Stage-Oriented Agent Role Simulator Toolkitと1億人規模COVID-19シミュレーションによる実行性能評価

○小野 功，吉井 尉哲，永金 幸大，小嶋 健太 （東京工業大学）
市川 学 （芝浦工業大学） 出口 弘 （千葉商科大学）

Stage-Oriented Agent Role Simulator Toolkit and Its Runtime Evaluation by 100,000,000-Agent-Scale COVID-19 Simulation

∗I. Ono, Y. Yoshii, K. Nagakane, K. Kojima (Tokyo Insititute of Technology),
M. Ichikawa (Shibaura Insitute of Technology), H. Deguchi (Chiba University of Commerce)

概要– 本稿では，著者らが開発中の Stage-Oriented Agent Role Simulator Toolkit (SOARS Toolkit) を紹介し，1 億人規模 COVID-19 シミュレーションによる実行性能評価の結果について報告する．SOARS Toolkit は，ロール・ステージモデルの実装を容易にするためのエージェントベースシミュレーションライブラリである．ロール・ステージモデルは，オブジェクトの振る舞いをロール，時間をステージにより定義するエージェントベースモデルの 1 つである．オブジェクトには，行動主体を表すエージェント，場所を表すスポット，公共交通機関を表すトランスポーテーションがある．ロールは複数の状態とルールで定義される．複数のロールを組み合わせて新しいロールを作成したり，複数のロールをオブジェクトに持たせて動的に切り替えたりすることができる．ステージは，時間が離散値として扱われるエージェントベースシミュレーションにおいて，同一時刻に実行される複数のルールの実行順序を制御するための概念である．各時刻は，順序が定義された複数のステージから構成される．ルールの実行条件としてステージを指定することにより，同一時刻内でのルールの実行順序を制御できる．ロール・ステージモデルでは，1) あるオブジェクトが同一時刻の同一ステージに実行できるルールは高々1 個である，2) 同一時刻の同一ステージで実行される複数のルールはその実行順序によって実行結果が変わらない，という 2 つの制約を課している．そのため，ステージ内のルールの並列実行を比較的容易に実現可能である．SOARS Toolkit 最新版では，ロールのモジュール化を支援する機能，および，並列実行を支援する機能をサポートしている．ロールのモジュール化を支援する機能を利用して，あらかじめ様々なロールをライブラリとして用意することにより，それらを組み合わせるだけで高度なシミュレーションを簡単に実現できる．また，並列実行を支援する機能を利用して，比較的に容易にシミュレーションを高速化できる．本研究では，単純な都市モデルと，感染シミュレーションモデルの 1 つであるフィルタモデルを，SOARS Toolkit を用いて並列実装した上で，首都圏規模を想定した 3,000 万人／ 1,415 万スポット，および，全日本規模を想定した 1 億人/4,717 万スポットでの実行性能を評価する実験を行った．その結果，56CPU コアを用いて 180 日間 8,640 ステップを並列実行したところ，3,000 万人／ 1,415 万スポットのシミュレーションが 6.44 時間，1 億人/4,717 万スポットのシミュレーションが 25.6 時間で終わることを確認した．

キーワード: エージェントベースシミュレーション，ライブラリ，SOARS Toolkit，COVID-19 感染シミュレーション

1 はじめに

エージェントベースシミュレーション[1, 2]は，ミクロレベルで行動主体であるエージェントの集合とエージェント間の相互作用を定義することにより，ボトムアップ的にマクロレベルの現象である社会現象をシミュレーションするための技術である．そのため，エージェントベースシミュレーションは，ボトムアップな行動変容のための人間中心の個人に寄り添った意思決定支援のためのシナリオ分析が可能であるという特徴をもつ．また，個人の行動や社会ルールを直接モデル化してシミュレーションすることが可能であるため，日常の人々の行動や社会ルールをモデル化して汎用ソフトウェアモジュールとして用意しておけば，その上で，地域包括ケア，サプライチェーン排出量，MaaS，防災，防疫などの様々な領域，または複合領域的なシミュレーションが容易に実現できるという特徴も併せ持つ．これは，SIR（Susceptible, Infectious, Removed/Recovered）モデルなどのマクロ変数ベースのシミュレーションでは実現し得ない特徴である．

しかし，エージェントベースシミュレーションは，1) シミュレーションプログラムの開発が煩雑である，2) 実行コストが高い，という 2 つの問題をもつ．そのため，首都圏 3000 万人規模の感染シミュレーションなどの大規模・複雑なシミュレーションは，実装の煩雑さ，実行時間，実行メモリの観点から困難であるとされてきた．

エージェントベースシミュレーションプログラムの開発の煩雑さを軽減するため，NetLogo[3]，MASON[4]，Repast[5]，CORMAS[6]，GAMA[7]，Artisoc[8]，S4[9]，SOARS[10, 11, 12] などが開発されてきた．NetLogo，MASON，Repast，CORMAS，GAMA，Artisoc，S4 は，エージェントと空間のモデル化およびシミュレーション実行を支援するための機能は充実している．しかし，本質的に並列システムである複雑な現実社会をモデル化するために必須となるエージェントの競合解消の実装についてはユーザ任せであるため，モデル化・実装が煩雑になるという問題がある．一方，SOARS は，時間管理にステージとよばれる新たな概念を導入して，システムレベルでサポートすることにより，競合解消を容易に実現できるようにしている．しかし，SOARS は，1) 実行速度が遅い，2) 鉄道などの公共交通機関のモデル化が面倒である，という問題をもつ．

本稿では，上述の SOARS の問題点を解決するため，著者らが開発中の Stage-Oriented Agent Role Simulator Toolkit (SOARS Toolkit) の最新版を紹介する[1]．

[1]Spot-Oriented Agent Role Simulator Toolkit から改名した．

SOARS Toolkit は，Java で実装されたライブラリである．著者らは，文献 [13] で，SOARS Toolkit の初期版を提案し，オリジナルの SOARS よりも約 240 倍の処理速度を達成している．SOARS Toolkit の最新版では，初期版の機能に加えて，1) 日を跨いでルールの実行時刻を指定する機能，2) ルールの実行条件としてステージのみを指定する機能，3) ルールの動的スケジューリング機能，4) エージェントとスポットの動的追加・削除機能，5) ロール及びオブジェクトのモジュール化機能，6) 並列化支援機能，がサポートされている．

SOARS Toolkit 最新版の実行性能を調査するため，単純な都市モデル [17, 19, 18] と，感染シミュレーションモデルの 1 つであるフィルタモデル [17, 19, 18] を，SOARS Toolkit を用いて並列実装した上で，首都圏規模を想定した 3,000 万人／ 1,415 万スポット，および，日本規模を想定した 1 億人/4,717 万スポットでの実行性能を評価する数値実験を行う．本実験の規模は，現時点で世界最大級である．既存研究では，数千〜数万人規模の研究がほとんどであり，数十万人規模の研究はごく少数である [14]．我々が調査した限りでは，独自に専用シミュレータを構築して行われた 2,400 万人の COVID-19 の感染シミュレーション [15] が最大規模である．

以下，2 節では，ロール・ステージモデルを紹介する．3 節では，SOARS Toolkit 最新版を紹介する．4 節では，都市モデルとフィルタモデルの並列実装の実行性能を評価する実験を行う．5 節は，本稿のまとめと今後の課題である．

2　ロール・ステージモデル

ロール・ステージモデルは，オブジェクトの振る舞いをロール，時間をステージにより定義するエージェントベースモデルの 1 つである．以下では，オブジェクト，ロール，ロールの構成要素であるルール，ステージの概念を順に説明した後，ルールの実行に関する制約条件，モデルの実行の流れについて順に説明する．

2.1　オブジェクト

図 1 に示すように，オブジェクトには，スポット，エージェント，トランスポーテーションがある．オブジェクトの振る舞いは，ロールによって定義される．図 2 に示すように，オブジェクトは，0 個以上のロールを持つことができる．オブジェクトが 2 個以上のロールを持つ場合は，高々1 つのロールしかアクティブにならない．ただし，2.2 節で述べるように，複数のロールを合成して 1 つのロールにすることができるため，ロール合成を使えば，2 つ以上のロールをアクティブにすることができる．

スポットは，エージェントが相互作用する場を表す．家や部屋，建物といった空間的な場に加えて，社会の組織やグループなどの抽象的な集合もスポットとして表現することができる．

エージェントは，行動主体を表す．エージェントは，ある時点で 1 つのスポットに存在し，スポット間を移動することができる．

トランスポーテーションは，公共交通機関を表し，複数のエージェントを乗せてスポット間を移動することができる．トランスポーテーションは，スポットの性質とエージェントの性質を両方持つオブジェクトといえる．

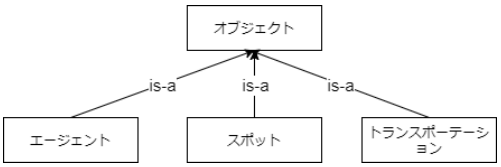

Fig. 1: オブジェクトの種類

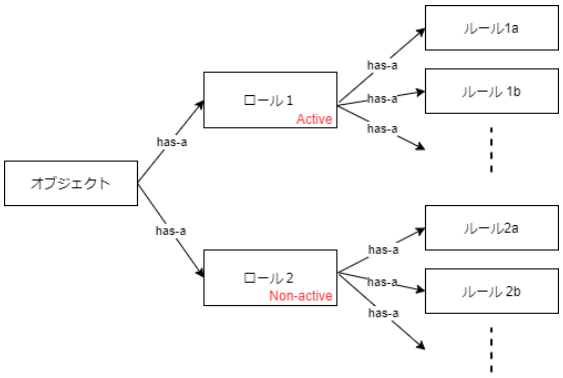

Fig. 2: オブジェクト，ロール，ルールの関係：オブジェクトは 0 個以上のロールをもつ．ロールは 0 個以上の属性と 0 個以上のルールをもつ．同時にアクティブになれるロールは高々1 つである．

2.2　ロール

ロールは，オブジェクトの振る舞いを決める．図 2 に示すように，ロールは，属性とルールの集合として定義される．属性は，ロールの内部状態変数である．ルールは，オブジェクトの振る舞いの最小単位であり，オブジェクトの動作を定義する．ルールの詳細については，2.3 節で説明する．

図 2 に示すように，オブジェクトは，複数のロールを持つことができ，適宜ロールを切り替えることができる．たとえば，家では父親のロール，会社では課長などの役職のロール，病院では病人のロールを選択することにより，エージェントに状況に応じた動作をさせることができる．

ロールは，複数のルールを定義する方法に加えて，複数のロールを合成することにより定義することができる．

2.3　ルール

ルールは，オブジェクトの振る舞いの最小単位であり，オブジェクトの動作を定義する．ルールの実行タイミングは，時刻とステージ（2.4 節を参照）で指定される．ただし，ルールは，自身が属するロールがアクティブであるときのみ実行される．図 2 においては，ロール 1 に属するルールは実行されるが，ロール 2 に属するルールは実行されない．

ルールは，実行タイミングの観点から以下の 3 種類に分類される．

- 定時実行ルール（regular rule）
  毎日決まった時刻に繰り返して実行されるルールである．実行時刻として hh:mm 形式を指定する．ここで，hh は時，mm は分を表す．

- 臨時実行ルール（temporary rule）
  指定された時刻に 1 回だけ実行されるルールである．あるルールの実行に依存して実行時刻が決ま

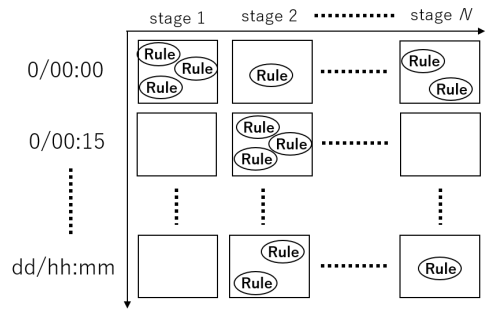

Fig. 3: 15 分ティック，N ステージの場合：各時刻において，stage 1，･･･，stage N の順にルールが実行される．同一時刻，同一ステージに属するルールは順不同で実行される．

るルールが相当する．実行時刻として dd/hh:mm 形式を指定する．ここで，dd は日を表す．同一日に実行したいルールについても dd/hh:mm 形式を指定する．

- ステージ実行ルール (stage rule)
  毎時刻の指定されたステージに繰り返して実行されるルールである．

ルールの動作としては，現在の時刻とステージ，自身および他のオブジェクトの状態，グローバル共有変数の値に応じて，自身と他のオブジェクトの状態変数の更新，ロールの切り替え，ルールの実行のスケジューリング，スケジューリング済みのルールのキャンセル，などを記述できる．エージェントの場合は，スポット間の移動も記述できる．ただし，他のオブジェクトの状態変数およびグローバル共有オブジェクトの更新は，並列化の際にボトルネックになるため，なるべく避けることが望ましい．

2.4 ステージ

ステージは，時間が離散値として扱われるエージェントベースシミュレーションにおいて，同一時刻に実行される複数のルールの実行順序を制御するための概念である．図 3 に示すように，各時刻は，順序が定義された複数のステージから構成される．ルールの実行条件としてステージを指定することにより，同一時刻内でのルールの実行順序を制御できる．

2.5 ルールの実行に関する制約条件

ロール・ステージモデルでは，ルールの実行に以下の 2 つの制約を課す．このため，ステージ内のルールの並列実行を比較的容易に実現可能である．

- あるオブジェクトが同一時刻の同一ステージに実行できるルールは高々1 個である．
- 同一時刻の同一ステージで実行される複数のルールはその実行順序によって実行結果が変わらない．

2.6 モデルの実行の流れ

ロール・ステージモデルの実行の流れは以下のとおりである．

1. ステージリストを構築する．
2. オブジェクトを生成する．
3. ロールを生成して，オブジェクトへの割当を行う．ロールの生成時に，定時実行ルールおよびステージ実行ルールがスケジュールされる．
4. 時刻を初期化する．
5. ステージリストに登録されたステージの順に以下を実行する．
   (a) 現在ステージにスケジュールされたステージ実行ルールを全て実行する．ただし，そのルールの属するロールが非アクティブである場合は実行しない．各ルール実行時に，必要に応じて，次ステージ以降に臨時実行ルールがスケジュールされる．
   (b) 現在時刻，現在ステージにスケジュールされた定時実行ルールを全て実行する．ただし，そのルールの属するロールが非アクティブである場合は実行しない．各ルール実行時に，必要に応じて，次ステージ以降に臨時実行ルールがスケジュールされる．
   (c) 現在時刻，現在ステージにスケジュールされた臨時実行ルールを全て実行する．ただし，そのルールの属するロールが非アクティブである場合は実行しない．各ルール実行時に，必要に応じて，次ステージ以降に臨時実行ルールがスケジュールされる．
6. 終了条件が満たされていなければ，時刻を 1 ティック進めて Step 5 へ戻る．

3 SOARS Toolkit

3.1 特徴

SOARS Toolkit は，Java のクラスライブラリとして提供される．そのため，Java 言語のもつ柔軟さ，実行の高速さ，豊富なライブラリ，Maven などのビルドツールとリポジトリ，その他の豊富なエコシステムをそのまま享受できる．SOARS Toolkit とその他のモジュール，および，それらのサンプルコードは，GitHub から入手可能である [16]．SOARS Toolkit[2] とその他のモジュールは，GitHub 上で Maven リポジトリとして公開されている．

SOARS Toolkit の特徴を以下にまとめる．

3.1.1 ロール・ステージモデルの実装の容易さ

SOARS Toolkit は，2 節で紹介したロール・ステージモデルに対応したクラスを提供している．SOARS Toolkit では，オブジェクトに対応する TObject クラス，エージェントに対応する TAgent クラス，スポットに対応する TSpot クラス，トランスポーテーションに対応する TTransportation クラス，ロールに対応する TRole クラス，ルールに対応する TRule クラス，時刻に対応する TTime クラスを提供が提供されている．SOARS Toolkit では，ステージは文字列で表現される．その他，スケジュールされたルールの管理と実行を行う TRuleAggregator クラス，エージェントの生成と管理を行う TAgentManager クラス，スポットの生成と

[2] 2022 年 1 月 30 日現在の公開版は並列化支援機能はサポートされていない．並列化支援機能をサポートする SOARS Toolkit は近日中に公開予定である．

管理を行う TSpotManager クラス，シミュレーションを統括する TModel クラス，などが提供されている．

SOARS Toolkit を用いた実装では，ユーザは，ロール・ステージモデルでモデル化されたシナリオを参照しながら，まず，TRule クラスを継承してルールを，TRole クラスを継承してロールを実装する．次に，メインメソッドのテンプレートコードを修正することにより，独自のステージリストの構築，オブジェクトの生成，ロールの生成とオブジェクトへの割当て，を書けばよい．ユーザは，メインメソッドの中で，ステージを実行するたびに実行されるコードを自由に記述することができる．

3.1.2 公共交通機関モデルの提供

SOARS Toolkit は，オリジナルの SOARS では実装が困難であった公共交通機関モデルとしてトランスポーテーションを提供している．トランスポーテーションを用いることで，鉄道，バス，飛行機，船を表現できる．

本モデルは，駅（空港，バス停，港で読み替え可能），トランスポーテーション，トランスポーテーションロール，トランスポーテーション管理，トランスポーテーションへの乗車ルール，トランスポーテーションからの下車ルールから構成される．

駅とトランスポーテーションは，スポットを拡張したものであり，トランスポーテーションはユーザが指定した路線と時刻表に従って駅間を移動する．

トランスポーテーションロールはトランスポーテーションの動作を定義するものであり，トランスポーテーション生成ルール，トランスポーテーション削除ルール，トランスポーテーション到着ルール，トランスポーテーション出発ルールから構成される．トランスポーテーション生成ルールは，指定された時刻に始発駅にトランスポーテーションを生成するルールである．トランスポーテーション削除ルールは，終着駅に到着したトランスポーテーションを削除するルールである．トランスポーテーション出発ルールは，指定された時刻に駅を出発するルールである．トランスポーテーション到着ルールは，指定された時刻に駅に到着するルールである．本トランスポーテーションモデルでは，時刻 t_1 に駅 A を出発して，時刻 t_2 に駅 B に到着するトランスポーテーションは，時刻 t_1 と t_2 の間は駅 A と駅 B の間に存在するものとして扱われる．

トランスポーテーション管理は，ユーザが指定した路線と時刻表に基づいてトランスポーテーションロールを構成してトランスポーテーションに登録する．

トランスポーテーションへの乗車ルールとトランスポーテーションからの下車ルールは，トランスポーテーションを利用するエージェントのためのルールである．トランスポーテーションへの乗車ルールの実行条件としては，1) 乗車時刻，乗車駅，乗車路線，乗車方面（上り・下りなど），トランスポーテーション名を指定して乗車する方法と，2) 乗車駅，乗車路線，乗車方面，トランスポーテーション名のみを指定して最初に来たトランスポーテーションに乗車する方法の 2 通りが用意されている．

3.1.3 モジュール化支援機能の提供

SOARS Toolkit では，モジュール化とその再利用機能を提供している．この機能を用いてあらかじめ様々なロールをライブラリとして用意することにより，それらを組み合わせるだけで高度なシミュレーションを簡単に実現できる．たとえば，人流をシミュレーションする都市モデルモジュール，感染モデルモジュール，災害モデルモジュールがリポジトリ上で公開されている場合，これらを組み合わせて，災害時の感染対策のシミュレーションを素早く構築することができる．これは，1) 各モジュールを利用できるように Maven の pom.xml を適宜修正する，2) 各モジュールで定義されているステージを適切な順序で並べるステージ合成を行う，3) ロールの切り替えを行うステージとルールを作成する，4) 必要ならば各モジュールのロールに追加・上書きするためのルールを作成する，5) 必要ならばロール生成時にルールの追加・上書き・削除を行ったうえで，エージェントとスポットに生成されたロールを割り当てる，ことにより実現できる．ここで，ロールへのルールの追加・上書き・削除機能は，TRole クラスのメソッドとして提供されている．

3.1.4 並列化支援機能の提供

ロール・ステージモデルでは，1) あるオブジェクトが同一時刻の同一ステージに実行できるルールは高々 1 個である，2) 同一時刻の同一ステージで実行される複数のルールはその実行順序によって実行結果が変わらない，ことになっている．そこで，SOARS Toolkit では，実行時に，ステージごとに逐次実行と並列実行を選択できるようになっている．

ただし，ルール内で，そのルールが属するロールが割り当てられているオブジェクト以外のロールの属性値を変更している場合，および，グローバル共有変数の値を変更している場合は，Java の同期機構を用いて，変数が複数のスレッドから同時に変更されないように，コードを修正する必要があることに注意されたい．

並列化時にも実行結果の再現性を実現するため，各オブジェクトに乱数系列を割り当てられるようになっている．あるオブジェクトが同一時刻の同一ステージに実行できるルールは高々1 個であるため，各オブジェクトが利用する乱数系列は，乱数シードを揃えておけば同じになる．

3.1.5 デバッグ支援機能の提供

ロール・ステージモデルおよびその実装のデバッグを行うためには，各ルールが正しいタイミングで実行されているかを確認するのが効果的である．そのための機能として，SOARS Toolkit では，実行時のルールのロギング機能を提供している．ログは，CSV 形式で出力され，現在時刻，現在ステージ，現在のスポット（エージェントのみ），ルール名，ルールの属するロールの名前，ロールの属するオブジェクトの名前，ルールの登録／実行／削除，ルール実行予定時刻，ルール実行予定ステージ，ユーザ定義のデバッグ情報などが出力される．ルールのログファイルのサイズは非常に大きくなるため，必要な時だけロギングの機能を有効にできるようになっている．

3.2 主なクラス

3.2.1 soars.core パッケージ

soars.core パッケージは，SOARS Tool の中核となるパッケージであり，シミュレータを書く際に必要と

なるクラスが収められている．

soars.core パッケージに含まれる主なクラスは以下のとおりである．

- TObject クラス
  TObject クラスは，オブジェクトを表すクラスであり，スポット，エージェント，トランスポーテーションの基となるクラスである．オブジェクトは，0 個以上のロールをもつ．TObject クラスを継承するクラスとして，TAgent クラス，TSpot クラス，TSpot クラスを継承した TTransportation クラスがある．

- TSpot クラス
  TSpot クラスは，スポットを表すクラスである．TObject クラスを継承している．スポットは，そのスポットにいるエージェント集合をもつ

- TSpotManager クラス
  TSpotManager クラスは，スポットを生成・管理するためのクラスである．本稿では「スポット管理」とよぶ．

- TAgent クラス
  TAgent クラスは，エージェントを表すクラスである．TObject クラスを継承している．エージェントは，現在位置するスポットの情報をもつ．

- TAgentManager クラス
  TAgentManager クラスは，エージェントの生成・管理を行うためのクラスである．本稿では「エージェント管理」とよぶ．

- TRole クラス
  TRole クラスは，ロールを表すクラスである．全てのロールはこのクラスを継承しなければならない．TRole クラスには，ロールをアクティブ化／非アクティブ化するメソッド，ルールを登録・削除・上書きするメソッド，他のルールをマージするメソッドなどのメソッドが定義されている．

- TRule クラス
  TRule クラスは，ルールを表す抽象クラスである．全てのルールはこのクラスを継承しなければならない．ルールには，定時実行ルールとしてスケジューリングするためのメソッド，臨時実行ルールとしてスケジューリングするためのメソッド，ステージ実行ルールとして実行するためのメソッド，キャンセルするためのメソッド，ルール実行のための抽象メソッド，自分が属するロールを取得するメソッド，乱数発生器を取得するメソッド，などが定義されている．

  ユーザは，TRule クラスを継承して独自のルールクラスを定義する．その際，ルール実行のための抽象メソッドをオーバーライドする．この抽象メソッドには，実行時の現在時刻，現在ステージ，エージェント管理，スポット管理，グローバル共有変数集合が引数として渡される．ユーザは，これらの情報を使って，自身と他のオブジェクトの状態変数の更新，ロールの切り替え，ルールの実行のスケジューリング，スケジューリング済みのルールのキャンセル，などを記述できる．エージェントの場合は，スポット間の移動も記述できる．

- TAgentRule クラス
  TAgentRule クラスは，エージェント用のルールを表すクラスであり，TRule クラスを継承している．スポット間の移動など，エージェントの動作に特有のメソッドが追加されている．

- TRuleAggregator クラス
  TRuleAggregator クラスは，ルール収集器を表すクラスである．ルール収集器は，実行時刻／ステージが決まっている全てのルールを保持し，ステージごとにルールをまとめて逐次実行／並列実行するためのクラスである．ルールは，全てのルールが継承している TRule クラスで定義されているスケジューリング用のメソッドにより登録される．

- TModel クラス
  TModel クラスは，モデルを表すクラスであり，シミュレーションを実行するのに必要な，TSpotManager クラス，TAgentManager クラス，TRuleAggregator クラス，乱数生成器，グローバル共有変数などを管理する．

- TTime クラス
  TTime クラスは，時間を表すクラスである．TTime クラスは，時間を分単位で管理している．TTime クラスには，dd/hh:mm 形式と hh:mm 形式の表現形式がある．ここで，dd は日，hh は時，mm は分を表す．

3.2.2 soars.transportation パッケージ

soars.transportation パッケージは，公共交通機関を利用する際に必要なクラスが収められたパッケージである．

soars.transportation パッケージに含まれる主なクラスは以下のとおりである．

- TStation クラス
  TStation クラスは，駅を表すクラスである．TSpot クラスを継承している．TStation クラスは，soars.transportation パッケージの他のクラスから利用され，ユーザが直接操作することはほとんどないと思われる．

- TTransportation クラス
  TTransportation クラスは，トランスポーテーションを表すクラスである．TSpot クラスを継承している．TTransportation クラスは，始発時刻になると始発駅に配置され，スポット管理クラス（TSpotManager）が管理しているスポット集合に登録され，終着駅に到着するとスポット集合から削除される．したがって，あるトランスポーテーションが運行中かどうかは，スポット集合にそのトランスポーテーションが登録されているかどうかを調べればよい．TTransportation クラスは，soars.transportation パッケージの他のクラスから利用され，ユーザが直接操作することはほとんどないと思われる．

- TTransportationManager クラス
  TTransportationManager クラスは，駅，トランスポーテーションとその運行に関する情報が書かれたファイルを読み込み，駅およびトランスポーテーションの生成，トランスポーテーション運行のためのルール生成，トランスポーテーションの管理を行うクラスである．

- TGettingOnTransportationRule クラス
  TGettingOnTransportationRule クラスは，エージェントがトランスポーテーションに乗車するためのルールであり，TAgentRule クラスを継承している．乗車するトランスポーテーションの指定方法としては，2 つの方法が用意されている．方法 1 は，乗車時刻，乗車路線名，乗車駅，乗車方面（上り／下り，内回り／外回りなど），乗車トランスポーテーション名を指定する方法である．方法 2 は，先行ルール（多くの場合は「駅への到着」ルールだと思われる），乗車駅，乗車路線，乗車方面，トランスポーテーションタイプ（普通，急行，全てなど），行き先を指定する方法である．

- TGettingOffTransportationRule クラス
  TGettingOffTransportationRule クラスは，エージェントがトランスポーテーションから降車するためのルールであり，TAgentRule クラスを継承している．降車の方法も 2 つ用意されている．方法 1 は，降車時刻，降車駅，路線，方面，トランスポーテーション名を指定する方法である．方法 2 は，対となる乗車のための TGettingOnTransportationRule クラスのオブジェクト，降車駅を指定する方法である．

3.3 メインメソッドの実装

SOARS Toolkit は，上述のように，ロール・ステージモデルを実装するためのクラスライブラリを提供するものであり，メインメソッドの実装は，テンプレートソースコードを提供したうえで，ユーザに委ねている．これにより，ユーザは，外部のライブラリやプログラムとの連携，ロギングを柔軟に行うことができるようになっている．

リスト 1 にメインメソッドの簡単なサンプルソースコードを示す．リスト 1 では，まず，3〜7 行目で，ステージとモデルを初期化している．次に，9〜12 行目で，スポットの初期化を行っている．ここでは，スポット管理（TSpotManager）をモデルから取得して，スポット管理を利用して，3 個の自宅 (HOME) と 1 個の会社 (CAMPANY) を生成している．次に，14〜25 行目で，エージェントを初期化している．ここでは，14 行目でエージェント管理（TAgentManager）を取得した後，15〜17 行目でエージェント管理を利用して 3 個の父親エージェントを生成している．19 行目で各父親エージェントを取り出している．20 行目で自宅名を生成して，21 行目で父親ロール（TFatherRole）を生成して父親エージェントに割り当て，22 行目で父親ロールをアクティベートしている．23 行目でスポット管理から自宅スポットを得て，24 行目で父親エージェントの初期スポットとして設定している．最後に，28〜38 行目がメインループであり，31 行目で時刻を 1 ティック進めている．

リスト 1: メインメソッドの例

```
1  public static void main(String[] args) {
2    //ステージの初期化
3    List<String> stages = List.of(TStages.MOVING);
4    int interval = 60; //60分ティック
5    long seed = 0; //乱数シード
6    //モデルの生成
7    TModel model = new TModel(stages, interval, seed);
8    //スポットの初期化
9    int noOfHomes = 3; //家の数
10   TSpotManager spotMgr = model.getSpotManager();
11   spotMgr.createSpots(TSpotTypes.HOME, noOfHomes);
12   spotMgr.createSpot(TSpotTypes.COMPANY);
13   // エージェントの初期化
14   TAgentManager agentMgr = model.getAgentManager();
15   ArrayList<TAgent> fathers
16     = agentMgr.createAgents(TAgentTypes.FATHER,
17                             noOfHomes);
18   for (int i = 0; i < fathers.size(); ++i) {
19     TAgent f = fathers.get(i);
20     String sn = TSpotTypes.HOME + (i + 1);
21     TFatherRole fr = new TFatherRole(father, sn);
22     f.activateRole(fr.getName());
23     TSpot home = spotManager.getSpotDB().get(sn);
24     f.initializeCurrentSpot(home);
25   }
26   // メインループ
27   // 0日0時0分から6日23時まで1時間単位でまわす
28   TTime endTime = new TTime("7/0:00"); //終了時刻
29   while (model.getTime().isLessThan(endTime)) {
30     System.out.print(model.getTime() + "\t");
31     model.execute(); // モデルの実行
32     for (TAgent a : fathers) {
33       //各エージェントがいるスポット名を表示する
34       System.out.print(a.getCurrentSpotName());
35       System.out.print("\t");
36     }
37     System.out.println();
38   }
39 }
```

4 実行性能の評価実験

本節では，単純都市モデル [17, 19, 18] と COVID-19 フィルタモデル [17, 19, 18] を SOARS Toolkit で並列実装し，その実行性能を評価する．

単純都市モデルの構成は以下のとおりである．スポットの種類としては，1 人世帯自宅，2 人世帯自宅，3 人世帯自宅，4 人世帯自宅，5 人世帯自宅，6 人世帯自宅，7 人世帯自宅，8 人世帯自宅，9 人世帯自宅，10 人世帯自宅，小学校，高校，小規模オフィス，中規模オフィス，大規模オフィス，交通チャンネル，病院，墓場がある．エージェントの種類としては，赤ちゃん，小学生，中学・高校生，社会人若年層，社会人中年層，社会人老年層がある．単純都市モデルでは，就労者や学生は交通チャンネル（車両の収容人員をもった電車の車両やバスのような場）を経由して通勤，通学し，感染者は病態によって病院へと収容される．

図 4 に COVID-19 フィルタモデルの病態遷移モデルを示す．本モデルでは，感染レベルとして，レベル 1，レベル 2，レベル 2m，レベル 3，レベル 3s，レベル 3m，レベル 4m，レベル 4c，レベル 5，レベル D が定義されており，レベル 1 から 2 へ移る確率 p_{12}，レベル 2 から 3 へ移る確率 p_{23}，レベル 3s から 4m へ移る確率 p_{3s4m}，レベル 4c からレベル D へ移る確率 p_{4cD} が，それぞれのエージェントの種類に対して定義されている．ウイルスはエージェントからスポット，スポットからエージェントへと伝染すると仮定している．人から場，場から人の感染プロセスには複数のフィルタが存在する．

シミュレーション実行のためのハードウェアの環境としては，CPU Xeon Gold 6348 28Cores 2.6GHz × 2CPU，メモリ DDR4-3200 4TB，SSD NVMe M.2

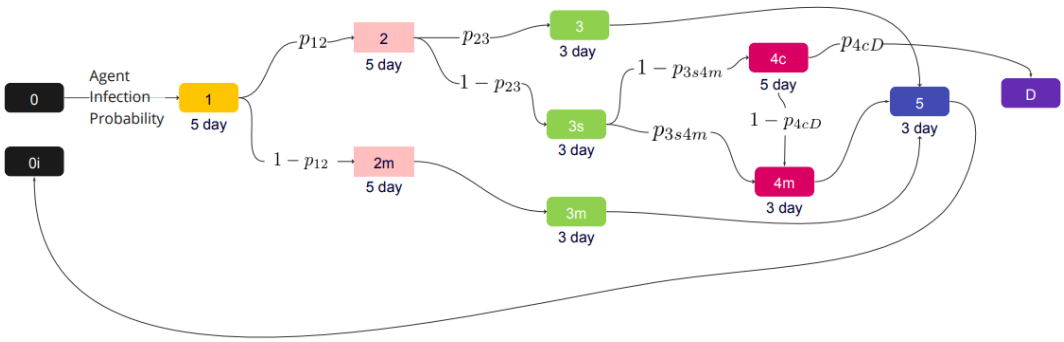

Fig. 4: 病態遷移モデル

Table 1: 3,000 万エージェント／1,415 万スポットの結果：オブジェクト初期化データ読込・生成にかかった時間と実行時の使用メモリ量

スポット初期化データ読込	1.87 秒
スポット生成	27.0 秒
エージェント初期化データ読込	8.78 秒
エージェント生成	259 秒
使用メモリ量	248GB

Table 2: 1 億エージェント／4,717 万スポットの結果：オブジェクト初期化データ読込・生成にかかった時間と実行時の使用メモリ量

スポット初期化データ読込	3.94 秒
スポット生成	93.4 秒
エージェント初期化データ読込	29.1 秒
エージェント生成	881 秒
使用メモリ量	769GB

1TB の計算サーバを用いた．ソフトウェアの環境としては，Ubuntu 20.04，Open JDK 11 を用いた．

実験設定としては，30 分ティックで 2 日間，3,000 万エージェント／1,415 万スポット，1 億エージェント／4,717 万スポットの 2 つの設定を用いた．3,000 万エージェント／1,415 万スポットは首都圏を想定した規模，1 億エージェント／4,717 万スポットは日本を想定した規模である．

表 1 と表 2 に，並列化の対象となっていない，スポットデータの読み込み，スポットの生成，エージェントのデータの読み込み，エージェントの生成にかかった時間と，使用メモリ量を示す．また，表 3 と表 4 に，スレッド数を 1〜56 に変化させた場合の 1 日 (48 ステップ) あたりのルール実行時間と 180 日実行にかかる時間の見積もりを示す．これより，首都圏規模を想定した 3,000 万人／ 1415 万スポット，30 分ティック 180 日のシミュレーションは，56CPU コアで並列実行すれば約 7.5 時間程度で終わると考えられる．一方，日本規模を想定した 1 億エージェント／ 4,717 万スポット，30 分ティック 180 日のシミュレーションは，56CPU コアで並列実行すれば約 24.5 時間で終わると考えられる．

図 5 と図 6 に，スレッド数を 1〜56 に変化させた場合のスピードアップ比のグラフを示す．これより，現状の実装では，56 スレッドで，3,000 万エージェントの場合に約 11 倍，1 億エージェントの場合に約 14 倍スピードアップしていることがわかる．並列実行性能のさらなる向上は今後の課題である．

Table 3: 3,000 万エージェント／1,415 万スポットの結果：スレッド数を 1〜56 に変化させた場合の 1 日 (48 ステップ) あたりのルール実行時間と 180 日実行にかかる時間の見積もり

スレッド数	1 日あたりの実行時間 (分)	180 日の実行時間の見積もり (時間)
1	27.5	82.6
2	18.1	54.3
4	10.2	30.5
8	6.26	18.8
16	3.66	11.0
32	2.50	7.49
56	2.47	7.42

2 日間における総ルール実行数は，3,000 万エージェント／1,415 万スポットの設定で 14,464,992,482，1 億エージェント／ 4,717 万スポットの設定で 48,216,640,482 であった．

実際に，首都圏規模を想定した 3,000 万人／ 1415 万スポット，30 分ティック 180 日のシミュレーション，および，日本規模を想定した 1 億エージェント／ 4,717 万スポット，30 分ティック 180 日のシミュレーションを 56CPU コアで並列実行した場合の実行時間を，表 5 に示す．これより，上記の実行時間の見積もりとほぼ同じになっていることが確認できる．

5 おわりに

本稿では，著者らが開発中の Stage-Oriented Agent Role Simulator Toolkit (SOARS Toolkit) を紹介し，1 億人規模 COVID-19 シミュレーションによる実行性能評価の結果について報告した．SOARS Toolkit は，ロール・ステージモデルの実装を容易にするための Java クラスライブラリであり，公共交通機関モデルの提供，モジュール化支援機能の提供，並列化支援機能の提供，デバッグ支援機能の提供を特徴とする．単純な都市モデルと，感染シミュレーションモデルの 1 つであるフィルタモデルを，SOARS Toolkit により並列実装し，首都圏規模を想定した 3,000 万人／ 1,415 万スポット，および，日本規模を想定した 1 億人/4,717 万スポットでの実行性能を評価する実験を行った．その結果，56CPU

Table 4: 1 億エージェント／4,717 万スポットの結果：スレッド数を 1〜56 に変化させた場合の 1 日 (48 ステップ) あたりのルール実行時間と 180 日実行にかかる時間の見積もり

スレッド数	1 日あたりの実行時間 (分)	180 日の実行時間の見積もり (時間)
1	114	341
2	75.0	225
4	37.2	112
8	20.9	62.7
16	12.9	38.8
32	8.54	25.6
56	8.17	24.5

Table 5: 56 スレッドで 180 日実行した場合の 3,000 万エージェント／1,415 万スポットと 1 億エージェント／4,717 万スポットのシミュレーションの実行時間の実測値

3,000 万エージェント／1,415 万スポット	1 億エージェント／4,717 万スポット
6.44 時間	25.6 時間

コアを用いて 180 日間 8,640 ステップを並列実行したところ，3,000 万人／1,415 万スポットのシミュレーションが 6.44 時間，1 億人/4,717 万スポットのシミュレーションが 25.6 時間で終わることを確認した．

今後の課題としては，並列化実行性能の向上，都市モデルモジュール，感染モデルモジュール，災害モジュールなどの各種モデルモジュールの充実と公開，モデリングを容易にするためのビジュアルモデリングツールの提供などを考えている．

謝辞

本研究は，戦略的イノベーション創造プログラム (SIP)「国家レジリエンス (防災・減災) の強化」（課題名：避難・緊急活動支援統合システムの研究開発）の支援を受けて行った．

参考文献

1) Axelrod, R.: The Complexity of Cooperation, Princeton University Press (1997).
2) 出口: エージェントベースモデリングによる問題解決ーエージェントベース社会システム科学としての ABM ー, 日本オペレーションズ・リサーチ学会，49-3, 161/167 (2004).
3) https://ccl.northwestern.edu/netlogo/
4) https://cs.gmu.edu/ eclab/projects/mason/
5) https://repast.github.io/
6) http://cormas.cirad.fr/indexeng.htm
7) https://gama-platform.org/
8) https://mas.kke.co.jp/en/artisoc4/
9) http://www.msi.co.jp/s4/
10) 田沼, 出口:エージェントベース社会シミュレーション言語 SOARS の開発，電子情報通信学会論文誌 D，J90-D-9, 2415/2422 (2007).
11) 市川，後藤：シミュレーション言語の特徴と比較，計測と制御，52-7, 595/600 (2013).

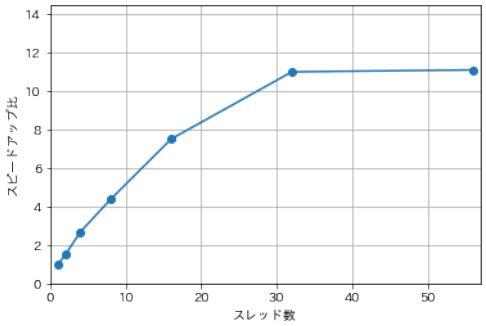

Fig. 5: 3,000 万エージェント／1,415 万スポットの結果：スレッド数 vs. スピードアップ比

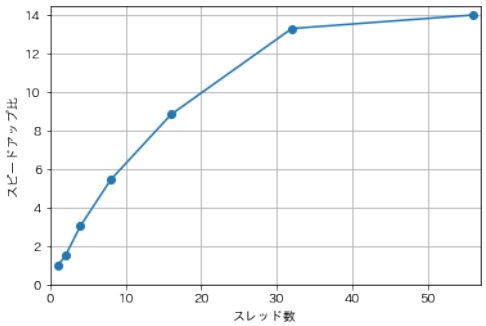

Fig. 6: 1 億エージェント，／4,717 万スポットの結果：スレッド数 vs. スピードアップ比

12) Kurata, T., Deguchi, H. and Ichikawa, M.: GUI for agent based modeling, Proc. the International Conference on Social Modeling and Simulation, plus Econophysics Colloquium 2014, 275/286, Springer International Publishing (2015).
13) 小野，市川，出口: 大規模エージェントベースシミュレーションのための SOARS Toolkit の提案，計測自動制御学会システム・情報部門学術講演会 2020(SSI2020) 講演予稿集 (2020).
14) Li, J. and Giabbanelli, P. J.: Returning to a normal life via COVID-19 vaccines in the USA: a large-scale agent-based simulation study, JMIR Med Inform 9(4):e27419 (2021).
15) Chang, S. L., Harding, N., Zachreson, C., Cliff, O. M. and Prokopenko, M.: Modelling transmission and control of the COVID-19 pandemic in Australia, Nature Communications volume 11, Article number: 5710 (2020).
16) https://github.com/soars-jp
17) Deguchi, H., Kanatani, Y., Kaneda, T., Koyama, Y., Ichikawa, M., and Tanuma, H.: Anti pandemic simulation by SOARS. 4581/4586 (2006).
18) Ichikawa, M., Tanuma, H., and Deguchi, H. : Infectious Disease Simulation Model for Estimation of Spreading, Developments in Business Simulation and Experiential Learning, 38, 358/366 (2011).
19) 出口，田沼，金谷，斎藤，兼田，小山，市川: 感染症対策の机上演習マニュアル — SOARS によるシミュレーション疫学入門 — (2008).