システムソフトウェアとオペレーティング・システム 86−12
(2001. 3. 6)

途中報告を行う分散システムフレームワークのアプリケーション

森田 卓也† 山本 英雄‡ 梅村 恭司†

† 豊橋技術科学大学 情報工学系 ‡NTT 情報流通プラットフォーム研究所
†441-8580 豊橋市天伯町雲雀ヶ丘 1-1
TEL: (+81)532-47-0111(ex.5430)
†morita@ss.ics.tut.ac.jp, umemura@tutics.tut.ac.jp

あらまし　エージェントを用いた分散処理は，エージェントに対し仕事を依頼し，仕事がすべて完了してから結果報告をエージェントから受けとる．しかし，分散処理の処理時間が長い場合など，処理途中の状態を知りたい場合があることが多いと我々は考えている．そこで我々は，ある条件が満たされる都度に途中報告を行うというメッセージパターンに注目し，そのための”繁説エージェント”を考案し，途中報告を行う分散システムフレームワークを構築した．実際にこのフレームワークに従ってアプリケーションを作成し，途中報告が有効である分散システムの構築に役立つことを確認した．また，作成した新聞記事検索システムを用いて途中報告処理の有効性を評価した．

キーワード　分散処理，エージェント，メッセージパッシング，Java，情報検索

A Distributed System with Multiple Reply Messages

Takuya Morita† Hideo Yamamoto‡ Kyoji Umemura†

† Dept. of Information and Computer Sciences, Toyohashi University of Technology
‡ NTT Information Sharing Platform Laboratories
†Tempaku, Toyohashi, Aichi, 441-8580, Japan,
TEL: (+81)532-47-0111(ex.5430)
†morita@ss.ics.tut.ac.jp, umemura@tutics.tut.ac.jp

Abstract　Distributed processing systems usually receive results from agents after the agents finish their tasks. However, we frequently need to know the status in the middle of the process. We have proposed “Verbose Agent” who replies in the middle of the process, as a framework for distributed system where replies are issued whenever certain condition is satisfied. We have built some applications using our framework, and show usefulness of our framework. We evaluated our approach who replies in the middle of the process on newspaper-article retrival system.

key words　Distributed Processing, Agent, Message Passing, Java, Information Retrieval

1 はじめに

処理に問題がないかの途中報告を受けることは人間が仕事を分担するときには普通のことであるので，それに対応する分散処理のフレームワークは有効ではないかと着想した．分散処理において，処理の結果を受け取る回数は通常一回であるが，応用によってはそれでは十分でない．たとえば，処理に時間がかかる場合には，途中経過の報告を受け，報告のあとでも処理を継続することが有用である．

そこで我々は頻繁に途中報告を行うソフトウェアの考え方，**繁説エージェント**という考え方を提案してきた．「繁説 (はんせつ)」とは，おしゃべり，多弁などの類義語である．この繁説エージェントは，処理に対する回答を一度にせず，ある条件が満たされる都度に自発的に途中報告を行うソフトウェアである．我々はこの繁説エージェントを利用した分散処理のシステムを作成するために，繁説エージェントを実現するフレームワークを整備した (文献[1])．まず最初に検索問題に応用した例を示し，本フレームワークを用いてどのように途中報告を処理するか述べる．また，構築した新聞記事検索システムで単報告処理と複報告処理の比較実験を行い，本フレームワークを利用して途中報告を行わせることが有効であることを評価した．最後に実際に記述例を示して説明を行う．

2 対象とする分散問題の性質と方針

本研究で対象とする問題の性質を述べる．途中報告は常に有効とも考えられるが，途中報告が有効である条件を示すことはこのフレームワークを利用するかどうかの判定に有用である．途中報告が有効である問題は，**処理時間は無限大，処理結果は複数，途中結果も有効**という3つの条件が当てはまると考えている．

上記の性質を新聞記事検索を例として説明する．

- 処理時間は無限大
  1つの新聞記事リソースは有限であるが，新聞記事リソースは，年度や新聞社などによっていくらでも追加することができる．つまり，ある新聞記事リソースを検索し終ったとしても，次の新聞記事リソースを探しに行くとすれば，いつまでも探し続けることができる．

- 処理結果は複数
  新聞記事検索では，ユーザが欲しい記事に対していくつかの候補記事が得られる．例えば企業合併という入力に対して，銀行の合併記事や鉄鋼会社の候補記事が得られる．

- 途中結果も有効
  あいまいな入力を許すようなコンセプトの検索では，その性質上唯一の解というものは存在しない．つまり，入力に対して99％の類似度で一致する記事があったとしても，その検索過程で見つかった80％の類似度で一致する記事もユーザが求める記事である可能性がある．

我々は上記の性質を満たす問題に対しては，途中結果の報告と打ち切り時間を導入して，速やかにサンプルを得ることや，その区間で何らかの情報を得られることは有効だと考えた．

繁説エージェントを導入して途中報告を行う場合は，途中報告処理を行うために遅延が生じてしまう (図1)．もし報告の処理による遅延が生じないのであれば，途中結果を報告した方が有利であることは明らかである．また，もし処理を無限に待つことができるという仮定があるならば，途中結果を報告しようがしまいが，最終的な結果は同じになる．しかし実際には報告の処理によるコストが存在し，処理を無限に待てるというのは現実的でない．遅延時間や打ち切りを踏まえた上でも途中報告が有効であることは，文献[1]の中でモデルを使って確認したが，本稿では実際に作成した新聞記事検索システムを用いて評価を行っている (第3.4節)．

3 新聞記事検索システム

フレームワークを設計しても，それが実際に十分な機能を提供しているかは分からない．そこで，動作するシステムを作成して確認した．

作成した新聞記事検索システムは新聞記事アーカイブを別々のホストに分散させておき，各ホストに繁説エージェントが移動し，新聞記事検索プログラムを実行する．そして得られた結果は検索がすべて終了するのを待たずに報告される．

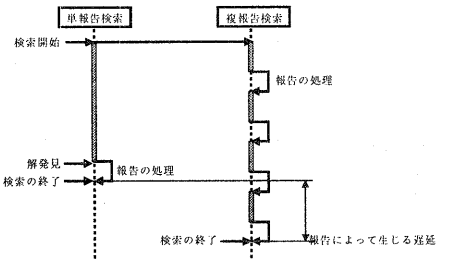

図 1: 単報告処理と複報告処理の遅延

検索アルゴリズムには，DP マッチングを採用している．DP マッチングを用いた検索 (文献 [2]) の特長は，あいまいな入力を許し，検索対象となるリソースも生データに近いものが利用できる点である．しかし，柔軟な入力を許す反面，計算量が多く，過去数年分の新聞記事を検索するというような場合には，多くの時間を要する．検索に多くの時間がかかったとしても，本システムでは途中報告を行なうので，ユーザは検索がすべて終了するまで待たなくてよい．

3.1 システムの動作

新聞記事検索システムの外観を図 2に示す．上部エリアでシステムの操作と query の編集をし，下部エリアで途中報告の閲覧をすることになる．

下部エリアでは，ソートされた途中結果が閲覧できる．右側のアイコンは変化通知オブジェクトである．変化通知オブジェクトは，常に最新のランキングが閲覧できるように，ランキングの変化をユーザに知らせる．あらかじめ上位何件までを注目するか設定しておくことができ，その上位ランキングに変動があったことを動くアイコンで通知する．また，新聞記事には削除ボタンが添えてあり，自分の要求する記事ではなかった場合は，削除することができる．

同時に複数の query で検索を行いたい場合は，新しいウィンドウを開いて行う．

基本的な検索手順は次のようになる．

1. query の作成 (ファイルから読み込み)
2. 報告条件の設定
3. 検索の開始
4. 途中報告の読み込み，閲覧，削除

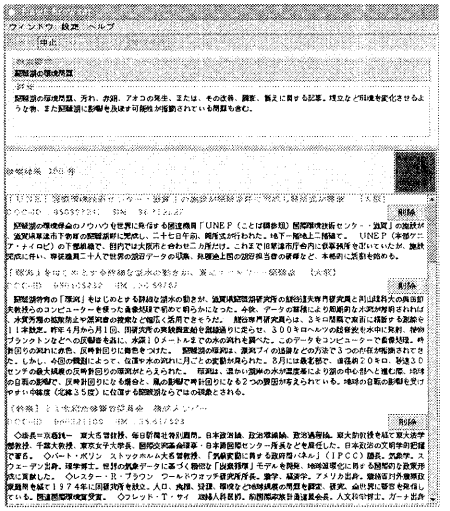

図 2: GUI

5. 検索の中断

繁説エージェントを利用した効果は報告条件が自然に設定できることである．報告条件というのは，ここでは途中報告される結果の最低スコアと途中報告を行う間隔である．この報告条件に従って繁説エージェントは，途中報告を行うことになる．報告条件は分散処理実行中でも変更することができる．つまり報告を行う最低スコアの設定が高すぎて，なかなか結果が得られない場合は最低スコアの設定を下げてやり，逆にあまり関係のない結果が多く得られる場合には最低スコアの設定を上げてやることができる．

3.2 エージェントフレームワークのクラス

本フレームワークには以下の 6 つの重要なクラスが存在する．以下にそれぞれのクラスについて説明する．

1. ワーカ (Worker):
   ネットワーク上に存在する仕事場である．主な役割は，繁説エージェントの受け入れと，移動の記録である．
2. ジョブ (Job):
   本フレームワークでは，繁説エージェントに

よって運ばれる処理をジョブと呼ぶ．繁説エージェントによって運ばれたジョブは，移動したワーカ上で実行される．エージェントを別のワーカに移動させる処理もジョブの一つである．

3. 繁説エージェント (Verbose Agent):
本フレームワークでは，分散させたい処理を繁説エージェントによって分散させる．繁説エージェントの役割は，単にユーザから受け取ったジョブを持って，ワーカ間を移動することである．そして，ワーカで実行されているジョブの途中結果を，報告条件に従ってホームワーカや受取サーバに報告する．

4. 報告条件 (Report Condition):
一般的な分散システムフレームワークにはない概念である．報告条件には，途中報告を行なうタイミングや，ある条件を満たした結果のみを報告するための条件を設定する．この報告条件は，繁説エージェント (またはジョブ) に設定でき，動的に変化させることができる．そしてワーカに一時的に貯められている途中結果は，報告条件に従って報告されていく．

5. 報告マネージャ(Report Manager):
途中報告の受け取り側を整備するために必要となる．報告マネージャに後処理 (ソートなど) を設定しておくことで，自動的に後処理を行ない，途中報告を自動的に整理して保持する．また，報告マネージャを複数設置することで，異なった仕事に対する途中報告を個別に管理することができる．

6. 変化通知オブジェクト (Notice Object):
ユーザに途中報告の変化を通知するためのオブジェクトである．得られる途中報告の中で，あらかじめユーザが注目する範囲を決めておき，その範囲に変化があったことをアイコンを変化させて知らせる．この概念は途中報告のフィルタの役割を果たし，ユーザに余分な情報を伝えない役割を持っている．

3.3 システム構成

新聞記事検索システムは図3のように構築される．本フレームワークは HORB(文献 [3]) を用いた Java によって記述している．各オブジェクトは本フレームワークで用意されたクラスを継承することで実現されている．繁説エージェントは，分割・分散された新聞記事アーカイブが存在するホストへ移動し，新聞記事検索プログラムを実行する．そして報告条件に従って途中報告を行なう．各クラスの具体的なメソッドについては記述例を後述することにする．

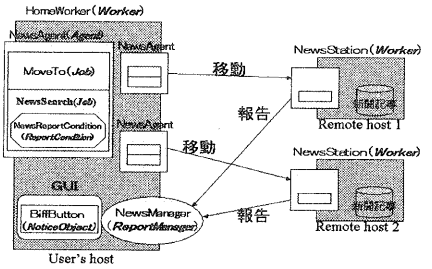

図 3: 構成図

3.4 評価実験

このフレームワークの特徴である報告条件オブジェクトの有効性を示す実験を行った．

3.4.1 実験環境

新聞記事アーカイブは，毎日新聞 95,96,97 年度分 (約 370MB) を，年度毎に各ホストに分散した．これらのホストへ繁説エージェントを送り込み，検索の途中結果を報告させて実験を行った．マシンのスペックはすべて PentiumIII800MHz，メモリ 1GB で，100Base-T で接続された LAN 環境である．

3.4.2 途中報告のある場合とない場合の比較

比較の結果は図 4に示す．この表の横軸は経過時間，縦軸はそれまでに報告された上位 100 件のスコアの平均である．single-report は単報告処理で各ホストで検索がすべて終了してからまとめて結果を報告した場合のグラフである．conditional-report と all-report は，複報告処理で 30 秒間隔で途中報告を行った結果である．conditional-report と all-report

の違いは，all-report はすべての結果の報告を行ったのに対し，conditional-report はスコアが 15 以上のものだけ報告を行ったことである．

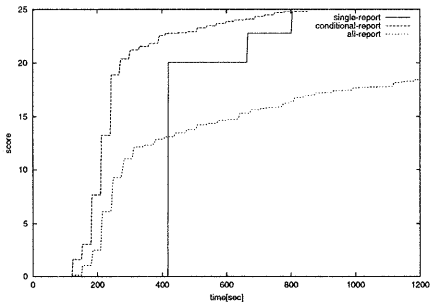

図 4: 比較結果

図 4をみると，最初から 130 秒ぐらいまでは新聞記事の suffix array を作成しているため報告の処理は行われていない．まず，all-report はすべての途中結果を報告してるため処理時間がかかっている．よって single-report で最低でも一つのホストでの検索が終了するのを待てるならば，single-report の方が良いことになる．これは選別を行わない途中報告では良い効果を生まないことを示している．また conditional-report は single-report よりも，常に早い段階で高いスコアを得ていることが分かる．つまり，報告条件が適切であれば，どの時点でもユーザはより有益な結果を得ていることになる．

この結果から，実際の応用システムにおいても途中報告処理は有効であることが観測できた．かった．また，途中報告処理の一番の問題である遅延問題も，報告条件に不必要な情報を報告しないように調整でき，情報検索の応用では，事実上問題にならないことも観測できた．

3.4.3 打ち切り時間までの期待値の比較

図 4の比較が一回の実験の偶然の結果でないことを示すため，今度は打ち切り時間をパラメータと考えた場合に，単報告処理と複報告処理で打ち切り時間内に得られる結果スコアの期待値を比べた．文献 [1] では検索モデルを作成してシミュレーションを行っていたが，本稿では実際に作成した新聞記事検索システム上で実験を行った．

単報告処理では得られるすべての結果のうち上位 5 件を正解とし，打ち切り時間内に正解を得ていればスコアは 100，得ていなければスコアは 0 とした．複報告処理では打ち切り時間内で見つかった結果の最大スコアがその時点でのスコアとした．この複報告処理は前の実験と同様に報告条件によってスコア 15 以上の結果を 30 秒間隔で報告した場合である．それぞれの処理を 100 回ずつ繰り返し (毎回各ホスト毎に保持している新聞記事のシャッフルを行う)，各打ち切り時間までのスコアの平均を期待値としてグラフに示してある (図 5)．グラフの横軸は打ち切り時間で，縦軸はその打ち切り時間までに求まるスコアの期待値である．ただしグラフでは複報告処理の期待値も最大 100 になるように正規化を行った値がを記してある．

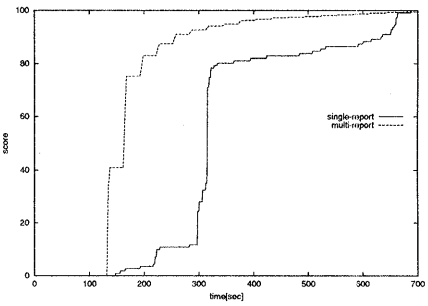

図 5: 実システムでの打ち切り時間までの期待値

まずグラフを見ると複報告処理 (multi-report) が常に高い期待値を示しているのが分かる．文献 [1] でのシミュレーション結果では，途中報告を行うことに結果発見間隔の 8 倍もかかる場合を想定していたため，最初の途中報告があるまでは単報告処理の方が有効である結果を示していた．実システムでは，報告条件によって不必要な報告を取り除いているためにほとんど遅延が生じず，常に複報告処理が高い期待値を得ることができている．また両処理とも 0 秒から 130 秒まで報告がないが，これは検索を行うために新聞記事の suffix array を作成しているためであり，実システムで検索を開始するまでの前処理である．

4 具体的記述例

新聞記事検索システムはフレームワーク以外の記述も多いので，単純なシステムを用いて記述の例を示す．計算機負荷監視システムを作成する場合を想定した．

4.1 計算機負荷監視システム

途中の報告が本質の役割をする例として，計算機負荷監視システムに応用した場合の記述例を示す．この計算機負荷監視システムは，繁説エージェントをリモートホストに送りこむことによって，各ホストのCPU稼働率とメモリ残量を一括してモニタリングするものとする．

4.1.1 システムの記述

本フレームワークに従って記述を行えば，途中報告を行う分散システムを作成することができる．基本的にはフレームワークで用意されているクラスで空の実装となっている部分について記述を行えばよい．表1に各クラスで実装しなければならないメソッドを列挙する．以下で各オブジェクトの記述の仕方を示す．

4.1.2 ジョブの記述例

計算機負荷監視システムのジョブは，2つ存在する．一つは，本フレームワークでは恐らくどんなシステムを構築するときも，使われるであろう，エージェントの移動(MoveTo)のジョブである(図6)．もう一つは，計算機の負荷を監視するジョブで，CPU稼働率，メモリ残量の監視(LoadObserver)を作成する必要がある(図7)．図では本フレームワークが用意しているクラス及びインタフェースは太字，メソッドは斜字，そして利用するに当たって必ず記述しなければならないメソッドには下線を引いた．なお紙面の都合上重要な部分しか記述していない．

まずMoveToは繁説エージェントの移動という短時間の動作であるので，同期ジョブとして実装される．それに比べLoadObserverはワーカ上で常に動作してCPU稼働率とメモリ残量を定期的にチェックを行うので非同期ジョブとして新しいスレッドとして実装される．

```
public class MoveTo extends JobAdapter{
  WorkerAdapter_Proxy remote;

  public MoveTo(String dest){
    this.dest = dest;
  }

  public void exec(Agent agent){
    ownerAgent = agent;
    while(home.enableJump() == false){
      Thread.sleep(1000);
    }
    remote = new WorkerAdapter_Proxy("horb://"+dest);
    ownerAgent.setFootPrint(remote);
    remote.startJob(ownerAgent);
  }
}
```

図 6: MoveTo の記述例

```
public class LoadObserver
      extends JobAdapter implements Runnable{
  boolean finishFlag = false;

  public void run()
    Worker worker = ownerAgent.getCurrentWorker();
    Vector results = worker.getResultPool();

    while(finishFlag == false){
      float cpu_load = getCPU_load();
      float memory_load = getMemory_load();
      results.addElement(new Load(cpu_load, memory_load));
      Thread.sleep(1000);
    }
    worker.observerStop();
    ownerAgent.reportRequest(LoadReceiver.END);
    if(ownerAgent.hasMoreJob()){
      worker.execNext();
    }
  }
```

図 7: LoadObserber の記述例

表 1: 空の実装となっているメソッド

クラス	メソッド	説明
Worker	receiveReport()	ワーカで報告を受け取る場合
Agent	reportRequest()	報告の処理を記述
Job	exec(),run()	処理の内容を記述
ReportCondition	checkResultPool()	レポート プールをチェックする
ReportManager	checkString()	受け取った報告をチェックする
NoticeObject	update()	状況に変化があったとき呼ばれる

図 6をみて分かるように，繁説エージェントの移動の手続きを exec() メソッドに記述してある．この exec() メソッドが Worker#startJob(Agent) によって呼ばれ，繁説エージェントの移動が行われるのである．LoadObserver では，実際に仕事をどのように行うかの記述を，run() メソッドに記述している．この例では getCPU_load(),getMemory_load() という CPU 稼働率とメモリ残量の取得をするメソッドを 1 秒間隔で呼び出している．そして得られた結果はリザルトプールに追加を行っている．

MoveTo はどのようなシステムでも利用するので，常にこれと同じように記述をすればよい．また LoadObserver の CPU 稼働率とメモリ残量の取得に関するメソッドはこのシステム固有の仕事であるので，1 から作成しなければならない．

4.1.3 報告条件の記述

報告条件は，CPU 稼働率，メモリ残量が一定値を越えたら報告するようにした (LoadReportCondition)．記述例は図 8のようになる．

フレームワーク利用者が記述しなければならないのは，checkResultPool() メソッドである．このメソッドはコンストラクタの引数として受け取る時間間隔ごとに呼び出されることになる．

```
public class LoadReportCondition
      extends ReportConditionAdapter{
  float cpu_limit;
  float memory_limit;

  public void checkResultPool(){
    Enumeration enu;
    Load load;
    if(resultPool.size() > 0){
      synchronized(resultPool){
        enu = resultPool.elements();
        while(enu.hasMoreElements()){
          road = (Road)enu.nextElement();
          if(road.cpuLimit() > cpu_limit ||
              road.memoryLimit() > memory_limit){
            reportTo.reportRequest(road.toString());
          }
        }
      resultPool.removeAllElements();
    }
  }
}
```

図 8: LoadReportCondition の記述例

4.1.4 報告マネージャの記述

報告マネージャの記述 (LoadReportManager) は図 9のようになる．ここでは checkString() メソッドで，報告の内容を区別している．

```
public LoadReportManager extends ReportManager{
  static final String END = "end";

  void checkString(String s){
    if((s.indexOf("<LOG>")) == 0){
      return;
    }
    if((s.indexOf("<LOAD>")) == 0){
      try{
        Load load = new Load(new String(s.getByte()));
        addReport(load);
      }catch(Exception e)
      broadcast("NewReport");
      return;
    }
    if((s.compareTo(Finish)) == 0){
      broadcast("Finish");
    }
  }
}
```

図 9: LoadReportManager の記述例

4.1.5 繁説エージェントとシステムの実行の記述

繁説エージェントの記述 (LoadAgent) とエージェントの作成，ジョブの格納，実行，の記述は図 10の

ようになる．

```
public LoadAgent extends AgentAdapter{
  public void reportRequest(String mesg){
    try{
      display.write(mesg);
      display.newLine();
      display.flush();
    }catch(Exception e){
        e.printStackTrace();
      }
    }
  }
}

Worker homeWorker = new WorkerAdapter();
ReportCondition cond = new ReportCondition(interval);
Agent agent = new LoadAgent(local_host, port);
agent.add(new MoveTo(remote_host));
agent.add(new LoadObserver());
agent.setReportCondition(cond);
agent.setWorker(homeWorker);
homeWorker.startJob(agent);
```

図 10: LoadAgent の記述例

繁説エージェントの記述 (LoadAgent) では，reportRequest() メソッドを記述しなければならない．繁説エージェント移動用のジョブである．このジョブは頻繁に利用されるので，フレームワークであらかじめ用意されている．

4.2 システム作成者の労力

以上のような記述を行うだけで，途中報告を行う分散システムを記述することができる．結局システム作成者は計算機負荷監視プログラムとその結果を表示する GUI のみを考えるだけで良い．その他は上記で示したようにフレームワークに従って個々のパラメータなどを設定するだけである．

本フレームワークを利用しないと，分散させた処理の管理や報告に関する管理をすべて記述せねばならず，作成者に負担がかかる．

5 まとめ

繁説エージェントを用いた分散フレームワークに従ってシステムを実際に作り，途中報告フレームワークの有効性を検証した．本フレームワークに従って記述することで途中報告処理が上手に行えることを示した．また新聞記事検索システムでは単報告処理と複報告処理の比較実験を行って，途中報告が有効な条件を明らかにした．

また，本報告では「検索タイプの問題において，ある有限時点での効用を向上させる」ということを目的としている．この目的を達成するだけならば，多くの分散処理システムにおいて，途中結果を報告するようにプログラムを記述することができる．途中報告を行うプログラムを可能としただけでは貢献とはいえない．しかし，途中報告に注目し，そこに必要な機能を点検し，効果を確認したことは本報告の貢献と考える．

参考文献

[1] 山本英雄，梅村恭司: 繁説エージェントによる分散システムフレームワークと情報検索システムの構築, 情報処理学会研究報告 99-OS-81, pp131-136, 1999.

[2] 山本英子，梅村恭司: 日本語のダイナミックプログラミングでの検索方法, 情報処理学会プログラミングシンポジウム, pp131-142, 2000-1.

[3] 中原真則，平野聡: 飛べ，オブジェクト! HORB プログラミングマジック, bit(共立出版), Vol.28, No.10 pp4-15, No.11 pp53-60, 1996-11.

[4] Jamusz Kowalik: PVM:Parallel Virtual Machine A User's Guide and Thuroial for Networked Parallel Computing, MIT Press, 1994.

[5] 小野沢 博文: 分散オブジェクト指向技術 CORBA, ソフト・リサーチ・ センター, 1996.

[6] Reid G. Smith : Frameworks for Cooperation in Distributed Problem Solving.

[7] 山崎重一朗，津田宏: Telescript 言語入門, アスキー出版, 1996.

[8] Eun-Seok LEE, Roberto Okada, Norio Shiratori: Agent-based Social Information Gathering on Internet, ICMAS-96, p448, 1996.

[9] Joo-Hwee Lin, Jiankang Wu, Siet-Leng Lai: A Multiagent Meeting Organizer that satisfies Soft Constraints, ICMAS-96, p449, 1996.