場の概念を持つモバイルエージェントフレームワークの開発について

木下 慎[†] 山根 慎太郎[†]
川村 尚生[†] 菅原 一孔[†]

分散環境で有効なソフトウェアアーキテクチャとして，モバイルエージェントシステムが注目されている．我々は，モバイルエージェントシステムの主要機能である，(1) エージェント間通信，(2) エージェントのコンピュータ間移動，(3) エージェントによる移動先コンピュータ上のデータ及びプログラムの利用を，統一的かつ簡潔なインタフェースによって提供するために,「場」の概念を持つモバイルエージェントフレームワーク Maglog を開発し，Java 環境に実装した．

Maglog ではエージェントの記述には Prolog を基にした言語を用いる．エージェントはネットワークで結合された各コンピュータにおいて，エージェントサーバと呼ぶプロセス上でスレッドとして動作する．エージェントサーバ内には，Prolog 節を格納できるフィールドと呼ぶオブジェクトが存在する．このフィールドが場の概念を表現するものである．エージェントはフィールドからの Prolog 節の読み出し，フィールドへの Prolog 節の書き込みを通じて他のエージェントと同期的，あるいは非同期的に通信できる．また，エージェントはフィールドに「入る」ことで，そのフィールドが存在するコンピュータに移動し，フィールド内のデータやプログラムを自らが持つそれと区別することなく利用できる．

本発表では Maglog について説明し，Maglog によって効率的なモバイルエージェントを簡潔に記述できることを示す．

Development of a Mobile Agent Framework with a Concept of a Field

Shin Kinosita ,[†] Shintaro Yamane ,[†] Takao Kawamura [†]
and Kazunori Sugahara[†]

Mobile agent systems have been attracted attentions. We have developed a mobile agent framework named Maglog and have implemented it on Java environment. Maglog has a concept of a field. With the concept, Maglog provides simple and unified interfaces for (1) inter-agent communication, (2) agent migration between computers, and (3) utilization of data and programs on computers.

In Maglog, mobile agents are written in a language based on Prolog. An agent runs as a thread in a process which we call an agent server. Agent servers are connected to networks. Agent servers have objects which store Prolog clauses. We call them *fields*. An agent can communicate with other agents synchronously or asynchronously by reading/writing Prolog clauses from/into fields. Moreover, by enterning a field, an agent can migrate to the computer in which the field exists and utilize data and programs in the field without distinguishing from it in the field and it of the agent itself.

In this presentation, we introduce Maglog and present that mobile agents can be described simply and efficiently in Maglog.

1. はじめに

分散環境で有効なシステムとしてモバイルエージェントシステムが注目されている.[1)] モバイルエージェントシステムとは，自律的なエージェント群がネットワークを移動しながら，互いに協力し，与えられた問題の達成を図るシステムである．自律的とは，エージェントが動的に変化する環境に合せ，自らの行動を決定することである．また，エージェントによって複雑なシステムを構築する場合，1 つのエージェントに全ての機能を持たせるのは現実的ではなく，通常複数のエージェントの協調動作によって実現される．そのため，モバイルエージェントには学習，推論，プラン

† 鳥取大学
Tottori University

ニング，他のエージェントとのコミュニケーション能力などが要求される．

これまでこのようなシステムを構築するためのフレームワークが提案されており，[2)~4)] それらの多くは Java 上に実装されている．Java はマルチプラットフォームであり，モバイルエージェントシステムを構築するのに適している．しかし，Aglets[2)] などは，エージェントの動作を Java で記述する必要があり，エージェントの学習や推論といった機能は実現しづらい．その為，Prolog インタープリターを Java 上に構築し，エージェントの動作を Prolog で記述できるフレームワークも提案されている．[3)] Prolog はユニフィケーションと呼ばれる強力なパターンマッチング機構と，バックトラック機構を持ち，エージェントの学習，推論，プランニングといった機構が実現しやすい．

また，複数のエージェント間で協調動作する為に，エージェント間のコミュニケーションや，情報の共有の為の仕組も必要である．このような仕組を実現するものとして，場と呼ばれる概念が提案されている．[4)] 場とは，ネットワーク上の仮想的な場所で，データやプログラムが格納されており，エージェント間での協調や，ホスト内のローカルな情報にアクセスするための手続きをエージェントに提供するために利用される．

これまで様々なモバイルエージェントフレームワークが提案されてきたが，上記の 2 つの特徴を合わせ持ったフレームワークは提案されていない．

我々は場の概念を導入したモバイルエージェントフレームワーク Maglog の開発を行なっている．Maglog では，エージェントのプログラムを Prolog で記述し，場の概念を実現するフィールドと呼ばれるオブジェクトが存在する．フィールドに Prolog 節を追加，削除するための組み込み述語や，エージェントのホスト間移動などのいくつかの組み込み述語が用意されており，自然な Prolog プログラムとしてモバイルエージェントを記述できる．また，それらの組み込み述語を用いて，フィールドを用いたエージェントの動作の動的な変更，エージェント間の同期/非同期通信が簡潔に記述できるようになっており，これまで述べた，モバイルエージェントに必要な機能が記述しやすい．

本稿では，Maglog の言語仕様について説明し，いくつかの記述例によってその記述性の高さを示す．そして，現在我々が開発中である，Maglog の Java 環境での実装について説明する．

本稿の残りは以下のように構成されている．2 節で Maglog について説明し，3 節で Maglog の実装について述べる．4 節で記述例，5 節で実験結果を示す．6 節で関連研究との比較を行ない，7 節で本稿をまとめる．

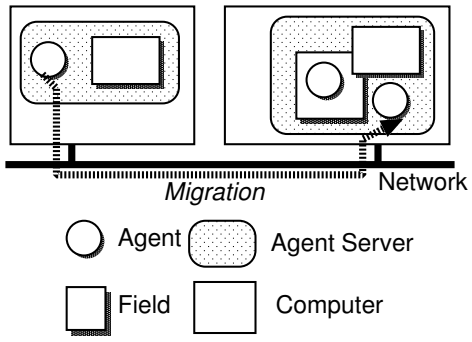

図 1 Maglog のモデル
Fig. 1 Model of Maglog

2. モバイルエージェントフレームワーク Maglog

Maglog は論理型言語 Prolog に基づくモバイルエージェントフレームワークである．Prolog はユニフィケーションと呼ばれる強力なパターンマッチング機構と，バックトラック機構を持ち，エージェントの推論機構や，プランニング機構を記述しやすい．

Maglog においてエージェントはエージェントサーバ上で動作する．エージェントはネットワークで繋がれたエージェントサーバ間を移動し，サーバの提供するフィールドを利用できる．フィールドは Prolog 節を格納することができ，エージェントはフィールドに入ることでその Prolog 節を利用できる．

2.1 エージェント

エージェントは以下の組み込み述語を用いて作られる．

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

`File` にはエージェントのプログラムが記述されているファイルの名前が与えられ，`Goal` には生成されるエージェントの初期ゴールが与えられる．`AgentID` には生成されたエージェントの ID が返される．生成されたエージェントは `Goal` の実行を終えると消滅する．この組み込み述語を実行する側のエージェントは，生成されたエージェントの実行の終了を待たずに次の実行に移る．

図 2 に示しように，エージェントは Prolog プログラムと Prolog エンジンから成り，`Goal` で与えられたゴールを実行する Prolog インタープリターである．

以下の組み込み述語を実行することで，`AgentID` で指定されるエージェントの実行を強制終了させることができる．

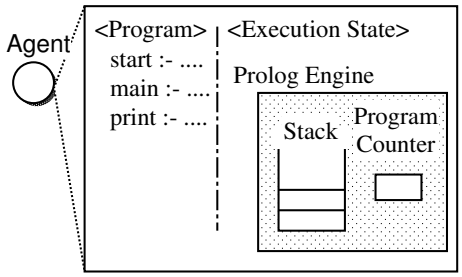

図 2 Maglog agent の構造
Fig. 2 Structure of Maglog agent

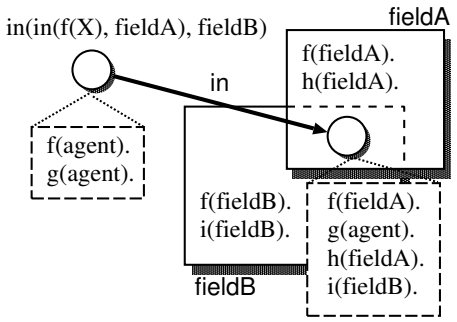

図 3 複数のフィールドに対する in/2 の動作
Fig. 3 Behaviour of in/2 with multiple fields

```
kill(AgentID)
```

エージェントは，

```
get_id(AgentID)
```

とすることで自らの ID を取得することができる．

2.2 フィールド

フィールドは以下の述語を用いて作られる．

```
fcreate(FieldID)
```

FieldID に atom が与えられている場合には，その ID を持つフィールドが作られる．すでのその ID を持つフィールドが存在する場合には何もしない．FieldID に何も割り当てられていない場合には，エージェントサーバー内でユニークな ID とそのフィールドが生成され，ID が返される．

```
in(Goal, FieldID)
```

は FieldID で指定されるフィールド内で Goal を実行する．この時，まず FieldID に Goal が存在すればそれが実行され，無ければエージェントのプログラムから探される．

エージェントは同時に複数のフィールドに入ることができる．

```
in(in(Goal, FieldID1), FieldID2)
```

とすると，FieldID1，FieldID2, エージェント，の順に Goal が探される．この時 FieldID1，FieldID2 内に Goal が存在していても，一つの述語として認識されない．最後に入ったフィールド，すなわちネストしている in の最も内側のフィールドの手続きだけが実行される．Goal のサブゴール以降は，Goal が見付かったフィールド内でのみ実行される．

図 3 の場合，f(fieldA) が f(fieldB) と f(agent) を隠している．

エージェントはフィールドを用いて動作を動的に変更することができる．**図 4** にその例を示す．この例では，print('Hello!') の実装が入るフィールドごとに

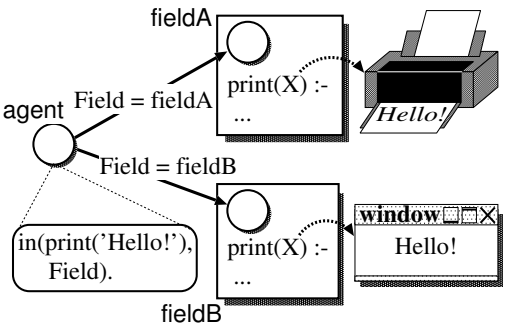

図 4 フィールドによるエージェントの動的な動作変更
Fig. 4 Dynamic change of behaviour of agent with fields

異なっている．fieldA ではプリンタに出力し，fieldB ではウィンドウに表示する．

in/2 によるフィールドの手続きの実行は，ホストをまたいだバックトラックが可能である．**図 5** の例で，エージェントはバックトラックの際に，host1，host2 間を移動し，フィールドに再び入っている．プログラマはバックトラックの際のホスト間移動やフィールドに対する操作を，記述する必要はない．

```
fassert(Clause, FieldID)
fretract(Clause, FieldID)
```

によってフィールドに Prolog 節を追加，削除できる．FieldID は Clause で表される節を追加，または削除するフィールドの ID である．fassert は常に成功する．fretract は Clause に対応する節が FieldID で指定されるフィールドに存在する場合成功し，存在しない場合はエージェントのモードによって動作が異なる．

エージェントはフィールドに関する以下の 3 つのモードの内，いづれかのモードになっている．

- error モード

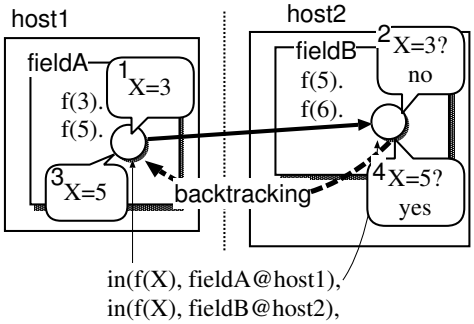

図 5 ホストをまたいだバックトラック
Fig. 5 Backtracking between hosts

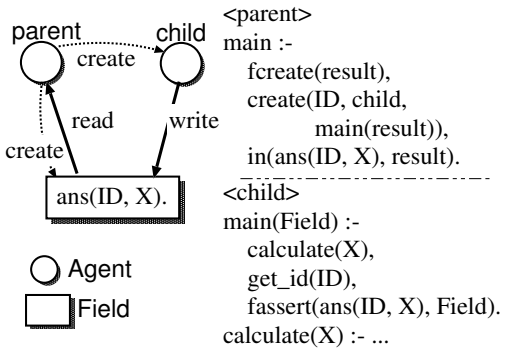

図 6 フィールドを用いた同期通信
Fig. 6 Synchronous communication via field

- fail モード
- block モード

error モードのエージェントは実行を終了する．fail モードのエージェントは失敗する．block モードのエージェントは `Clause` に対応する節が他のエージェントによって追加されるまでブロックされる．

上記の述語を用いて，エージェント間コミュニケーションが実現できる．エージェントが子エージェントを生成し，子エージェントが結果を返すまで待つ例を**図 6** に示す．

3. Maglog の実装

1 節で述べたように，モバイルエージェントシステムを実装する環境として Java が適している．そこで，Prolog から Java へのトランスレーターシステム PrologCafé[5] を利用し，Maglog を Java 環境に実装した．エージェントサーバーは Java で書かれており，エージェントは PrologCafé の Prolog インタープリターを拡張し，エージェントサーバー内で動作するスレッドとして実装した．Java RMI を用いてエージェントのマイグレーションを実現した．

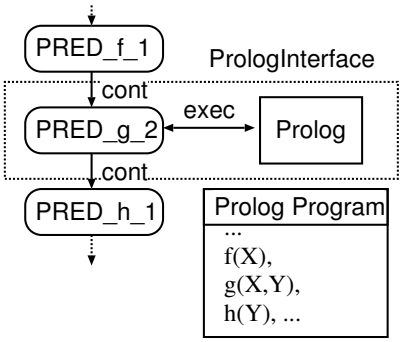

図 7 PrologCafé での Prolog プログラムの実行
Fig. 7 Execution of prolog program in PrologCafé

この節では，PrologCafé の概要について触れ，Maglog での手続きの実行方式について述べた後，マイグレーション，フィールドの実装について述べる．

3.1 PrologCafé

PrologCafé は Prolog から Java へのトランスレータである．PrologCafé においてプログラムの実行は大体以下のクラスによってなされる．

- Prolog
- Predicate
- PrologInterface

Prolog クラスは Prolog エンジンであり，チョイスポイントスタックやトレイルスタックなどを管理している．Predicate クラスは抽象クラスであり，`Predicate exec(Prolog engine)` メソッドが定義されている．PrologCafé のトランスレータは Prolog 節を Predicate クラスのサブクラスに変換する．例えば，`append(X,Y,Z)` は Predicate クラスを継承する `PRED_append_3` クラスに変換される．Predicate のサブクラスの `exec` メソッドは次に実行すべき Predicate を返す．PrologInterface クラスは**図 7** のように，Prolog クラスのオブジェクトと手続きである Predicate クラスのオブジェクトを持ち，それらを組み合わせて Prolog プログラムの実行を行なっている．

PrologCafé の Prolog インタープリターのプログラム実行方法は 2 通りある．プログラムを `Predicate` クラスに変換して実行する方法と，インタープリターの実行後にプログラムを consult して実行する方法である．

前者の実行方式をプログラムの Java レベルでの実行，後者を Prolog レベルでの実行と呼ぶことにする．

3.2 手続きの実行方式

3.1 節で述べた 2 つの実行方式の実行速度の差を，

表 1 2 つの実行方式の実行速度の比較
Table 1 Comparison of execution times of 2 execution methods

	Java level	Prolog level
8-queens	181	48226

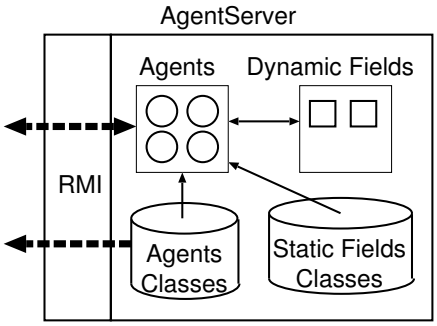

図 8 エージェントサーバの構造
Fig. 8 Structure of agent server

8-queens 問題によって検証した．**表 1** に示すように，約 250 倍の速度向上が見られた．

そのため，Maglog では，エージェントのプログラムをあらかじめ `Predicate` クラスに変換する方式を採用した．また，フィールドも静的フィールドと動的フィールドに区別し，静的フィールドのプログラムも同様の方式を採用することとした．

エージェントのプログラムの実行方式に Java レベルの実行を採用したため，エージェントが本籍地を離れた場合，移動先にはエージェントのプログラムのクラス定義は存在しない．そのため，エージェントが実行しようとした手続きの定義が存在しない場合，**図 8** に示すように，本籍地サーバーに必要な手続きのクラス定義を転送してもらうように実装した．この動作は透過的であり，エージェントのプログラマはこの動作を意識する必要は無い．

3.3 マイグレーション

Maglog エージェントの移動を Java RMI を用いて実現した．他のホストへ移動する為の述語として，`$go(Dest)` という組み込み述語を作成した．この述語は，`in/2`，`fassert/2`，`fretract/2` の内部で利用されており，`Dest` に与えられたアドレスが示すホストへ移動する．

3.1 で述べたように，PrologCafé では手続きを表わす `Predicate` オブジェクトが次の `Predicate` オブジェクトを指している．`$go(Dest)` 内では，エージェントが持っている実行中の `Predicate` へのリファレンスを`$go/1` の次の `Predicate` オブジェクトに変更し，目的のサーバーにエージェントを転送する．したがって，目的地では`$go/1` の次の手続きから再開されることになる．このようなマイグレーションの方式は強マイグレーションと呼ばれ，エージェントの実行状態がエージェントと共に転送されるため，Maglog では，ホストをまたいだユニフィケーションが可能となる．

```
public synchronized void assertz(Term t) {
    t をテーブルに追加;
    notifyAll();
}
```

図 9 assertz

```
synchronized Term retract(Term t,
                          boolean block) {
  matched = [];
  new_list = [];
  do {
    matched = t とユニファイ可能な Term のリスト;
    new_list = t とユニファイ可能でない Term のリスト;
    if(block && matched == [])
      wait();
    else
      break;
  } while(block && matched == []);

  テーブルを new_list で更新;
  return matched;
}
```

図 10 retract

3.4 フィールド

DynamicField クラスは，動的フィールドを実装している．DynamicField クラスには，Prolog 節の追加，削除を行なう，

- `assertz`
- `retract`

などのメソッドが用意されている．それぞれのメソッドは**図 9**, **図 10** のように実装されている．Java スレッドモデルを利用して，エージェント間の同期を可能にした．

静的フィールドは，エージェントのプログラムと同様に Java クラスに変換され，エージェントサーバーが管理する．静的フィールドのクラスは，`in/2` の内

部でロード，インスタンス化され，実行される．

4. 記 述 例

4.1 契約ネットプロトコル

契約ネットプロトコル[6] は，仕事を入札形式で他のエージェントに委託するためのプロトコルである．Maglog による契約ネットプロトコルの記述例を付録 A.1.1 に示す．

この例ではフィールドが契約ネットプロトコルを実現するための手続きを含んだライブラリとして提供される．また，ID を含んだ事実節をフィールドを介してやりとりすることにより，エージェント間で，相手エージェントを特定しながらメッセージ交換が行なわれる．

4.2 ISBN 検索

本の著者とタイトルが分かっている場合に，その本の ISBN を検索する記述例を付録 A.1.2 に示す．

この例では，3 つのフィールドが様々な形式でデータベースを持っており，`search` という共通の手続きを持つ．そのため，検索するエージェントは `search` という手続き名だけ知っていればよく，それぞれのフィールドがどのようにデータベースを構築しているかを知る必要はない．また，フィールドの利用はホスト間の移動も伴なうため，プログラマはホスト間移動とフィールドの利用を別々に考える必要が無く，プログラムの複雑さを減少させることができる．

5. 実 験

100Mbps のイーサネットで接続された 2 つのコンピュータ間を 100 往復するエージェントを，Maglog と Aglets[2] で記述し，比較した．結果を**表 2** に示す．

Maglog は強マイグレーションをサポートしており，エージェントは Prolog で記述されるため，プログラムサイズは Aglets のものと比べて少なくなっている．Aglets は弱マイグレーションのみをサポートしている．

強マイグレーションであることから，Maglog ではエージェントのサイズがより大きくなっている．これは，エージェントの内部にエージェントの実行状態を保存するオブジェクト，すなわち Prolog エンジンが存在するためである．

しかし，Maglog のエージェントのサイズは Aglets のエージェントより大きいにもかかわらず，移動時間は Aglets のそれより少ない．その理由として，エージェントを他のホストへ転送する際には，シリアライズされたエージェントのデータを転送することよりも，エージェントのシリアライズ/デシリアライズ，スレッドの生成により多くの時間が掛かっていると考えられる．

表 2 Maglog と Aglets の比較
Table 2 Comparison between Maglog and Aglets

	Maglog	Aglets
プログラムサイズ (行)	5	41
プログラムサイズ (bytes)	93	1047
エージェントサイズ (bytes)	7764	767
移動時間 (msec)	12089	14281

Maglog では強マイグレーションがサポートされている為，エージェントのサイズが大きくなり，ホスト間移動により多くの時間が掛かることが予想されたが，一般的なネットワーク環境ではその影響は少なく，既存のフレームワークと比べ，十分な移動性能が実現されていることが確認された．

実験に用いた Aglets のプログラムを付録 A.2.1 に，Maglog のプログラムを付録 A.2.2 に示す．

6. 関連研究との比較

MiLog[3] は Prolog に基づくモバイルエージェントフレームワークである．エージェント間の通信は HTTP による”query”や”request”などの問い合わせメッセージを，エージェント同士 1 対 1 で交換することによって行なわれる．

MiLog ではエージェント間の通信が 1 対 1 であり，エージェント間での情報の共有が実現しづらい．Maglog では，エージェント間の通信は，フィールドを介して行なわれ，マルチキャスト型，ブロードキャスト型，ユニキャスト型の通信が簡単に実現できる．

Flage[4] もまた Prolog に基づくモバイルエージェントフレームワークであるが，Flage ではエージェントのプログラムは独自の文法で記述される．Flage には場の概念が存在し，エージェントが場に入るための条件を定義する．エージェントは `enter`，`exit` などのメソッドを使って，フィールドに明示的に出入りする．

Maglog ではエージェントは任意のフィールドに自由に入ることができ，`in/2` などの組み込み述語を利用する場合，フィールドへの出入りを意識する必要は無く，プログラマは「あるフィールド内であるゴールを評価する」という目標を 1 命令で記述することができる．

7. おわりに

本稿では，場の概念を持つモバイルエージェントフレームワーク Maglog を紹介した．Maglog は以下の特徴を持つ．

- Prolog に基くため，エージェントの学習，推論，プランニングが記述しやすい
- フィールドを用いてエージェントの動作を動的に変更できる
- フィールドを用いてエージェント間の同期/非同期通信が実現できる
- エージェントの移動方式は強マイグレーションである
- ホストをまたいだバックトラックが実現されている

また，記述例，実験により以下の性質を明らかにした．

- エージェントのプログラムを簡潔に記述できる
- ホスト間の移動が十分効率的である

参 考 文 献

1) 本位田真一ほか: エージェント技術, 共立出版 (2000).
2) Lange, D. B. and Oshima, M.: *Programming and Deploying Java Mobile Agents with Aglets*, Addison Wesley (1998).
3) N, Fukuta. T, I. and T, S.: MiLog: A Mobile Agent Framework for Implementing Intelligent Information Agents with Logic Programming, *Proc. of the First Pacific Rim International Workshop on Intelligent Information Agents*, pp. 113–123 (2000).
4) 粂野文洋, 佐藤仁孝, 加藤哲男, 永井保夫, 本位田真一: エージェント指向言語 Flage, マルチエージェントと協調計算のワークショップ MACC'97 (1997).
5) Banbara, M. and Tamura, N.: Translating a Linear Logic Programming Language into Java, *Proceedings of the ICLP'99 Workshop on Parallelism and Implementation Technology for (Constraint) Logic Programming Languages* (M.Carro, I.Dutra et al.(eds.)), pp. 19–39 (1999).
6) G.Smith, R.: Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver, *IEEE Transactions on Computers*, Vol. C-29, No. 12, IEEE, pp. 1104–1113 (1980).

付　　録

A.1 記 述 例

A.1.1 契約ネットプロトコル

```
%%%%%%%%%% contract.pl %%%%%%%%%%
:-public delegate/3, get_goal/2,
        propose/2, inform/2.
delegate(Goal, Result, Time) :-
    fcreate(cfp),
    get_id(ID),
    fassert(cfp(ID, Goal), cfp),
    sleep(Time),
    noblock(
        findall(
            proposal(Wkr, Cost),
            fretract(proposal(Wkr,ID,Cost),
                     cfp),
            Proposals)),
    fretract(cfp(ID, Goal), cfp),
    sort(Proposals, [Best|Reject]),
    fassert(reply(ID, Best, accept),
            cfp),
    reject(Reject, cfp),
    fretract(inform(ID, Result), cfp),
    Result == yes.
get_goal(Mngr, Goal) :-
    fcreate(cfp),
    in(cfp(Mngr, Goal), cfp).
propose(Mngr, Cost) :- get_id(ID),
    fassert(proposal(ID, Mngr, Cost),
            cfp),
    fretract(reply(Mngr,ID,R), cfp),
    R == accept.
inform(Mngr, Res):-
    fassert(inform(Mngr, Rel), cfp).
reject([], _) :-!.
reject([proposal(Wkr,Cost)|T], Field) :-
    get_id(ID),
    fassert(reply(ID, Wkr, reject),
            Field),
    reject(T, Field).
%%%%%%%%%% worker.pl %%%%%%%%%%
main :-
    in(get_goal(Mngr, Goal), contract),
    calculate_cost(Goal, Cost),
    in(propose(Mngr, Cost), contract),
    process(Goal, Res),
    in(inform(Mngr, Res), contract).
process(Goal, yes) :- Goal, !.
process(Goal, no).
%%%%%%%%%% manager.pl %%%%%%%%%%
main(Goal) :-
    in(delegate(Goal, Result, 3),
       contract).
```

A.1.2 ISBN 検索

```
%%%%%%%%%%% agent.pl %%%%%%%%%%%
book_search(Author, Title, ISBN) :-
    bs(Author, Title, ISBN,
       [fieldA@hostA,
        fieldB@hostB,
        fieldC@hostC]).
bs(Author, Title, ISBN, [Field|Tail]) :-
    in(search(Author, Title, ISBN),
       Field), !.
bs(Author, Title, ISBN, [_|Tail]) :-
    bs(Author, Title, ISBN, Tail).
%%%%%%%%%%% fieldA.pl %%%%%%%%%%%
:- public search/3.
search(Author, Title, ISBN) :-
    book(Author, Title, ISBN).
book(author0, title0, 'ISBN0-0000-0000-0').
book(author0, title1, 'ISBN1-1111-1111-1').
%%%%%%%%%%% fieldB.pl %%%%%%%%%%%
:- public search/3.
search(Author, Title, ISBN) :-
    data(Author, List),
    s(Title, ISBN, List).
s(Title, ISBN, [book(Title, ISBN)|_]).
s(Title, ISBN, [_|Tail]) :-
    s(Title, ISBN, Tail).
data(author0,
     [book(title2, 'ISBN2-2222-2222-2'),
      book(title3, 'ISBN3-3333-3333-3')]).
data(author1,
     [book(title4, 'ISBN4-4444-4444-4')]).
%%%%%%%%%%% fieldC.pl %%%%%%%%%%%
:- public search/3.
search(Author, Title, ISBN) :-
    java_constructor('example.SQL', SQL),
    java_method(SQL,
                query(Author,Title),ISBN).
```

A.2 実験に用いたプログラム

A.2.1 Aglets

```
public class HelloAglet extends Aglet {
    SimpleItinerary si = null;
    String home = null;
    String dest = null;
    int count = 100;
    public boolean
    handleMessage(Message msg) {
        if (msg.sameKind("atHome"))
            atHome(msg);
        else if (msg.sameKind("atDest"))
            atDest(msg);
        else
            return false;
        return true;
    }
    public void onCreation(Object init) {
        si = new SimpleItinerary(this);
        home = getAgletContext().
               getHostingURL().toString();
    }
    public void atHome(Message msg) {
        if(count > 0)
            try {
                si.go(dest, "atDest");
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        dispose();
    }
    public void atDest(Message msg) {
        try {
            count--;
            si.go(home, "atHome");
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
    public void setDest(String dest) {
        this.dest = dest;
    }
}

public class HelloAglet extends Aglet {
    SimpleItinerary si = null;
    String home = null;
    String dest = null;
    int count = 100;
    public boolean
    handleMessage(Message msg) {
        if (msg.sameKind("atHome"))
            atHome(msg);
        else if (msg.sameKind("atDest"))
```

```
            atDest(msg);
        else
            return false;
        return true;
    }
    public void onCreation(Object init) {
        si = new SimpleItinerary(this);
        home = getAgletContext().
               getHostingURL().toString();
    }
    public void atHome(Message msg) {
        if(count > 0)
            try {
                si.go(dest, "atDest");
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        dispose();
    }
    public void atDest(Message msg) {
        try {
            count--;
            si.go(home, "atHome");
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
    public void setDest(String dest) {
        this.dest = dest;
    }
}
```

A.2.2 Maglog

```
main(_, 0):-!.
main(Dest, N):-
    in(true, test@Dest),
    N1 is N - 1,
    main(Dest, N1).
```