マルチモバイルエージェントシステム記述用言語 Maglog

川 村 尚 生† 半 場 寛 之†† 金田 悠紀夫††

情報がコンピュータ群に分散して存在しているネットワークにおいて，複数の移動型エージェントが並行動作し，協調・競合するようなシステムを記述するための言語 Maglog を提案する．ネットワークを構成するホストはあらかじめ登録されている．ホストの持つデータやプログラムは，フィールドと呼ばれる受動的オブジェクトに格納される．フィールドの情報は，エージェントからの参照可否を示すスコープ属性や，エージェントからの変更可否を示す許可属性を持つ．エージェントはフィールドに入ることで，自らの情報とフィールドの提供する情報を区別することなく利用できる．また，エージェント同士はメッセージ通信が行える．エージェントおよびフィールドの記述には拡張 Prolog を用いる．エージェントのホスト間移動は往復が組になっており，ホストをまたいだユニフィケーションやバックトラックが可能である．Maglog の応用分野としては，グループウェアや分散制約問題などが挙げられる．本発表では，Maglog の概要を示すとともに，例題を通じて，Maglog の記述性，適用性を示す．

Maglog: A Programming Language for Multi Mobile Agent Systems

Takao Kawamura,† Hiroyuki Hanba†† and Yukio Kaneda††

In this presentation, we propose a programming language named Maglog for multi mobile agent systems. In these systems, agents move around hosts offering some services and use not only data but program in them to solve their problem. Agents are written in extended Prolog so that unifications and backtracking over hosts are available. We also present examples written in the language to demonstrate the effectiveness of our model.

1. はじめに

モバイルエージェントシステムとは，ユーザの代理人となるプログラム (エージェント) が，ネットワーク上に複数存在するホストを移動しながら，与えられた問題の解決をはかるシステムをいう．

これまでに，モバイルエージェントシステムを記述するための言語や枠組が提案されている．Aglets[1)]，Concordia[2)]，MobileSpaces[3)] など，その多くが Java をベースにしているが，以下に挙げる理由により，複数のホストに移動し，そこに存在するデータやプログラムを利用して問題解決を行うエージェントを記述するのが困難であった．

(1) エージェントは移動後特定のコールバックルーチンの先頭から実行することしかできないので，移動先が複数ある場合，コールバックルーチンが複雑になる．どのホストに移動してきたかということと，どこから移動してきたかということによって処理内容を場合分けしなければならないからである．

(2) 複数ホスト間にまたがる制約を記述する場合，あるホストで得たデータをインスタンス変数に保存し，他のホストのデータと比較するといった処理をプログラマが陽に書く必要がある．

本発表では，このような問題点を解決するため，以下に述べる特徴を有した，マルチモバイルエージェントシステム記述用言語 Maglog を提案する．

(1) Prolog をベースとする

(2) 複数ホストにまたがるバックトラックとユニフィケーションが可能

(3) エージェントの移動が必ず往復で対になる

2. Maglog システム

Maglog(Mobile AGent system based on proLOG の略) システムは，ネットワークで結合されたホスト群において動作する．ホストはエージェントの実行環境であるエージェントサーバをプロセスとして走らせ

† 鳥取大学工学部
Faculty of Engineering, Tottori University
†† 神戸大学大学院自然科学研究科
Graduate School of Science and Technology, Kobe University

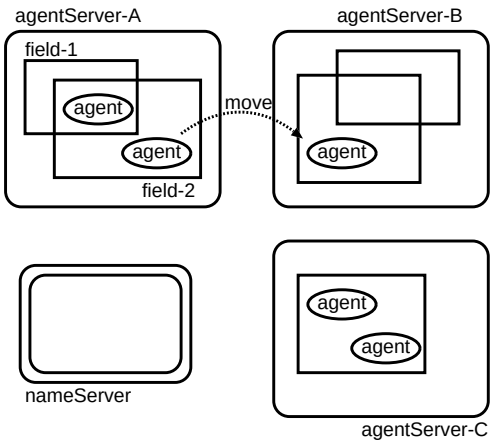

図 1 Maglog システムの構成
Fig. 1 Components of a Maglog system.

ている．エージェントサーバ内には，動作主体であるエージェントと，エージェントが利用するデータやプログラムを集めたフィールドが存在する．ホストには，エージェントサーバの他に，ユーザがシステムを使用するためのユーザインタフェースプロセスが存在する．また，特定のホスト上に，システム全体で一意でなければならない名前などの管理を行なうネームサーバが存在している．この様子を図 1 に示す．

2.1 エージェントサーバ

エージェントサーバは，各ホストに 1 つ存在し，エージェントが活動する場を提供する．すなわち，Prolog インタプリタを内蔵し，エージェントのプログラムを実行する．また，フィールドを保持している．

2.2 エージェント

エージェントは Prolog プログラムと実行状態からなる．エージェントサーバ上の Prolog インタプリタによって解釈・実行される．

エージェントはユーザから直接生成されるか，既存のエージェントが

```
create(AgentID, File, Goal)
```

という形式の組み込み述語を実行することにより生成される．ここで，`AgentID` は出力引数であり，生成されたエージェントの識別子が返される．`File` および `Goal` には，それぞれエージェントのプログラムが格納された URL，生成されたエージェントが最初に実行すべきゴールを与える．

与えられたゴールを実行し終わったエージェントは消滅する．

組み込み述語 `create` を実行したエージェントを「親のエージェント」，それによって生成されたエージェントを「子のエージェント」とみなすことにすると，エージェント群は親子関係の階層構造を形成する，任意のエージェントについて，親をたどることにより，ユーザに直接生成されたエージェントに行き着く．これをユーザエージェントと呼ぶことにする．ユーザエージェントはシステムに複数存在し得る．

ユーザエージェントが生成されたエージェントサーバを，ユーザエージェントおよびその子孫のエージェントの本籍地と呼ぶ．また，ユーザエージェントを生成したユーザを，ユーザエージェントおよびその子孫のエージェントの所有者と呼ぶ．

エージェントの所有者，または祖先のエージェントは

```
kill(AgentID)
```

という組み込み述語によって，任意の時点でエージェントを消滅させることができる．

組み込み述語 `kill` のために，エージェントの位置を追跡する必要がある．これは，エージェントが移動するごとに，本籍地のエージェントサーバに登録することで実現する．この方法は，[4] で述べられている Logging，Brute-Force，Registry の 3 つの方法のうち，Registry に近いが，単一のレジストリサーバに処理が集中するという欠点を持たない．

エージェントは組み込み述語 `create` が返す，システム全体で一意な識別子によって特定される．エージェントが自身の識別子を得るには

```
get_id(AgentID)
```

という組み込み述語を用いる．また，

```
bind(AgentID, Alias)
```

という形式の組み込み述語を実行することで，別名として `Alias` が使用できるようになる．

2.3 フィールド

フィールドはエージェントが利用する手続を保持するオブジェクトであり，エージェントサーバの起動時に静的に生成され，移動したり消滅したりすることはない．

フィールドの手続は public なものと private なものに分けられ，エージェントは public な手続だけを実行することができる．private な手続はフィールド固有の名前空間に置かれるので，エージェントから参照することはできないが，public な手続から private な手続を呼び出すことはできる．

手続を public とするには，フィールド内に

```
:-public PredSpec [ , PredSpec ... ].
```

という宣言文を書く．ここで，`PredSpec` は `name/arity` という形式を持つ．

public な手続は原則として読み出ししかできない．

エージェントが節を削除したり追加したりできるようにするには，フィールド内で

```
:-dynamic PredSpec [ , PredSpec ... ].
```

と宣言する必要がある．dynamic な手続は自動的に public になる．

dynamic な手続は一時には高々1 つのエージェントが占有できる．占有は，エージェントが参照することで自動的に行われ，エージェントがその手続を参照しなくなったとき，すなわち，その手続以前にバックトラックした場合か，消滅した場合に自動的に解かれる．あるエージェントが，すでに他のエージェントに占有されている手続を参照しようとしたときは，占有が解除されるまで停止する．占有が一定時間解除されない場合に，停止したままにならず，参照を失敗終了させたい場合は，後述の組み込み述語 `within/2` を使う．

2.4 エージェントとフィールド

エージェントは

```
in(Goal, Field)
```

という組み込み述語（中置記法を用いて `Goal in Field` と記述できる）によって，フィールド `Field` の public な手続のうち，あらかじめ参照すると指定してあるものをスコープに入れて (以後，このことを，フィールドを attach するという) ゴール `Goal` を実行することができる．`Goal` の実行が終われば，`Field` で定義された手続はスコープから外される (以後，このことを，フィールドを detach するという)．

フィールドを attach するとき，`Field` の public な手続は自動的にはスコープに入らない．Maglog では，スコープに入ったフィールドの public な手続を自ら定義した手続と区別なく扱えるため，意図しない手続がスコープに入ることを避けなければならないからである．これは Java におけるクラスやパッケージの参照手段である import 宣言が，クラスが提供する public な部分を一括してスコープに入れることと対照的である．

手続をスコープに入れるためには

```
import(Goal, [PredSpec, ...])
```

という組み込み述語（中置記法を用いて `Goal import [PredSpec, ...]` と記述できる）を使う．これにより，`PredSpec` で指定した手続をスコープに入れて `Goal` を実行することができる．例えば

```
(A in F_A, B in F_B) import [p/2]
```

は手続 `p/2` をスコープに入れて，フィールド `F_A` でゴール `A` を実行し，フィールド `F_B` でゴール `B` を実行する．

フィールドから import する手続は，原則として，エージェントの手続や，他のフィールドの手続と融合されない．すなわち同じ手続を構成するものとはみなされない．同じ名前を持つ手続については，後から import されたものだけが有効となる．融合したい場合は `import/2` の代わりに，`merge/2` を使う．`merge/2` は複数のフィールドに分割されたデータベースを統合するときなどに使える．

エージェントは複数のフィールドを同時に複数 attach できる．また，複数のエージェントが同じフィールドを同時に attach することもできる．

エージェントが attach できるフィールドは，自身が存在しているエージェントサーバ内のものに限られる．他のエージェントサーバ内のフィールドを attach するためには，まずそのエージェントサーバに移動しなければならない．そのためには

```
on(Goal, Host)
```

という組み込み述語（中置記法を用いて `Goal on Host` と記述できる）を用いる．これは，`Host` に移動して `Goal` を実行することを表している．`Goal` の実行が終われば，エージェントは自動的に移動前の場所に戻る．つまり，Maglog ではエージェントの移動単位が往復で対になっていて，一方通行の移動はできない．

エージェントの移動は Strong Migration であり，移動後は移動直前の状態から実行を再開する．

あるフィールドを attach したまま他のホストに移動することはできない．すなわち，`in/2` の第 1 引数で指定するゴールの中で，`on/2` を実行することはできない．

2.5 エージェント間通信

エージェント間の通信手段として，同期メッセージ通信を提供する．送信には組み込み述語

```
send(AgentID, Message)
```

を，受信には組み込み述語

```
recv(AgentID, Message)
```

を用いる．いずれの述語もメッセージの送受信が完了するまで停止する．

送信されたメッセージは，まず受信エージェント固有のメッセージボックスに先着順に格納される．`recv/2` が実行されたとき，第 2 引数とメッセージボックス内のメッセージがユニフィケーションによって検索される．

2.6 タイムアウトを指定したゴール実行

次の 3 つ場合に，エージェントのゴール実行がいつまでも完了しないことがあり得る．

(1) 移動しようとするホストが稼働していなかったりネットワークに故障が発生している場合の `on/2`.

(2) 相手エージェントと同期できない `send/2,recv/2`.
(3) 他のエージェントが占有している dynamic な手続への参照.

そこで，指定時間内にゴールを実行することを指定する組み込み述語 `within/2` を用意する．すなわち，

```
within(Goal, Sec)
```

は，`Sec` 秒以内に `Goal/2` の実行が完了しない場合は失敗する．

3. 記 述 例

簡単な記述例を通じて Maglog の機能を説明する．

3.1 買　　物

次のような問題を考える．

host_A，host_B，host_C，host_D がある，host_B，host_C，host_D は同一の商品を販売している．host_A のエージェントが 3 つの子エージェントを生成し，host_B，host_C，host_D に派遣する．各エージェントは商品の値段を調べ，親エージェントに報告 (通信) する．親エージェントは最も安い商品を選んで購入指令を出すとともに，他には不購入の指令を出す．

付録 A.1 に買物プログラムを示す．各ホストは market というフィールド内に，商品の名前と値段を `price/2` という事実節に，商品の在庫数を `stock/2` という事実節に格納しているものとしている．また，商品を購入する子エージェントは，`stock/2` を直接書き換えるものとする．

3.2 スケジューリング

Maglog の特徴の一つは，Prolog をベースとしているため，ホストをまたいだユニフィケーションやバックトラックが行なえることである．そのため，複数のデータベース間での制約を満たすといったことが容易に記述できる．一例として，会議のスケジューリングを行なうプログラムを検討する．

複数人の社員 (別々のホスト上) がいる会社で，各社員は，自分固有のフィールドにスケジュールデータを格納しており．空き時間は 1 時間単位で管理されていて，`free(Day,Hour)` という事実節によって表現されるものとする．また，会議を召集するエージェントは，会議参加者全員の空き時間のうち，もっとも早いものを予約するものとする．

プログラムを付録 A.2 に示す．複数ホスト間にまたがる制約の解決は手続 `search/3` において，再帰を用いて簡潔に表されている．なお，`free/2` は予約完了まで占有されるので，複数のエージェントが並行して別々の会議の予約を行うこともできる．

4. 関連研究との比較

Aglets や MobileSpaces などの Java をベースにした既存のモバイルエージェントシステムのほとんどは，JVM の制限もあり，エージェントの移動は Weak Migration であり，移動に際してプログラムカウンタやローカル変数は保存されず，移動後のエージェントはコールバックルーチンの最初から実行することになる．移動先が一ヶ所に決まっている場合はよいが，複数の移動先を持つエージェントを記述しようとすると，コールバックルーチンに複雑な条件分岐が必要になる．また，エージェントの移動が一方通行なのも，コールバックルーチンが複雑化する原因で，ユーザがエージェントの移動経路を常に意識しなければならない．一方，Maglog においては，エージェント移動は Strong Migration であり，移動が往復で対になっているため，プログラムの記述を簡潔に行える．

また，これらのシステムでは，エージェントが移動先のプログラムやデータを利用するには，間接的な方法を利用することになる．例えば Aglets では，あるコンテキストに移動してきたエージェントは，コンテキスト内にあらかじめ生成しておいたステーショナリ・エージェントと対話することで，コンテキスト内のサービスを利用する．一方，Maglog では自らの手続と移動先の手続を区別なく利用できるので，プログラムが簡潔に書ける．

5. おわりに

エージェントが複数のホストを移動し，移動先のデータだけではなくプログラムも直接利用しつつ問題解決を行なう適応型モバイルエージェントシステム Maglog を提案した．Maglog は，エージェントの移動が往復でセットになっていること，ホストをまたいだユニフィケーションやバックトラックが可能であることから，エージェントの記述が簡潔に行える．このことを簡単な例で示した．

今後，システムの実装を進める．

参 考 文 献

1) Lange, D. B. and Oshima, M.: *Programming and Deploying Java Mobile Agents with Aglets*, Addison-Wesley (1998).
2) Wong, D., Paciorek, N., Walsh, T. and Dicelie, J.: Concordia: An Infrastructure for Collaborating Mobile Agents, *Proceedings of the First International Workshop on Mobile Agents*, Vol. 1219, Springer-Verlag, pp. 86–97 (1997).

3) Satoh, I.: MobileSpaces: A Framework for Building Adaptive Distributed Applications using a Hierarchical Mobile Agent System, *Proceedings of IEEE International Conference on Distributed Computing Systems*, IEEE Press, pp. 161–168 (2000).
4) Aridor, Y. and Oshima, M.: Infrastructure for Mobile Agents: Requirements and Design, *Proceedings of the Second International Workshop on Mobile Agents*, Vol. 1477, Springer-Verlag, pp. 38–49 (1998).

付　　録

A.1 買物のプログラム

```
% 親エージェントのプログラム

main:-
  create_children([hostA, hostB, hostC],
                  Agents),
  collect_prices(Agents, Prices),
  buy(Prices), !.

create_children([], []):- !.
create_children([H|Hrest], [Id|Arest]):-
  create(Id, child,
         main(parent, Id, H, computer)),
  create_children(Hrest, Arest).

collect_prices([], []):- !.
collect_prices([A|Arest], [A/P|Prest]):-
  recv(A, P) within 300,
  collect_prices(Arest, Prest).

buy(Prices):-
  find_floor_price(Prices, Buyer),
  order(Prices, Buyer).

find_floor_price([Buyer], Buyer):- !.
find_floor_price([A,B|Rest], Buyer):-
  find_floor_price([B|Rest], X),
  min(A, X, Buyer).

min(A/P, B/Q, A/P):- P =< Q, !.
min(A/P, B/Q, B/Q).

order([], _) :-!.
order([Buyer/Price|Prest], Buyer/Price):-
  send(Buyer, buy) within 300, !,
  order(Prest, Buyer/Price).
order([A/_|Prest], Buyer/Price):-
  send(A, notbuy) within 300, !,
  order(Prest, Buyer/Price).

% 子エージェントのプログラム}

main(Parent, Id, Host, Goods):-
  (get_price(Goods, Price),
   send(Parent, Price, 300),
   recv(Parent, Direction) within 300,
   Direction = notbuy;
   retract(stock(Goods, N) within 300),
   N1 is N-1,
   assert(stock(Goods, N1)),
  ) in market import [get_price/2,stock/2]
    on Host.

% HostA の market フィールドのプログラム
:- public get_price/2.
:- dynamic stock/2.

get_price(Goods, Price) :-
  price(Goods, Price).

price(computer, 1000).
stock(computer, 10).

% HostB の market フィールドのプログラム
:- public get_price/2.
:- dynamic stock/2.

get_price(Goods, Price) :-
  price(Goods, Price).

price(computer, 500).
stock(computer, 5).

% HostC の market フィールドのプログラム
:- public get_price/2.
:- dynamic stock/2.

get_price(Goods, Price) :-
  price(Goods, Price).

price(computer, 800).
stock(computer, 20).
```

A.2 スケジューリングのプログラム

```
% エージェントのプログラム

main(Day, Hour) :-
  Member = [person(fieldA, hostA),
            person(fieldB, hostB),
            person(fieldC, hostC)],
  search(Day, Hour, Member),
  book(Day, Hour, meeting_1, Member),!.

search(Day, Hour, []) :-!.
search(Day, Hour,
       [person(Field, Host)|Rest]) :-
  (free(Day, Hour) within 300
     in Field import [free/2],
   search(Day, Hour, Rest)) on Host.

book(Day, Hour, App, []) :-!.
book(Day, Hour, App,
     [person(Field, Host)|Rest]) :-
  ((retract(free(Day, Hour)),
    assert(schedule(Day, Hour, App))
      within 300
   ) in Field import [free/2, schedule/3],
   book(Day, Hour, App, Rest)) on Host.

% HostA の fieldA フィールドのプログラム

:- dynamic free/2, schedule/3.
```

```
free(1,9).
free(1,10).
free(1,11).
free(1,18).
free(2,11).
free(2,14).

% HostB の fieldB フィールドのプログラム

:- dynamic free/2, schedule/3.

free(1,10).
free(1,11).
free(1,14).
free(2,9).
free(2,14).
free(2,15).
free(2,16).

% HostC の fieldC フィールドのプログラム

:- dynamic free/2, schedule/3.

free(1,11).
free(2,10).
free(2,14).
free(2,16).
```