫性保持問題は、直列可能性と関係ないことが分かる。言い替えれば、補題1は、変更結果の一貫性管理のために、処理単位を後退復帰させる必要はなくなることを意味している。

3.2 検索結果の一貫性の判定と調整方法

処理単位は同じデータ項目の異なるバージョンを検索することがあるので、処理単位のある時点での検索結果を、各データ項目に対して、最後に検索されたバージョンからなる集合とする。また、同じデータ項目 x に対して T_iが複数回変更した場合、$x_{i_1}, x_{i_2}, \cdots$と記述する。

定義4より、処理単位の検査結果の一貫性は、検索結果に属するバージョン集合が同一インスタンスに属するかどうかの問題に帰する。従って、処理単位の検索結果の一貫性は判定可能となり、従来のように直列可能性に依存する必要がない。次に、各々の検索操作に対する検索結果の一貫性の判定と一貫していない場合の対処法について検討する。

R^{k-1}を処理単位Tの$k-1$番目の検索操作が実行された時点での検索結果とする。y_jを検索したk番目の検索操作$R(y)$に対して、定義4より、$R^{k-1} \cup \{y_j\}$の一貫性は、各$x_i \in R^{k-1}$に対してx_iとy_jの間にワークフロー$WF(D,E)$によって定められたxとyの関連が存在するかどうかを検査することで判定される。

例5 図5において、T_3とT_5は各変更操作が$WF_1(D_1,E_1)$に従う処理単位である。T_4とT_6は検索のみを行う処理単位である。

T_4	値	T_3	値
		$W_3(b)$	b_3
$R_4(b)$	b_3		
$R_4(c)$	c_2		
		$W_3(c)$	c_3

(a)

T_4	値	T_3	値
$R_4(b)$	b_2		
		$W_3(b)$	b_3
		$W_3(c)$	c_3
$R_4(c)$	c_3		

(b)

T_5	値	T_6	値
$W_5(a)$	a_5		
		$R_6(a)$	a_5
$R_5(a)$	a_5		
$W_5(b)$	b_{5_1}		
		$R_6(b)$	b_{5_1}
$W_5(b)$	b_{5_2}		

(c)

図5　非直列可能検索

明らかに、(a)、(b)、(c)のどの場合のスケジュールも直列可能ではない。しかし、(c)の場合は、T_5によって$\{a_5, b_{5_1}\}$および$\{a_5, b_{5_2}\}$という2つのインスタンスを作成している。T_6は$\{a_5, b_{5_1}\}$という同一インスタンスに属する結果を検索しているので、検索結果の一貫性を満たす。 □

検索結果の一貫性を満たす場合には、直列可能性であるかどうかに関わらず検索操作の実行が許可される。満たさない場合は、次のような機能によって調整する。

定義6 一貫性制御機能とは、一貫するR^{k-1}に対して、k番目のy_jを検索する$R(y)$によって$R^{k-1} \cup \{y_j\}$が一貫しなくなる場合に、次の2つの場合に分けて調整する機能である。

(1) R^{k-1}と一貫するyのバージョンと比べて、y_jが古い場合は、新しいバージョンを検索させる。一貫するバージョンがまた作成されてない場合には、$R(y)$の実行を待機させる。

(2) R^{k-1}と一貫するyのバージョンと比べて、y_jが新しい場合は、R^{k-1}と一貫しているyの古いバージョンを検索させる。 □

例6 例5の場合(a)と(b)に対して、それぞれ一貫性制御機能(1)と(2)の調整法が適用される。結果は、図6で示されるような実行となる。すなわち、(a)の場合は、$R_4(c)$が待機された後c_3を検索することになり、(b)の場合は、$R_4(c)$は一貫している古いバージョンc_2を検索することになる。 □

T_4	値	T_3	値
		$W_3(b)$	b_3
$R_4(b)$	b_3		
		$W_3(c)$	c_3
$\underline{R_4(c)}$	c_3		

(a)

T_4	値	T_3	値
$R_4(b)$	b_2		
		$W_3(b)$	b_3
$\underline{R_4(c)}$	c_2		
		$W_3(c)$	c_3

(b)

図6 調整された結果

一貫性制御機能は、一貫性情報を利用してR^{k-1}と一貫しているy_jを検索させている。従って、次の結論が得られる。

補題2 一貫性制御機能によって、調整された後のスケジュールにおいて、各処理単位は検索結果の一貫性を満たす。 □

補題2は、検索結果の一貫性管理のため、処理単位を後退復帰させる必要はなくなったことを意味する。補題1とまとめると、次のような結果が得られる。

定理1 ワークフロー$WF(D,E)$に従って作成された結果を記憶する$MDB(MV,MR)$において、一貫性管理のために、処理単位を後退復帰させる必要はない。 □

3.3 効率的な判定方法

3.2節では、検索結果の一貫性問題は判定可能であることを示した上で、判定結果を活用した調整方法を与えた。しかし、検索結果の一貫性判定が実用的な時間内でできなければ、3.2節の結果は理論的なレベルで終ってしまう。このため、本節では、判定問題の効率的な実現方法について検討する。

3.2節で述べたように、$R^{k-1} \cup \{y_j\}$の一貫性を判定する問題は、各々の$x_i \in R^{k-1}$とy_jの一貫性を判定する問題に帰着できる。バージョン間の関連の1対多である性質を利用すると、判定すべきバージョンの組を大幅に減らせる。

アルゴリズム1 R^{k-1}とy_jの一貫性判定法。

$(x,y)^*$はワークフロー$WF(D,E)$におけるxからyへの経路を表している。また、$k=2$の場合、*head-list*の初期値は空であるが、その後は、蓄積された計算結果になる。

```
NON = φ;
foreach x_i ∈ head-list do
begin
if (x,y)* ∈ E then
    if (x_i,y_j)* ∈ MR then
      begin
        head-list=(head-list − {x_i}) ∪{y_j};
        exit;
      end
    else begin NON = NON ∪ {x_i};end
  else if (y,x)* ∈ E
    then if (y_j,x_i)* ∈ MR then exit
      else begin NON = NON ∪ {x_i}; end
end
if NON = φ
then head-list=head-list∪{y_j}
```

$NON = \phi$ならば、R^{k-1}とy_jは一貫している。$\exists x_i \in NON$ならば、x_iはy_jと一貫していない。x_iを代表とするグループ中のその他のバージョンがy_jと一貫しているかどうかはさらに再帰的に検査することによって決める。 □

例7 a_1, b_2, c_2, d_1を順に検索する処理単位Tがあるとする。*head−list*は、b_2を検索する時点で$\{a_1\}$、c_2を検索する時点で$\{b_2\}$、d_1を検索する時点で$\{c_2\}$となる。 □

すなわち、R^{k-1}をグループに分け、$R^{k-1} \cup \{y_j\}$の一貫性問題を、y_jとそれぞれのグループの代表との一貫性問題に帰着している。

補題3 アルゴリズム1の一貫性判定は、正しく判定している。 □

証明： 2章で述べたように、$(x,y) \in E$に対して、MRにおいてxとyのバージョン間の関連はyのバージョンを決めれば、xのバージョンが一意に決まるという1対多の関係にある。従って、y_jとx_iが一貫していれば、y_jはx_iへの経路を持つグループ内のすべてのバージョンと一貫している。逆に、y_jとx_iが一貫していなければ、y_jはx_iと一貫せず、x_iを代表とするグループ内の他のバージョンとの一貫性は一概に決められないことになる。それも再帰的に判定しているので、補題が成り立つ。 □

ワークフローにおける経路$(x,y)^*$と経路の長さを記憶していれば、一貫性判定法は、その経路の長さの線形時間で実行できる。

定理 2 n 個のデータ項目に属するバージョンを記憶する $MDB(MV,MR)$ において、処理単位の各検索操作に対する検索結果の一貫性の判定時間は、$O(n)$ である。 □

証明： アルゴリズム 1 は、$k(k>1)$ 番目の y_j を検索する $R(y)$ の操作に対する検索結果の一貫性の判定を、y_j と R^{k-1} によって分けられた各々のグループの代表との一貫性問題に帰している。y_j と各グループの代表との一貫性判定は、最大 $O(n)$ で行える。一方、グループ数は定数と見なせるため、定理が成り立つ。 □

検索結果の一貫性は効率的に判定可能であり、判定結果を利用して一貫性を満たすように実行させる調整も可能である。また、本論文の一貫性制御では、$R(y)$ の検索対象となるバージョン y_j についていっさい制限を設けていない。従って、利用者が明示的に古いバージョンを利用したければ、一貫性上で問題さえなければ、利用できる。これも従来のマルチバージョン並行処理制御方式との違いの 1 つである。

4. 協同作業支援

ここまでは、一貫性管理を中心に検討してきたが、協同作業を行う場合に、利用者は一貫する古いバージョンを利用するよりも、最新のバージョンを利用したい場合が多い。また、関連しているバージョンを操作している場合は、データベースシステムがバージョン更新に関する情報を把握し、関連する利用者に知らせることによって協同作業の効率化を計ることも重要である。

4.1 最新バージョンの検索

例 5の (b) では、$R_4(c)$ が最新バージョン c_3 を検索しようとしたが、検索結果の一貫性を満たさないため、古いバージョン c_2 を検索するように調整されている。しかし、このような調整は、必ずしも利用者の要求を満たすとは限らない。利用者は一貫する古いバージョンを利用するよりも、最新のバージョンを利用したい場合が多い。

ワークフローに関する一貫性情報を利用できれば、利用者の最新バージョンを検索したいという要求は、一部のデータ項目を検索し直すことによって満たせる。

定義 7 最新のバージョンを検索させるための調整は、一貫性制御機能の (2) において、$R(y)$ を R^{k-1} と一貫している古いバージョンを検索させるのではなく、y_j と一貫しない R^{k-1} 中のバージョンを検索した操作を再実行させることである。 □

例 8 例 5の (b) に対する最新バージョンを検査するために調整した結果の実行は、図 7となる。 □

T_4	値	T_3	値
$R_4(b)$	b_2		
		$W_3(b)$	b_3
		$W_3(c)$	c_3
$R_4(c)$	c_3		
$R_4(b)$	b_3		

(b)

図 7 最新バージョンを検索させる調整

最新のバージョンの検索のための調整は、一部の検索操作を再実行させるのみで、処理単位全体を後退復帰させる必要はない。さらに、検索操作の再実行は、他の処理単位に後退復帰の連鎖といった影響を及ぼすこともない。

4.2 検索操作の実行順序

本節では、検索操作の実行順序と検索結果の一貫性及び必要な調整の関係について考察する。

例 9 例 5の (a) に対して、T_4 の検索操作の実行順序が逆となる T_4' による実行が図 8となる。検索結果の一貫性上の問題が $R_4(b)$ が実行される時に検出され、最新のバージョンを検索させるための調整によって、$R_4(c)$ が再実行される。 □

T_4'	値	T_3	値
		$W_3(b)$	b_3
$R_4(c)$	c_2		
$R_4(b)$	b_3		
		$W_3(c)$	c_3

(a)

図 8 検索操作の実行順序について

例 5の (a) では、検索操作の待機で解決できたのに対して、実行順序が逆となった場合は、問題の検出も遅くなり、一部の検索操作の再実行ということになる。すなわち、検索操作の実行順序は必要な調整と関係している。検索操作の実行順序がワークフローの順に沿うなら、新しいバージョンを検索できるが、それを利用した作業が完成しているとは限らないので、実行が待機になることがある。一方、ワークフローと逆順なら、一貫するバージョン集合は検索できるが、最新バージョンを検索できるという保証はない。最新のバージョンを利用したい場合には、一部の検索操作の再実行になりかねない。従って、利用者に検索操作の実行順序とそれによってどのような調整が起こりえるかを周知させる必要がある。

4.3 処理単位の意味論

ワークフローで表される作業手順を利用して、処理単位の定義を従来の構造的なものから意味論を利用したものに変更できる。それによってより早い段階でバージョン更新予定を把握でき、関係者に知らせることによって協同作業の効率化を計ることができる。

従来は、「begin transaction」コマンドで処理単位を定義し、「commit transaction」または「abort transaction」コマンドで処理単位の終了または後退復帰を宣言する。処理単位の定義機能が利用されてない場合は、1つのプログラムや1つのコマンドの実行が1つの処理単位として扱われる。

定義 8 意味論に基づく処理単位定義は、

「*begin transaction*

<始点 *1*, 終点 *1*, 操作種類 *1*>,

$\cdots$,

<始点 m, 終点 m, 操作種類 m>」

コマンドで処理単位を開始し、利用者の指定された範囲内のデータ項目に対する指定された種類以下の操作がそれに属するものである。 □

例 10 例 9の場合に、T_3 と T_4'に対する意味論を用いた定義は次のようになる。

「T_3: begin transaction <b, c, w>」

「T_4': begin transaction <b, c, r>」

従って、T_4'はその実行の開始時に、検索したいデータ項目に関して新しいバージョンが作成されている途中であることを把握できる。作成される結果を利用したければ、$R_4(c)$ の実行を最初から待機させるという対応をとれる。 □

処理単位の定義に用いられた意味論は、さらに、協同作業によって生じうる次のような干渉を検出するのにも役立つ。

(1) 変更する予定のデータ項目が重なる場合の検出。互いに同じデータ項目を変更していることを承知していれば、一方が変更を遅延させるまたは互いに参照しながら変更していくことがあるので、その種の協同作業を支援できる。

(2) $WF(D, E)$ において経路上で関連するデータ項目を変更する場合の検出。経路の先頭部のデータ項目のバージョンアップの予定を知っていれば、最新の結果を利用して経路上のデータ項目に関する作業を進めていくことがあるので、その種の協同作業を支援できる。例えば、$WF_1(D_1, E_1)$ において、仕様 a に関するバージョンアップの予定を知っていれば、ソースプログラム b に関する作業はその完成を待つことになるかもしれない。

(3) $WF(D, E)$ において経路上で関連するデータ項目を検索・変更する場合の検出。経路の先頭部のデータ項目のバージョンアップの予定を知っていれば、経路上のデータ項目に関する検索操作は最新結果を利用する場合があるので、その種の協同作業も支援できる。

以上のように、ワークフローで表される一貫性情報は、協同作業支援のためにも役立つ。

5. 従来の研究との比較

協同作業を支援するデータベース側からは、高度な処理単位モデルの提案という立場から研究がされている。

- 処理単位の意味論を利用することによって、並行実行が等価であることの定義を拡張することで、直列実行と等価となるスケジュールの範囲を拡大する方式[1]。
- 処理単位間の干渉を限定することによって、直列実行と等価となるスケジュールの範囲を拡大する方式[6]。
- 処理単位を階層的に分割し、直列可能性を満たす単位や後退復帰の単位を縮小する処理単位の階層モデルを用いる方式[8]。
- 応用分野の一貫性情報を利用することによって、独立化可能性という新たな正当な並行実行のクラスを提案する方式[12]。

本論文で提案している方式は、作業手順という一貫性情報を用いることによって、変更結果の一貫性を能動的に保証している。3章で示したように一貫性を保証するためには、いっさい直列可能性に頼っていない。これが従来のデータベース直列可能性を拡張する立場[5] からの研究とまったく異なる点である。

一方、検索結果の一貫性に対して現実的に受け入れられるコストで判定する方式を採用している。判定の結果が一貫しているということであれば、直列可能でない実行であるかどうかに関わらず実行を許可する。一貫していなければ、問題の原因を利用者に示した上で、解決方法を利用者に選択させる。独立化可能性[12]と比べると、マルチバージョンまで対応できるだけでなく、実用方法まで検討している。また、従来のマルチバージョン並行処理制御方式[3]のように、システムが強制的に古いバージョンを読ませることはしていない。逆に、利用者は必要に応じてバージョンを選択して使用できる。

さらに、作業手順という一貫性情報を利用することによって、協同作業支援機能の管理対象を、従来の並行処理制御の同じデータ項目に対する操作によ

る競合から、関連するデータへの作業間の影響まで拡張できた。かつ影響の検出は、問題が発生してからではなく、作業の予定を決めた時点から視野に入れている。これは従来のデータベースの立場からの研究に考慮されてない点である。

一方、ワークフロー管理システムは理論的な基盤に欠け、並行処理制御基準が明確に定義されておらず、並行処理制御に関する機能が弱いなどの問題点を抱えている[11]。本論文では、協同作業環境の意味論とワークフロー的な手法をデータベースの処理単位技術に導入することによって、問題の総合解決を計っている。

6. おわりに

本論文では、グラフで表された一貫性情報を、修正結果ともに記憶するマルチバージョンデータベースにおける活用について検討した。処理単位やデータベースのための一貫性管理要求を形式的に定義し、それらを処理単位を後退復帰させることなく効率良く管理できることを示した。一方、協同作業側における最新バージョンを検索したいという要求や潜在的な影響への対応などの要求も考慮に入れ、一貫性情報がそれらの要求を満たすためにも役立つことを示した。また、従来のマルチバージョン並行処理制御方式と異なって、一貫性管理要求を満たした上で利用者が必要に応じてバージョンを選択して使用できるようになった。一貫性情報は、一貫性管理と協同作業支援という相反する部分を持つ特徴を支援することができた。

参考文献

1) Agrawal, D., Abbadi, A. E., and Singh, A. K.: Consistency and Orderability: Semantics-Based Correctness Criteria for Databases, *ACM Trans. Database Syst.*, Vol.18, No. 3, pp. 460–486 (1993).
2) Barghouti, N. S. and Kaiser, G. E.: Concurrency Control in Advanced Database Applications, *ACM Comput. Surv.*, Vol. 23, No. 3, pp. 269-317 (1991).
3) Bernstein, P. A., Hadzilacos, V., and Goodman, N.: *Concurrency Control and Recovery in Database Systems*, Addison-Wesley (1987).
4) Cichocki, A., Helal, A., Rusinkiewicz, M., and Woelk, D.: *Workflow and Process Automation: Concepts and Technology*, Kluwer (1998).
5) Elmagarmid, A. K. (ed.): *Database Transaction Models for Advanced Applications*, Morgan Kaufmann, 1992.
6) Frrg, A. A. and Ozsu, M. T.: Using Semantic Knowledge of Transactions to Increase Concurrency, *ACM Trans. Database Syst.*, Vol. 14, No. 4, pp. 503–525 (1989).
7) Furukawa, T., Xu, H., and Shi, Y.: Supporting Collaborative Work by Process-based Transaction Model, Lecture Note in Coumpter Science, Springer Verlag Vol. 1552, pp. 421-433 (1999).
8) Nodine, M. H., Ramaswamy, S., and Zdonik, S. B.: A Cooperative Transaction Model for Design Databases, in 5), pp. 53–83 (1992).
9) Ramamritham, K. and Chrysanthis, P. K.: *Advances in Concurrency Control and Transaction Processing*, IEEE Computer Society Executive Briefing (1996).
10) Xu, H., Furukawa, T., and Shi, Y.: *Supporting Cooperative Work Based on the Semantics of Workflows*, Inter. Symp. on Database Applications in Non-Traditional Environments, pp. 279-282 (1999).
11) Worah, D. and Sheth, A.: *Transactions in Transactional Workflows*, in Advanced Transaction Models and Architectures (Jajodia, S. and Kerschberg, L. (eds.)), Kluwer (1997).
12) 徐海燕, 古川哲也, 史一華：一貫性情報を用いたデータベースの並行処理制御, 情報処理論文誌, Vol. 35, No. 12, pp. 2752-2761 (1994).