大規模マルチエージェントシステムのためのメタレベル制御機構

山根昇平[†] 石田亨[†]

大規模マルチエージェントシミュレーションでは，シミュレーションの状況が多様に変化したり，シミュレーションの期間が長くなったりすることが想定される．このとき，交通シミュレーションでは，1つ1つの行動シナリオが複雑化するのを防ぐために，シミュレーションの状況に応じた行動シナリオの切替えが有用である．また，一部のエージェントを人間が操作する参加型シミュレーションにおいては，操作者に意思決定を行う時間を与えるために，人間が操作するエージェントの動作状況に基づいたシミュレーション速度の切替えが必要となる．本研究では，大規模シミュレーションの柔軟な制御を実現するため，制御の記述を行うメタシナリオと，メタシナリオに基づいた制御を実現するメタレベルアーキテクチャを開発した．そして例題により，メタシナリオを用いて行動シナリオやシミュレーション速度の切替えが記述できることを示した．また，アーキテクチャに関しては，参加型シミュレーションや交通シミュレーションに関して，大規模マルチエージェントシステムのスケーラビリティを損なうことなく実装できることを示した．

Meta-level Control Architecture for Massively Multiagent Systems

Shohei Yamane[†] and Toru Ishida[†]

In a massively multiagent simulation, it is assumed that the situation of the simulation changes variously or the period of the simulation is long. In these cases, changing action scenarios of agents depending on the situation is required to avoid making each action scenario too complicated in traffic simulations. And, when we consider participatory simulations in which some agents are operated by humans, changing simulation speed depending on the agents' processing condition is required to give operators time to make their decisions. In this paper, we developed meta-scenario description language to describe control of agents and simulations, and meta-leval control architecture that makes it possible to control simulations according to a meta-scenario description. And we show that meta-scenario can describe action scenario switching and smilation speed switching. In addition, we show the meta-level control architecture does not lose scalability in applications such as participatory simulations and traffic simulations.

1. はじめに

高性能コンピュータや分散コンピューティング環境を利用した，数万から数百万のエージェントを扱うことのできる，大規模マルチエージェントシステム（MMAS）について議論がされ始めている[6]．

MMAS の応用として，1つの都市全体を対象とした交通シミュレーション[2]や災害時シミュレーションを行うことが考えられる．このような大規模シミュレーションでは，多数のエージェントの多様な動作のためにシミュレーションの状況も多様に変化したり，1つの事象の広域的な影響を観測するためにシミュレートする期間が長期間になったりすることが想定される．このような場合，シミュレーションの柔軟な制御が必要である．たとえば，以下の2つの制御があげられる．

第1に，大規模マルチエージェントシミュレーションでは，エージェントの行動シナリオは非常に複雑なものになるという問題がある．たとえば，交通シミュレーションにおいては，しばしばエージェントのトリップ☆の作成が問題となる．トリップ生成の手法として，OD 表を用いた手法[1]や，遺伝的アルゴリズムを用いた手法[3]が提案されている．多様に変化する状況に対して，起こりうるすべての状況に対応できる行動シナリオを作成するのではなく，特定の状況でのみ動作する行動シナリオを複数用意して状況に応じて切り替えることで，1つ1つの行動シナリオを簡潔にすることができる．

† 京都大学大学院情報学研究科社会情報学専攻
Department of Social Infomatics, Kyoto University

☆ 出発地から目的地までの 1 回の移動

第2に，シミュレーションをトレーニングに用いたり，その過程を通じてより正確なエージェントのモデルを得るには，一部のエージェントを人間が直接操作する参加型シミュレーションが有効である[4),7)]．参加型シミュレーションではシミュレーション内で人間が意思決定を行うため，実時間でのシミュレーションが求められる．しかしながら，長期間なシミュレーションすべてを実時間で行うのは非効率的である．そこで，人間が操作するエージェントを監視して操作者に意思決定が必要な場面のみ実時間に切り替える制御が必要となる．

このように，MMASを用いてシミュレーションを行う場合，エージェント数だけでなく，シミュレーションの制御も問題となる．本研究の目的は，大規模シミュレーションにおけるこれらの柔軟な制御を実現することである．この目的の達成のため，シミュレーションの観測/制御およびエージェントの行動シナリオの観測/制御を記述するための言語として，メタシナリオを提案する．メタシナリオに基づく制御の実現のため，我々は以下の課題に取り組んだ．

(1) 制御機能の定義
メタシナリオを用いて行動シナリオの切替えや時間管理を記述するために必要な機能を定義する必要がある．

(2) 制御アーキテクチャの考案
メタシナリオに基づいた制御を実現するため，シミュレーション実行中に行動シナリオやシミュレーション速度が制御できるのアーキテクチャを考案する必要がある．また，MMASに導入することを想定しているため，MMASのスケーラビリティを損なわないアーキテクチャでなければならない．

以下，2章で行動シナリオの記述に用いるシナリオ記述言語の説明，3章でメタシナリオの説明，4章でメタシナリオに基づく制御を実現するメタレベルアーキテクチャの説明，5章でメタシナリオの参加型シミュレーションへの適用例，6章でアーキテクチャのスケーラビリティに関する評価を行う．

2. シナリオ記述

我々は，エージェントの行動シナリオを記述する言語として，京都大学で開発されたシナリオ記述言語$Q^{☆}$[5)]を選択した．本章では，このシナリオ記述言語について説明する．

☆ http://www.lab7.kuis.kyoto-u.ac.jp/Q/index_e.htm

2.1 シナリオ記述言語の言語仕様

シナリオ記述は，エージェントと外部（人間やエージェント）とのインタラクションを記述する．シナリオ記述は拡張状態遷移モデルで表され，状態遷移のきっかけをキュー，状態遷移にともなう外部への作用をアクションと呼ぶ．以下で，シナリオ記述に用いられる基本的な言語機能を説明する．

(1) キュー/アクション
キューとは，エージェントに外界の観測を依頼するものであり，インタラクションのきっかけとなるものである．外界への副作用はない．一方アクションは，エージェントに外界へなんらかの作用を行うよう依頼するものである．キュー，アクションはユーザによって定義され，依頼を受け取ったエージェントがどのように動作するかはエージェントの実装によって異なる．

(2) ガード付きコマンドとシナリオ
ガード付きコマンドは，複数のキューに対して並行して待ち受けを行うのに用いられる．いずれかのキューが観測されると，後続する形式が実行される．シナリオ記述は状態遷移で記述され，状態遷移はgo式によってなされる．状態が違えば，同じキューを観測しても異なったアクションが実行される．

(3) エージェント/アバタ
エージェントやアバタは，名前を指定して定義することができる．エージェントに対してシナリオを割り当てることで，シナリオ記述に従った動作を行う．一方，アバタは人間の操作者によって操作されるものであるが，エージェントと同様にシナリオを与えることが可能である．アバタにシナリオを与えることによって，操作者に意思決定が必要ない状況においては，アバタをエージェントのように自動的に動作させることができる．

2.2 シナリオ処理系

シナリオ処理系はシナリオ記述を解釈し，キュー，アクション，ガード付きコマンドの実行依頼をエージェントシステムに送る．エージェントシステムがキュー，アクション，ガード付きコマンドを実行する機能を持っていない場合，エージェントシステムが提供する機能に変換することで，これらの命令を実行させる．シナリオ処理系は，エージェントシステムからの実行結果をもとに次に実行する命令を決定し，エージェントシステムに実行依頼を送る．

このように，シナリオ処理系はエージェントシステムと切り離されている．そこで，メタシナリオ処理系は，シナリオ処理系を制御することで，エージェントシステムには手を加えることなく，エージェントのシ

ナリオ実行の制御が可能となる．

3. メタシナリオ

本章では，シミュレーション制御を記述するための言語であるメタシナリオについて述べる．

3.1 機能定義

参加型シミュレーションにおいて，アバタの操作者の意思決定の完了は，ただちにシミュレーション環境に反映されるのではなく，アバタの内部状態の変化として現れる場合が多い．したがって，操作者の意思決定を観測するためには，シナリオの実行状況を観測する必要がある．また，シミュレーションの状況に応じたシナリオの変更を行うためには，シミュレーションの状況の観測やシナリオの変更といった機能が必要である．そのほか，シミュレーション速度の変更などのためのシミュレーションの環境情報の操作や，シミュレーション実行者とのインタフェースのための機能が必要となる．

メタシナリオは，シミュレーションの状況に応じた制御を行う．したがって，拡張状態遷移モデルを用いてメタシナリオを記述する．状態遷移モデルを用いることにより，シミュレーションの状況を状態遷移機械の状態，制御を状態遷移機械の遷移として表現できる．

2 章で説明した Q も，拡張状態遷移モデルを用いている．そこで，メタシナリオの言語機能としても，キュー，アクション，ガード付きコマンドなどを採用する．メタシナリオのキュー/アクションを**表 1** にまとめた．表中において，?で始まるコマンドはキュー，!で始まるコマンドはアクションを表している．

3.2 記述例

メタシナリオを用いて，シナリオの切替えを実現した例を**図 1** に示す．図中では，各状態を円で表し，状態遷移を矢印で表した．また，状態遷移のきっかけとなるキューの観測と，状態遷移にともなうアクションの実行は，(キューの観測/アクションの実行）という形で状態遷移の矢印に添えて示した．

図 1 の例では，メタシナリオは昼と夜の 2 状態を持つ．また，エージェントには昼用のシナリオと夜用のシナリオが用意されている．図 1 のメタシナリオでは，昼に夜用のシナリオの実行を観測すると，エージェントに夜用のシナリオの実行を停止させ，代わりに昼用のシナリオを割り当てて実行させる．夜の場合も同様に，エージェントのシナリオを変更する．

表 1 のキュー・アクションを用いて図 1 のメタシナリオを記述したものを，**図 2** に示す．

表 1 メタシナリオのキュー・アクション
Table 1 Cues and actions of meta-scenario description language.

エージェントの制御	
?observeAction	アクションの実行を観測
?observeCue	キューの実行の観測
?observeTransition	状態遷移の観測
?observeScenario	シナリオ実行の観測
!runCue	エージェントにキューの実行を依頼
!runAction	エージェントにアクションの実行を依頼
!releaseScenario	エージェントからシナリオを解放
!assignScenario	エージェントにシナリオを割り当て
シミュレーションの制御	
?observeEnvironment	シミュレーション環境を観測
!getEnvironment	シミュレーション環境情報を取得
!setEnvironment	シミュレーション環境の設定
!createAgent	エージェントの作成
!deleteAgent	エージェントの削除
!createCrowd	群集の作成
!createAvatar	アバタの作成
!startSimulation	シミュレーション開始
!stopSimulation	シミュレーション停止
シミュレーション実行者の環境の観測	
?observeMetaEnvironment	シミュレーション実行者の環境を観測
!getMetaEnvironment	シミュレーション実行者の環境を取得

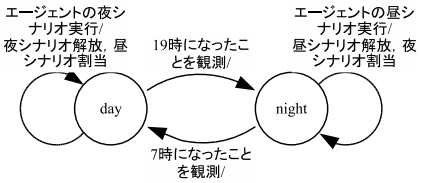

図 1 エージェントのシナリオを切り替えるメタシナリオの例
Fig. 1 A state machine of the meta-scenario that changes the scenarios of the agents.

4. シナリオ実行制御アーキテクチャ

本章では，メタシナリオの記述に従ってシミュレーションを制御するアーキテクチャを説明する．さらに，このアーキテクチャを用いたシナリオ実行制御の流れを説明する．

4.1 メタレベル制御アーキテクチャ

大規模システムにメタシナリオを導入する場合，いかにオーバヘッドを軽減するかが問題となる．そこで我々は**図 3** のようなアーキテクチャを考案した．

```
(defmetascenario day-and-night
                 (&pattern (agent #f))
  (day
     ; 19:00 になるのを観測する
    ((?observeEnvironment :name "time"
                          :value "19:00")
     (go night))
     ; nightScenario の実行を観測し,
     ; エージェントを変数 agent に代入する
    ((?observeScenario :name 'nightScenario
                       :agent agent)
     ; agent から nightScenario を解放する
     (!releaseScenario :name 'nightScenario
                       :agent agent)
     ; agent に dayScenario を割り当てる
     (!assignScenario :name 'dayScenario
                      :agent agent)
     (go day))
  (night
     ; 7:00 になるのを観測する
    ((?observeEnvironment :name "time"
                          :value "7:00")
     (go day))
     ; dayScenario の実行を観測し,
     ; エージェントを変数 agent に代入する
    ((?observeScenario :name 'dayScenario
                       :agent agent)
     ; agent から dayScenario を解放する
     (!releaseScenario :name 'dayScenario
                       :agent agent)
     ; agent に nightScenario を割り当てる
     (!assignScenario :name 'nightScenario
                      :agent agent)
     (go night)))
```

図 2 エージェントのシナリオを切り替えるメタシナリオの記述例
Fig. 2 A meta-scenario description that changes the scenarios of the agents.

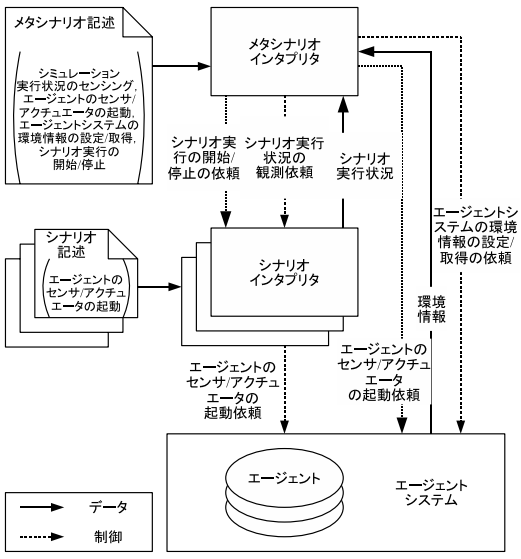

図 3 メタレベル制御のアーキテクチャ
Fig. 3 The meta-level control architecture.

シナリオ記述 シナリオ記述は，2 章で述べたように，エージェントの行動シナリオを記述したものである．

シナリオインタプリタ シナリオインタプリタは，シナリオ記述を解釈実行するもので，エージェントシステム上の特定のエージェントと対応している．シナリオ記述はシナリオインタプリタによって解釈実行され，シナリオインタプリタは対応するエージェントに対してセンサ・アクチュエータの起動依頼を行う．

メタシナリオ記述 メタシナリオ記述では，実行する命令の変更といった，シナリオインタプリタによるシナリオの解釈実行に対する観測や制御を記述する．また，メタシナリオ記述では，シミュレーション速度の変更のような，エージェントシステムの環境設定・取得を記述することができる．

メタシナリオインタプリタ メタシナリオインタプリタは，メタシナリオ記述の解釈実行を行う．シナリオ記述の解釈実行の観測はシナリオインタプリタ対して，エージェントのセンサ・アクチュエータの起動やエージェントシステムの環境設定・取得はエージェントシステムに対して，それぞれ実行依頼が行われる．

エージェントシステム エージェントシステムは，シナリオインタプリタやメタシナリオインタプリタから受け取った，エージェントシステムやエージェントに対する依頼を実行する．

メタシナリオ導入によるオーバヘッドは，主にシナリオインタプリタの観測/制御に起因する．たとえば，メタシナリオインタプリタがすべてのシナリオインタプリタの動作を逐一観測し，必要に応じた制御を行うことで，より柔軟な制御が実現できる．しかし，このような方式では，メタシナリオインタプリタの処理や，通信のため，オーバヘッドが大きくなってしまう．図 3 の方式では，このオーバヘッドを軽減するため，シナリオの観測はシナリオインタプリタに依頼し，制御が必要な場合にのみメタシナリオインタプリタに観測結果を返している．この観測結果は，メタシナリオインタプリタが持つ Queue に入れられ，順番に処理される．

また，シミュレーションに対して複数のメタシナリオを適用することもできる．たとえば，シナリオを切り替えるためのメタシナリオと時間管理を行うためのメタシナリオを実行することで，シナリオ切替えと時間管理の両方を行うことができる．この場合，メタシナリオインタプリタが複数起動されることになる．

4.2 メタシナリオ実行の流れ

1 つ例をとりあげて，メタシナリオの実行手順について説明する．ここで取り上げる例は次のようなものである．メタシナリオはまず，シナリオインタプリタのアクション `!run` の実行を観測する．シナリオインタプリタから観測通知を受け取ると，メタシナリオイ

ンタプリタはエージェントに対して!walkの実行を依頼する．

この例について，メタシナリオの実行手順を説明する．シーケンス図で記述すると**図 4** のようになる．

（1） メタシナリオインタプリタは，メタシナリオ記述を読み込んで解釈することで，実行すべき命令を決定する．この例ではまず，シナリオインタプリタがエージェントに対して!runの実行依頼を送るのを観測するため，観測依頼を各シナリオインタプリタに送る．

（2） 一方でシナリオインタプリタは，エージェントの実行を行うため，シナリオ記述を読み込んで実行すべき命令を決定する．

（3） (2) で決定された命令はまず，メタシナリオインタプリタが観測を依頼している命令，すなわち!runと一致するか判定が行われる．

（4） !runではないと判定された場合，エージェントシステム上のエージェントに対してセンサ・アクチュエータの起動依頼が送られ，(2) の手順に戻る．

（5） !runであると判定された場合，メタシナリオインタプリタに観測結果として，シナリオの実行状況を送り，その後シナリオの実行を続行するために (2) の手順に戻る．

（6） シナリオインタプリタから観測結果としてシナリオ実行状況受け取ったメタシナリオインタプリタは，メタシナリオ記述を読み込んで次に実行する命令を決定する．

（7） 図 4 の例では，次に実行する命令は，観測通知を送ったシナリオインタプリタに対応するエージェントに対する，!walkの実行であるため，エージェントシステム上のエージェントに対して，!walkの実行依頼を送る．

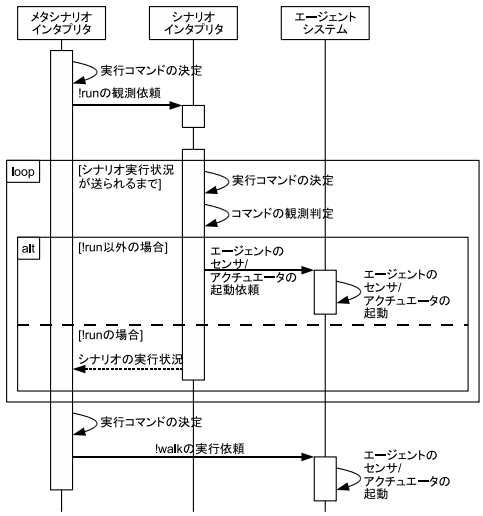

図 4 シナリオ実行制御のシーケンス図
Fig. 4 Sequence diagram of scenario execution control.

5. 参加型シミュレーションへの適用

本章では，メタシナリオを参加型シミュレーションに適用することで，メタシナリオの有効性を示す．

参加型シミュレーションでは，シミュレーション内にアバタが含まれる．アバタの操作者に意思決定を行う必要が生じた場合，シミュレーション速度を落とす必要がある．また，意思決定が終了すれば，速度を元に戻さなければならない．メタシナリオを用いて，シミュレーションとアバタの状況を観測して，それに基づいたシミュレーションの制御を行うことにより，この時間管理が実現できる．

5.1 シミュレーションの説明

本章で行うシミュレーションの GUI を**図 5** に示す．このシミュレーションでは，格子状の道路があり，道路上をエージェントが通行している．道路上にはアバタが一体配置されている．このアバタが特定の交差点（図 5 で円で囲った交差点）に近づいたとき，操作者が，右折，左折といった意思決定を行うため，シミュレーション速度を遅くする．そして意思決定が完了したら，速度を元に戻す．

図 5 の右側のウィンドウが，エージェントやアバタが通行する道路である．図 5 中の左側のウィンドウは，アバタの操作ウィンドウである．このウィンドウ上の “LEFT”，“RIGHT”，ボタンをクリックすることにより，アバタに左折，右折の指示を行うことができる．

5.2 アバタのシナリオ

アバタのシナリオの状態遷移図を**図 6** に示す．アバタのシナリオでは，前方に車があるときのみ停止し，前方に車がなければ直進し続ける．一方で，このシナ

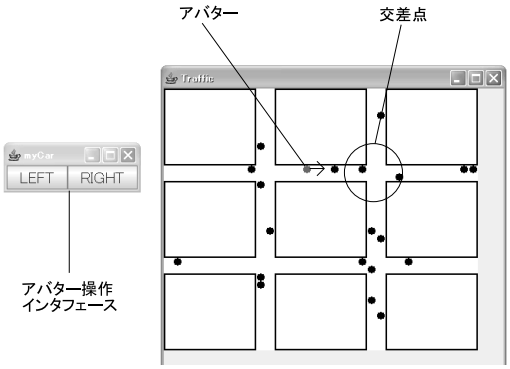

図 5 シミュレーションの GUI
Fig. 5 GUI of the simulation.

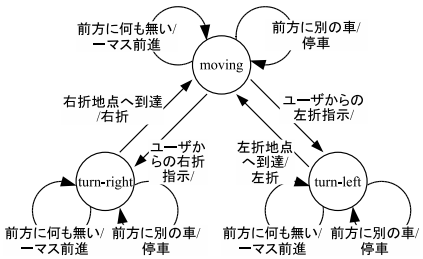

図 6 アバタのシナリオ
Fig. 6 A scenario for avatars.

リオでは図 5 で示した GUI によって，ユーザからの入力を受け付けている．この入力を読み取ることで，交差点で右折，左折を行う機能を持つ．入力がなければ，そのまま直進する．

図 6 に示したように，ユーザから「右折」という指示があった場合は `turn-right` 状態へ，「左折」という指示があった場合は `turn-left` 状態へ遷移する．`turn-right`/`turn-left` 状態では，交差点の右折/左折が可能な地点まで前進を続ける．そして到達したら，右折/左折を行い，`moving` 状態へ遷移する．

このように，アバタに対してシナリオを与えることで，アバタの操作者は意思決定が必要な部分だけ操作を行い，それ以外の部分，たとえば他のエージェントとの衝突を避けるためにアバタを停止するといったような部分は，自動化することができる．

5.3 時間管理を行うメタシナリオ

本シミュレーションでは，アバタが特定の交差点に近づくことで，操作者に右折するか左折するかという意思決定が発生する．そこで，アバタが交差点の近くにいて意思決定を行っている間だけシミュレーションの速度を遅くし，意思決定が完了するとシミュレーションの速度を元に戻す．この時間管理を行うメタシナリオを**図 7** に示した．状態遷移図は**図 8** に示す．

このメタシナリオは次のように動作する．まず，`initial-state` でエージェントの配置などを行い，`on-road` へ遷移する．`on-road` ではアバタが交差点に近づくのを観測し，観測されると速度を実時間まで落とし，`near-junction` に遷移する．`near-junction` ではアバタの状態遷移を観測することで，操作者の意思決定の完了を観測する．意思決定の完了が観測されると，速度を元に戻す．`within-junction` はアバタの意思決定が完了しているが，アバタがまだ交差点の近くにいる状態である．意思決定の完了後ただちに `on-road` へ遷移すると，そのまますぐに `near-junction` へと遷移してしまうため，この状態でアバタが交差点から遠ざかるのを待つ．

```
(defmetascenario meta-traffic ()
  (initial-state
   (#t
    (!createCrowd :name 'Cars :population 20)
    (!createAvatar :name 'myCar)
    (!assignScenario :name 'car :agent Cars)
    (!assignScenario :name 'car-avatar
                     :agent myCar)
    (!startSimulation)
    (go on-road)))
  (on-road
    ; myCar が交差点 (2,1) に近づくのを観測する
   ((?observeEnvironment :name 'onIntersection
                         :args (list myCar 2 1))
    ; シミュレーション速度を遅くする
    (!setEnvironment :name 'sim-speed
                     :args "slow")
    (go near-junction)))
  (near-junction
    ; myCar が交差点 (2,1) から遠ざかるのを観測する
   ((?observeEnvironment :name 'outOfIntersection
                         :args (list myCar 2 1))
    ; シミュレーション速度を速くする
    (!setEnvironment :name 'sim-speed
                     :args "fast")
    (go on-road))
    ; myCar が car-avatar シナリオの
    ; turn-right 状態に遷移するのを観測する
   ((?observeTransition :scenario 'car-avatar
                        :scene 'turn-right
                        :agent myCar)
    ; シミュレーション速度を速くする
    (!setEnvironment :name 'sim-speed
                     :args "fast")
    (go within-junction))
    ; myCar が car-avatar シナリオの
    ;turn-left 状態に遷移するのを観測する
   ((?observeTransition :scenario 'car-avatar
                        :scene 'turn-left
                        :agent myCar)
    ; シミュレーション速度を速くする
    (!setEnvironment :name 'sim-speed
                     :args "fast")
    (go within-junction)))
  (within-junction
    ; myCar が交差点 (2,1) から遠ざかるのを観測する
   ((?observeEnvironment :name 'outOfIntersection
                         :args (list myCar 2 1))
    (go on-road))))
```

図 7 時間管理のメタシナリオ
Fig. 7 A meta-scenario description of time management.

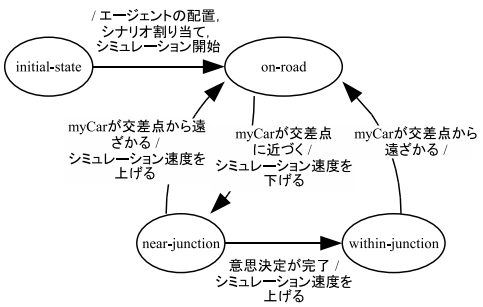

図 8 時間管理を行うメタシナリオの状態遷移図
Fig. 8 A state machine of meta-scenario of time management.

本章の例のように，操作者の意思決定の完了は，ただちにシミュレーション環境へ反映されるのではなく，エージェントの内部状態の変化として現れることが多い．メタレベルアーキテクチャにより，シナリオの状態遷移といったシナリオ実行状況の観測を行うことで，エージェントの内部状態の変化を観測して時間管理を行うことが可能となった．このように，メタレベルでエージェントのシナリオ実行を監視，制御することにより，エージェントの動作を含んだシミュレーションの状況の観測に基づいた，柔軟なシミュレーションの制御が可能となる．

6. 評　　価

前章までで，メタシナリオによって大規模シミュレーションに必要な制御がいかに実現されるかを示した．メタシナリオの目標はシミュレーションの柔軟な制御であるが，大規模なシステムを想定した場合，スケーラビリティも重要な課題となる．前章でシミュレーションに用いたシステムは試作品であり，十分なスケーラビリティは得られていない．しかしながら，IBM で開発された Caribbean[8] 上にシナリオインタプリタを実装することにより，1 台の PC 上において数十万程度のシナリオインタプリタを同時に動作させることができるという実験結果を得ている．したがって，この Caribbean を用いたシステムにメタレベルアーキテクチャを導入するとき，導入によるオーバヘッドが十分小さければ，メタシナリオを用いた大規模システムが実現可能である．そこで本章では，試作品を用いて，シナリオ処理系にメタレベルアーキテクチャを導入したときのオーバヘッドを計測する．

6.1 実　　験

4 章で説明したように，メタシナリオインタプリタとシナリオインタプリタの間でやりとりされる主な情報は，シナリオ実行状況の観測依頼とその結果である．そこで，以下のような実験を行い，オーバヘッドを測定した．

- エージェント数は 100 体.
- 各々のエージェントは，個別のカウンタを持っている．
- エージェントのシナリオは**図 9** で示す.
- メタシナリオは図 9 中の「空のアクション」を観測する．
- 「空のアクション」は全アクションの N 分の 1 を占める．
- N = 1，10，100 のそれぞれについて，1 分間測定を行い，全エージェントのカウンタ値の総数を計算する．

以上の実験を 3 回繰り返し，平均値をとった．

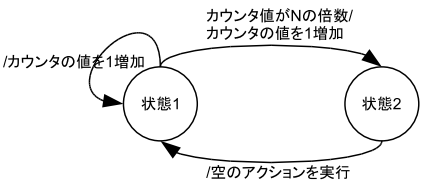

図 9 エージェントのシナリオ
Fig. 9 A scenario for evaluation.

表 2 オーバヘッドの測定結果
Table 2 Evaluation result.

N	メタシナリオなし	メタシナリオあり
1	30,361	20,238 (337.30)
10	34,708	34,277 (57.13)
100	35,183	33,937 (5.66)

6.2 測 定 結 果

測定結果を**表 2** に示す．表 2 において，括弧内の値は一秒あたりのメタシナリオによる観測回数を示している．表 2 では，メタシナリオなしの N=1 の値が N=10，100 に比べて小さくなっている．これは，エージェントが図 9 状態 1 から状態 1 へ遷移する場合と，状態 1，状態 2，状態 1 の順に遷移する場合の計算量が違うためで，N=1 ではすべて後者の遷移となるために値が小さくなっている．したがってオーバヘッドの評価は，同じ N の値における，メタシナリオなしの場合とありの場合を比較することで行う．

表 2 が示すように，観測頻度が比較的低い場合（N=10，N=100 の場合）には，オーバヘッドはほとんど見られなかった．これは，図 3 のアーキテクチャでは，観測が行われない場合には通信などの処理が発生しないためである．観測頻度が比較的高い場合（N=1）には，観測結果メッセージの処理のため，若干のオーバヘッドが生じていることが分かる．

このことから，参加型シミュレーションの時間管理のように，観測する対象が少ないときはオーバヘッドによる実行速度への影響はないと考えられる．また，シナリオ切替えのような例では，切替えを行う状況になったときに大量の観測結果メッセージが送られるため，一時的に動作速度が遅くなる．しかし，それ以外の状況ではオーバヘッドは微少であるため，シミュレーション全体としての影響は少ない．したがって，本研究で想定するような参加型シミュレーションや交通シミュレーションに関して，図 3 のアーキテクチャは妥当であると考えられる．

7. おわりに

交通シミュレーションや参加型シミュレーションを大規模環境下で行うには，実行時のシミュレーション状況やエージェントの行動シナリオを監視し，それに基づいた制御を行う必要が生じる．このため，我々が達成した目標は以下の 2 点である．

(1) メタシナリオ記述言語の開発
シミュレーションの状況を拡張状態遷移機械の状態として，制御を拡張状態遷移機械の遷移として表現することで，シミュレーションの制御を記述するための言語を提案した．

(2) メタレベルアーキテクチャの考案
シナリオインタプリタをメタレベルで制御することで，メタシナリオに基づいた柔軟な制御を可能とするメタレベルアーキテクチャを考案した．

これらに関して，例題により，時間管理やシナリオの切替えが実現できることを示した．また，アーキテクチャに関して，我々の想定する参加型シミュレーションや交通シミュレーションに関して，大規模マルチエージェントシステムのスケーラビリティが損なわれないことを示した．

5 章で述べたように，メタシナリオは複数実行可能である．複数のメタシナリオが同じ対象を制御する場合，メタシナリオ間で競合が起こりうる．このような競合をどのように発見し，解決するかが今後の研究課題となっている．

謝辞 本研究は，日本学術振興会科学研究費基盤研究（A）（15200012，2003-2005）の助成を受けて行われました．

参考文献

1) Balmer, M. and Rieser, M.: Generating Daily Activity Chains from Origin-Destination Matrices, *Transportation Research Board 84th Annual Meeting* (2004).
2) Balmer, M., Cetin, N., Nagel, K. and Raney, B.: Towards Truly Agent-Based Traffic and Mobility Simulations., *3rd International Joint Conference on Autonomous Agents and Multiagent Systems* (*AAMAS 2004*), pp.60–67 (2004).
3) Charypar, D. and Nagel, K.: Generating Complete All-Day Activity Plans with Genetic Algorithms, *presented at the 10th International Conference on Travel Behaviour Research* (2003).
4) Guyot, P., Drogoul, A. and Lemaitre, C.: Using emergence in participatory simulations to design multi-agent systems, *Proc. 4th International Joint Conference on Autonomous Agents and Multi-Agent Systems* (*AAMAS*), pp.199–203 (2005).
5) Ishida, T.: Q: A Scenario Description Language for Interactive Agents, *IEEE Computer*, Vol.35, No.11, pp.42–47 (2002).
6) Ishida, T., Gasser, L. and Nakashima, H. (Eds.): *Massively Multi-Agent Systems I*, Lecture Notes in Artificial Intelligence, 3446, Springer-Verlag (2005).
7) Murakami, Y., Sugimoto, Y. and Ishida, T.: Modeling Human Behavior for Virtual Training Systems, *The 20th National Conference on Artificial Intelligence* (*AAAI-05*), pp.127–132 (2005).
8) Yamamoto, G. and Tai, H.: Performance Evaluation of an Agent Server Capable of Hosting Large Numbers of Agents, *International Conference on Autonomous Agents* (*Agents-01*), pp.363–369 (2001).

(平成 17 年 10 月 3 日受付)
(平成 18 年 3 月 2 日採録)

山根 昇平
2005 年京都大学工学部情報学科卒業，現在，同大学院社会情報学専攻修士課程に在学中．マルチエージェントシステムに関心を持つ．

石田 亨（フェロー）
1976 年京都大学工学部情報工学科卒業，1978 年同大学院修士課程修了．同年日本電信電話公社電気通信研究所入所．ミュンヘン工科大学，パリ第六大学，メリーランド大学客員教授等経験．工学博士．IEEE フェロー．情報処理学会フェロー．現在，京都大学大学院情報学研究科社会情報学専攻教授，上海交通大学客員教授．自律エージェントとマルチエージェントシステム，セマンティック Web 技術に取り組む．デジタルシティ，異文化コラボレーションプロジェクトを推進．