社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2005－EVA－14 (3)
2005／8／5

マルチエージェントシミュレーション用ツールの実装と評価

村田美穂、上田晴康、今村信貴、小橋博道、門岡良昌、宮澤君夫

株式会社富士通研究所

近年、社会科学の分野でマルチエージェントシミュレーションがよく使われるようになってきた。我々は、このマルチエージェントシミュレーションのプログラム作成とシミュレーション実行を支援するためのライブラリを開発した。このライブラリは、シミュレーションを制御するための機能やモデル構築に必要なコンポーネントを提供するため、モデル作成者のプログラム実装の負担を軽減できる。このライブラリを用いてシミュレーションプログラムを実装し、そのコード量や実行時間などを基に評価を行った。

Implementation and Evaluation of Tool for Multi-Agent Simulation

Miho Murata, Haruyasu Ueda, Nobutaka Imamura, Hiromichi Kobashi, Yoshimasa Kadooka, and Kimio Miyazawa

Fujitsu Laboratories Limited

Recently, multi-agent simulation is actively used in the social science field. We have developed a library in order to provide the environment that enables a user easily to create a multi-agent simulation program and to execute the simulation. The library provides various functions for simulating and necessary components for building a model, the user can build a model without any particular knowledge. We implemented a simulation program with the library, and evaluated the library based on the lines of code and the execution time.

1. はじめに

近年、経済学をはじめとする社会科学分野において、様々な社会現象をシミュレートするためにマルチエージェントシミュレーションが使われている。マルチエージェントシミュレーションとは、エージェント同士が相互作用を繰り返し、それによって引き起こされるマクロな現象の変化をシミュレートする手法である。例えば人間をエージェントとする避難シミュレーション[1]、車をエージェントとする交通流シミュレーション[2]などがある。

マルチエージェントシミュレーションでは多数のエージェントが登場する。またパラメータを変えて大量の計算を行う場合がある。従ってモデルの規模が大きくなると全体の計算量が膨大になるという問題があった。このような計算には、複数のマシンを接続したグリッド環境が有効である。我々は大量のシミュレーションを効率よく行うためのグリッドミドルウェアCyberGRIP[3],[4]を開発しており、このCyberGRIP上で政策のためのマルチエージェントシミュレーションを行い、高い成果を上げている[5], [6]。しかし一方で、モデルを構築する社会学者がプログラミングそのものに慣れていないため、そもそもシミュレーションプログラムを実装する段階で非常に時間がかかっているという問題があった。

そこで我々は、上記のような研究者をターゲットとし、マルチエージェントシミュレーション用のツールを開発した。このツールはモデル構築やシミュレーション制御に必要なコンポーネントをJavaのクラスとして提供するライブラリである。このライブラリを用いることでモデル作成者のプログラム実装の負担をどの程度軽減させることができるか、汎用的なライブラリであるため性能が犠牲になっている部分がないかという点に着目し、評価を行った。本稿では、開発したライブラリの機能と上記評価結果について述べる。

2. 解決すべき課題

本章では、プログラム作成を支援するためのライブラリとして、解決しなければならない課題について述べる。

2.1 プログラム実装の負担の軽減

一般的にマルチエージェントシミュレーションの流れは以下のようになっている。

(1) モデルの初期化：あらかじめ属性(特徴)や行動のルー

ルを定義されたエージェントが、ある環境上に複数配置される（図1参照）。

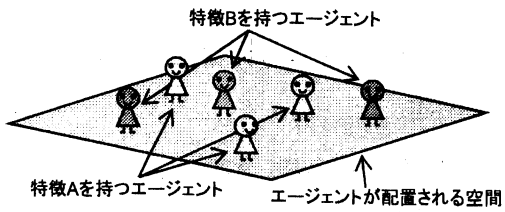

図1 エージェントの配置

(2)1ステップ分の計算:(1)で配置されたエージェント各々が自分の行動ルールに従って行動したり（図2参照）、環境が変化する。

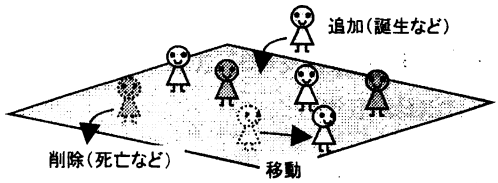

図2 エージェントの行動の例

(3) (2)の計算が繰り返され、エージェント全体や環境に変化が現れる。その変化を観察する。

この流れをプログラムに実装する際には、エージェントや環境をどんなクラスや関数を使って実装すればいいのか、エージェントの追加や削除、行動を矛盾なく行うにはどうすればいいのか、他のエージェントの情報をどのように取得すればいいのかといったことを考える必要がある。さらにシミュレーション実行の途中でパラメータを変えたい、進行状況を可視化したいなどといった要望が考えられる。このような部分はモデルの内容に直接依存せず、マルチエージェントシミュレーションに共通な部分である。これら共通部分の機能を提供できれば、モデル作成者の負担を軽くすることができる。

2.2 より現実世界に近いモデルの構築

社会科学分野でのマルチエージェントシミュレーション研究では、モデルの中に現実世界を再現し、シミュレーション結果を政策に応用したいという要望がある。従ってどれだけ現実に近いモデルを構築できるかということが重要である。例えば人間同士が影響を及ぼしあうことを、モデル上ではエージェント同士の相互作用という形で再現する。現実の人間世界では全ての人間ではなく、近所という物理的に近い人や同年代という境遇の近い人から影響を受けたりするだろう。このような関係をモデル上で再現するために「近傍」という概念がよく使われる。この「近傍」を自由自在に定義できれば、「近所」、「同年代」といった人間関係も再現できると考えられる。

2.3 エージェント同士の衝突・行動の矛盾の回避

マルチエージェントシミュレーションでは、複数のエージェントが自分の行動ルールに従って行動する。この時各エージェントが全く同時に行動すると、以下のような問題が生じる可能性がある。

- 座標が重複し、エージェント同士が衝突する。
- 現在は存在しないエージェントにアクセスする。
- 新たに増えたエージェントに気づかずに行動する。

各エージェントの行動を随時処理させるなら、矛盾が生じないようモデル作成者が気をつけて実装すればよいが、並列処理させるなら排他処理が必要である。全エージェントを監視・管理する機能があれば、排他制御が可能であり、また随時処理するような場合でもモデル作成者が矛盾を気にせずに実装できる。

3. CyberGRIP/MASの実装

第2章で挙げた課題を解決するため、以下の機能を含むマルチエージェントシミュレーション用のクラスライブラリCyberGRIP/MASを実装した。

- 様々なシミュレーション実行を可能にするシミュレーション制御機能
- モデル構築のための基本的な部品であるエージェント・モデル定義機能
- 現実世界に近いモデルの構築を可能にするための近傍定義機能
- エージェント同士の衝突や行動の矛盾を回避するためのエージェント管理機能

CyberGRIP/MASライブラリの構造を図3に示す。このライブラリはシミュレーション制御機能であるMasCommanderとMasController、モデル構築のための各種部品を含むModeling Componentsから構成される。Modeling Componentsにはエージェント・モデル定義機能、近傍定義機能、エージェント管理機能が含まれている。以下、これらの機能について詳細に説明する。

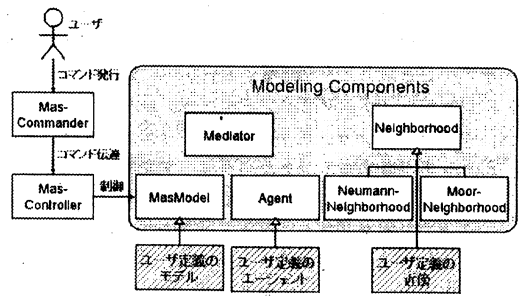

図3 CyberGRIP/MASの概要

3.1 シミュレーション制御機能

MasCommander：MasCommanderはモデル作成者が実装したモデルのクラスをロードし、指定されたMAS実行パラメータをMasControllerに渡す。MAS実行パラメータと

は、モデルに渡す初期値、計算の終了条件、結果を出力するタイミング、ログ出力を指定するためのパラメータである。モデル作成者は実行時にこのパラメータを指定するだけでよい。シミュレーション実行のユースケースを図4に示す。

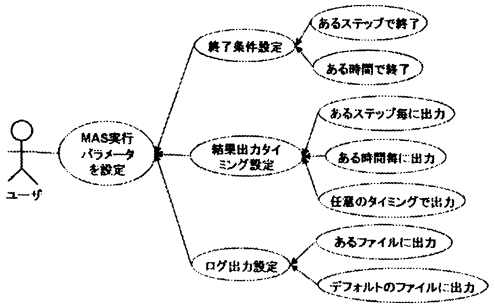

図4 シミュレーション実行のユースケース

MasController：MasControllerはモデルの初期化を行い、終了条件まで1ステップ分の計算を繰り返す。必要に応じて結果や実行ログも出力する。モデルの初期化と1ステップ分の計算の具体的な内容は、後述するModeling ComponentsのMasModelを用いてモデル作成者が実装することになる。

以上2つのクラスがシミュレーションを制御するため、モデル作成者はシミュレーション実行に関する部分を実装する必要がない。

3.2 Modeling Components

モデル作成者はModeling Componentsのクラスを用いてシミュレーションプログラムを作成する。モデル作成者が実装するのはモデルの内容に依存する部分(図3の斜線部)のみである。以下、部品となる各クラスの機能とそれを用いたモデル構築の仕方について述べる。

Agent：Agentはエージェント定義のためのクラスである。モデル作成者はこのクラスを継承した独自のエージェントクラスを実装し、そのエージェント独自の属性を追加したり、具体的な行動ルールを記述する。さらにどの属性を座標とするかを指定すると、エージェントが配置される空間が自動的に定義される。

MasModel：MasModelはモデル全体を定義するためのクラスである。モデル作成者はこのクラスを継承した独自のモデルクラスを実装し、モデルの初期化、1ステップ分の計算の具体的な内容を記述する。

モデルの初期化では予め定義したエージェントを所望の数だけ生成する。その他計算を繰り返す前にしたい処理があれば、この部分に記述する。

1ステップ分の計算では基本的に全エージェントに行動させることになる。後述するMediatorに全エージェントに行動させるよう依頼すれば、全エージェントにアクセスする処理を記述する必要がない。

Mediator：Mediatorはエージェントを管理する機能を持つ。全エージェントを登録したリストを保持しており、エージェントの増減や移動がある度にそのリストを更新する（図5）。エージェントの追加や移動の際に座標の重複のチェックもできるため、モデル作成者はチェック機能を実装する必要がない。また、個々のエージェントは他のエージェントについての情報を持たないため、このMediatorに問い合わせることでその情報を得ることができる。生成したエージェント全てをMediatorに登録することで、上記機能を利用することができる。

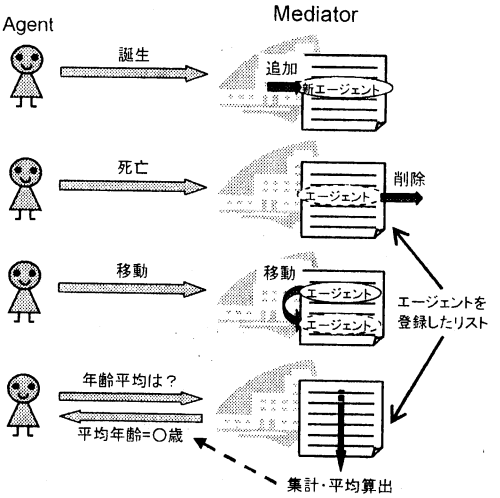

図5 Mediatorへの各種問い合わせや要請

Neighborhood：エージェントの行動ルールとして近傍のエージェント（以下、近傍エージェント）との相互作用を定義する場合、全エージェントの中から近傍エージェントを抽出し、それらにアクセスする必要がある。前述のMediatorがその処理を担当する。「近傍」というだけでは抽出の基準があいまいであるが、例えば図6のように「半径3km以内、年齢差5歳以内」と指定すれば、その条件に当てはまるエージェントを抽出できる。

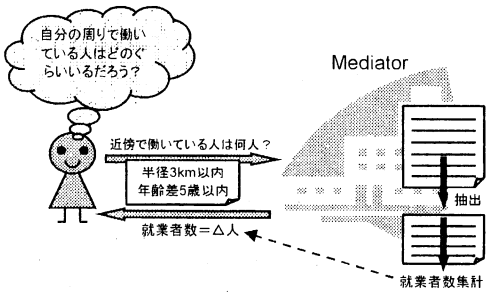

図6 近傍エージェントに関する問い合わせの例

モデル作成者はNeighborhoodクラスを継承した独自の近傍クラスを実装し、予め近傍を定義しておく。具体的

には「半径Akm以内、年齢差B歳以内」といった近傍の条件を記述する。この条件の場合、図7に示すような円柱内に存在するエージェントが近傍エージェントとなる。

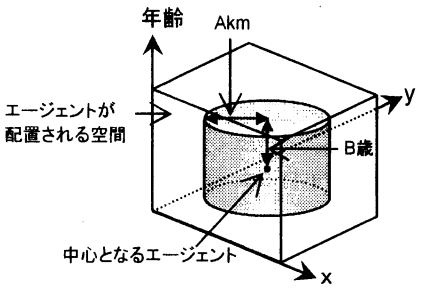

図7 空間内の近傍の範囲

エージェントの行動ルールを記述する際には、定義した近傍クラスのオブジェクトと「3km」、「5歳」といった具体的な範囲の値を指定し、Mediatorへ問い合わせる。Mediatorはそれらの情報を基に近傍エージェントを抽出し、就業者数を集計して返答する。

NeumannNeighborhood、MoorNeighborhood：マルチエージェントシミュレーションではノイマン近傍（自分の上下左右が近傍）やムーア近傍（ノイマン近傍＋斜め方向が近傍）がよく使われるため、CyberGRIP/MASはこれら典型的な近傍を定義したクラスも提供する。これらのクラスを利用する場合は、近傍を新たに定義する必要がない。

4. 評価

4.1 プログラムの規模

CyberGRIP/MASを用いることで、モデル作成者の実装の負担をどれだけ軽減させることが出来るのかを評価するため、あるモデルのプログラムを以下の二通りの方法で実装し、その規模を比較した。

① ライブラリ等を用いずJava言語のみで実装
② CyberGRIP/MASを用いて実装

以下、①の方法で実装したプログラムをプログラム①、②の方法で実装したプログラムをプログラム②とする。モデルは以下の2つを採用した。

ハンターモデル：2次元空間内に複数の獲物とハンターが存在し、ハンターが獲物を見つけて捕獲する。ハンターは見つけた獲物の方向に移動し、ある距離まで近づいたら獲物を捕獲する。獲物はハンターから逃げる。

シュガーモデル：2次元空間内に砂糖の山と複数のアリが存在し、アリが砂糖を求めて動き回る。アリは砂糖が多い場所に移動し、砂糖を摂取する。砂糖を摂取して財産がたまると子アリを生む。移動することで財産を消費し、0になると死亡する。ある年齢に達した場合も死亡する。砂糖は一定周期ごとに再生する。

それぞれのプログラムの行数を表1に示す。なお、プログラム①はCyberGRIP/MASと同等のシミュレーション実行(終了条件や結果を出力するタイミング等の指定)が出来るよう作成している。ハンターモデル、シュガーモデルともに、プログラム②の方が大幅にコード量を減らすことができた。その理由の一つはシミュレーション実行に関する部分の実装が必要ないためである。実際には、1つのモデルの実行パターンはある程度限定されると考えられるので、プログラム①のコード量はこれより少なくなると予想される。しかし何種類かモデル構築とシミュレーション実行を繰り返す過程で実行パターンが増える可能性もあり、その度に新たな機能を追加していくのは大変な作業である。CyberGRIP/MASを用いれば、モデル構築のみに専念できるという利点がある。

表1 プログラム①、②の行数

	プログラム①	プログラム②
ハンターモデル	601	317
シュガーモデル	767	446

2番目の理由は、CyberGRIP/MASを用いれば全エージェントにアクセスする処理をMediatorに任せられるためである。これによるコード量の差は小さいが、プログラム作成者が考慮しなければならない点を減らすことが出来る。ハンターモデルの中でMediatorに任せられる処理をプログラム①、②でどう実装しているかを示す。

エージェント生成・移動時の座標の重複のチェック：
① 生成・移動のたびに他のエージェントの座標を全て調べ、重複をチェックする。
② 生成時にMediatorに登録する。

全ハンター/全獲物エージェントの行動の開始：
① 全エージェントにアクセスし、各自に行動させる。
② Mediatorに全エージェントの行動開始を依頼する。

近傍エージェントの抽出：
① 全エージェントにアクセスして近傍の条件に合うかを調べ、該当するものだけをピックアップする。
② 近傍の種類・範囲を指定しMediatorに抽出を依頼する。

4.2 プログラミングの容易さ

CyberGRIP/MASを用いることで、コード量を減らすことができる以外に以下のような利点がある。

- 近傍エージェントの抽出処理をMediatorに任せれば、正確に抽出できているか確認する作業がほぼ必要なくなる。
- モデルの構造を持つクラスを提供するため、どのような手順で計算を進めればいいのかを気にせずにモデル構築ができる。
- エージェントに必要な属性、機能を持つクラスを提供するため、エージェントプログラムに何が必要かを気にせずにエージェントを定義できる。

以上はJava言語のみでプログラムを作成した場合と比較した結果であるが、CyberGRIP/MASと同様な機能を持つ既存のツールも存在する。例えば、(株)構造計画研究所が開発したツールKK-MAS[7]も近傍エージェントを抽出する機能などを持っている。しかし近傍の定義が固定であるため抽出される範囲がある程度決まってしまい、モデルとして不自然な場合がある。CyberGRIP/MASではモデル作成者自身が近傍を定義できるため、より現実世界に近く自然なモデルの構築が可能である。

4.3 実行時間

CyberGRIP/MASは様々なモデルを構築できるよう汎用的に作られているため、あるモデルに特化して実装したプログラムに比べ、性能的に劣る可能性がある。そこでどの程度遅いか、遅い場合何が原因かを探るため、4.1の評価に用いたプログラム①、②の実行時間を計測・比較した。既存の他のツールは性能よりインタラクティブな実行や可視化を重要としていたり、限定されたパターンのシミュレーションしかできなかったりするため、CyberGRIP/MASと直接性能を比較することは難しい。従ってCyberGRIP/MASと同じJava言語で実装したプログラム①を比較対象とした。実行時間は10回計測の平均値である。実行環境を表2に、ハンターモデルの初期値と実行条件を表3に示す。

表2 実行環境

CPU	Intel Pentium 4　3.2GHz
Memory	1 GB
OS	Windows XP Professional
Java	J2SE 5.0

表3 ハンターモデルの初期値と実行条件

空間の大きさ	500×500（固定）
エージェント数	100～1000（ハンター：獲物＝3：7）
計算ステップ数	100～1000
結果の出力	10ステップ毎（固定）

4.3.1 計算ステップ数と実行時間の関係

エージェント数を500に固定し、計算ステップ数を100～1000まで変化させた場合の実行時間を図8に示す。プログラム②の実行時間はプログラム①の約1.6倍であり、計算ステップ数が多いほど差が大きい。CyberGRIP/MASでは、計算を始める前にユーザが実装したクラスをロードするなどの前処理が入るが、計算ステップ数が増えるに従って差が広がっていることから、むしろ1ステップ分の計算時間の差が効いている。プログラム②の1ステップ分の計算に用いているMediatorでは、エージェント管理のためJavaの様々なCollectionを採用している。これらは更新が多い場合に高速なものと検索が多い場合に高速なものが混在しているため、モデルによっては遅い場合がある。一方プログラム①では検索が多い場合に高速なCollectionを用いている。ハンターモデルは検索が多いためプログラム①の方が高速になっていると考えられる。

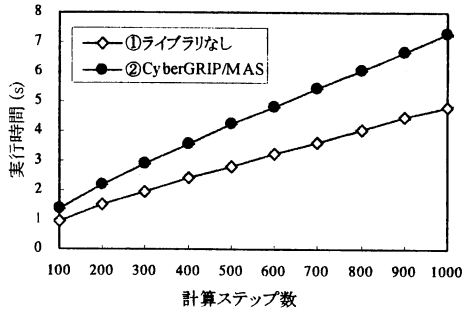

図8 ハンターモデルの計算ステップ数と実行時間の関係

4.3.2 エージェント数と実行時間の関係

モデルの規模が実行時間に与える影響を調べるため、計算ステップ数を500に固定し、エージェント数を100～1000まで変化させた場合の実行時間を計測した。そのグラフを図9に示す。プログラム②の実行時間はプログラム①の約1.6倍であり、エージェント数が多いほど差が大きい。これは前述したようにプログラム①が用いているCollectionが高速であり、エージェント数が多いほどその効果が効いているためと考えられる。

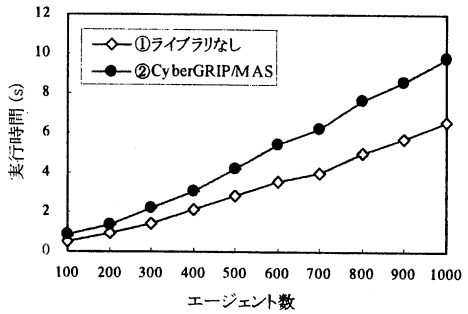

図9 ハンターモデルのエージェント数と実行時間の関係

一方、CyberGRIP/MASのMediatorは座標毎にエージェントを管理しているため、座標を基にエージェントを取得する処理を高速に行うことができる。例えば「自分の隣に他のエージェントがいないか調べる」などの処理である。「隣」を座標で表現することができれば、その座標をキーとしてMediatorがエージェントを取得し、取得できなければその座標のエージェントはいないと判断できる。そこで、ハンターモデルのハンター・獲物が移動する際の行動ルールに「自分の隣に他のエージェントがいないか調べる」というルールを追加し、実行時間を計測した。新しいルールを追加する前後の、エージェントの移動時のルールは以下のようになる。

追加前：常にエージェント毎に決められた速度で移動。
追加後：自分の隣（上下左右斜め）に他のエージェントがいないか調べ、いなければエージェント毎に決められた速度で移動、いればその半分の速度で移動。

エージェント数を100～1000まで変化させた場合のグラフを図10に示す。エージェント数が多い場合、プログラム①の実行時間はルール追加前に比べて大幅に増加しているが（1000エージェントで約2倍）、プログラム②の実行時間は大きく増加していない（1000エージェントで約1.3倍）。しかもエージェント数が増えるほど両者の差は小さくなっている。これは隣のエージェントの所在を調べる際、プログラム②では座標をキーとしてエージェントを一回で取得するのに対し、プログラム①は全エージェントの座標を調べ、自分の「隣」の座標のエージェントを探しているためである。

このように、座標を基にエージェントを取得したい場合、CyberGRIP/MASのMediatorが有効である。

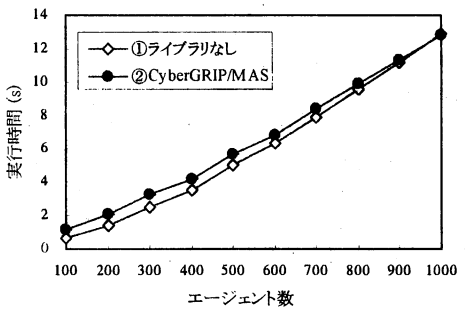

図10 ハンターモデル（新ルール追加後）のエージェント数と実行時間の関係

4.4 評価に関する考察

CyberGRIP/MASを用いてプログラムを実装した場合、大幅にコード量を減らすことができた。しかしコード量はモデル作成者のプログラミング能力や経験に依存し、必ずしも実装時間と比例しない。プログラム完成まではデバッグ等も必要であるため、それらの作業も含めてさらに評価する必要がある。また、プログラミングの容易さをさらに客観的に評価するため、実際の研究者に利用してもらい、意見を収集する必要もあるだろう。

プログラム①、②の実行時間を計測した結果、両者の差はエージェントの行動ルールを変更しただけで大きく変化することが示された。このように実行時間はモデルの内容に大きく依存する。従って本研究のデータのみでCyberGRIP/MASの性能を評価することは難しい。エージェントの行動がさらに複雑な場合、エージェント数の増減が激しい場合、空間に対するエージェント数の割合を変えた場合等、あらゆるモデル、実行パターンに対して実行時間を計測し、評価する必要がある。

5. まとめ

本研究では、マルチエージェントシミュレーション用のクラスライブラリCyberGRIP/MASを開発した。さらに本ライブラリを用いて代表的なシミュレーションモデルのプログラムを実装し、コード量、実装時に考慮すべき点、実行時間について評価を行った。その結果、CyberGRIP/MASを用いた場合、コード量や実装時に考慮すべき点を軽減できることが示された。一方、実行時間はモデルの内容に大きく依存するため、本研究のデータのみで性能を正確に評価することは難しい。評価のためには、今後実装例を増やす必要がある。

また、あらゆるモデルに対応することと高速化を同時に実現することは非常に困難であるため、CyberGRIP/MASでは実装の負担を減らし、現実に近いモデルが構築できることを最優先とする。将来的にはグリッドミドルウェアであるCyberGRIPと連携させることで、全体の実行時間の短縮を図っていきたい。

謝辞：本研究にあたり、マルチエージェントシミュレーションに関しての資料や各種アドバイスを下さった、関西大学ソシオネットワーク戦略研究センター長・鵜飼康東教授、関西大学政策グリッドコンピューティング実験センター長・村田忠彦助教授、大阪府立大学院生の北埜裕子氏に感謝いたします。

参考文献

1) 南一久, 村上陽平, 河添智幸, 石田亨：マルチエージェントシミュレーションによる避難シミュレーション, 第16回人工知能学会全国大会, 2B1-04, 2002
2) 畑貴司, 原尾政輝, 平田耕一：マルチエージェントモデルを用いた交通流シミュレーション, 電気関係学会九州支部連合大会(第57回), 2004
3) 山下智規, 中村武雄, 上田晴康, 今村信貴, 岩松隆則, 斉藤直之：グリッド環境「CAD-Grid」構築と移動通信シミュレーションへの適用. 第6回問題解決ワークショップ論文集, pp.31-36
4) 小橋博道, 清水智弘, 今村信貴, 上田晴康, 野口弘, 山下智規, 門岡良昌, 宮澤君夫：グリッドミドルウェアCyberGRIPによる組織を横断した計算機利用, PSE Workshop & Grid Seminar in Oita, 2004
5) T. Murata, H. Kitano, Y. Kadooka, and Y. Ukai, "Political Multi-Agent Simulation with Grid Computing," The Second International Conference of Socionetwork Strategies, Osaka, Japan, 2004
6) 村田忠彦, 北埜裕子, 小橋博道, 鵜飼康東, 伊達進 "社会科学応用のためのマルチエージェントシステムによる商用プロバイダ経由とスーパーSINET経由のグリッドシステム比較実験", 第49回システム制御情報学会 講演論文集, pp.341-342, 京都, 2005
7) http://www2.kke.co.jp/mas/