システムソフトウェアと
オペレーティング・システム 73－12
（1996．8．27）

トランザクション間同期機構による協調作業制御機構の実現

吉本雅彦　黒澤貴弘
深澤寿彦　井上　淳†

協調型アプリケーションの制御に適したデータ管理機構DMFと，その協調作業への適用について述べる．DMFはトランザクション的なデータ共有機構として設計されているが，データの更新通知など，協調型アプリケーションに適した拡張がなされている．本論文では，正当性基準として直列化可能性を採用することで，更新通知の形式的な意味付けを明確にする．

A Collaboration Control Facility based on Intertransaction Synchronization Mechanisms

MASAHIKO YOSHIMOTO, TAKAHIRO KUROSAWA,
TOSHIHIKO FUKASAWA and SUNAO INOUE†

We present a data management facility DMF and its application to controlling cooperative applications. Although DMF is basically designed as a transactional data manager, it also provides several extended features, such as update notification mechanisms, suitable for those applications. In this paper, we clarify the semantics of update notification formally by adopting serializability as a correctness criterion.

1. はじめに

グループウェアや分散システム一般の構築において，トランザクションは有用な機構である．トランザクションの利点はいわゆるACIDプロパティ，すなわちatomicity，consistency，isolationおよびdurabilityの4点に集約される．これらの性質は，処理内容や障害発生の予測が一般に困難であるという分散システムの特質を考慮すれば，概ね好ましいものといえよう．

反面1)など多くの文献で指摘されているように，トランザクションをそのままの形で協調型作業に適用するのは難しい．その理由は大別して，(i) atomicityとisolationの保証機構が資源を不必要に占有するため並行性が低下する，(ii) 協調型作業で重要な利用者間の情報交換がisolationプロパティと矛盾する，となる．(i)については，トランザクションの正当性基準として直列化可能性を採用する場合，多くのDBMSにおいて2フェーズロックプロトコル[2]が用いられていることによる．並行性の向上手法としては極めて多くの提案がなされており，直列化可能性に基づくもの[3]，読み出しトランザクションに関する直列化可能性の緩和[4]，不要ロックの早期解放[5]，トランザクションの分解[6]，などがある．ただしこれらによっても協調型作業に対しては機能的に不充分であり，また容易には実現できないものも多い．一方(ii)については，具体的かつ実用的と思われる手法が幾つか提案されている[7][8][9]が，概して正当性基準が曖昧であって，最終的には利用者の判断に委ねるという傾向が強い．これらは一貫性を重視したデータ管理機構というよりは，CAD，SDE，ワークフローなどヒューマンファクタの大きい協調型アプリケーションに適したフレームワークととらえるのが適切であろう．

本稿では，厳密な一貫性を保証しつつ協調作業に適した拡張機能を持つデータ管理機構と，これに基づく協調作業制御手法について述べる．基本的な考え方は，正当性基準として直列化可能性を採用し，データアイテムの読み手と書き手の共存を許可しつつ，更新通知に関わる直列化可能性を検査するというものである．

2. 並行制御のモデル化

従来より直列化可能性は，トランザクションの先行順位グラフにおける閉路の有無という観点から議論されてきた[10]．しかしこの方法では，並行制御プロトコルとの関係が不明瞭であって，更新通知機構を統合する上でも見通しが悪い．本稿ではオペレーション列の可換性に基づくモデルを導入して，並行実行可能性と

† キヤノン株式会社 情報メディア研究所
Media Technology Laboratory, Canon Inc.

$$V_i = \begin{bmatrix} 1 & \cdots & 1 & \cdots \\ \vdots & \ddots & \vdots & \\ 0 & \cdots & 0 & \cdots \\ \vdots & & \vdots & \ddots \end{bmatrix}$$

$$W_i = \begin{bmatrix} 0 & \cdots & 0 & \cdots \\ \vdots & \ddots & \vdots & \\ 1 & \cdots & 1 & \cdots \\ \vdots & & \vdots & \ddots \end{bmatrix}$$

$$P_{i,p} = \begin{bmatrix} 1 & \cdots & 0 & \cdots \\ \vdots & \ddots & \vdots & \\ 0 & \cdots & p & \cdots \\ \vdots & & \vdots & \ddots \end{bmatrix}$$

ただし行列は i 行と i 列についての表示であり，その他の部分については対角成分は1，非対角成分は0である．以下の関係は明らかであろう．

$V_iW_i = V_i^2 = V_i \quad W_iV_i = W_i^2 = W_i$

$P_{i,p}V_i = V_i \quad P_{i,p}P_{i,q} = P_{i,pq}$

また $i \neq j$ のとき，次が成り立つ☆．

$V_iV_j = V_jV_i \quad P_{i,p}P_{j,q} = P_{j,q}P_{i,p}$

$V_iP_{j,q} = P_{j,q}V_i \quad W_iP_{j,q} = P_{j,q}W_i$

コミットはオペレーション列をデータベース上の値に施す操作，アボートはオペレーション列の内容を破棄する操作であった．今の場合データアイテムは1個であるから，実質的に V_i はアボート，W_i はコミットに相当する．すなわち $S = \langle V_i, P_{i,p} \rangle_{i \in \overline{T}, p \in F}$ であって，コミット付きオペレーション列は $G_{i,p} = V_iP_{i,p}W_i$ か $H_{i,p} = P_{i,p}W_i$ の2種類に限られる．g は $(1,1)$ 成分が1で他は全て0であるような n 次正方行列であるから，結局次の結論が得られる．

$G_{i,p} \Rightarrow C(\overline{T} \setminus i) = \{V_jP_{j,q} \mid pq = qp, j \neq i\}$

$H_{i,p} \Rightarrow C(\overline{T} \setminus i) = \emptyset$

改めて $G_{i,p}$ と $H_{i,p}$ の意味を考えると，$G_{i,p}$ はデータベース上に置かれている値に基づいて実行されるトランザクションであるのに対して，$H_{i,p}$ はデータベースとは全く独立な値に基づいて実行されるトランザクションである．$H_{i,p}$ はデータアイテムの値をリセットするトランザクションとして解釈されるが，その値はトランザクションが独自に決定した値に基づくものであって，他のトランザクションからの参照は不可能である．従って p による処理内容にかかわらず，$H_{i,p}$ と並行実行可能なトランザクションの存在は許されない．互いに並行実行可能なトランザクションは $G_{i,p}$ 型であって，かつ p が他の全てのトランザクションに対して可換なものに限られる．

☆ $P_{i,p}$ と $P_{j,q}$ の可換性は，トランザクション間で独自の情報交換が行われないという前提に基づくが，この仮定は妥当であろう．

2.3 複数データアイテムの場合

複数のデータアイテムに関する系の状態 U は，各データアイテム $d \in DB$ に関する直積として表される．

$$U = \prod_{d \in DB} U_d,\ U_d = D_d^n$$

ただし D_d は，d の取りうる値全体である．コミットおよびアボートについても同様で，行列としての直和に相当するものを考えればよい．

一方 U に対するオペレーション列 S は，基本的には各データアイテムに関する基本データオペレーションの'直和'から生成されるが，一般にはこれらを'成分'とする行列として，各列が高々1個を除いて零行列 O であるようなものから生成されるものであってもよい．以下データアイテム m 個からなる U の部分系を考え，データアイテム $1 \leq d \leq m$ に対する標準的な基底をとる．(i,j) 成分としての X_j を除いて他は O であるような行列を X_je_{ij} とし，X_je_{ij} 全体を S_{ij} で表せば，$e_{ij}e_{jk} = e_{ik},\ e_{ij}e_{kl} = O\ (j \neq k)$ に注意して☆☆

$$\left(\sum_{k=1}^{m} X_ke_{i_kk}\right)\left(\sum_{l=1}^{m} Y_le_{j_ll}\right) = \sum_{l=1}^{m} X_{j_l}Y_le_{i_{j_l}l}$$

となり，S の構造は指定の (i_k) の組から決定される．ところで O でない非対角成分 S_{ij} と S_{ji} を同時に含む場合，i と j が相互依存関係を持つことになり，独立したデータアイテムとして扱うことはできない．すなわち S の元は，基底を適当に並べ替えることによって上半三角行列となるものに限られる．

S の空でない部分集合 T で，$ST \subseteq T$ かつ $TS \subseteq T$ となるものは特に重要である．以下これを I 部分集合とよぶことにする．I 部分集合の共通部分は，やはり I 部分集合である．

(1) **S が I 部分集合に分解される場合**

S が O 以外の共通元を持たない I 部分集合 S_1 と S_2 に分解されるとする．このとき $S_1S_2 \subseteq S_1 \cap S_2 = O$ となって，同じトランザクションが S_1 に属するオペレーションと S_2 に属するオペレーションを同時に用いることはできない．換言すれば，データベースが S_1 が作用する部分系と S_2 が作用する部分系に分けられているということである．初めに設定した U の部分系は，本来このように選ばれるべきものである．

(2) **I 部分集合 T が存在する場合**

T に属するオペレーションによって，部分系が段階的に切り離される．特に $T^n = 0$ であれば，高々 n 個の T のオペレーションによって，トランザクションは完全に切り離される．部分系の切り離しは，直観的にはデータアイテムに対するアクセス順序が木あるいはDAGとして規定されていると解釈される．$T^m \neq 0$ となる部分については，次の場合に帰着する．

☆☆ (i,j) 成分としての $P_{*,p}$ においては，$p : D_i \to D_j$ とする．V_i と W_i については，D_d によらずに同一視する

いう観点から議論を進めることにする．

2.1 直列化可能性

トランザクション処理システムは，一連のデータアイテムから構成されるデータベースと，データアイテムに対する処理を行う一連のトランザクションから構成される．例えばトランザクション i は，オペレーション列 $x_i = s_{i,1}\, s_{i,2} \cdots s_{i,m}$ の実行単位である．オペレーション列 x_i は，トランザクション開始時に空に初期化され，終了時にコミットまたはアボートオペレーションが付加される．オペレーション列による処理内容は，コミットオペレーションが付加されたときはデータベースに反映され，アボートオペレーションが付加されたときは破棄される．

一方各オペレーション $s_{i,j}$ は，データベースとトランザクション全体を表す「状態 u」に対する作用，すなわち系の状態全体の集合 U に関する写像 $s_{i,j} : U \to U$ とみなされる．このときデータベースオペレーション間に写像の合成として積が $(ss')(u) = s'(s(u))$ と定義され，結合法則 $(ss')s'' = s(s's'')$ をみたす．このことから，データベースシステムの提供するオペレーションによって生成されるオペレーション列全体を S とすれば，S は空オペレーション列を単位元とする積に関して閉じた集合である．

直列化可能性のもとで，異なるトランザクションのオペレーション列 x，y を並行に実行することができるための条件は，$\overline{x}\,\overline{y}\,g = \overline{y}\,\overline{x}\,g$ で与えられる．ただし $\overline{x}$ は x にコミットオペレーションを付加したオペレーション列，g は系の状態 u からデータベースの状態への射影である．実際 $\overline{x}\,\overline{y}\,g \neq \overline{y}\,\overline{x}\,g$ であれば，x と y の直列化は $\overline{x}\,\overline{y}$ か $\overline{y}\,\overline{x}$ のいずれかでなければならない．仮に $\overline{x}\,\overline{y}$ であるとすれば，y は x の処理結果に依存することになるが，トランザクションはアボートされうるから，最終的に x が終了するまで y は x の処理結果を参照することはできない．このことは x と y の並行実行が許されないことを意味する．

トランザクション全体を $\overline{T}$ とし，$T \subseteq \overline{T}$ に対して S の部分集合 $C(T)$ を以下のように定める：

$$C(T) = \{x \in S \mid \forall y \in \{x_i\}_{i \in T},\ \overline{x}\,\overline{y}\,g = \overline{y}\,\overline{x}\,g\}$$

このとき $C(T)$ の元は T の各トランザクションと並行実行可能である．従ってトランザクション i によるオペレーション列 y の実行が許されるための条件は

$$x_i y \in C(\overline{T} \setminus \{i\})$$

で与えられる．直列化可能性を検査するタイミングは，オペレーション要求時に行う pessimistic ポリシーと，コミット時に一括して行う optimistic ポリシーに大別される．これらの実装上／利用上の違いは大きいが，直接化可能性の判定結果は同じである．

2.2 単一データアイテムの場合

以上概念的なモデル化について述べたが，より詳細な直列化可能性の検討を行うため，行列を用いた表現を導入する．そのため以下ではトランザクション数を，

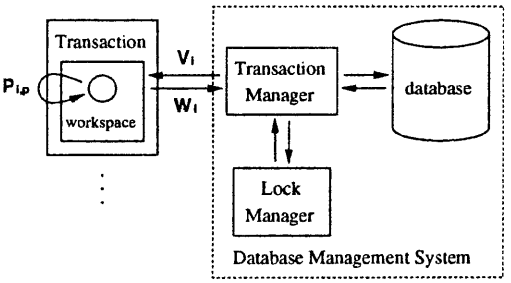

図1 基本データベースオペレーション
Fig. 1 Primitive database operations

オペレーション列が空であるものも含めて n 個に固定して，$\overline{T} = \{1, \cdots, n\}$ とみなす．これを不定個に拡張することも可能であるが，現実的なシステムのモデルとしては十分であろう．

データアイテムを1つ固定して，とりうる値全体を D，D から D への写像全体を F とする．F には写像の合成による乗法が定義されて，恒等写像がその単位元となる（以下'1'と表す）．データアイテムに関する系の状態 U を D の n 個の直積 D^n で表せば，オペレーション列は F 上 n 次正方行列に，オペレーション列の合成は行列の積にそれぞれ対応する．ただし D と F には加法が定義されていないので，一般の行列に関する意味付けは得られない．そこで行列の成分として0を許し，$f \in F$ について形式的に $f + 0 = 0 + f = f$，$f0 = 0f = 0$ と約束するとともに，D^n を行／列で表したとき，行列の各列／各行は高々1個の成分を除いて0であるものに限定して扱うものとする☆．

なお F の乗法は一般に非可換であって，D に対する F の作用は左右の区別が必要である．すなわち

右作用 : $D \times F \to D$, $(x, f) \mapsto xf = f(x)$

左作用 : $F \times D \to D$, $(f, x) \mapsto fx = f(x)$

左作用／右作用は，それぞれ列形式／行形式において用いられる．

以上に基づいて，オペレーション列全体 S を以下のように定義する．まず U のトランザクション $i \in \overline{T}$ に対する基底を

$$e_i = (\underbrace{0 \cdots 0}_{i-1\text{ 個}}\ 1\ \underbrace{0 \cdots 0}_{n-i\text{ 個}})$$

ととり，De_i にトランザクション i の作業領域を対応づける（e_1 はトランザクションマネージャとみなす）．このとき i に関する基本データベースオペレーションとして，データアイテムの参照要求 V_i，データアイテムの更新要求 W_i，i 固有の作業領域における内部処理 $P_{i,p}$ が，それぞれ以下のように定まる（図1）．

☆ 行列の作用によって D^n の成分に'0'が生じるときは，対応するトランザクションにおいて，データアイテムが未定義になったとみなされ，以後のデータベースオペレーションは全て無効化される．詳しい意味付けについては後述する．

(3) **I部分集合が存在しない場合**

部分系の切り離しは起こらないので，各データアイテムについてオペレーション列が可換でなければならない．Sの元が全て対角行列となる場合は簡明であって，各データアイテムについて競合する$H_{i,p}$型トランザクションが存在せず，かつ$G_{i,q}$型トランザクションが存在すれば$pq = qp$が成り立てばよい．一方非対角成分S_{ij}が含まれる場合，これはU_iからU_jへの写像であるが，これに対応するS_{jk}との可換性を一般的に意味付けるのは困難である．従って非対角成分については，常に非可換であると考えるのが妥当であろう．

2.4 部分的コミットとアボート

トランザクションiのコミットを$\overline{W}_i$で表す．$\overline{W}_i$は，トランザクションのステップ毎に動的に構成される．つまりトランザクションの開始時に$\overline{W}_i$を単位行列に初期化し，データアイテムjに対する処理において

$$\overline{W}_i\,(\underbrace{I_m \oplus \cdots \oplus I_m}_{j-1\text{ 個}} \oplus W_i \oplus \underbrace{I_m \oplus \cdots \oplus I_m}_{m-j\text{個}})$$

として構成すればよい．トランザクションがステップkまで進んだとき，$\overline{W}_i$の変化は次のようになる．

$$\overline{W}_{i,0} = I_{mn} \to \overline{W}_{i,1} \to \cdots \to \overline{W}_{i,k}$$

ここで$\overline{W}_{i,k}$の系列を，トランザクションの内部状態として保持される実体と考えると，$l < k$に対応する$\overline{W}_{i,l}$はステップlまでの部分的コミットと見なされ，トランザクションを未コミット部分に再構成することが可能になる．これは従来のsplitトランザクション[6]の一形態にほかならない．

部分的コミットに対応して，部分的アボート$\overline{V}_{i,l}$も全く同様に構成される．意味的にはlステップ以降の取り消しであって，従来のチェックポインティングとロールバックに相当する．

なお部分的コミット／アボートにおいては，$\overline{V}_{i,k}$と$\overline{W}_{i,k}$の系列が再構成されなければならない．アボートの場合は簡単であって，単にl以降の$\overline{V}_{i,l'}$と$\overline{W}_{i,l'}$を廃棄すればよい．一方コミットの場合は，l以前の$\overline{V}_{i,l'}$と$\overline{W}_{i,l'}$を廃棄するとともに，$l' > l$について

$$\overline{\overline{V}}_{i,l'} = I_{mn} - \overline{V}_{i,l} + \overline{V}_{i,l'}$$

として，$\overline{\overline{V}}_{i,l'}$を$\overline{V}_{i,l'-l}$に置き換えることになる．行列の和は一般には定義されないが，実質的には'0'に関して仮定された加法が用いられるだけであり，$\overline{\overline{V}}_{i,l'}$はwell-definedになる．$\overline{W}_{i,l'}$についても同様である．

2.5 Fの構造について

単純なデータアイテムの読み書きについて述べる．まずデータアイテムを仮想的に二値とみなして，読み出し処理rと$f \in F$による更新処理p_fを

$$r = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix},\ p_f = \begin{bmatrix} f & 0 \\ 0 & 1 \end{bmatrix}$$

とする．$r^2 = r$，$p_f p_g = p_{fg}$であって，$rp_f \neq p_f r$が成り立つ．特に$\forall f, g \in F$に対して$fg \neq gf$とすれば，通常の読み書きの競合と等価である．

より複雑なデータアイテムについても，多値としてある程度は取り扱うことができるが，一般には写像の意味付けを明確にした上での議論が必要である．

なお読み出しのみのトランザクションについては，rの代りに恒等写像を用いてもよい．これを無制限に許すのがcursor stability[4]であり，直列化可能性の点ではfive color protocol[3]が考えられている．

2.6 並行制御プロトコル

並行制御プロトコルは，正当性基準としての直列化可能性を保証するものである．以上の議論によれば，直列化可能性はデータベースとFの構造によって決定される．例えばフラットなデータベースにおける単純な読み書きに対しては，従来的な2フェーズロックが適合する．これはほぼ明らかであろう．

さらに部分的なコミット／アボートによってトランザクションを再構成すれば，直列化に関して無関係なロックを解放してもよい．2フェーズロックの問題点として，一旦確保したロックが不要になったとしても，他に必要なロックを保持している限り解放することができないということがある．これに対してトランザクションの再構成によるロックの解放は，必要な直列化可能性を維持しつつトランザクションの並行性を向上させる上で有用である．

逆にトランザクションは直列化の最小単位であるが，連続するトランザクションにまたがってロックを保持し続けたとしても，直列化可能性を損なうことはない．通常のトランザクションにおいては無意味であるが，読み手と書き手が共存できる場合は，必要な更新権を保持しつづけることができることになり，協調作業の制御において極めて有用である．

3. 更新通知機構の統合

単純なデータアイテムの読み書きに関して，データ更新通知機構を導入する．まずトランザクションjに対する2つのメタオペレーションN_j^+とN_j^-を設け，参照するデータアイテムに対して更新通知の受け入れ期間をN_j^+からN_j^-に設定する☆．

更新通知機構のもとでは，W_iは更新通知オペレーションNとの積に変更される．ここでNは初期値を単位行列とするn次正方行列であって，N_j^+とN_j^-によって

$$N := N_j^+(N) = N + N_j$$
$$N := N_j^-(N) = N - N_j$$

として随時変更されるものとする．ただしN_jは次のようなn次正方行列である．

☆ N_j^+とN_j^-はデータアイテムへの作用を持たない冪等オペレーションであって，直列化可能性とは無関係である．なおN_j^+とN_j^-はトランザクションjに対応するものとしたが，実際にはトランザクションとは独立であって，トランザクション間にまたがっていてもよい．

$$N_j = \begin{bmatrix} & & 1 & & \\ & & \vdots & & \\ 0 & \cdots & -1 & \cdots & 0 \\ & & \vdots & & \\ & & 0 & & \end{bmatrix}$$

これによって W_iN は，トランザクション j に対する更新通知を含む，次のようなオペレーションとなる．

$$W_iN = \begin{bmatrix} 0 & \cdots & 0 & \cdots & 0 & \cdots \\ \vdots & \ddots & \vdots & & \vdots & \\ 1 & \cdots & 1 & \cdots & 1 & \cdots \\ \vdots & & \vdots & \ddots & \vdots & \\ 0 & \cdots & 0 & \cdots & 0 & \cdots \\ \vdots & & \vdots & & \vdots & \ddots \end{bmatrix}$$

もちろんこのままでは，更新トランザクション間に変更通知のサイクルが生じる可能性があり，一般には直列化可能にならない．これに対する現実的な解決策としては，以下のものが考えられる．

(1) **pessimistic ポリシーによる外部更新の排除**
更新通知機構のもとでは，$P_{i,r}$ はトランザクションによる外部更新を排除するためのオペレーションとみなされる．従って，少なくとも最初の $P_{i,p\neq r}$ 以前に全てのデータアイテムに対して $P_{i,r}$ の直列化を確立するとともに，トランザクション中の N_i^+ を $P_{i,r}N_i^+$ に置き換えればよい☆．

(2) **optimistic ポリシーによる局所的な衝突解消**
pessimistic ポリシーによる方法では，並行性の低下とデッドロックの多発という運用上深刻な問題が生じる．しかし外部更新との間の衝突は，通常の optimistic な並行制御による衝突とは異なり，局所的に解決できる可能性が高い．さらにトランザクションマネージャによって最終的にアボートされたとしても，その原因は明白であって，トランザクションのやり直しは比較的容易である．

(3) **トランザクションの細分化**
直列化の対象はトランザクションであって，アプリケーション全体ではない．トランザクション間の情報交換が厳しく制限されていても，アプリケーションの間で必要な情報を交換できさえすれば，実用上問題はない．基本的なことであるが，直列化の単位としてのトランザクションは，最小限の大きさにすべきである．特に変更通知機構のもとでは，トランザクション終了時に更新権を放棄する必要はなく，後のトランザクションに備えて必要な権限を確保しておくことが可能であり，トランザクションの細分化によって不測の待ち状態に陥ることは避けられる．

☆ 厳密には，読み出しトランザクションについても同様である．

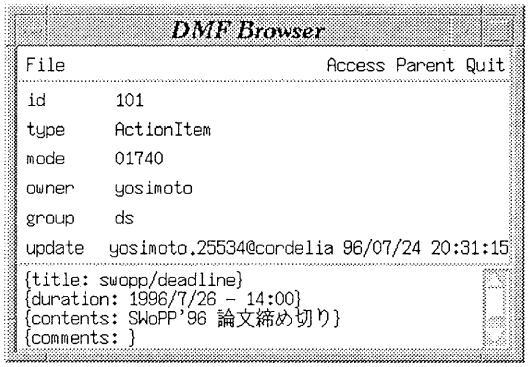

図2 DMF アプリケーションの例
Fig. 2 An example of DMF application

4. 協調作業制御への応用

以上の概念に基づくデータ管理機構 DMF，および幾つかのアプリケーションを試作した．以下 DMF の概要と応用例について述べる．より高度なグループウェア制御については，文献 11) を参照されたい．

4.1 DMF の概要

DMF は一個の集中型データ管理サーバ **D**，ホスト毎に稼働するキャッシュマネージャ **C**，および代表的アプリケーションである **P** などで構成されている．

D ：並行制御とアクセス制御，およびデータの更新に伴うイベントの生成を行う．

C ：**D**～**P** 間の通信管理，キャッシュ管理，およびクライアントの認証を行う．

P ：Tcl[12] の DMF 拡張版である．コマンドとして pavar（データの作成と参照），transaction（トランザクション制御）などが追加されている．

P では共有データが Tcl の配列変数に束縛される．以下これを共有変数とよぶ．共有変数の属性を**表1**に示す．また**図2**は，属性の一部をシンボリックに表示したものである☆☆．

表1 共有変数の属性
Table 1 Attributes of a shared variable

@, @@	データアイテムの値（@@は代入用）
@id, @dir	データアイテムとそのディレクトリの識別子
@type	データアイテムの'型'の識別子
@mode	永続性とパーミッション
@owner	所有者識別子
@group	グループの識別子
@update	最終更新クライアントと更新時刻
@writer	排他ロックを保持するクライアント
@readers	共有ロックを保持するクライアントのリスト
@awaiters	ロック待ち状態のクライアントのリスト
@watchers	参照を行っているクライアントのリスト

☆☆ 「クライアント」は，**D** におけるユーザ，ホスト，ホスト上の固有の識別子の組で表される．

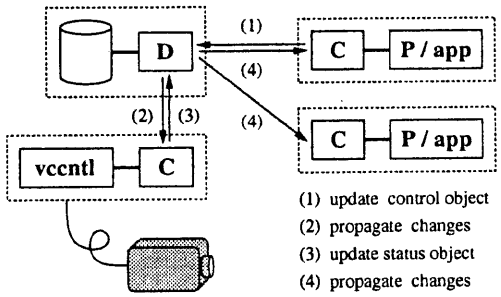

図3　共有カメラ制御系のシステム構成
Fig. 3　System structure of shared camera control system

DMF では値に関する更新通知に加えて，アクセス状況の通知も行う．これが表1の`@writer`, `@readers`, `@awaiters`, `@watchers` であって，協調作業の制御において有用である．ただし厳密にいえば，トランザクションにとってアクセス状況通知は外乱であるから，積極的な理由がない限り参照すべきでない．

並行制御ポリシーは，$P_{i,p}$については pessimistic である．N_j^+については pessimistic が標準であるが，optimistic を用いることもできる．ロックの確保は，専用のロック確保コマンドによるか，さもなければ P の trace callback においてオンデマンドで行われる．各種イベントの配送は D が定める全順序に基づいて行われ，アプリケーションの trace callback において処理される☆

4.2　カメラ制御への応用

図3にカメラ制御系のシステム構成を示す．カメラドライバ vccntl は，C のクライアントとして P/app などのアプリケーションにサービスを提供する．制御要求は 'control object の更新' に対応し，カメラの操作権は排他的である．また 'status object' は vccntl が更新権を占有する共有データであって，ロック待ちによる封鎖は起こらない☆☆．

vccntl は optimistic ポリシーに基づいて動作し，制御要求との衝突によって頻繁にアボートされるが，結果的に問題は発生しない．vccntl は新たな要求を受けてカメラ制御を行い，'status' としては結果的にアボートされなかったデータだけが残ればよいからである．

5. おわりに

トランザクション的な並行制御を基本とし，データ更新通知機構を統合することにより，協調作業の制御に適したデータ管理機構のモデル化と実装を試みた．現在まで，主として Tcl を用いた試作と評価を行ってきたが，基本的なメカニズムについては概ね実用化の目処が得られている．

今後の課題としては，プログラマインタフェースの見直し，本格的なグループウェアフレームワークへの応用，などがある．

☆ P の callback はアプリケーションの callback の後で呼ばれるため，ロック待ちの間見かけ上値が更新されたかのような状況が生じる．これを防ぐために代入用の属性@@を設けて，ロックが実際に確保された時点で，改めて@に設定するようにしている．

☆☆ さらにクライアントは，'status object' に対する@writer の存在から，サーバが動作状態にあることを知る．

参考文献

1) N. S. Barghouti, G. E. Kaiser, "Concurrency Control in Advanced Database Applications", ACM Computing Surveys, 23 (3), pp.269-317, 1991.
2) K. Eswaran, J. Gray, R. Lorie, I. Traiger, "The notions of consistency and predicate locks in a database system", CACM, 19, pp.624-633, 1976.
3) P. Dasgupta, Z. M. Kedem, "The Five Color Concurrency Control Protocol: Non-Two-Phase Locking in General Databases", ACM TODS, 15 (2), pp.281-307, 1990.
4) H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, P. O'Neil, "A Critique of ANSI SQL Isolation Levels", Proc. of ACM SIGMOD '95, pp.1-10, 1995.
5) K. Salem, H. Garcia-Molina, J. Shands, "Altruistic Locking", ACM TODS, 19 (1), pp.117-165, 1994.
6) C. Pu, G. E. Kaiser, N. Hutchinson, "Split Transactions for Open-Ended Activities", Proc. of 14th Intl. Conf. on VLDB, pp.26-37, 1988.
7) M. H. Nodine, S. B. Zdonik, "Cooperative Transaction Hierachies: A Transaction Model to Support Design Applications", Proc. of 16th Intl. Conf. on VLDB, pp.83-94, 1990.
8) G. E. Kaiser, "A Flexible Transaction Model for Software Engineering", Proc. of 6th Intl. Conf. on Data Eng., pp.560-567, 1990.
9) M. Rusinkiewicz, W. Klas, T. Tesch, J. Waesch, P. Muth, "Towards a Cooperative Transaction Model — The Cooperative Activity Model —", Proc. of 21th Intl. Conf. on VLDB, pp.194-205, 1995.
10) J. D. Ullman, "Principles of Database Systems", Computer Science Press, 1980.
11) 深澤 他, "グループウェア・アプリケーションフレームワーク CCF (Collaboration Control Facility) の構想", 情処第52回全国大会予稿集, 6-271, 1995.
12) J. K. Ousterhout, "Tcl and the Tk Toolkit", Addison-Wesley, 1994.