修 士 論 文

モバイルエージェントを用いた
情報共有システムの設計と構築

指導教官　渡部 卓雄 助教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

小林 一樹

2001 年 2 月 15 日

要 旨

近年，インターネットの急速な発達により，ユーザは非常に多くの情報を手に入れられるようになった．その反面，ユーザは情報過多により容易に有益な情報を得られないという問題がある．このようなインターネットにおいて，同じような関心を持つ他のユーザと情報を共有するシステムは大変有用である．さらに，情報共有システムを用いて有益な情報を多く得るためには，範囲があらかじめ限定されない，不特定多数のユーザと情報を共有する仕組みが必要となる．また，ユーザの負担を減らすために自律性をもった情報共有システムが望まれている．

本研究では，これらの問題に対するアプローチとしてモバイルエージェントを用いた情報共有システムを設計・構築する．本システムは，モバイルエージェントが持つ移動性と，本システムが提供するディレクトリサービスを利用することによる自律的な情報共有を特徴とし，ユーザのデータと好み (嗜好) を持ったエージェントが，似たような目的を持つエージェントを自律的に探しだして情報共有を行うことができる．

また，開発者が新たな情報共有サービスを容易に開発できるようにするために，柔軟なエージェント間通信 API を備えたフレームワークを提供する．開発者は，提供されるフレームワークを用いることにより，本システム上で容易に独自の情報共有サービスを構築することができる．

本研究では，提案した情報共有システムを Java ベースのモバイルエージェントシステムである AgentSpace を用いて実装し，情報共有サービスの例題として，ブックマーク共有システムを実装することによって，本システムの有用性を実際に確かめることができた．

目 次

図 目 次

表 目 次

第 1 章

はじめに

本章では本研究の背景，目的，提案するシステムについての概要を述べ，本論文全体の構成を述べる．

1.1 研究の背景

本節では，本研究の背景である，情報共有における問題点とモバイルエージェント技術について述べる．

1.1.1 情報共有における問題点

WWW(World Wide Web) をはじめとする多くの情報共有システムの登場と利用者の増加によって，ユーザは非常に多くの情報を共有できるようになった．しかし，近年「情報過多」という言葉が叫ばれ始めている．これは，情報が多すぎることによって，何が自分にとって重要な情報か見極めることが非常に困難になり，かつ本当に必要な情報を検索するためには多くの時間を必要とするという問題である．

ここで，ユーザは自分と興味分野が重なっていないユーザとも情報共有を行っている場合が多い．一般に，ユーザにとって，自分の興味分野に含まれない情報は，不必要な情報でありユーザの情報収集を阻害し，情報過多の一因となっている．

このような問題に対して，同じような関心を持つ他のユーザと情報を共有するシステムは大変有用である．さらに，ユーザが情報共有システムを用いて有益な情報を多く得るためには，範囲があらかじめ限定されない，不特定多数のユーザと情報を共有する仕組みが必要となる．また，ユーザの負担を軽減するために，自律的な情報共有システムが望まれている．

1.1.2　モバイルエージェント技術の誕生

モバイルエージェントは，実行状態を保持したままコンピュータ間を移動することができるプログラムであり，次世代の分散システムの中核技術として注目されている．

モバイルエージェントは，一般的には以下の特徴を持つとされる．

- 移動性： ネットワーク上においてホスト間を移動しながら処理を行なうことができる．
- 自律性：移動先で他からのアクションやイベントが無くても処理を維持継続できる．他のエージェントで発生した障害の影響を受けにくい．エージェントを変更する場合も，ほかのエージェントへの影響を小さくできる．
- 協調性：エージェント同士が互いに通信し情報を交換し合うことによって，協調的に動作することができる
- 局所性：システム全体の情報を持つことはなく，局所的な情報のみで動作する．

これらの特徴により，モバイルエージェントを用いることによって柔軟な分散システムを構築が期待できる．

1.2　本研究の目的

前節で述べたように，同じような関心を持つ不特定多数のユーザを対象とした情報共有システムが望まれている．しかし，様々なのユーザを対象とした情報共有システムを構築する際には，通信量の削減，ネットワーク状況の変化，異なる計算機環境の存在など多くの解決すべき問題がある．

本研究では，これらの問題に対して次に挙げる特徴をもった情報共有システムを構築することを目的とする．

- 似たような情報を求めるユーザ間の情報共有が可能
- 不特定多数を対象とした情報共有が可能
- 情報共有サービスの開発が容易

1.3 本研究の特徴

本研究では，前節で述べた特徴をもった情報共有システムを設計・構築するにあたり，モバイルエージェントを用いた情報共有システムを提案する．

本システムは，モバイルエージェントが持つ特徴と，本システムによって提供されるディレクトリサービスを用いた，自律的な情報共有サービスを特徴とする．これは，ユーザのデータと嗜好を持ったエージェントが，似たような目的を持つエージェント (ユーザ) を自律的に探し出し，互いの情報を交換することによって達成される．よって，ユーザは複雑な操作を行うことなく，効率的な情報共有を行うことができる．

また，本システムは，開発者提へ，柔軟で効率的なエージェント間通信 API を備えた，情報共有サービスを開発するためのフレームワークを提供する．開発者は，提供されるフレームワークを用いることにより，柔軟で効率的な情報共有サービスを容易に構築することができる．

1.4 本論文の構成

本稿の構成を以下に挙げる．

第 1 章: 本研究の背景を述べ, 情報共有システムに求められている事柄を述べた．また，本研究で提案する情報共有システムの概要とその特徴を述べた．

第 2 章: 既存の情報共有システムをいくつか取り上げ，情報共有システムにおける問題点を述べる．

第 3 章: モバイルエージェントの基本的な概念と特徴を述べ，既存のモバイルエージェントシステムの特徴と応用例を述べる．

第 4 章: 本研究で提案する情報共有システムの設計を述べる．

第 5 章: 4 章で提案した情報共有システムを実際に実装する．

第 6 章: 実装した情報共有システムの例題として，WWW ブラウザの持つブックマーク共有サービスを実装し，考察する．

第 7 章: 本研究で設計・実装した情報共有システムについて考察する．また，関連研究についても述べる．

第 8 章: 本研究全体をまとめ，本研究で得られた結論と今後の展望を述べる．

第 2 章

情報共有システム

2.1 既存の情報共有システム

コンピュータとネットワークの発達に伴い，様々な情報共有システムが開発されてきた．本節では，既存の主な情報共有システムについて概要を述べる．

2.1.1 WWW

WWW(World Wide Web) は，インターネットやイントラネットで標準的に用いられているドキュメントシステムであり，欧州核物理学研究所 (CERN) の Tim Berners-Lee 氏によって，1994 年に開発された．

情報提供者は，HTML(Hypertext Markup Language)[4] によって文書を記述し，WWW サーバによって公開する．情報利用者は，WWW ブラウザに代表されるクライアントソフトウェアを用いて，WWW サーバに HTTP[5] 用いて通信を行うことにより情報を利用することができる．HTML には，文書の中に画像や音声など文字以外のデータや, 他の文書の位置 (ハイパーリンク) を埋込むことができる．利用者はハイパーリンクをたどることによって，様々な情報に容易にアクセスできる．

1993 年に WWW ブラウザである Mosaic が米国のイリノイ大学で開発された．Mosaic が無料で一般に配布されたことと，HTML による文書の記述が比較的容易であったことから，WWW は，爆発的に普及し，現在では世界規模での巨大な WWW 網が築かれている．Inktomi と NEC による 最近の発表 [3] では，10 億ページを超える情報が蓄積されており，現在もっとも社会に浸透している情報共有システムと言える．

2.1.2 ディレクトリサービス

本来，ディレクトリには住所録，人名簿といった意味がある．情報共有システムにおけるディレクトリとは，ユーザまたはアプリケーションがネットワーク上のオブジェクトのリファレンス情報を取得することができるデータベースといえる．

近年，企業等では，コンピュータやネットワークの高速化・低価格化などに伴い，多くのコンピュータがネットワークで相互に接続されており，情報共有といったメリットが生まれている半面，システム管理の複雑さが増し，管理コストも増大するといった問題が生じている．このような管理負担を軽減し，管理コストを削減することが求められており，その有力解としてディレクトリサービスが注目されている．

例えば，Sun Microsystems が提供している NIS(Network Information Service) がある．NIS は，ホスト名と IP アドレス，ユーザアカウントとパスワードなどの情報を中央サーバ上で集中管理し，これらにかかわるサービスをクライアントに提供し，ユーザやホストが存在する物理的な場所を意識することなく，その名前を指定するだけで個々のユーザやホストと通信を可能にする環境を提供している．この他，ホスト名と IP アドレスとを結び付ける DNS(Domain Name System) も，ディレクトリサービスといえる．

このように，NIS や DNS のようなネットワーク上の情報を一元管理しているディレクトリが存在すれば，ユーザーはディレクトリサービスにアクセスするだけで，必要なデータやアプリケーションがどこにあるのか，あるいはユーザーとどのように連絡をとればよいのかを知ることができ，管理コストの軽減や操作の簡略化につながるため，今日では欠かせないシステムとなっている．

2.1.3 グループウェア

グループウェアとは，企業内 LAN を活用して情報共有やコミュニケーションの効率化をはかり，グループによる協調作業を支援するソフトウェアの総称である．ここで，グループウェアが対象とするグループとは，共通の仕事 (目的) を持った集団であり，一般的に全社会的な規模ではなく比較的小規模 (部署，研究室など) な集団を指す．

主な機能としては，グループ内のメンバー間および外部とのコミュニケーションを円滑化する電子メール機能，メンバー間の打ち合わせや特定のテーマについて議論を行うための電子会議室機能，メンバー間のリアルタイムな打ち合わせに利用されるテレビ会議機能，グループ全体に広報を行う電子掲示板機能，メンバー間でスケジュールを共有するスケジューラ機能，アイデアやノウハウなどをデータベース化して共有する文書共有機能，稟議書など複数のメンバーで回覧される文書を電子化して流通させるワークフロー機能

などがある．実際の製品はこれらの機能のうちいくつかを組み合わせたものが多い．

近年ではインターネット・イントラネットの普及に伴い，ユーザが Web ブラウザからすべての機能を利用できるようにした製品が主流になりつつある．

2.2 情報共有システムにおける問題点

前節で述べたシステム以外にも，様々な情報共有システムが運用されている．しかし，既存の情報共有システムには以下のような問題点がある．

2.2.1 情報過多

WWW をはじめとする多くの情報共有システムの登場と利用者の増加によって，ユーザは非常に多くの情報を共有できるようになった．しかし,「情報過多」という言葉が叫ばれ始めている．これは，情報が多すぎることによって，何が自分にとって重要な情報か見極めることが非常に困難になり，かつ本当に必要な情報を検索するためには多くの時間を必要とするという問題である．

ここで，情報過多の原因の一つに，ユーザが，共有する相手 (他のユーザ) を絞り込んでいないという問題がある．既存の多くの情報共有システムでは，ユーザは，自分と興味分野が重なっていない他のユーザと情報共有を行っている場合が多い．自分と興味分野が重なっていない他のユーザが持つ情報は，多くのユーザにとって不必要な情報であり，ユーザの効率的な情報収集を阻害している．また，グループウェアのようにあらかじめ興味分野が合っているユーザを対象としている情報共有システムもあるが，得られる情報量が少なく，分野が特定されるといった問題がある．よって，似たような情報を求める不特定多数のユーザと情報共有できるシステムが望まれている．

2.2.2 不特定多数のユーザへの対応

多数のユーザ間で情報がやり取りされるため，ユーザ数の増加に伴い，情報共有システム全体の通信コストの増大，ユーザごとの計算機環境の違いなどの問題が生じる．ここでの計算機環境とは，CPU や通信回線などのハードウェア環境と OS やロケールなどのソフトウェア環境の両方を指す．

2.2.3 自律的なシステムの必要性

ユーザは，情報共有システムに対して何らかの検索をかけることにより，情報収集を行うが，効率的な検索は検索者個人の知識と熟練に依存しているという問題がある [1].

例えば，最も一般的な検索システムである WWW 上の検索エンジンでは，ユーザが特定のキーワードを入力し，データベースがキーワードに適合する情報をユーザに返す．このとき，有用な情報を得ることができるかどうかは，得ようとする情報に適したキーワードを指定できるかにかかっており，ユーザの知識と熟練に依存している．また，詳細な検索条件の指定は初心者には扱いにくい.

このような問題に対して，ユーザに代わって自律性をもったエージェントが他のエージェントと情報交換を行うことによって，ユーザのとって有益な情報を集めてくるようなシステムが望まれている．さらに，このようなエージェントの開発を支援するようなシステムであることが望まれている.

第 3 章

モバイルエージェントシステム

本章では，移動性や自律性といったモバイルエージェントの基本的な概念を述べ，既存のモバイルエージェントシステムの概要を述べる．

3.1 モバイルエージェントとは

エージェント (Agent) という用語が使われるようになってから日が浅いため，モバイルエージェントに対しての明確な定義は定まっていないが，本研究では，モバイルエージェントを「コンピュータ間の移動能力を持つ能動的なプログラム」と定義する [13]．また，モバイルエージェントは，一般的に以下の特徴を持つとされる [14, 12].

- 移動性： ネットワーク上においてホスト間を移動しながら処理を行なうことができる．
- 自律性：移動先で他からのアクションやイベントが無くても処理を維持継続できる．他のエージェントで発生した障害の影響を受けにくい．エージェントを変更する場合も，ほかのエージェントへの影響を小さくできる．
- 協調性： エージェント同士が互いに通信し情報を交換し合うことによって, 協調的に動作することができる
- 局所性： システム全体の情報を持つことはなく， 局所的な情報のみで動作する．

3.2 既存のモバイルエージェントシステム

1994 年に米 General Magic 社が Telescript と呼ぶエージェント技術の仕様を公開してから，さまざまなモバイルエージェントシステムが企業や各研究機関などから提案されてきた．本節では，既存の主なモバイルエージェントシステムの概要を述べる．

3.2.1 Telescript

Telescript [9] は, General Magic 社が開発したモバイルエージェントを利用したオンラインアプリケーションを記述するためのプログラミング言語である．

Telescript では，プレースと呼ばれる非移動プロセスがネットワークの各所に分散しており，エージェントは go 命令によってこのプレース間を移動することができる．言語の実行環境である Telescript Engine の他に Tabriz と呼ばれる開発環境があり，Web から Telescript を操作するための機能や Telescript プロセスから HTTP リクエストを行うための機能がライブラリとして提供されている．

Telescript は，モバイルエージェントの概念を初めて商用レベルの技術にした点で画期的であったが，プラットフォームがアーキテクチャに依存しているといった問題などがあり商業的にはあまり成功しなかった．

3.2.2 Aglets

Aglets [10, 11] は, IBM が開発した Java ベースのモバイルエージェントシステムである．

Aglets の特徴としては，様々な移動パターンやエージェントのロード方式などが選択可能な柔軟性が挙げられる．開発者は，エージェントの処理内容によって，Aglets が提供するいくつかのパターンを用いることにより，プラットフォーム独立で，ネットワーク上を移動しながら実行されるエージェントプログラムを容易に作成することができる．

Aglets のエージェントは AgletContext という環境上で実行され，エージェントの生成，削除，その他の制御のためのビジュアルな環境として Tahiti も用意されており，インターネット上を動き回るエージェントの制御を比較的簡単に行うことができる．

3.2.3 Plangent

Plangent [12] は，東芝が開発した Java ベースのモバイルエージェントシステムである．Plangent は，移動機能だけでなくエージェントの行動計画を自ら生成するプランニング機能を備えていることが大きな特徴である．Plangent のエージェントは，移動機能，プ

ランニング機能を備えることで，ネットワーク上を移動し，その場その時の最適な処理を自ら判断し行動することができる．

エージェントは，ユーザによって与えられたゴールを達成するために自らプランを生成し (プランニング機能)，そのプランにしたがって行動することができる (プラン実行機能)．また，移動先のホスト (Plangent ではノードと呼ぶ) でエージェントの実行が失敗した場合には，失敗の根拠となった情報の更新を行って再プランニングを行うことにより，適応性と自律性を実現している．

3.2.4 AgentSpace

AgentSpace [13] は，お茶の水女子大学の佐藤一郎氏が開発した Java ベースのモバイルエージェントシステムである．

AgentSpace の特徴としては，エージェント移動の高速化が挙げられる．既存のモバイルエージェントシステムの多くは，エージェントの実行状態の転送後に，プログラムコード (クラス) をオンデマンドに転送する．つまり，移動先でクラスプログラムのバインド要求が発生する毎にクラスプログラムの転送要求を送信し，そのクラスプログラムを移動先に転送させる．このため，クラスの数に従って通信回数が増加し，エージェントの移動コストが増大する．

AgentSpace では，エージェントの実行に必要なクラスのうち，移動先に存在しないクラスプログラムをエージェントの状態とともに一括転送できる．このため，エージェントの高速な移動が可能となる．

3.3 モバイルエージェントの応用例

本節では，モバイルエージェントの応用例をいくつか取り上げ，モバイルエージェントを用いたシステムの特徴を述べる．

3.3.1 たび Can

「たび Can」[15] は，Aglets をベースに構築された旅行情報提供サイトであり，海外旅行のパッケージ・ツアーや航空券を検索できる．Aglets は，たび Can の検索エンジンとして用いられている．

たび Can の特徴を以下に挙げる．

- ユーザーが入力した検索条件をエージェントが 24 時間にわたり記憶するため，いったん接続を切ってから再接続しても，同じ検索条件を再度入力しなくてよい．
- ユーザが提示した検索条件に合致したツアーが，検索時に満杯だったとしても，24 時間以内に空きが出た場合にはメールでユーザーへ通知することができる．
- ユーザーが指定した検索条件に合致した結果以外にも，情報提供者がすすめる「おすすめ」を提供できる．

これらの特徴は，ユーザが実際に旅行代理店で行う情報収集とよく似た側面を持っている．従来型の WWW とデータベースの連携技術だけでこのような機能を実現するのは，かなり面倒な作業となる．たび Can では，こうした実世界で収集する情報と似た検索結果を返す検索エンジンを実現するために Aglets を用いている．

[16] によると，エージェントを使って検索エンジンを構成したことの技術的な主なメリットは，以下の 2 つがある．

- 実装の容易さ

  複雑な検索結果を返すエンジンをエージェントを使うことによって比較的わかりやすく設計することができる．
- システムの安定動作

  エージェントはほかのソフトウエア・コンポーネントとは独立に自律的に動作する．このため，ある 1 個の代理店エージェントに不具合が発生した場合にも，ほかのエージェントには影響を与えない．

3.3.2 ゆがしまん

「ゆがしまん」[17, 18] は，Plangent をヒューマンナビゲーションに応用したシステムである．ヒューマンナビゲーションとは，歩行中の利用者に現在位置や周辺情報を与えることで，移動や観光を支援するシステムである．

ヒューマンナビゲーションシステムでは，多様な利用者のニーズを満たすため，利用者の欲する情報をネットワーク上から適切に抽出して提供する必要がある．携帯情報端末を介した情報提供の際には複雑な操作を不要とすることはもちろん，ときには利用者の操作が行なわれていない場合にもシステム側から情報提供を行なうことが求められる．また，通信は PHS を用いて行い利用者の位置は PHS が交信しているアンテナにより特定されるため，効率的で回線状況の変化の影響を受けにくいシステムが必要とされる．

ゆがしまんは，Plangent にパーソナライズ機能を組み込んだ p-Plangent ををを利用して構築することによって，以下の特徴を持つ．

- 個人性 :

  パーソナライズ機能が個人情報・嗜好を管理し，位置情報などの個人状況と合わせることで，利用者の望む情報とそうでない情報にフィルタリングをし情報提供今日を行う．

- 移動性 :

  モビリティ機能がプログラムを実行状態も含めて転送することにより，利用者の利用端末上へ常に追従できる．また，移動性により通信コストの削減と予期せぬ通信回線の切断への対応が可能となる．

- 柔軟性 :

  プランニング機能が操作に失敗した場合の代替操作を生成することにより，ネットワークの環境変化に対応できる．

ゆがしまんはこれらの特徴により，ネットワーク上から利用者に合った情報を抽出し，自発的に提示することができる．なお，ゆがしまんは，伊豆天城湯ヶ島町にて実証実験が 1999 年 8 月から実施されている．

第 4 章

本情報共有システムの設計

本章では，本研究で提案する情報共有システムの設計について述べる．

4.1 設計方針

第 2章で述べたように，同じような関心を持つユーザ間の情報共有システムが望まれている．また，情報共有システムを用いて有益な情報を多く得るためには，範囲があらかじめ限定されない，不特定多数のユーザと情報を共有する仕組みが必要となる．しかし，不特定多数のユーザを対象とした情報共有システムを構築する際には，通信量の削減，ネットワーク状況変化，異なる計算機環境の存在など多くの解決すべき問題がある．

本研究では，これらの問題に対して次に上げる特徴をもった情報共有システムを構築することを目的とする．

- 似たような情報を求めるユーザ間の情報共有が可能
- 不特定多数を対象とした情報共有が可能
- 情報共有サービスの開発が容易

本研究では，これらの特徴をもった情報共有システムを設計・構築するにあたり，モバイルエージェントを用いた情報共有システムを提案する．また，本システムを用いて柔軟な情報共有サービスの開発を支援するために，エージェントの情報を一元管理するディレクトリサービスと，柔軟で効率的なエージェント間通信 API を備えたフレームワークを開発者へ提供する．

4.2 情報共有システムの特徴

本システムは，モバイルエージェントによる自律的な情報共有を特徴とする．これは，ユーザのデータと嗜好を持ったエージェントが，似たようなプロファイルを持つエージェント (ユーザ) を自律的に探し出し，互いの情報を交換することによって達成される．よって，ユーザはあらかじめ情報共有する相手を知っておく必要はない．

また，開発者へ新たな情報共有サービスを容易に開発するためのフレームワークを提供する．開発者は，提供されるフレームワークを用いることにより，次に挙げるような特徴を持った情報共有サービスを容易に構築することができる．

- 似たような情報を求めるユーザ間の情報共有を提供可能

  モバイルエージェントが持つ移動性と，本システムが提供するディレクトリサービスを利用することにより，自律的なエージェントが可能となる．さらに，エージェントがユーザのデータと好み (嗜好) をもつことにより，似たような目的を持つエージェントを自律的に探しだして情報共有を行うことができる．

- 不特定多数を対象とした情報共有サービスが提供可能

  不特定多数のユーザによって利用されることを想定した場合，システム全体の通信コスト，(ユーザの場所などの) ネットワークの動的変化，ユーザ毎の計算機環境の相違を考慮する必要がある．

  本システムは，モバイルエージェントを用いて構成されているため，移動による通信量の削減が可能，ネットワークの動的変化に対処可能，計算機環境に対する依存度が低い等，不特定多数のユーザで運用した場合に想定される環境への親和性が高い．

- 動的な負荷分散や配置変更が可能

  ディレクトリサービスとモバイルエージェントの移動性を生かして，動的な負荷分散や配置変更が可能である．また，移動による資源獲得と獲得した資源に応じた処理の変更が可能である．

- 柔軟で低コストのエージェント通信が可能

  本システムでは，モバイルエージェントによって構成されており，移動による通信コストの削減が可能である．しかし，通信のためにエージェントそのものが移動することは，非効率的な場合がある．本システムでは，メッセージ送受信専用の軽量モバイルエージェントを用いたエージェント間の通信 API を提供する．提供される API を用いることにより，柔軟なエージェント間通信が容易に可能となる．

4.3 情報共有システムの構成

本節では，本研究で提案する情報共有システムの構成について述べる．

本システムは，モバイルエージェントシステム上で動作する 4 種類のエージェント (Server Agent, Directory Agent, Service Agent, User Agent) によって構成され，これらのエージェントの移動と通信を中心とした作業によって情報共有を行う (図 4.1 参照)．

各エージェントの概要と，エージェント間の関係を図 4.2と図 4.3に示す．Server Agent は，各ホストにひとつ存在し，ホスト上のエージェントに対してエージェントの管理，計算機資源の割り当てなどを行う．さらに，Server Agent はホスト上の各エージェントにエージェント間通信を提供する．Directory Agent は，システム上の各エージェントにディレクトリサービスを提供する．Service Agent は，自ホストにいる User Agent へ情報共有サービスを提供する．また，ユーザに提供するサービス用のクライアントソフトウェアとして User Agent を配布する機能を持つ．User Agent は，ユーザのプロファイルとデータを保持しネットワーク上を移動する．Service Agent の提供するサービスを受けることによって，他の User Agent と情報共有を行う．

本節の以降で，それぞれのエージェントの機能と役割について述べる．

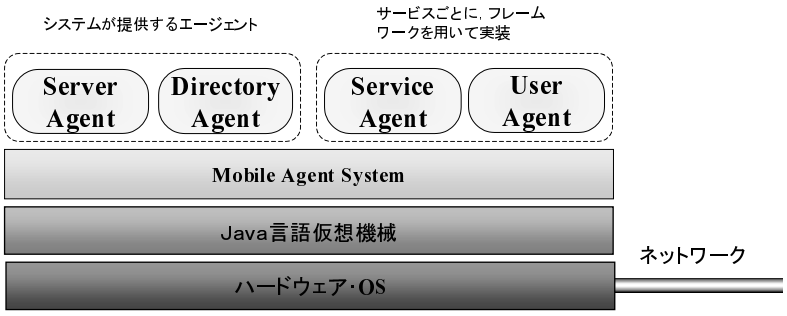


図 4.1: システム構成図

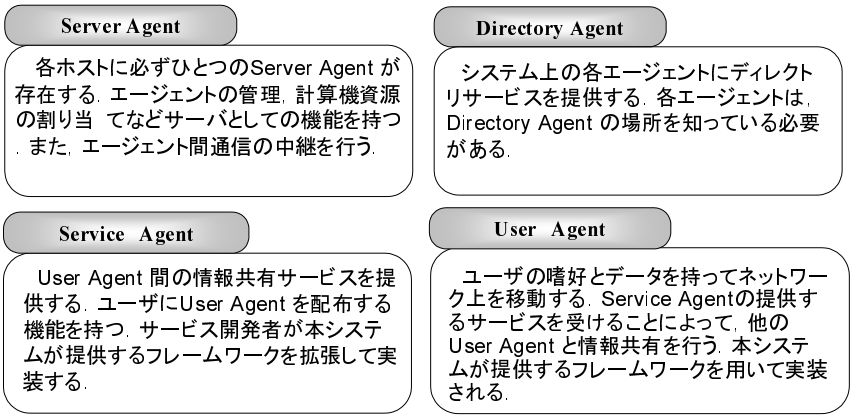


図 4.2: 各エージェントの概要

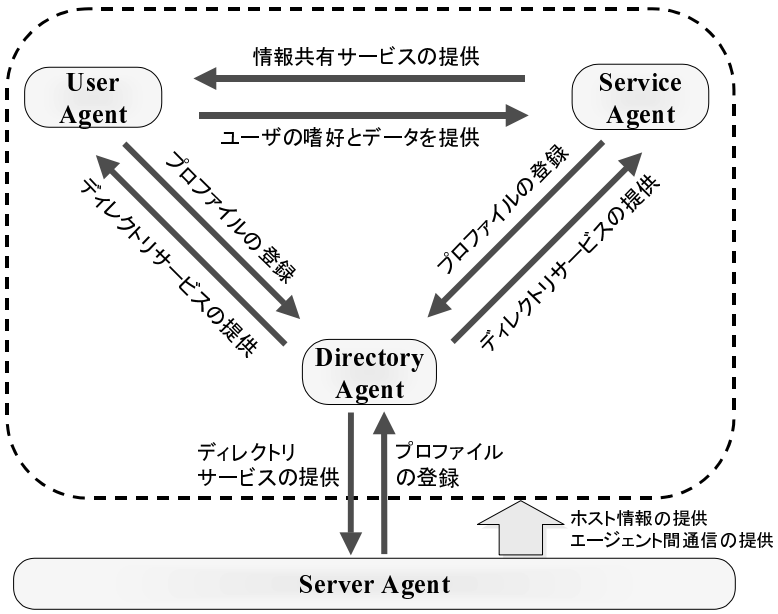


図 4.3: エージェント間の関係

4.3.1 Server Agent

Server Agent は，各ホストにひとつ存在し以下に挙げる機能を持つ．

- 情報共有システムへのユーザインタフェース

  ユーザへシステムにアクセスするためのユーザインタフェースを提供する．ユーザは，Server Agent が提供する GUI を用いることによって，サーバの設定や Directory Agent への問い合わせを行うことができる．

- モバイルエージェントシステムへのインタフェース

  エージェントとユーザへモバイルエージェントシステムの API を利用するための API を提供する．例としては，新たなエージェントをロード，ホスト上で活動するエージェント情報 (ID など) の入手などがある．

- 情報共有システムのサーバ

  ホスト上のエージェントの管理，計算機資源 (ハードウェアや許容できる負荷) の割当て設定など情報共有システムのサーバとしての機能を持つ．

  また，Directory Agent へ Server Agent の状態を登録することによって，他のエージェントは，ホストの状態を知ることができる．登録される情報は，4.4節にて述べる．

- エージェント間通信の中継

  異なるホストに存在するエージェント間のメッセージ通信を中継する．Server Agent を中継したエージェント間通信については，4.5節にて述べる．

4.3.2 Directory Agent

Directory Agent は，情報共有システム上の各エージェントにディレクトリサービスを提供する．

本システムでは，活動中のエージェントは一つ以上の Directory Agent に対して現在のホスト，活動状態などの情報を含んだプロファイルを登録する．本システム上のエージェントは，Directory Agent に対してエージェント名，ホストなどの条件を指定して問い合わせることにより，様々な情報を得ることができる．Directory Agent から得ることのできる情報の例を以下に挙げる．

- 現在提供されているサービスの一覧
- 特定のエージェントの現在位置と状態
- 特定のホストについての状態

ここで各エージェントは，Directory Agent に問い合わせることによって移動先のホストを決定するため，常に Directory Agent の場所を知っている必要がある．よって，最低でも一つの Directory Agent については，well-known なホストへの配置，あるいはブロードキャスト等によって容易に発見できるような工夫が必要である．

4.3.3 Service Agent

Service Agent は，ユーザの嗜好とデータを持った User Agent へ情報共有サービスを提供する．また，ユーザに提供するサービス用のクライアントソフトウェアとして User Agent を配布する機能を持つ．

本システムでは，共有するデータと共有方法によってさまざまな Service Agent が存在する．新たな情報共有サービスを提供したい開発者は，本システムが提供するフレームワーク (ServiceAgent クラス) を拡張することによって独自の情報共有サービスを開発することができる．さらに，開発した Servcie Agent を Direcotry Agent へ登録することによって情報共有サービスを公開することができる．

4.3.4 User Agent

User Agent は，ユーザの嗜好とデータを持ち，ネットワーク上を移動する．移動先 (Service Agent のいるホスト) で Service Agent の情報共有サービスを受けることにより他の User Agent と情報共有を行う．

サービスの開発者が，ユーザクライアントとして本システムが提供するフレームワーク (UserAgent クラス) を拡張して実装する．ユーザからの要求があった場合，Service Agent によってロード，移動されることによりユーザのホストにインストールされる．

4.4 エージェントプロファイル

本システム上のすべてのエージェントは，現在のホスト，活動状態，エージェントの持ち主などエージェントについての情報を含んだプロファイルを持ち，ひとつ以上の Directory Agent に対してプロファイルを登録する．

Directory Agent は，登録されたプロファイルを元にディレクトリを作成し，ユーザやエージェントに対してディレクトリサービスを提供する．本システムで活動するエージェントは，Directory Agent への問い合わせによって得た情報を元に，似たような情報を求める他のユーザのエージェントを自律的に探し出して情報共有を行う．

プロファイルは，すべてのエージェントに共通のプロパティと Directory Agent, Server Agent, Service Agent, User Agent それぞれ独自のプロパティから構成される．以後，Directory Agent がもつプロファイルを Directory Agent Profile とし，同様に他の種類のエージェントについても Server Agent Profile, Service Agent Profile, User Agent Profile とする．

各エージェント共通のプロパティを，表 4.1に示す．Agent Type は，エージェントの種類を表し “Directory Agent”, “Server Agent”, “Service Agent”, “User Agent” のいずれかである．Agent Identifier は，本システム上でエージェントを一意に特定するために用いられるため，他のエージェントの Agent Identifier と重複することはないとする．Directory Agent URL は，エージェントがプロファイルを登録している Directory Agent の場所 (URL) である．ここで，本システム上のエージェントは，必ずひとつ以上の Directory Agent に登録されている必要があるため，Directory Agent URL は，実在しアクセス可能な URL でなければならない．

表 4.1: すべてのエージェント共通のプロパティ

プロパティ	説明	値 (例)
Agent Type	エージェントのタイプ	User Agent
Agent Name	エージェントの名前	Bookmark Agent
Agent Version	エージェントのバージョン	20010215
Agent Identifier	他のエージェントと重複しない ID	50829337592
Current URL	現在活動中のホスト	http://150.65.193.211
Pre-URL	直前に活動していたホスト	http://150.65.194.53
Owner	エージェントの所有者	kkobaya@jaist.ac.jp
Owner URL	所有者のホスト	http://150.65.193.211
Directory Agent Name	エージェントが登録されている Directory Agent の名前	StanderdDirectoryAgent
Directory Agent URL	エージェントが登録されている Directory Agent のホスト	http://150.65.193.208
Birthday	生成された日時	Tue Nov 14:23:12:59
Birth-place	生成されたホスト	http://150.65.193.208

Server Agent Profile

Server Agent Profile は，表 4.1の共通プロパティと表 4.2に示す Server Agent 固有のプロパティから構成される．

Active Agnet Count は，現在同一ホスト上で活動中のエージェントの数である．Active Agnet Count には，Server Agent 自身も含まれるため，Active Agnet Count の最小値は 1 となる．

表 4.2: Server Agent 固有のプロパティ

プロパティ	説明	値 (例)
Active Agnet Count	ホスト上で活動中のエージェント数	3
Server Performance	ホストのコンピュータの性能 (very high, high, normal, low, very low のいずれか)	normal
Network Speed	ホストの通信回線速度 (very high, high, normal, low, very low のいずれか)	high
Accept Unknown Agent	知らないエージェントを受け入れるかどうかの設定	true

Directory Agent Profile

Directory Agent Profile は，表 4.1の共通プロパティと Directory Agent 固有のプロパティから構成される．

Directory Agent 固有のプロパティを表 4.3に示す．Agent Count は，ディレクトリに登録されているエージェントの総数を表し，Directory Agent Count, Server Agent Count, Service Agent Count, User Agent Count はそれぞれのエージェントの登録数を表す．それぞれの値は自然数であり，Agent Count は，Directory Agent Count, Server Agent Count, Service Agent Count, User Agent Count の和と等しい．

表 4.3: Directory Agent 固有のプロパティ

プロパティ	説明	値 (例)
Agent Count	登録されているエージェントの総数	72
Server Agent Count	登録されている Server Agent の数	36
Directory Agent Count	登録されている Directory Agent の数	1
Service Agent Count	登録されている Service Agent の数	5
User Agent Count	登録されている User Agent の数	30

Service Agent Profile

Service Agent Profile は，表 4.1の共通プロパティと表 4.4に示す Service Agent 固有のプロパティから構成される．

Service Abstract は，提供するサービスについての概要であり，ユーザが Directory Agent へ問い合わせたときなどに表示される．User Agent Name は，クライアントソフトウェアとしてユーザの元へ配布される User Agent の名前である．Sharing は，現在サービスを行っているかどうかの真偽値である．

表 4.4: Service Agent 固有のプロパティ

プロパティ	説明	値 (例)
Service Abstract	サービスの概要	ブックマーク共有サービス
User Agent Name	クライアントとなる User Agent の名前	Bookmark Agent
User Agent Count	現在サービスを受けている User Agent の数	3
Sharing	現在サービスを提供しているかどうか	true
Developer Name	エージェントの開発者の名前	kkobaya@jaist.ac.jp
Developer URL	エージェントの開発者のホスト	http://150.65.193.211

User Agent Profile

User Agent Profile は，表 4.1の共通プロパティと表 4.5に示す User Agent 固有のプロパティから構成される．

ここで，User Preference はユーザの嗜好を表す．嗜好の表現方法 (型) は，User Agent の実装に依存するが，User Preference はユーザの嗜好の概要を表すキーワードなど，比較的サイズの小さいデータし，詳細についてはエージェント自身に持たせることが推奨される．これは，プロファイルは定期的にエージェント間通信を用いて Directory Agent へ送信されるため，ネットワーク負荷を軽減する必要があるためである．同様の理由で，ユーザのデータは，プロファイルには含まれない．

表 4.5: User Agent 固有のプロパティ

プロパティ	説明	値 (例)
My ServerAgent Profile	サービスを受けている Server Agent のプロファイル	
Sharing	現在サービスを受けているかどうか	true
User Preference	ユーザの嗜好 (キーワードなど)	”Linux”
Distributed Time	ユーザへ配布された日時	Tue Nov 14:23:12:59
Developer Name	エージェントの開発者の名前	kkobaya@jaist.ac.jp
Developer URL	エージェントの開発者のホスト	http://150.65.193.211

4.5 エージェント間通信

本節では，本システムにおけるエージェント間通信について述べる．

一般的に，モバイルエージェントを用いたシステムでは，エージェントの移動によってネットワークを介してコンピュータ間で行われていた通信を，1 つのコンピュータ内のローカルな通信[1]に還元することができ，コンピュータ間通信の回数削減や通信遅延短縮が可能となる．

本システムは，モバイルエージェントを用いて構成されているため，これらのモバイルエージェントが持つ通信の利点を情報共有に活かすことができる．しかし，通信のためにエージェントそのものが移動することは，状況によっては非効率的な場合がある．

例えば，GUI を持ったエージェントがひとつのメッセージを伝えるためだけに (送信するメッセージには関係の無い)GUI をもって移動することは通信量の無駄である．よって，状況によって移動と移動を用いないメッセージ通信を使い分ける必要がある．また，できる限り柔軟で効率的な通信方式が利用できることが望ましい．

そこで，本システムでは，モバイルエージェントを用いたエージェント間通信 API を提供する．

[1]通常モバイルエージェントシステムの API を用いて行われる．

4.5.1 Delivery Agent を用いたエージェント間通信

本システムは， Server Agent と メッセージ送受信専用の軽量モバイルエージェント Delivery Agent を用いたエージェント間通信 API を開発者へ提供する．Delivery Agent を用いたエージェント間通信の流れを図 4.4に示す．

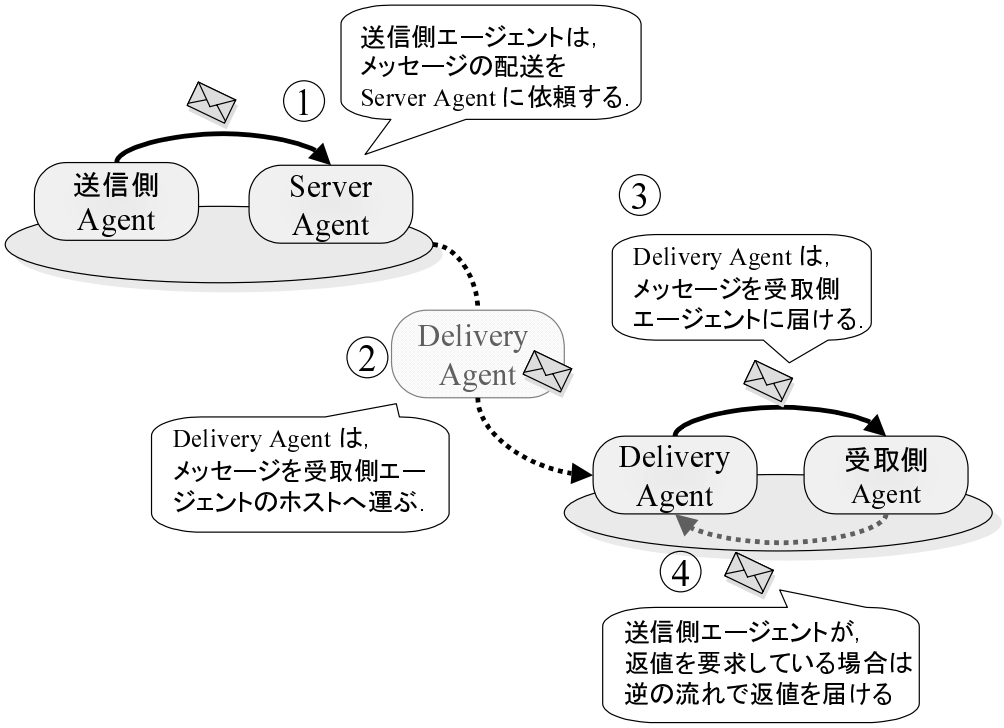


図 4.4: Delivery Agent を用いたエージェント間通信

本システムでは，Delivery Agent を用いたエージェント間通信を用いることで，次に挙げる利点が期待される．

- **自律的なメッセージが実現可能**

  Delivery Agent は，モバイルエージェントであり自律性を持たせることができる．よって，受取側エージェントが移動した場合に受信側エージェントを追跡してメッセージを渡すといった処理や，受信側エージェントが現れるまで特定のホストで待機するといった処理が可能であり，自律的なメッセージが実現できる．

- **通信回線の影響を受けにくい**

メッセージをモバイルエージェントを用いて配送するため，送信時と受信時に回線が接続されていれば通信可能である．よって，断続的に接続されるような環境においても通信可能となる．

- **開発者の記述量を軽減**

  モバイルエージェントを用いたエージェント間通信を API として提供することにより，送信側のエージェントは，メッセージを作成し API を呼び出すだけでエージェント間通信が可能となる．よって，開発者が，エージェント間通信を行うための記述量を軽減できる．また，エージェントの軽量化にもつながる．

4.5.2 通信の方式

本システムが提供するエージェント間通信 API には，非同期メッセージ通信 (send)，同期メソッド呼び出し (call)，非同期メソッド呼び出し (future)，コールバック呼び出し (callback)，の 4 種類の方式があり，それぞれ以下の特徴を持つ．

なお，プログラマは，これらの API を用いることにより Server Agent , Delivery Agent を意識することなくエージェント間通信を行うことができるため，各方式を説明する図では，Server Agent ,Delivery Agent を省略している．

非同期メッセージ通信 (send)

メソッド名と引数を指定したメッセージを作成し，エージェントが活動するホストと，名前またはエージェント Agent Identifier で指定した相手に送信する．受信側エージェントを名前で指定し，同名のエージェントが複数存在した場合は，該当するエージェント全員に送信するかを選択することができる．

受信側エージェントは，指定されたメソッド名，引数の組み合わせが存在し，アクセスできる場合に，メソッドを起動する．送信側エージェントは，送信後はブロックされないが，返値，例外を得ることは出来ない．

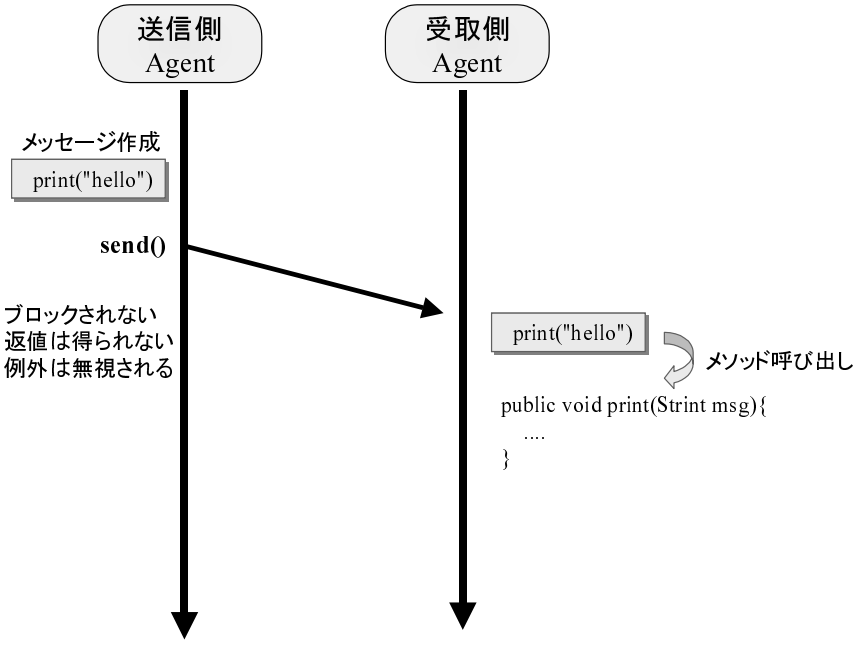


図 4.5: 非同期メッセージ通信 (send)

同期メソッド呼び出し (call)

メソッド名と引数を指定したメッセージを作成し，エージェントが活動するホストと，名前またはエージェント Agent Identifier で指定した相手に送信する．受信側エージェントは，指定されたメソッド名，引数の組み合わせが存在し，アクセスできる場合に，メソッドを起動する．

送信側エージェントは，受信側エージェントが起動したメソッドの返値を得ることができる．受信側エージェントが存在しなかった場合や，メソッドの中で例外が発生した場合は，送信側エージェントで例外が発生する．送信側エージェントは，返値を得るか例外が発生するまでブロックされる．

受信側エージェントを名前で指定し，同名のエージェントが複数存在した場合は，該当するエージェント全員に送信するかを選択することができる．全員に送信した場合，全員のメソッドの返信がリストとして返される．

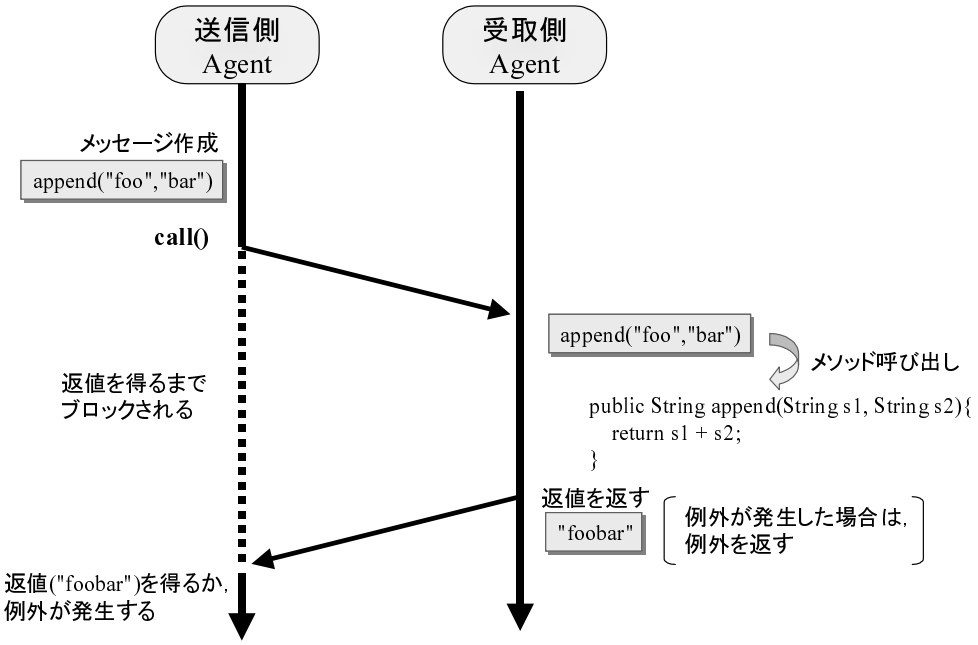


図 4.6: 同期メソッド呼び出し (call)

非同期メソッド呼び出し (future)

メソッド名と引数を指定したメッセージを作成し，エージェントが活動するホストと，名前またはエージェント Agent Identifier で指定した相手に送信する．受信側エージェントは，指定されたメソッド名，引数の組み合わせが存在し，アクセスできる場合に，メソッドを起動する．

送信側エージェントは，受信側エージェントが起動したメソッドの返値を getReply() を呼び出すことによって得ることができる．受信側エージェントが存在しなかった場合や，メソッドの中で例外が発生した場合は，getReply() で例外が発生する．送信側エージェントが getReply() を呼び出したときに，受信側エージェントからの返信，例外が届いてない場合は，返値を得るか例外が発生するまでブロックされる．

受信側エージェントを名前で指定し，同名のエージェントが複数存在した場合は，該当するエージェント全員に送信するかを選択することができる．全員に送信した場合，getReply() からは全員の返信がリストとして返される．

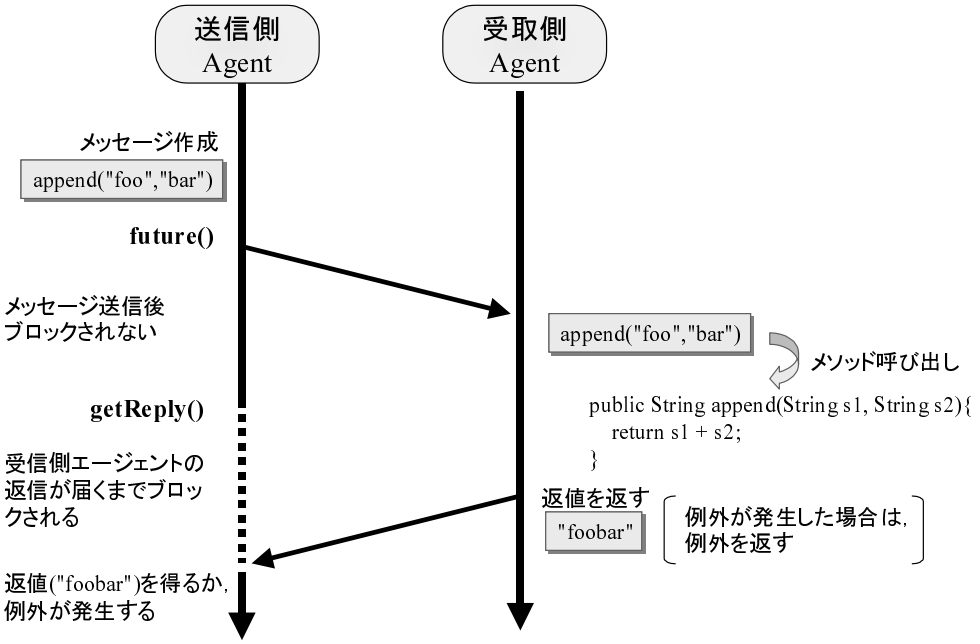


図 4.7: 非同期メソッド呼び出し (future)

コールバック呼び出し (callback)

メソッド名と引数を指定したメッセージと，コールバックするメソッド名と引数を指定したメッセージを作成し，エージェントが活動するホストと，名前またはエージェント Agent Identifier で指定した相手に送信する．送信側エージェントは，メッセージ送信後ブロックされない．

受信側エージェントは，指定されたメソッド名，引数の組み合わせが存在し，アクセスできる場合に，メソッドを起動する．受信側エージェントがメソッドを呼び出せた場合，送信側エージェントでコールバックメッセージで指定したメソッドを呼び出す．このとき，受信側エージェントが呼び出したメソッドの返値を最後の引数にセットする．

受信側エージェントが存在しない場合，呼び出したメソッドで例外が発生した場合でもエラーを返すことは無く，コールバックメソッドも起動されない．

受信側エージェントを名前で指定し，同名のエージェントが複数存在した場合は，該当するエージェント全員に送信するかを選択することができる．全員に送信した場合，それぞれがコールバックを行う．

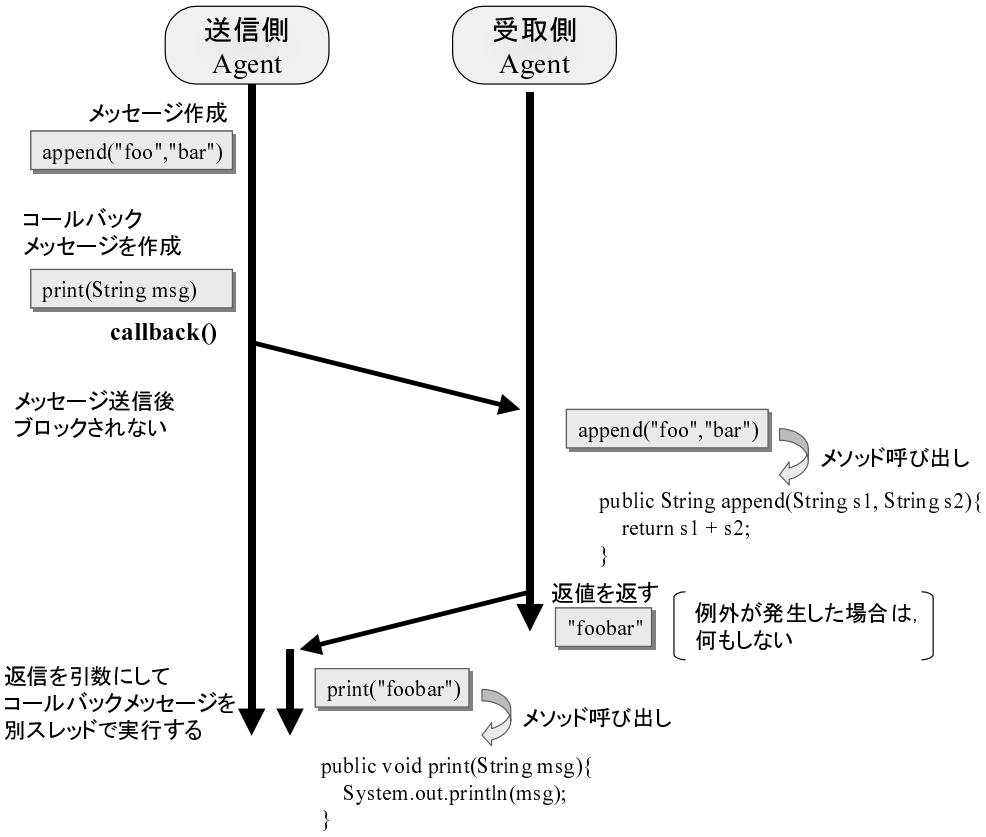


図 4.8: コールバック呼び出し (callback)

4.6 情報共有サービスの公開方法

本節では，開発者が独自の情報共有サービスを開発し，公開する手順について述べる．なお，第 6章にて，ブックマーク共有サービスを具体例に挙げ説明しているので，そちらも参照されたい．

1. 独自の Service Agent と User Agent の開発

   新たな情報共有サービスを提供したい開発者は，本システムが提供するフレームワーク (Service Agent と User Agent の雛型) を拡張して，独自の Service Agent と User Agent を開発する (図 4.9参照)．

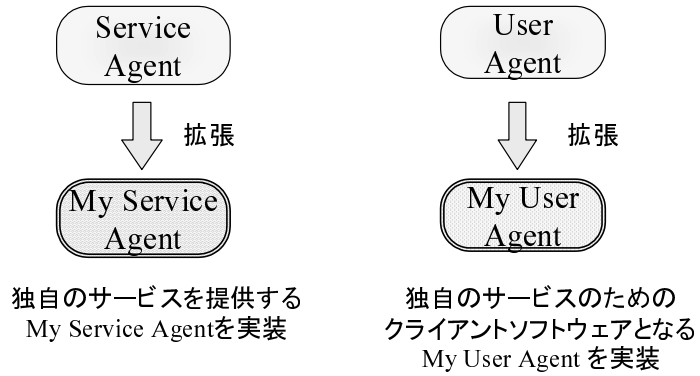


図 4.9: 独自のサービスの開発

2. Directory Agent への登録

   開発した独自の Service Agent を本システムでロードすることにより活動状態にし，Direcotry Agent へ Service Agent のプロファイルを登録する．プロファイルを Directory Agent へ登録することによって，開発者は，他のユーザとエージェントに対してサービスを公開することができる．なお，登録は通信コストに配慮して Delivery Agent を利用したエージェント間通信で行うことが推奨される．

4.7 情報共有サービスの利用方法

本節では，利用者 (一般ユーザ) が本システムを利用して情報共有を行うための手順について述べる．なお，以下の仮定が成立しているとする．

- いくつかの情報共有サービスがすでに公開されサービスを提供中である
- ユーザのホストには，本システムが起動され，Server Agent が活動している
- Directory Agent が活動中である

ユーザが，情報共有サービスの利用する方法を以下に述べる．

1. Directory Agent にサービスの一覧を問い合わせる

   ユーザは，Directory Agent に対して現在共有中のサービスの一覧を問い合わせる (図 4.10参照[1])．Directory Agent は，該当する Server Agent のプロファイルの一覧を返す．

   なお，Directory Agent への問い合わせを行うためのユーザーインターフェース (GUI) は Server Agent が提供する．

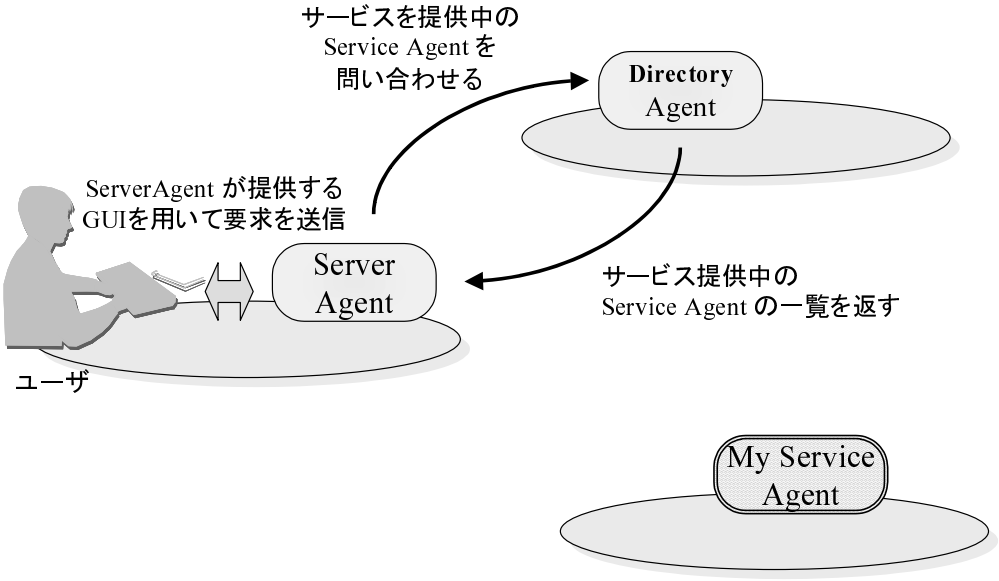


図 4.10: サービスの一覧の問い合わせ

[1] これらの問合せは，Delivery Agent を用いた通信によって行われる．しかし，本システムではユーザとエージェントは，エージェント間通信 API を用いることによって，Delivery Agent の存在を意識することなく異なるホストのエージェントと通信できる．よって，本節での図では全体の流れをわかりり易くするため省略している．

2. Service Agent に対して User Agent を要求する

   ユーザは，問い合わせの結果得られたサービスの中から受けたいサービスを選択し，選択したサービスを提供している Service Agent に対して，サービスのクライアントソフトウェアとなる User Agent を要求する．

   ユーザから User Agent を要求された Service Agent は，User Agent を活動状態にし，ユーザの元へ移動させる．このとき，必要に応じて初期化処理などを行う．User Agent がユーザの元へ移動することによってクライアントソフトウェアのインストールは完了する (図 4.11参照)．

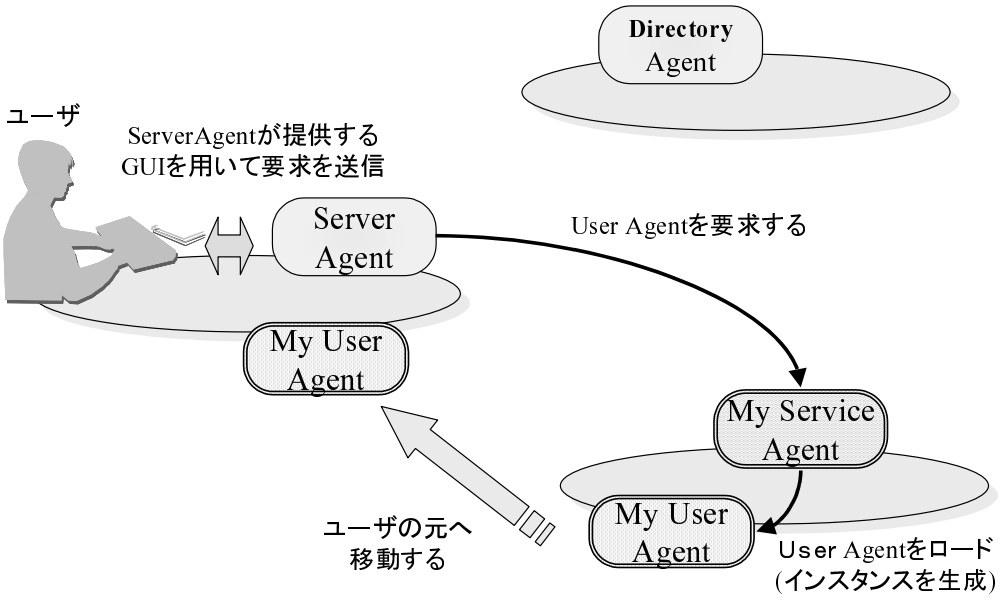


図 4.11: User Agent のインストール

3. User Agent にユーザのデータと嗜好を与える

   ユーザは，Service Agent によって送られた User Agent に対して，自身の共有するデータと嗜好を与える．ここで，嗜好とはユーザがどのような情報を求めているかを示す値であり，文字列，ファイル名など様々なものが考えられる．

   なお，本システムでは，嗜好をどのように定義するかは，サービス開発者 (Service Agent と User Agent の開発者) に任されている．

4. User Agent をネットワークに放つ

User Agent をネットワークに放つ．以後の作業は，User Agent によって自律的に行われるため，ユーザは何ら指示を与える必要は無い．また，回線状態を維持する必要は無く，オフラインにしてもかまわない．なお，User Agent の活動状況については，Server Agent を用いて Directory Agent に問い合わせることにより把握することができる．

User Agent が行う処理の流れは次のようになる．

(a) 似たような嗜好を持つ UserAgent を探す

User Agent は，ユーザと似たような嗜好を持つ User Agent を探す．これは，ユーザと似たような嗜好を持つ User Agent へサービスを提供している Service Agent を探すことによって実現される．

どのようにして Service Agent を探すのかは User Agent の実装に依存するが，本システムが提供するディレクトリサービスを用いることが推奨される．具体的な手法については，第 6 章にてブックマーク共有を例に述べる．

(b) Service Agent のサービスを受ける

User Agent は，(a) で得た Service Agent のホストへ移動する．Service Agent により提供されるサービスを受けることによって他の User Agent(ユーザ) と情報共有を行う．

(c) ユーザの元へ帰る

UserAgent は，ユーザの元へ帰り共有した情報をユーザに還元する．このとき，ユーザのホストへ移動できるかは，Directory Agent へ問い合わせることにより判断する．

第 5 章

本情報共有システムの実装

本章では，第 4 章で設計した情報共有システムを実装する．

5.1 実装環境

- OS : Windows 2000 Professional
- Java : Java2 SDK Standard Edition (JDK 1.2.2)
- モバイルエージェントシステム : AgentSpace 1 (2000 年 10 月 5 日版)

5.2 構成と概要

本システムは，以下のカテゴリのクラス群から構成される Java 言語のクラスライブラリとして実装されている．なお，Service Agent と User Agent は，サービス開発者が本システムが提供するフレームワーク (ServiceAgent クラス,UserAgent クラス) を拡張して実装する．

- 情報共有システムフレームワーク
  - エージェントプロフィールを実装したクラス
  - エージェント間通信のためのクラス
  - 各エージェントの雛型クラス
  - ユーティリティクラス

- Server Agent を構成するクラス
- Directory Agent を構成するクラス
- Delivery Agent を構成するクラス

5.3 AgentSpace の概要と仕組み

本節では，本システムがプラットフォームとして用いている AgentSpace について，その概要と仕組みを述べる．

AgentSpace は，モバイルエージェントを実現するためのエージェントランタイムシステムと，エージェント記述に有用なフレームワークの 2 つの部分から構成されている．ランタイムシステムは，エージェントの実行制御と移動を管理・制御し，フレームワークは，エージェントの移動，通信，停止などのエージェント記述に必要な API を提供する．

5.3.1 エージェントランタイムシステム

AgentSpace のエージェントランタイムシステムは，Java 仮想機械 (VM) 上で動作し，エージェントの起動，永続化，停止を管理する．また，他のエージェントランタイムシステムとのエージェント送受信を行うことができる．エージェントの移動は，エージェントランタイムシステムのエージェント送受信によって実現される．

エージェントランタイムシステムは，各エージェントのインタフェースとなるエージェントコンテキストと，エージェントの移動や永続化を管理するエージェントランタイムから構成されている (図 5.1参照)．

エージェントランタイムシステムの機能には以下のものがある．

- **エージェントの実行管理** :
  各エージェントの実行状態 (生成，永続化，活性化，停止など) の管理と制御を行う．
- **エージェントの移動** :
  エージェントを他のコンピュータへ転送する．また，到着したエージェントの実行再開を行う．
- **エージェント間通信** :
  ランタイム内のエージェント間通信の実現．エージェント間通信には，非同期メッセージ通信，同期メソッド呼び出し，フューチャー通信がある．

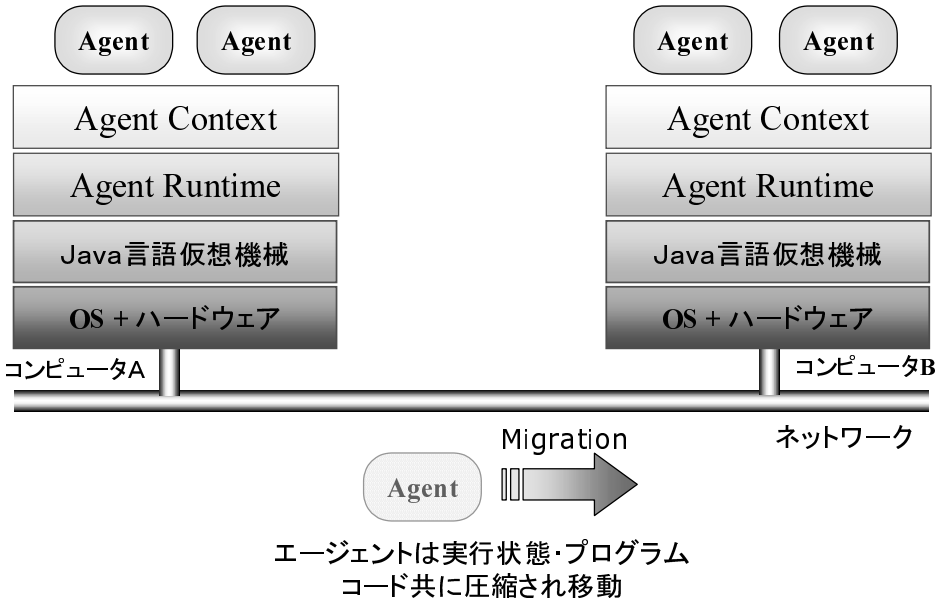


図 5.1: AgentSpace システム構成図

5.3.2 コールバックメソッド

AgentSpace におけるエージェントプログラムは，各エージェントの動作を記述する Java 言語のプログラムであり，クラス Agent のサブクラスとして定義される．Agent クラスは，表 5.1に示すコールバックメソッドから構成され，各メソッドはエージェントの状態変化の前後に呼び出される．エージェント開発者は，これら Agent クラスが持つメソッドを上書きして，エージェントの動作を記述する．

表 5.1: AgentSpace のコールバックメソッド

メソッド名	起動タイミング	概要
void init()	合成直後	クラスファイルからエージェントを合成するときに一度だけ呼び出される.
void create()	生成の直後	エージェント生成の直後に呼び出される. 初期化などはここに記述する.
void destroy()	停止の直前	エージェント停止の直前に呼び出される.
void dispatch(URL url)	移動の直前	エージェントが他のコンピュータに移動する直前に呼び出される. 引数 url には, 移動先の URL アドレスが入る.
void arrive()	移動の直後	エージェントが, 移動先のコンピュータに到着した直後に呼び出される.
void suspend()	永続化の直前	エージェントが永続化される直前に呼び出される.
void resume()	活性化の直後	エージェントが活性化された直後に呼び出される.
void duplicate()	複製の直前	エージェントの複製が生成される直前に呼び出される.
void parent(AgentIdentifier aid)	複製の直後	エージェントの複製が生成された直後に, 複製元のエージェントで呼び出される. 引数 aid には, 複製によって生成されたエージェントの識別子が入る.
void child(AgentIdentifier aid)	複製の直後	エージェントの複製が生成された直後に, 複製によって新たに生成されたエージェントで呼び出される. 引数 aid には, 複製元となったエージェントの識別子が入る.

5.3.3 エージェント間通信

AgentSpace では，セキュリティに配慮するため，エージェントが別のエージェントを直接参照することはできない．そこで，エージェント間で通信する必要があるときには AgentSpace が提供する AgentContext クラスの API を用いて通信を行う．本節では，AgentSpace が提供するエージェント間通信について解説する．

なお，以下で述べる AgentSpace が提供するエージェント間通信 API は，同一ホスト内のエージェントとしか通信ができない．これは，AgentSpace では，他のホストにいるエージェントとは，移動後をした後に，ローカル通信を行うべきであるという設計思想になったいるためである．

AgentContext が提供するエージェント間通信用 API

AgentContext クラスによって提供されるエージェント間通信には，非同期メッセージ通信，同期メソッド呼び出し，フューチャー通信がある．

送信するメッセージは，Message クラスによって指定する．Message クラスのコンストラクタで受信側エージェントが呼び出すメソッドの名前を指定し，Message クラスの setArg(Object arg) でメソッドの引数を指定する．

例えば，

```
Message msg = new Message("append");
msg.setArg("foo");
msg.setArg("bar");
```

のように作成したメッセージ msg を送信した場合，受信側エージェントでは，メソッド append(”foo”,”bar”) が呼び出される．

それぞれの通信方式の，機能，エラー時の処理を以下で述べる．

- 非同期メッセージ通信

  メソッド名: void send(AgentIdentifier aid, Message msg)

  識別子 aid をもつエージェントで Message クラスのインスタンス msg で与えられたメッソドを呼び出す．送信側エージェントは，メソッドの返値を受け取ることはできないが，受信側エージェントの状態に関わらず処理を継続することができる．

  ただし，送信した時点で受信側エージェント自体及び対応するメソッドが存在しない場合，そして引数の数や型が一致しない場合でもエラーを返すことはない．

- 同期メソッド呼び出し

  メソッド名: Object call(AgentIdentifier aid, Message msg)

識別子 aid をもつエージェントで Message クラスのインスタンス msg で与えられたメッソドを呼び出す．返値はメソッドの返値となる．このとき，送信側エージェントは受信側エージェントが結果を返すまでブロックされる．

ただし，送信した時点で，受信側エージェント自体及び対応するメソッドが存在しない場合，そして引数の数や型が一致しない場合はエラーになる．

- フューチャー通信
  メソッド名: void future(AgentIdentifier aid, Message msg)

  識別子 aid をもつエージェントで Message クラスのインスタンス msg で与えられたメッソドを呼び出す．送信側エージェントは，メソッドの返値を受け取ることはできないが，受信側エージェントの状態に関わらず処理を継続することができる．送信側エージェントは，Future クラスの getReply() メソッドを通じて返値を得ることができる．

  ここで，送信側エージェントが getReply() を呼び出した時点で，受信側エージェントが返値を返していない場合，受信側エージェントが返値を返すまで送信側エージェントはブロックされる．

  なお，送信した時点で受信側エージェント自体及び対応するメソッドが存在しない場合，または引数の数や型が一致しない場合でもエラーを返すことはない．

5.4 エージェントプロファイルの実装

各エージェントのプロファイルは図 5.2に示す継承関係で実装される．AgentProfile クラスは，各エージェント共通のプロパティをフィールドに持ち，各プロパティへの操作を実装している．また，AgentProfile クラスを継承している，ServerAgentProfile, DirectoryAgentProfile, ServiceAgentProfile, UserAgentProfile クラスは，各エージェント固有のプロパティとそれに関する操作を実装している．

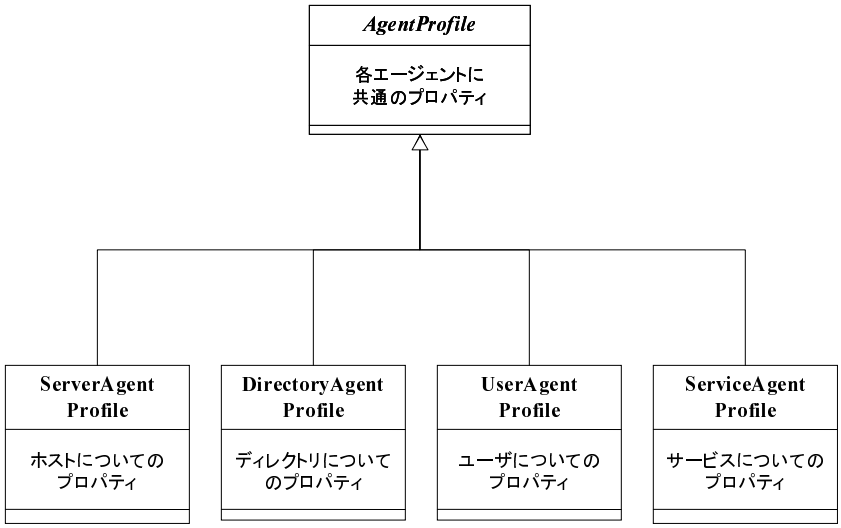


図 5.2: エージェントプロファイルの実装

5.5 エージェント間通信 API の実装

本節では，エージェント間通信 API に関連したクラスについて述べる．

本システムにおけるエージェント間通信 API は，API 内部で AgentMail クラスのインスタンスを作成し，Server Agent へ送信を依頼する．メッセージの送信を依頼された Server Agent は，Delivery Agent を活動状態にし，メールを受信側エージェントのホストへの移動させることで，メッセージを配送する．以下では，これらの処理に関連したクラスについて述べる．

5.5.1 AgentMail クラス

AgentMail クラスには，送信側エージェント，受信側エージェント，届けるメッセージなど，エージェント間で通信を行うための値を持ち，Delivery Agent は，AgentMail クラスのインスタンスが持つ値を元に，メッセージを配送する．なお，受信側エージェントで呼び出されるメソッドの指定には，5.3.3節で述べた AgentSpace が提供する Message クラスのインスタンスを用いている．

AgentMail クラスの概要を表 5.2に示す．

表 5.2: AgentMail クラスの概要

クラスの概要
エージェント間通信を行うための情報を格納したクラス.
主なフィールドの概要
private Message message 受信側エージェントへ届けるメッセージ.
private Object reply メッセージに対する受信側エージェントの返値.
主なメソッドの概要
public void setSender(URL url, AgentIdentifier aid, String name) 送信側エージェントについての情報を設定する.
public void setReceiver(URL url, AgentIdentifier aid, String name) 受信側エージェントを指定する.
public void setMessage(Message msg) 受信側エージェントへ届けるメッセージを設定する.
public void setReplyObject(Object reply) 送信側エージェントへ届ける返値を設定する.
public Message getMessage() 受信側エージェントへ届けるメッセージを返す.
public void setRequestReply(boolean request) 送信側エージェントが，メッセージの返信を要求しているかを設定する.
public boolean requestReply() 送信側エージェントが，メッセージの返信を要求しているかを返す.
public void setBroadcast(boolean value) 受取側エージェントの候補が複数存在した場合に，全員に送るかを設定する.
public boolean isBroadcast() 受取側エージェントの候補が複数存在した場合に，全員に送るかを返す.

5.5.2 Delivery Agent の実装

Delivery Agent は，DeliveryAgent クラスに実装される．DeliveryAgent クラスの概要を表 5.3に示す．送信側エージェントが返値を要求している場合は，deliverToRecieverAgent() メソッドの中で受信側エージェントからメッセージの返値を受取り，送信側エージェントの元へ移動した後，returnReplyObject() メソッドで送信側エージェントへ返値を渡す．

表 5.3: DeliveryAgent クラスの概要

クラスの概要
Delivery Agent を実装したクラス
主なフィールドの概要
private AgentMail mail 受信側エージェントへ届けるメッセージ
主なメソッドの概要
public void deliver(AgentMail mail) メッセージ mail を受信側エージェントのホストへ運ぶ． 受信側エージェントのホストは，mail から取得する．
private void deliverToRecieverAgent() メッセージ mail を受信側エージェントへ渡す． このメソッドは，受信側エージェントのホストで呼び出される．
private void returnReplyObject() 受信側エージェントからの返値を，送信側エージェントへ届ける． このメソッドは，送信側エージェントのホストで呼び出される．

5.5.3 エージェント間通信 API

本節では，本システムが提供するエージェント間通信 API について述べる．エージェント開発者は，本節で述べる API により Server Agent, Delivery Agent を意識することなくエージェント間通信を行うことができる．

なお，以下で述べる各 API は，後述する CommunicatableAgent クラスに実装される．

非同期エージェント間通信

```
void send(Message message)
void send(String name, Message message, boolean broadcast)
void send(String name, URL url, Message message, boolean broadcast)
```

```
void send(AgentIdentifier aid, Message message)
void send(AgentIdentifier adi, URL url, Message message)
```

名前または，Agent Identifier で受信側エージェントを指定し，message を送信する．受信側エージェントが活動するホストの指定を省略した場合は，送信側エージェントと同一ホストを指定したとする．受信側エージェントを指定しなかった場合自分自身に送信する．

名前で指定した場合，同名のエージェントが複数存在したときにメッセージを全員に送るかを broadcast で指定する．受信側エージェントが存在しない場合，message で指定したメソッドが存在しない場合，受信側エージェントで例外が発生した場合などのエラー時にもエラー (例外) を返すことはない．

同期メソッド呼び出し

```
Object call(String name, Message message, boolean broadcast)
Object call(String name, URL url, Message message, boolean broadcast)
Object call(AgentIdentifier aid, Message message)
Object call(AgentIdentifier adi, URL url, Message message)
```

エージェントが活動するホストと，名前またはエージェント Agent Identifier で受信側エージェントを指定し，message を送信する．受信側エージェントが活動するホストの指定を省略した場合は，送信側エージェントと同一ホストを指定したとする．受信側エージェントが起動したメソッドの返値を返す．

送信側エージェントは受信側エージェントが結果を返すまでブロックされる．受信側エージェントを名前で指定した場合，同名のエージェントが複数存在したときにメッセージを全員に送るかを broadcast で指定する．全員に送った場合，それぞれのエージェントの返信を Vector に格納して返す．

非同期メソッド呼び出し

```
void future(String name, Message message, boolean broadcast)
void future(String name, URL url, Message message, boolean broadcast)
void future(AgentIdentifier aid, Message message)
void future(AgentIdentifier adi, URL url, Message message)
```

エージェントが活動するホストと，名前またはエージェント Agent Identifier で受信側エージェントを指定し，message を送信する．受信側エージェントが活動するホストの指定を省略した場合は，送信側エージェントと同一ホストを指定したとする．送信側エージェントは getReply を呼び出すことで返値を得ることができる．

受信側エージェントを名前で指定した場合，同名のエージェントが複数存在したときにメッセージを全員に送るかを broadcast で指定する．全員に送った場合，getReply() はそれぞれのエージェントの返信を Vector に格納して返す．

コールバック呼び出し

```
void callback(Message callback, String name, Message message, boolean broadcast)
void callback(Message callback, String name, URL url, Message message)
void callback(Message callback, AgentIdentifier aid, Message message)
void callback(Message callback, AgentIdentifier adi, URL url, Message message)
```

エージェントが活動するホストと，名前またはエージェント Agent Identifier で受信側エージェントを指定し，message を送信する．受信側エージェントが活動するホストの指定を省略した場合は，送信側エージェントと同一ホストを指定したとする．メッセージ送信後，送信側エージェントでは，callback で指定したメソッドが受信側エージェントからの返値を最後の引数にして呼び出される．

受信側エージェントを名前で指定した場合，同名のエージェントが複数存在したときにメッセージを全員に送るかを broadcast で指定する．全員に送った場合，それぞれのエージェントから callback で指定したメソッドが呼び出される．

5.6 各エージェントの実装

本節では，本システムにおける各エージェントの実装について述べる．本システムにおける各エージェントは，Java 言語のクラスとして実装され，図 5.3に示す継承関係を持つ．

すべてのエージェントは，AgentSpace 上で振舞うため AgentSpace が提供する Agent クラスを継承する．CommunicatableAgent クラスは，本システムが提供するエージェント間通信 API を実装する．ここで，DirectoryAgent クラス, ServiceAgent クラス, UserAgent クラスは，抽象クラスであり，ユーザが独自のエージェントを作成するためのフレームワークとして提供される．

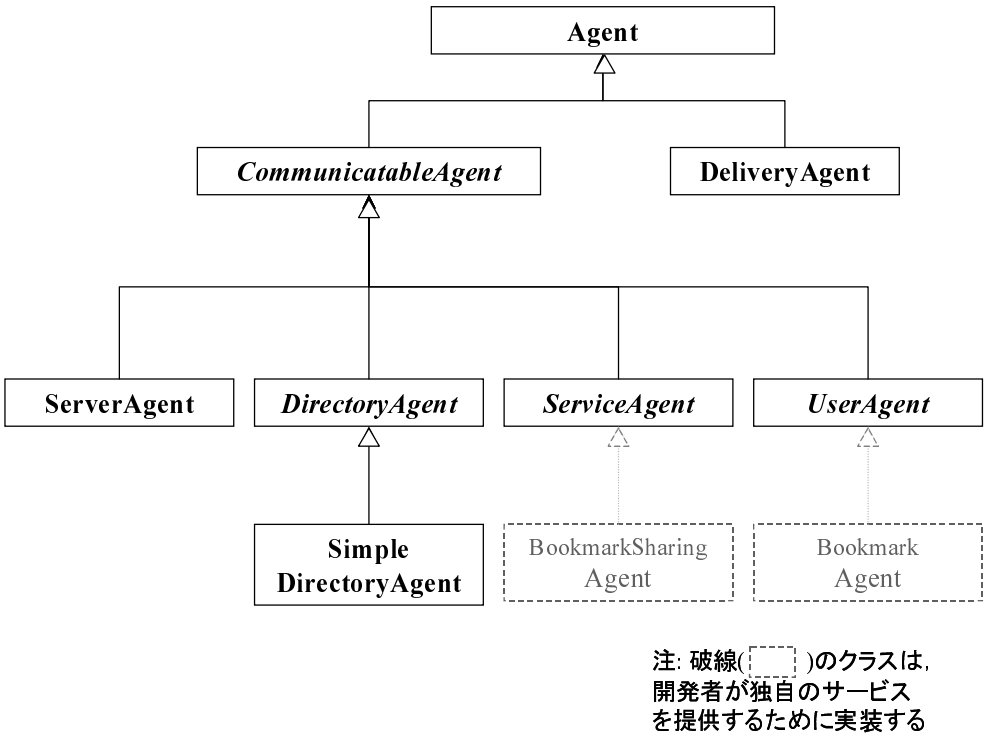


図 5.3: 各エージェントの継承関係

5.6.1 CommunicatableAgent クラス

CommunicatableAgent クラスは，本システムが提供するエージェント間通信 API を実装する．なお，本システムにおける異なるホスト間のエージェント間通信は，Server Agent , Delivery Agent を用いて行われるが，CommunicatableAgent クラスによって提供される API を用いることによって，これらのエージェントを意識することなくエージェント間通信を行うことができる．

例えば，CommunicatableAgent クラスを継承しているエージェントが，ホスト url にいる Agent Identifier aid のエージェントからの append(“foo”,”bar”) の返値を得たい場合は次のように記述する．

```
Message msg = new Message("append");  //メッセージを作成
msg.setArg("foo");                    //引数を指定
msg.setArg("bar");

try{
    //受取側エージェントを指定してメッセージを送信
    String result =  (String) call(aid,url,msg);
}catch(Exception e){
    //相手ホストで例外が発生した場合は，例外を出力
    System.err.println("caught Exception "+e.getMessage());
}
```

この記述では，受信側エージェントが存在しなかった場合，または，受信側エージェントがメソッドを実行した際に例外が発生した場合に，例外が捕捉され，例外のメッセージが出力される．

CommunicatableAgent クラスの概要を表 5.4に示す．

表 5.4: CommunicatableAgent クラスの概要

クラスの概要
エージェント間通信 API を実装したクラス.
主なフィールドの概要
public static final String SERVER_AGENT_NAME Server Agent の名前.
主なメソッドの概要
public void send(Message message) public void send(String name, Message message, boolean broadcast) public void send(String name, URL url, Message message, boolean broadcast) public void send(AgentIdentifier aid, Message message) public void send(AgentIdentifier adi, URL url, Message message) 非同期エージェント間通信. 受信側エージェントの実行したメソッドの返値は得ることができない.
public Object call(Message message) public Object call(String name, Message message, boolean broadcast) public Object call(String name, URL url, Message message, boolean broadcast) public Object call(AgentIdentifier aid, Message message) public Object call(AgentIdentifier adi, URL url, Message message) 同期エージェント間通信. 受信側エージェントの実行したメソッドの返値を得ることができる.
public void future(Message message) public void future(String name, Message message, boolean broadcast) public void future(String name, URL url, Message message, boolean broadcast) public void future(AgentIdentifier aid, Message message) public void future(AgentIdentifier adi, URL url, Message message) public Object getReply() 非同期エージェント間通信. エージェントは getReply を呼び出すことで返値を得ることができる.
public void callback(Message callback, Message message) public void callback(Message callback, String name, Message message, boolean broadcast) public void callback(Message callback, String name, URL url, Message message, boolean broadcast) public void callback(Message callback, AgentIdentifier aid, Message message) public void callback(Message callback, AgentIdentifier adi, URL url, Message message) コールバックメソッド呼び出し. メッセージ送信後, 送信側エージェントでは, callback で指定したメソッドが, 受信側エージェントからの返値を引数の最後にセットしてして呼び出される.

5.6.2 ServerAgent クラス

Server Agent は，各ホストのサーバとして活動し，ユーザインタフェース (GUI) の提供，エージェント間通信の中継といった機能を持つ．ここで，Server Agent は，エージェントのロード，エージェントの状態の監視など AgentSpace 上の通常のエージェントでは利用できない API を利用する．また，本システムでは各ホストに必ず Server Agent が存在していなければならないため，Server Agent は抽象クラスではなく具象クラスとして提供する．

ServerAgent クラスの概要を表 5.5に示す.

表 5.5: ServerAgent クラスの概要

クラスの概要
Server Agent を実装したクラス
主なフィールドの概要
protected ServerAgentProfile profile プロファイル
主なメソッドの概要
public void remoteSend(AgentMail mail) Delivery Agent をロードし， 受信側エージェントのホストへ移動させる.
public void remoteCall(Future futureObj,AgentMail mail) Delivery Agent をロードし， 受信側エージェントのホストへ移動させる.
public void remoteFuture(Future futureObj,AgentMail mail) Delivery Agent をロードし， 受信側エージェントのホストへ移動させる.
public void remoteCallback(Message callbackMessage, AgentMail mail) Delivery Agent をロードし， 受信側エージェントのホストへ移動させる.
public void requestUserAgent(ServiceAgentProfile profile) profile を持つ Service Agent へ User Agent を要求する.
public void requestOwnAgentList() Directory Agent へユーザが所有者となっているエージェントのプロファイルを問い合わせる．問い合わせ結果は，ServerAgent の GUI へ反映される.
public void requestServiceAgentList() Directory Agent へ 公開されている Service Agent の一覧を問い合わせる．問い合わせ結果は，ServerAgent の GUI へ反映される.
public void report() プロファイル (サーバの状態) を Directory Agent へ profile を送る.

ServerAgent の実行画面を図 5.4に示す．ユーザは，ServerAgent の GUI を用いて Directory Agent への問合せ，Service Agent への User Agent の要求などを行うことができる．

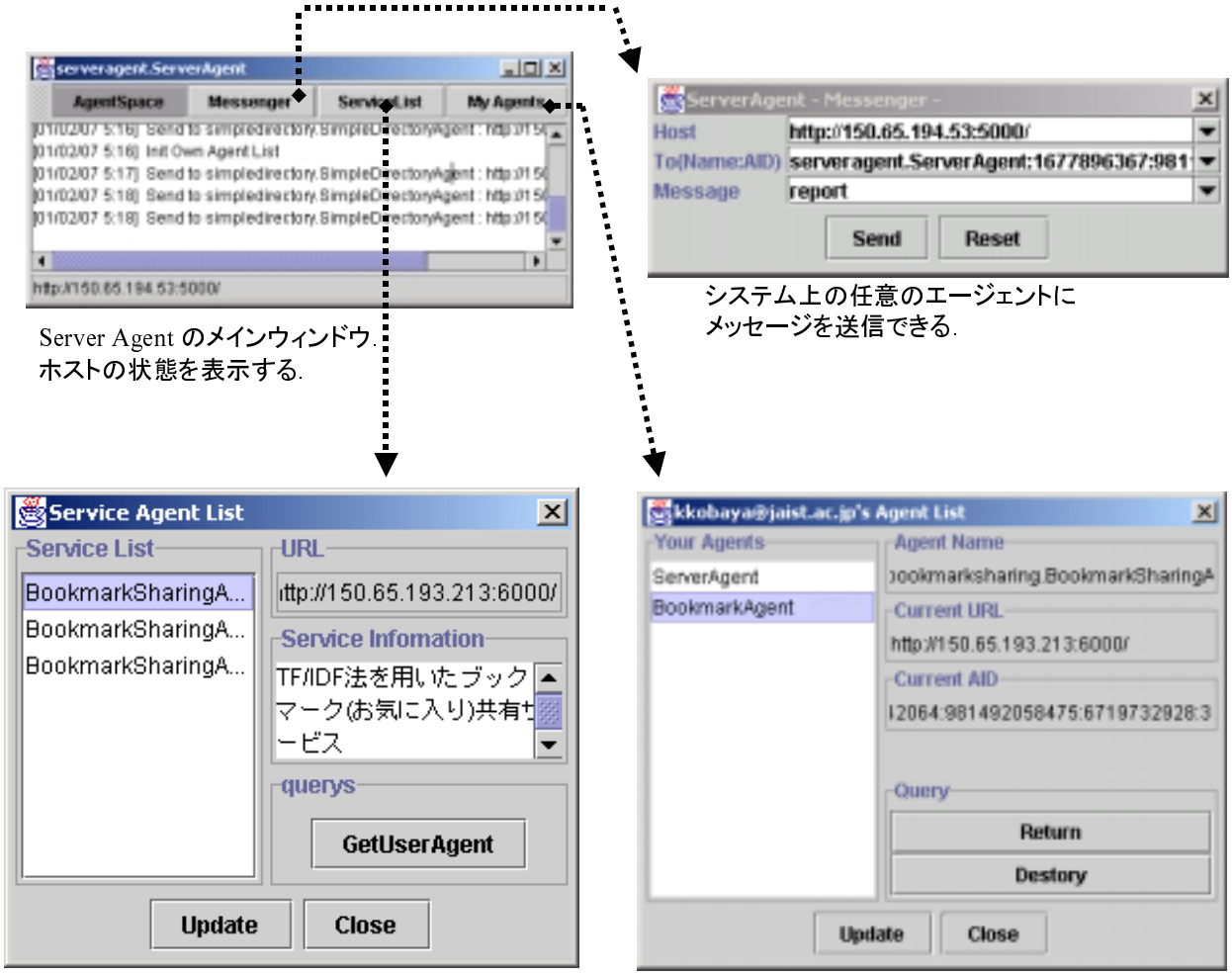


図 5.4: Server Agent の実行画面

5.6.3 DirectoryAgent クラス

DirectoryAgent クラスの概要を表5.6に示す．Directory Agent は，本システム上のエージェントにディレクトリサービスを提供するため，DirectoryAgent クラスの主なメソッドは，ディレクトリへの問い合わせである．また，サービス開発者は，本システムが提供する DirectoryAgent クラスのサブクラスとして独自の Directory Agent クラスを実装することにより，独自の Directory Agent を実装することができる．

なお，本システムでは，最低ひとつの Directory Agent が存在していなければならないため，SimpleDirectoryAgent を Directory Agent として実装している．

表 5.6: DirectoryAgent クラスの概要

クラスの概要
Directory Agent の雛型クラス
主なフィールドの概要
protected DirectoryAgentProfile profile プロファイル
主なメソッドの概要
public abstract Vector select(String name) 名前が name であるエージェントの一覧を返す
public abstract Vector select(URL url) ホスト url に存在する，エージェントの一覧を返す
public abstract Vector select(Class profileClass) プロファイルとして profileClass クラスを持つエージェントの一覧を返す．profileClass は，AgentProfile クラスまたは，AgentProfile のサブクラスとする．例えば，profileClass として ServiceAgentProfile クラスを指定すると，Service Agent の一覧を返す．
public Vector ownAgents(String owner, URL url) 引数で指定した所有者が持つエージェントのプロファイル一覧を返す
public URL where(AgentIdentifier aid) 引数で指定したエージェントが現在活動する URL を返す．該当するエージェントがいなかった場合は，null を返す．
public abstract void insert(AgentProfile profile) エージェントのプロファイル profile をディレクトリに登録する
public abstract void delete(AgentProfile profile) エージェントのプロファイル profile をディレクトリから削除する
public abstract void update(AgentProfile profile) エージェントのプロファイル profile を更新する
public abstract void report() エージェント自身が登録されている，Directory Agent へ profile を送る．

Simple Directory Agent

本システムでは，ひとつ以上の Directory Agent がシステム上のエージェントにディレクトリサービスを提供していなければならない．よって，本研究では，Simple Directory Agent をデフォルトの Directory Agent として提供する (図 5.5参照)．Simple Directory Agent は，抽象クラスである DirectoryAgent クラスを実装し GUI を付加したものである．

ここで，複雑な条件を指定しての検索などの機能を持った独自の Directory Agent を用意したいユーザは，DirectoryAgent クラスを継承した独自の Directory Agent を実装することができる．

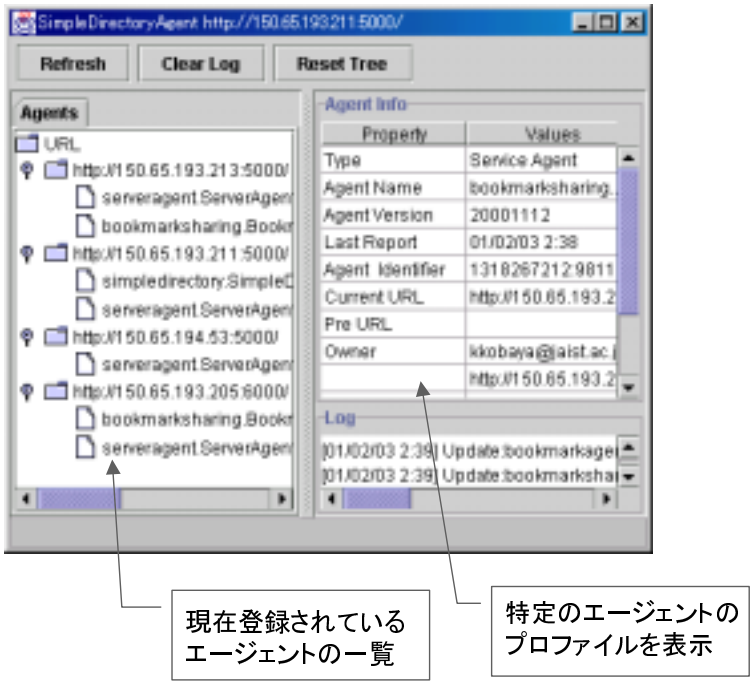


図 5.5: Simple Directory Agent の実行画面

5.6.4 ServiceAgent クラス

本システムでは，Service Agent は，CommunicatableAgent クラスを継承した抽象クラスとして定義される．サービス開発者は，本システムが提供する ServiceAgent クラスのサブクラスとして独自サービスを実装する．ServiceAgent クラスの概要を表 5.7に示す．

ここで，match メソッドは，引数で指定されたプロファイルを持ったエージェントが提供しているサービスにどれぐらい適合しているかを返す．User Agent は，同じサービス

を提供している Service Agent が複数存在した場合，match メソッドを用いてユーザの条件に合う Service Agent を選択する．

表 5.7: ServiceAgent クラスの概要

クラスの概要
Service Agent の雛型クラス
主なフィールドの概要
protected ServiceAgentProfile profile プロファイル
主なメソッドの概要
public abstract void startService() サービスの提供を開始する
public abstract void stopService() サービスの提供を停止する
public abstract Integer match(UserAgentProfile profile) profile を持った User Agent と提供しているサービスとの適合度を返す
public abstract Boolean addMember(AgentContext ac, UserAgentProfile profile) profile を持った User Agent をメンバーに登録する．登録に成功したら真を，失敗したら偽を返す.
public abstract Boolean removeMember(AgentContext ac, UserAgentProfile profile) profile を持った User Agent をメンバーから削除する．削除に成功したら真を，失敗したら偽を返す.
public abstract Vector listMember() 共有中のエージェントのプロファイル一覧を返す.
public abstract void dispatchUserAgent(URL url) UserAgent をロードし,url へ移動させる
public abstract void report() Directory Agent へ自身の profile を送る.

5.6.5 UserAgent クラス

本システムでは，User Agent は，CommunicatableAgent クラスを継承した抽象クラスとして定義される．サービス開発者は，本システムが提供する UserAgent クラスのサブクラスとして独自 User Agent を実装する．UserAgent クラスの概要を表 5.8に示す．

表 5.8: UserAgent クラスの概要

クラスの概要
User Agent の雛型クラス
主なフィールドの概要
protected UserAgentProfile profile プロファイル
protected ServerAgentProfile myServiceAgentProfile エージェントがサービスを受ける Service Agent のプロファイル．
主なメソッドの概要
public void goHome() UserAgent を所有しているユーザのホストへ移動する
public void setUserPreference(Object preference) ユーザの嗜好を設定する．
public Object getUserPreference() ユーザの嗜好を返す．
public void setServiceAgentProfile(ServiceAgentProfile profile) このエージェントがサービスを受ける Service Agent のプロファイルを指定する．profile は，Directory Agent に問い合わせるか，他のエージェントからエージェント間通信で受け取る．
public ServiceAgentProfile getServiceAgentProfile() このエージェントがサービスを受ける Service Agent のプロファイルを返す．
public abstract void report() Directory Agent へ自身の profile を送る．

第 6 章

例題:ブックマーク共有システム

本章では，本研究で提案した情報共有システムの例題としてブックマーク共有システムを実装し，考察する．

6.1 ブックマークとは

本研究におけるにおけるブックマーク[1] を次のように定義する．

ユーザが興味がある Web ページと判断した URL

ここで，ブックマークは，ユーザの嗜好を強く表しているという特徴を持つ．よって，似たようなブックマークを持つエージェント同士でブックマーク共有することにより似たような情報を求めるユーザ間の情報共有が可能である [25]．さらに，比較的サイズが小さいためエージェントがユーザのブックマークを持ちネットワーク上を移動するのは現実的であり，本システムの実装例題に適しているといえる．

6.2 システムの概要

本章で実装するブックマーク共有システムは，同じような興味分野を持つ不特定多数のユーザ間のブックマーク共有を目的としたシステムであり，本研究で提案した情報共有システム上のサービスとして構築する．

ブックマーク共有システムの概要を図 6.1に示す．

[1]ブラウザによっては,「お気に入り」,「ホットリスト」と呼ばれる

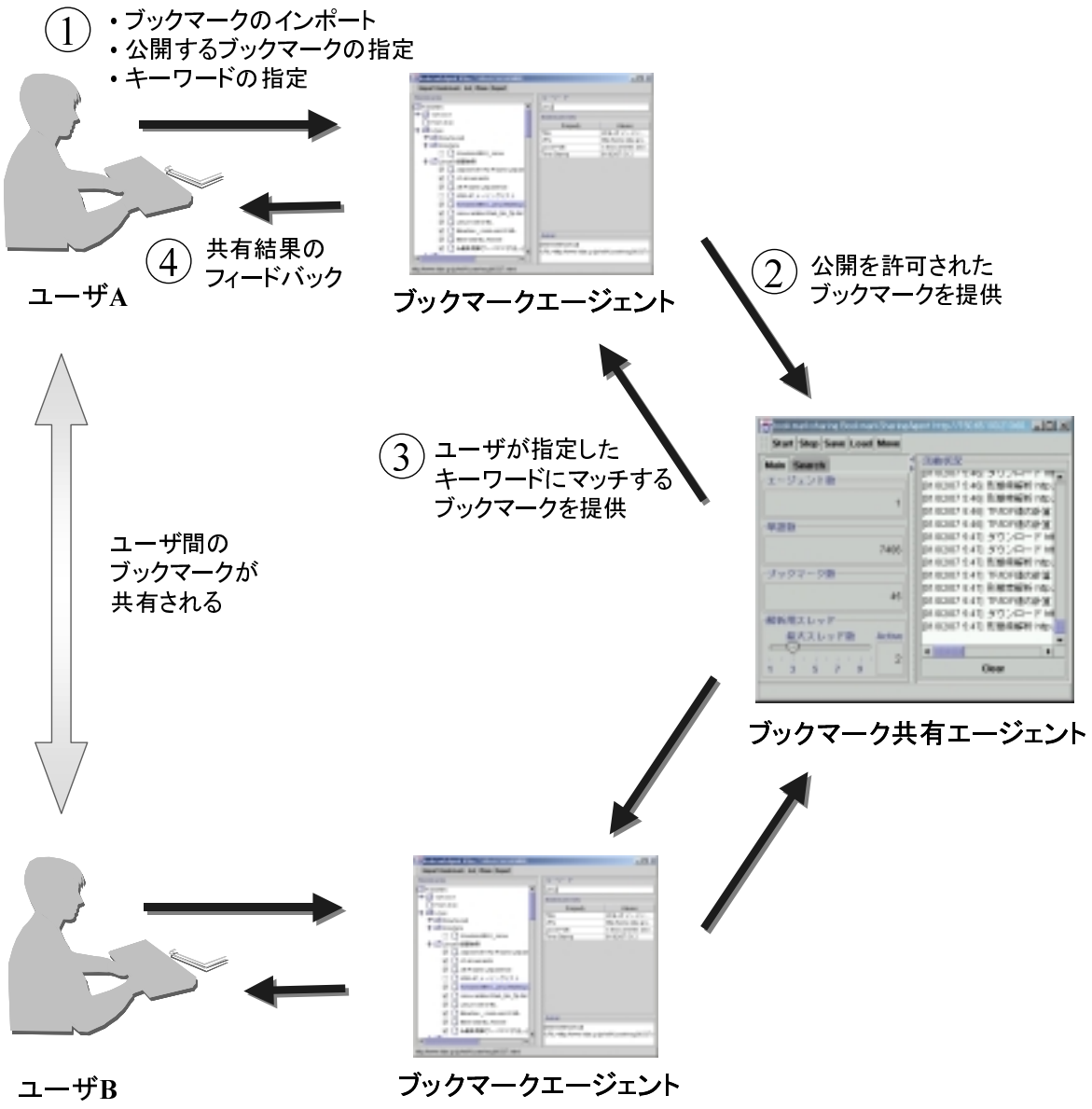


図 6.1: ブックマーク共有システムの概要

ブックマーク共有システムは，ブックマーク共有サービスを提供するブックマーク共有エージェントと，ブックマーク共有システムのユーザクライアントソフトウェアとなるブックマークエージェントから構成される．

ブックマークの共有は，以下の流れ (概要) で行われる．なお，6.3節にて，さらに詳しいブックマーク共有の流れを述べる．

1. ユーザは，ブックマークエージェントにブラウザのブックマークをインポートし他のユーザに公開してよいブックマークを指定すると共に，自分の興味分野を表したキーワードを与える．

2. ブックマークエージェントは，ユーザによって公開を許可されたブックマークを，ブックマーク共有エージェントへ提供する．

3. ブックマーク共有エージェントは，ブックマークエージェントへユーザが指定したキーワードにマッチするブックマークを提供する．

4. ブックマークエージェントは，ブックマーク共有エージェントから得たブックマークをユーザへフィードバックする．ユーザは，ブックマークエージェントから新たなブックマークを得ることで他のユーザとブックマークを共有することができる．

6.2.1 ブックマーク共有エージェント

ブックマーク共有エージェント (Bookmark Sharing Agent) は，Service Agent のサブクラスとして定義され，ユーザから提供されたブックマークを他のブックマークエージェントと共有するサービスを提供する (図 6.2参照)．

ブックマーク共有エージェントの主な機能を以下に挙げる．

(a) 定期的に Directory Agent へプロファイルを登録する

Directory Agent に対して，自身のプロファイルを登録する．Directory Agent に登録することによって，ユーザと情報共有システム上のエージェントは，ブックマーク共有エージェントについての情報を得ることができる．

(b) ユーザへブックマークエージェントを配布する

ユーザからの要求があった場合，ユーザへブックマークエージェントを配布する．

(c) ブックマーク共有サービスを提供する

ブックマーク共有エージェントは，ブックマークエージェントに対してブックマーク共有サービスを提供する．ブックマークの共有方法は後述する．このとき，計算機資源 (高速な CPU, 広帯域回線など) に応じて，ブックマークの解析処理の負荷を調節することができる．

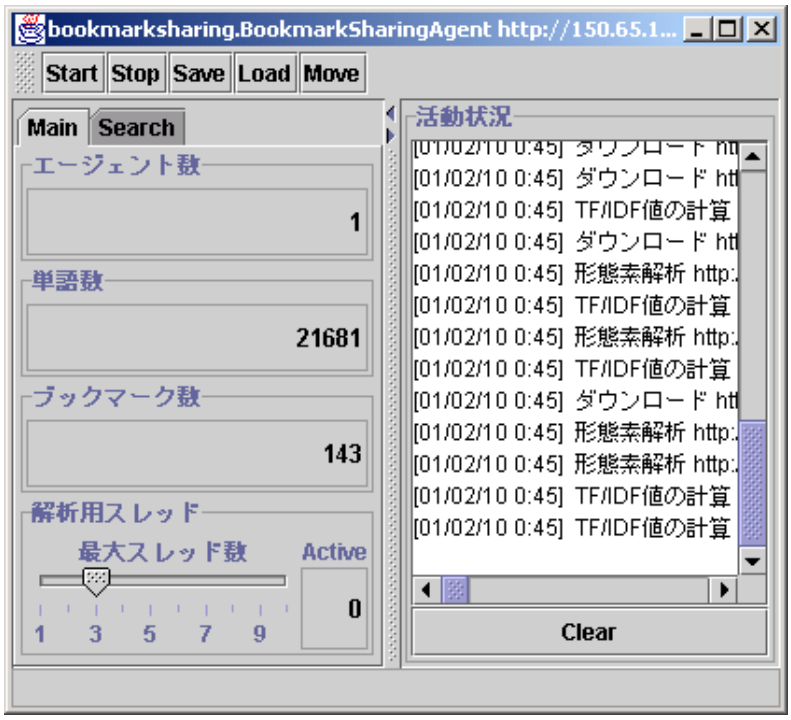


図 6.2: ブックマーク共有エージェントの実行画面

6.2.2 ブックマークエージェント

ブックマークエージェントは，User Agent のサブクラスとして定義され，ブックマーク共有サービスのクライアントソフトウェアとしての機能を持つ．ブックマークエージェントの主な機能を以下に挙げる．

(a) 定期的に Directory Agent へプロファイルを登録する

定期的に Directory Agent へ登録することにより，ユーザは自分のブックマークエージェントの現在位置と活動状態を把握することができる．

(b) ユーザのブックマークをインポートする

ユーザのブラウザからブックマークをインポートする．このとき，ユーザはブックマーク共有エージェントへ提供する (他のユーザへ公開する) ブックマークを指定する．

(c) ブックマーク共有サービスを受ける

ユーザによって公開を許可されたブックマークを，ブックマーク共有エージェントに提供する．ブックマーク共有エージェントは，ブックマークエージェントへユーザが指定したキーワードにマッチするブックマークを提供する．

(c) ユーザへブックマーク共有の結果をフィードバックする

ユーザへ共有の結果得たブックマークを提示し，必要であれば，ユーザのブックマークをブラウザに反映させる．

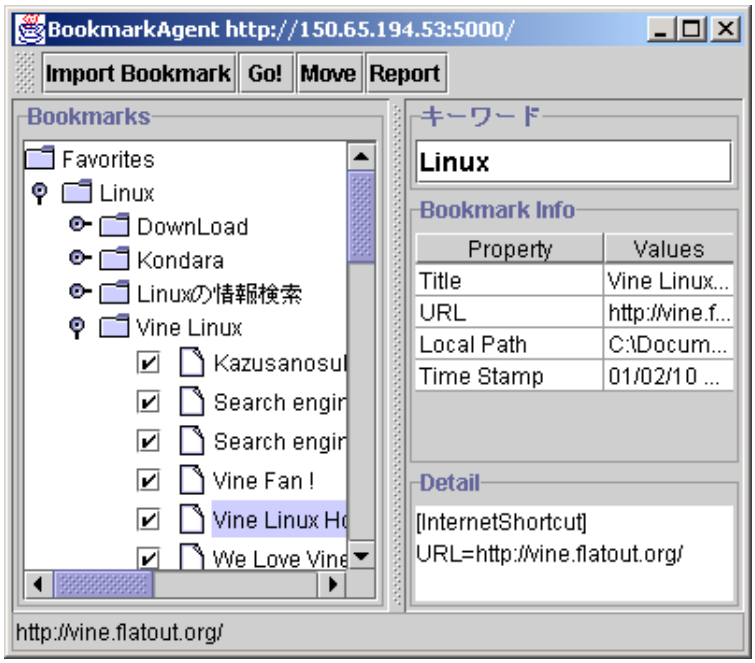


図 6.3: ブックマークエージェントの実行画面

6.3 ブックマーク共有の流れ

本節では，ブックマーク共有システムにおけるブックマーク共有の流れを述べる．なお，以下の条件が満たされていると仮定している．

- 情報共有システム上で，Directory Agent がディレクトリサービスを提供中であること．
- ユーザのホストを含む各ホストには，Server Agent が活動中であること．
- いくつかのブックマーク共有エージェントが活動中であること．

ユーザが行う作業

1. ブックマークエージェントの入手

   ユーザは，ブックマーク共有エージェントに，ブックマークエージェントを要求する．このときのブックマーク共有エージェントの場所 (ホスト) は，Directory Agent に問い合わせることにより得る．なお，ブックマークエージェントの要求と，Directory Agent への問い合わせは，Server Agent が提供する GUI を用いて行う．

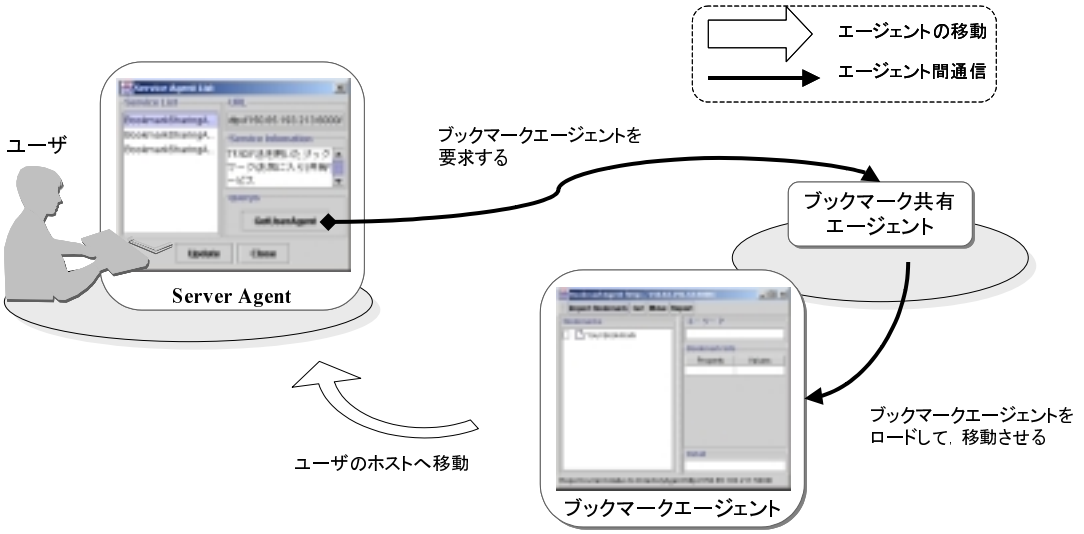


図 6.4: ブックマークエージェントの入手

2. ブックマークのインポートとキーワードの指定

   ユーザは，ブックマークエージェントにブラウザのブックマークをインポートし，他のユーザへ提供しても良いブックマークを指定する．また，入手したいブックマークに関連したキーワードを指定する．

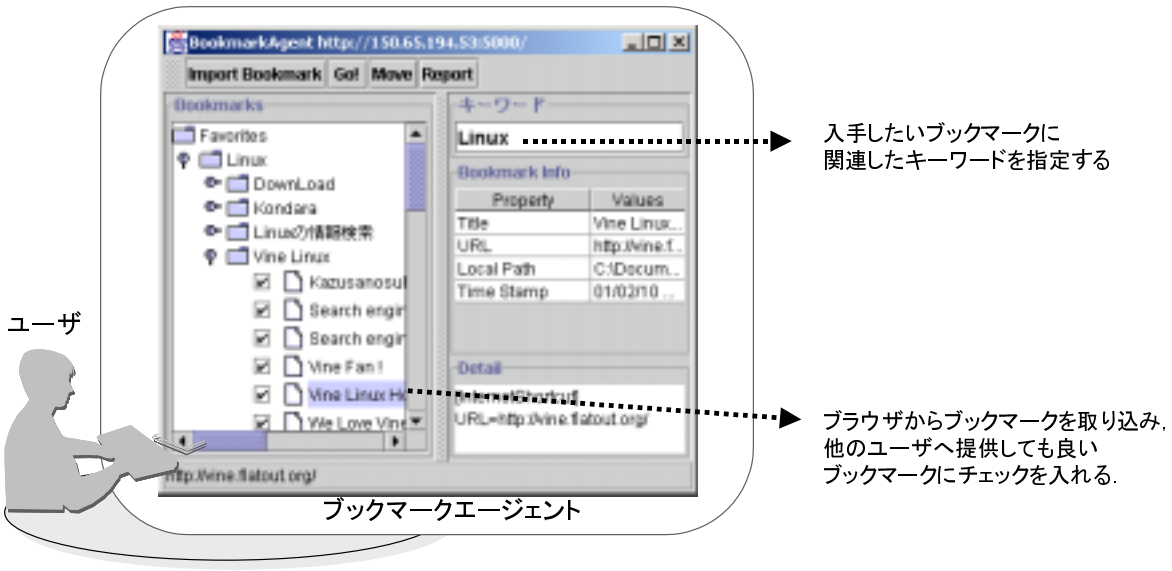


図 6.5: ブックマークのインポートとキーワードの設定

3. ブックマークエージェントをネットワークに放つ

   ユーザは，ブックマークエージェントにブックマーク共有を行うように命令する．以後の作業は，ブックマークエージェントにより行われる．

ブックマークエージェントによって行われる作業

4. 巡回経路の作成

   ブックマークエージェントは，ユーザの望むブックマークを効率よく収集できるように，巡回経路を決定する．これは以下の流れで作成される．なお，以下の作業はエージェント間通信 API を用いて行われる．

   (a) ブックマーク共有エージェントの一覧を Directory Agent に問い合わせる．

   (b) 同期メソッド呼び出しを用いて，それぞれのブックマーク共有エージェントから match メソッドの返値を取得する．このとき，match メソッドの引数に，ブックマークエージェントのプロファイル (ユーザが指定したキーワードが含まれている) を設定する．

   match メソッドは，引数で指定されたプロファイルを持ったエージェントが提供しているサービスにどれぐらい適合しているかを返す (5.6.4節参照)．ブッ

クマーク共有システムでは，各ブックマーク共有エージェントは，ユーザの指定したキーワードを満たすブックマークがいくつあるかを返す (図 6.6参照).

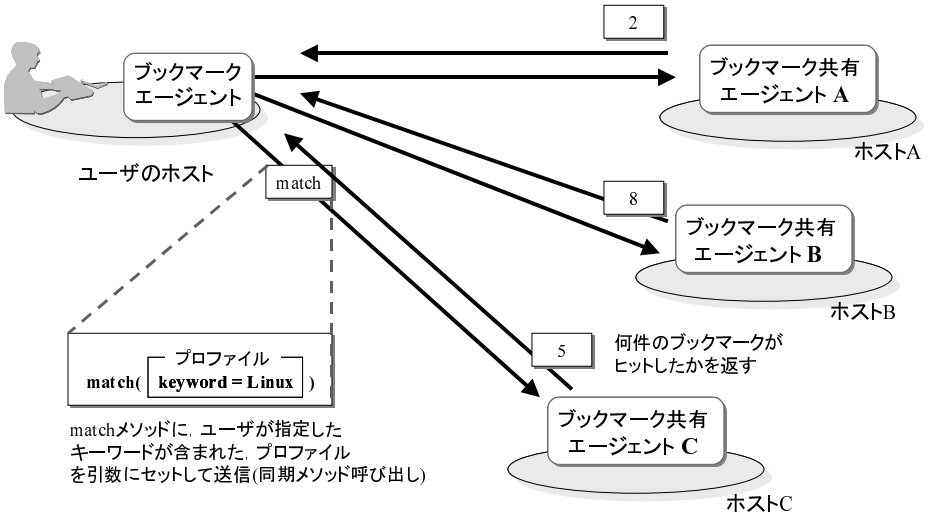


図 6.6: 巡回経路の作成

(c) match メソッドの返値が大きい順に，ブックマーク共有エージェントを巡回する．図 6.6の場合，ブックマークエージェントは，ホスト B,C,A の順に移動する．

つまり，ブックマークエージェントは，ユーザが指定したキーワードに関連したブックマークを最も多く持っているブックマーク共有エージェントを優先的に訪問する．

これは，ブックマークとキーワードは共にユーザの興味を強く表しており，ブックマークエージェントが，ユーザが指定するキーワードに関連したブックマークを多く持つブックマーク共有エージェントを優先的に回ることによって，同じような分野に興味を持つ (同じようなキーワードを指定する) 他のユーザのブックマークエージェントに出会う確率が増し，効率的にブックマーク共有できるためである．

5. ブックマークの共有

4. で作成した経路上のホストを順にたどり，ブックマーク共有を行う．ブックマーク共有は，図 6.7の手順で行われる．

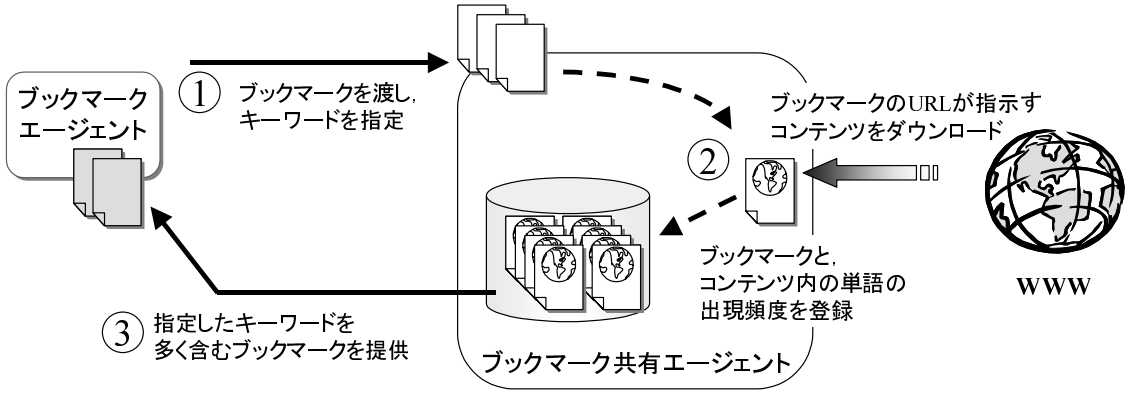


図 6.7: ブックマークの共有方法

ブックマークエージェントは，ユーザから公開を許されたブックマークをブックマーク共有エージェントに提供する (手順 1)．ブックマーク共有エージェントは，提供されたブックマークが指示すコンテンツ (Web ページ) をダウンロードし，単語ごとの出現頻度を調べておく (手順 2)[1]．また，ブックマークエージェントによって指

[1] これらの作業は，ダウロードした HTML ファイルから HTML タグを取り除き，形態素解析を行うことによってなされる．このとき，HTML タグの取り除きに Regex for Java[20] を，形態素解析に茶筅 [21]

定されたキーワードを多く含むブックマークをブックマークエージェントに提供する (手順 3).

6. ユーザへ共有したブックマークを提示

   一定件数のブックマークを収集するか，すべてのホストを巡回し終わった場合，ユーザのホストへ帰りユーザへ収集したブックマークを提示する．

   このとき，ユーザのホストに移動可能であるかを Directory Agent に問い合わせることにより判断する．移動不可能であった場合，現在のホストにて待機し，ユーザのホストへ移動可能になるまで定期的に Directory Agent に問合せを行う．

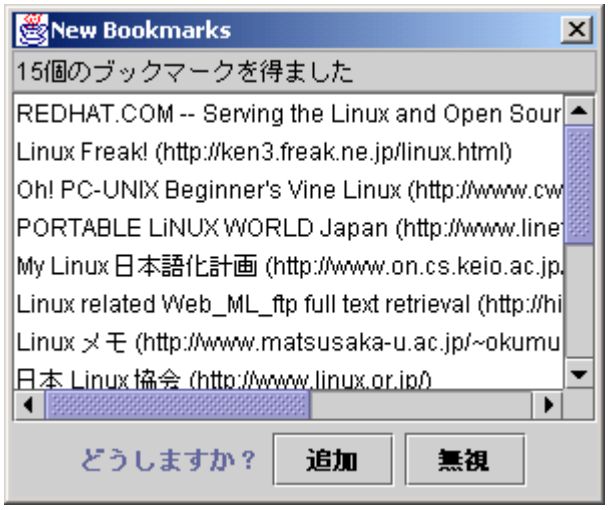


図 6.8: 収集したブックマークの提示

6.4 考察

本節では，本章で構築したブックマーク共有システムについて考察する．

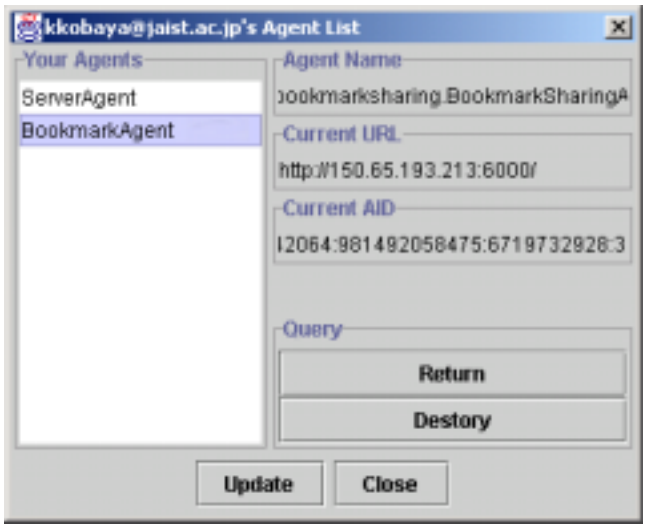
6.4.1 ブックマークエージェントの自律性について

ブックマークエージェントは，Directory Agent へ問い合わせることによって，ブックマーク共有エージェントの場所を得ることができ，移動とエージェント間通信を行うことによって，自律的に行動することができた．

を用いてる．

このため，ユーザがブックマーク共有システムを利用するために行うことは，ブックマークエージェントをブックマーク共有エージェントに要求し，ブックマークエージェントにブックマークをインポートした後，キーワードを指定するだけである．残りの作業は，ブックマークエージェントによって自律的に行われ，ユーザは何ら指示を与える必要は無く，回線の接続状態を維持する必要もない．

また，ブックマークエージェントの状態は常に Directory Agent に登録されているため，ユーザは，Server Agent の GUI を用いて Directory Agent へ 問い合わせることにより，ブックマークエージェントの現在地を得たり，必要な指示を与えることできる (図 6.9 参照)．



kkobaya@jaist.ac.jp が所有者と
なっているエージェントの一覧(左)と
エージェントの現在位置などの情報

図 6.9: ブックマークエージェントの状態の問合せ

6.4.2　ブックマーク共有システムの柔軟性について

ブックマーク共有エージェントは，モバイルエージェントとしての移動性を有しており実行状態を保持したまま移動することができる．よって，移動による負荷分散，資源獲得によるサービス内容の変更といった処理が可能である．

例えば，ブックマークエージェントが集中したことにより過負荷となった場合，負荷に余裕があるホストへブックマーク共有エージェントを複製し移動させることで負荷分散が可能となる (図 6.10参照)．このとき，どうやって移動させるホストを探すかという問題があるが，本システムでは，Directory Agent へ問い合わせることで必要な情報を得ることができる．

さらに，移動直前と移動直後に Directory Agent に対してプロファイルの更新 (現在位置の更新) を行うため，ユーザやブックマークエージェントは，ブックマーク共有エージェントの移動による影響を受けない．

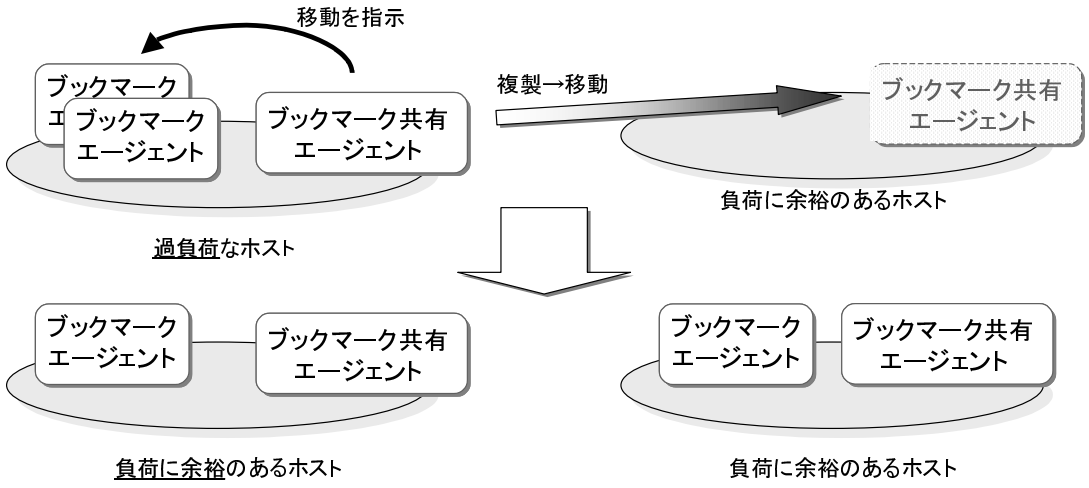


図 6.10: 複製・移動による負荷分散

また，ブックマーク共有エージェントを不活性化状態 (シリアライズ状態) にすることで，ファイルへの保存が可能である．これは，任意のタイミングでサービスのバックアップを取ることができ，エージェントをロードすることにより，いつでも保存した状態に容易に復元できることを意味する．

6.4.3 実装の容易さについて

ブックマーク共有システムは，本研究で提案・実装した情報共有システム上のサービスとして実装され，本研究が提供するフレームワークを拡張することによって実装されている．このため，本システムが提供する，エージェント間通信 API や，ディレクトリサービスを利用することで比較的容易にブックマーク共有システムを実装することができた．

例えば，ブックマークエージェントにおいて，Directory Agent に対して，ブックマーク共有エージェントの一覧を問い合わせて返す requestBookmarkSharingAgentList メソッドは，次のように記述されている．

```
public Vector requestBookmarkSharingAgentList(){
  Vector serviceAgentList = new Vector(); //一覧を格納するベクター

  //名前が bookmarksharing.BookmarkSharingAgent である
  //エージェントの一覧を問い合わせる
  Message query = new Message("select");
  query.setArg("bookmarksharing.BookmarkSharingAgent");

  //Directory Agent を受取側エージェントとして指定
  String toName = profile.getDirectoryAgentName();
  URL    toURL  = profile.getDirectoryAgentUrl();

  try{
    //同期メソッド呼び出しで問い合わせる
    serviceAgentList = (Vector) call(toName,toURL,query,false);
    ...
  }catch(Exception e){
    ...
  }

  return serviceAgentList;
}
```

また，本システムは情報共有を行う際に有用である，いくつかのユーティリティクラスを提供しており，それらのクラスを用いることによって，WWW からの HTML ファイルのダウンロードなどを容易に記述することができる．例えば，NetworkUtility クラスを用いると URL が示す HTML ファイルをダウンロードするためのコードは以下の一行で実装できる．

```
String html = NetworkUtility.getContent(url);
```

このように，本システムを用いることにより，エージェント間通信などを容易に記述でき，ブックマークの共有という主目的に集中して開発することができた．

6.4.4 既存のブックマーク共有システムとの比較

現在，いくつかのユーザ間のブックマーク共有システムが知られている．以下で，既存のブックマーク共有システムと本研究で実装したブックマーク共有システムを比較した．なお，本研究で実装したブックマーク共有システムは，ブックマークの共有手法については本研究のテーマでは無いため，システムの運用，利用するユーザなどに焦点をあて議論している．

- WWW 上のブックマークシステム

  現在,WWW 上にはいくつかのオンラインブックマークサービスが存在する．これらのサイトでは，単に WWW 上にユーザのブックマークを保存するだけではなく，他のユーザとブックマーク共有を行えるサイトもある [22, 23]．

  これらの WWW 上のオンラインブックマークサービスで情報共有を行う場合，ユーザはブラウザからブックマークをアップロードし，他のユーザへ公開しても良いブックマークを指定する．ここまでは，本ブックマーク情報共有システムと同じであるが，以後のブックマークの収集 (共有) は，ユーザの手で行う必要がある．つまり，ユーザは希望するブックマークが得られるまで試行錯誤を繰り返し，その間回線状態を維持していなければならない．本ブックマークシステムでは，ブックマークエージェントが自律的にブックマークを共有するため，ユーザはの負担は軽くなり，回線の接続状態を維持する必要もない．

- エージェントを用いたブックマーク共有システム

  ブックマークエージェント [25] は，協調的情報フィルタリングに基づいたユーザの情報収集を支援するシステムである．このシステムでは，ブックマークエージェントがユーザごとに起動され，ユーザのブックマークに関してのデータベースを持つ．ブックマークエージェントは，ユーザがブラウジング[1]を行うバックグラウンドで，他のユーザのブックマークエージェントと通信することによって，ブックマーク情

[1] Web ページのリンクをたどりながら情報収集を行うこと

報を共有し，ユーザへ表示している Web ページと類似したページを推薦することによって情報収集を支援する．

ここで，ブックマークエージェントの情報交換は互いのデータベースへの検索により実現されているため，ユーザ数が多くなると加速度的にエージェント間の通信回数が大きくなる．このため，ブックマークエージェントが活発に活動すればするほど，ユーザのブラウジングを阻害するといった問題がある．本ブックマーク共有システムでは，モバイルエージェント技術に基づいて構築されており，エージェントの移動により通信回数を削減できる．

第 7 章

考察および関連研究

本章では，本研究で設計・構築した情報共有システムについて考察する．また，関連研究についても本研究と比較し議論する．

7.1 本情報共有システムについての考察

本節では，本研究で設計・構築した情報共有システムについて考察する．

7.1.1 似たような情報を求めるユーザ間の情報共有の実現

本システムでは，モバイルエージェントが持つ移動性と，本システムが提供するディレクトリサービスを利用することにより，自律的なエージェントの実現が可能である．さらに，エージェントがユーザのデータと好み (嗜好) をもつことにより，似たような情報を求めるエージェントを自律的に探しだして情報共有を行うことができ，似たような情報を求めるユーザ間の情報共有を可能とした．本研究では，実際にブックマーク共有システムを構築することで，これらの特徴を実際に確かめることができた．

ここで，本システム上のエージェントは，本システムによって提供されるディレクトリサービスとエージェント間通信 API を用いることによって，エージェントが自律的に行動するための情報 (ホストの状態，エージェントの場所など) を容易に入手することができる．しかし，自律的なエージェントでは与えられた情報 (知識) を元に如何にして行動プランを練るかが重要になってくるが，本システムではエージェントの行動プランの作成を支援する仕組みを持たないため，開発者の負担となる．特に，複数のホストを巡回して情報共有を行う User Agent の実装には，行動プランを作成を支援する仕組みが望まれる．

モバイルエージェントの行動プラン作成に関して，推論技術のひとつであるプランニング機構をモバイルエージェントに持たせた研究として Plangent[12] や MiLog[19] がある．これらのシステムでは，Prolog ライクな言語によってプランニング部分を記述でき，より高度な自律性を持ったエージェントの開発が可能である．

本研究が提案した情報共有システムは，特定のモバイルエージェントシステムに依存した設計ではないため，これらプランニング機構を備えたモバイルエージェントシステム上で，本システムを実装することにより，より高度な自律性を持ったエージェントの実現が期待できる．

7.1.2 不特定多数のユーザへの対応

本システムは，モバイルエージェントによって構成されており，移動による通信量の削減が可能，回線状態変化の影響が少ない，計算機環境に対する依存度が低い等，様々なのユーザの利用に対応できると思われる．

しかし，本システムを多数のユーザで運用した場合，ディレクトリサービスを提供している Directory Agent へ通信が集中しシステム全体のボトルネックになる可能性がある．今後，多数のユーザでの運用実験を行い，実際に不特定多数を対象とした場合のシステムの安定性などを確かめる必要がある．

7.1.3 情報共有サービスの開発の容易さ

本システムは，開発者へ本システム上で稼動する情報共有システムを開発するためのフレームワークを提供した．提供されるフレームワークには，モバイルエージェントを用いたエージェント間通信 API が含まれ，開発者は，柔軟で効率的なエージェント間通信を容易に実現することができる．さらに，エージェントの状態を一元管理するディレクトリサービスを提供することでエージェントの活動に必要な情報を容易に入手することができ，サービス開発者は，提供するサービスの開発に集中することができる．

本研究では，実際にフレームワークを拡張してブックマーク共有システムを構築することによって，容易に情報共有サービスを開発し，公開することができることを実証した．

また，本システムが提供するフレームワークは，特定のドメインに特化したものではなく，汎用的に利用できる．このため，本システムのフレームワークにセキュリティ機構を組み込むなどの拡張も十分可能である．

7.1.4 エージェント間通信 API について

本システムが提供するエージェント間通信 API は，開発者へ Delivery Agent, Serve Agent の存在を意識させることなく利用でき，容易に柔軟で効率的なエージェント間通信を提供できた．

例えば，メッセージが正しく伝わらなかった場合や，受信側エージェントで例外が発生した場合には，Delivery Agent が適切に例外を処理し，送信側エージェントへ伝えることができる．また，回線接続状態を維持する必要があるのは，Delivery Agent の送受信時のみであり，携帯情報端末など良好な回線状態の維持が困難な状況でも通信が可能である．

また，効率的なエージェント間通信の実現という側面では，メッセージを持った Delivery Agent が移動する際には，AgentSpace により圧縮されてからネットワーク上を移動する．このため，メッセージのサイズが大きく，文字列など圧縮率が高くなる場合には，特に効率的にエージェント間通信が可能である．

しかし，現在の Delivery Agent の実装では，受信側エージェントのホストでの例外処理とブロードキャスト送信しか実現しておらず，メッセージがモバイルエージェントであり自律性を持つことのメリットを活かしきれていない．本システムが提供するエージェント間通信では，受取側エージェントが他のホストに移動した場合に，Delivery Agent が受信側エージェントを追跡してメッセージを渡すといった処理や，受信側エージェントが現れるまで特定のホストで待機するといった処理が可能であり，より柔軟な通信方式の提供が今後の課題である．

また，Delivery Agent を用いた通信では，直列化，圧縮，転送, 直列化復元といった Delivery Agent の送受信のための処理が必要であり，計算能力が低いコンピュータではメッセージ送受信にかかるオーバヘッドが問題といえる．

セキュリティへの配慮

情報共有を行う際には，ユーザが見せたくない情報を隠す必要がある．現在の実装では，セキュリティについては考慮しておらず，エージェント間通信の暗号化，エージェントの認証などが必要である．

7.2 関連研究

モバイルエージェントを用いた情報共有を目的としたシステムとして，MAGNET(Mobile AGent NETwork)[7] がある．MAGNET は，モバイルエージェントにより実現される通

信ネットワークシステムであり，アドホックネットワークでの利用を想定している点が特徴的である．

ここで，アドホックネットワークとは，必要に応じ一時的に構築するネットワークであり，ネットワークを構成する端末は事前に互いの存在を知らず，集中管理を行うサーバは存在しない．このため，その場で一時的に接続される任意の端末と通信を行うことが想定され，通信を行う端末が必要なソフトウェアを持っていない場合が必然的に多くなる．また，ネットワーク構成や回線状態も動的に変化するため，情報共有を行う際にはこれらの点に柔軟に対応できる仕組みが必要とされる．

MAGNET では，このようなアドホックネットワークにおける情報共有を目的としているため，ネットワーク上の端末情報の提供，ネットワーク全体の把握，ネットワークの動的な変化の通知といった機能を持ち，シンプルなモバイルエージェントを組み合せ，様々なネットワークプロトコルと同様の動作を実現することが可能である．

MAGNET を用いたアプリケーションとしては，アドホックネットワーク上のファイル監視・転送エージェント [8] がある．ファイル監視・転送エージェントは，ローカルファイルを監視し，ファイルに変更があった場合，エージェントがファイルを持って移動することにより，変更があったファイルを他の端末へ転送を行う．エージェントは，移動先の端末でファイルを復元し，関連づけられたアプリケーションを起動することにより，疑似的にファイル共有を実現することが可能になる．また，移動先でファイルに変更があった場合，同様にして元の端末へ移動し，変更を反映させることが可能になる．さらに，ファイル監視・転送エージェントに各ホストを巡回させることにより，直接通信ができない端末間でもファイルの疑似的な共有が実現できる．

ここで，本研究で提案した情報共有システムは，

- Directory Agent がシステム全体を把握し，ディレクトリサービスを提供することによって，動的なネットワーク構成の変化に対応可能
- Server Aget がホストの状況を監視し，ホスト上のエージェントに情報を提供することによって，ホストの動的な環境変化に対応することが可能
- Delivery Agent を用いたエージェント間通信により不安定な通信回線でも通信可能

といった特徴を持ち，アドホックネットワークにおいても十分適用可能であるといえる．

また，アドホックネットワークでは，特定のメンバーだけによるプライベートな利用法も考えられる．本システムでは，Directory Agent を複製することにより，任意の場所でディレクトリサービスを提供することができる．さらに，抽象クラスである DirectoryAgent

クラス を拡張することにより独自 (プライベートな) の Directory Agent を構築することができ，その場に集まったメンバーだけが利用できる小規模な情報共有システムを構築することもできる．

第 8 章

おわりに

本章では，本研究全体をまとめ，本研究で得られた結論と課題を述べる．

8.1 まとめ

本研究では，モバイルエージェントを用いた情報共有システムを設計・構築した．

本システムでは，モバイルエージェントが持つ移動性と，本システムが提供するディレクトリサービスを利用することによる，自律的なエージェントを特徴する．さらに，エージェントがユーザのデータと好み (嗜好) をもつことにより，似たような目的を持つエージェントを自律的に探しだして情報共有を行うことが可能となった．

また，開発者が新たな情報共有サービスを容易に開発できるようにするために，柔軟なエージェント間通信 API を備えたフレームワークを提供した．開発者は，提供されるフレームワークを用いることにより，本システム上で容易に独自のサービスを構築することができる．

本研究では，実装した情報共有システム上のサービスとして，ブックマーク共有システムを実装し，本システムの有効性を確かめることができた．

8.2 今後の課題

本研究で設計・構築した情報共有システムには，以下の課題がある．

- より高度な自律的の実現

  エージェントのより高度な自律性を実現させるためには，行動プランの作成支援が望まれる．

- より柔軟で効率的なエージェント間通信

  Delivery Agent と Server Agent との組み合わせを活かしたより柔軟で効率的なエージェント間通信が望まれる．

- Direcotry Agent の負荷分散

  本システムを多数のユーザで運用した場合，ディレクトリサービスを提供している Directory Agent へ通信が集中しシステム全体のボトルネックになる可能性がある．よって，多数のユーザで運用するためには，Directory Agent の負荷分散が必要である．

- セキュリティへの配慮

  情報共有を行う際には，ユーザが見せたくない情報を隠す必要がある．現在の実装では，セキュリティについては考慮しておらず，エージェント間通信の暗号化，エージェントの認証などが必要である．

謝辞

本研究を行うにあたり，終始ご指導して下さった渡部 卓雄 助教授，二木 厚吉 教授に心から感謝いたします.

天野 憲樹 助手 からは，有益な助言を多数頂きました．ここに深く感謝の意を表します.

お茶の水女子大学 理学部 情報科学研究科 佐藤一郎 助教授 には，モバイルエージェントシステム AgentSpace の利用を快諾して頂きました．ここに感謝いたします.

また，日頃より共に研究活動を行ってきた言語設計学講座の諸氏に感謝いたします.

参考文献

[1] 武田英明, インターネットにおける情報の知的利用 -情報収集から情報 統合へ-, 専門講習会講演論文集「インターネットの最新技術と 家庭への浸透」, 電子情報通信学会関西支部, 1998.

[2] 松塚健, 谷口雄一郎, 武田英明, kmedia:ブックマークを用いた情報推薦・ユーザ間関連発見システム, 電子情報通信学会 人工知能と知識処理研究会/知能ソフトウエア工学研究会, 2000.

[3] Inktomi and NEC "Web Surpasses One Billion Documents". http://www.inktomi.com/new/press/2000/billion.html , Jan. 2000

[4] World Wide Web Consortium, "HTML 4.01 Specification", http://www.w3.org/TR/html401/, W3C Recommendation 24 December 1999.

[5] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., and T. Berners-Lee, " Hypertext Transfer Protocol – HTTP/1.1 ", RFC 2068, January 1997.

[6] Windows2000 Active Directory 入門, Daniel Blum 著,QUIPU LLC 訳2000/1.

[7] 河口 信夫, アドホックネットワークにおけるモーバイルエージェントの応用マルチ・エージェントと協調計算ワークショップ (MACC'98),1998/12.

[8] 杉浦俊一, 河口信夫, 外山勝彦, 稲垣康善 アドホックネットワーク上の柔軟な情報共有システム 平成 11 年度電気関係学会東海支部連合大会講演論文集 , 1999.

[9] White,J.E. , Telescript technology, the foundation for the electronic marketplace, White Paper. General Magic,Inc. 1994.

[10] IBM Aglets Software Development Kit, http://www.trl.ibm.co.jp/aglets/ .

[11] Lange,D.B. and Oshima,M, Programming and Deploying Java Mobile Agents with Aglets, Addison-Wesley,1998.

[12] 永井保夫，田原康之，入江豊，大須賀昭彦，本位田真一, Plangent I: インテリジェント・ネットワークエージェント, ホットトピックスと並列人工知能研究会 (第 2 回), JSAI Technical Report SIG-HOT/PPAI-9603-6, pp.29-36, November 23, 1996

[13] 佐藤一郎 , AgentSpace: モーバイルエージェントシステム, 第 7 回マルチ・エージェントと協調計算ワークショップ (MACC'98),1998/12 .

[14] MonJa, http://www.melco.co.jp/rd_home/java/monja/index.html .

[15] Lead Story - Aglets -, http://www.trl.ibm.co.jp/news/lead_aglets.html , 1997/12.

[16] 旅行情報サイト「たび Can」検索エンジンを Java 移動エージェントで開発, http://java.nikkeibp.co.jp/Java/Report/971226TabiCan.shtml ,1997/12.

[17] 加瀬直樹，池谷直紀，大須賀昭彦，柴田康弘, ゆがしまん (1)　－ヒューマンナビゲーションをエージェントで実現できるか－, 情報処理学会 第 61 回全国大会, 2000/10.

[18] 池谷直紀，加瀬直樹，大須賀昭彦，柴田康弘ゆがしまん (2)　－ヒューマンナビゲーションをエージェントで実現できるか－, 情報処理学会 第 61 回全国大会, 2000/10.

[19] Naoki FUKUTA, Takayuki ITO and Toramatsu SHINTANI, MiLog: A Mobile Agent Framework for Implementing Intelligent Information Agents with Logic Programming", In proc. of the First Pacific Rim International Workshop on Intelligent Information Agents(PRIIA'2000),pp.113-123, 2000/8

[20] IBM alphaWorks http://www.alphaworks.ibm.com

[21] 松本裕治，北内啓，山下達雄，平野善隆，松田寛, 高岡一馬, 浅原 正幸 日本語形態素解析システム『茶筌』 version 2.2.1 使用説明書 奈良先端科学技術大学院大学, 2000/12.

[22] Blink.com - manage,organize,and share your Web bookmarks online, http://www.blink.com

[23] ブリンクフォルダ, http://www.blink.co.jp/

[24] Book まーく, http://www.bookmark.ne.jp/

[25] 森，山田ブックマークエージェント：ブックマークの共有による情報検索の支援 電子情報通信学会論文誌，Vol. J83-D-I，No. 5，pp. 487-494,2000