相互関連タスクの実時間処理のための連携的実行制御機構

渡辺裕之[†] 渡邉豊英[††]

オブジェクト指向パラダイムは，実世界上の様々な実体を独立したオブジェクトとして自然にモデル化可能な枠組みを提供するが，モデル化されたオブジェクトを動的に制御することには適していない．オブジェクト指向パラダイムでは，オブジェクトの静的な性質はクラスによって事前に定義されるが，オブジェクト間の動的な相互作用には視点がおかれていないからである．特に，秘書が行う効率的で柔軟な処理を計算機上に実現する場合，複雑で多岐にわたる秘書の仕事を自然にモデル化する枠組みに加えて，複数の関連し合う仕事を効果的に制御・実行させる機構が必要である．本稿では，秘書モデルの構築の観点から，オブジェクトをタスクの実行主体と見なし，協調性と連携性の概念に基づいたタスクの柔軟な実行制御機構について言及する．本稿で述べる制御機構はイベント・ハンドラとタスク・スケジューラから構成される．イベント・ハンドラはタスク間の相互作用を静的な視点で実現し，タスク・スケジューラは現在の実行状況に応じてタスクを動的に制御する．これら2つの構成要素は，連続し，かつ相互に関連し合うタスクの効率的な実行を可能にし，加えて，要求として与えられる外界の状況変化と資源の制約などの内的要件の2つに対しても柔軟に対応する．また，タスクの失敗処理におけるイベント・ハンドラとタスク・スケジューラの連携についても言及する．

A Cooperative and Successive Control Model for Real Time Processing of Interrelated Tasks

HIROYUKI WATANABE[†] and TOYOHIDE WATANABE[††]

The object-oriented paradigm is very successful to model various kinds of entities as independent objects. However, this paradigm is not sufficient to control independent objects dynamically. The reason is that the object-oriented paradigm does not focus on the dynamic interactions among objects. With a view to constructing a secretary model which can manage many different jobs effectively and flexibly like a well human secretary, it is important not only to model various kinds of operational objects but also to establish a flexible control mechanism for managing the interrelated objects effectively. In this paper, we regard each object as a task executer, and address a dynamic control mechanism among interrelated tasks on the basis of the concepts of cooperativeness and successiveness. In particular, we focus on the event handling and task scheduling: the event handler organizes the interrelated tasks derived from the requested event, and the task scheduler controls them dynamically. These two components are important to execute the interrelated tasks effectively and also powerful to cope with both the changes of outside situations given as requests and the conditions of inside states like the resource restrictions. In addition, we describe the cooperation between the event handler and task scheduler through the task failure handling.

1. はじめに

オブジェクト指向パラダイムはソフトウェア・モデリング，プログラミング方法論，システム構築法などの計算機技術のひとつとして確立されてきた[1),2)]．このパラダイムでは実世界の実体を個々に独立したオブジェクトとして定義し，メッセージ・パッシングを介した相互作用により処理/手続きを実現する．このような枠組みは実世界の様々な現象を自然にモデル化可能とし，モデル化された現象を論理的に設計し，設計されたシステムやツールを効果的に計算機上に実装するのに適している．しかし，オブジェクト指向パラダイムでは，オブジェクトが本質的に他とは独立に存在するとして定義されるにもかかわらず，個々のオブジェクトを動的に動作させることには適していない[3)]．これはオブジェクトの性質があらかじめクラス定義によって固定化され，またオブジェクト間の相互作用がメッセージ・パッシングにのみ依存しているからである．

† 富士通エフ・アイ・ビー株式会社
FUJITSU FACOM INFORMATION PROCESSING CORPORATION
†† 名古屋大学大学院 工学研究科 情報工学専攻
Department of Information Engineering, Graduate School of Engineering, Nagoya University

本稿では，タスク間の連携性と協調性の概念に基づいて，タスクの効率的な実行制御機構について言及する．我々は優れた秘書のように，多くの異なった仕事を効率的かつ柔軟に処理可能とするモデルの構築を目指しており，本稿で述べる実行制御機構は研究の最終目標に対する第一段階である．オブジェクト指向パラダイムは，秘書が扱う様々な処理対象や処理対象間の関連のモデル化に有効である．しかし，秘書は複数の仕事を要求や状況に応じて知的に処理する．もし，仕事が互いに関連し合っていれば，それらを効果的に集約させたり，瞬時に同期させたりする．このような秘書の処理能力をモデル化するためには，オブジェクト指向パラダイムの枠組みに加えて，複数の関連し合うタスクを効果的・効率的に制御，実行する機構が必要となる．

このような要件に対して，主に分散環境において，独立したオブジェクトを自律的に動作するエージェントと見なし，複数のエージェントによる問題解決戦略がさかんに議論されている[4)~8)]．このアプローチでは，個々のエージェントが目標を遂行するための協調動作/競合回避機構が重要となる．オブジェクトを自律的なエージェントと見なすことは，複数のオブジェクトを動的に動作させることを可能にするが，このような枠組みは秘書モデルの構築に適しているとはいえない．なぜならば，優れた秘書は，新しく生成された仕事や一時的に中断された仕事を，自らの処理状況や環境に応じて敏速に実行/再実行できるからである．マルチ・エージェントに基づく分散協調モデルは，制御機構を個々のエージェントに委譲してしまうために，分散された複数のエージェントが様々な状況や環境の変化に敏速に対応することができない．

一方，複雑で多岐にわたる処理対象に対する連携的な操作は，データベースにおけるトランザクションの直列実行可能性の問題としてとらえることもできる．特に，長時間トランザクション管理の研究は一貫性のある制御機構に焦点を当てている[9)]．様々なデータに対する複雑な操作を管理するという視点は，我々の研究目的に非常に似通っている．しかし，多くのアプローチが長時間トランザクションを階層構造として制御している．すなわち，上位のトランザションはすべての下位のアクション（またはトランザクション）から送られる委任メッセージを待たなければならない．少なくとも，このような制御機構は全体のスループットを小さくする我々の研究目的に適合しているとはいえない．一方，SAGA は非階層構造を持つ長時間トランザクション・モデルとして提案されている[9)]．SAGA の補償トランザクションの概念は，データベースの無矛盾性を保つのに有効であるが，すべてのトランザクションに対応する補償トランザクションを用意しなければならず，秘書業務を容易にモデル化することは困難である．

複数の互いに関連し合うタスク間の協調性と連携性の観点から，個々の要求に関連するタスクを区別し，それらの間の実行順序と相互作用を制御する機構が必要になる．加えて，実行されるタスク間の同期制御とコミット/アボート制御機構は高いスループットを実現する視点から設計，実装されなければならない．これらの要件より，我々の実行制御機構は主にイベント・ハンドラとタスク・スケジューラから構成される．イベント・ハンドラはユーザからの要求をイベントとして受け取り，要求に応じた最適なタスクをその相互関係に基づいて選択，生成する．タスク・スケジューラは現在の実行状況に従って，すでに順序付けされたタスクと新しく生成されたタスク間の実行順序を決定し，選択されたタスクを実行する．いいかえれば，イベント・ハンドラは静的な視点に基づいてタスク間の相互関係を解析し，タスク・スケジューラはタスクの動的な振舞いに基づいてタスクのスケジューリングを行い，実行する．

本稿は以下のように構成されている．2 章では，秘書業務をモデル化するための枠組みと我々の実行制御機構の概略について述べる．3 章では，要求されたイベントとタスク間の関係を解析するイベント・ハンドラの構造について述べる．4 章では，タスクの動的な振舞いに基づいて実際にタスクを実行するタスク・スケジューラについて述べる．5 章ではタスクの実行が失敗した場合を考え，その処理のためのイベント・ハンドラとタスク・スケジューラの連携について述べる．6 章では我々の実行制御機構について考察し，7 章でまとめと今後の課題を述べる．なお，我々は Smalltalk-80 上でプロトタイプ・システムを作成し，本稿で述べる実行制御機構の動作を確認している．3 章および 4 章，5 章では，我々のプロトタイプ上での実装についても言及する．

2. モデル化と実行制御機構

我々の研究目的は秘書モデルの構築のためにタスクの動的な実行制御機構を確立することである．たとえば，優れた秘書は要求に応じて実行できる仕事を選択し，選択された仕事を状況に従って知的に処理する．秘書モデルの構築の観点から，我々の実行制御機構は適切なタスクを要求に応じて選択し，つねに最適な手

順で実行しなければならない．さらに，関連し合うタスクの実行はそれらの相互関係に基づいて柔軟に制御されなければならない．我々の実行制御機構に関する要件は以下のようにまとめられる．

- 与えられた要求に応じて適切なタスクを選択する．
- 選択されたタスクをつねに全体のスループットが小さくなるように実行する．
- 実行されるタスクをタスク間の相互関係と現在の状況に基づいて逐次的に実行するか，または並行的に評価する．

このような要件を満たす実行制御機構として，オブジェクト指向パラダイムにおけるオブジェクトのように，個々のオブジェクトが独立であり，かつオブジェクト間の相互作用がメッセージ・パッシングで実現されるという枠組みでは対処不可能である．すなわち，個々のタスクが関連している場合も少なくなく，それらを連携的に効率良く処理する必要がある．また，オブジェクト（またはエージェント）自身に自律的な能力をすべて分散させると，様々な実行手順を制御し，効率性を向上させることが困難である．さらに，タスクの実行が失敗した場合や何らかの原因でタスクの実行終了が遅延している場合，それを大域的に監視し，状況に合わせて適切な処理を実施する必要があり，システム全体を集中的に管理，制御する機構の方が優れている．このような視点に基づいて，処理の実行器とは別に，システムの制御を司るモジュールの下に我々の実行制御機構を実現するというアプローチを採用する．

2.1 秘書業務のモデル化

実行制御機構を実現するためには，多岐にわたる秘書業務をモデル化する枠組みが必要となる．はじめに，秘書業務のモデル化のためのいくつかの構成要素を定義する．

- **イベント** ··· イベントはいくつかのタスクを実行するきっかけとなる．実際には，ユーザからの要求や外界の状況変化などがイベントとして定義される．イベントにはその種類や実行条件などの情報が含まれ，それらの情報に基づいて複数のタスクが生成される．
- **アクション** ··· アクションはイベントから導出され，実行されるタスクの集合を生成する．すなわち，アクションはイベントからタスクを生成する仲介者である．概念的には，アクションはイベントに応じて秘書が行う仕事の段取りを表し，結果的に複数のタスクの実行を引き起す．
- **タスク** ··· タスクは自己完結した1つのコマンドであり，すべての秘書業務はこれらのタスクの集まりとして表現される．我々のモデルでは，タスクはオブジェクトへのメッセージとして定義される．すなわち，処理対象への操作がタスクとして定義される．
- **オブジェクト** ··· オブジェクトはタスクの実行主体である．オブジェクトはタスクをメッセージとして受信し，それを他のオブジェクトと並行的にも実行できる．

これらの構成要素を用いて秘書業務をモデル化する．例として，会議を準備する要求が発生した場合を考える．**図1** にこの要求に関連するタスクを示す．図1では，六角形がイベントを表し，四角形がタスクを表している．また，矢印はそれらの間の実行順序を表している．一般に，ある業務に関連するタスクはつねに1つのイベントがきっかけとなって実行されるとは限らない．たとえば，図1の場合，タスク「会議室予約」はイベント「会議開催の決定」から直接生成され，タスク「出欠確認処理」はイベント「案内状の返事到着」をきっかけとしてその実行を開始できる．図1のタスクは**図2** に示される2つのアクションに分割できる．アクションはこのようにイベントとタスク間の関係を明確にする．さらに，タスクとオブジェクトの関係を**図3** に示す．図3では，四角形がタスクを表し，円がオブジェクトを表している．たとえば，「会議室」や「電子メール・システム」は秘書が扱う処理対象であ

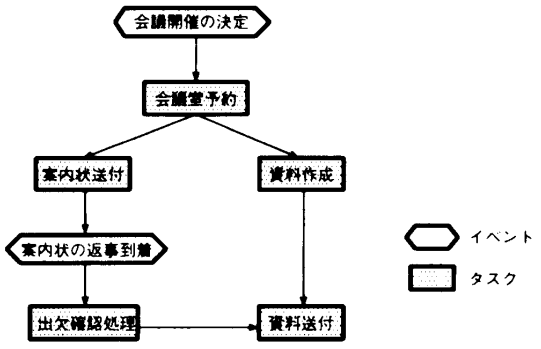

図1 秘書業務の例
Fig. 1 Tasks for meeting preparation.

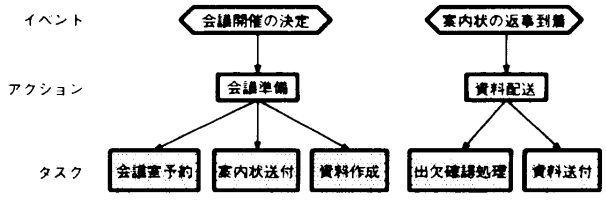

図2 イベント，アクション，タスクの関係
Fig. 2 Correspondence among events, actions and tasks.

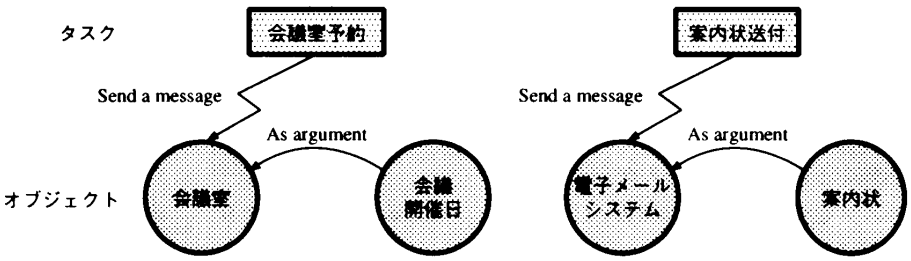

図3 タスクとオブジェクトの関係
Fig. 3 Correspondence between tasks and objects.

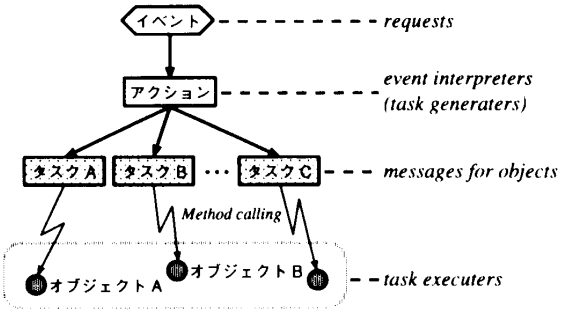

図4 構成要素間の関係
Fig. 4 Relationship among composite concepts.

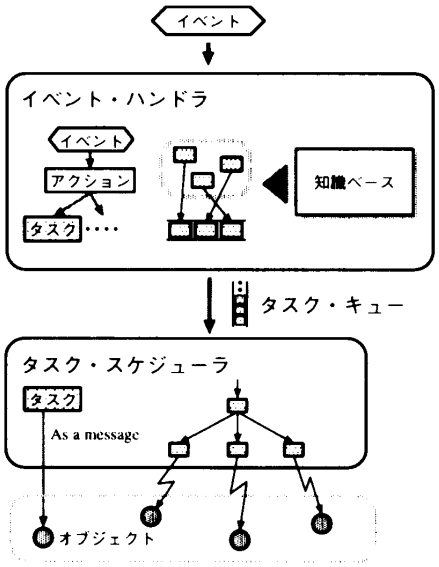

図5 実行制御機構の概要
Fig. 5 System architecture.

り,「会議室予約」や「案内状送付」がそれらの処理対象に対する操作である．これらの構成要素は秘書業務のモデル化を容易にするだけでなく，要求されたイベントから互いに関連し合うタスクへの変換をも容易にする．これらの構成要素の関係を図4に示す．

2.2 実行制御機構

要求としてのイベントは関連する複数のタスクへと変換され，それらのタスクは最終的に並行に動作するオブジェクトへと割り当てられて実行される．この変換の効率性に加えて，個々のオブジェクトは与えられたタスクを正しく遂行するために並行的かつ同期的に制御されなければならない．しかし，実行段階において，この事前に割り当てられたタスクの実行順序はつねに効率的な実行を可能とするとは限らない．たとえ，他のタスクの実行が対応するオブジェクトによって完全に終了されていないとしても，現在実行されているタスクの次に順序づけされたタスクを先に実行した方がよい場合もある．すなわち，要求に応じて適切なタスクを選択する機構に加えて，タスク間の実行順序を保証した手続きのもとで動的に制御する機構も必要である[13]．この考察により，我々の実行制御機構は図5のようになる．我々の実行制御機構は以下の2つの要素から構成される．

- **イベント・ハンドラ** … すべてのイベントはイベント・ハンドラによって処理される．イベント・ハンドラは要求されたイベントから適切なアクションを選択し，アクションを介して実際に実行するタスクを決定する．さらに，実行順序はこれらの関連し合うタスクに割り当てられる．タスクは最終的に実行順序に従ってタスク・キューに挿入される．すなわち，イベント・ハンドラは対応するアクションを介してイベントからタスクを生成し，またこれらの生成されたタスクを静的な視点で順序づけする．
- **タスク・スケジューラ** … タスクは現在の状況に従って効率的に制御されるべきであり，静的に割り当てられた順序に従う必要はない．そのため，タスク・スケジューラはタスクの動的な相互作用を解析し，それらの再スケジュールを事前に定義された実行順序に基づいて行う．実行可能なタスクはつねに現在の状況に合わせて選択され，保証された手続きの下に並行的に実行される．加えて，タスク・スケジューラはタスクの実行主体であるオブジェクトのすべての振舞いを管理する．すなわち，タスク・スケジューラはタスクの実行を現在の状況に従って制御し，必要ならばタスク間の

実行順序の再スケジュールを行う．

3. タスク生成と順序づけ

我々の枠組みではタスク間の相互作用はあらかじめイベント・ハンドラによって静的に解析される．イベント・ハンドラの役割は要求としてのイベントに従って適切なアクションを決定し，決定されたアクションに従って実際に実行するタスクを選択することである．さらに，タスク間の実行順序も決定する．イベントからタスクへの変換処理は知識ベースに基づいて行われるが，その知識は仕事の種類や要求に応じたイベントやアクション，タスク間の関係を的確に表現しなければならない．これらの関係を状態遷移図と AND-OR グラフを用いて表現する．

3.1 状態遷移図

一般に，いくつかのイベント間には，事前に定義可能な順序関係が存在する．たとえば，案内状を送付したならば，次にイベント「案内状の返事到着」の発生を予測できる．状態遷移図はイベント間の評価順序とイベント-アクション間の関係を表現する．**図6** に状態遷移図の例を示す．図 6 は会議に関する仕事の状態遷移図を表している．ここでは，S_i $(i = 1, \ldots)$ が状態であり，矢印 (↓) がイベントとアクションの対応を示す．すなわち，矢印の上の引数がイベントであり，下の引数が対応するアクションである．図6では，もしイベント「会議開催の決定」が発生したなら，状態は S1 に推移し，アクション「会議準備」が開始される．

すでに提案されている多くのオブジェクト指向パラダイムに基づいたソフトウェア・モデリングの手法においても，状態遷移図はイベント間の評価順序とイベントに対する動作を記述するために用いられている[2),10)]．ところが，多くの手法において，状態遷移図は個々のオブジェクトの内部状態の遷移だけを表現しているが，多くのオブジェクトが個々に独自の状態を持つならば，複数のオブジェクトの集合が様々な要求や状況の変化に敏速に対応することが困難になる．

我々が用いる状態遷移図はイベント・ハンドラの動作を表すために使用され，要求としてのイベントはすべてイベント・ハンドラによって処理される．加えて，1 つの状態遷移図は概念的に 1 つの自己完結的な仕事を表し，その行動の遷移を表現している．たとえば，図 6 の状態遷移図は会議準備からその終了までの仕事を表し，次々と要求されるイベントに応じてどのアクションを実行するかを表している．イベント・ハンドラは現在実行中の仕事に応じて，複数の状態遷移図を持つ．イベント・ハンドラがイベントを受け取ると，どの状態遷移図がイベントを受理するかを決定して状態を遷移させるか，イベントが新しい仕事を示しているならば新しい状態遷移図を生成する．

3.2 AND-OR グラフ

状態遷移図によってイベントと関連づけられたアクションは複数のタスクを生成する．AND-OR グラフはアクションと関連するタスク間の関係を表現し，タスク間の実行順序を表す．AND-OR グラフの例を**図7** に示す．図 7 はイベント「会議開催の決定」から生成されるタスク間の関係を表している．AND-OR グラフは以下に示す 4 つの要素から構成される．

- **AND ノード** … AND ノードはそのすべての下位ノードが選択されることを表す．図 7 では，「案内状」の AND ノードは「案内状作成」と「案内状送付」の両方のタスクが選択されるべきことを示している．
- **OR ノード** … OR ノードはたった 1 つの下位ノードが選択されることを表す．適切な選択のために，OR ノード下のすべての枝に特定の条件が

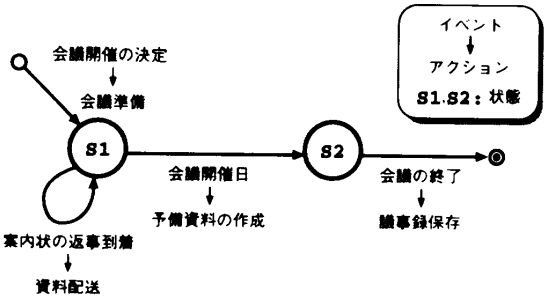

図6 状態遷移図の例
Fig. 6 Example of state transition graph.

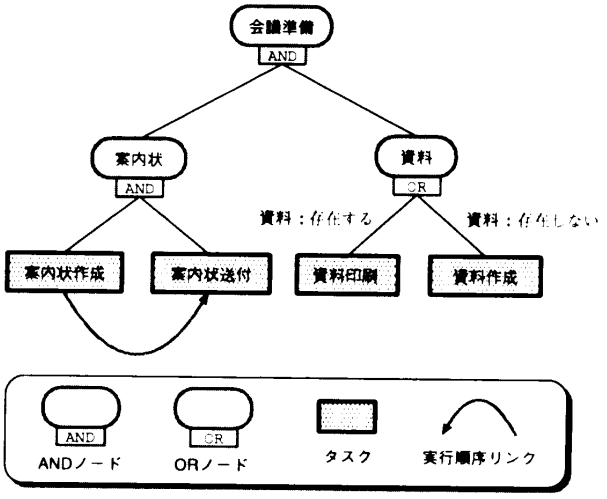

図7 AND-OR グラフの例
Fig. 7 Example of AND-OR graph.

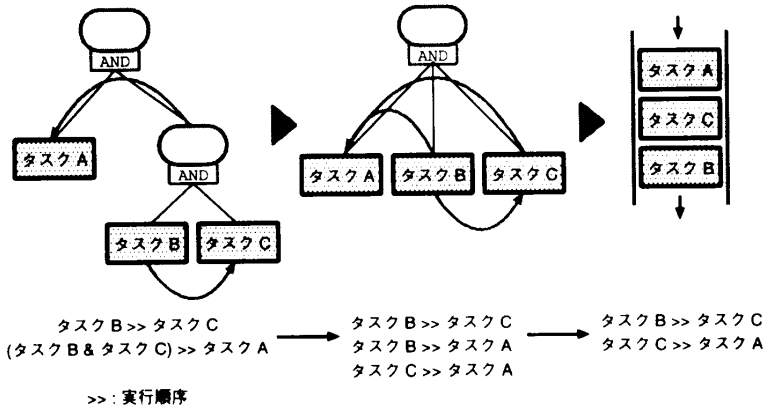

図8 実行順序リンク
Fig. 8 Ordering tasks using directed branches.

付帯される．図7では，「資料」のORノードは，もし資料が存在するならば，タスク「資料印刷」が選択されることを示す．これらのORノードの選択に関する情報はイベントに含まれる．

- **タスク** … すべてのタスクは必ず終端ノードに置かれる．すなわち，タスクは下位ノードを持たないノードである．
- **実行順序リンク** … ノード間に張られた方向付けリンクはタスク間の実行順序を表す．図7では，タスク「案内状作成」はタスク「案内状送付」が実行される前に実行される．非終端ノード間に実行順序リンクが張られたときは，開始ノードのすべての下位ノードが終了ノードのすべての下位ノードに先だって実行されることを示す（**図8**を参照）．

状態遷移図では，イベントを1つのアクションとしか関連づけられないが，ANDノードを用いることで，概念的な複数のアクションを論理的な1つのアクションと見なすことができる．たとえば，図7では，「案内状」と「資料」の2つの概念的に異なるアクションが，「会議準備」のANDノードによって1つにまとめられている．

多くのソフトウェア・モデリングの手法では，イベントに対応するアクションの詳細を記述するためにデータ・フロー図を用いている[2),10)]．データ・フロー図はタスク間の実行順序を表現可能であるが，タスクの選択性を表現することはできない．AND-ORグラフは，タスク間の実行順序を表すと同時に，様々な要求に応じたタスクの選択を可能にする．

3.3 イベント・ハンドラ

イベント・ハンドラは，状態遷移図やAND-ORグラフを用いて，要求されたイベントから導出されるタスクを生成し，生成されたタスク間の実行順序を決定する．イベント・ハンドラの構造を**図9**に示す．イベント・ハンドラは以下に示す4つの構成要素からなる．

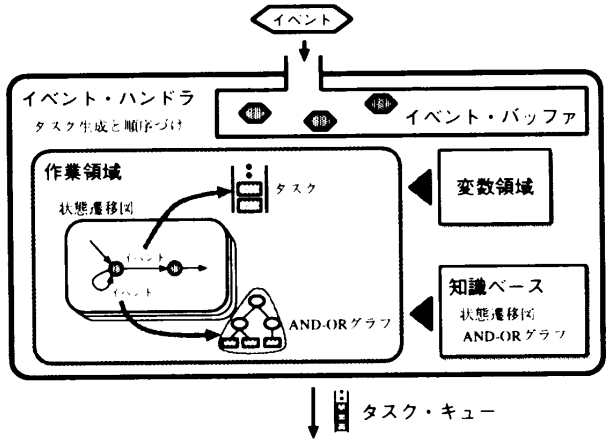

図9 イベント・ハンドラの構造
Fig. 9 Structure of event handler.

- **イベント・バッファ** … イベント・バッファはまだ受理されていないイベントを保持する．もしイベント・バッファに緊急のイベントが到達したら，イベント・ハンドラは優先的にそのイベントを受理する．
- **知識ベース** … 知識ベースは状態遷移図とAND-ORグラフを調達するデータベースである．イベント・ハンドラは要求されたイベントに従って状態遷移図とAND-ORグラフを知識ベースから引き出す．
- **変数領域** … 変数領域は現在の状況を表す変数の集合である．変数はAND-ORグラフ中のORノードの選択に使われ，ユーザから要求されたイベントによって更新される．もし適当な変数が存在しなければ，イベント・ハンドラはどの選択が適当かをユーザに尋ねる．
- **作業領域** … 作業領域は現在実行中の行動を表す

複数の状態遷移図を保持している．効率性のために，イベント・ハンドラは将来受け取るかもしれないイベントをあらかじめ解析し，関連するタスクを生成する．たとえば，案内状のイベント「案内状の返事到着」に対応するタスクは，案内状の返事が到着する前に生成される．このため，作業領域はAND-ORグラフや解析されたタスクの集合も保持している．

我々がSmalltalk-80上で作成したプロトタイプでは，それぞれの構成要素はイベント・ハンドラが持つ部品として実装される．イベント・ハンドラはイベント・バッファよりイベントを取り出し，状態遷移図やAND-ORグラフを解析するという動作サイクルを繰り返す．

4. タスク実行とオブジェクト制御

タスクの実行はタスク・スケジューラによって動的に制御される．すなわち，タスク・スケジューラは，イベント・ハンドラによって生成されたタスクを状況に応じて並行的に評価したり，逐次的に実行したりする．また，イベント・ハンドラによって静的に割り当てられたタスクの実行順序に基づいて，タスクの再スケジューリングを行う．我々の枠組みでは，タスクはオブジェクトへの操作であるので，同じオブジェクトを操作する2つのタスクの実行は，そのオブジェクトをつねに無矛盾な状態に保つように制御しなければならない[9),11)]．たとえば，たとえ，タスクの再スケジュールが必要であっても，タスク「案内状作成」はつねにタスク「案内状送付」よりも前に実行されなければならない．また，同じ「電子メール・システム」を用いる2つのタスクは，たとえ両者の間に明確な順序関係が存在しなかったとしても，排他的に制御される必要がある．このようなタスクの実行制御における一貫性を明らかにするため，本章では最初にタスクの実行主体であるオブジェクトの性質とオブジェクトとタスクの関係を定義する．次に，タイムスタンプ順序法を用いたタスクの動的実行制御について述べ，最後にタスク・スケジューラの構造について述べる．

4.1 オブジェクトとタスク

我々のオブジェクトは単に受け取ったメッセージに応じて，そのメソッドを並行的に実行するタスクの実行主体である．一般的なオブジェクト指向パラダイムでは，オブジェクトの相互作用は一対一の同期通信であるメッセージ・パッシング機構によって実現され，あるオブジェクトは他のオブジェクトとメッセージ・パッシングによって積極的に通信し合う．しかし，我々のモデルでは，タスク間の相互作用はあらかじめイベント・ハンドラによって解析されているため，オブジェクトはメッセージとして受信したタスクを他のオブジェクトとは独立に実行するだけでよい．このオブジェクトの性質を明らかにするために，まず以下の2種類のメソッドを定義する．

- **参照メソッド**・・・あるオブジェクトの参照メソッドが実行されると，そのオブジェクトは自分自身の状態に応じた値を返す．参照メソッドは単にオブジェクトの状態を参照するだけであるので，実行後の状態は実行前の状態と同じである．これはある参照メソッドが連続して実行されれば，オブジェクトが同じ値を返すことを示している．加えて，すべての参照メソッドは必ず成功する．
- **更新メソッド**・・・あるオブジェクトの更新メソッドが実行されると，そのオブジェクトは「成功」か「失敗」かを示す値を返す．更新メソッドが成功すると，実行前の状態は実行後の状態に置き換えられる．更新メソッドが失敗すれば，実行後の状態は実行前の状態に戻される．これは更新メソッドで起こったエラーがオブジェクト自身によって修復されることを示す．

上の2種類のメソッドにより，オブジェクトは以下の性質を満たすものとする．

- すべてのメソッドは参照メソッドか更新メソッドとして定義される．
- あるオブジェクト内のすべてのメソッドは，他のオブジェクトの更新メソッドを呼び出すことができない．
- あるオブジェクト内の参照メソッドは，他のオブジェクトが引数として割り当てられない限り，呼び出すことができない．

これらの性質によって，タスク・スケジューラはすべてのオブジェクトの振舞いを管理することができる．一般的なオブジェクト指向パラダイムでは，このようなメソッドの性質に関する制約は存在しない．しかし，仮にあるオブジェクトが他のオブジェクトを更新することが許されれば，多くのオブジェクトが状態を次々と変化させることになる．個々のオブジェクトが並行的に動作することを考慮するならば，これは複数のオブジェクトの一貫性の保持を困難にする．いいかえれば，オブジェクト間のすべての相互作用はイベント・ハンドラやタスク・スケジューラによって管理されるべきであり，そのためにはすべてのオブジェクトが上の性質を満たさなければならない．

一方，タスクはオブジェクトの更新メソッドを呼び

出すメッセージとして定義される．実際には，更新メソッドを実行するレシーバとメソッド名およびいくつかの引数からなる．タスクの実行はレシーバの更新メソッドを呼び出し，同時に引数のすべての参照メソッドを呼び出すことを表す．すなわち，タスクの実行はレシーバとなるオブジェクトの状態だけを変化させ引数となるオブジェクトの状態を変化させない．

4.2 タイムスタンプ順序法

タスク・スケジューラはオブジェクトの振舞いに従って，互いに関連し合うタスクの再スケジューリングを行う．例として，あるタスクの実行に比較的多くの時間がかかった場合を考える．この場合，タスク・スケジューラはその時間のかかるタスクの終了を待たずに，他に実行可能なタスクを実行する．すなわち，イベント・ハンドラは静的な視点でしかタスクの実行順序を決定していないため，この実行順序は動的に変化する実行状態においてつねに最適であるとは限らない．このため，我々はタイムスタンプ順序法を導入する．タイムスタンプ順序法はタスクにタイムスタンプと呼ばれる数値を設定し，そのタイムスタンプに従って実行できるタスクを選択する．

タイムスタンプが持つべき性質を検討するために，特定のオブジェクトに関係して2つのタスクが連続して実行される場合を考える．この場合，特定のオブジェクト "X" に関係した2つのタスクの実行は**図10**に示される4種類に大別できる．"O(P)" はレシーバがオブジェクト "O"（すなわち，"O" が更新される）で，引数がオブジェクト "P"（すなわち，"P" が参照される）であるタスクを表し，"O(X) → P(X)" はタスク "O(X)" がタスク "P(X)" の実行に先だって実行されることを示している．図10では，参照-参照関係における2つのタスクの実行順序は，それらがけっして競合しないため，変更することができる．すなわち，"O(X) → P(X)" という実行順序でも，"P(X) → O(X)" という実行順序でも，関連するオブジェクト "O"，"P"，"X" の実行結果は同じである．しかし，参照-更新，更新-参照，更新-更新関係においては，2つのタスクは互いに競合するので，実行順序を変える

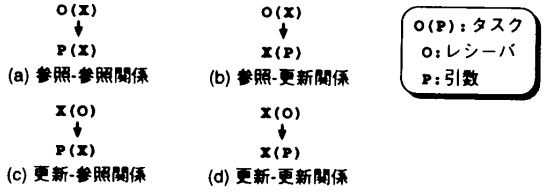

図10 2つのタスクの実行順序
Fig. 10 Execution order of two tasks.

ことはできない．たとえば，参照-更新関係において，"O(X) → X(P)" という実行順序と "X(P) → O(X)" という実行順序では，オブジェクト "O"，"X" の実行結果は明らかに異なる．以上の考察より，タイムスタンプの性質を以下のように定義できる．

[定義] タイムスタンプの性質

タスク t_{pre} はすでにタイムスタンプ $TS[t_{pre}]$ が設定され，かつまだ実行されていないものとする．まだタイムスタンプが設定されていないタスク t のタイムスタンプ $TS[t]$ は必ず以下の性質を満たさなければならない．

(1) もし t のレシーバが t_{pre} のレシーバまたは引数ならば $TS[t] > TS[t_{pre}]$ である．

(2) もし t の引数が t_{pre} のレシーバならば $TS[t] > TS[t_{pre}]$ である．

(3) $TS[t]$ と $TS[t_{pre}]$ はつねに $TS[t] \geq TS[t_{pre}]$ を満たす． □

性質(1)と(2)は図10から導かれ，性質(3)はタイムスタンプをすべてのタスクに設定する際に必要となる．タスク・スケジューラはこれらの性質を満たすタイムスタンプを各タスクに設定し，つねに最小のタイムスタンプを持つタスクを実行する．タイムスタンプ順序法により，オブジェクトの一貫性を保持しつつ，実行状況に応じた効率的なタスクの実行を可能にする．

4.3 タスク・スケジューラの構造

タスク・スケジューラはタイムスタンプ順序法を用いて，互いに関連し合うタスクを動的に制御する．タスク・スケジューラの構造を**図11**に示す．タスク・スケジューラは以下の3つの構成要素を保持している．

- **準備集合** ··· 準備集合は最小のタイムスタンプを持つタスク（いつでも実行できるタスク）を保持している．これらのタスクは関連するすべてのオブジェクトが活動中でなければ実行される．タスク・スケジューラはオブジェクトのメソッドを排他的に制御するため，活動中のオブジェクトにはメッセージを送らない．
- **実行待ちキュー** ··· 実行待ちキューはタスク・キューから取り出されたタスクを保持している．これらのタスクにはタイムスタンプが設定され，そのタイムスタンプに従って並べられている．準備集合が空になれば，タスク・スケジューラは最小のタイムスタンプを持つタスクを実行待ちキューより取り出し，新たな準備集合を生成する．
- **オブジェクト・モニタ** ··· オブジェクトの動作は

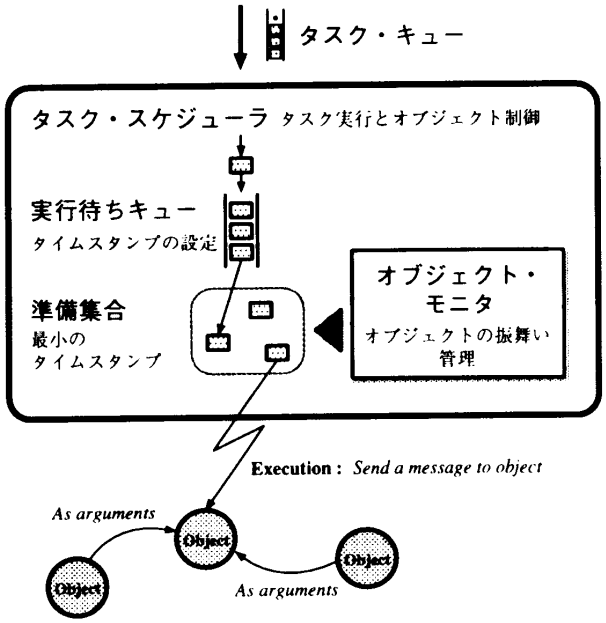

図 11 タスク・スケジューラの構造
Fig. 11 Structure of task scheduler.

つねにオブジェクト・モニタによって管理される．タスク・スケジューラはオブジェクト・モニタから提供される情報に従ってオブジェクトを排他的に制御する．

我々が Smalltalk-80 上で作成したプロトタイプでは，これらの構成要素は部品としてタスク・スケジューラが持つ．タスク・スケジューラは，タスク・キューより取り出したタスクを実行待ちキューに挿入し，準備集合中のタスクを実行するという動作サイクルを繰り返す．

5. 失敗処理

タスクが計算機資源を必要とする限り，タスクの実行失敗は避けることができない．予期せぬタスクの実行失敗は，他に関連するタスクの実行を停止させ，システムはユーザから要求された仕事を完全に遂行することができなくなる．たとえば，電子メール・システムの故障によりタスク「案内状送付」の実行が失敗した場合，タスク「出欠確認処理」やタスク「資料送付」は実行できなくなる．もしタスクがその実行に失敗したならば，システムは即座に代替タスクを準備するとともに，失敗したタスクの実行主体であるオブジェクトへの影響を取り除かなければならない．我々が作成したプロトタイプでは，失敗処理は次の段階を経て実行される（**図 12** を参照）．

(1) 失敗したタスクは失敗イベントを生成する（図 12 の (1)）．
(2) イベント・ハンドラは失敗イベントに従って代替タスクと取消タスクを決定する（図 12 の (2)）．
(3) タスク・スケジューラはイベント・ハンドラの指示に従って複数の取消タスクのロール・バックを行い，代替タスクを実行する（図 12 の (3)）．

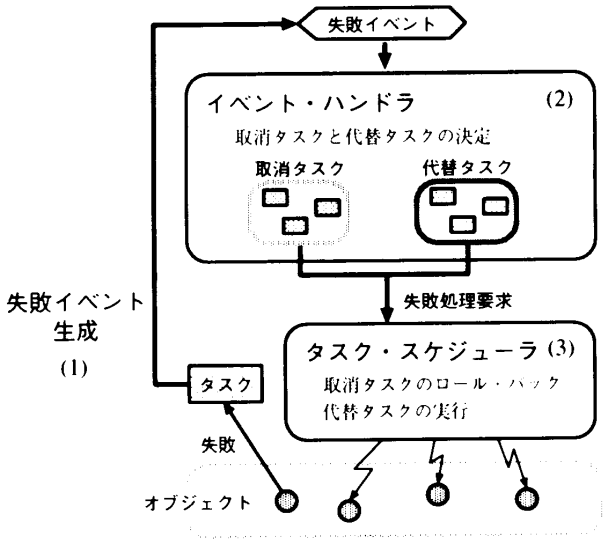

図 12 失敗処理
Fig. 12 Failure handling.

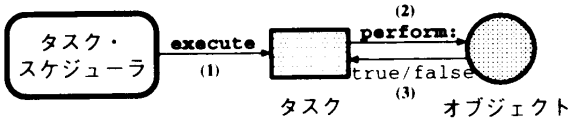

図 13 タスクの実行手順
Fig. 13 Task execution.

(1) 失敗イベントの発生

失敗イベントはタスクが実行に失敗したことを示し，同時に失敗に対処するアクションを引き起す．我々のプロトタイプでは，デフォルトで代替タスクを選択するようにイベントを発生するが，これらの失敗に対するアクションはいつも同じであるとは限らない．失敗イベントはユーザがタスクごとに異なる失敗イベントを定義できる方が有効である．このような定義を可能にするために，我々のプロトタイプではタスクは以下のように実行される（**図 13** を参照）．

(1) タスク・スケジューラはタスクに実行を要求する（図 13 の (1)）．
(2) 要求されたタスクはレシーバのオブジェクトにメッセージを送る（図 13 の (2)）．
(3) オブジェクトは対応する更新メソッドを実行し，「成功」か「失敗」かを示す値を返す（図 13 の (3)）．

このようなタスクの実行はタスクを個々に独立したプロセスとして扱うことを可能にする．すなわち，タスクは他のタスクやタスク・スケジューラから独立し

```
Object subclass: #Task
    instanceVariables: 'receiver method arguments failureEvent...'
    classVariables: ''
    poolDictionaries: ''
    category: 'Secretary-System'!

!Task MethodsFor: 'execution'!

execute
    "execute the task"

    | value |
    [
    value := receiver perform: method withArguments: arguments.
    value = #failure
        ifTrue:
            [failureEvent generate]
  ] fork! !
```

図 14 Smalltalk におけるタスクの実行
Fig. 14 Task execution on Smalltalk.

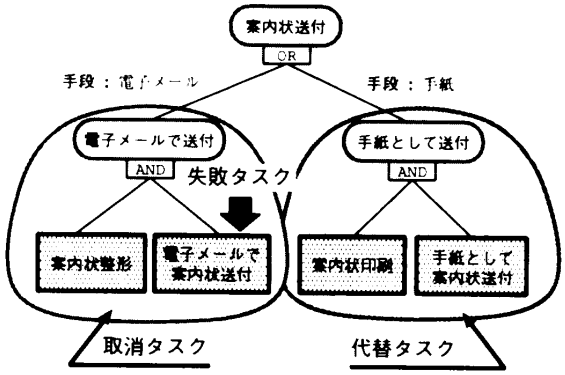

図 15 AND-OR グラフ上でのタスク失敗の例
Fig. 15 Example of task failure on AND-OR graph.

て失敗イベントを生成することができる．オブジェクトが「失敗」を示す値を返した場合，タスクはあらかじめタスクごとに定義されている失敗イベントを生成する．

Smalltalk-80 は，ブロック文を独立したプロセスと見なすことができる[12]．このため，我々のプロトタイプでは，タスクの実行は図 **14** のように行われる．図 14 では，“[...] **fork**”がブロック文“[...]”を独立して評価されることを表している．

(2) 取消タスクと代替タスクの決定

失敗イベントはイベント・ハンドラで優先的に受理される．失敗イベントを受理したイベント・ハンドラはユーザに対してその処理方法を尋ねる．ユーザが選択できる処理方法は以下の 4 つに大別される．

(1) 失敗したタスクの実行のみを取り消す．
(2) AND-OR グラフ上での OR ノードの条件を変更する．
(3) タスクを生成したアクションを変更し，新しい AND-OR グラフを選択する．
(4) 状態遷移図自体を取り消す．

ユーザがこれらの処理を選択することで取消タスクが決定される．加えて，(2) と (3) では代替タスクも用意される．例として，電子メール・システムの故障のためにタスク「電子メールで案内状送付」が失敗した場合を考える（図 **15** を参照）．図 15 では，OR ノード「案内状送付」の条件が変更されると，「案内状整形」と「電子メールで案内状送付」の各タスクが取消タスクとなり，「案内状印刷」と「手紙として案内状送付」の各タスクが代替タスクとして選ばれる．これらのタスクは失敗処理要求とともにタスク・スケジューラへ送られる．

(3) 取消タスクのロール・バックと代替タスクの実行

失敗処理要求を受け取ったタスク・スケジューラは，指定されたタスクを保障された手続きに基づいてロール・バックする．なぜならば，すべてのオブジェクトの状態はタスクを実行するたびに連続的に変更されているからである．ロール・バック法に関しては様々な方法が提案されている[9]が，我々のプロトタイプでは以下に述べる単純なチェック・ポイント法を採用した．

(1) タスク・スケジューラはつねに適当な間隔で全オブジェクトの無矛盾な状態を保存している．
(2) タスク・スケジューラが失敗処理要求を受け取ると，受け取った要求に従って全オブジェクトを適切な状態に戻す．
(3) 更新メソッド，参照メソッドの概念に従ってタスク・スケジューラは取消タスクの影響を取り除き，代替タスクを実行する．

```
procedure ロール・バックと再実行タスクの決定
  input   T_undo : 取消タスクの集合
          Log : チェック・ポイントを含むタスクの実行履歴
  output  T_redo : 再実行しなければならないタスクの集合
begin
  foreach t in Log (もっとも最近に実行された順に取り出す) do
    if t が T_undo に含まれる then
      T_undo ← T_undo − {t}
    else if t がチェック・ポイント and T_undo が空 then
      すべてのオブジェクトをチェック・ポイント t の状態に戻す
      return
    else
      T_redo ← T_redo + {t}
    endif
  done
end
```

図 16 取消タスクの決定アルゴリズム
Fig. 16 Algorithm for determining redone tasks.

図 **16** に適切なチェック・ポイントまでのロール・バックと再実行タスクを決定するアルゴリズムを示す．図 16 では，実行履歴をもとに全オブジェクトは適切な状態にまで戻され，すべての取消タスクと更新されるオブジェクトに関連するすべてのタスクの両方が再実行タスクとして選択される．選択された再実行タス

クと代替タスクはスケジューリングが行われて（すなわち，タイムスタンプが設定されて）実行待ちキューに入れられる．この方法は最適ではないが，失敗処理のためには十分である．

6. 考　　察

本稿で述べた実行制御機構は秘書モデルの構築のための最も基本的な方法を与える．このような実行制御機構では，タスク間の相互作用を識別できるように，与えられた要求や現在の状況などを的確に判断する枠組みと，識別された相互作用に基づいたタスクの効率的な制御法が必要となる．以下，我々の実行制御機構について，一般的な視点から考察する．

我々の実行制御機構では，新たな仕事の要求や返事の到着など，システム外部で生じる非同期的な事象はイベントとしてイベント・ハンドラによって扱われる．一方，タスクの実行主体であるオブジェクトの振舞いの識別や個々のタスクのコミットメント制御はタスク・スケジューラによって行われる．すなわち，外界の変化と内部の状況を2つの構成要素によって別々に管理している．たとえば，1章で述べた階層トランザクション・モデル[9]の場合，我々がイベントとして扱っているシステム外部の事象に対しては比較的適切なアクション（またはトランザクション）を選択することができるが，個々のオブジェクトの振舞いやタスクの失敗など，システムの内部状態に関しては，上位のトランザクションが下位のトランザクションの振舞いを管理する階層構造を持つため，効率的に処理することはできない．一方，協調性の概念に着目したエージェント指向パラダイムに代表される分散モデル[4)~8)]では，個々のエージェントで起こる振舞いや環境の変化にうまく対処することはできるが，システム全体に与えられる要求や外界で起こる事象に対しては，敏速に対処することはできない．我々の実行制御機構は，このような外界と内部の状況変化の両方を効率的に扱うために，外界から与えられる要求を処理するイベント・ハンドラと，内部の実行状況の変化に対応するタスク・スケジューラから構成される．

一方，タスクの実行に関しては，イベント・ハンドラがタスク間の静的な関係に基づいてタスクの順序づけを行い，タスク・スケジューラがタスクの動的な振舞いに基づいてタスクの再スケジューリングを行う．すなわち，タスクがタスク・スケジューラによって実際に実行される前に，イベント・ハンドラによってあらかじめ静的にスケジューリングが行われる．この枠組みは，実行時にすべてのタスクのスケジューリングを行うよりも効率的である．なぜならば，実行時にすべてのスケジューリングを行う場合，システムが複数の仕事を実行しているときに，次に実行すべきタスクの候補が多くなり，結果的に全体のスループットを低下させてしまうからである．さらに，イベント・ハンドラはイベント間の評価順序を表した状態遷移図を用いることによって，将来発生するかもしれないイベントに関連するタスクについて，そのイベントが発生する前にあらかじめ静的にスケジューリングを済ませておくことができる．これらの枠組みにより，我々の実行制御機構はタスクの効率的な実行制御に成功している．加えて，我々の実行制御機構では，タスクの生成はイベント・ハンドラによって基本的に静的な視点で行われるが，失敗処理のように実行時の状況によってタスクを生成しなければならない場合，イベント・ハンドラとタスク・スケジューラの連携によって，動的にタスクを生成する仕組みも提供している．

7. おわりに

オブジェクト指向パラダイムはオブジェクトを動的に制御するには十分であるとはいえない．本稿では，秘書モデルの構築に関して，オブジェクトをタスクの実行主体としてとらえ，互いに関連し合うタスク間の実行制御機構を確立した．特に，静的な視点でタスク間の相互作用を識別するイベント・ハンドラと，動的な振舞いに基づいてタスクを効果的に制御するタスク・スケジューラについて述べた．これら2つの構成要素は，外界の状況変化と，資源の制約などの内的要件の2つを同時に扱うことを可能にする．このようなアプローチは連続し，かつ相互に関連し合うタスクの効率的な制御に十分なものである．加えて，タスクの実行が失敗した場合におけるイベント・ハンドラとタスク・スケジューラの連携についても述べた我々が提案した実行制御機構は秘書モデルの構築のための第一段階ではあるが，イベント・ハンドラによって扱われる状態遷移図やAND-ORグラフは秘書が扱う複雑な仕事を表現でき，さらにタスク・スケジューラによって実行されるタイムスタンプ順序法や失敗処理はオブジェクト全体の一貫性を保つために有効である．

本稿では，タスク間の相互作用として主にタスク間の順序関係に着目したが，秘書はタスク間の順序関係だけでなく，大きなタスクを複数のタスクに分割したり，逆に複数のタスクをより大きなタスクに集約させるなど，タスクを再構成しながら，効率的かつ柔軟に仕事を遂行している．このようなタスクの動的な組織化に関する研究は今後の課題である．加えて，我々は

本論文で述べた実行制御機構のプロトタイプを作成し，その基本的な動作は確認しているが，今後は実システムとしてタスクやオブジェクトなどの構成要素を実装して評価しなければならない．

謝辞 日頃よりご指導いただいている名古屋大学工学部・稲垣康善教授，鳥脇純一郎教授，ならびに中京大学情報科学部・福村晃夫教授，名城大学理工学部・杉江昇教授に深く感謝いたします．また，熱心に討論していただいた研究室の皆様に感謝いたします．さらに，本論文をより良くするために貴重なご意見，ご指摘をいただいた査読者の方々に深く感謝します．

参考文献

1) Meyer, B.: *Object-Oriented Construction*, Prentice Hall (1988).
2) Rambaugh, J.: *Object-Oriented Modeling and Design*, Prentice Hall (1991).
3) Agha, G. and Hewitt, C.: Actors: A Conceptual Foundation for Concurrent Object-Oriented Programming, *Research Directions in Object-Oriented Programming*, Shriver, B. and Wegner, P. (Eds.), pp.49–74, MIT Press (1987).
4) 石田 亨，桑原和宏：分散人工知能 (1)：協調問題解決，人工知能学会誌，Vol.7, No.6, pp.45–54 (1992).
5) 桑原和宏，石田 亨：分散人工知能 (2)：交渉と均衡化，人工知能学会誌，Vol.8, No.1, pp.17–25 (1993).
6) Ishida, T., Gasser, L. and Yokoo M.: Organization Self-Design of Distributed Production System, *IEEE Trans. Knowledge and Data Engineering*, Vol.4, No.2, pp.123–133 (1992).
7) 中内 靖，三由英輔，岡田豊史，安西祐一郎：エージェントモデルに基づく協調作業支援環境について，電子情報通信学会論文誌，Vol.J75-D-II, No.11, pp.1874–1883 (1992).
8) Lee, K.-C., Mansfield, W.H. Jr. and Sheth, A.P.: A Framework for Controlling Cooperative Agents, *IEEE Computer*, Vol.26, No.7, pp.8–16 (1993).
9) Elmagarmid, A.K. (Ed.): *Database Transaction Model for Advanced Applications*, Morgan Kaufmann (1992).
10) Kung, D.C.: The Behavior Network Model for Conceptual Information Modeling, *Information Systems*, Vol.18, No.1, pp.1–21 (1993).
11) Peng, D.T. and Shin, K.G.: Optimal Scheduling of Cooperative Tasks in a Distributed System Using an Enumerative Method, *IEEE Trans. Softw. Eng.*, Vol.19, No.3, pp.253–268 (1993).
12) Goldberg, A. and Robson, D.: *Smalltalk-80: The Language and Its Implementation*, Addison-Wesley (1983).
13) Watanabe, H., Watanabe, T. and Sugie, N.: A Flexible Control Mechanism for Managing Interrelated/Interdependent Tasks Successively, *Proc. COMPSAC '94*, pp.78–83 (1994).

(平成7年7月10日受付)
(平成8年3月12日採録)

渡辺 裕之（正会員）

1970年生．1993年名古屋大学工学部情報工学科卒業．1996年同大学院工学研究科情報工学専攻博士前期課程修了．同年，富士通エフ・アイ・ピー（株）入社．在学中，オブジェクト指向パラダイムに基づくスケジューリングの研究に従事．情報処理学会会員．

渡邊 豊英（正会員）

1948年生．1972年京都大学理学部修了．1974年同大学院工学研究科数理工学専攻修士課程修了．1975年同博士課程中途退学．同年京都大学大型計算機センター助手．1987年名古屋大学工学部情報工学教室助教授．京都大学工学博士．統合化環境，分散協調環境，データベース環境，データベースの高度インタフェース，知的CAI，文書理解，地図理解に興味を持つ．電子情報通信学会，日本ソフトウェア科学会，人工知能学会，ACM，IEEE Computer Soceity，AAAI各会員．