3D1-03

マルチエージェント技術を用いた共同作業システムの設計と実装

Design and Implementation of Collaboration System by Multi Agent Programming

山崎航 YAMAZAKI Wataru　西山裕之 NISHIYAMA Hiroyuki　平石広典 HIRAISHI Hironori　溝口文雄 MIZOGUCHI Fumio

東京理科大学　理工学部
Faculty of Sci. and Tech., Tokyo University of Science

In this paper, we design and implement the collaboration system by using the multi-agent programming language. We show that many complicated functions such as sending message, sharing object, and distributed processing are easily defined by our multi-agent programming language.

1. はじめに

近年，グリッドや P2P に見られるように，広域ネットワークにおいて分散したプログラムを実行するような環境が広く普及しつつある．そのような環境では，プログラム同士は互いに通信をを行いながら，協調を実現する．さらに，それらの通信は，メッセンジャープログラムやネットワーク上でのコラボレーションシステムといったプログラムを利用するユーザ環境のコミュニケーションにも利用される．

しかしながら，このような分散環境におけるメッセージ通信は，非常に複雑になる傾向がある．既存のメッセージ通信のライブラリでは，本来の言語使用と異なるプログラミングスタイルを要求されるため，多様なメッセージの形態を適切に記述することは，プログラム作成者の大きな負担となっている．

本論文では，分散環境におけるプログラムの通信を我々の設計・実装したマルチエージェント記述言語，JMRL を用いてこれらの問題に対応する．具体的には，分散コラボレーションシステムを実現するための，オブジェクトの共有，移動，協調計算といった事項を行うためのメッセージ通信を JMRL によって容易に記述できることを示す．JMRL では，メッセージは論理型言語の項として実現されるため，複雑なメッセージの送受信とその処理はユニフィケーションによって半自動的に行われる．これによって，プログラム作成者のメッセージの送受信に対する負担を軽減することを目的とする．

本論文は以下のように構成される．第二章では，システムの概要と必要となる要素についてまとめる．第三章では言語 JMRL の概要と，その利用される範囲について述べた後，メッセージ通信及びオブジェクトの共有を中心にシステムの実装について述べる．第四章では，関連研究や今後の展開を示し，本論文をまとめる．

2. 設計方針

本論文で扱う分散環境における共同作業とは，分散したプロセスと，オブジェクトの共有，及びホスト間でのメッセージのやり取りによって実現される作業（プログラム）のことを指す．以下では，それぞれの具体的内容について述べる．

2.1 分散プロセス

分散プロセスは，個々のホストや計算機における計算能力である．個々のプロセス同士はメッセージを通信し，計算の分担や，オブジェクトの共有を実現する．我々は，分散プロセスをエージェントとし，通信はエージェント間通信として定義し，これらをマルチエージェント言語でこれらを記述する．プロセス間の通信は，一対一通信の他ブロードキャストが利用できる必要があり，プロセス自体は，動的に生成されたり，消滅したりすることを考慮する必要がる．JMRL ではプロセスは階層構造を持っているため，これによって以上の問題を解決する．

2.2 分散オブジェクトの共有

各プロセスは，メッセージ通信と，オブジェクトを仮想的に共有することによって，協調的な動作を可能とする．オブジェクトを共有するための方式としては，オブジェクトを作成したホストが管理を行う方法，及び，一元的にオブジェクトを管理する方式の二通りが考えられるが，我々は，黒板システムによって一元的に管理する方式とした．共同作業のためのオブジェクトを共有と，ホスト間でのメッセージ通信を黒板を用いて実装する場合，以下のようなことを実現する必要がある．

- プロセスの生成と登録
  黒板にプロセスを登録し，オブジェクトの共有やメッセージ通信を行うことが出来る状態にする．
- 共有オブジェクトの登録
  共有すべきオブジェクトを黒板に登録，登録したオブジェクトの更新の機能が必要となる．オブジェクトの更新には，オブジェクトの履歴を保存し，上書きを行わない場合と，オブジェクトを上書きする場合が考えられる．
- 更新の通知
  黒板は共有されているオブジェクトが更新されたことを，登録されたプロセスに対して通知する．
- オブジェクトのダウンロード
  各プロセスは，必要であれば，更新されたオブジェクトをダウンロードする．オブジェクトはローカルで更新中である場合も考えられるため，ダウンロードするかどうか，及びそのタイミングは，各プロセスで判断される必要がある．

2.3 コラボレーションシステム

本論文で言う共同作業システム（コラボレーションシステム）とは，分散プロセス，共有オブジェクト，及びプロセス間メッセージを利用して構築されるプログラムのことを言う．JMRL は，次章で示すように，並列論理型言語のシンタックスを持っているため，メッセージの送受信を容易に記述することができる．一方，共有オブジェクトとそれを利用するプログラムは，

連絡先: 山崎航，東京理科大学理工学部，〒 278-8510 千葉県野田市山崎 2641, Tel:0471-24-1501ex6011,Fax:0471-21-4225, Email:yamazaki@imc.tus.ac.jp

そのすべてを論理型言語で記述するのは，現実的ではない．そのため我々は，プログラムのうち，メッセージのやり取りのみをマルチエージェント言語を用いて記述すると仮定する．具体的な方法としては，JMRL と Java のインタフェースを利用して，プログラムを作成することを仮定する．共有オブジェクトは Java のオブジェクトを仮定し，JMRL 側とは名前で管理されるオブジェクトプールを介して統合される．

3. 実装

3.1 JMRL の記述形式

JMRL では，MRL[1] のシンタクスを継承しており，いくつかの拡張を持つ．プロセスはエージェントという単位で記述され，エージェントは，状態と，ルールの集合からなる．各ルールはガードつき並列論理型言語のシンタクスを持ち，加えて，外部とのメッセージ通信のための特殊な述語として，そのプロセスを起動した親エージェントへの通信のための’’^’’, 子エージェントへの’’!’’, 特定エージェントとの一対一通信のための’’.’’ 及び java との通信のための”java” のそれぞれを組み込み述語として用意される．これらはガードで出現した場合入力を，ボディで出現した場合出力を意味する．例えば，

```
run(S1,S2):- ^hello|!world,run(S3,S4).
```

は，状態が，S1 及び S2 にマッチした状態で，親から項 hello が届いた場合，すべての子に項 world を送信し，状態を S3 及び S4 に変化させることを意味する．

3.2 黒板への登録

前節の記述形式を用いて，プロセスの黒板への登録は以下のような記述で行われる．

```
run(..,List,..) :- X.regist(Name,Host)|
  new(agent(Name,Host)),run(..,[Name|List],..).
```

ここで，List は黒板に登録されているエージェントのリストであり，任意のホスト X からメッセージ `regist(Name,Host)` の形式の項が到着した場合に，新たにエージェント Name をホスト Host で生成し，新たなエージェントを現在のエージェントリストに加えることを意味する．これによって，黒板プロセスを親エージェント，黒板を利用する利用プロセスを子エージェントとして管理する．

3.3 共有オブジェクトの送受信

オブジェクトの送信は，次のように記述することができる．

```
run(..,Name,ShareOBJ,..):-java(j_updata(ID))|
  ^obj(ID,update(Name,ShareOBJ)),
   run(..,Name,ShareName,..).              ...(a)
```

ここでは，Java からのメッセージが `j_update(ID)` であった場合に，オブジェクトプールからオブジェクトを ID で取り出し，そのオブジェクトをメッセージ `update(Name,ShareOBJ)` と共に黒板に送信する．ここで，ShareOBJ は，黒板上で管理されている共有オブジェクト名を表す．

黒板は，オブジェクトが更新されると，オブジェクト ShareOBJ がホスト Host によって更新された事実をすべてのクライアントにメッセージ update(SharOBJ,Host) によって通知する．これを受けるための各クライアントのルールは，

```
run(..,Name,..):-^updata(X,A),A !=Name |
  getobj(X,ID),java(foo(ID)),run(..,Name,..).
```

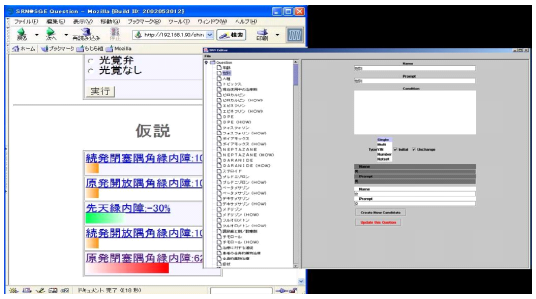

図 1: 共同作業アプリケーション

のように記述できる．これは，上記のメッセージが到着した場合で，その更新メッセージが自分が送信したものでない場合，オブジェクトを X を用いて取り出し，それをオブジェクトプールに ID として登録する．さらに，java プログラム `foo.java` を ID と共に呼び出し，処理を行うことを意味している．getobj 述語は，組み込みではなく別に定義されているが，ここでは詳細は省略する．

3.4 アプリケーション

図 1 は，本節で述べた記述形式によって，作成されたアプリケーションである．共有オブジェクトとして指定しているのは，エキスパートシステムのルールファイルであり，複数のホストからのルールの変更やその時点でのキスパートシステムの動作を共同作業システムとして実装した．ユーザインタフェースでアップデートボタンが押されると，その状態のオブジェクトを更新するため，ローカルなオブジェクトプールにオブジェクトを保存し，ローカルのエージェントにメッセージを送信する．エージェントは，Java からのメッセージを受けるとプログラム (a) のようなルールがマッチし実行される．本アプリケーション，共有されるルールオブジェクトは，黒板上で上書きせず，その履歴を保存している．従って，上記 (a) とは若干異なるルールが実行される．更新の履歴の保存によって，本アプリケーションではルールを古い状態に戻したり，試験的に利用することが可能となっている．

4. おわりに

本論文では，マルチエージェント記述言語 JMRL を用いて，分散環境における共同作業システムの設計と実装を行った．共同作業を行うためのプロセスの分散とオブジェクトの共有は，JRML によって簡潔に記述できる．本言語は，Java によって実装された黒板システムを用いたエージェント言語 Jinni [2] と近いが，我々の言語はより低レベルのメッセージ記述を行う点が異なる．黒板システム自体を本言語で実装することは容易なため，より汎用的であると考える．今後は，記述の容易さをより客観的に明らかにし，エラーハンドリング等の機能を充実させる予定である．

参考文献

[1] 西山裕之，山崎航，溝口文雄，並列論理プログラミングに基づくマルチエージェント言語 MRL, ソフトウェア科学 Vol.20 No1，2003

[2] Paul Tarau. Jinni: Intelligent Mobile Agent Programming at the Intersection of Java and Prolog. Proceedings of PAAM’99, 1999.