並行実行を考慮したワークフローの記述法

Specifications of Workflows for Concurrent Executions

徐 海燕☆ 古川 哲也☆☆
Haiyan XU Tetsuya FURUKAWA

同じまたは異なるワークフローの複数個のインスタンスが並行に実行されるときに生じるワークフローの定義方法に関わる問題について検討する．ワークフローの実行の各時点において，祖先タスクの出力データが満たす条件によって禁止される他のインスタンスの実行というインスタンス間の実行順序には閉路が生じうることを示す．その中には，直列実行しか許されない場合や並行に実行可能な場合が存在する．本論文では，ワークフローの定義による回避方法を与える．

This paper discusses the problems about specifications of workflows which arise when instances of the same or different workflows are executed concurrently. To ensure the isolation property of each workflow instance, it is necessary to guarantee the conditions of input data which are the results of the execution of its ancestor tasks. However, such kinds of control introduce deadlocks among workflow instances. We give the design principles to solve those problems.

1. はじめに

ビジネスの自動化を図るために，様々な分野においてワークフローが導入され，それに関する研究も活発に行われてきている[2),6)]．しかし，ワークフローの役割を発揮するためには，ビジネスプロセスが正しく定義されることが重要である[1),4),7)]．それには，まず個別に実行される個々のワークフローインスタンスが正しく終了できなければならない[8)]．さらに，同じまたは異なるワークフローに属する複数個のインスタンスが並行実行されるときに，互いに相手のいままでの実行によって保証される条件を満たさなくすることのないようにする必要もある．本論文では，後者の問題をワークフローの定義の立場から捉え，場合別に対策を提案し，それらを通して並行実行を考慮したワークフローを定義するための指針を与える．

ワークフローの定義には，タスクと呼ばれる各々のビジネスプロセスの定義，タスク間の制御フローやデータフローが含まれる．例えば，図 1 に示しているクレジットカードの申請を処理するワークフローでは，受付の後，カードを所有していない申請者に対しては，審査，発行の順で処理する．しかし，同一顧客が同時に 2 枚のクレジットカードを申請すると，ワークフローシステムで並行実行に対する制御が行われていない場合，2 枚とも許可され，利用上限が倍になってしまう．

並行に行われるワークフローに従う実行が互いに影響せず，最終の実行結果にも影響しないためには，隔離性を満たす必要がある[3),5),9)]．論文 5) では，ワークフロー管理システムにおける隔離性の実現方法について検討している．そこでは，データベース管理システムに基づいて実現されているワークフローシステムに対して，隔離性を SQL データベースのアサーション機能によって実現する方法を提案している．しかし，与えられたワークフローに対して，具体的にどのようなアサーションをどの時点で設定・解除すればよいのかまでについては触れていない．

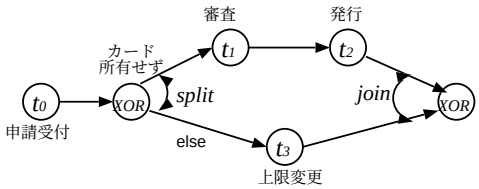

図 1 制御フロー CF_1
Fig. 1 Control flow CF_1

一方，論文 3) は，ワークフローを形式的に定義し，データベースの一貫性の必要十分条件が一貫性制約として利用できる場合の正当性基準を (1) 各節点の入力データは入力条件を満たす，(2) 外部データは一貫性制約を満たす，(3) スケジュール終了時でのデータベースは一貫している，としている．しかし，データベースの一貫性の必要十分条件を定義できるアプリケーションは限られているので，タスクの入出力条件のみに基づくワークフロートランザクションの隔離性基準が重要である．そのため，論文 9) では，ワークフローに従う実行は単独実行時に，各タスクの実行が開始される時点では祖先タスクの出力データの満たす条件が満たされることを指摘し，並行実行時にも各時点で同じような条件を満たせば隔離性を満たすことを示している．

したがって，並行実行において，各時点で祖先タスクの出力データの満たす条件をアサーションとして設定していれば，ワークフロートランザクションの隔離性は保証されるのではないかと考えられる．しかし，実際の場合を考察してみると，問題はそのように単純ではない．例えば，CF_1 に対して，同じ顧客が同時に 2 枚のクレジットカードを申請すると，発行の作業が互いに相手の「カードを所有せず」という条件のアサーションによって禁止されることになり，いわゆる，すくみに陥ってしまう．本論文では，このようなすくみを分析し，それには直列実行しか許されないものと並行実行を許すものが存在することを指摘するとともに，ワークフローの設計による対策を論じる．

本論文は，次のように構成される．2 節でワークフローモデルを定義し，3 節ではアサーションによるワークフロートランザクションの隔離性の実現方法を提案する．各種のすくみについての分析と対策の検討は 4 節で行い，5 節は全体のまとめである．

2. ワークフローモデル

ワークフローは，タスク（アクティビティ）とタスク間の実行順序を与える制御フローによって記述される．タスク t はその特徴を表す 5 つのパラメータ，入力/出力データ項目の集合 I/O，入力/出力データに対する条件 IC/OC，出力データの計算関数 F からなる組 $t(I, O, IC, OC, F)$ によって記述される．入力条件 IC/出力条件 OC は，入力データ/出力データに対する条件の集合である．

例 1 図 1 の制御フロー CF_1 には，t_0, t_1, t_2, t_3 という 4 つのタスクがある．その中の一部のタスクの記述を次に示す．

- 申請受付 t_0:
  I_0 ={ 顧客 ID:x_1, 申請額:x_2},
  O_0 ={ 申請番号:x_3},
  $IC_0 = \phi$,
  OC_0 ={ 顧客 x_1 には申請番号 x_3 がある },
  F_0 ={ 顧客 x_1 に申請番号 x_3 を与える }.
- 発行 t_2:
  I_2 ={ 顧客 ID:x_1, 許可額:x_8},
  O_2={ カード ID:x_4},
  $IC_2 = \{x_8 > 0\}$,
  OC_2 ={ 顧客 x_1 は ID が x_4 であるカードを所有 },
  F_2 ={ 顧客 x_1 に ID が x_4 であるカードを配布 }.

すなわち，t_0 は顧客の申請を受付け，申請番号を配布する．申請者がクレジットカードを所有していなければ，t_1 は申請者に対して許可額を審議する．t_2 は許可額を上限とするカードを発

☆ 正会員 福岡工業大学情報工学部情報工学科
xu@cs.fit.ac.jp
☆☆ 正会員 九州大学大学院経済学研究院
furukawa@en.kyushu-u.ac.jp

行する．申請者がクレジットカードをすでに所有していれば，t_3 は上限変更について審議と処理を行う．□

並行実行において，タスク $t(I,O,IC,OC,F)$ は原子性の単位である．タスク t は入力データ I が入力条件 IC を満たすかどうかを検査する．満たしていれば，生成された出力データ O は，出力条件 OC を満たす．本論文では，入出力データ項目はすべてデータベースに記憶され，データベースを系由してアクセスされるとする．同一項目に対しては，必要に応じて肩字で入力データと出力データを区別する．

タスクの実行順序を，タスクを節点とする有向巡回グラフである制御フローによって記述する．順序には，順次，反復，条件分岐（XOR 分岐），並列分岐（AND 分岐）がある．XOR 分岐にはいくつかの枝分かれがあり，それらの枝にはいずれか 1 つが真となる経路条件が記述される．並列分岐における異なる経路上の作業間の実行順序については制限がなく，同時に行うこともできる．図 1 に示しているのはクレジットカード申請における制御フローである．

例 2　図 2 は，単一商品を売る販売店における販売処理の制御フローを示している．注文受付の後，在庫数が注文数量を上回るなら，支払や配達の作業に移るが，そうでなければ，その注文を受け付けない後処理を行う．

- 支払 t_5:
  I_5={ 顧客 ID:y_1, 注文数量:y_2},
  O_5={ 支払額:y_5},
  $IC_5 = \{y_2 > 0\}$,
  $OC_5 = \{y_5 > 0\}$,
  $F_5 = \{y_5 = y_2 * tanka\}$.
- 配達 t_6:
  I_6={ 注文数量:y_2, 在庫数:y_4},
  O_6={ 在庫数:y_4},
  $IC_6 = \{y_2 > 0,\ y_4 \geq y_2\}$,
  $OC_6 = \{y_4^O \geq 0\}$,
  $F_6 = \{y_4^O = y_4^I - y_2\}$. □

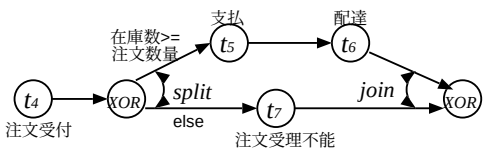

図 2　制御フロー CF_2
Fig. 2　Control Flow CF_2

トランザクションは，ワークフローに従うタスクの実行列である．形式的に記述するために，データ項目 x およびデータ項目集合 X のインスタンスを $\mathcal{I}_x$ および $\mathcal{I}_X$ とし，$\mathcal{T}(\mathcal{I}_I,\mathcal{I}_O,IC,OC,F)$ でタスク $t(I,O,IC,OC,F)$ の実行を記述する．トランザクションは，タスクの実行を表す節点と XOR 分岐節点からなる節点集合とそれらの間の実行順序を表す枝集合からなる有向グラフ $WT(TN,TE)$ である．XOR 分岐節点に対してはそれを始点とする枝は唯一に存在し，真となる経路条件がその節点の出力条件となる．

例 3　制御フロー CF_1 に対して，顧客がクレジットカードを所有してないとき，審査，発行の順で実行される（図 3 (a)）．同じく CF_2 に対して，在庫数が注文数量を上回っているときは，支払，配達の順序で処理される（図 3 (b)）．□

申請受付 $\mathcal{T}_0^1 \rightarrow XOR^1 \rightarrow$ 審査 $\mathcal{T}_1^1 \rightarrow$ 発行 $\mathcal{T}_2^1$
(a)
注文受付 $\mathcal{T}_4^3 \rightarrow XOR^3 \rightarrow$ 支払 $\mathcal{T}_5^3 \rightarrow$ 配達 $\mathcal{T}_6^3$
(b)

図 3　トランザクション WT_1 と WT_3
Fig. 3　Transaction WT_1 and WT_3

トランザクション $WT(TN,TE)$ は，非巡回グラフである．これは，制御フローが巡回の場合は複数回実行されるタスクは異なる節点に展開され，条件分岐は特定の経路のタスクに従って実行されるためである．タスクの実行である節点は原子性の単位であるが，XOR 分岐節点は $WT(TN,TE)$ におけるそれを始点とする経路上で最初に出会うタスクの実行である節点の原子性に含める．

最後にトランザクションの並行実行を表すスケジュールを定義する．制御フロー CF_j $(j = 1,2,\cdots)$ 上のトランザクション $WT_i(TN_i,TE_i)$ $(i = 1,2,\cdots)$ からなるスケジュールは，次のような有向非巡回グラフ $WH(HN,HE)$ である．

- $HN = \bigcup_i TN_i$
- $HE \supseteq \bigcup_i TE_i$
- 異なる処理単位の節点 $\mathcal{T}_s$ と $\mathcal{T}_t$ が競合する $(O_s \cap (I_t \cup O_t) \neq \phi) \vee (O_t \cap (I_s \cup O_s) \neq \phi))$ なら，それらの間には実行順序を示す枝か推移的な枝が HE に含まれる．

例 4　同一顧客が同時に 2 枚のクレジットカードを申請するスケジュール WH_1 は，図 4 で示されるようになる．□

申請受付 $\mathcal{T}_0^1 \rightarrow XOR^1 \rightarrow$ 審査 $\mathcal{T}_1^1 \rightarrow$ 発行 $\mathcal{T}_2^1$
　　↓　　↓　　↓
申請受付 $\mathcal{T}_0^2 \rightarrow XOR^2 \rightarrow$ 審査 $\mathcal{T}_1^2 \rightarrow$ 発行 $\mathcal{T}_2^2$

図 4　スケジュール WH_1
Fig. 4　Schedule WH_1

3. 隔離性を実現するための機能

トランザクションの並行実行では，各トランザクションは単独実行の時と同様に正当な実行でなければならない．論文 5) では，トランザクションの隔離性を満たすために，従来の施錠，解錠操作のように制約を設定，解除する方法を提案している．制約にはデータ項目に対するものとタスクの状態に対するものとがあり，SQL データベース言語のアサーション機能によって実現できる．アサーションは，次の SQL 文によって設定される．

CreateAssertion < *name* > **check** < *condition* >

設定されるアサーションの条件を偽にする操作は，データベースによって禁止されることになる．必要でなくなったアサーションは，次のように解除すればよい．

DropAssertion < *name* >

アサーションによって設定された制約は，一種のセマンティックロックに相当する．専有施錠のように他のトランザクションの検索，変更操作をすべて禁止するのではなく，制約を満足できなくするようなタスクの実行のみを禁止している．しかし，論文 5) は，与えられたワークフローに対して，具体的にいつどのように制約を設定・解除すれば隔離性が実現できるかについては触れていない．

一方，論文 9) は，単独実行時に各トランザクションが満たさなければならない性質を一貫性として形式的に定義し，並行実行時に同じ性質を満たすようにするための隔離性について検討している．各節点の入力データはトランザクション内の他の節点の出力データであることもあり，そうでない場合もある．前者を内部データ，後者を外部データと呼ぶ．並行実行における隔離性は，トランザクションの入力一貫性と実行結果のデータベースの一貫性を保証するためのものである．データが内部入力データ，外部入力データ，出力データに分かれているので，隔離性も同じくその 3 つに対するものに分けられることになる．

トランザクションの実行において，節点間にはデータの流れを示すデータフローが存在する．各トランザクションが単独で実行されるときには，データフローにおける親節点の実行終了時に満たしている各データ項目に対する出力条件や経路条件は，子節点が実行されるときまで保持される．しかし，並行実行時には WH_1 で示したように，そうとは限らない．論文 9) は，各トランザクションが隔離性を満たすために並行実行時において各時点で満たすべき条件について，次のように提案している．

定義 1　9) トランザクション $WT(TN,TE)$ の実行中のプロ

セス $\mathcal{T}_i \in TN$ に対し，それまでの各プロセスの入力データ項目と出力データ項目の和集合を $\mathcal{T}_i$ の使用データ項目 D_i といい，それらの最新値を $\mathcal{I}_{D_i}$ で表す．$\mathcal{T}_i$ までの入出力条件で最新値 $\mathcal{I}_{D_i}$ のみに関わるものからなる集合を，$\mathcal{T}_i$ の結果条件 TC_i という．また，トランザクション WT の最終プロセスの使用データ項目，最新値，結果条件を，トランザクション WT の使用データ項目，最新値，結果条件という． □

定理 1 [9] トランザクション $WT_i(TN_i,TE_i)$ $(i = 1, 2, \cdots)$ からなるスケジュール $WH(HN, HE)$ において，WT_i の各実行時点で使用データ項目のデータベースにおける値が直前に終了したプロセスの結果条件を満たせば，WT_i は内部入力データの隔離性と出力データの隔離性を満たす． □

例えば，WH_1 において WT_1/WT_2 中の審査 $\mathcal{T}_1^1/\mathcal{T}_1^2$ と発行 $\mathcal{T}_2^1/\mathcal{T}_2^2$ が実行されるとき，結果条件には申請者 x_1 がカードを所有していないという条件が含まれている．それぞれの結果条件は相手のトランザクションの発行 $\mathcal{T}_2^2$ と $\mathcal{T}_2^1$ の実行によって満たされなくなる．

外部入力データの隔離性は内部入力データの隔離性および出力データの隔離性とは異なる性質を持ち，データに最終変更結果かどうかを示す状態を利用することで実現することも考えられる．このため，本論文は内部入力データの隔離性と出力データの隔離性をトランザクションの隔離性といい，それに絞って検討する．

アサーション機能は SQL 92 規格に定義されているが，ORACLE や PostgreSQL などのデータベースシステムでは実装されていない．異なるシステムの実装状況とは独立して隔離性の実現を考察するために，本節ではアサーション機能を次のように定義する．

定義 2 ワークフロー管理システムに用いられるアサーションは，次の性質を満たす．

- アサーションが設定されるときには，与えられる条件は満たされていなければならない．
- タスクの実行がすでに設定されているアサーションによって与えられた条件を満たさなくするのならば，そのタスクは実行できず，待機する．
- 必要でなくなったアサーションは，設定したトランザクションにより解除される． □

アサーションが設定されていれば，同じトランザクションでもそれを偽にする操作ができないので，必要に応じてアサーションの解除と設定を行う必要がある．

定義 3 各タスク t が開始されるとき次のように入力/出力データ I/O に関わるアサーションの設定・解除を行うことにより隔離性を満足させる方法を，アサーションによる隔離性の実現法という．ここで，begin transaction と end transaction は既存のデータベースのトランザクション機能を利用している．

begin transaction
出力データ O に関わるアサーションを解除；
タスク実行；
原子性の単位内の偽になっていない経路条件と出力条件を用いてアサーションを設定；
end transaction □

各時点で満たすべき結果条件がアサーションによって保証されているので，定理 1 により次の結果が成り立つ．

定理 2 定義 3 に従って実行されるスケジュールにおいては，各トランザクションは隔離性を満たす． □

4. すくみの対策について

定義 3 の実現法を用いると，例えば，WH_1 においては，WT_1 の審査 $\mathcal{T}_1^1$ と WT_2 の審査 $\mathcal{T}_1^2$ が実行されるときに「カードを所有せず」という条件で設定されたアサーションによって，それぞれ WT_2 の発行 $\mathcal{T}_2^2$ と WT_1 の発行 $\mathcal{T}_2^1$ の実行が禁止されることになる．これにより 2 枚のクレジットカードが同時に発行されることは禁止される．しかし，互いのトランザクションの実行が相手のアサーションの解除を待つというすくみの問題が新たに生じている．

すくみの対策を検討するためには，まずすくみを引き起こす原因について分析しなければならない．そのため，すくみ検出グラフ (wait-for-graph) を定義する．

定義 4 トランザクション $WT_i(TN_i,TE_i)$ からなるスケジュール $WH(HN,HE)$ のすくみ検出グラフ $WG(WN, WE)$ は，次のように定義される．

- 節点集合 WN：トランザクション $WT_i(i = 1,2,\ldots)$ を節点とする．
- 枝集合 WE：トランザクション WT_j 内の節点の実行が WT_i によって設定されたアサーションによって禁止されていれば，WT_i から WT_j へ枝を持つ $((WT_i,WT_j) \in WE)$． □

例 5 図 5 は，WH_1 のすくみ検出グラフ WG_1 である． □

$$WT_1 \rightleftharpoons WT_2$$

図 5 すくみ検出グラフ WG_1
Fig. 5 Wait-for-graph WG_1

すなわち，トランザクション間の結果条件とそれによって禁止されるタスクの実行を含むトランザクション間に枝がある．そのようなグラフに閉路が生じれば，すくみになる．以下では，設計によるすくみを防ぐ方法について検討する．

4.1 設計規準 1

WH_1 で示しているように発行 $\mathcal{T}_2^1$ と $\mathcal{T}_2^2$ は，原子性の単位を超える経路条件「カードを所有せず」を偽にしている．このような節点が複数のインスタンスに属しているので，すくみを引き起こしている．

設計規準 1：原子性の単位を超える経路条件を偽にするタスクの実行を含むインスタンスを複数個生成させないか，または原子性の単位のタスクで経路条件によって確認された資源を確保する．

まず，原子性の単位を超える経路条件を偽にするタスクの実行を含むインスタンスが直列実行しかできないなら，複数個生成させないように設計する．例えば，クレジットカード申請の制御フロー CF_1 を図 6 の CF_1' とすればよい．それによって，1 枚目が申請中であれば 2 枚目の申請は受付されない．

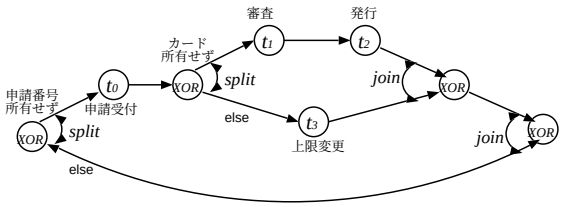

図 6 制御フロー CF_1 の改良版 CF_1'
Fig. 6 CF_1': a refine version of control flow CF_1

一方，原子性の単位を超える結果条件を偽にするタスクの実行を含むインスタンスが並行に実行可能なら，対策が異なる．

例 6 図 2 で示している CF_2 において，在庫数が 5 の時に，注文数量が 3 と 4 の 2 つの CF_2 に従うトランザクション WT_3 と WT_6 が図 7 のように実行されるとする．どちらのトランザクションが実行される時も，経路条件「在庫数 $y_4 \geq$ 注文数量 y_2」を満たす．従って，XOR^3 や XOR^6 の実行時に，条件が「$y_2 \geq 3$」と「$y_2 \geq 4$」である 2 つのアサーションが設定される．しかし，配達 $\mathcal{T}_6^3$ や $\mathcal{T}_6^6$ の実行は相手のトランザクションによって設定されるアサーションを偽にするので，共に実行が禁止させることになり，すくみが生じる． □

$$\begin{array}{ccccccc} \text{注文受付}\ \mathcal{T}_4^3 & \rightarrow & XOR^3 & \rightarrow & \text{支払}\ \mathcal{T}_5^3 & \rightarrow & \text{配達}\ \mathcal{T}_6^3 \\ \downarrow & & & & \downarrow & & \downarrow \\ \text{注文受付}\ \mathcal{T}_4^6 & \rightarrow & XOR^6 & \rightarrow & \text{支払}\ \mathcal{T}_5^6 & \rightarrow & \text{配達}\ \mathcal{T}_6^6 \end{array}$$

図 7 スケジュール WH_2
Fig. 7 Schedule WH_2

設計規準 1 に従うと，CF_2 において経路条件によって確認さ

れる在庫数はそれと同じ原子性の単位である支払 t_5 で確保しなければならない．すなわち，支払 t_5 は次のように改める必要がある．

支払 t'_5:
I'_5={ 顧客 ID:y_1, 注文数量:y_2，在庫数:y_4，受付済総数:z},
O'_5={ 支払額:y_5，在庫数:y_4, 受付済総数:z},
$IC'_5 = \{y_2 > 0, z \geq 0\}$,
$OC'_5 = \{y_5 > 0, z > 0, y_4 \geq z\}$,
$F'_5 = \{y_5 = y_2 * tanka, z^o = z^I + y_2\}$.

経路条件も「在庫数 − 受付済総数 ≥ 注文数量」のように改めなければならない．このように，実行された時に真であった経路条件によって確認された在庫数が，出力条件「在庫数 $y_4 \geq$ 受付済総数 z」を通して確保され，配達といった経路上の後続タスクの入力条件を保証する．

4.2 設計規準 2

次に，同一資源に対する異なるワークフローにおける経路条件の統一的な設定方法について検討する．

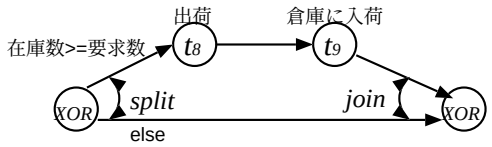

図 8 制御フロー CF_3
Fig. 8 Control Flow CF_3

例 7 図 8 は例 2 の販売店の在庫数が要求数以上なら，その分の商品を出荷し，倉庫に入荷する制御フローを示している．例 3 の WT_3 と CF_3 に従う実行 WT_4 からなるスケジュール WH_3 を図 9 に示している．WT_3 の顧客が注文し，在庫数から受付済総数を引いた数が注文数量を上回っていることが確認され，支払作業が始まった．その時，WT_4 が実行され，在庫数が要求数以上なのでその分を出荷し，倉庫への入荷作業が行われる．その後，顧客への配達 $\mathcal{T}_6^3$ が行われようとするが，WT_4 の並行実行によって在庫不足という問題が生じうる． □

$$\text{注文受付 } \mathcal{T}_4^3 \rightarrow XOR^3 \rightarrow \text{支払 } \mathcal{T}_5^3 \rightarrow \text{配達 } \mathcal{T}_6^3$$
$$\downarrow \nearrow$$
$$XOR^4 \longrightarrow \text{出荷 } \mathcal{T}_8^4 \rightarrow \text{倉庫に入荷 } \mathcal{T}_9^4$$

図 9 スケジュール WH_3
Fig. 9 Schedule WH_3

アサーションによる隔離性の実現方法を用いると，出荷 $\mathcal{T}_8^4$ の実行が WT_3 のアサーションによって禁止されるが，WT_3 が終了してアサーションが解除されても，経路条件「在庫数 ≥ 要求数」が満たされず出荷 $\mathcal{T}_8^4$ はいつまでも待つことになる．最初から経路条件が満たされなければ，別の経路を辿ることになるので，これも一種のすくみと見なせる．

設計基準 2 並行実行においても経路条件を満たせば原子性の単位であるタスクの実行を保証できるように経路条件を設ける．

例えば，CF_3 における経路条件は,「在庫数 − 受付済総数 ≥ 要求数」に改めなければならない．そうすれば，WH_3 が実行された場合と同じデータベースにおいて，WT_3 が支払 $\mathcal{T}_5^3$ まで実行された時点で CF_3 に従うトランザクションが始まっても，経路条件「在庫数 − 受付済総数 ≥ 要求数」を満たさないので，else の経路に従って実行され，WH_3 で示した問題が生じなくなる．

定理 3 設計基準 1 と 2 を満たしたワークフローにおいて，アサーションによる隔離性の実現方法ではすくみが生じない． □

証明： 各節点 $\mathcal{T}$ に対して，実行できること，すなわち入力条件を満たすことは，経路条件とデータフローの親節点の出力条件によって保証されている．経路条件については，設計基準 1 によって原子性の単位であるタスクにおいて利用され，設計基準 2 によって並行実行においても原子性の単位であるタスクの実行が保証されている．出力条件は自身の属するトランザクションが設定されているアサーションによって保証される．このため，定理が成り立つ． □

5. おわりに

本論文では，並行実行の視点からワークフローの設計について検討した．各ワークフローに従う実行が個別に行われるとき，祖先の節点によって確保された資源を子孫の節点で利用できるが，並行実行時にはそうとは限らない．仮にワークフロー管理システムが，祖先の節点によって確保された資源を他の処理単位によって利用させないように制御しても，互いに相手に確保された資源の解放を待ってどのトランザクションも進まなくなるすくみが生じうる．本論文では，すくみを防止するための設計基準を与えた．資源の利用状況が動的に変化する場合の設計と制御による対策が今後の課題である．また，提案した方法に対する定性的・定量的な分析も行っていく予定である．

[**謝辞**] 本研究の一部は，文部省科学研究費補助金基盤研究 (C)15500079 の援助を受けている．

[文献]

1) Attie, P. C., Singh, M. P., Sheth, A. and Rusinkiewicz, M.: Specifying and Enforcing Intertask Dependencies, *Proc. of the 19th VLDB Conference* (1993).
2) Alonso, G., Agrawal, D., Abbadi, A. E. and Mohan, C.: Functionality and Limitations of Current Workflow Management Systems, *IEEE Expert: Special Issue on Cooperative Information Systems* (1997).
3) Arpinar, L. B., Halici, J., Arpinar, S. and Dogac, A.: Formalization of Workflows and Correctness Issues in the Presence of Concurrency, *Distributed and Parallel Databases*, Vol. 7, No. 2, pp. 199-248 (1999).
4) Davulcu, H., Kifer, M., Ramakrishnan, C. R. and Ramakrishnan, I. V.: Logic Based Modeling and Analysis of Workflows, *Proc. ofACM Symp. on Principles of Databas Systems*, pp. 25-33 (1998).
5) Puustjarvi, J.: Workflow Concurrency Control, *Computer Journal*, Issue 1, pp. 42-53 (2001).
6) Senkul, P., Kifer, M. and Toroslu, I. H.: A Logical Framework for Scheduling Workflows Under Resource Allocation Constraints, *Proc. of the 28th VLDB Conference* (2002).
7) Trajcevski, G., Baral, C. and Lobo, J.: Formalizing the Specifications of Workflows, *Proc. of the Inter. Conf. on Cooperative Information Systems* (1996).
8) Van der Aalst, W. M. P., Hirnschall, A. and Verbeek, H. M. W.: An Alternative Way to Analyze Workflow Graphs, *Lecture Notes in Computer Science*, pp. 535-552. Springer-Verlag, Berlin (2002).
9) 徐海燕，古川哲也：ワークフロートランザクションの隔離性，情報処理学会論文誌:データベース Vol. 44, No. SIG 8 (TOD 18), pp. 55-64 (2003).

徐 海燕　　Haiyan XU
福岡工業大学情報工学部情報工学科教授．平成 2 年九州大学大学院博士後期課程修了．工学博士．並行処理制御，WEB 型教育支援システムなどの研究に従事．ACM，IEEE，情報処理学会，電子情報通信学会，日本データベース学会等会員．
古川 哲也　　Tetsuya FURUKAWA
九州大学大学院経済学研究院助教授．昭和 63 年九州大学大学院博士後期課程修了．工学博士．データベースの設計論・質問処理論，情報システムの研究に従事．情報処理学会，電子情報通信学会，ACM，IEEE，日本 OR 学会等会員．