共生/寄生エージェント・モデルによる発展的P2Pアプリケーション構築法

飯島 正[†] 本位田 真一[††,†††] 土居 範久[†]

ソフトウェアエージェントと呼ばれる自律的な分散構成要素の集まりとしてネットワークソフトウェアを記述する方法が注目されている．その方法は，本質的にそうした構造を持つP2P（Peer-to-Peer）アプリケーションに適している．本論文では，環境の変化に追従して機能を動的に変化させていくP2Pアプリケーション（発展的P2Pアプリケーション）の構築を，構造の動的再構成機能を備えたエージェントモデルを使って行う方法に関して論じる．そうしたアプリケーションの調査は，動的ワークフロー管理システムを通して行った．さらに，それをふまえて，逆に動的ワークフロー管理の発想を取り入れた，エージェント間の連携を記述し実現していくための階層的な構築技法に関して提案する．

A Construction Methodology of Evolutional P2P Applications by Using Symbiotic/Parasitic Agent Model

Tadashi Iijima,[†] Shinichi Honiden[††,†††] and Norihisa Doi[†]

Recently a method to describe network software as a group of autonomous distributed agents has attracted a great deal of attention. The method seems to be conformable to P2P (Peer-to-Peer)-style applications with a like structure inherently. In this paper there is a discussion about an approach constructing evolutional P2P applications which can be adapted to the change of surrounding environment by using a dynamic restructurable hierarchical agent model, called Symbiotic/Parasitic Agent Model. We have investigate such an application through the dynamic workflow management system. In addition this paper describes a multi-layered construction methodology to analysis and design incorporated behavior of agents with the idea of dynamic workflow manegament, conversely.

1. はじめに

近年，システム構成要素の自律分散化を目指したソフトウェアエージェント技術が注目されている．ネットワーク・エージェントは，ネットワーク移動機能，ACL（Agent Communication Language）と呼ばれる言語による通信，プランニング，交渉といった機能の一部もしくは全部を備えることにより，互いに柔軟な結合をし合うことのできるコンポーネントである．

ネットワークソフトウェアは，一般に異なる管理ドメインが（空間的に）複雑に重なり合うという性質を持っているため，集中的な管理メカニズムを作ることは難しい．そこで，そうした集中管理に代わり，ソフトウェアエージェントに基づいた自律的な構成要素の水平分散化による柔軟性の導入に期待が集まっている．

自律的な構成要素の協調的な結合によって，システム設計者は，システム全体につねに目を配ることなく，一種のビュー（view）としてシステムの一面だけに注目し，その一面をとらえたエージェントの個別な開発に集中することができる．

しかし，一方で，そうしたシステム開発における局所性は，全体的な振舞いの整合性や効率の保証を難しくしてしまうことになりがちである．したがって，水平分散化と一極集中管理とを両立させるハイブリッドな構成を必要に応じて導入することが望ましい．その実現のためには，動的に階層的な構造を構築するメカニズム，どのようなビューに注目すればよいのかという開発技法，ならびに，個別のビューとして開発されたエージェントをどのように組み合わせて全体的に協調的に連携させるかという協調性の戦略が必要となる．

† 慶應義塾大学
Keio University
†† 国立情報学研究所
National Informatics Laboratory
††† 東京大学
The University of Tokyo

また，オープンなネットワーク環境においては，もう 1 つ重要な課題がある．オープンな環境において，構成要素やそれを取り巻く状況の変化は必然であり，システム設計者が初期段階から将来を見越した設計をすること，ならびに全体を把握して環境の変化につねに適合するように手を入れ続けることはきわめて困難である．それには，自律的に動的にシステム構成を組み替えることによって機能を変化させ適応する能力が解決方法の 1 つとなりうる．そのように機能を変化させていく能力をソフトウェアの発展性と呼び，そうした能力を備えたソフトウェアを発展的ソフトウェアと呼ぶこととする．

本論文では，局所性と集中性の両立，協調性ならびに発展性を実現するための手段として，共生/寄生エージェントモデルを採用する．このモデルは，エージェントの集まりを動的に組み合わせたマルチエージェントシステムとして，より大きな粒度のエージェントの階層構造を動的に構成していく寄生機能と，資源を共有し合うエージェントどうしを同調させて動作させる共生的振舞いを有している．これによって，必要に応じて動的に機能を追加/獲得するような発展性をも実現することを試みている．

本論文では，例題事例として動的ワークフロー管理を取り上げて，まず，ネットワーク環境におけるソフトウェア構築の課題と，解決の方針を検討する．動的ワークフロー管理とは，部分的なワークフローを動的に結合したり，必要に応じて動的にワークフローを生成したりする機能を備えたワークフロー管理モデルである．その例題の検討によって見出された，

- 複雑な依存関係の下での的確な連携，
- 分散資源の共有と有効活用，
- 動的な変化への適応，

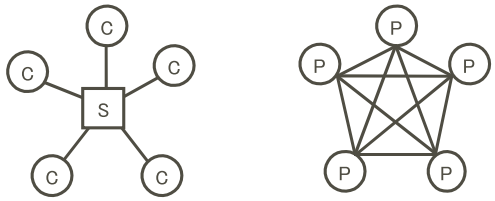
といった課題の解決を目指す．

現在のネットワークソフトウェアは，データ資源や計算資源の一元化と共有化の目的からクライアント/サーバ（C/S）コンピューティングが主流であり，多くのアプリケーションの基本形態となっている．この形態では，(1) サーバはクライアントのリクエストを受け付けて適切なサービスを提供する責任を持ち，サーバに資源の集中化がなされている．また，(2) そのインタラクションは，リクエストに対してリプライを返すという単純な形態を基本とする．実際，一般的なワークフロー管理システムにおけるアーキテクチャ（物理レベル）もワークフローエンジンによる集中管理方式のモデルを基本としているといえる．その理由の 1 つは，従来のワークフロー管理システムの対象が，1 企業内，1 部署内という比較的小さな領域を想定していることが多いためである．しかし，今後は，企業間商取引におけるワークフローの相互接続といった広域化の必要性が期待されている．また，そこには，グローバルなパブリックワークフローとローカルなプライベートワークフローといった，セキュリティやプライバシのポリシドメインの違いも現れてくる．こうした点が，将来のネットワークソフトウェアの物理レベルのアーキテクチャを考えるうえでの課題を洗い出すための重要なキーポイントと考えて，論理的なレベルにおいて自律的な分散要素の連携を目的とするワークフロー管理システムを例題として取り上げる．

本論文では，さらに，こうしたワークフロー管理による連携制御を，最近注目されている P2P コンピューテーションの考え方を，より進展させるものとして提案する．物理的アーキテクチャとしての P2P コンピューテーション上で構築されるアプリケーションでは，ピア（自律的な分散要素）の連携動作の実現が期待されている．しかし，現時点で，P2P コンピュテーションに関して論じられている連携のレベルは，必要なサービスを提供するコンポーネントを検索するディレクトリサービスなどが中心的なテーマとなっている．すなわち，ワークフローとして表現されるような連携制御のシナリオを扱うレベルではない☆．それが，P2P コンピューテーションというアーキテクチャの用途があまり広がっていない理由と考えて，そこにワークフロー管理メカニズムを導入し，P2P アーキテクチャによる一般的なアプリケーション（P2P アプリケーション）のための構築基盤とすることを提案する．

本論文の構成は以下のとおりである．次章では，動的ワークフロー管理を例題として，P2P アプリケーションの特性と課題に関して整理する．3 章では，そのアプローチの基礎となる共生/寄生モデルについて紹介し，4 章においてワークフロー定義のエージェントによるモデル化を行う．5 章では，ワークフロー管理の考え方を導入し，協調の戦略を構築し実現するための階層的技法を与える．

☆ JXTA プロジェクトにおいては，とりあえず通信，情報共有グループの形成，ディスカバリサービスのためのプログラム群を提供している[12]．一方，Web サービス連携という文脈では，WSFL[13] や DAML-S[14] などプロセス記述を扱う表現も提案されている．



(a) C/Sアーキテクチャ (b) P2Pアーキテクチャ

図 1 C/S アーキテクチャと P2P アーキテクチャ
Fig. 1 C/S architecture and P2P architecture.

2. P2P アプリケーションとその構築技法への課題

2.1 P2P アプリケーションとは

P2P（Peer-to-Peer）技術という用語は，元々は，端点間通信（Point-to-Point コミュニケーション）を指す語であったが，最近では，ネットワークソフトウェアの物理レベルのアーキテクチャとして，集中管理型のクライアント/サーバ（C/S）コンピューティングと対比される水平分散型モデル（**図 1**）を指し，P2P コンピューテーションという用法が多くなっている[2)~5)]．もっとも，そのアーキテクチャの用途としては，Napster[7)]，Gnutella[8)] などの個人間ファイル共有/交換システムといった特定のアプリケーション（特に，できるだけ集中管理するサーバを設けないことで，匿名性を高めたもの）が主流と見なされている☆．

これに対して，本論文では，この P2P アーキテクチャをベースにより一般的なアプリケーションに適用することを試みる（そうしたアプリケーションを P2P アプリケーションと呼ぶこととする）．そのための方策として，動的なワークフロー管理のメカニズムを取り入れることでピア（自律分散要素）間の連携制御を強化することを提案する．そこでのワークフロー管理のメカニズムは，完全な集中管理ではなく，必要に応じて階層構造を導入するような動的なハイブリッド構成と，移動エージェントといった非集中的な要素をマネージャとするワークフロー管理の発想を取り入れたものとしている．

2.2 P2P アプリケーション構築上の課題

この節では P2P アプリケーション構築上の課題とそれに対処するアプローチを検討する．検討用例題は，オープンで広域的なネットワーク環境における，(たとえば企業間電子商取引などの）ワークフロー管理である．LAN の中に閉じたワークフロー技術では，物理的レベルにおいては，集中的なサーバによって管理するモデルが多いが，ここではワークフローによって連携される各ピア（ソフトウェアエージェントもしくは人間）は，物理レベルにおいて，非集中的なアーキテクチャ，すなわち，ここでいう P2P アーキテクチャで構成されているものとする．

ソフトウェアアーキテクチャの適格性を論じる場合，(a) 実行制御（対象レベル）と，(b) 管理/運用/保守（メタレベル）といった 2 つのレベルで考える必要がある．さらに分散アプリケーション構築という観点からみると，「分散性（ばらつき）」が，それらの各レベル（特に (b)）にどのように影響するかを検討しなければならない．分散性は，分散アプリケーションにとって本質的であるが，その開発に用いるアーキテクチャによって，アプリケーション構築上で受ける影響が異なる．そこで，現在主流の C/S（集中的）アーキテクチャとの対比をもって，P2P（非集中）アーキテクチャを使ったアプリケーション構築をより容易にするために解決すべき課題を検討する．

分散システムは，ネットワーク上に分散した構成要素から成り立つシステムである．すなわち，その構成要素が，空間的に離れた位置にばらついて存在するという「空間的な分散性」は本質的な属性である．それゆえ，「空間的な分散性」の実行制御に対する影響は，従来よりいろいろな分散アーキテクチャの提案にあたって必ず検討されてきた．しかし，アプリケーション構築（特に管理保守）ということを取り上げる場合には，「空間的な分散性」だけでなく，もう 1 つ「時間的な分散性」の影響を考慮しなければならない．

「時間的な分散性」とは，システムの各構成要素の構築（改訂）のタイミングが，それぞればらついていることを意味する．時間の経過に従って，新しい構成要素が追加されたり，それぞれの構成要素が独立に改版されたりすることにより，全体としての整合性が崩れてしまうことがある．そこで，この 2 つの分散性に起因する P2P アプリケーション構築の困難さも，【課題 1：空間的困難さ】と【課題 2：時間的困難さ】とに分割して取り扱うこととする．

【課題 1：空間的困難さ】を，さらに 2 つに分ける．分散システムをノードとアークからなるネットワーク（グラフ）構造で表現した場合に，ノード（計算資源）に関係する【課題 1-1】と，アーク（資源間の依存関係）に関係する【課題 1-2】である．

【課題 1-1】は「計算資源の多様性への効率的な対応」

☆ 現実には，完全な水平分散と集中管理との中間的なハイブリッド方式[4),5)] も多い．また，水平的な P2P の基本発想自体も，RM-ODP[6)] などにおけるトレーダの連邦化（federated trader）などに見られるように，技術的にまったく新しいものというわけではない．

である．ネットワーク上には複数の多様な計算資源が存在し，P2P アーキテクチャのピアを構成する．複数のドメイン（管理組織）にまたがって広がるシステム全体を見通すことは難しく，ドメインごとに独立に管理保守されることも多い．また，管理ポリシもドメインごとに決定され，多様性も大きい．たとえば，企業間（B2B）電子商取引におけるワークフロー管理は複数の企業（ドメイン）にまたがって構築される．この場合，集中的なワークフローエンジン（集中サーバ）で，全体のワークフローを管理する方式の C/S アーキテクチャは，セキュリティ，障害復旧，パフォーマンスなどの面で望ましくない．また，それぞれのドメインにおいては，ワークフロー要素のタスク遂行は，固有のポリシのもとで実施されることから，そうした各企業すべてに対応するワークフローを記述することは，その組合せのうえからも困難である．そこで，それぞれの管理主体ごとの固有性（多様性）は，集中サーバに集めるのではなく，個々の管理主体に局在させておき，動的に組み合せる（局所化）ことのできる一方で，ドメイン単位の集中的な管理とも両立させたハイブリッド，すなわち階層化アプローチが解決策として考えられる．その実現例として，4.2.1 項では，3 章で導入する共生/寄生エージェントモデルの寄生機能をもとに 5.2 節図 13 (4) の待ち伏せパターンを使って，部品機能を動的に結合する例を記述している．これによって，エージェントは，移動先で結合するロールごとに，異なる振舞いを動的に獲得する局所化が実現できる．

また，P2P アプリケーションでは，分散した資源を同時並行的に利用することと引き換えに，分散している共有資源管理の負担が増すこともある．C/S モデルのように資源をサーバに集中させるアーキテクチャでは，資源管理も集中的に行うことができる．しかし，自律的構成要素に資源を分散させる P2P アーキテクチャでは，多数の構成要素間で資源管理が必要となり，そのための情報のやりとりやプランニング，スケジューリングが必要となることがある．そこで問題になるのは,【課題 1-2：分散要素間の依存関係や制約の取扱い】である．

C/S アーキテクチャや，そのバリエーションである三層（3-Tier）アーキテクチャでは，データや（ビジネス）ロジックがアプリケーションサーバに集中して配置されるのに対して，非集中的な P2P アーキテクチャでは，それらが分散して記述されることになる．同じアプリケーションであっても，個々のドメインの環境に応じてサブシステムの構成が異なるものとなることがある．したがって，構成要素の配置（deployment）は動的に再構成される．そうした状況で，依存関係や制約の保守を手続き的に作り込むことは困難であって，実行時にならないと依存関係が決まらないことさえありうる．そこで，構成要素の組合せを決めた後で，必要な依存関係のルール定義ならびにその保守手続きを動的にアタッチできるような一貫性監視保守機能を実現できるアプローチが (b) 管理/運用/保守のレベル（メタレベル）で有効である．一貫性監視保守手続きは，たとえばデータの依存関係や一貫性制約について監視をし続けて，データ値の変更に際して発生したイベントに対して，依存関係ルールに基づいて，あるいは一貫性制約を満たすように，データを更新するという手続きである．3.4.3 項では，イベント通知機能に基づく一貫性保守手続きを，例としてあげており，これも，共生/寄生エージェントモデルの寄生機能を使って対象に動的に結合させる．

【課題 2：時間的困難さ】は，時間の経過によって出現してくる問題点であり，メタレベルにおいて「動的な変化への適応」という課題につながるものである．設計当初には矛盾なく設計されていたシステムにおいても，時間が経つにつれて，次の 3 つのズレが次第に大きくなってくる．これは，システム実装（設計）と要求，システムと外部環境，システムの中といったシステムに関わる境界面において起こりうるものである．

(2-1) 運用中のシステム実装（設計）と機能要求との間のズレ
(2-2) システム内の構成要素間のズレ
(2-3) システム内とシステム外との間でのズレ

こうしたズレを修復することが【課題 2】である．一般にこうしたズレは P2P アーキテクチャでなくても発生しうるものであって，固有の問題ではない．しかし，複数の開発者が必ずしも十分な打合せをしないままコンカレントに開発/保守を行う分散開発を前提とし，なおかつ，全体を見通すことの難しい複数組織にまたがった運用を前提とした P2P アーキテクチャでは，この問題は深刻となる．

【課題 2-1】は,「システムに要求される機能に生じるズレへの対応」である．たとえば，ワークフロー管理において，当初，設計されていなかったようなトラブルが発生することがある．特に，モノリシックなシステムとして構築されたシステムではなく，動的に構成要素を組み合わせるような（分散開発される）P2P システムでは，いろいろな組合せについて事前に十分に分析することは難しい．したがって，そうした場合，あらかじめ起こりうるトラブルを分析しておくアプ

ローチの代わりに，動的に対処していくアプローチを採用することを提案する．このような考え方はワークフロー管理の文脈では，動的ワークフロー管理と呼ばれており，トラブルを例外事象として検出し，それに対処するためにワークフローを動的生成するといった実現方法が提案されている（たとえば文献 17)，18))．本論文でも同様のアプローチを採用する．さらに，その実現方法の一具体例としては，4 章で，トラブルを例外イベントとして検出し，トラブル処理手順をワークフローを表現したエージェントとして生成し，それを寄生機能を使って取り込むことで，振舞いの一部を変更する「部分機能の交換/追加」という方法を採用する．4.2 節 (2) では，その記述例として「組織改変による担当者変更への対応」を与えている．

【課題 2-2】は,「システムの構成要素間で生じるズレへの対応」である．こうしたズレは，システムを構成する各構成要素が，必ずしも同じ設計者ないしプログラマによって同一時期に作られたものばかりではなく，異なるタイミングで別々の開発者によって構築され，改版されることに起因する．また,【課題 1-1】への局所化アプローチや,【課題 2-1】への具体的な対処方法として採用する「部分機能の動的交換/獲得」は，当初設計の考慮外であった構成要素の結合をもたらし，構成要素間のインタフェース（I/F）に不適合を発生させることがある．特に従来の API 関数呼び出しやメソッド呼び出しは，I/F を強く型付けする傾向にある．これはソフトウェア工学において，バグ検出に有効な手段ではあるが，柔軟性に欠けることは否めない．そこで，まず第 1 に，構成要素としてエージェントを採用し，構成要素間の I/F にエージェント通信言語（ACL）を利用するアプローチを採用する．ただし，この目的での ACL の利用は，API 関数呼び出しやメソッド呼び出しにおいて，パラメータの並び順や，暗黙値の扱いを柔軟にするといった程度の範囲にとどまっている．呼ぶ側と呼ばれる側の相互のズレを仲介エージェントが解消するアプローチもありうる．これに関しては，共生/寄生エージェントモデルの利用を前提とした場合について，動的結合戦術のパターン（たとえばフィルタリングパターンやクッションパターン）として，5 章で整理している．

【課題 2-3】は,「システム内とシステム外との間でのズレの解消」である．より具体的には，システム外の組織構造と，システム内部の構造の関係におけるズレを考える．情報システムは，その内部にしばしばシステム外の構造を反映している．たとえば，データベースのスキーマは，対象とする組織の構造を反映して設計されている．しかし現実世界の組織構造は，時間経過に沿って，しばしば変化していく．たとえば企業内情報システムにおいて，組織改変，人事異動への対応は不可欠である．したがって，それを反映するように，システムの側も変更させていかなければならない．たとえば現実世界の組織構造の変化を監視し，その変化に応じて，システムの側も変更させていくような対応関係の保守機構を，このズレの解消のためのアプローチとして検討している．これにも「一貫性保守機構」と「部分機能の動的交換」が有効と考えられるが，システム外の組織構造を検出する方法が必要であり，それに関しては本論文では扱わない．しかし，4.2.3 項の記述例「組織改変による担当者変更への対応」は,「担当者不在という例外イベントを発生させることができる」という前提の下で，システム外の組織構造の一時的な変化（不在）に対応した振舞いの実現ともいえる．

表 1 P2P ネットワークアプリケーション構築上の課題
Table 1 Problems for building P2P networking applications.

課題	解決への主なアプローチ
(1) 空間的困難さの克服	
(1-1) 計算資源の多様性	階層化と局所性の導入
(1-2) 依存関係の取扱い	一貫性保守機構の導入
(2) 時間的困難さの克服	
(2-1) システムに要求される機能のズレ	部分機能の動的交換
(2-2) 構成要素間の I/F のズレ	I/F の柔軟な結合
(2-3) 現実の組織構造の変化とのズレ	実世界との対応関係の保守

以上の課題と対策を整理すると**表 1** のようになる．

2.3 解決策の実現に向けての要件の整理

この節では 2.2 節であげた課題の解決アプローチで必要とされる，システムならびにその構成要素に必要な機能ならびに性質（要件）を整理する．

【課題 1-1：計算資源の多様性への対応】に対しては，

(a) 階層化と局所性を導入できるモジュール構造

が必要である．3 章で導入する共生/寄生エージェントモデルにおけるエージェントは，動的に階層的な構成を再構築する機能（寄生機能；3.2 節）を持ち，この要件を満たしている．

次に，そのモジュール単位（エージェント）が備えるべき性質と機能を検討する.【課題 1-1】への対処法である「部分機能の動的交換」と【課題 2-3：実世界との対応性の保守】に関しては，

(b) 動的結合性

を備えていることが必要であり,【課題 1-2】に対応す

るために一貫性の監視保守手続きを動的に付加するためにも役立つ．

【課題 2-2】のためには，

(c) モジュール間 I/F の柔軟性

が必要であり．共生/寄生エージェントモデルでは，エージェント通信言語（ACL）の採用（3.4.4 項）により，この性質を取り入れている．また，I/F のズレを補償するような仲介エージェントを 5.2 節において整理した動的結合パターンで挿入するためには，(b) の動的結合性も必要となる．

【課題 1-2】,【課題 2-1】,【課題 2-3】の解決に必要な性質には，次の (d) と (e) がある．

(d) リアクティブ性

システム内部ならびにシステム外部の状態を検出して，それに反応するリアクティブ性は，一貫性監視保守機能実現するエージェントにとって不可欠な機能であり，イベント通知機能（3.4.3 項）は，このためにある．

(e) 協調性

ここでいう協調性は単に，あらかじめ作りこまれた協調プロトコルに従って協調的な振舞いをするということを意味するものではない．ここでは，共通の目標に対して連動して動作すること，ならびに，つねに自らの保有する資源を監視し，その資源の許す限り，他者の目標達成を妨害しない（できることなら，他者の目標達成に協力する）という共生的な資源管理を意味する．したがって，エージェントの目標達成へのプランニングは，他のエージェントの目標達成に必要な資源を意識して行われる．共生/寄生エージェントモデルの記述言語の 1 つ DOPA-L（3.4.1 項）の持つ task 属性は，この性質を持ったプランナを呼び出すためにある．

以上のような，(a)〜(e) の性質（要件）を備えた構成要素として，3 章では共生/寄生モデルに基づくエージェントを示し，4 章において動的ワークフロー記述に採用する．さらに，P2P アプリケーションを構築するには，エージェントモデルとそのプラットフォームだけでなく，それらを有効に活用するための方法論（開発技法）が必要である．動的ワークフロー管理を単なる P2P アプリケーションの典型例にとどめず，むしろ，その考え方を一般的な P2P アプリケーション構築の方法論に導入する技法（4 階層の階層的技法）を 5.1 節で提案する．

3. 共生/寄生エージェント モデル

3.1 共生/寄生モデルの概要

共生/寄生モデル（S/PAM; Symbiotic/Parasitic Agent Model）[1] は，動的に階層構造を形成/再形成することによって，開放ネットワーク環境に不可欠な「機能の動的獲得」などのための基本能力を提供する可変エージェント（mutable agent）モデルである．そのモデルでは，共生する（互いに資源を共有する）エージェントのグループによって構成されたマルチエージェントシステムを，再帰的に 1 つのエージェントと見なすことで，階層構造を形成する．そのグループを形成する操作が寄生操作である．エージェントは別のエージェントに動的に寄生することができる（寄生されたエージェントを宿主エージェント，寄生するエージェントを寄生者エージェントと呼ぶ）．この操作はエージェントを生成するときに実行できるだけでなく，任意のタイミングで行うことができるので，この機能を使ってエージェントは，自分自身の内部構造を動的に変化させ，自身の機能を入れ替えることができる．

逆に寄生者エージェントから見ると，宿主エージェントは，その寄生者エージェントの相互通信を含め各種の活動の管理を行うものであり，ポリシの適用範囲や，イベント通知の有効範囲なども表現する．エージェントは，他のエージェントに寄生することで，その宿主エージェント内で発生するイベントのうち関心を持つものの通知を受けることができる．エージェント間の通信は，一般にエージェント通信言語（ACL）を使って行われる．通信の経路は，互いの共通の宿主に至るまで，その宿主に遡って伝えられ，共通の宿主を介して伝えられる．この際，宿主は寄生者からのもしくは寄生者への通信を監視し書き換えることができる．

また，このモデルは，移動エージェント（mobile agent）の概念的な基礎モデルでもある．このモデルでは，移動エージェントの「移動」は，他のエージェントへの寄生の一種として扱われる（**図 2**）．一般に，宿主エージェントが，物理的に他のホストマシン上にあるときの「寄生」は「ネットワーク移動」を意味することになる☆．このとき，宿主はいわゆるプレースに相当する．寄生は，寄生先が同一ホストマシンにある場合も含むので,「ネットワーク移動」の概念を含む，より抽象的な概念と考えることができる．このように，物理的な位置と独立に組織構造を中心にモデルを記述できることは，概念レベルと物理レベルの分離をより容易にすることに貢献し，ワークフロー定義においても有効である．

寄生エージェントのための抽象アーキテクチャを図

☆ 移動不可属性が設定されている場合には物理的なネットワーク移動をせず，遠隔通信を介してあたかも物理的に移動しているかのように振る舞う．

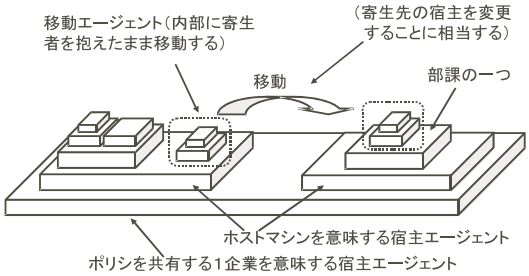


図 2 物理的な位置関係との独立性（移動も寄生の 1 ケースと見なす）
Fig. 2 Independence from physical locations (mobility as parasitism).

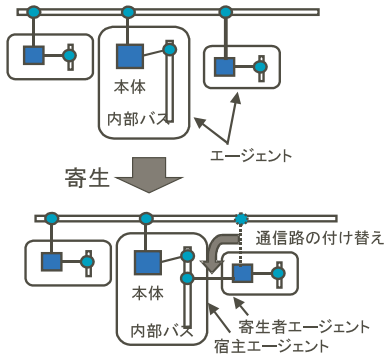


図 3 寄生エージェントのための抽象アーキテクチャ[1)]
Fig. 3 The abstract architecture for parasitic agents.

のように与える（**図 3**）．宿主と寄生者が持つ図 3 のように階層的な入れ子構造は，メッセージの通信路の結合性で表現する．エージェント（これを A とする）の中には，さらに内部バスがあり，A に寄生するエージェント B は，その A が持つ内部バスに接続しているようなイメージである．個々のエージェントは，単一のスレッドもしくは，単一のプロセスといったなんらかのレベルの並行実行単位である．

以降では，2.4 節で列挙した問題解決のアプローチ (a)〜(e) と関連付けて，共生/寄生エージェントモデルの持つ機能を説明する．(a) の「階層性と局所性を導入できるモジュール構造」は，このモデルのエージェントを意味し，寄生によって動的に階層構造を導入することができる．さらに，(b) の「動的結合性」も，ここでの寄生操作に相当する．(c) の「I/F の柔軟性」は，エージェント通信言語（ACL）によるコミュニケーションで実現されている．エージェント通信言語に基づく通信は，メッセージの構文的かつ意味的な解析をともなうので，メソッド呼び出しよりも柔軟な I/F を表現することができる．(d) の「リアクティブ性」は，イベント通知機能をベースとして与えられる

表 2 寄生に関わる操作
Table 2 Operations for parasitism.

操作名	内容
1) 寄生	エージェントが他のエージェントの内部に寄生する
2) 獲得	エージェントが自分の内部に他のエージェントを寄生させる
3) 排出	宿主エージェントが寄生者であるエージェントを排出する
4) 脱出	寄生者であるエージェントが宿主エージェントから脱出する
5) 統合	エージェントを結合して，1 つの「個体」としてのアイデンティティを持ったエージェントを構成する．寄生とは異なり排出や脱出は禁止される
6) 分割	エージェントが自分の能力・権限の一部を新しいエージェントとして分割・委譲する

（3.5 節のイベントハンドラの記述を参照）．(e) の「協調性」はイベントハンドラから呼び出されるプランナによって実現されている．

(a) と (b) に関しては，3.2 節で説明する．(c)，(d)，(e) は，3.4 節において，エージェント記述言語の一例を通して説明する．ワークフロー記述において，これらの機能がどのように使われるかは，次の 4 章において記述する．

3.2 エージェントの寄生に関わる操作

共生/寄生モデルでは，エージェントの寄生に関して，**表 2** のような操作を定義している[1)]．3.4 節で述べるエージェント記述言語 DOPA-L では，1) と 4) を go 要求，2) を get 要求，3) を put 要求で取り扱い，5) と 6) は扱わない．その他の基本操作としては，エージェント生成，エージェント削除，エージェント複写などのライフサイクル管理がある．

抽象アーキテクチャにおいて，寄生操作は，抽象的には通信路（パイプ）の付替えに相当する（図 3）．これらの操作は，ACL においては共生/寄生オントロジとして与えられ，関与するエージェント間での request 通信行為によって依頼される．

図 4 に示すように，入れ子の内側の（寄生者）エージェントと，その外部のエージェントの通信は，通信経路上にある宿主エージェントの管理下にあり，通信発生時には，宿主によって，横取りや，書き換えといった操作を受ける（メッセージフィルタリング）．これによって宿主と寄生者の間で一種の支配関係が形成される．図 4 に示されているドットで区切られたパス式（path expression）としての識別子表現などの詳細は，文献 1) を参照されたい．通常は，パス式のドットで区切られた最後の要素（固有識別子と呼ぶ）で，位置

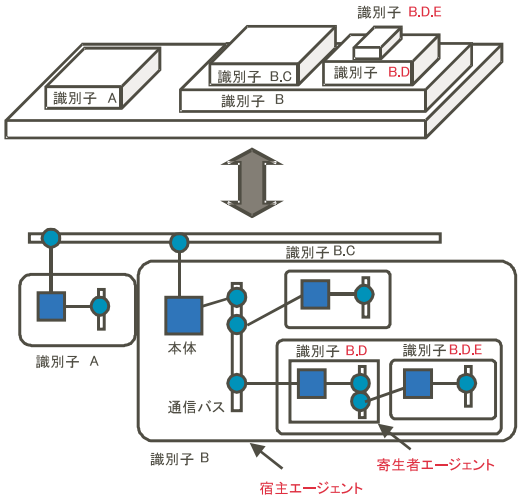


図 4 抽象アーキテクチャにおける寄生[1)]
Fig. 4 Parasitism in the abstract architecture.

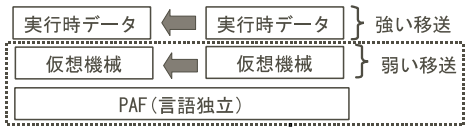


図 5 強い移送の表現の階層
Fig. 5 A representation hierarchy of strong mobility.

に依存しない形でエージェントを特定する．

3.3 ワークフロー定義のための強い移送

ワークフロー定義を移動エージェントを用いて行う場合，活動系列の継続性を表現する「強い移送（strong migration）」が不可欠である．ここでは，共生/寄生モデル（S/PAM）を実現するための概念アーキテクチャ（とライブラリ）である寄生エージェントフレームワーク（PAF; Parasitic Agent Framework）によって,「弱い移送」を実現し，その上でエージェント記述言語の仮想機械を動かすことで,「強い移送」を表現する（**図 5**）．PAF は，IIOP，HTTP，SMTP といった特定の通信プロトコルと独立で，かつ，同モデルを採用する各種エージェント記述言語の共通機能を提供することを目指したレイヤである．この強い移送を 2 段階で実現する方法は，一般に直接実現する方法に比べて実行性能の面で劣るが，仮想機械のコードをあらかじめインストールしておいて，その移送を抑制することで改善できる．

共生/寄生エージェントモデルの持つ機能は，次節でエージェント記述言語 DOPA-L の例を通して説明する．

3.4 XML ベースのエージェント記述言語

現在，PAF の上にエージェント記述言語の 1 つとして，XML ベースの言語 DOPA-L（DOM Object-based Parasitic Agent Language）の定義を試みている．DOPA-L はワークフロー・シナリオの記述専用の言語ではないが，一般的な手続き的プログラミング言語が持つ制御構造と共生/寄生モデルの各種機能を持つので，これを例としてワークフロー記述に有用な機能を紹介する．

3.4.1 記述言語 DOPA-L におけるエージェント記述の構造

DOPA-L の構文は DTD で与えられ，XML 文書としてプログラム（振舞い記述）を定義する（**図 6**）．さらにこれをスタックマシンである仮想機械のコードにコンパイルして実行する．コンパイルしたコードも XML 文書である．さらに，エージェントの実行中の内部状態（プログラムカウンタ，実行時スタック，変数表の内容，ならびにエージェントの階層的な寄生関係）も XML 文書に変換して書き出すことができる（**図 7**）．移動エージェントとして使うときには，これら（コンパイルされたコードと実行状態）の XML 文書を転送することで移動先で実行を再開することができる．つまり，強い移送が実現されることとなる．

DOPA-L の記述例の一部を図 6 に示す．DOPA-L では制御構造や代入文もタグとして定義されており，ワークフロー記述の活動系列の記述に使うことができる．DOPA-L のプログラムは，エージェントクラスとして XML 文書として書かれる．図 6 は，x.xml というファイルの内容であり，エージェントクラス名 x とファイル名が対応する．

エージェントクラスの構成要素は下記のとおりであり，各要素は XML タグで囲まれる．

- ロール宣言
  （スクリプト中で使うロール名の宣言であり，
  寄生するエージェントに割り当てられる）
- 能動モードのスクリプト記述
  - 生成直後から実行されるメイン・スクリプト
    （`<main>`タグで囲まれる）
- 受動モードのスクリプト記述
  - ACL メッセージに反応するメッセージハンドラ・スクリプト群
    （`<message-handler>`タグで囲まれる）
  - 各種イベントに反応するイベントハンドラ群
    （`<event-handler>`タグで囲まれる）
  - そのエージェントの寄生者ロールへのメッセージフィルタ
    （`<message-filter>`タグで囲まれる）

また，各スクリプトは，task 属性として，そのスク

```
<class id = "x">
  <role>
    <agent role="r1" >
      <agent role="r4" />
    </agent>
    <agent role="r2" />
  </role>
  <main require="r1">
    <assign id="A"> 1</assign>
    <assign id="B"> 2 + 3 </assign>
    <if>
      <cond> B == 5 </cond>
      <then>
        <send-message>
          <to> r1 </to>
          <act> act </act>
        </send-message>
        <wait-message from="r1"
                      act="reply" />
        <println> 200 </println>
      </then>
      <else>
        <print> "a" </print>
        <while>
          <cond> A &lt;= 4  </cond>
          <do>
            <println> A </println>
            <assign id="A"> A + 1 </assign>
          </do>
        </while>
      </else>
    </if>
  </main>

  <message-handler
      act="query"
      content="valueOf(X)"
      from="F"
      task="reply(valueOf(X))"
      require="none">
    <send-message> <to> F </to>
      <act> reply </act>
      <content> X </content>
    </send-message>
  </message-handler>
        ...
  <event-handler event="e1">
        (略)
  </event-handler>
        ...
  <message-filter to="r1.r4" act="...">
    <send-message> <to> r1.r4 </to>
        (略)
    </send-message>
  </message-filter>
</class>
```

図 6 DOPA-L の記述例（エージェントクラス x.xml）の一部
Fig. 6 A piece of description (x.xml, a class of agent) in DOPA-L.

```
<agent id="a" class="x">
  <parasite>
    <agent id="a1" role="r1" >
      <agent id="a2" role="r4" />
    </agent>
    <agent id="a2" role="r2" />
  </parasite>
  <meta-var id="pc" val="20" />
  <stack>
     (略)
  </stack>
  <env>
    <var id="A" type="int" val="1" />
    <var id="B" type="int" val="2" />
  </env>
</agent>
```

図 7 実行時のエージェントの直列化表現（a.xml）の一部
Fig. 7 A piece of serialized representation (a.xml) of a live agent.

リプトのゴールの述語表現であるタスク式を持つことができる．資源の不足などの理由によりそのスクリプトが実行できない場合には，このタスク式をゴールとしてプランナを起動し代替スクリプトを生成する．必要な資源（すなわち，ロール名）は，require 属性に列挙される．プランナは，アクションとその事前条件/事後条件の集合を知識として持ち，必要とする/消費する資源もロールとして事前条件/事後条件に含めて与える．

3.4.2 能動モードにおける基本的な振舞いの記述

メインスクリプトはエージェントが生成後に実行する能動的な振舞いを記述するもので，ワークフローの記述にも使われる．

スクリプト中のロールは，そのエージェントへの寄生者の識別子によって置き換えられる．逆に寄生者からは host という名前で宿主を参照できる．これによって，宿主の定義と寄生者の定義とが動的結合される．資源不足は必要なエージェントが寄生できないことを意味する．たとえば，ある作業の担当者エージェント（資源に相当する）が，実行時（act メッセージ送信時）に不在であれば必要な寄生が行われず，例外イベントが発生する（イベント通知機構については次項参照）．その際に動的にワークフローを生成する例を 4.2.2 項でイベントハンドラとして記述する．

エージェントクラス定義をもとに生成されたエージェントは，能動モードとなりメインスクリプトを逐次実行する．メインスクリプトが終了すると受動モードになり，明示的に削除されるまではその時点の宿主にとどまる．そして，ACL メッセージの受信もしくはイベントの発生を待ち，それに反応して動作する．ACL メッセージは，一般にエージェントの識別子を

指定して送られるが，宿主エージェントのスクリプトからはロール名を介して配送される．

3.4.3 リアクティブ性をもたらすイベント通知機能

イベントは，イベント種別を指定した raise 節で発生し，そのイベントを発生させたエージェント自身と，その寄生者がそれを捕捉することができる（つまり，イベントの通知範囲を，寄生関係によって規定している）．捕捉されたイベントは，イベント種別ごとに記述されているイベントハンドラの処理を起動する．典型的なイベントとしてエージェントの到着を意味する arrival イベントや，出発を意味する departure イベントがある．あるエージェントが，その宿主エージェントに寄生したとき，そのエージェントクラス記述中の全イベントハンドラがチェックされ，そのエージェントが注目するイベントの通知を受けられるように，宿主エージェントに登録される．

3.4.4 エージェント通信言語に基づく柔軟な I/F

メッセージハンドラはオブジェクト指向言語のメソッドに相当するが，通信行為とコンテンツのパターンごとに記述される．通信行為は，FIPA [15] の ACL の一部に準拠するが，それに限らない．大文字のリテラルは変数を意味し，メッセージとのパターンマッチングで具体化される．ACL メッセージを受け取ると，それに応じた（マッチするパターンを持った）ハンドラが起動される．図 6 では，valueOf(“A”) というコンテンツをともなう問合せに対して変数 A の値を返すものである．この問合せは，特に資源を必要としない．

メッセージフィルタは，他のエージェントからの寄生者へのメッセージをインタセプトするために使われるスクリプトである．このスクリプトによりメッセージを横取りしたり，書き換えたりすることができる．これによって，宿主は寄生者の振舞いを制御する一種の支配関係を構成する．これを使った，発展的ソフトウェアの構成パターンに関しては，5.1 節で与える．メッセージフィルタとして与えられたパターンにマッチしないメッセージは宿主を素通りして寄生者に届く．

4. 共生/寄生モデルに基づく動的ワークフロー管理

この章では，前章で紹介した共生/寄生エージェントモデルを用いた動的ワークフロー管理のモデル化を試みる．まず，4.1 節で構成要素をモデル化する．これに基づいて次章でのべる階層的構築技法のうちの第 1 層（タスクレイヤ）と第 2 層（ワークフローレイヤ）が与えられる．4.2 節では，実際の動的ワークフロー定義の記述例のいくつかを，特に動的な側面（イベントハンドラとして与えられる）を中心に示す．これによって，ネットワーク上に展開されたワークフローを構成する自律要素をモデル化し，さらに環境の状況変化に適応して，その連携を動的再構成することができる．

ここで示す例は，分散したエージェント間の連携作業の中で，「機能」を提供するエージェントを動的に交換したり追加したりする例となっている．そこでは「機能」エージェントを「資源」として取り扱い，環境の状況変化によって引き起こされた資源の不足や目的に対する不適格性を，「寄生によるロールへの（動的）結合ができない」ことで検出し，イベントハンドラによって対処している．こうした振舞いは，続く 5 章において「振舞いの動的結合のパターン」として整理されている．

4.1 ワークフロー構成要素

本節では，ビジネスプロセスを定義するための構成要素として，プロセスとプロダクトとパーティシパントの 3 要素を考え，その 3 要素をすべて共生/寄生エージェント・モデルによって，（移動エージェントもしくは常駐エージェントとして）モデル化することを試みる．これにより，従来よりワークフロー（プロセス）に固定されていた視点だけでなく，プロダクトの視点や，パーティシパントの視点を導入することができ，記述を局所化することができる．

プロセスは，ワークフロー全体の振舞いの記述でありアクティビティの系列である．プロダクトはワークフロー中で生成されるドキュメント類であり，典型的なビジネスプロセスでは伝票がこれにあたる．ほかに，生産活動を含むプロセスでは，そのすべての成果物がこれに含まれる．パーティシパントは，いわゆるアクティビティの担当者であり，人間の場合とアプリケーションプログラムの場合がある．

4.1.1 プロセスのモデル化

移動エージェントは，一般にその利用者の意図を代行して，ネットワーク中を移動しながら，その意図を満足させるタスクを遂行する．継続実行が可能なエージェントによって，ワークフローを表現する（ワークフロー・エージェントと呼ぶ）．DOPA-L の場合，メインスクリプトもしくは，act メッセージに反応する act メッセージハンドラというスクリプトによって，活動（activity）の系列を記述する．

動的ワークフローにおけるワーフロー間の演算としては，具体化（詳細化結合），線形結合，並列結合などが考えられる．具体化はある活動を，より詳細な活動系列で置き換えることである．線形結合は複数のワークフローを逐次実行する結合であり，並列実行は複数

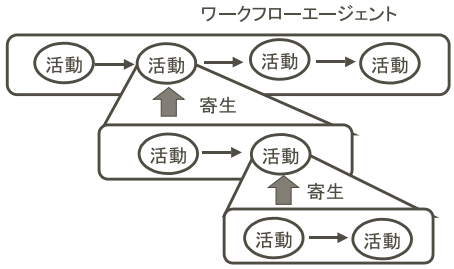


図 8 具体化による部分ワークフローの動的獲得
Fig. 8 Dynamic acquisition of partial workflow by instantiation.

のワークフローが並列実行されるように結合することをいう．このうち線形結合と並列実行は，そのためのパターン（コネクタと呼ぶ）を部品ワークフローとして与えておいて，それを具体化することで表現する．具体化は，寄生機能によって動的に行われる．**図 8** は，具体化（詳細化結合）を寄生によって行う様子を示したものである．実行時に決まる部分活動系列は，ロール名を使って表現され，実行時に寄生するエージェントによって具体化される．

具体化をワークフロー・エージェントの移動先で行うことによって，局所性が実現できる．たとえば「納品」の手順が企業内の部署ごとに，若干異なる場合には，ワークフローがその部署に到着した際に，その部署の「納品」エージェントを動的に寄生させ（5.1 節の待伏せパターンもしくは現地調達パターン),「納品」ロールと結合させて，そこに act メッセージを送ることで部分系列を実行させる（図 8）．これによって，ワークフローエージェントがどのドメインに移動しているときに結合を行うかで，適切な活動系列（納品手順）が選択されることになる．

4.2.2 項では，この動的結合のモデルを使って，ワークフローとして当初定義されていなかった異常事態が発生したときの振舞いを動的再計画する例を示す．

4.1.2 プロダクトのモデル化

典型的なプロダクトは，伝票であり，それを移動エージェントとしてモデル化することは，電子的な伝票をオブジェクトとしてモデル化することに近い考え方である．プロダクトエージェントがプロダクトエージェントに寄生することでプロダクトに階層的な構造を持たせたり，内容の一部の追加や交換を行ったりすることは，寄生の概念を持って行われる（**図 9** (a)）．プロダクトの構成要素は，それぞれ異なるライフサイクルを持つので，個々に個別のエージェントとして表現することが有効である．

また，ワークフローエージェントに対し，中間生成

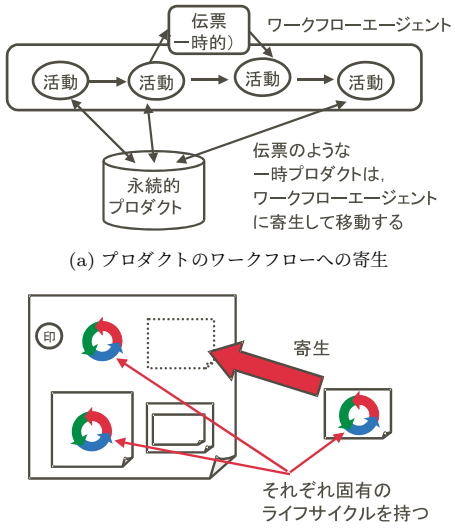


図 9 ワークフローエージェントへの寄生とプロダクトエージェントへの寄生
Fig. 9 Parasitism of a product agent to a workflow agent and a product agent.

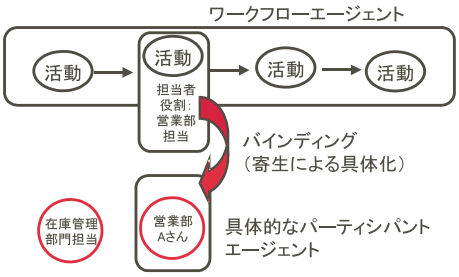


図 10 パーティシパントエージェントとワークフローエージェントの結合
Fig. 10 Binding of a participant agent and a workflow agent.

物であるプロダクトエージェントを寄生させて，一緒に移動させるというパターンもある（図 9 (b)）．あくまで一時的なプロダクトを扱うには有効な手段である．

4.1.3 パーティシパントのモデル化

活動（activity）を実際に遂行するのは，担当者（パーティシパント）である．担当者が人間である場合には，個人情報の管理や活動の遂行を依頼/管理するワークリストハンドラなどの機能を備えたパーソナル・エージェントとなる．担当者がアプリケーションソフトウェアである場合には，それをラッピングしたエージェントがこれにあたる．ワークフローエージェントとパーティシパントエージェントの結合も寄生機構を使って実現され，同期にも使われる（**図 10**）．

結合にはどちらが主導権をとるかで 2 通りの表現が

```
<event-handler event="arrival">
 <param id="p1"> Host-candidate </param>
 <main>
  <if>
    <condition>Host-candidate=="h1"</condition>
    <then>
      <send-message>
        <to>  h1 </to>
        <act> request </act>
        <content> go(act-role) </content>
      </send-message>
      <wait-message from="h1" />
    </then>
  </if>
 </main>
 <message-handler act="reject">
  <raise event="fatal" />
 </message-handler>
</event-handler>
```

図 11 部分プロセスを現地獲得する例（イベントハンドラのみ）
Fig. 11 An example of dynamic acquisition of partial process (event handler only).

ある．1つは，パーティシパントエージェントが，関連するワークフローエージェントの宿主への到着イベントに反応して，そのワークフローエージェントに寄生する（待伏せパターン）．あるいは，ワークフローエージェントが作業遂行に必要なパーティシパントエージェント（担当者）を現地調達する．これは，宿主に対し問合せ（派遣依頼）を行うことで実行される．

結合されたパーティシパントは，ワークフローからはロールとしてアクセスでき，act メッセージを送られる．それに対して，パーティシパントは，ワークリストへの追加（非同期的な活動の場合），担当者（人間）への何らかの手段での即時通知などを行う．

4.2 実際の記述例

動的なワークフロー記述の具体例として，
(1) プロセスを移動先で獲得する例，
(2) 組織改変による担当者変更への対応，
(3) 大域的一貫性監視保守エージェントの動的追加，
を取り上げる．

4.2.1 部分プロセスを移動先で獲得する例

ここでは，ワークフローエージェントに対して，部分プロセスを意味する別のワークフローエージェントが動的に結合する待伏せパターン[1)]（5.2 節）を使う．このパターンは，待伏せる側のエージェントのイベントハンドラで表現できる．

図 11 のイベントハンドラでは，エージェントは寄生したい h1 という名前のエージェントの到着を到着通知イベント（arrival）で知り，act-role というロール名で寄生する．宿主であるワークフローエージェントの側の記述は省略しているが，act-role に対して act というメッセージを送ることによって，そのメッセージへのハンドラとして記述されている部分プロセスを実行する．

```
<event-handler event="absence">
 <param id="p1"> Abs-role </param>
 <main>
  <send-message>
    <to>  planner </to>
    <act> request </act>
    <content>
      <plan>
        <goal>    get-task(Abs-role)</goal>
        <resource>without(Abs-role) </resource>
      </plan>
    </content>
  </send-message>
  <wait-message from="planner" />
 </main>
 <message-handler act="reply"
                 content="Other-resource">
  <set-role role="Abs-role">
   Other-resource
  </set-role>
 </message-handler>
 <message-handler act="reject">
  <raise event="fatal" />
 </message-handler>
</event-handler>
```

図 12 動的再プランニングの例
Fig. 12 An example of dynamic replanning.

4.2.2 組織改変による担当者変更への対応

この事例においては，担当者の不在という例外イベントの発生に応じて，イベントハンドラでプランナエージェントを起動し，その結果，寄生要求を送る相手先エージェントを切り替えて，別の担当者と動的結合するような例題を考える．

図 12 のイベントハンドラは，寄生によって動的結合されるロール（担当者）が不在のとき，発生するイベント absence に反応する例外処理ハンドラである．不在のロールが変数 Abs-role に返される．そのロールが act というメッセージに反応して実行すべきゴールが task という名前で参照できる．Abs-role は不足している資源を意味する．

absence イベントに対してハンドラのメインスクリプトが起動される．それはロール planner への再プランニング要求を出し，Abs-role のタスクを入手して，Abs-role が不在という資源条件下で，再プランニングを依頼し返事を待つ（wait-message はメッセージの到着を待ち対応するメッセージハンドラを起動する）．結果として生成されたエージェント（プランである活

動系列）が得られたら，それを Abs-role と結合し，そこへ act メッセージが送られてプランが実行される．再プランニングが失敗もしくは拒絶された場合には reject メッセージを受け取る．これに対しては，fatal イベントを発生させるが，図 13 ではそのハンドラ記述は省略している．

4.2.3 大域的一貫性監視保守エージェントの動的追加

この項では，動的ワークフロー管理の具体例の 3 番目として，大域的な一貫性を保守するために，依存関係制約を監視し，制約違反が発生したときにイベントを発生させる監視エージェントと，イベントが発生したときにそれを保守する手続きを計画し起動する保守エージェント，ならびに，大域的なイベント通知のためにイベントを伝播させる，イベント伝播エージェントを示す．

監視エージェントの寄生には，クッションパターンを使う．クッションパターンは，宿主と寄生者の間に割り込んで，その間の通信を傍受すると同時し，内部状態を監視することができる．これによって，監視対象エージェントが必要な I/F を提供していれば，それ以上の手を加えずに監視することができる．

保守エージェントは監視エージェントの発火させたイベントの発生通知を受けて，保守手続きを計画し，その結果として生成されたエージェントが，保守手続きを実施する．

本エージェントモデルでは，イベントは，そのイベントのための raise 節を実行したエージェントが寄生している宿主において発生し，その宿主ならびに，全寄生者（のうちそのイベントのハンドラを持っているエージェント）が通知対象となる．したがって，そのままでは，大域的な通知は行われない．そこで，その宿主に寄生していないエージェント，ならびに，移動しているエージェントに着実にイベントを通知するために，イベント伝播エージェントがある．イベント伝播エージェントは，イベントを仲介し，別のエージェントにイベント発生を依頼する．これによって，イベントの即時性は完全には満たされないが，大域的なイベント通知が実現できる．

4.3 共生/寄生モデルによるワークフロー記述とその一般化

従来，ワークフロー定義は，プロセス（活動系列）を中心に表現することが多く，モデル化のための視点が固定化される傾向にあった．これに対し，本論文で提案している，プロセス，プロダクト，パーティシパントを，それぞれエージェントとして記述し，動的に結合させることを可能にするモデルは，3 つのビューから複合的に対象をとらえることを可能にしている．

また，それらの階層性を動的に再構築できるモデルであることから，位置に依存した振舞い，振舞いの動的な変化などを統一的に表現することができる．こうしたメリットは，ワークフロー管理システムへの応用ばかりではなく，一般に発展的な P2P アプリケーションの構築に有効であるといえる．

しかし，3 章と 4 章で示した共生/寄生モデルの機能だけでは．2.2 節でできるだけ高い抽象レベルで与えた P2P アプリケーション構築の課題のすべてに完全な解を提供しているとはいえない．というのは，共生/寄生モデルが提供する機能は，ごく基本的なプリミティブの機能にすぎず，そのモデルに基づいた記述言語が P2P アーキテクチャのプログラムを書くのに便利な機能を持っているいうことにすぎない．そうしたプログラムをどのように構築していくかという方法論については論じていない．たとえば，いくつかの課題に対して,「部分機能の動的交換/獲得」というアプローチで対処することを 2.2 節で述べたが，4.2.1 項で具体的記述例を示しているとはいえ，それは一例にすぎない．ほかにもいろいろな動的結合のやり方がある．そうしたいろいろなやり方（動的結合戦術）のいくつかは，5.2 節でパターンとして認識している．また，共生/寄生モデルは，リアクティブ性を持たせるためにイベント通知機能を備えているが，具体的にどのようなイベントを組み込みプリミティブとして用意するべきかに関しては，解が得られていない．適用するアプリケーション分野に依存するからである．それによって，保守できる一貫性制約や依存関係の種類に限界が生じることとなる．また保守手続き自体についても，プログラミング可能であるというだけで，たとえば，相互に依存しあうような依存関係や矛盾する制約を与えられたときに，デッドロックに陥ったり，無限にループしたりすることを禁止するような機能は，何も与えていない．これから，そういった機能を実現していくうえでの，プラットフォームとなる基本概念を提供するものとして，共生/寄生モデルを位置付けている．

5. 結合の戦略と階層的モデリング技法

5.1 階層的モデリング技法

前章では，ワークフロー定義を共生/寄生エージェントモデルに基づいて行った．これによって，動的なワークフロー管理に限らず，発展的な P2P アプリケーションの構築に役立つ各種の利点を得ることができた．

この章では，動的なワークフロー管理の考え方を，単なる P2P アプリケーションの一例にとどめず，P2P アーキテクチャにおいて自律的分散要素の的確な連携を実現するための構築技法として導入することを試みる．

基本的な考え方は，

(a) アプリケーションの構成要素（ピア）をエージェントで表現すること，

(b) 構成要素間の連携は，エージェント間のワークフローとして表現する（ワークフロー自体もエージェントとして表現する）こと，

(c) 協調プロトコルといったパターンを利用してワークフローを実現すること，

というものである．ここで，アプリケーション全体の表現としてワークフローを位置付けて，協調プロトコルはその実現のための設計パターンと考えていることに注意してほしい．

まず，第 1 層（タスク層）として，アプリケーションを構成するエージェント（ワークフロー管理におけるパーティシパントに相当するが，必ずしも人間の担当者を意識したものではなく，一般のソフトウェアエージェントを考える）を認識する．さらにその個々のエージェント上で実行される典型的なタスクをモデル化する．

次に第 2 層（ワークフロー層）としてノード間にまたがった振舞い（ワークフロー）と，そこでやりとりされるデータをモデル化する．これにより，エージェント間の基本的な連携の方法を定義する．

さらに第 3 層（制約層）として，第 1 層，第 2 層でモデル化した要素の間の依存関係を整理し宣言的に記述する．ここでは，典型的なケースに対して例外イベントを整理し，それに対処するためのイベントハンドラの記述も行う．これによって動的な側面が強化される．

第 4 層（結合設計層）として 1〜3 層でモデル化した要素を動的結合する戦略をパターンに基づいて規定する．このように，この技法は，4 階層の階層的技法となる．このうち，第 1 層から第 3 層までに関しては，どのようなエージェントを使って，どのような観点から記述するかについては前章で述べた．そこで，次節では，第 4 層における戦略パターンを中心に述べる．

5.2 分散要素の結合設計パターンの例

この節では，第 4 層で使われる結合設計パターンの例をいくつか示す．ここでは，その連携のパターンをさらに，2 つに分類している．

(パターン-1) 振舞いの動的結合[1)]

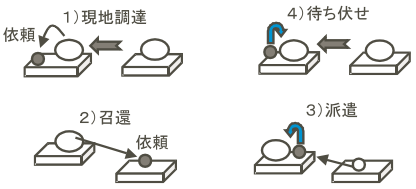


図 13 振舞いの動的結合戦術のパターン
Fig. 13 Patterns of dynamic binding tactics for behavior.

表 3 振舞いの動的結合戦術のパターン[1)]
Table 3 Patterns of dynamic binding tactics for behavior.

		主導権	
		宿主	寄生者
移動	宿主	1-1) 現地調達	1-4) 待伏せ
	寄生者	1-2) 召喚	1-3) 派遣

(パターン-2) 宿主による寄生者へのメッセージ制御

これらのパターンは 4.2 節の例題で利用している．

まず，第 1 に，下記の

(1) 振舞いの動的結合のパターン（**図 13**）

がある．これらはすでに文献 1) において**表 3** のように整理されている．

宿主は機能の利用者，寄生者は利用される機能を意味する．(1-1) と (1-4) は，機能の利用者である移動エージェントが，移動先で必要な機能エージェントを獲得するものである．(1-2) と (1-3) は，ネットワークを介して遠隔地に，機能を送り込んだり，呼び寄せたり，あるいは，移動してきた機能を捕まえたりするものである．(1-2) 召喚パターンにおいて，求める機能を実現したエージェントを知っていれば，そこに直接的に依頼を送ればよく，さもなくば，そうしたエージェントを検索する能力を持ったエージェント（ファシリテータ）に召喚依頼を送ればよい．1 つ注意すべきことは，必要な機能が来るまで機能エージェントを，利用者エージェントが待つという振舞いは，待伏せパターンではなく，派遣パターンに分類されるという点である．

これらのパターンは，エージェントの到着イベントに反応するイベントハンドラによって記述される（4.2.1 項の例題参照）．

次に，

(2) 宿主による寄生エージェントへのメッセージの制御

による動的結合戦術のパターンを提案する．これには，

(2-1) 横取り（intercept），

(2-2) フィルタ（filtering），

(2-3) クッション（cushion），

(2-4) 多重化（multiplication），

(2-5) 単一化（unification），

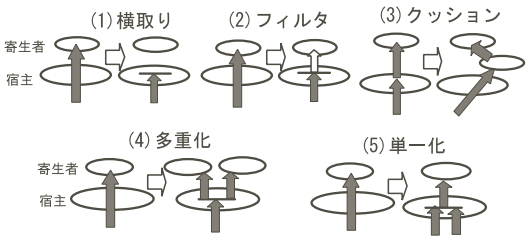


図 14 宿主によるメッセージ制御の動的結合戦術のパターン
Fig. 14 Patterns of dynamic binding tactics for message controlling by host.

などがある（**図 14**）.

ここで，(2-1) の横取りは宿主が通信を奪い取って処理してしまうものである．(2-2) のフィルタは宿主が通信内容を書き換えてから本来の受け取り手である寄生者に送るものである．(2-3) クッションは別のエージェントが寄生して，元々の受け手である寄生者を内部に取り込んでラッピングしてしまう（つまり，宿主と本来の受け取り手の間に入ってしまう）ものである．(2-2) と (2-3) の 2 つは，本来の受け手への通信内容が変化する．残りの 2 つも互いに深い関係にある．多重化は通信をコピーして本来の受け取り手以外へ同一通信内容を送るもの，逆に単一化は高信頼化などのために冗長化された通信を一本化するものであり，同一内容のものを 1 つにまとめるだけではなく，場合によっては多数決原理による一本化なども含む.

本節で示した基本パターンを組み合わせることで，さらに，特定の目的に合致した複合パターンを作ることができる.

5.3 共生/寄生モデルと発展性

今回，対象とするアプリケーションにおいては，ソフトウェアが時間的に応じて変化していき，環境の変化に適応するというものであった．そのようなソフトウェアをここでは発展的ソフトウェアと呼び，それを支えるアーキテクチャを発展的ソフトウェアアーキテクチャと呼ぶこととする．たとえば，リフレクティブアーキテクチャは，その一例である．また，形式的記述などの高レベル記述からコードを生成することを目指す自動プログラミングや，部品の組合せによってソフトウェアを開発するコンポーネントに基づく開発も，そのプロセスをアーキテクチャに組み入れることで（たとえば高レベル記述の自動書き換え機能や，コンポーネントの自動選択機能によって）発展的アーキテクチャとなりうる.

そこで，コードを変更してソフトウェアの機能を動的に変化させるにあたって，コードをどの程度直接的に扱うかに関してレベル分けができる.

(1)「直接書き換え」（コードを直接書き換える）
(2)「コード生成」（高レベル記述からコードを生成）
(3)「部品交換」（コードを含むモジュールを交換）

これらは，独立のものではなく，たとえば，高レベルの表現を書き換えて，そこからコードを生成する方法は (1) と (2) の組合せである．本論文の共生/寄生エージェントによる方法は，基本的に (3)「部品の交換」によるものであるが，一部に，プランナによってコードを生成するといった (2) の方法も含んでいる.

本章では，この共生/寄生エージェントによる発展的な動的側面と，ワークフロー管理の考え方で直接的に協調方法/連携方法を管理する P2P アプリケーションの構築法を与えた.

6. 評 価

2 章であげた各課題へのアプローチの基本方針が，共生/寄生エージェントモデルでどのように表現できたか，その具体化の 1 つであるエージェント記述言語 DOPA-L でそのように表現できたか，どういうものが取り扱えないかといったことを定性的に述べる．これによって，課題が，どの程度達成されたかを示す.

まず,【課題 1-1：資源の多様性への対処】は，寄生機能に基づいて動的に導入できる階層性と，ワークフロー，パーティシパント，プロダクトのそれぞれを独立してエージェントとしてモデル化するという局所性によって行った．その結果，たとえばパーティシパントごとの多様性は，宿主エージェントに記述した基本的なパーティシパント記述に加えて，固有性を寄生エージェントとしてアタッチしていくことで，分かりやすく管理することができるようになった．システム設計者は，ビューごとに設計を進めて，個別のエージェントとして実装し，実行時に動的にそれらを結合することができる．ワークフロー例題に関しては，ビューへの分解と，その連携を分かりやすく表現できた．ワークフロー例題以外の P2P アーキテクチャに一般化できるかどうかという点では，5 章でワークフロー管理の考え方を，一般の P2P アーキテクチャに適用する方法をまとめた．しかし，この方法の有効性を評価することは，今後の記述実験の結果を待たなければならない.

バラバラに開発された構成要素を統合して全体として一貫した振舞いをさせるには，構成要素間の依存関係に基づいた一貫性を保守する機能が必要になる．これが 2 つ目の【課題 1-2：依存関係の取扱い】に相当していた．共生/寄生エージェントモデルでは，構成要素である各種のエージェント（データを表現するプロ

ダクトエージェントも，プロセスを表現するワークフローエージェントも同じように区別なく取り扱える）に動的に寄生させることのできる一貫性監視保守エージェントがそれを行う．その一貫性監視保守エージェント，寄生によって形成されたドメインごとに通知範囲を決めるイベント通知メカニズムに基づいて，ワークフロー（作業プロセスの基本的な流れ）とは独立にリアクティブかつ協調的に機能する．4 章で与えた例では，エージェントの到着や出発，そして不在といったことに関して用意されている組み込みイベントに関しては，取り扱うことができることを示せた．しかし，ほかにどのようなイベントを扱う必要があるのかについては適用するアプリケーション分野に依存するので取り扱っていない．また，保守手続き（イベントハンドラ）としてどういうプログラムを書くのかはエージェントの記述プログラムの開発者に任されている．そしてイベント通知の即時性（リアルタイム性）を保証するかどうかも，モデルで規定することではなく，エージェント記述言語ならびにその実行時環境（プラットフォーム）の実装に依存することである．

第 3 の【課題 2-1：要求機能の時間的なズレへの対応】は，5.1 節で与えた寄生機能に基づく部分エージェントの動的交換によって達成される．実際に，どのようなタイミングで，どういう方法で動的交換をするかは，エージェント記述プログラムの開発者のプログラミング次第である．5.2 節において，移動エージェントのための部分エージェントの動的交換のパターン「(パターン-1) の振舞いの動的結合」を整理した．こうしたパターンの共有化と再利用によって，目的に合致した動的交換の方法を用意に選択して取り入れることができるようにする努力を続けている．

第 4 の課題【2-2：構成要素の I/F のズレへの対応】は，ACL（エージェント通信言語）に基づくモジュール間結合で達成されている．関数やメソッドをベースとした API 呼び出しに比べて，ACL によるモジュール間の結合は，柔軟かつ動的な結合を提供することができる．しかし，3.5 節のエージェント記述言語 DOPA-L の例では，メッセージとメッセージハンドラの結合は API 呼び出しに比べてそれほど柔軟性が増しているわけではなく，せいぜい，一部の言語で導入されているキーワードパラメータリストのレベルである．しかし，5.1 節で紹介したパターン (3-3)「クッション」パターンは，レガシソフトウェアの再利用で使われるようなラッピング（wrapping）を実現するものであり，(パターン-1) の動的結合パターンと組み合わせることで，動的な仲介エージェントの導入を実現することができる．

第 5 の【課題 2-3：現実の組織構造の変化とのズレへの対応】は，現実の組織構造をエージェントをベースにモデル化することで，寄生機能に基づく動的な再構成によって，ある程度は対応することができる．エージェントに再構成のためのイベントハンドラを用意しておくことで，トリガとなるイベントの発生により，人手を介さずに現実の組織構造の変化にともなって，再組織化を行うシステムを構築することができる．その基本的な例を 4.2.2 項で与えた．しかし，現実の組織構造の変化を検知する方法としては，DOPA-L では，現在のところ，ロールの不在イベント以外に想定していない．したがって，それで記述できる範囲のものしか扱うことができない．

本論文で提案する方法において達成できていない課題についてまとめる.【課題 1-1：資源の多様性への対応】を効率良く実現することや，そもそも 2 章であげた課題の根本原因となっているオープン性は，必要な機能をサービスしてくれるモジュールを検索するマッチメーキング機能と，サービス連携の方法を決めるプランニング知識を要請する．これらに関しては，プランナの起動を示唆したにすぎず，情報/知識の流通に関しては取り扱っていない．

また，P2P ネットワークアプリケーションの実用においては，セキュリティ基本機能など，エージェントモデルとは独立に，その実装基盤レベルで実現されているべき機能にも考慮が必要である．セキュリティ技術は通信レベル，暗号化レベル，電子署名レベルといった基本機能を必要とする．通信には，たとえば SSL，暗号化には XML 文書用に W3C や IETF に提案されている DOMHASH [9]，電子署名には XMLDSIG [10] などを使うことが期待できる．そのうえで，ドメイン間のセキュリティポリシの競合解消などに関しては，さらなる研究が必要である．

7. 関 連 研 究

本章ではワークフロー管理システムへのエージェント技術の適用という観点に絞って，(1) ワークフローモデリングの側面（従来のプロセスならびにオブジェクトに着目したモデリングとエージェントによるモデリングの比較など）と，(2) エージェントによる実装の側面を通して，本研究を位置付ける．

ワークフロー管理モデルの標準化は WfMC（Workflow Management Coalition）において行われている [11]．すなわち，WfMC モデルでは，ワークフローを定義をしていくうえでの，視点がプロセス中心に偏っ

ており，その実行システムも集中的である．

エージェントをベースとしたワークフロー管理の研究は多くあるが，その多くは，実装面でエージェント技術（移動エージェントなど）を使うものと，モデリングにエージェントの考え方を使うものに二分できる．

文献 16) においては，ワークフローのモデリングと実現においてオブジェクト指向の考え方を取り入れ，アクティビティを遂行する担当者などを意味する役割と，その間を受け渡される電子伝票をオブジェクトとしている．

ワークフロー管理システム WorkWeb System[17]，INA/LI[18] は，マルチエージェントによる動的な再計画，最適化を特徴とするものである．ここではパーティシパントエージェントに相当する電子秘書エージェントの概念や，ワークフローの再プランニングの機能が実現されている．しかしワークフローそのものや，エージェント間でやりとりされる情報は，エージェントとしてモデル化されてはいない．

これらに対して，本論文では，プロセス中心の見方だけでなく，プロダクトやパーティシパントといった分散した要素も同じレベルで取り上げるだけでなく，ドメイン固有の操作手順といったプロセスもドメインごとに分散して管理することを実現している．

アンダーセンコンサルティングの TSE（TRP（Technology Reinvestment Project）Support Environment）[19] は．TRP のコンポーネントに基づくソフトウェア開発環境を使って作業する作業者同士の能動的な協調作業を可能にするものである．そこでは，機能エージェント（authoritation agent），アクタエージェント（actor agent），個人エージェント（personal agent）がある．機能エージェントのうちワークフロー AA は，他のエージェントと，既存のワークフローツールやリポジトリとの間のファシリテータとして機能する．個人エージェントは，実際の利用者に対応する．アクタエージェントは，オブジェクトモデル作成者のような特定のロールを持った個人エージェントであり，移動エージェントである．文献 19) ではこれらエージェント間の通信言語 TACL（TRP Agent Communication Language）を与えている．これは，モデリングのためにエージェント技術を使っている例に属する．

文献 20) では，AWA（Agent-based Workflow architecture）アーキテクチャとして，トリガエージェント，ワークフローエージェント，プロセスエージェント，タスクエージェント，個人エージェント，登録エージェントなどのタイプのエージェントをあげている．AWA や文献 21) では複数の異種エージェントの間での相互作用として，ワークフローが実現される．

HP 研究所の「電子商取引自動化（E-Commerce Automation）」[22),23)] では，Dynamic Agent と呼ぶ，アプリケーション固有の動作が変更可能な移動エージェントの集まりとして電子商取引を取り扱う．そのエージェント間通信の ACL による相互作用に，XML を使う点などは本研究と似ている．しかし，動的なワークフロー管理の詳細に関して，具体的な Dynamic Agent の利用法の詳細ならびに，実際に活用していくうえでのパターンなどについては与えられていない．

主に実装面でエージェント技術を利用した研究について示す．非集中型システムの開発に移動エージェントを利用することを論じるものとして，ペトリネットで振舞い記述をするエージェントによって既存の分散プラットフォーム COSM（Common Open Service Market）を拡張し，組織間ワークフローの取り扱うものがある[24]．文献 25) は，移動エージェントをワークフロー管理システムの実装に使うにあたり，性能をチューニングするためのアーキテクチャの提案である．既存の分散ワークフローシステム（CORBA ミドルウェアを使っている）と既存の分散エージェントシステム（メッセージ指向ミドルウェア：MOM）を CORBA サービスを通して相互通信できるようにするプラットフォームに TCCS（Task Control and Coordination Service）がある[26]．ANAISoft（Advanced Network and Agent Infrastructure for the Support Of Federations of Workflow Trading System）プロジェクト[27] は，交換取引市場（マーケット）に基づく，ワークフロー管理アーキテクチャを提供するものである．ECA（Event-Condition-Action）ルールを使い，同調してタスク遂行するようなエージェントによって，非集中的で柔軟なワークフローの実現を行う研究もある[28]．Magi（Micro-Apache-Generic Interface）-agent[29] は，Web をベースにした Workflow のためのメッセージングサービスである．

関連研究 30) は，ワークフローモデリングと分析設計の方法論のレベルを扱っていることで他のものと異なる．プロジェクト ADEPT（Advanced Decision Environment for Process Task）[31),32)] は分析から実装までを具体的に取り扱っている．他の関連研究としては，異なるアプリケーション分野での事例（通信分野[33)]，ヘルスケア分野[34)]）もある．

以上のような関連研究に対して，本研究の意義は，共生/寄生モデルというエージェント自体の構成を変化させるモデルを採用して，ワークフロー定義の各要素をうまく表現していること，設計プロセスにも踏み

込んでエージェント間相互作用の一面のパターン化を行っていること，そして，それをワークフロー管理にとどまらず，むしろワークフロー管理の考え方を取り入れた P2P アプリケーションの構成を与えている点である．

8. おわりに

本論文では，C/S アーキテクチャから P2P アーキテクチャへという，これからのネットワークアプリケーションの流れをふまえて，そのためのアプリケーション構築法にアプローチした．今後すべてのアプリケーションが C/S 構成から P2P 構成にとって代わられるわけではなく，それぞれの得失を活かした適切な棲み分けがなされなければならない．しかし，自律性を備えた分散要素によるアーキテクチャの占める比重は高まりつつある．本論文では，マルチエージェントエージェントを 1 つのエージェントと見なし，動的に構成を変化させることのできるソフトウェアモデルを使って，その自律性と発展性を活かすように多様性や変化に耐える動的な P2P アプリケーションのための構築する方法を与え，動的ワークフロー管理への応用を通してその有用性を示した．さらに，その構築技法を発展させ，動的ワークフロー管理の考え方を取り入れることで，エージェント間の的確な連携を分析設計できるような 4 階層の構築技法を提案した．

参考文献

1) 本位田，飯島，大須賀：エージェント技術，共立出版 (1999).
2) Oram, A. (Ed.): *Peer-to-Peer Harnessing the Power of Disruptive Technologies*, O'Reilly (2001).
3) O'Reilly Research: *2001 P2P Networking Overview—the Emergent P2P Platform of Presence, Identity and Edge Resources*, O'Reilly (2001).
4) 山崎重一郎：P2P ネットワークシステム，人工知能学会誌，Vol.16, No.6, pp.834–840 (2001).
5) 河井　淳：ピア・トゥ・ピアシステムとモバイルコンピューティングへの適用，人工知能学会誌，Vol.16, No.6, pp.762–767 (2001).
6) Putman, J.R.: *Open Distribution Software Architecture Using RM-ODP*, Prentice Hall PTR (2000).
7) Napster, http://www.napster.com/index.html
8) Gnutella, http://www.jnutella.org/
9) http://www.trl.ibm.com/projects/xml/domhash.htm
10) http://www.trl.ibm.com/projects/xml/xmldsig.htm
11) WfMC (The Workflow Management Coalition) ホームページ，http://www.aiim.org/wfmc/
12) http://www.jxta.org/
13) Leymann, F.: Web Services Flow Language (WSFL 1.0), IBM (2001). http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf
14) The DAML Services Coalition: DAML-S: Semantic Markup For Web Services (2001). http://www.daml.org/services/daml-s/2001/10/daml-s.pdf
15) http://www.fipa.org/
16) 稲本，佐藤，水野，片岡：ワークフロー管理システム構築のための汎用的なオブジェクト指向モデル，情報処理学会論文誌，Vol.39, No.7 (1998).
17) 垂水，喜田，柳生，石黒：エージェントによるワークフローの動的再計画，情報処理学会論文誌，Vol.39, No.7 (1998).
18) Ishiguro, Y., Tarumi, H., Asakura, T., Kida, K., Kusui, D. and Yoshifu, K.: Office Work Support with Software Agents, *NEC Res. and Develop.*, Vol.38, No.2 (1997).
19) Chang, J.W. and Scott, C.T.: Agent-based Workflow: TRP Support Environment (TSE), *Proc. 5th International World Wide Web Conference* (1996).
20) Hawryszkiewycz, I. and Debenham, J.K.: A Workflow System Based on Agents, *Database and Expert Systems Applications, 9th International Conf., DEXA '98*, Quirchmayr, G., Schweighofer, E. and Bench-Capon, T.J.M. (Eds.), pp.135–144 (1998), and also LNCS-1460, Springer (1998).
21) Stormer, H.: A flexible Agent-Workflow System, *Workshop on Agent-Based Approaches to B2B at the 5th International Conference on Autonomous Agents* (*AGENTS 2001*) (2001).
22) Chen, Q., Hsu, M., Dayal, U. and Griss, M.: Multi-Agent Cooperation, Dynamic Workflow and XML for E-Commerce Automation, *Proc. 4th International Conference on Autonomous Agents* (2000).
23) Chen, Q., Dayal, U., Hsu, M. and Griss, M.: Dynamic Agents, Workflow and XML for E-Commerce Automation, *Proc. 1st International Conference on E-Commerce and Web-Technology* (2000).
24) Merz, M., Liberman, B. and Lamersdorf, W.: Using Mobile Agents to Support Interorganizational Workflow-Management, *Proc. Int. Journal on Applied Artificial Intelligence*, Vol.11, No.6, pp.551–572 (1997).
25) Foster, S.S., Nebesh, B.A., Moore, D. and

Flester, M.J.: Performance Tuning Mobile Agent Workflow Applications, *Proc. Technology of Object-Oriented Languages and Systems* (*TOOLS*) (1999).
26) Shrivastava, S., Bellissard, L., Feliot, D., Herrmann, M., De Palma, N. and Wheater, S.: *A Workflow and Agent based Platform for Service Provisioning*, Control and Coordination of Complex Distributed Services ESPRIT Long Term Research Project, TR-32 (2000).
27) Hulaas, J., Stormer, H. and Schonhoff, M.: ANAISoft: An Agent-based Architecture for Distributed Market-based Workflow Management, *Proc. Software Agents and Workflows for Systems Interoperability workshop of the 6th International Conference on CSCW* (2001).
28) Joeris, G.: Decentralized and Flexible Workflow Enactment Based On Task Coordination Agents, *Proc. 2nd Int'l. Bi-Conference Workshop on Agent-Oriented Information Systems* (*AOIS-2000 and CAiSE*00*), pp.41–62 (2000).
29) Bolcer, G.: Magi: An Architecture for Mobile and Disconnected Workflow, *IEEE Internet Computing* (May/June, 2000).
30) Yu, L. and Schmid, B.F.: A Conceptual Framework for Agent Oriented and Role Based Workflow Modelling, *CaiSE WorkshoControl and Coordination of Complex Distributed Services ESPRIT Long Term Research Project on Agent Oriented Information Systems* (1999).
31) Jennings, N.R., Faratin, P., Norman, T.J., O'Brien, P. and Odgers, B.: Autonomous Agents for Business Process Management, *Int. Journal of Applied Artificial Intelligence*, Vol.14, No.2, pp.145–189 (2000).
32) Jennings, N.R., Norman, T.J., Faratin, P., O'Brien, P. and Odgers, B.: *Implementing a Business Process Management System using ADEPT: A Real-World Case Study*, Vol.14, No.5, pp.421–463 (2000).
33) EURESCOM Project P815: *The Application of Agent Technology to Workflow Management in Telecommunications* (2001).
34) Kaldoudi, E., Zikos, M., Leisch, E. and Orphanoudakis, S.C.: Agent-Based Workflow Processing for Functional Integration and Process Re-engineering in the Health Care Domain, *Proc. EuroPACS'97* (1997).

(平成 13 年 10 月 9 日受付)
(平成 14 年 3 月 14 日採録)

飯島 正（正会員）

1991 年慶應義塾大学大学院理工学研究科博士課程単位取得退学.（株）東芝勤務を経て，慶應義塾大学理工学部助手．現在に至る．著書（共著）に「分散オブジェクトコンピューティング」,「エージェント技術」（共立出版，1999 年）等.

本位田真一（正会員）

1953 年生．1976 年早稲田大学理工学部電気工学科卒業．1978 年同大学院理工学研究科電気工学専攻修士課程修了.（株）東芝を経て，2000 年より国立情報学研究所教授．2001 年より東京大学大学院情報理工学系研究科教授を併任.工学博士（早稲田大学）．主としてエージェント技術，オブジェクト指向技術の研究に従事．1986 年度情報処理学会論文賞受賞．日本ソフトウェア科学会，IEEE，ACM 各会員．著訳書多数.

土居 範久（正会員）

1969 年慶應義塾大学大学院工学研究科博士課程修了．慶應義塾大学理工学部教授，国立民族学博物館客員教授．工学博士．現在，日本学術会議会員・第 4 部（理学）副部長，総合科学技術会議専門委員，文部科学省科学技術・学術審議会委員，文部科学省情報科学技術委員会委員長，経済産業省ソフトウェア関連戦略タスクフォース座長，経済産業省情報セキュリティマネジメントシステム（ISMS）適合性評価制度運営委員会委員長，経済産業省 IT セキュリティ評価・認証プログラム運営委員会委員長，情報処理振興事業協会（IPA）技術委員会委員長，科学技術振興事業団（JST）計算科学技術委員会委員長，ACM 日本支部長，Object Management Group Japan-SIG 議長等.