役割指向を用いた並行システムの記述と実装手法の提案

柏木孝仁 ○佐々木晃（法政大学） 田沼英樹（東京工業大学）

Study of Methods for Describing and Implementing Concurrent Systems with Role-Orientation

T. Kashiwagi, *A. Sasaki (Hosei University), and H. Tanuma (Tokyo Institute of Technology)

Abstract－ Role-orientation is one approach for building Agent-Based Models (ABMs). Role-orientation provides a method of constructing ABMs with concept of "agent" and "stage", "role". Role-orientation approach to describing systems is expected to be applied for building parallel systems. For example, "agent" of role-orientation's concept performs parallel processing, and "stage" of role-orientation's concept synchronizes of parallel processing. We propose a method for describing parallel and distributed systems with role-orientation.

Key Words: Agent Base Modeling, Agent Processor, Programming Languages

1 はじめに

役割指向は、SOARS 社会シミュレーションシステム[1]におけるエージェントベースモデリングの手法の中心的な概念である。SOARS によるシミュレーションは、ステージと呼ばれる時間境界で個々の役割を持つエージェントが同期を取りながら進行する並行システムとして捉えられる。本研究の目的は、社会システムを記述するために見いだされた役割指向を、より一般的なソフトウェアとしての並行システムを実現する手法として応用することである。本発表では、我々の先行研究である役割指向テンプレートジェネレータ[2]を利用し、そのような並行システムを役割指向により記述、実現できることを示すとともに、その応用について考察する。

役割指向は、社会シミュレーションエージェントベースモデル（Agent-Based Model, ABM）として構築するための一つの手法として見いだされた概念である。社会シミュレーションにおける ABM は，エージェントを用いて社会現象を表現することが目的である。役割指向における「役割」および順序制約を表す「ステージ」の概念は，社会現象をエージェントの振る舞いの種類、および時間の順序という 2 軸に基づいて複数のサブモデルに分割して記述することを可能とし，直感的な ABM の構築方法を提供する．このように、役割指向は ABM の構築に用いられる概念であるが，実世界において並列に動作するエージェントを直観的にモデル化する性質をもつため，この概念を並行システムの記述に応用できると考えられる．例えば、並列分散システムを構築するために役割指向がもつ上記の概念を用いることによって，エージェントによる並列処理の実行，ステージ概念を用いた並列処理の同期を行う，などのシステムの記述，実装法が可能となると考えられる．

本発表では、並行システムとして、1 つのクライアントと複数のサーバ（ワーカー）群からなる構成の並列分散システムを、役割指向を用いて記述し実現することを試みる。この実験を通して、役割指向の利点を踏まえた記述法が、並列分散システムを実現するにあたって利用可能かどうかを確認する。すなわちシステムをエージェントやステージを用いて記述できるとともに，具体的なシステムの実装を隠蔽してシステムの設計が行える，といった特質が得られるかどうかなどについて考察する．なお，本来並列分散システムにおいては，実行速度の効率が求められるはずだが，本研究では役割指向によるプログラムの記述性について注目する．

2. 背景

役割指向は，エージェントシミュレーションシステム SOARS（Spot Oriented Agent Role Simulator）[1]における，エージェントベースモデルの構築手法から抽出した概念である。これは、動作主体である「エージェント」、エージェントの動作の種類を表す「役割」、また動作の順序制約を表す「ステージ」の３つの概念からなる。SOARS では，これらの概念およびエージェントの相互作用の場を表現する概念である「スポット」を中心にシミュレーションモデルを構築する．SOARS において，感染症シミュレーション，経済シミュレーションなど，社会シミュレーションの開発に役割指向の概念が有効に働くことがわかってきた．特に、エージェントの動作を「役割」という単位、および「ステージ（時間制約）」という単位の２軸に従って記述する点がモデルの記述性と可読性を高めた。さらに「役割」はモジュールとして作用することが特徴であり、役割継承およびステージ連結という直截的な合成法でモジュール同士の連結を可能としている。これらの特質は、複雑なモデルのボトムアップアプローチによる構築に貢献している。

SOARS において，役割指向のステージと役割が，シミュレーションモデルの構造化に効果的に働くことに着目して，役割指向をソースコードの生成に利用したツールが，役割指向テンプレートジェネレータである[2][3]. 本ツールに与える入力記述（役割指向記述）は次のような特徴をもつ。(1)ステージはソースコードの構造（ソースコードの断片の順序）を定義する．(2)エージェントは，ステージに定義された構造に従いソースコードを出力するオブジェクトと捉えられる．(3)役割は，ソースコードの断片をステージと関連付けてまとめる機能としてはたらく．この入力記述では，メタ文字をデ

ィレクティブの1種として用いることによって役割指向の構造を表し、ジェネレータはそれに従ってソースコードを再構成する。これを利用して、役割指向による記述から汎用言語によるエージェントシステムを実現できる。また、シミュレーションモデルの構築以外への役割指向の応用も、本ツールを用いて実現できることが期待される。

本研究では，役割指向を用いて一般的な並行システムを記述することを目指す。今回は、1 つのクライアントと複数のサーバ（ワーカ）群からなる構成の並列分散システムを並行システムの例として挙げ、そのうちシステムのクライアント部を役割指向によって記述することを試みた．クライアントの処理には，サーバとの通信処理があり，また，各サーバへの仕事の依頼と結果の受け取りは並列に行う必要がある．このように，並列分散システムでは、ネットワーク処理や並列動作含む処理を正しい構造として記述する必要があるが、このような構造を汎用言語における関数やメソッド等の抽象化機構やその組み合わせのみで表現することは簡単ではない。役割指向を用いることで、エージェント概念による並行動作を自然に記述できること、さらにネットワーク等の処理の実装内容を役割の中に隠蔽できること、などの利点があると期待できる。

3. 役割指向による並列分散システムの記述および実装方法

3.1 役割指向による並列分散システム記述の概要

役割指向における「ステージ」は，全エージェントが参照する大域的なタイマーととらえられる。SOARS および本手法で提案するエージェントシステムでは、このタイマーに同期してエージェントは割り当てられた「役割」に従って並行に動作することで、システムが進行する。したがって、役割指向によって並列分散システムを表現するための自然な考え方は、サーバーの数と同数のエージェントを用意し、個々のエージェントが特定の 1 つのサーバーとやりとりを行う、というものである。より具体的には、処理全体のうちエージェントが実際に並列処理を行う部分は，1 つのステージ（並列処理ステージ）として定義し、他のステージ処理と区別する。並列処理ステージでは、エージェントは並列に動作する処理単位を非同期に起動させる．一方で，並列に処理できる箇所をまとめて 1 つのステージとしてしまうと，役割指向記述の汎用性が失われてしまうことが考えられる．すなわち、役割指向記述をカスタマイズしていくことを考慮して，適切にステージを定義する必要がある．

並列分散システムのクライアントを記述するためのもう一つのアイデアは，複数のエージェントの情報を集める場を SOARS の概念である「スポット」とする点である。SOARS では，エージェント間は直接通信をせず，エージェントとスポットの間の通信のみを許すという記述上の制約を持たせている。これによりシステムの記述が複雑になるのを防ぐ。本研究では、これらの手法を応用し，クライアントを 1 つのスポットとして表現する。

役割指向の記述は，役割指向テンプレートジェネレータを用いて行う．また，エージェント実行系の実装は，SOARS における実行系を参考にしたものを用いる．役割指向テンプレートジェネレータの記法により，エージェントの処理を、役割指向によって明確に記述することが可能になる．また、SOARS を参考にしたエージェント実行系を用いることにより，役割指向により記述したシステムのエージェントやスポットを実行することができる．

3.2 役割指向テンプレートジェネレータによる実行系の利用

役割指向テンプレートジェネレータとエージェント実行系によるシステムの記述と実装方法を示す．システムの実装には、汎用言語を用いて実装したエージェント実行系が必要である．役割指向テンプレートジェネレータは、汎用言語によるエージェント実行系と，役割指向によるエージェントの動作記述部を分離することを可能とする（図 1）．役割指向によるエージェントの動作記述部と，エージェント実行系を切り離すことにより，エージェント実行系を複数のシステムの実装に利用することができる．逆に，役割指向によるエージェントの動作記述部の構造を変えずに、エージェント実行系や、使用するライブラリやフレームワークを変更することも可能である．

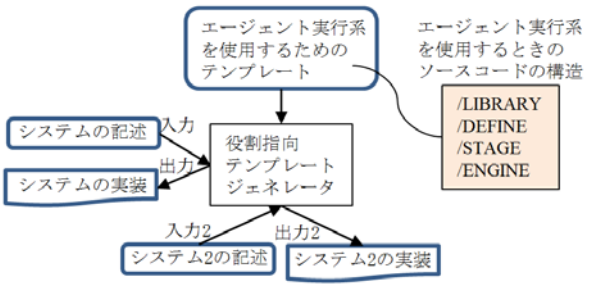

図 1　役割指向テンプレートジェネレータによる実装

役割指向テンプレートジェネレータは、役割指向の概念を用いて構成された構造的な記述（ドキュメント）を入力とし、構成要素となる部分記述を適切な順序で再構成し出力するツールである。役割指向テンプレートジェネレータの入力記述の構造は、以下を用いて構成される。

- エージェント : コードの出力を主要な実行内容とする主体
- ステージ : コードの出力順，ソースコードの構造を定義
- 役割 : コードをまとめるとともに、コードを複数のエージェント間で共有

役割指向テンプレートジェネレータでは，行頭のメタ文字をディレクティブの1種として用いることによって役割指向を記述できる．行頭のメタ文字には，役割指向のステージ (/) やエージェント（@），役割（なし）を定義するためのものや，出力コード（タブ）を示すためのものが存在する．その

他にも，ソースコード生成を効果的に行うためのメタ文字を備えている．

以下では，例を用いて役割指向テンプレートジェネレータ機能の概要を示す．

```
1    Role
2    /stageA
3              これは@.stageA のコードです.
4    /stageB
5    /stageC
6    @Agent1
7    :         Role
8    /stageC
9              これは@.stageC のコードです.
10
11   @Agent2
12   :         Role
13   /stageB
14             これは@.stageB のコードです.
```

上記のプログラム例では，2 体のエージェント{Agent1, Agent2}と，3 つのステージ{stageA, stageB, stageC}，1 つの役割{Role}を定義している．行頭の「@」はエージェント定義（6 行目，11 行目），「/」はステージ定義（2，4，5，8，13 行目）を行っている．1 行目の Role は，役割 Role を定義しているが，行頭にメタ文字がないことで役割定義であることがわかる．「これは@.stageA のコードです．」などの行が出力されるコードであるが，行の先頭にタブ文字をつけることで，ツールに出力コードであることを教えている．行頭の「: 」は，役割の継承を意味する．上記プログラム例では，2 体のエージェント{Agent1, Agent2}が，役割 Role で定義された出力コードを保持する．ステージの順序は，役割 Role の記述において，空行をいれずにステージを記述することで定義する．上記プログラム例において，ステージの順序は{stageA→stageB→stageC}となる．

上記プログラム例において，具体的にエージェント Agent1 が出力するコードは，3 行目と 9 行目のコードになる．エージェント Agent1 は，役割 Role を継承しており，役割 Role に定義した 3 行目のコードも出力する．Agent2 が出力するコードも Agent1 と同じ仕組みで定義され，3 行目と 14 行目のコードになる．エージェントはコードを出力する際，出力コード内のメタ文字「@」を認識し，その@が，エージェント自身の名前と置換される．エージェント Agent1 が出力するコード内の@は，文字列 Agent1 に置換される．

出力例は次の通りである．

```
これは Agent1.stageA のコードです.
これは Agent2.stageA のコードです.
これは Agent2.stageB のコードです.
これは Agent1.stageC のコードです.
```

なお，1 行目と 2 行目の順序は任意である．役割指向テンプレートジェネレータの内部実装では，ステージ内でのエージェントの実行順は，エージェントの定義された順に依存することになる．しかし，ステージ内でエージェントが任意の順序で実行されて良いように記述することを推奨する．ステージ内におけるエージェントの実行順序を考慮した記述は，想像以上に困難であるためである．

4. 役割指向による並列分散システムの記述例

本節では，役割指向による並列分散システムの記述例について，並列分散システムの基本的な動作を行う単純なシステムの実装を例として説明する．本研究では、実装言語として Clojure[4]を用いた．実際には、テンプレートジェネレータの入力記述のうち出力コード部分に Clojure プログラムの断片を記述するが、ここにに記述する言語は，汎用言語であればこの言語のみに限らない[3].

例として，複数掛算同時実行を行う並列分散システム（図 **2**）を考える．複数掛算同時実行を行う並列分散システムでは，クライアントが式 Z=A*B+C*D+… を解くために，項の掛算をサーバに依頼する．サーバには掛算機能が実装してあり，サーバはクライアントから掛算の依頼を受けると掛算結果をクライアントに返す．クライアントにおいて，一つの項の掛算依頼と掛算結果の取得と他の項の掛算依頼と掛算結果の取得は，独立しているため並列に処理する．

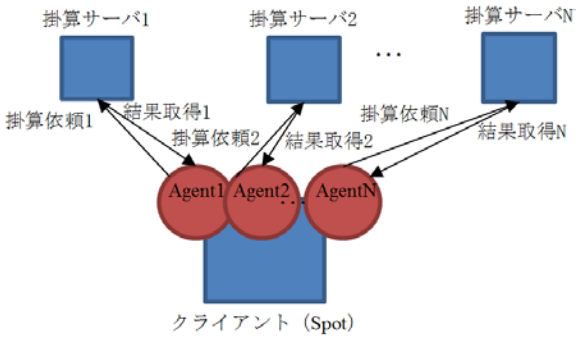

図 2　エージェントとスポットによる複数掛算同時実行システムのクライアントの実装

リスト 1 に役割指向による複数掛算同時実行システムのクライアントの記述を示す．この記述例は，式 Z=A*B+C*D を解く場合であり，エージェントは 2 体である．クライアントの全体の処理は，ステージにより分割する．クライアントの処理は，/CONNECT（サーバとの接続生成），/SEND（掛算依頼），/RECEIVE（掛算結果取得），/SYNC（同期），/CALCULATE（項の和を求める），/TERMINATE（サーバとの接続終了）という順序の各ステージに分割する（リスト 1）．1 つのサーバへの処理（ネットワーク処理の/CONNECT と/TERMINATE や，並列性のある仕事の依頼の処理の/SEND と/RECEIVE）を 1 体のエージェントが実行する．さらに、エージェントの活動の場は SOARS の概念であるスポットによって表し、個々のエージェントのもつ情報を収集する。

```
@@
:    ScenarioStage ;; ← Client のステージ
/CONNECT
+
```

```
/SEND
+
/RECEIVE
+
/SYNC
+
/CALCULATE
+
/TERMINATE
+
```

リスト 1　クライアントのステージの記述

エージェントやスポットは属性を持つことが出来る（リスト **2**）．エージェントは，サーバへの接続（属性 socket）や並列処理のスレッド（属性 future）を属性に保持することを示している．また，接続するサーバの情報（属性 server-info）やサーバへ依頼する掛算のオペランド（属性 operands），サーバから取得した掛算結果を格納する場所（属性 place）を属性として保持する．エージェントの役割は WorkerRole を設定する（属性 role にて設定する）．エージェントが存在するスポットは”client”であることが属性 spot からわかる．

```
@agent-1
:    Agent
{
    (add-attr! self :server-info server-1)
    (add-attr! self :place :place-1)
    (add-attr! self :operands operands-1)
    (add-attr! self :socket nil)
    (add-attr! self :future nil)
    (set-attr! self :spot
                      (soars-spot “client”))
    (set-attr! self :role
                   (soars-role "WorkerRole"))
}
```

リスト 2 サーバへの処理を実行するエージェントの記述

役割は継承させることが可能であり．ベースとなる役割のステージと関連付けられた処理を、それを継承した役割で利用することができる．例えば，サーバへの接続を行う処理は，並列分散システムだけでなく，サーバ・クライアントシステムにおいて一般的に必要になる処理である．そこで，サーバとの接続を行うための処理を役割（ConnectorRole）としてモジュール化する．複数掛算同時実行システムのサーバへ仕事を依頼するエージェントの役割は，サーバへの接続の機能をもった役割 ConnectorRole を継承して作成する（役割 WorkerRole とする）．役割 WorkerRole において/SEND と /RECEIVE，/SYNC のステージの処理を定義する．

役割 WorkerRole の記述をリスト 3 に示す．/SEND や /RECEIVE は，各エージェントが独立にサーバへの依頼を行う箇所である．エージェントは，/SEND や/RECEIVE を並列処理する．/SEND において，Java スレッドを生成しサーバへ仕事を依頼し、/RECEIVE では，Java の Future ライブラリ（Future は，マルチスレッドにおけるパターンの 1 つ）を利用して並列に処理する．/SYNC において/RECEIVE の処理の同期を行う。/SYNC においてはエージェントが，/RECEIVE で呼び出された並列処理の結果を待ち、得られた結果をエークライアント(Spot)に格納する．

```
@WorkerRole
:    ConnectorRole
/SEND
{
    (.start (make-send-thread
              (ref-attr self :socket)
              (ref-attr self :operands)))
}
/RECEIVE
{
    (set-attr! self :future
      (future (socket-recv
                (ref-attr self :socket))))
}
/SYNC
{
    (set-result! (ref-attr self :spot)
                 (ref-attr self :place)
                 (ref-attr self :future))
}
```

リスト 3　サーバへ仕事を依頼する WorkerRole 役割

本節にて示した記述以外には，役割 ConnectorRole やスポット(client)，スポットの役割が必要である．

5. 実装

本節では役割指向テンプレートジェネレータによって，4 節出示した複数掛算同時実行システムのクライアントの記述に対する実装が生成されることを確かめる．

４節の記述は実際には、エージェント(Agent)、ロール(Role)といったプロトタイプオブジェクト、エージェント実行エンジンの実装（これらをエージェント実装系と呼ぶ）と合成され、最終的な並列分散システムの実装を得る必要がある。エージェント実行系の記述は、役割テンプレートジェネレータの入力記述の形式をもち、本研究における clojure の実行系の記述では/LIBRARY，/DEFINE，/STAGE，/ENGINE の 4 つのステージによって定義される。リスト 4 の記述を４節の記述に連結することで、最終的なシステムを得る．/LIBRARY ではシステムにおいて使う関数群を出力する．/DEFINE ではエージェントやスポット，役割の定義するコードを，/STAGE では役割にステージと関連付け処理が定義す

るコードを，/ENGINE では，エージェントやスポットを実行するためのコードを出力する．

```
@@
/LIBRARY
/DEFINE
/STAGE
/ENGINE
```

リスト 4　エージェント実行系を使うときのソースコードの構造を示したステージ記述

最終的にテンプレートジェネレータが合成したコードの一部を示す（リスト 5）. /DEFINE 部では, 役割やスポット, エージェントを定義するためのコードである．/STAGE 部では，役割に処理を追加するコードが出力される．最後の add-action!関数の呼び出し部のうちイタリックで表記している断片は、リスト４における WorkerRole 記述の/SEND 内の行頭にタブ（空白）がある行を合成したものである。

```
...
;; /DEFINE
(add-role! (make-role "WorkerRole"))
(add-role! (make-role "ClientRole"))
(define-spot! "client" code1)
(define-agent! "agent-1" code2)
...
;; /STAGE
(update-stage stage) ;; stage CONNECT
(add-action! (soars-role "WorkerRole") stage
                                        code3)
(update-stage stage) ;; stage SEND
(add-action! (soars-role "WorkerRole") stage
   (.start (make-send-thread
            (ref-attr self :socket)
            (ref-attr self :operands))))
...
```

リスト 5　役割指向テンプレートジェネレータから出力された複数掛算同時実行システムのクライアントの実装

6. カスタマイズ例

本節では，役割指向による設計を活かした並列分散システムのカスタマイズと方法を示す．5 節に示した複数掛算同時実行システムのカスタマイズを行う．各ステージにおける処理結果を表示するログ出力機能を追加する．例えば，ステージ SEND においては依頼した掛算のオペランドを表示する．拡張方法の一つは，役割とステージを用いて，ログ出力機能をモジュール化することある（表 1）．手順は以下の通りである．

(1)システムのステージにログを出力するためのステージを追加する．

(2)ログを出力する役割 LoggerRole を作成する．

(3)システム上のオブジェクト{Worker1，Worker2，Client}の役割に LoggerRole を継承する．

表 1　ログ出力機能追加カスタマイズ時のステージ・役割

stage / role	Connec-torRole	Wor-kerRole	Client Role	Logger-Role
/CONNECT	サーバ接続生成			
/CONNECT_LOG				接続ログ
/SEND		依頼		
/SEND_LOG				依頼ログ
/RECEIVE		結果取得		
/SYNC		同期		
/SYNC_LOG				取得ログ
/CALCULATE			解を求める	
/CALCULATE_LOG				計算ログ
/TERMINATE	サーバ接続終了			
/TERMINATE_LOG				終了ログ

以下にログ出力機能を追加した記述を示す（リスト 6）．

```
1   @@
2   : ScenarioStage
3   /CONNECT
4   +
5   /CONNECT_LOG
6   +
7   /SEND
8   +
9   /SEND_LOG
10  +
11  ...
12  LoggerRole
13  : Role
14  /CONNECT_LOG
15  {
16      (print (str
17          (ref-attr self :name) ": CONNECT: "))
18      (println (str
19            (get-adr (ref-attr self :socket))))
20  }
21  /SEND_LOG
22  {
23      (print (str
24        (ref-attr self :name) ": SEND request:"))
25      (println (Arrays/toString
26                  (ref-attr self :operands)))
27  }
28  /SYNC_LOG
29  {
30  ...
```

```
31
32  @WokerRole
33  : ConnectorRole
34  : LoggerRole
35  /SEND
36  {
37  ...
38  @ClientRole
39  : Role
40  : LoggerRole
41  /CALCULATE
42  {
43  ...
```

リスト 6　ログ出力機能追加カスタマイズ時の記述

記述の変更箇所は，(1)システムのステージ記述部にステージを挿入，(2)役割 LoggerRole の記述，(3)役割 Worker と役割 ClientRole に役割 LoggerRole を継承する記述（「: LoggerRole」）の追加である．

複数掛算同時実行システムのクライアントは，上記の拡張することで次のようにログを出力する．エージェント{Agent-1, Agent-2}は，それぞれ（3 * 4）と（2 * 3）を依頼し，掛算結果をスポット Client が加算して解 18 を得ている．

```
agent-1: CONNECT: localhost
agent-2: CONNECT: localhost
agent-1: SEND request:[3.0, 4.0]
agent-2: SEND request:[2.0, 3.0]
agent-1: RECEIVE result:12.0
agent-2: RECEIVE result:6.0
client: RESULT(A * B + C * D):18.0
agent-1: STOP
agent-2: STOP
```

7. 考察

本節では提案した役割指向による並列分散システムの記述や実装について考察する．5 節において，単純な並列分散システム（複数掛算同時実行システム）のクライアントの役割指向記述から，クライアント側のシステムの実装が生成できることを確かめた．役割指向を用いることによって，エージェントやステージ，役割に集中しシステムを記述することができた．また，並列分散システムを構築する上で困難になりうる具体的なネットワーク処理や並列処理などの実装を隠蔽してシステムを設計することができた．具体的な実装を役割指向により隠蔽することで，システムの実装に用いているライブラリやフレームワーク，エンジン（実行系）などの要素を，取り替えることが可能であることを示している．

並列分散システムの実装において,役割指向の利点を得るために考慮しなければならない点について考察する．まず，システムにおいてエージェントを捉える観点である．本研究では，エージェントを 1 つのサーバとやりとりをするオブジェクトとして捉えたが，サーバへ依頼する仕事を表すオブジェクトといった別の捉え方も可能である．

次に，ステージの構成方法もいくつかの可能性がある．本研究で実装した手法では，役割指向の構造の中で並列性を出すことで、役割指向が持つモジュール性やカスタマイズ性を活かせるよう考慮した。一方で，ステージの構成によっては，処理を必要以上に細かく分割してしまう可能性がある．また，ステージという概念がシステム構築に適さないことも考えられる．

さらに、役割指向によるエージェントの動作記述とエージェント実行系の実装とを分離して記述する方法についても考慮する必要がある。提案手法では、並行システムはエージェントシステムとして次のように実現される：(1)エージェントの実行系を実装し、(2)役割指向概念それ自体によって実行系とエージェント動作記述部の結合を行う。(3)この際、役割指向テンプレートジェネレータによる記法はメタ言語のように作用する。役割指向の利点を得るために重要となる点は特に(2)，(3)の設計である。これは，どのように実行系とその上で動作するシステムとの境界を定めるかの選択であり、さらにその境界においてシステムの実装をどのように隠蔽するかという設計上の問題を提起する。実際には、エージェントの役割指向による記述とエージェント実行系を切り離す適切な境界を定めることは簡単ではない。本研究では、まずエージェント実行系を設計、実装し、この実行系を利用したシステムの汎用言語による実装例をプロトタイプとして作成した。そして，個々のシステムの実装において共通に必要となる記述部を想定し、これを再利用できるよう役割としてまとめた。今後、エージェント記述とその実行系を分離するための具体的手法を確立することが重要である。

8. おわりに

本研究では、並列分散システムを例として、役割指向による並行システムの記述と実装の方法について述べた。並列分散システムの役割指向記述および実装例、さらに拡張例を示し，役割指向の利点および、提案手法のもつ利点について考察した．役割指向の有効性を探るためには，応用事例を増やす必要がある．例えば，役割指向による記述のカスタマイズ性など，役割指向の利点に注目した応用事例である．そのためにも，役割指向を記述するための手法やツールの開発，改良が求められる．

謝辞

本研究の一部は、科研費(課題番号 23700043, 23500034)の助成を受けた。

文　献

[1] H. Deguchi, H. Tanuma, and T. Shimizu, “SOARS: Spot Oriented Agent Role Simulator - Design and Agent Based Dynamical System -,” in Proc. AESCS04, pp.49-56, 2004.

[2] 田沼英樹，“役割指向テンプレートジェネレータによる従来言語ルール記述と役割指向 ABM の結合,” JAWS2011.

[3] 柏木孝仁，佐々木晃，田沼英樹，“Scheme 言語をルール記述言語とした役割指向記述の試み,” 情報処理学会第 74 回全国大会，2012

[4] R. Hickey , Clojure, http/clojure.org.