Flying Secretary
－ Javaによるオフィスエージェント通信基盤 －

朝倉 敬喜*、石黒 義英*、渡辺 幸光†、喜田 弘司*、垂水 浩幸*

{asakura,ishiguro,kida,tarumi}@obp.cl.nec.co.jp, yukimi@nis.nec.co.jp

*NEC 関西C&C研究所　　†NEC情報システムズ

筆者らが提案するワークウェブシステムでは様々なエージェントがリソースの調整、業務の代行を行うが、通常のオフィス業務を考えた場合、オフィスのデスクトップのコンピュータで作業をするだけでなく、出張先等でネットワークを介して作業をすることも考えられる。これまで提案してきたエージェント間の通信基盤ではこの遠隔地アクセスが不可能だったため、本稿では、「世界中どこでもWWW Browserさえあれば自分の秘書エージェントを呼び出して様々なオフィスエージェントを利用することができる」ことを目指したFlying Secretaryの提案を行う。この通信基盤はJavaのクラスとして実装し、クラス間でやりとりされるINA/LI(INtelligent Agent LIte) Javaプロトコルはこれまでの INA/LIプロトコルと互換性を持つ。

Flying Secretary
- A communication Infrastructure for the Office Agent by Java -

Takayoshi Asakura*, Yoshihide Ishiguro*, Yukimitsu Watanabe†, Koji Kida*, Hiroyuki Tarumi*

*NEC Kansai C&C Research Laboratories　　†NEC Informatec Systems,Ltd.

In the WorkWeb System we proposed, various agents coordinate resources and execute tasks for users. In a usual office, we not only work with desktop computer, but work through network from remote sites. In our communication infrastructure for the office agent, it was impossible that we access agents through network. This paper describes a communication architecture for the agent office system. Furthermore on this communication platform, we propose the `Flying Secretary', aiming to enable users to ``call secretary agent and use various office agents from anywhere with a WWW browser''. A new communication platform is constructed with Java's classes, and is compatible with old INA/LI (INtelligent Agent LIte) protocol.

1. はじめに

筆者らは、従来のワークフローに代わるシステムとしてワークウェブシステムを提唱している[1]。このシステムでは、複数の業務によるリソースの奪い合いの調整や、個人の裁量性を広げることをエージェントで支援している。ワークフローを管理するBPTエージェントや組織のデータを管理するGIMエージェント[2]、個人のリソース管理や秘書として働くパーソナルエージェント等から構

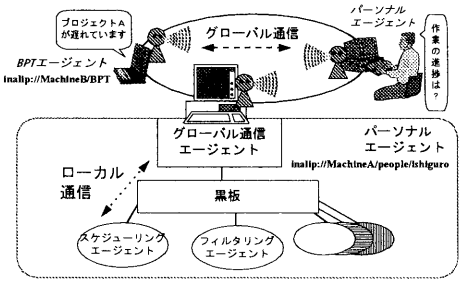

図１：エージェントの構成

成される(図１)。パーソナルエージェントはさらにフィルタリングサブエージェント[3]やスケジューリングサブエージェント[4]から構成されている。

これらのエージェントは互いに通信できるが、本システムではORBとWindowsのDDEをベースとした通信基盤を提供し、ここでの通信基盤をINA/LI通信基盤、基盤上のプロトコルをINA/LIプロトコルと称している[5]。INA/LI通信基盤は次の二層から構成されている(図２)。

(1)パーソナルエージェント内のサブエージェント間のローカル通信(Windows DDE)

(2)パーソナルエージェント外の各エージェント間のグローバル通信(ORB)

通信を二層にわけたのは、パーソナルエージェント内のサブエージェントを外部から隠蔽しセキュリティーを保持するとともにパーソナルエージェント内の構造にかかわらず同じようにアクセスできるからである。ローカル通信とグローバル通信の結合はパーソナルエージェントに含まれるグローバル通信エージェントが行い、グローバル通信側のエージェントとローカル通信に接続しているサブエージェントはグローバル通信エージェントを介してのみアクセスを許している。単一のオフィスのみで利用する場合はこれで充分であると考えているが、研究を進めてゆく過程で以下の要望が出てきた。

- 歴史的な経緯からローカル通信、グローバル通信の実装が異なっている。
- パーソナルエージェントは個人の端末上に存在することを前提にしているため、ワークフロー管理等を円滑に進めるためには全員の端末の電源を常時入れておく必要があり、現実的でない。
- Windowsでしか利用できない。

また、出張等でオフィスに在席していない時にも、ワークフローの進捗管理や新規スケジュールの確認等で本システムの必要性が高いが、以下の理由でこれまでは利用できなかった。

- ORBがLAN内でしか使えないため、グローバルエージェントの利用範囲が狭く、出張時等LAN外からのアクセスがファイアウオール内であっても不可能であり、利用端末にエージェントをインストールしなければならない(ORBの採用は開発効率上の理由)。

そこで、HTTP、Javaを用いて以下の目的を達成できるようINA/LI通信基盤を拡張する。

- ローカル通信、グローバル通信の実装上の区別をなくす。
- 常時動作している必要があるエージェントはサーバ上で動作させる。
- 必要なときにどこからでもエージェント、データをロードし利用可能とする。
- プラットフォーム(マシン環境)依存をなくす。

これにより、ORBベースのワークウェブシステムの問題点が解決される。INA/LI Java通信基盤を利用して遠隔地からオフィスのエージェントを呼び出して利用することが、あたかも「オフィスの自分の秘書を呼びだしている」ように見えることから、INA/LI Java通信基盤を用いた環境全体を

"Flying Secretary" と呼び、本稿でその構成について説明してゆく。

2. 利用イメージ

図２に *"Flying Secretary"* の利用イメージを示す。遠隔地からWWWブラウザを用いてオフィスのエージェントサーバにアクセスすると、サーバ上で活動している秘書エージェントがワークフローの実行状況、メール到着状況、スケジュール受付状況等のデータとともにWWWブラウザ上に出現し、それらデータを表示する。さらに、利用者とサーバ上で実際に動作している様々なエージェントとのインタラクションを可能にする。

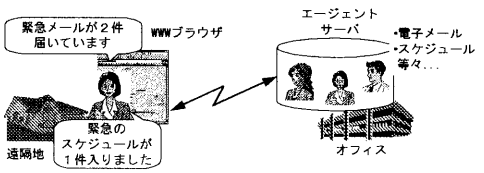

図２：Flying Secretaryの利用イメージ

3. 構成

3.1 概要

Flying Secretaryでは様々なエージェントがネットワーク上を移動するが、起動時の通信量、起動時間を最小にするため、エージェントを以下の２モジュールで表現する。

- サーバ上に常駐して機能を提供する Server Agent
- ユーザインタフェースを構成するINA/LI Agent Applet

このうちクライアントにロードされるのはINA/LI Agent Appletである。また、既存のORBベースのAgentが存在する場合は、Server AgentはAgent AppletとORB版Agentとの橋渡しをする。さらに、Server Agentの動作するサーバの負荷を分散するためや、本来のオフィスでINA/LI Agent Applet、これまでのAgentを利用した時にデータ、ユーザインタフェース等の整合性をとるために、グループ、部等のまとまった単位を管理単位とし、エージェントはそこに属するマシン上であれば動作可能とする。

図３に構成を示す。サーバにはエージェントを管理するAgent Manager、各エージェント(INA/LI Agent AppletとServer Agent)が存在する。さらに、ユーザログインサーバ(遠隔地のユーザが最初にアクセスするサーバ)には、WWWサーバ(HTTP Daemon)とログイン認証エージェントを置く。また、管理単位の任意のサーバ上に管理単位全体のエージェントの実行状況を管理するDirectory Agentを置く。

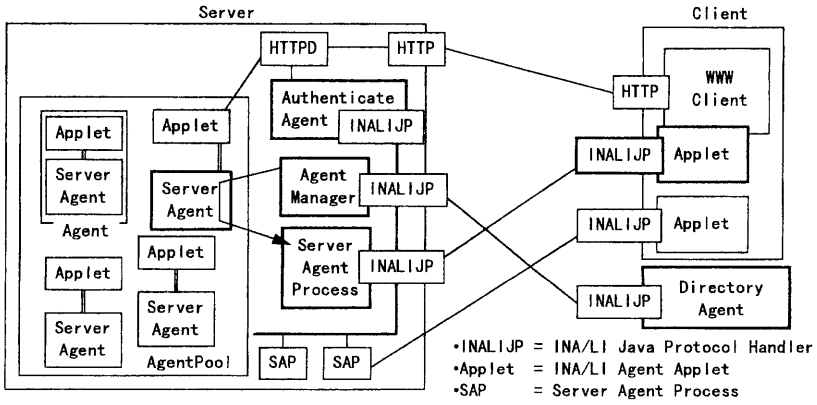

図３：Flying Secretaryの構成

サーバは複数存在し、例えば個人のデスクトップマシン１台１台がサーバにあたるが、各サーバでエージェントを動作させるためにはマシン所有者の許可(認証エージェントが管理)が必要であるとする。ただしNetscape Navigatorの制限によりAppletはロード元のサーバとしか通信できないため、認証サーバにすべてのAgentのAppletを置く必要がある。Server Agentについてはどのサーバに存在しても問題ない。以下にそれぞれの構成要素について説明してゆく。

3.2 INA/LI Java Protocol Handler

INA/LI Java プロトコルを解釈するのはJavaのクラスセットInaliComで、これがProtocol Handlerとなり、クライアント、サーバ両方の機能を持つ。実際の接続はホスト名とポート番号で行われるが、INA/LI Java Protocol Handlerはこれらを隠蔽し、Agent構築者が論理的アドレスのみでアクセスできる環境を提供している。INA/LIプロトコルはオープンな通信基盤のため、エージェント間の接続可否の設定によるセキュリティー管理が必要であるが、現状では本来のJavaのセキュリティーに制限されるのみである。

INA/LI Java Protocol Handler上のプロトコルはKQMLに準拠している。一例を下記に示すが、本来この部分はProtocol Handlerによって隠蔽されているものである。

```
(performative     :content(通信内容)
                  :language inali
                  :ontology inali
                  :id id
                  :sender sender-address
                  :receiver receiver-address)
```

*performative*にはメッセージに対する要求が入り、inalijpではevaluate(実行依頼)、reply(返信)のどちらかが入る。「通信内容」の部分にエージェント間で定められた通信内容が入る。認証の場合は、認証サーバに対して以下の内容が送られる。

```
(authenticate  :username asakura
               :password pasazo)
```

language, ontologyは通信内容の文法がinaliであり、概念もinaliであることを示している(固定)。sender, receiverはメッセージ送受信者のアドレスで、次節で述べる論理的URL表記である。

3.3 エージェント

エージェントはそのユーザインターフェースを実現するINA/LI Agent Applet(以下、単にAppletとする)と、エージェントの機能本体であるServer Agentから構成される。

Appletは必要になるたびにサーバ上にあるエージェントの状態とともにクライアントにロードされる。ロードされたAppletはAgent Managerに対してAppletをロードしたユーザの権限でServer Agentの立ち上げを要求し、Server Agent Processを生成させる。

AppletはServer Agentとの通信確立後にエージェントのユーザインタフェースとなり、サーバからのデータを表示するとともにユーザからの操作によって対応するServer Agent ProcessとINA/LI Javaプロトコルで通信を行う。Server Agentはサーバ上でいわゆるデーモンとして常時立ち上がっているものと、必要なときだけAgent Managerによって立ちあげられ、機能完了時に終了するものの2種類ある。Applet格納サーバとServer Agent格納サーバは同じマシンである必要はない。

Server AgentはすべてInaliComPackageを利用して通信を行うため、基本的にはJavaで記述するがJavaがC言語とのリンクも可能なことから、データベース等既存のアプリケーションを利用するエージェントはJavaとC言語で記述することになる。

各エージェントには一意の名前をURL表記で与える(論理アドレス)。これはINA/LIグローバル通信とプロトコル名(inalijp)以外同一の表記法であり、例えば以下に示すような表記法である。

・Server Agent
inalijp://fows.co.jp/asakura/SchAg

・Applet
inalijp://fows.co.jp/asakura/SchAg/Applet

この例では，ログイン認証エージェントが存在するマシンfows.co.jp上のユーザasakuraのスケジュールエージェント(SchAg)を指定している。ユーザの最初のアクセス時には、まずTop UI Agentと呼ぶメインとなるエージェントにアクセスするが、これは特例として以下の表記とする(ユーザasakuraのTop UI Agentの例)。

inalijp://fows.co.jp/asakura

3.4 Agent Manager

サーバのエージェント(Server Agent/Server Agent Process)を管理するプロセスである。以下の機能を提供する。

3.4.1 アドレス解決

他サーバのAgent Managerまたは自サーバのServer Agent Processから、他のServer Agentへの通信要求があったときに、通信要求元にその実アドレスを返却する。このアドレス解決には後述するディレクトリエージェントを利用する。

3.4.2 Server Agentの起動

起動していないServer Agentへの通信要求が他のAgent ManagerやServer Agentから来た場合に、Server Agentをプロセス化して実アドレスを返却する。このときサーバに存在するエージェント名と実際のエージェント実行ファイルパスの対応関係を示すエージェントリストを参照する。

3.4.3 Server Agentの終了

Appletや他のServer Agent Processからエージェント終了要求がきた場合に、Agent ManagerはServer Agent Processを終了させるとともに、ディレクトリエージェントへ登録内容削除を通知する。

3.4.4 メッセージ振り分け

Agent Appletから各Server Agentにメッセージが来た時、その内容に従い処理を行うが、解釈できない内容の場合、Server AgentはAgent Managerにメッセージを転送する。これを受け取ったAgent Managerは内容を解釈し適当なServer Agentへメッセージを転送する。転送すべきServer Agentが見つからない場合はBroadcastする。メッセージ振り分けの知識はW2-SHELL[6]の対話フローを用いる。

3.5 ディレクトリエージェント

Server Agent Processの生成状態を記録するテーブルであるエージェントプロセステーブル(エージェント論理アドレス、エージェント物理サーバ名、ポート番号の対応)を管理する。Agent ManagerがServer Agentの起動・終了に利用するとともに、INA/LI Protocol HandlerがServer Agentへの通信の確立(アドレス、ポート番号取得)のために利用する。

3.6 認証(Authenticate)エージェント

利用ユーザの認証を行うエージェントであり、利用者名、パスワード、利用者に属するエージェントの物理アドレス、各エージェントのデフォルト物理アドレスの各テーブルを保持している。

3.7 TopUI Agent

利用者が最初にアクセスしたときにロードされるのがTopUI Agentである。TopUI Agentは認証インタフェースを持ち、認証エージェントと通信を行い利用者の特定を行う。その後、TopUI Server Agentと通信を行い、以下の(ユーザの)各データを収集し、クライアント上に表示する。

- ユーザが発行したワークフローの実行状況、指示
- ユーザが関わっているワークフローの実行状況、指示
- 新規に入ったスケジュール一覧と内容
- 新規に到着した電子メール一覧と内容
- これらに対する簡単なインタラクション(Yes/No)

これ以上の操作を行いたい場合には、各機能をサポートするエージェントを呼び出すことになる。

3.8 関連研究

移動型エージェントの通信基盤の研究としては、Telescript[7]、April[8]が有名である。Telescript、Aprilはクライアント側(エージェント利用側)で処理手順を決定しておき、電話回線等を通じてサーバに移動し、いくつかのサーバを渡り歩いて必要な処理を実行し、処理が完了したらクライアント側に戻り結果を表示するアプリケーションの記述言語である。特にAprilはサーバの負荷を関知して自由にサーバを動き回ることができるようになっている。

今回利用したJavaも移動型エージェントを構築するためのベースとなりうるが、片方向のメッセージ伝送しかできない。

INA/LIでは、Telescript、Aprilのような、機能やデータを持って移動するエージェントは現在はサポートしていない。これは、ユーザの秘書として他のエージェントに対する交渉窓口となるエージェントは常にオフィスに常駐しているべきと考えたからである。したがって、Telescript等とは根本的に設計指針が違うが、その点を除いてこれらの研究と比較したときのINA/LI Java通信基盤の特徴として以下の点があげられる。

- Javaを基盤として両方向のメッセージ伝送を可能にし、Javaを利用できるWWWブラウザさえあればどこでもエージェントが利用可能となる。
- ユーザインタフェースであるINA/LI Java AppletとServer Agentの2つに分割することで移動型エージェントの非同期性(一旦エージェントが活動すると、その状態を知ることができず処理の中止もできないこと)がなくなった。

また、ワークフローと遠隔地アクセスの組み合わせでは、日立のGroupMax[9]がある。WWW経由で遠隔地から普段オフィスで利用している機能(スケジュール管理、ワークフロー管理、文書管理、電子メール)を利用することができる点で本研究と同じである。しかしながら、遠隔地から利用できる機能は、CGIでも可能なレベルにとどまっている。また、単に機能を利用するだけで、WWW Browserにロードしたappletと他のAppletとの協調等が考慮されていない。

4. おわりに

6月末現在、INA/LI Java通信基盤(InaliCom Package)、Agent Manager(メッセージ振り分けを除く)、Directory Agent等は実装されている。

現在の仕様ではURLの漏洩によってServer Agentを他人の権限で起動する事が可能であるため、Appletをロードする毎にパスワードを要求してセキュリティーを確保しているが、利用上煩わしいという問題点があり、これを解決する必要がある。

今後はInaliCom Packageをベースとした各エージェントの実装を進め、*"Flying Secretary"*を実現してゆくとともに、一部をフリーソフトウェアとして公開する予定である。

参考文献

[1] 垂水ほか:ワークウェブシステムの実現,情報処理学会グループウェア研究会,GW-15-22,1996

[2] 吉府ほか:ワークフローとデータベースの相互連携システム,情報処理学会グループウェア研究会,GW-9-23,1995

[3] 朝倉ほか:エージェントによる情報フィルタリング,情報処理学会情報メディア研究会,IM-20-9,1995

[4] 喜田ほか:仕事依頼交渉を支援するスケジューラエージェント,情報処理学会第52回全国大会論文集,1X-7,1996

[5] 石黒ほか:マルチエージェントオフィスシステムとその実現,MACC'95,1995

[6] 久寿居ほか:対話I/F構築ツールW2-SHELL,情報処理学会ソフトウェア工学研究会,SW-108-6,1996

[7] http://www.telescript.com/ts.html

[8] Hans Haugeneder,et.al.:IMAGINE:AFramework for Building Multi-Agent Systems, CKBS'94, pp31-64, 1994

[9] http://www.hitachi.co.jp/Prod/comp/soft1/gmax/html/gmax.html