

コミュニケーションを用いた作業プロセスの動的な獲得

井上 創造[†] 岩井原 瑞穂^{††}

本論文では、コミュニケーションプロセス、つまり、プロセスの構造があらかじめ明確には定義されないまま実行され、実行と並行して人間同士の議論によって定義が詳細化されるプロセスを管理する M-Trans システムを述べ、その方式の有効性の評価結果を示す。コミュニケーションプロセスは設計と並行して実行される可能性があり、設計と実行の間にやりとりが行なわれることも考えられる。コミュニケーションプロセスの管理は、ワークフローの例外処理や動的再構築を含めた、予測困難な状況への適応を実現するという意味で重要である。本システムはプロセスとコミュニケーションを統合的に記述するモデルに基づいており、プロセスの例外的な、あるいは未定義の動作に対してその動作を局所的に許す方法と、過去の局所的な動作の記録に基づいてプロセスの仕様を変更する方法をコミュニケーション支援に提供することによって作業プロセスを参加者が動的に獲得できる。

Dynamic Process Specification using Communication based Processes

SOZO INOUE[†] and MIZUHO IWAHARA^{††}

In this paper, we introduce the notion of communicative process, which is a process to be executed without complete specification and designed in a discussion in parallel with the process. The execution of the communicative process may be performed on parallel with the design process, and the design process and the implementation may have interactions for coordination. Managing communicative processes is important since it realizes adaptation to unexpected situations, including exception handling and dynamic re-composition of a process in WfMS. We introduce the M-Trans System, which manages communicative processes on the model that provides integrated specification of a process and communication. The system provides communication support for localizing unexpected or undefined behavior of a process, and for dynamically capturing process specifications by generalizing the record of once localized processes.

1. はじめに

現実の世界においては、ワークフロー管理システム (Workflow Management System, WfMS) において扱うことが難しい協調作業が存在する。本稿ではそのような作業を扱うモデルとしてコミュニケーションプロセスを導入する。コミュニケーションプロセスは、プロセスの構造があらかじめ明確には定義されないまま実行され、実行と並行して人間同士の議論によって定義が詳細化されるプロセスである。以下では、プロセスを定義するための議論を、設計プロセスと呼び、定義されたプロセスの仕様をプロセス仕様、プロセスの実体をプロセスインスタンスと呼ぶ。コミュニケーションプロセスは、図 1(a) のようにプロセスインスタンスと設計プロセスの間にやり取りが行われる

こともある。

コミュニケーションプロセスの管理は、ワークフローの例外処理^{2),7)} や動的再構築^{16),18)} を含めた、予測困難な状況への適応を実現するという意味で重要である。なぜなら、予測困難な状況においては、その場で適応方法が検討され実行されるため、適応方法の検討が設計プロセスであり、適応自身がプロセスインスタンスと言えるためである。現実の世界は、頻繁に変わりうる環境や予測困難な人間の行動にさらされるため、柔軟なプロセス管理が WfMS における重要な要求である。

M-Trans システムは、プロセスとその設計プロセスを統合的に扱い、参加者間のコミュニケーションの記録を元にコミュニケーションプロセスを支援するシステムである。本論文では、設計プロセスとプロセスインスタンスの間のやり取りに注目し、設計プロセスと 1 つ以上のプロセスインスタンスの間に整合性がとれなくなる場合に設計プロセスとプロセスインスタンスの双方を調整して整合性を確保する方法を提案する。

[†] 九州大学 大学院システム情報科学研究科 情報工学専攻
Graduate School of Information Science and Electrical Engineering, Kyushu University

^{††} 京都大学 大学院情報学研究所 社会情報学専攻
Graduate School of Informatics, Kyoto University

設計プロセスとプロセスインスタンスの不整合は、プロセスに例外が発生した場合や、プロセスインスタンスの実行が設計プロセスより先行する場合に起きる。提案する手法では、次の方法を用いる。

- 局所化: プロセスインスタンスが設計プロセスに反した、あるいは未定義の動作をする場合に、その動作を局所的に許す方法である。
- 一般化: 一旦局所化されたプロセスインスタンスをプロセス仕様に取り込み、他のプロセスインスタンスからも利用できるようにする方法である。

提案する手法では、設計プロセスとプロセスインスタンスが不整合の場合に、局所化を行なう方法と、プロセス仕様を変更して整合性を確保する方法を用いる。プロセス仕様を変更する方法は、過去に局所化されたプロセスインスタンスを設計プロセスの参加者に提示し選択させ、選択されたプロセスインスタンスをシステムが一般化することにより行なう。つまり、過去のプロセスインスタンスの記録を利用してプロセス仕様を詳細化、あるいは修正する方法を設計プロセスに実現するものである。また一般化をプロセス仕様の動的な獲得ととらえると、プロセスインスタンスの動作が2度以上繰り返されるまではプロセス仕様にはたよらずに局所化して実行できるので、実際に繰り返される動作のみをプロセス仕様を採用することのできる効率の良い方法である。

M-Trans システムは、プロセスと利用者間のコミュニケーションを統合的に記述するモデル^(8),9),20)に基づく。我々は文献^(8),20)で、参加者間の議論をトランザクションと考え、参加者の発言の分類に基づきトランザクションを動的に構築する手法を提案しており、文献⁽⁹⁾では、議論のトランザクションとプロセスを統合したシステムにおいて、プロセスの例外を議論のトランザクションを用いて解決する方法を示した。これらの手法における議論のトランザクションとプロセスは、それぞれ本論文における設計プロセスおよびコミュニケーションプロセスで実現できるが、本論文ではさらにコミュニケーションプロセスが設計プロセスに先行する場合に有効な方法を提案する。

本論文の構成は次のとおりである。2章ではコミュニケーションプロセスとその設計プロセスを議論し、3章で M-Trans システムのモデルとその実行について述べる。4章ではプロセスの実行記録を用いてプロセスを動的に獲得する手法を述べる。5章ではその手法を実際の電子メールの蓄積に適用した結果を述べる。6章では関連研究を述べ、7章でまとめを述べる。

2. コミュニケーションプロセス

この章では、コミュニケーションプロセスの性質を明らかにし、技術的な課題を明らかにする。

協調作業の例として、学会会議のプログラムを作成する場合を考える。プロ

求し収集する。収集のためのプロセスは、それぞれの著者に応じて異なることが考えられる。ある著者は電子メールで提出するかもしれないし、別の著者はFAXで提出するかもしれない。また著者によってはタイトルと著者情報を別々に送るかもしれないし、著者によっては誤りの修正のための例外的な処理が行なわれる可能性がある。このための議論は、著者とプログラム委員の間の提案や質問といった交渉により決定される。タイトルがそろった時点で、プログラム作成の担当者は発表をセッションに分け、セッションの座長を割り当てる。

2.1 プロセスの設計の分類

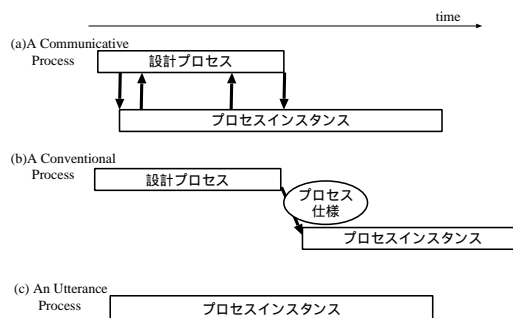


図1 プロセスの設計の分類

コミュニケーションプロセスは、プロセスの構造があらかじめ明確には定義されず、作業の参加者どうしの合意によって定義され実行されるプロセスである。以下では、他のプロセスを設計するために行なわれる意思決定のプロセスを、設計プロセスと呼ぶ。また、設計されたプロセスの定義をプロセス仕様と呼び、プロセス仕様をもとに生成されたプロセスの実体をプロセスインスタンスと呼ぶ。一つのプロセス仕様に対しプロセスインスタンスは複数個存在してよいものとする。コミュニケーションプロセスの例を図1(a)に示す。

ワークフロープロセス (Conventional Process) は、設計者によってあらかじめプロセス仕様が決定的に実行されるプロセスである。例えば、タイトルを要求する方法は、プログラムを作成する担当の人々がその手順をワークフローで記述することが可能である。通

常の WFMS は、あらかじめ設計者がプロセスを定義していることを想定している。図 1(b) はワークフロープロセスを示している。設計プロセスはプロセスインスタンスのためのプロセス仕様を出力し、プロセスインスタンスの実行は設計プロセスが終了した後に行われる。

文献^{1),5),11),12)}では、発言自体により実行されるプロセスが議論されている。上記の例で座長を打診する発言は、打診した時点で返答を期待しているため、要求に引き続く返答というプロセスを実行していると考えることができる。このように参加者の発言により生成され実行されるプロセスを、発言プロセス (Utterance Process) と呼ぶ。質問、提案といった参加者の発言の分類がシステムの支援の対象である。図 1(c) は発言プロセスを示している。発言プロセスは設計プロセスを独立して持たず、参加者の発言によりその場で生成され実行される。発言プロセスのプロセス仕様は、システムに静的に記述されている。

コミュニケーションプロセスの特徴は次のとおりである。

- プロセスは議論によって設計され、その中で採用されたプロセス仕様が実行される。上記の例では、論文情報を提出する方法はプログラム委員と各著者との間の交渉により決定される。
- プロセスインスタンスの実行が設計プロセスよりも先に行なわれる可能性がある。設計プロセスは、議論がその大半を占めるため、プロセス仕様や対象領域の知識を完全には記述する時間が無いことが考えられる。例えば、タイトルの間違いは後続の作業の遅延を防ぐために早急に修正されなければならないため、プロセス仕様が完全に定義されないまま実行される可能性がある。
- 設計プロセスは、単独で実行できずに、むしろそのプロセスインスタンスに依存する場合がある。プロセスインスタンスが設計プロセスより先行してしまう場合は、設計プロセスにおける主な議論はプロセスインスタンスの正当性や、他のプロセスインスタンスへの影響の検討になるはずである。例えば、プログラム委員が例外的な処理として、著者情報を FAX で再送してもらうように著者に要求した後は、その行為が設計プロセスで了承され、例外を通常処理に組み込むかどうか決定される必要がある。もし組み込まれるなら、FAX で再送するという処理がプロセス仕様に組み込まれる。
- コミュニケーションプロセスは発言プロセスやワー

クフロープロセスと混在する。上記の例では、プログラム委員と各著者の交渉は発言プロセスにより行なわれる。またプログラムを作成する全体のプロセスは、著者情報収集後、プログラムを編集し座長を割り当てるというように、WFMS により管理されることも考えられる。

2.2 設計プロセスと実行の一貫した管理

コミュニケーションプロセスの前節に示した特徴から、設計プロセスの結果のプロセス仕様とプロセスインスタンスの間の整合性を保ちつつ、柔軟な管理を行なうことが重要である。しかし伝統的な WFMS では、プロセスは設計プロセスが終了した後に実行されるというトップダウンの方法しか保証しない。コミュニケーションプロセスを支援するためには、プロセスインスタンスから得られる情報を活用することが重要である。

本論文では、設計プロセスの中ではプロセスインスタンスから得られる情報が次のように利用されると考える。

- 整合性の検査 (Consistency Check): まず、プロセス仕様とそのプロセスインスタンスの間で整合性を検査する手法が必要である。コミュニケーションプロセスにおいては、プロセスインスタンスが実行されるときだけではなく、設計プロセスが進行する際にも既存のプロセスインスタンスとの間にこの検査が必要である。
- 局所化 (Localization): プロセス仕様とプロセスインスタンスの不整合を防ぐための単純で、強力な方法は、プロセス仕様とプロセスインスタンスを独立させ、プロセスインスタンスをその場限りの実行と見なしてしまう方法である。例えば、論文情報を再送することは、特定の著者に限った臨機応変の対応であると見なす方法である。図 2(b) は、図 2(a) のような状態から実行された実行が、プロセス仕様に対応する部分がないような場合に、局所化により整合を取る例である。
- 一般化 (Generalization): 局所化は、しばしば危険な状況をもたらす。なぜなら、他のプロセスインスタンスで著者情報の再送がプログラム委員の承認があれば認められるというような場合に、局所化されたプロセスインスタンスが他のプロセスインスタンスの進行から孤立してしまうためである。そのため、本論文ではプロセスインスタンスの一般化を導入する。一般化は、局所化されたプロセスインスタンスをプロセス仕様に取り込み、他のプロセスインスタンスからも利用できるよう

にする方法である．一般化は、次のように利用される．すなわち、設計プロセスとプロセスインスタンスの不整合が発生した時に、過去に局所化されたプロセスインスタンスのうち不整合を解決できるものを一般化し、プロセス仕様を変更して整合性を確保する．例えば、著者情報の再送が繰り返されたときに、1 度目のプロセスインスタンスを一般化することによりプロセス仕様の変更からの孤立を避けることができる．図 2(c) は局所化されたプロセスを一般化し、他のプロセスインスタンスに適用する例である．

コミュニケーションプロセスの管理は、ワークフローの例外処理^{2),7)}や動的再構築^{16),18)}を含めた、予測困難な状況への適応を実現するという意味で重要である．現実の世界は、頻繁に変わりうる環境や予測困難な人間の行動にさらされる．柔軟なプロセス管理が WfMS において重要な要求である．予測困難な状況においては、その場で適応が検討され実行されるため、適応自身がコミュニケーションプロセスの一部と言える．

2.3 設計プロセス

図 3 は我々のモデルにおけるコミュニケーションプロセスの枠組みを示す．設計プロセスは、設計プロセス仕様に従って進行する．設計プロセス仕様は、設計プロセスの仕様を記述したものである．本システムでは、設計プロセス仕様を記述するために MG-テンプレートというモデルを用いる．設計プロセスの実行記録は、プロセス仕様として見ることが可能である．プロセス仕様も設計プロセス仕様と同様に MG-テンプレートで記述される．プロセスインスタンスは通常は、プロセス仕様と整合性をとりながら生成され実行される．

設計プロセスの実行記録からプロセス仕様に変換する方法は本論文では扱わないが、iDCSS¹⁴⁾などで用いられている意思決定支援とプロセス設計支援を組み合わせた手法を用いることができる．

また本システムでは、プロセス仕様と設計プロセス仕様、プロセスインスタンスと設計プロセス仕様それぞれ同一のデータモデルを用いるため、局所化と一般化の機構を二つのレベルに用いることが可能である．つまり、図 3 の (b) に示すように、設計プロセス—設計プロセス仕様の間、そしてプロセス仕様—プロセスインスタンスの間の両方に適用可能にする．ただし以下では簡単のため、プロセス仕様とプロセスインスタンス間の局所化と一般化に限定して議論する．

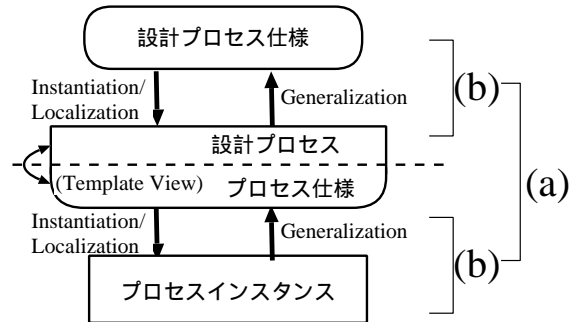


図 3 M-Trans システムにおけるプロセスの構成

3. M-Trans システム

本節では、M-Trans システムを説明する．

現在の M-Trans システムは Java 言語で実装されており、Java applet と Java servlet が協調して動作する．利用者は Web browser 上で利用可能である．図 4 は、M-Trans システムのユーザインタフェースである．システムで扱うデータは XML (eXtensible Markup Language) で記述される．XML を用いることにより、普及している XML のツールを用いてシステムのデータを流通させることが可能である．

以下では、3.1 節でプロセスインスタンスのモデルであるメッセージグループを述べる．また、3.2 節で MG-テンプレートを述べ、3.3 節でメッセージグループと MG-テンプレート間の整合性について議論する．この節では設計プロセスと他のプロセスインスタンスは区別せず、MG-テンプレートとメッセージグループの関係に注目する．つまり、図 3 の (a) の違いは区別せず、(b) の関係に着目する．

3.1 メッセージグループ

本システムでは、メッセージやプロセスはメッセージグループ (MG) として統一的に扱う．図 5(b) は MG の例である．MG は、タプル：

$[m_id, m_class, Children, Tdeps, body, Roles, Rscs]$.

である． m_id は MG の識別子である． $body$ は文章、音声、動画といった任意の媒体による参加者のメッセージである． m_class は MG-クラス と呼ばれ、発言者の意図や、プロセスの目的を表す．MG-クラスの例として「Issue」「Argument」「Position」といった構造化議論モデル (IBIS)⁴⁾ で用いられているものや、ソフトウェアプロセス¹³⁾ における「バグ発見」「改善要求」「設計変更」「評価結果」があげられる．MG は、複数の MG を $Children$ に子グループとして持つこ

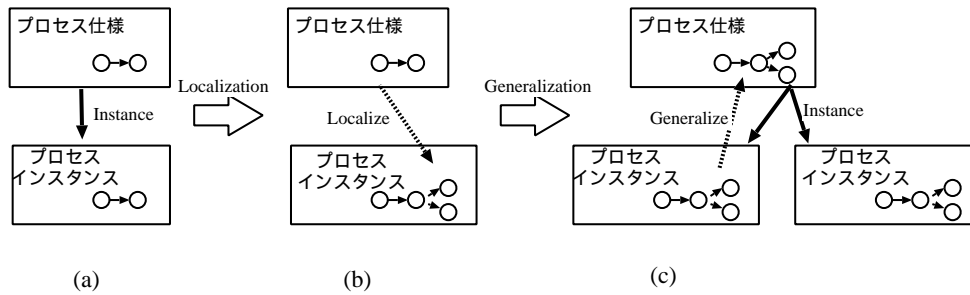


図2 コミュニケーションプロセスの局所化と一般化

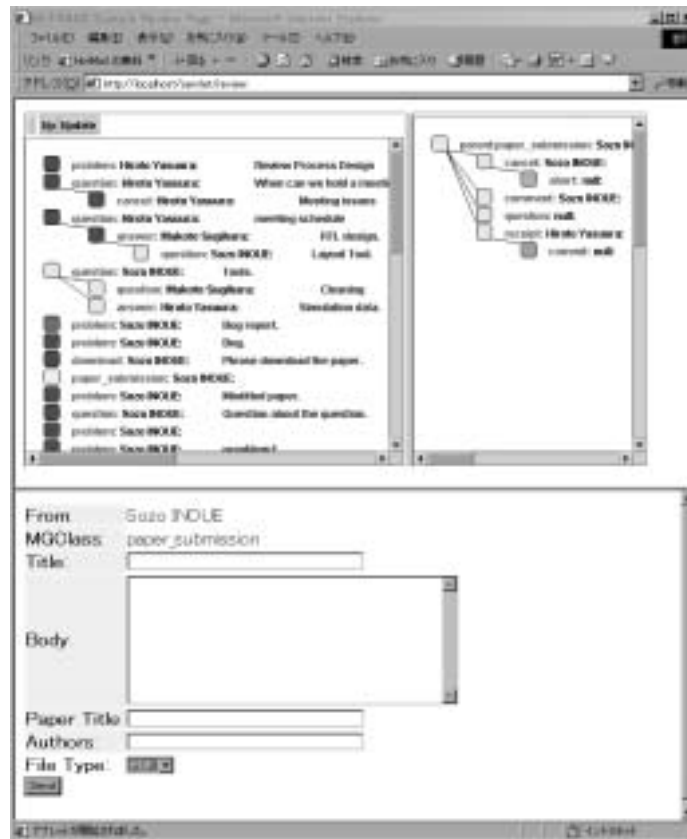


図4 M-Trans システムのユーザインタフェース

とができ、*Children* と後述する *Tdeps* により一連のメッセージのやりとりを表すことができる。*m_class* は、MG を生成した参加者が与える。場合によっては複数のメッセージが発言者の意図やプロセスの目的に対応することがあるが、その場合は複数の MG を *Children* に含む MG を生成することで表現することができる。このための MG も含め、MG の生成は MG を子グループに持つ MG の参加者に許される。

Children に加えることの出来る子グループの種類

は、3.2節で導入する MG-テンプレートで指定する。子グループの *Children* にもさらに子グループを加えることが可能なので、階層的な構造をもつ MG も記述可能である。

MG *m* が生成される事象を *root*, *m* の子グループが生成される事象を *m* の「実行」、意図を達成して終了する事象を *commit*、達成せずに終了する事象を *abort* と呼ぶ。MG は全ての子グループが「終了」、つまり *commit* または *abort* した後に *commit* すること

が許される。また MG は、MG への参加者を役割、使用するデータの情報を、リソースとして複数個持つことができる。Rscs は役割の集合、Roles リソースの集合である。

Tdeps はトランザクション 従属性 (TD) である。図5では、MG 間に与えられる有向枝で示されている。TD は、MG の実行に関する MG 間の従属性である。MG m_1 から MG m_2 への TD を $label(m_1 \rightarrow m_2)$ のように表す。TD がいずれかが終了していない MG の間に割り当てられている場合、TD は WfMS におけるアクティビティ間のコントロールフローと同じ意味を持ち、始点が commit した後でなければ終点は実行されない。一方、両端の MG が終了している場合は、TD は終点が始点の内容を参照していることを意味する。TD はあらかじめ決められた文字列の集合の要素を *label* として持つ。図中で root, commit および, abort は二重円で示されている。

3.2 MG-テンプレート

この節では、MG-テンプレートを定義する。MG-テンプレートは、MG の仕様を記述したものであり、次のタプルで表される。

$[t_id, t_class, TChild, TDTmpls, TRscs, TRoles]$.
 t_id は MG-テンプレートの識別子、 t_class は MG-クラス、 $TChild$ は MG-クラスの集合、 $TDTmpls$ は TD-テンプレートの集合、 $TRscs$ はリソースの集合、 $TRoles$ は役割の集合である。これらは本節内で説明する。MG-テンプレートは、1 つの MG-クラスに対し 1 つが対応付けられる。以下では MG-クラス c を t_class にもつ MG-テンプレートを $mgTemplate(c)$ と表す。図 5(a) は MG-クラス Decision に対する $mgTemplate(c)$ である。Decision の意味的な構造は SIBYL¹⁵⁾ から導入した。

MG-テンプレートは MG の定義と似ているが、以下の点で異なる：MG-テンプレートは子グループの代わりに MG-クラスの集合を持ち、TD の集合の代わりに TD-テンプレートの集合を持つ。

MG-テンプレート T の定義に従い MG が生成されることをインスタンス化と呼び、インスタンス化された MG m を T のインスタンスと呼ぶ。逆に T は m の $mgTemplate$ であると呼ぶ。ある MG m のインスタンス化とは、ある MG m_{parent} の m_class の値 m_class_{parent} について、 $mgTemplate(m_class_{parent})$ 中の $TChild$ に属する MG-クラスを持つ MG m を利用するあるいはシステムが生成し、 m_{parent} の子グループに加えることである。 $TChild$ に属する MG-クラスを t_class とする MG-テンプレートを用意すること

で、「MG の子グループの子グループ」のように MG の階層を定義できる。

図 5(b) の有向枝は MG-テンプレートのインスタンスがインスタンス化されたときに与えられる TD を表し、TD-テンプレートと呼ぶ。TD-テンプレートは

$$label(m_1[multi_1 \rightarrow multi_2]m_2)$$

で表される。*label* はあらかじめ決められた文字列集合の要素であり、 m_1 と m_2 は両端の MG を表す。 $multi_1$ と $multi_2$ は、TD-テンプレートの両端に与えられる、後述する多重度制約である。TD-テンプレートには、AND-split, OR-join といった分岐と結合を与えることができる。AND-split は、始点が commit した後に全ての終点がインスタンス化されることを意味し、OR-join は、始点のうち 1 つでも commit すれば終点がインスタンス化可能であることを意味する。

TD-テンプレートの両端には、TD-テンプレートの属する MG-テンプレートからインスタンス化された MG に関する多重度制約を与える。多重度制約とは、次のものである。この TD-テンプレートからインスタンス化された TD の両端の MG が commit する際に：

- 1: 反対側に接する MG1 つに対し、同じ側の MG が 1 つ存在する。
- *: 反対側に接する MG1 つに対し、同じ側の MG が 0 個以上の任意個存在する。
- +: 反対側に接する MG1 つに対し、同じ側の MG が 1 つ以上存在する。

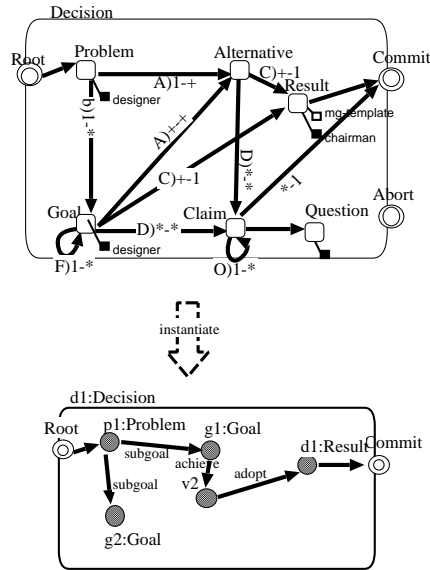
AND(OR)-split(join) は MG のインスタンス化の時に検査され、多重度制約は MG が commit する際に検査される。MG のインスタンス化の状況によっては commit 時に多重度制約を満たさないことがあるが、この場合は 4 節で述べる局所化や一般化の機構を用いて多重度制約を補償する。

図 4 では、左上の領域に MG を表示している。ここで各 MG が root であるか、commit, abort したかが四角の色で塗り分けられる。図では、Paper_submission を持つ MG、最上段の Problem, Bug_report をタイトルに持つ Problem が、それぞれ root, commit, abort の色である。また右上の領域には、利用者が注目する MG に関する MG-テンプレートが表示され、commit と abort は別の色で区別される。下の領域には、利用者が注目する MG の内容が表示される。

3.3 MG と MG-テンプレートの整合性

MG-テンプレートがインスタンス化され、MG-クラスから MG が生成されると、その MG の MG-クラスを終点を持つ TD-テンプレートのインスタンスもインスタンス化される。この際、終点のインスタンス化

(a)MG-テンプレートの例



Labels:
 a): achieve
 b): subgoal
 f): subgoal/follow/precede
 d): support/object/assign_role/assign_resource
 /AND /OR
 o): support/object
 c): adopt
 q): query

Legend:

Decision MG class name
 □ MG class

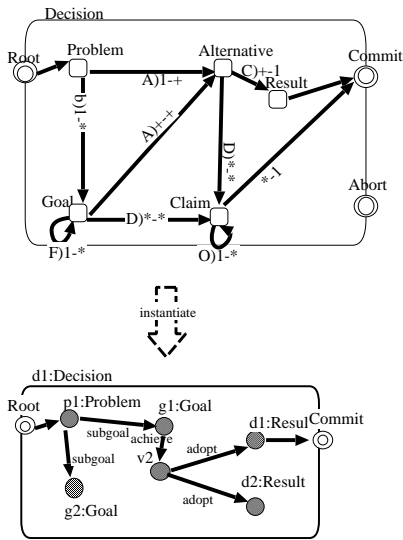
→ 1-1 dependency
 →+1 +1 dependency
 →*-* *-* dependency
 →++ ++ dependency
 →1-ALL 1-ALL dependency
 →ALL-1 ALL-1 dependency

MG id:
 MG class:
 root event:
 label:
 TD

(b) MGの例

図5 MG と MG-テンプレートの例

(a)MG-テンプレート



(b)"achieve(Goal[+ ->+]Alternative)" and "adopt(Alternatie[+>+]Result)"について不整合

図6 整合でないMGの例

は始点が commit した後に行なわれる。

しかし、コミュニケーションプロセスにおいてはそのMG-テンプレートに反してMGがインスタンス化される可能性がある。そのため、MGとMG-テンプレートの整合性を定義する。

定義1 MG m : $[m_id, m_class, Children, Tdeps, body, Rscs, Roles]$ が以下の条件を満たすとき、整合であると呼ぶ。ただし $mgTemplate(m_class)$: を $[t_id, t_class, TChild, TDTmpls, TRscs, TRoles]$ とする。

- 任意の $child \in Children$ について、 $child$ のMG-クラスが $TChild$ に存在する。
- 任意の $tdtmpl \in TDTmpls$ について、その多重度制約を満たす。
- 任意のTD-テンプレート $tlabel(t_class_1[multi_1 \rightarrow multi_2]t_class_2) \in TDTmpls$ について、 $Children$ の中でそれぞれ t_class_1, t_class_2 を持つ任意のMG m_1 と m_2 について、TD $(m_1 \rightarrow m_2)$ が $Tdeps$ に存在する。ただし、 $tlabel, multi_1, multi_2$ は任意の値とする。
- 任意の $rsc \in Rscs$ が $TRscs$ に存在する。
- 任意の $role \in Roles$ が $TRoles$ に存在する。
- 任意の $child \in Children$ が整合する。

この定義は MG の MG-クラス, 多重度制約, リソース, 役割, そして再帰的な子グループについて整合性を定義している. 図 6 では, 整合ではない MG の例を示している. 図の MG は, g_2 から Alternative に接続される TD が存在しないので, $\text{achieve}(\text{Goal}[+ \rightarrow +]\text{Alternative})$ について整合である. これは現実の場面では, 解決策が全ての部分問題に対しては検討されていないことに相当する. さらに, v_2 に接する Result が 2 つ存在するので, $\text{adopt}(\text{Alternative}[+ \rightarrow 1]\text{Result})$ についても整合ではない. これは設計プロセスにおいて 2 つの結論が決定されたことに相当する.

MG が整合であるかを検査し, 整合しない場合はその原因となる部分を得るアルゴリズム *match* を示す. *match* の中では, MG m が MG 中の TD-テンプレート t に関して多重度制約を満たすかどうかを検査するサブルーチンを $\text{checkMultiplicity}(t, m)$ と記述する. このサブルーチンの詳細は省略するが, t の種類に応じて m の構成要素の数を数え上げればよい.

入力: MG m : $[m_id, m_class, Children, Tdeps, body, Rscs, Roles]$,

MG-テンプレート: $[t_id, t_class, TChild, TDTmpls, TRscs, TRoles]$

出力: m と m の再帰的な子グループのうち整合でない部分の集合

アルゴリズム: *match*

- (1) 空集合 *ret* を用意する.
- (2) **if** ($m_class \neq t_class$) **then** m を *ret* に加え *ret* を出力し終了.
- (3) **foreach** $child \in Children$,
 - (a) **if** ($class_{child} \notin TChild$) **then** $child$ を *ret* に追加 (ただし $class_{child}$ は $child$ の MG-クラスとする.).
 - (b) **if** $\text{match}(child, \text{templ}_{child}) \neq \text{空集合}$ **then** その出力を *ret* に追加 (ただし templ_{child} を $\text{mgTemplate}(class_{child})$ とする.).
- (4) **foreach** TD ($m_1 \rightarrow m_2$) $\in Tdeps$,
 - **if** (ある $TDT \notin TDTmpls$) **then** ($m_1 \rightarrow m_2$) を *ret* に追加 (ただし TDT は, m_class_1, m_class_2 をそれぞれ m_1, m_2 の MG-クラスとし, $multi_1, multi_2$ は任意の多重度制約としたときの TD-テンプレート ($m_class_1[multi_1 \rightarrow multi_2]$

m_class_2)).

- (5) **foreach** $tdtmpl \in TDTmpls$,
 - **if** ($\text{checkMultiplicity}(tdtmpl, m) = false$) **then** m を *ret* に追加.
- (6) **foreach** each $rsc \in Rscs$,
 - **if** ($rsc \notin TRscs$) **then** rsc を *ret* に追加.
- (7) **foreach** $role \in Roles$,
 - **if** ($role \notin TRoles$) **then** $role$ を *ret* に追加.
- (8) *ret* を出力して終了.

4. MG-テンプレートの動的な獲得

この節では, 2.2 節で述べた局所化と一般化の機構を用いて MG と MG-テンプレートの間の整合性を柔軟に管理する手法を述べる. 本システムでは, 局所化と一般化の機構を, MG-テンプレートと MG の間に実現する. MG-テンプレートと MG の組み合わせは, 設計プロセス仕様と設計プロセス, プロセス仕様とプロセスインスタンスのどちらの組み合わせも実現するため, 図 3 に示した二つのレベルにおける局所化と一般化が実現できる.

利用者の MG に対する要求は, システムに MG-操作列として伝えられる. MG-操作列は, MG に対する操作の列である. MG に対する操作は, MG の内容の更新, *commit*, *abort* のいずれかである. MG の内容の更新は, MG-クラスの変更または, 子グループ, TD, リソース, 役割のいずれかの追加, 削除である.

MG-操作列に対し, システムはその時点で MG-操作列が安全, つまり MG-操作列を適用しても全ての MG が整合であるかを検査する. もし安全なら, MG-操作列は即時に適用される. そうでないなら, システムは不整合となる原因を, MG-操作列を要求した利用者に示し, 次の選択肢を与える.

- (キャンセル)MG-操作列をキャンセルし, 破棄する.
- (局所化)MG-操作列を局所化として適用し, 対象の MG のテンプレートへの影響を避ける.
- (一般化)対象の MG のテンプレートを共有するすでに局所化されたインスタンスのうち, 一般化することによって対象の MG が整合になるものを利用者に選択させ, 一般化をする.

この方法をプロセス仕様の動的な獲得ととらえると, プロセスインスタンスの動作が 2 度以上繰り返されるまではプロセス仕様によらずに局所化して実行できるので, 実際に繰り返される動作のみをプロセス仕様

に採用することのできる効率の良い方法である。

以下では、MG-操作列の安全性の検査、不整合の原因を利用者に提示する方法、MG-操作列の局所化、局所化された MG のテンプレートへの一般化を説明する。

安全性の検査

システムは、MG-操作列の各列について、順に **match** を用いて安全性の検査を行なう。MG-操作列の各列 i の操作対象を x_i とする。MG-操作列の各列は例えば「MG m を生成する」という内容であり、このときの捜査対象は m となる。 x_i の型は MG, リソース, 役割, TD のいずれかである。 i 列までを適用した時点での x_i を含む MG を m_{x_i} とする。またこのときの m_{x_i} のテンプレートを $T_{m_{x_i}}$ とする。MG-操作列の各列を適用した結果について、**match**($m_{x_i}, T_{m_{x_i}}$) を適用する。MG-操作列内の全ての列について **match** が空集合であれば、MG-操作列は安全である。そうでなければ、**match** の出力の和集合が不整合の原因である。

不整合の可視化

MG-操作列が安全なら、MG-操作列は即時に適用されるが、そうでなければ、システムはその原因を利用者に視覚的に表示する。図 7 はその原因を表示する例である。左のフレームで、MG-操作列によって変更される部分、この場合は論文を再送するという追加処理が強調された色で表示される。

この画面は m_x の参加者が閲覧することが可能である。この画面に続いて、利用者は「キャンセル」、「局所化」、「一般化」のいずれかを選択することができる。「キャンセル」を選ぶと、システムは MG-操作列を消去して MG-操作列の適用を破棄する。残りの 2 つについては次の 2 つの節でべる。

局所化

利用者が「局所化」を選ぶと、システムは MG-操作列を実行するが、操作の対象となる MG は、以降の **match** では無視される。

一般化

MG の構成要素 x を MG-テンプレート T に一般化する方法を説明する。 x を MG, リソース, 役割, TD のいずれかとする。一般化の手続きは次のとおりである。

- (1) x がリソースまたは役割なら、 x をそれぞれ T の $TRscs$, $TRoles$ に追加する。
- (2) x が MG なら、
 - (a) x に対応する新しい MG-クラス c を作り、 T の $TChild$ に追加する。

(b) C に対応する新しい MG-テンプレート T_c を作る。

(c) x の各子グループと T_c にこの手続きを再帰的に適用して一般化する。

- (3) x が TD なら、TD-テンプレート " $c_s(1 \rightarrow 1)c_d$ " を T に加える。ただし c_s , c_d はそれぞれ x の始点と終点の MG-クラスである。

一般化された後の MG-テンプレート T' について、 x および既存のインスタンスが整合なら、 T は T' と置き換えることができる。

システムは MG-テンプレートと置き換え可能な MG-テンプレートの候補を探索し、利用者にもとの MG-テンプレートからの変化を表示する。図 7 は視覚的に表示された候補の例である。図では、右の領域に一般化可能な MG-テンプレート **Abort**, **Commit**, **Send_by_FAX**, **Comment** が表示されている。

利用者が MG の候補を一つ選択すると、システムは MG を一般化し T と置き換える。ただ、この際に既存の T のインスタンスのうち不整合となる MG が出来てしまう可能性がある。この場合には、次の 3 つの対応が考えられる。

- (キャンセル) 当初の MG-操作列と一般化を破棄する。
- (局所化) 既存の T のインスタンスを局所化する。これにより既存の MG は以降 T との整合性の検査対象から外されるが、一般化の候補の探索対象にはなりうる。この方法は、次に述べるテンプレートのバージョンを分離すべきかどうかを判断できない場合に有用である。
- (バージョンの分離)：一般化された MG-テンプレートを T と置き換えず、新しいバージョン T' とする。これにより既存のインスタンスは T' の影響を受けないが、後に T' に関する一般化が発生した時に T の情報を利用できないという問題がある。

この対応の選択は、通常の MG-操作列に不整合が生じた時と同じように利用者が選択する。これらを実現するために、システムは一般化に対し、通常の操作と同様に「安全性の検査」、「不整合の可視化」、「局所化」の機構を提供する。そして一般化が選択された場合にこれらの機構が MG-操作列の要求に対する処理の入れ子として呼び出される。

最後に、MG-操作列が適用される。図 8 は一般化の例を示している。



図7 不整合となる箇所の表示と一般化の候補の提示

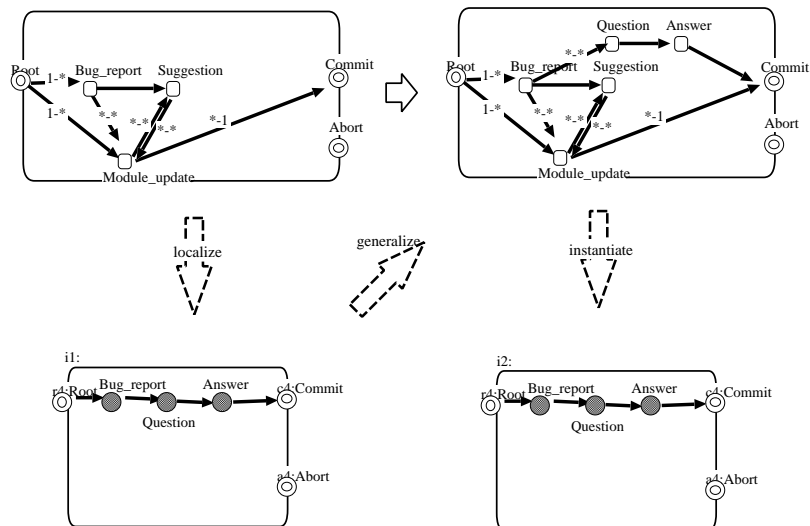


図8 プロセスインスタンスの一般化の例

5. 電子メール蓄積への適用結果

我々は、提案する手法をソフトウェア開発プロジェクトで流れた電子メールの蓄積に適用し、その効果を評価した。約2週間に流れた130通の電子メールを解析した。

各電子メールに対し、その内容を考慮しMG-クラスを割り当て、各電子メールをMGと想定した。今回は局所化と一般化の効果を計るため、はじめは空のMG-テンプレートのみを用意し、局所化と一般化の手法を用いてMG-テンプレートを構築していった。また、電子メールのヘッダから参照関係を導き、それをTDと想定した。そして各MGの生成がインスタンス化、局所化、一般化のいずれの契機となったかを分類し、その個数を調べた。各電子メールをMGと想定し

ており、またMGの生成はインスタンス化、局所化、一般化のいずれかの契機となりうるため、これらの契機の総数は130である。

図9に解析の結果を示す。図のグラフは、単位期間にやり取りされた電子メールのうちインスタンス化、局所化、一般化それぞれの契機となった電子メールの割合を示すものである。単位期間は2日とした。図から、局所化により生成されたMGが多くの割合で存在することから、局所化の必要性が分かる。一般化の割合は大部分が全体の10%以内であり、一般化の結果生成されたMG-テンプレートからのインスタンス化の割合は一般化よりも大きく、大部分が全体の20%以上である。このことから、一般化を用いることにより一般化の約2倍のMGが通常のワークフロープロセスと同様の支援を受けることができることが観察でき

る。また、時間の進行とともに、一般化の割合はわずかながら減少する傾向にあり、インスタンス化の割合は概して増加する傾向にある。このことから、長期的なプロセスになればさらに一般化の有効性が増すと推測できる。図8は、解析で見つかった一般化の一例である。

6. 関連研究

協調作業支援における要求の一つとして、予期しない状況への適応があげられる。現実の世界は頻繁に変化する環境や予測困難な人間の行動にさらされる。プロセス管理技術においては、例外的な状況に対応するための動的ワークフロー¹⁸⁾や適応可能ワークフロー¹⁹⁾が提案されている。文献⁶⁾では、木構造を持つ作業のトランザクションに対し、形式的な分割結合操作を定義することで動的に構造を変更する手法を提案している。文献⁶⁾では、プロセスの仕様を部分的に記述したものを蓄積しておき、例外時にその記述を詳細化することによって適応する方法が導入されている。さらにこの文献では、多数の作業者にコピー可能なアクティビティや、定められた期間に何度でも実行可能なアクティビティという拡張を行なっている。このように適応の形式化は議論されているが、設計プロセス、つまり議論の結果や組織間の合意といった、プロセスの設計の背景にある人間の意思決定のプロセスを利用することは考慮されていない。これに対し本論文の手法は、設計プロセスをも統一的に扱うことで設計プロセスの柔軟な実行を許し、設計プロセスとプロセスインスタンスとの整合性を管理するため、予期しない状況をプロセス仕様の変更を含めて処理できる。文献³⁾では過去のプロセスを拡張したり、ECAルールを用いて例外処理を記述することにより再利用性を促進する方法を提案しているが、コミュニケーションプロセスにおいては、あらかじめ定義される必要のあるECAルールは用いることができない。

文献¹⁰⁾では、プロセス仕様の変更を柔軟に管理し、プロセスインスタンスに柔軟に関連付ける方法が提案されている。積極的伝播 (eager propagation) ではプロセス仕様の変更が即時に全てのプロセスインスタンスに繁栄され、消極的伝播 (lazy propagation) では既存のプロセスインスタンスには反映しない方法である。一般化に相当する上向き伝播 (upward propagation) も提案されている。この手法は上向き伝播を導入している部分は本論文の手法と同じ主張であるが、本論文の手法は、一旦局所化されたMGも一般化することにより再利用を可能にすることにより、いわば局所化

されたMGもMG-テンプレートの一部と考えて積極的に利用するという具体的な利用法を示し、その効果を確認した。また本手法は局所化と一般化を設計プロセスにも適用する部分が文献¹⁰⁾と異なる。

設計プロセスを支援するシステムの1つとして、意思決定のための議論を支援するシステムが考えられる。が考えられる。文献^{1),5),11),12)}では「提案」、「約束」、「取り消し」といった発言をプロセスの生成、または進行と見なしている。つまり発言プロセスを支援するシステムである。議論の構造を記録することを目的としたシステム⁴⁾では、発言は議論の状態を表すととらえている。また、ゴール指向議論モデル^{15),17)}は、問題や要求を、部分問題 (subgoal) や解決 (achieve)、評価 (support, deny) といった語彙を用いて詳細化していく。しかしこれらのモデルは、議論の結果得られる成果物をプロセス仕様に限定していないため、議論の記録を動的にプロセス仕様に変換する必要がある設計プロセスに用いるには不十分である。文献¹⁴⁾では特に、“Has-Subtask”、“Has-Temporal-Relationship”、“Has-Attribute”といった、プロセスを具体化するための語彙を用意している。しかしこの手法では設計プロセス仕様が固定されているため、設計プロセス仕様自体に例外的な状況が発生した場合の対応が難しい。設計プロセスはプロセスインスタンスの進捗状況などの要因に影響されることが考えられるので、本手法のように局所化や一般化といった手法を用いて設計プロセス自体の例外的な状況に対応できる必要がある。

7. おわりに

M-Transシステムは、MGとMG-テンプレートを柔軟に管理することにより、コミュニケーションプロセスにおける設計プロセスとプロセスインスタンスを整合性を保ったまま協調作業の予測困難な状況に対応できるプロセス管理システムである。本論文では、局所化されたMGの記録と一般化を組み合わせてMG-テンプレートとMGの整合性を保つ方法を示し、電子メールの蓄積に適用して有効性を調べた結果、プロセスインスタンスの局所化が頻繁に発生し、プロセスインスタンスの一般化用いてプロセス仕様を獲得する方法が有効に利用されることを示した。M-Transシステムは、ワークフロープロセス、発言プロセスも扱うことが可能であるため、広い応用領域を持つことが考えられる。

参考文献

- 1) Bogia, D. P., Tolone, W. J., Kaplan, S. M.

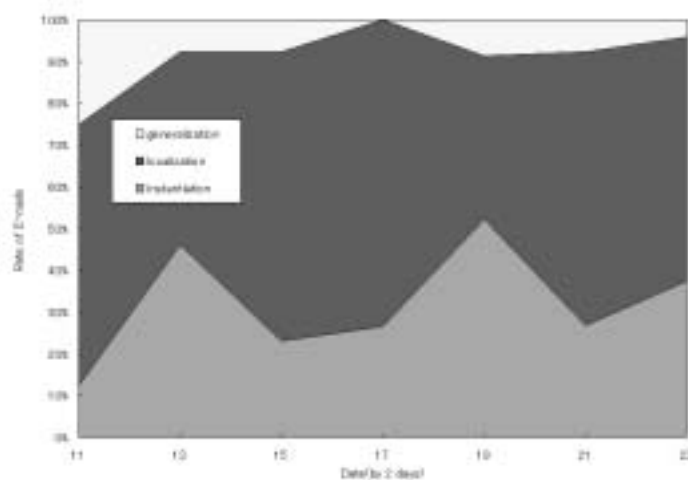


図9 インスタンス化, 局所化, 一般化の占める割合の推移

- and de la Tribouille, E.: Supporting dynamic interdependencies among collaborative activities, *Proc. ACM Conf. Organizational Computing Systems*, pp. 108–118 (1993).
- 2) Casati, F., Ceri, S., Parabosci, S. and Pozzi, G.: Specification and Implementation of Exceptions in Workflow Management Systems, *ACM Trans. Database Systems*, 3, Vol. 24, pp. 401–451 (1999).
 - 3) Chiu, D. K. W., Karlapalem, K. and Li, Q.: E-ADOME: A Framework for enacting E-Services, *Proc. 1st Workshop on Technologies for E-Services*, Egypt (2000).
 - 4) Conklin, J. and Begeman, M. L.: gIBIS: A Hypertext Tool for Exploratory Policy Discussion, *ACM Trans. Office Information Systems*, 4, Vol. 6, pp. 303–331 (1988).
 - 5) Flores, F., Graves, M., Hartfield, B. and Wingrad, T.: Computer Systems and the Design of Organizational Interaction, *ACM Trans. Office Information Systems*, 2, Vol. 6, pp. 153–172 (1988).
 - 6) Georgakopoulos, D., Schuster, H., Baker, D. and Cichocki, A.: Managing Escalation of Collaboration Processes in Crisis Mitigation Situations, *Proc. 16th Int'l Conf. Data Engineering*, pp. 45–56 (2000).
 - 7) in chief, E. E., Number, V., Hagen, P. and Alonso, G.: Flexible Exception Handling in the OPERA Process Support System (1998).
 - 8) Inoue, S. and Iwaihara, M.: Structured Message Management for Group Interaction, *Proc. Int'l Workshop. New Database Technologies for CSCW and Spatio-Temporal Data Management (NewDB'98)*, Singapore (1998).
 - 9) Inoue, S. and Iwaihara, M.: Adapting Transactions to Exceptional Situations Using Structured Messages, *Proc. Int'l Symp. Database Applications in Non-Traditional Environments (DANTE'99)*, Kyoto, IEEE Press, pp. 264–271 (1999).
 - 10) Joeris, G. and Herzog, O.: Managing Evolving Workflow Specifications, *Proc. of 3rd IF-CIS Int'l Conference on Cooperative Information Systems (CoopIS '98)*, New York (1998).
 - 11) Kaplan, S. M., Carrol, A. M. and MacGregor, K. J.: Supporting Collaborative Process with ConversationBuilder, *Proc. ACM Conf. Organizational Computing Systems*, pp. 69–79 (1991).
 - 12) Kaplan, S. M., Tolone, W. J., Bogia, D. P. and Bignoli, C.: Flexible, Active Support for Collaborative Work with ConversationBuilder, *Proc. ACM Conf. Computer-supported Cooperative Work*, pp. 378–385 (1992).
 - 13) Kellner, M. I. et al.: Software Process Modeling Example Problem, *Proc. 6th Int'l Software Process Workshop*, pp. 19–29 (1990).
 - 14) Klein, M.: iDCSS: Integrating Workflow, Conflict and Rationale-based Concurrent Engineering Coordination Technologies, *CERAs* (1995).
 - 15) Lee, J.: SIBYL: A tool for Managing Group Decision Rationale, *Proc. Conf. Computer-supported collaborative work*, pp. 79–92 (1990).
 - 16) Liu, L. and Pu, C.: Methodological Restructuring of Complex Workflow Activities, *Proc. 14th Int'l Conf. Data Engineering*, California, pp. 342–350 (1998).
 - 17) Mylopoulos, J., Chung, L. and Nixon, B.: Representing and Using Nonfunctional Require-

- ments: A Process-Oriented Approach (1992).
- 18) Reichert, M. and Dadam, P.: ADEPT_{flex} – Supporting Dynamic Changes of Workflows Without Loosing Control, *Journal of Intelligent Information Systems* (1997).
 - 19) Sheth, A.: From Contemporary Workflow Process Automation to Adaptive and Dynamic Work Activity Coordination and Collaboration, *Proc. Int'l Conf. Database and Expert Systems Applications*, Toulouse, pp. 24–27 (1997).
 - 20) 井上創造, 岩井原瑞穂: 非同期型コミュニケーションにおけるトランザクションの動的構築, *情報処理学会論文誌: データベース*, 40, No. SIG 8 (TOD4), pp. 1–12 (1999).
-