データベースシステム 119−43
（1999. 7. 22）

協調作業の一貫性管理のためのメッセージ構造の動的再構成

井上 創造　　岩井原瑞穂
九州大学大学院システム情報科学研究科
〒816-8580 福岡県春日市春日公園 6-1
E-mail: {sozo,iwaihara}@c.csce.kyushu-u.ac.jp

アブストラクト
電子メールに代表される非同期型のメッセージは，発言者と受信者が非同期に通信するコミュニケーションの手段として幅広く用いられている．特に，コミュニケーションの内容を蓄積できる，分散システムとのインタフェースに適する点が特徴であるが，メッセージは参加者が自由に発言する性質のものであるため，現実の状況との不整合が生じる恐れがある．本稿では，コミュニケーションを現実の作業に対し一貫性の取れたものにするために，メッセージ中の作業の情報や，複数のメッセージで形成される構造を，利用者の要求に応じて動的に再構成することにより，意思決定の過程や作業プロセスを一貫性を保証しながら記録することを検討する．特に，再構成の影響が進行中のコミュニケーションに与える影響を特定する手法を検討する．

Dynamic Reconfiguration of Message Structures for Integrity Maintenance in Collaborative Work

Sozo INOUE　　Mizuho IWAIHARA
Graduate School of Information Science and Electrical Engineering, Kyushu University,
6–1 Kasuga-Koen, Kasuga-Shi, Fukuoka 816-8580, JAPAN
E-mail: {sozo,iwaihara}@c.csce.kyushu-u.ac.jp

abstract
Asynchronous messages, such as E-mails, are widely used in asynchronous human communication. Asynchronous messages have advantages that the context of communication are recorded, and that they are suitable for an interface between users and distributed systems. However, since messages are created freely by users, they may lack for integrity maintenance between the real situation and the messages. In this paper, we propose a method for recording communication processes keeping integrity by dynamically reconfiguring message structures, and for specifying the influence of reconfiguration on the current communication.

1　はじめに

電子メールや電子掲示板に代表される非同期型コミュニケーションは，計算機支援による協調作業におけるコミュニケーションの手段として，幅広く用いられている．非同期型コミュニケーションは，次のように利用できることが特徴である．

- コミュニケーションの内容を容易に蓄積できるため，後で議論の過程をたどるというように再利用できる．
- 利用者と応用システムの，また異種分散システム間のインタフェースとして用いられる．

しかし，通常のコミュニケーション支援システムは，利用者に対し自由に発言を許し，一旦発言したメッセージに変更を加えることはできないため，次のような問題がおこる．

- 現実の作業の状況とコミュニケーションの内容に不整合が生じる．
- データベース管理システムやワークフロー管理システムといった他システムのデータとの間に一貫性が保証されない．

これらの問題に対し，コミュニケーションの履歴を現実の作業に対し一貫性の取れたものにするためには，一旦生成されたメッセージに対し，作業の実行中にメッセージに表されている作業の情報や，複数のメッセージで形成される構造を，利用者の要求に応じて動的に再構成することが必要である．

ただし，不用意にメッセージの構造を変更していくと，かえって現実の状況と整合性がとれなくなることになりかねない．そこで，次のアプローチが重要であると考える．

1. コミュニケーションの中に現れる作業の情報やルールをコミュニケーションのモデルに導入する．
2. 再構成が進行中のコミュニケーションの状況に与える影響を特定し，利用者に明示する．利用者にはコミュニケーションモデルの一貫性が保証される範囲での再構成を許す．

我々はこれまでに，コミュニケーションのやり取りをメッセージトランザクションとしてモデル化し，その一貫性として，やり取りの順序や，やり取りの進行が他のやり取りに与える影響をトランザクション従属性として記述する手法を提案した [3]．本稿では，現実の作業で扱われる情報をメッセージ中で記述するリソースを導入し，メッセージ構造を再構成する際にトランザクション従属性やリソースの制約を保つ手法を検討する．

以下では，2節で動的な再構成が必要になるようなコミュニケーションの例をあげる．また，3節で協調作業の情報を持つ

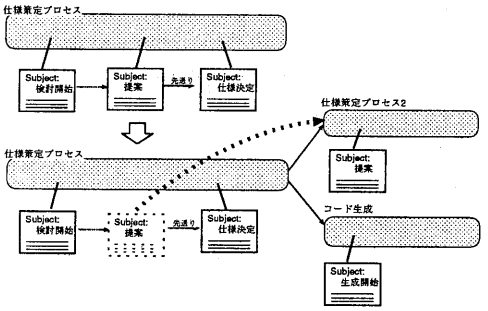

図 1: メッセージ構造の変更が必要となる例

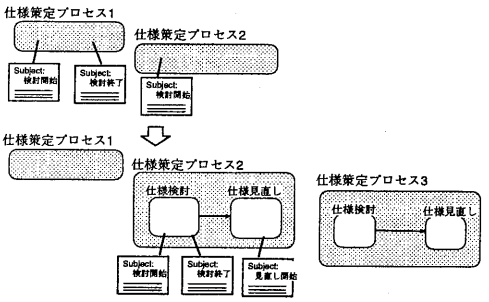

図 2: メッセージ構造の変更が必要となる例

コミュニケーションをモデル化し，4節でメッセージ構造を動的に再構成していく手法を検討する．

2 メッセージ構造の変更が必要となる例

この節では，メッセージ構造の変更が必要となる例を示す．

図1は，ソフトウェア開発プロセスの例を示している．この例で，コード生成プロセスは，仕様策定プロセスで仕様が決定した後に開始されるものとする．時間的に余裕がない場合は，急を要しない検討事項は先送りされる場合がある．この場合，検討されていない提案を次の仕様の議論で取り上げて検討されるため，利用者にとっては提案のメッセージを次の仕様策定プロセスですぐに再利用できる形で蓄積することが要求される．

図2は，仕様策定プロセスを，仕様の検討と見直しの２つの部分プロセスに分け，以降の仕様策定はこの部分プロセスが適用される例である．このようにプロセスが変更されることによって，今後生成されるメッセージにも，検討終了，見直し開始といった新しいメッセージが追加される．このようなことはコミュニケーション支援システムがプロセスの情報を得ていればある程度予測できる．また，利用者にとっては，現在実行されているプロセスのメッセージを，構造が変化したプロセスと整合性がとれた状態で蓄積することが要求される．

図1の例は，(a) 現実の世界におけるプロセスのルールが一時的に変更され，コミュニケーションの蓄積にも変更を加える必要がある例である．一方，図2の例は，(b) 現実の世界におけるプロセスのルールが永久的に変更され，コミュニケーションの蓄積が整合性を保つために必要となる変更である．このような２つの場合について，本稿では以下の方針で対応する．コミュニケーションに現れる現実の作業の状況や，現実の作業におけるプロセスなどの種々の制約のモデルとして，m-group や mg テンプレート [3] を用い，それらに利用者の要求に応じた変更を許すよう拡張する．m-group は，メッセージ間の関連，およびメッセージと現実の世界のオブジェクトの関連を表現するものである．また，mg テンプレートは，発言の順序や，作業に必要な情報といった，現実の作業で定められる制約を記述するものである．

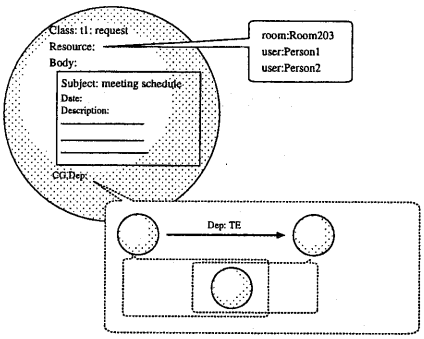

図 3: m-group

- (a) の場合は，m-group が変更されると考え，その変更が他の m-group へ及ぼす影響を特定する．
- (b) の場合は，mg テンプレートが変更されると考え，mg テンプレートの変更による m-group への影響を特定する．

3 コミュニケーションのモデル化

メッセージ構造のモデルとして，意味的なメッセージのまとまりを概念化した m-group[3] と，メッセージの生成の順序や，コミュニケーションの進行が他のメッセージに与える影響に関する従属性を表すトランザクション従属性 [3] を用いる．本稿ではさらに，メッセージと現実の世界のオブジェクトを関連づけるリソースを導入する．

3.1 m-group

m-group S は，タプル

$$[Sid(S), Class(S), Resource(S), Body(S), CG(S), Dep(S)]$$

で記述される（図 3）．$Class(S)$ は，$n_i{:}C_i$ の形式の項の集合である．n_i を t 識別子と呼び，C_i を **m-group クラス**と呼ぶ．m-group クラスは，発言者の意図を提示するために用いられる．m-group クラスの例として，「提案」「意見」「賛成」「結論」といった構造化議論モデル（IBIS）[2] で用いられているものや，ソフトウェアプロセス [6] における「バグ報告」「改

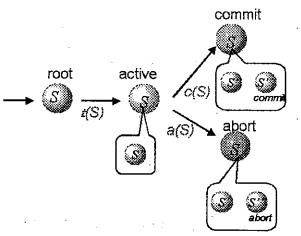

図 4: トランザクションイベント

善要求」「設計変更」「評価結果」といったものがあげられる．$Resource(S)$ は，メッセージで参照される作業中の情報リソース（3.2節）Rへの参照の集合である．リソースは，m-groupの内容に関連する協調作業の情報を記号で表したものである．メッセージ本体 $Body(S)$ は，電子メールなどの通常のメッセージである．**子 m-group 集合** $CG(S)$ は，m-group 識別子 S_i の集合 $\{S_0, S_1, \ldots, S_k\}$ である．それぞれの S_i は S の**子グループ**と呼ばれ，S は S_i の**親グループ**と呼ばれる．また $Dep(S)$ は，S のトランザクション従属性の集合（3.4節参照）であり，子グループ間の従属性を記述する．$Body(S)$, $CG(S)$, $Dep(S)$ のいずれも空値を許す．子グループによる m-group の階層構造は，複数の m-group の子グループとなるような DAG 構造を許す．

3.2 リソース

リソース Rは，タプル

$$[Rid(R), RClass(R), Object(R)]$$

で記述される．$Rid(R)$ は，リソースの識別子である．$Object(R)$ は，協調作業に現れる情報であり，リソース実体と呼ぶ．$RClass(R)$ は，$Object(R)$ の型であり，リソースクラスと呼ぶ．

リソースの例として，次のような物があげられる．

- 他の m-group：$[R_1, \texttt{m-group}, s_1]$
- 他の m-group のリソース：$[R_2, \texttt{resource}, R_1]$
- ワークフローにおけるプロセス：$[R_3, \texttt{process}, p_1]$
- 利用者やソフトウェアエージェント：$[R_4, \texttt{user}, inoue]$
- 場所，ハードウェア資源：$[R_5, \texttt{room}, room103]$
- ソフトウェアや，それに使われるデータ：

  $[R_6, \texttt{URL}, "http://www..."]$

3.3 メッセージ上のトランザクション

メッセージのやり取りの進行を，m-group のトランザクションイベント [4] として記述する．

各 m-group Sは，トランザクションの実行状態の遷移に応じて，以下のトランザクションイベントを発生させる（図 4）．

- $r(S)$：(*root*) m-group S が生成された．
- $t(S)$：(*active*) m-group Sの子グループ S'が $r(S')$ または，$t(S'), c(S')$，$a(S')$ のトランザクションイベントを発生している．
- $a(S)$：(*abort*) m-group S の子グループに m-group クラス abort の m-group が生成された．
- $c(S)$：(*commit*) m-group S の子グループに m-group クラス commit の m-group が生成された．

メッセージトランザクションは次のように変化することで実行状態を変化させると考える．(1) 利用者やシステムにより m-group S が作られる．これを実行が開始されたととらえる．(2) S で表される作業の進行にともない，利用者は新しい m-group を Sの子グループに加える．(3) 利用者が m-group クラス commit または commit を持つ m-group を生成すると，実行が終了したととらえる．

利用者に作業の目的と作業の指針を示すため，メッセージ本体を持たない，**空の m-group** を実際の作業が始まる前に生成することが可能である．実際の作業が実行されると，あらかじめ生成された m-group にメッセージが与えられる．

3.4 トランザクション従属性

トランザクション間の従属性を記述するために，文献[7], 文献 [1] で用いられている以下の記号<, →を用いる．

- $e_1 < e_2$：(先行制約) イベント e_1とイベント e_2の両方が起きるなら，e_1が先に起こる．ただし，e_1と e_2が発生するかどうか関しては制限しない．
- $e_1 \rightarrow e_2$：(発生制約) イベント e_1 が起きるなら，イベント e_2も起きる．これは e_1と e_2の起きる順序に関しては制限しない．

上記トランザクションイベントの式に対し，論理記号∧（論理積），∨（論理和），⇒（含意）を用いて m-group 間のトランザクション従属性を表す [3].

以下では，m-group 上のトランザクション従属性を定義する．トランザクション従属性とは，次にあげる記述のいずれかである．ただし $1 \leq i, j \leq k$，$i \neq j$である．

- $Atomic(S)$：Sの子グループのすべての組 (S_i, S_j) に対する制約 $a(S_i) \rightarrow a(S_j)$，$c(S_i) \rightarrow c(S_j)$ の宣言．
- $Exclusive(n_i)$：Sの子グループのすべての組 (S_i, S_j) に対する制約 $c(S_i) \rightarrow a(S_j)$ の宣言．

以下の定義では，S_iとS_jは，SまたはSの子グループであるとする．

- $AA(S_i, S_j)$: (abort dependency) $a(S_i) \rightarrow a(S_j)$ の宣言.
- $CC(S_i, S_j)$: (commit dependency) $c(S_i) < c(S_j)$ の宣言.
- $CO(S_i, S_j)$: (commit occurrence) $c(S_i) \rightarrow c(S_j)$ の宣言.
- $AC(S_i, S_j)$: $(c(S_j) \rightarrow a(S_i)) \wedge (a(S_i) < c(S_j))$ の宣言.
- $CA(S_i, S_j)$: $c(S_i) \rightarrow a(S_j)$ の宣言.
- $TE(S_i, S_j)$: (terminate) $(c(S_i) < c(S_j)) \wedge (a(S_i) < c(S_i))$ の宣言.

これらを組み合わせて用いることにより，入れ子トランザクションや代替トランザクションといった構造化されたトランザクションの記述が可能である [4].

m-groupに対しトランザクション従属性を不用意に与えていくと，トランザクション従属性が充足不能になる可能性がある．この問題については，文献 [4] で議論されている充足可能な状態を保つための十分条件を用いることができる．

4 メッセージ構造の動的再構成

この節では，メッセージ構造を動的に再構成する手法を検討する．メッセージ構造の再構成は，次のように分類できる．(1) アドホックな変更：m-groupの構造がその場限り一時的に変更される．(2)**m-groupクラスの変更**：m-groupクラスにより分類される協調作業の制約が変更され，同じm-groupクラスを持つm-groupは全て変更される．(3)**m-groupクラスの部分的な変更**：m-groupクラスにより分類される協調作業の制約が変更され，同じm-groupクラスを持つm-groupのうちいくつかのm-groupが変更される．

これらに対し，次に述べる手法で対応する．利用者がm-groupを生成する際，m-groupクラスからm-groupの構造を導出する枠組みであるmgテンプレート [3] を用いる．また，mgテンプレートによりm-groupの構造を導出する手続き **bind** を定義する．(1) に対しては，m-groupに対し直接リソースや，子グループ，トランザクション従属性を変更する．(2) に対しては，mgテンプレートを作業の実行時に変更し，再び **bind** することで対応する．(3) に対しては，mgテンプレートを2つのバージョンに分け，mgテンプレートに **bind** されたm-groupをいずれかのmgテンプレートに **bind** しなおす．そのあと一方のmgテンプレートに (2) を適用することで実現できる．

以下では，mgテンプレートを説明し，上記の手法を議論する．

4.1 t表現

m-groupの構造をmgテンプレートに記述するために，t表現を導入している [3]．t表現は次のいずれかである．ただし，Cはm-groupクラス，各e_iはt表現である．

1. 項 $n{:}C$.
2. 項 $n{:}\langle e_1, e_2, \ldots, e_k \rangle$ (**AND-group**).
3. 項 $n{:}[e_1, e_2, \ldots, e_k]^+$ (**+-group**).
4. 項 $n{:}[e_1, e_2, \ldots, e_k]^*$ (**∗-group**).

上記で，各項のnは，t識別子と呼ばれるt表現内で唯一のラベルである．

mgテンプレートでは，m-groupを予測できる程度に応じて，m-groupの構造を次のように記述する．(1) 定型業務の報告のように，m-groupの生成が予定されている場合，AND-groupのt表現で記述する．ソフトウェア開発におけるバグ発見のように，m-groupを定義できるが実際に生成されるかどうか分からない場合，∗-groupで記述する．バグに対する対策案のように，1つ以上生成されることが分かっているが，実際にいくつ生成されるか分からない場合，+-groupで記述する．予測できないため，あらかじめm-groupとして定義できない場合，m-groupはt表現には記述されず，利用者が手動でm-groupの構造を定義する．

4.2 mgテンプレート

次にmgテンプレートを定義する．m-groupクラスCのmgテンプレート $T(n_i{:}C)$ は，タプル

$$[Tid(C), Texp(C), Resource(C), Dep(C)]$$

である．$Tid(C)$ はmgテンプレートの識別子である．$Texp(C)$ は**t表現**である．$Dep(C)$ は，Cの**トランザクション従属性集合**である．$Resource(C)$ は，リソースクラスの集合である．$Dep(C)$ は，3.3，3.4節のトランザクションイベント，トランザクション従属性に対し S_i,S_jを形式的にそれぞれ n_i,n_jに置き換えて得られる，トランザクション従属性の集合である．

4.3 mgテンプレートによるm-group構造の構築

利用者が，$n{:}C$を持つm-group Sを生成し，t表現中に $n{:}C$ を持つm-group S'の子グループに加えようとする時，システムは $\mathcal{T}(n{:}C)$ から Sを構築する手続き **bind** を行う．**bind** を以下に記述する．

1. $e = Texp(C)$ に対し，その種類に応じて次の (a)–(d) のいずれかを適用する．
   (a) eが項 $n_i{:}C_i$なら，$n_i{:}C_i$を持つ Sの空のm-groupを生成し，Sの子グループに加える．
   (b) eが AND-group の項 $n{:}\langle e_1, e_2, \ldots, e_k \rangle$ なら，各 e_1 に対し，その種類に応じて (a)–(d) のいずれかを適用する．

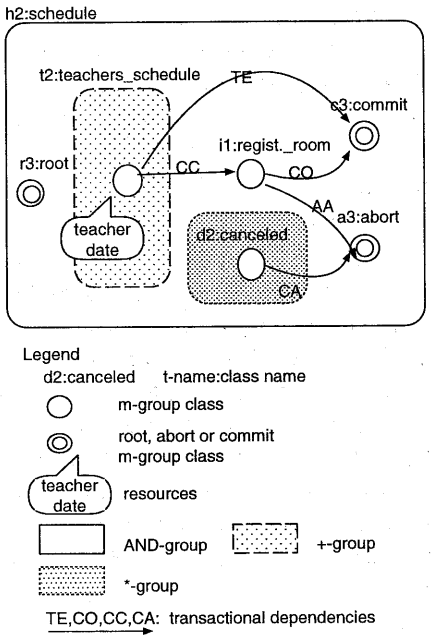

図 5: mg テンプレートの例

(c) e が+-group の項 $n{:}[e_1, e_2, \ldots, e_k]^+$なら，システムが利用者に $e_1, e_2, \ldots, e_k$を提示し，利用者にいずれか 1 つ以上を選択させる．利用者が選択した e_iに対しその種類に応じてシステムは，(a)–(d) のいずれかを適用する．

(d) e が*-group の項 $n{:}[e_1, e_2, \ldots, e_k]^*$なら，システムが利用者に $e_1, e_2, \ldots, e_k$を提示し，利用者にいずれか 0 個以上を選択させる．利用者が選択した e_iに対しその種類に応じてシステムは，(a)–(d) のいずれかを適用する．

2. (a)–(d) で t 表現中の $n_i{:}C_i$より生成された m-group S_i に対して，$Dep(C)$ の要素に現れる各 n_iを S_iに置き換え，$Dep(S)$ に加える．例えば $Dep(C)$ に従属性 $Atomic(n_i)$ が含まれるとき，$Dep(S_i)$ に対し，$Atomic(S_i)$ を加える．

3. (a)–(d) で t 表現中の $n_i{:}C_i$より生成された m-group S_iに対して，システムは $Resource(C)$ の各要素であるリソースクラス L_iを利用者に提示する．利用者は各リソースクラスに対し，L_iを持つリソースを定義し，$Resource(S)$ に加える．

上記で，**bind** 後の $\mathcal{T}(n{:}C)$ を，Sに **bind** された mg テンプレートと呼ぶ．

図6では，図5のスケジュール調整の mg テンプレートを **bind** する様子を示す．はじめに，利用者が Sを $Texp(S')$ 中に $n{:}C$

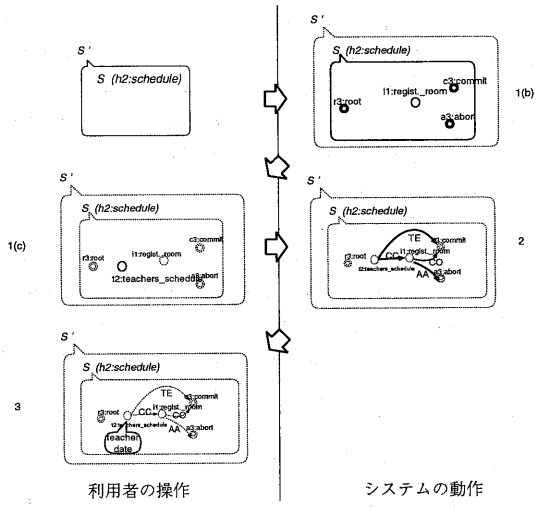

図 6: m-group の **bind**

を持つ S'の子グループに加える．1(b) システムが AND-group により定められたスケジュール調整の手順を示す空の m-group を生成する．1(c) +-group により 1 個以上の生成が許された，先生とのスケジュール調整を表す m-group が生成される．2 システムが，S の子グループ間に mg テンプレートに記述されたトランザクション従属性を割り当てる．3 利用者が，mg テンプレートに記述されたリソースクラスを持つリソースを S の子グループに割り当てる．

4.4 m-group の再構成

利用者が行う m-group Sの構造の変更には，次の種類を仮定する．

1. t 識別子または m-group クラスの変更
2. リソースの変更
3. 子グループの変更
4. トランザクション従属性の変更

これらは相互に影響を及ぼす．また，**bind** された mg テンプレートによる制約により変更が出来ない場合がある．これについて以下で議論する．ただし以下の議論では，Xから X'への変更は，X'の追加，Xの削除，Xから X'への置換のいずれかを意味する．

1 については，m-group が **bind** した mg テンプレートにより制限がある．また，新しい m-group クラスに対応する mg テンプレートに対し，再度 **bind** する必要がある．これについては，4.5の (a) で述べる．

2 について，リソースの変更には次の種類がある．(a)$Resource(S)$ の参照をリソース Rからリソース R'に変更

する．この場合，RとR'のリソースクラスは同一でないと変更は許可されない．(b) リソース実体の変更．リソース実体OをO'に変更する場合は，Oをリソースに持つ他の m-group S_iに対しても影響がある．そのため，全てのS_iから変更の許可がなければ変更できない．利用者の許可が得られなければ，RからR'への変更は許可されない．

3について，子グループS_iを子グループS_i'に変更しようとする場合，S_iに対しS_iの変更部分に応じて1,2,3,4のうちいずれか1つ以上を適用する．その結果，適用した部分の変更が許可されない場合，S_iからS_i'への変更は許可されない．

4について，トランザクション従属性の集合D_1をD_1'に変更しようとする場合，変更後の$TDeps(S)$に対して充足不能になるなら，この変更は許可されない．

4.5 mg テンプレートの再構成

m-group に **bind** された mg テンプレートを変更する方法には，次の2通りが考えられる．

(a) m-group に対し m-group クラスを変更する．m-group クラスが変更されると，別の mg テンプレートが **bind** される．

(b) m-group に **bind** されている mg テンプレートを修正する．修正された mg テンプレートを **bind** する m-group 全てに修正の影響が及ぶ．

(a) において，m-group クラスの変更は，mg テンプレートの変更につながる．この場合，m-group の既存の子グループにどのように mg テンプレートを **bind** するかが問題となる．そこで，mg テンプレート $T(n_i{:}C) = T$，$T(n_i{:}C') = T'$に対し，m-group の m-group クラスCをC'に変更する場合，次のような手順を定める．

1. Tから **bind** されたリソースとトランザクション従属性を，m-group から取り除く．
2. m-group を T'に **bind** する．
3. 2の結果充足不能になる場合は，T'に変更できない．この場合，上記4.4節のいずれかの方法で m-group を変更するか，T'を修正し2を適用する．

(b) について，mg テンプレート $T(n_i{:}C) = T$を $T(n_i{:}C) = T'$に変更すると，$Class(S) = n_i{:}C$であるような m-group S_i 全てに対し (a) を行うことと同じ結果になる．このような変更は，コミュニケーションがあまり進んでいない場合は有効であるが，コミュニケーションが進行するにつれて充足不能になることが多くなる．このような場合は，Tを複製して2つのバージョンT_1, T_2に分け，Tに **bind** された m-group を T_1,T_2のいずれかに **bind** しなおしたあとでT_1またはT_2を変更することで，全体に影響が波及することを防ぐ．ワークフロー分野においては，進化するプロセスの構造をスキーマのバージョンにより管理する手法が文献 [5] で提案されている．このようなプロセスの進化に対しコミュニケーションの整合性を保つことが必要である．

5 おわりに

本稿では，コミュニケーションにおいて協調作業の一貫性を保証するため，メッセージに m-group クラスやリソース，トランザクション従属性といった作業の状況を表す情報を導入し，その間の制約を満たしながらメッセージの構造を変更する手法を検討した．実際には，m-group への変更を許可できる利用者は，変更の権限を持つ者に限られると考えられる．特に，4.4節の2のように，複数の m-group に変更を加える場合には，複数の利用者が協調して許可する必要があり，変更要求や，変更承認のための m-group クラスを定義することも考えられる．今後は m-group を変更する権限についても検討していく．また，現実のコミュニケーションに適応し，その効果を評価する予定である．

参考文献

[1] Attie, P., Singh, M. P., Sheth, A. and Rusinkiewicz, M.: Specifying and Enforcing Intertask Dependencies, *Proc. 19th Int. Conf. Very Large Databases*, pp. 134–145 (1993).

[2] Conklin, J. and Begeman, M. L.: gIBIS: A Hypertext Tool for Exploratory Policy Discussion, *ACM Trans. Office Information Systems*, Vol. 6, No. 4 (1988).

[3] Inoue, S. and Iwaihara, M.: Structured Message Management for Group Interaction, *Proc. Int. Workshop. New Database Technologies for CSCW and Spatio-Temporal Data Management (NewDB'98)*, Singapore (1998).

[4] Iwaihara, M., Inoue, S. and Matsuo, H.: On Unifying Message and Transaction Management for Collaborative Design Work, *Proc. Int. Symp. Digital Media Information Base (DMIB97)*, Nara, pp. 300–304 (1997).

[5] Joeris, G. and Herzog, O.: Managing Evolving Workflow Specifications, *Proc. of 3rd IFCIS Int'l Conference on Cooperative Information Systems(CoopIS '98)*, New York (1998).

[6] Kellner, M. I. et al.: Software Process Modeling Example Problem, *Proc. 6th Int. Software Process Workshop*, pp. 19–29 (1990).

[7] Klein, J.: Advanced Rule Driven Transaction Management, *Proc. COMPCON* (1991).