特別研究報告書

大規模交通シミュレーションのための可視化ツールの開発

指導教員　　石田 亨 教授

京都大学工学部情報学科

島田 裕司

平成16年4月入学
平成20年3月卒業

大規模交通シミュレーションのための可視化ツールの開発

島田 裕司

内容梗概

近年，複雑な社会システムを理解・分析するためのアプローチとして，マルチエージェントシミュレーションが注目を集めている．マルチエージェントシミュレーションは，システムを構成する個々の要素を，個別の内部状態と意思決定のための任意の関数を持ったエージェントとして定義し，複数のエージェント間の相互作用によって全体の挙動を生み出すボトムアップ的なアプローチを採る．マルチエージェントシミュレーションの適用分野は多岐にわたるが，有望な適用先として交通問題が挙げられる．交通では，車両等の多数の構成要素が相互作用する事によって，複雑な交通現象を創発すると見ることができる．個々の車両をエージェントとしてモデル化することで，様々な交通現象を再現したり，都市交通を理解・分析するための研究が行われている．

マルチエージェントに基づく交通シミュレーションでは，目的地への到着台数や，目的地までの所要時間といった，シミュレーションの結果得られる数値データを出力するだけでは不十分である．交通現象の理解・分析のためには，シミュレーションの過程で発生する現象を容易に観測できることが求められる．特に，都市全域に関する交通状況を対象とした大規模な交通シミュレーションを行う場合，視覚的に状況が把握できる機能の提供が必須と考えられる．例えば，時々刻々と変化する道路の混雑状況が可視化される事で，シミュレーションの過程での交通状況を広範囲に把握しやすくなる．

交通に関する分析のためには，車両挙動の変化による全体の交通への影響や，交通施策や事故などによる交通環境の変化による影響が観測できる事が望ましい．そのため，可視化されたシミュレーション状況に基づき，シミュレータの挙動を直接操作する機能，すなわち (1) 車両の挙動を人間が直接操作するシミュレーションへの参加機能，および (2) 交通環境を任意に操作する機能が求められる．

本研究では，マルチエージェントシミュレーションに基づく交通シミュレーションに関連して，シミュレートする交通状況の理解・把握を容易化し，かつ分析のためのシミュレーション操作の機能を提供可能な，マルチエージェント交通シミュレーション可視化ツールの設計，および実装を行う．ここでは特に，

日本 IBM 東京基礎研究所との共同研究によって開発された，数百万規模のエージェントが動作可能な大規模マルチエージェント交通シミュレータを用いて，以下の機能を備えた可視化ツールの設計と実装を行う．

- 交通シミュレータ-クライアントソフトウェア（ビューワ）間でのデータ変換および送受信機能
- 交通シミュレータの挙動の外部定義と割り込み実行機能

ユーザはクライアントソフトウェアを通して，シミュレーション状況の確認や，車両の操作，道路の操作を行う．

このような可視化ツールを実現するため，交通シミュレータからのログデータを管理してクライアントソフトに表示したり，クライアントからの要求に応じて交通シミュレータへユーザの操作を反映させたりするための通信サービスを実装した．通信サービスは，シミュレータからのログデータからクライアントの要求に従って必要な情報を選別する機能，交通シミュレータの車両の経路選択を行う機能，そしてクライアントとの通信を行うサーバ機能を持つ．これにより，可視化ツールに以下のような機能を実現した．

- シミュレーション状況の可視化機能．
  クライアントは通信サービスが解析したログを受け取り，ユーザインタフェース上に表示する．この機能により，シミュレーション実行過程において発生する様々な交通現象を容易に観測することが可能となった．
- シミュレーションの操作機能．
  シミュレータの車両の経路選択の部分を外部で定義できるように交通シミュレータの拡張を行った．これにより，車両エージェントを操作することが可能となった．さらに，ユーザが指定した道路を経路選択時に常に選択対象外とすることで，仮想的に道路の閉鎖機能を実現した．これらにより，シミュレーションの車両や環境を操作したときに，全体の交通状況にどのような影響を与えるか観測することが出来る．

以上のような可視化ツールを設計・実装し，クライアントソフトの情報の取得に関わるパフォーマンスを計測した．その結果，通信サービスが情報を選別することにより，大規模シミュレーションの可視化を実用的な実行速度で実現できることがわかった．

Visualization Tool for a Massively Multiagent Simulation

Hiroshi Shimada

Abstract

Recently, multiagent simulation attracts attention as a approach to understand and analyze complex social system. Multiagent simulation is a bottom-up approach that define several factors composing the system as a agent which has each internal state and any function for decision-making and cause whole behavior by the interaction between some agents. Multiagent simulation is applied to many fields, and one of them is traffic. Some complex traffic phenomena are occurred by the interaction between many factors such as vehicles. By modeling each vehicle as a agent, urban traffic is analyzed and various traffic phenomena are reproduced.

At traffic simulation based on multiagent, it isn't enough to output numerical data such as the number of vehicles that arrive the goal or the necessary time to the goal. To observe some phenomena occurred in the process of simulation easily is required to understand and analyze traffic phenomena. Especially at the massively traffic simulation about the whole urban traffic, the function that understand the state of simulation by visual information is necessary. For example, to visualize the information of ever-changing traffic congestion make to understand the state of traffic on the process of the simulation widely ease.

To analyze traffic, to observe the influence that vehicle or traffic phenomena such as a traffic policy or an accident give the traffic state is to be desired. So the function that operate the behavior of the simulation directly is required. That is a function that human operates behavior of the vehicle and operate the traffic circumstance.

In this research, I develop a visualization tool for a traffic simulation based on multiagent simulation, which provides the function that understand a state of the simulation easily and operate the behavior of simulation to analyze traffic phenomena. This time especially I use a massively multiagent traffic simulator that is developed at joint research with IBM Tokyo Research Laboratory. It can move some million-scale agents. I develop a visualization tool which has followed functions.

- Converting, sending and receiving data between traffic simulator and client software.
- Defining the behavior of the traffic simulator from outside, and interrupt action.

User check a state of the simulation and operate vehicles and road state through the client software. To actualize the visualization tool like this, I implement the communication service that manage log data from traffic simulator, send it to client software and reflect user's operation to traffic simulator. Communication service have functions that select required data from log data from traffic simulator at client requests , select vehicle's route on traffic simulator and communicate with client software as a server. Through these function, I actualize followed functions to the visualization tool.

- Visualize the state of the simulation.
  Client software receives log which communication service and displays it on the user interface. Through this function, we can observe various phenomena occurred in the process of simulation easily.
- Operate the state of the simulation.
  I extend the traffic simulator to define a part of selecting vehicle's route on simulator from outside. Through this, we can operate vehicle agents. Furthermore through removing roads which user selects from choices at selecting route, we can close some roads. Through these functions, we can observe the influence that is occurred when user operates vehicles or surroundings on the simulation.

I develop a visualization tool like this, and measure the performance that client software get data from communication service. As a result, it is noticed that visualization of massively simulation can actualize with practical execution speed by selecting data to send at communication service.

大規模交通シミュレーションのための可視化ツールの開発

目次

第1章　はじめに

近年，複雑な社会システムを理解・分析するためのアプローチとして，マルチエージェントシミュレーションが注目を集めている．物理や科学の分野では，現象の理解のための手段としてシミュレーションが用いられ，既に多くの実績が存在する．これらのシミュレーションは，システム全体の挙動を既定する支配方程式を用いるトップダウン的なものである．一方，マルチエージェントシミュレーションは，システムを構成する個々の要素を，個別の内部状態と意思決定のための任意の関数を持ったエージェントとして定義し，複数のエージェント間の相互作用によって全体の挙動を生み出すボトムアップ的なアプローチを採る．実世界における人間を，シミュレーションにおけるエージェントに素直にマッピングできる事から，マルチエージェントシミュレーションは，様々な社会システム・メカニズムの分析・検証のツールとして適している．

マルチエージェントシミュレーションの適用分野は多岐にわたるが，有望な適用先として交通問題が挙げられる．交通では，車両等の多数の構成要素が相互作用する事によって，複雑な交通現象を創発すると見ることができる．個々の車両をエージェントとしてモデル化することで，様々な交通現象を再現したり，都市交通を理解・分析するための研究が行われている [1]．例えば，文献 [2] では，個々のエージェントが旅行時間を考慮した走行ルートの見直しを繰り返すマルチエージェント交通シミュレータを開発している．文献 [3] では，3 種類の異なる運転者モデルを用いた交通シミュレーションを実現している．また，交差点，踏切，高速道路の合流部，勾配が連続的に変化するサグなどで発生する，局所的な渋滞の対策を検討する際にも，マルチエージェントシミュレーションが用いられている [4, 5, 6].

マルチエージェントシミュレーションの一形態として，参加型シミュレーションがある．本論文における参加型シミュレーションとは，実世界に存在する人間が，仮想空間上のシミュレーションに入り込む事を可能にしたマルチエージェントシミュレーションの形態を指す．参加型のマルチエージェントシミュレーションでは，人間の参加者がゲームのような感覚で仮想空間上のキャラクターなどを操作し，内在的な視点からシミュレーションの状況を観測し，他のエージェントとインタラクションをしながら行動できる．参加型シミュレーションにより，実際的な状況下での人間の行動履歴の獲得が可能となる他，人間の振

る舞いにより，様々なシミュレーションの状況を生み出すことができる．

マルチエージェントに基づく交通シミュレーションでは，目的地への到着台数や，目的地までの所要時間といった，シミュレーションの結果得られる数値データを出力するだけでは不十分である．交通現象の理解・分析のためには，シミュレーションの過程で発生する現象を容易に観測できることが求められる．特に，都市全域に関する交通状況を対象とした大規模な交通シミュレーションを行う場合，視覚的に状況が把握できる機能の提供が必須と考えられる．例えば，時々刻々と変化する道路の混雑状況が可視化される事で，シミュレーションの過程での交通状況を広範囲に把握しやすくなる．

交通に関する分析のためには，車両挙動の変化による全体の交通への影響や，交通施策や事故などによる交通環境の変化による影響が観測できる事が望ましい．そのため，可視化されたシミュレーション状況に基づき，シミュレータの挙動を直接操作する機能，すなわち (1) 車両の挙動を人間が直接操作するシミュレーションへの参加機能，および (2) 交通環境を任意に操作する機能が求められる．

本研究では，マルチエージェントシミュレーションに基づく交通シミュレーションに関連して，シミュレートする交通状況の理解・把握を容易化し，かつ分析のためのシミュレーション操作の機能を提供可能な，マルチエージェント交通シミュレーション可視化ツールの設計，および実装を行う．ここでは特に，日本 IBM 東京基礎研究所との共同研究によって開発された，数百万規模のエージェントが動作可能な大規模マルチエージェント交通シミュレータを用いて，以下の機能を備えた可視化ツールの設計と実装を行う．

- 交通シミュレータ-クライアントソフトウェア（ビューワ）間でのデータ変換および送受信機能
- 交通シミュレータの挙動の外部定義と割り込み実行機能

以下本論文では，第 2 章において，本研究の背景であるマルチエージェントシミュレーションと先行研究について述べ，第 3 章で可視化ツールの設計の詳細について述べる．第 4 章では設計に基づく実装と具体的な実行例を示し，最後に第 5 章で，結論と今後の課題を論じる．

第2章　研究の背景

2.1　マルチエージェントシステム

本論文では，エージェントとはソフトウェアエージェントの事を指し，ある環境に存在し，かつ自身の目的を達成するための自律的動作が可能なソフトウェアと定義する．ここで ”自律的 ”であるとは，文献 [7] にある定義に従い，エージェントが，自身の内部状態と振る舞いを制御し，人間や他のシステムからの干渉無しに動作可能である事と定義する．

エージェントを構成要素とし，複数のエージェントが協調的に振る舞う事で問題解決を行うシステムをマルチエージェントシステムという．エージェントは，自身が置かれた環境の変化に対応して，柔軟に自律的な行動が可能である．なおかつ，エージェントは各々非同期に問題解決が可能である事から，マルチエージェントシステムは，大規模かつ複雑性の高い問題の解決に適している．

2.2　マルチエージェントシミュレーション

社会における様々な現象を模擬するための手段として，マルチエージェントシミュレーションは近年高い注目を集めている．マルチエージェントシミュレーションでは，個々の意思決定主体，例えば人間をエージェントとしてモデル化し，複数のエージェント間の相互作用によって様々な現象を再現する事により，新たなシステムの設計に役立つ分析・解析を行う．人間社会は，マルチエージェントシステムとしてのモデル化が容易である事から，種々の社会現象の理解・分析のための有用なアプローチと考えられる．マルチエージェントシミュレーションの応用範囲は多岐にわたる．経済現象や，世論形成などの社会現象の分析をはじめ，都市計画や交通問題の解析等に対する適用事例が存在する．

マルチエージェントシミュレーションを実現するために，エージェント間の社会的インタラクションや，多様な環境下での振る舞いを既定する必要がある．本研究では，シナリオとして記述された振る舞いに基づくエージェントを用いて，マルチエージェントシミュレーションを行う．

2.3　参加型シミュレーション

参加型シミュレーションとは，仮想空間上で行われるマルチエージェントシミュレーションに対し，一部のエージェントに変わって人間が参加する事で，シ

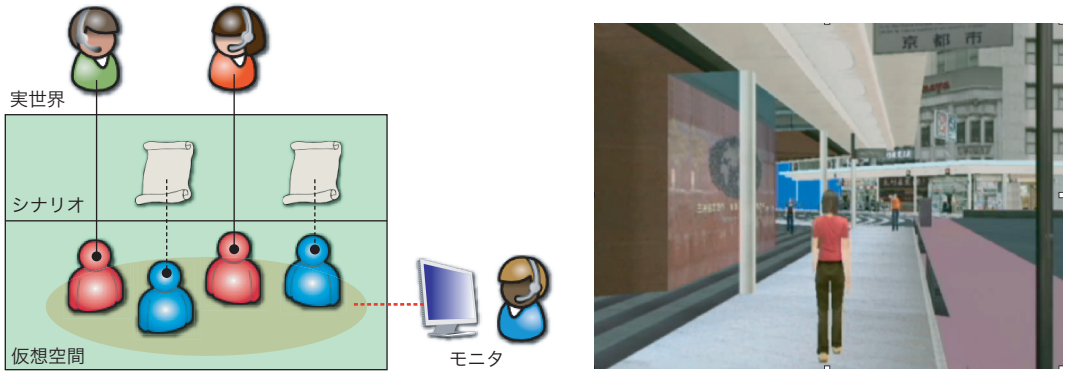

図 1: (a) 参加型シミュレーションの概要および (b) 実例（FreeWalk）

ミュレーションに実際の人間の行動を取り込む事が可能なシミュレーションの方式である．図 1(a) に参加型シミュレーションの概要を示す．エージェントは割り当てられたシナリオに基づいて機能する．人間は，アバターの操作を通してシミュレーションに参加し，仮想空間内で環境を知覚し，行動する．すなわち，内在的な視点からシミュレーションを観測する事が可能である．なお，参加型シミュレーションでは,（自律的に機能している）エージェントは，他のエージェントとアバターを区別することなく動作する．参加型シミュレーションの実行環境の例として FreeWalk[8] がある．Freewalk では，エージェントとアバターが仮想空間上に共存し，人間の参加者とエージェント間でのインタラクションが実現できる（図 1(b)）．

参加型シミュレーションにより，様々な環境下での参加者の実際的な振る舞いを観察できる．シミュレーションにおける行動ログから人間の行動モデルを抽出し，エージェントに与えるシナリオとしての活用が可能である．また，能動的にシミュレーション環境に作用し，シミュレーションに変化を与える事も可能である．すなわち，個のエージェントの特殊な振る舞いによって，シミュレーション全体にどのような影響が生まれるかをシミュレートする事で，可能性のある様々なケースについて分析する事ができる．

2.4 交通シミュレーション

マルチエージェントシミュレーションの具体的な適用分野として交通に関する分野がある．マルチエージェント交通シミュレーションに対するアプローチは，2 つに大別される．すなわち，(1) 車列の流れに注目するマクロ的なシミュ

レーション，(2) 個々の車両の挙動に注目するミクロシミュレーションである．

(1) では，車の流れを流体として捉え，運動法的式などを用いて，その動きをシミュレートするもので，車両密度，平均速度などをリアルタイムに計算するものである．例えば，交通施策の変更や交通環境の変化（事故による渋滞など）によって発生する交通流の変化の全容を掴みたい場合に適している．

(2) では，個々の車両の挙動をシミュレートすることにより，ボトムアップに交通の流れを再現するアプローチである．このミクロ的なアプローチでは，運転挙動の多様性を表現する事ができ，創発的な現象としての交通現象を再現することが可能となる．また，個々の車両の挙動変化による全体への影響と，全体の交通状態が個々の挙動に与える影響，すなわちミクロ-マクロの関係を分析する場合，ミクロ的なアプローチが適している．例えば，ある交差点において，右折レーンを付加した場合や，信号のタイミングを変更した場合などに，個々の車両がどのように挙動を変化させ，周辺の交通の流れがどのように変化するかを把握する，といった分析に対して有用である．

第3章　交通シミュレーション可視化ツールの設計

3.1　交通シミュレータ

本研究では，可視化の対象として，日本 IBM 東京基礎研究所と京都大学との共同研究によって開発された交通シミュレータを用いる．この交通シミュレータは，日本 IBM 東京基礎研究所で開発された大規模マルチエージェントシミュレーション環境 ZASE を基盤として，個々の運転者をエージェントとして実装したミクロシミュレーションを行うものである．交通シミュレータは各車両の位置と速度，道路情報，交通情報を管理し，それらのログデータを出力する機能を持つ．運転者エージェントは，現在の速度，前方の車との距離，道路情報をセンシングし，速度の変更と経路の選択を行う．

3.2　操作機能の設計

交通シミュレーションを行う際には，発生する交通現象を確認することが必要であるため，シミュレーションの結果だけではなく，シミュレーションの過程を観測することが必要である．また，作成したモデルの妥当性の確認や発生する交通現象の解明のためには，シミュレーション実行中，シミュレーション

の状況に外部から何らかの変化を与え，その後の現象を観測することで，影響を調べることが必要である．そのため，シミュレーションの状況を観測し，シミュレーションに参加することが可能なソフトウェアを設計する．

3.2.1 ユースケース

本研究で開発するソフトウェアを用いてシミュレーションに参加する参加者は，各車両を操作するため一般の運転者が想定される．また，ユーザとしては運転者とは別の立場として道路など環境を操作する管理者が存在する．それぞれについて，以下のようなユースケースを想定し，可視化ツールの設計を行った．

エージェントの操作．車両挙動の変化による全体の交通への影響を観測するため，シミュレーション内のエージェントを選択し，操作する．運転者ユーザは，交通状況を観測することにより，走行速度や経路の変更を行う．これらの判断は，交通シミュレーションの交通状況の観測に基づいて行う．例えば，前後を走行している車両との距離を見て速度を変更したり，全体の交通状況から最適と思われる経路を選択したりといったことが行われる．経路の変更は，経路上の交差点を GUI 上で指定し，続いてその交差点での進行方向を指定することで行う．また，速度の変更は随時行うことが出来る．

シミュレーション環境の操作．交通環境の変化による影響を観測するため，GUI を通して道路の閉鎖と開放を行う．広域視点，拡大視点の両方において，表示されている道路を選択し，その道路を通行止め（両方向，片方向，指定車線）する．そして，その後の交通状況の変化，例えば，通行止めにした道路の周辺にいる車両の挙動や，広域的な影響を観測する．通行止めの解除についても同様に行う．

3.2.2 機能要求

本研究では，日本 IBM 東京基礎研究所と京都大学の共同研究により開発している交通シミュレータを基に，その上で行われる交通シミュレーション内のエージェントの操作，シミュレーション環境の操作といった機能を実現する．そのためには，まず，シミュレーションの状況を可視化する機能が必要である．その上で，エージェントの操作とシミュレーション環境を操作する機能を実装する．また，複数の運転者が参加することを考慮し，特別な装置を不要にするため，ユーザインタフェースは Web を通してブラウザ上から使用することを可能とする．

シミュレーション状況を可視化する機能

シミュレーションの状況をクライアントソフトウェアに表示する．この際，鳥瞰的な視点から見た状況を２Ｄで描画する．アバターが存在する場合は，アバターを中心にその周辺の情報を表示する．ここでアバターとは，ユーザが操作する車両エージェントのことである．表示する情報は，一定時間ごとにシミュレータから情報を取得することで更新する．

この機能によってもたらされる情報から，ユーザは行動を決定し，操作を行う．この情報は，Web を通してクライアントに送信される．ここで，通信における遅延が問題となる．ユーザの情報の受け取りに遅延が生じると，行動を決定するための情報が古い情報になるため，シミュレーションの状況に応じた操作を行えなくなる．例えば，ユーザが「すぐ先の交差点で左折」という操作を行った際，表示する情報の通信に遅延が発生しており，シミュレーション上ではすでに交差点を通過しているため，操作の影響を受けないということが起こる．このような状況では，ユーザの意思が反映されないことになる．そこで，この情報の送信における遅延を小さくする必要がある．ここでは，送信する情報の限定することにより，通信にかかる負荷を軽減し遅延を小さくする．

表示する領域は任意に指定可能である．その際拡大表示では，狭い範囲の詳細な情報が求められるため，領域内の各車両を地図上に１台ずつ表示しする．一方広域表示では，車両一台一台の位置のような細部の詳細な情報は要求されない．例えば京都府全域を表示しそこに全車両を表示した場合，ユーザにとって不要な情報があるばかりでなく，必要な情報を読み取ることも困難である．また，表示する情報は Web を通してかつ定期的に獲得されるため，情報量が膨大になることは望ましくない．そこで広域表示ではユーザが操作するアバターを除き，他の車両は表示せず，各道路ごとの混雑度の情報表示する．混雑している道路は赤，すいている道路は青といったように，その道路の色によって表示する．

情報の更新間隔については最も狭域で詳細を表示する状態で 100ms とした．これはシミュレータの１サイクルに要する時間が約 100ms であるためクライアントがこれ以上に頻繁に更新しても表示する情報が変化しないためである．また更新間隔は表示する領域が広域になるにしたがって長くなる．表示する領域が広域になると縮尺が小さくなるため，単位時間当たりに各車両が移動する画面上の距離が短くなる．そのため，詳細表示時と同一の間隔で表示情報を更新してもユーザには変化が小さすぎるため伝わらないため処理が無駄になる．ま

た表示する情報自体は増加するので処理を軽減するため，更新間隔を長くする．さらに広域を表示する際には車両の位置ではなく混雑度が表示される．混雑度は道路上の車両数と道路の許容量で決まるものであり，短時間で急激に変わることは考えにくい．そのため，この場合も更新間隔を長くする．

車両エージェントを操作する機能

ユーザによるアバターの操作をクライアントを通して受け取り，シミュレーションに反映させる機能である．今回実装した機能は交差点における車両の進行方向の操作である．ユーザは周囲の情報から進路を決定し車両を操作することができる．これにより様々な状況下で各運転者がどのような経路選択を行ったかという情報を得ることができる．また，ユーザが複数の車両を操作することで，故意的に周囲の道路の混雑度を変化させ，その後の状況を観測することで，周囲の道路の変化の影響を確認することが可能となる．

環境を操作する機能

シミュレーション上の環境を変化させる機能として道路を閉鎖する機能および開放する機能を実装する．この機能は主にシミュレーションの状況に変化を与え，その後のシミュレーションの状況から影響を観測するための機能である．シミュレーション実行中に道路を閉鎖することにより，本来その道路を使用する予定であった車両は迂回することになる．そうすることにより迂回路となった道路は通行量が増加する．道路が混雑することにより経路を変更する車両が出てくることが考えられる．このような影響を観測する．また，逆にシミュレーション開始時にある道路を閉鎖しておき，ある時間でその道路を開放することにより，擬似的に新しい道路を設置した状況を再現できる．また，同じ箇所に許容量の異なる道路を作成し，一方を閉鎖し他方を開放する．ここで開放する道路を入れ替えることで，擬似的に道路を拡張した状況が再現できる．これらにより周囲の道路の混雑状況がどのように改善されるかを確認することによって，どこにどのような道路を新設するべきであるかを検討する際の指標となる．

クライアントソフトウェア上で閉鎖，開放する道路を指定し，その道路の閉鎖および開放を行う．道路が閉鎖された時点でその道路上に存在する車両は，道路から退出するまでの走行は許可される．その後進入しようとする車両に対しては閉鎖が反映され，進入は拒絶される．今回使用する交通シミュレータには道路に車線という概念が存在しないため，通行止めは車線単位ではなく道路単位で行う．しかし全ての道路は一方通行であるため，双方向の道路は２本の道

路として存在する．そのため，１本の道路を通行止めにすることで双方向の通行が遮断されるわけではない．

3.2.3　ユーザインタフェース

ユーザが交通シミュレーションの交通状況を把握し，エージェントやシミュレーション環境を操作するためのユーザインタフェースを設計する．前節で述べたそれぞれの機能について，それらがどのようなインタフェースによって実行されるかを説明する．

シミュレーション状況の可視化

交通シミュレーションにおける交通状況を把握するためには，シミュレーションされている道路および車両の位置情報を表示する必要がある．また，統計的な情報を視覚的に表示するため，混雑度を色で表示する必要がある．これらの表示の切り替えは，表示する範囲の大きさにしたがって自動的に切り替えられる．これは，縮尺が小さい場合には，一度に表示される道路も少なく，車両一台一台が表示されることによっても混雑度が把握できると考えられるからである．また，縮尺が大きい場合には，車両一台一台を表示しても全体の把握が困難であり，さらに，大量の車両の位置データをサーバから取得・表示するための通信料と計算量が大きくなる．したがって，シミュレーション状況の可視化に関わるユーザインタフェースとして，交通状況を表示するためのメインウィンドウと，縮尺の変更や表示範囲の移動を行うためのボタンを配置する．

車両エージェントの選択と操作

車両を操作するユーザは，シミュレーション上の車両を選択し，その車両の速度や進路の指示を行う．車両を選択する場合，メインウィンドウ上で表示されている車両をクリックなどで選択するのが最も直感的である．しかしながら，シミュレーションは常に動き続けているため，その上でひとつの車両を捕らえるのは容易でない．そこで，車両を直接選択するのではなく，交差点を選択し，その交差点に最初に到達した車両を操作の対象とする手法をとった．ユーザによる車両の操作は以下の手順で行われる．

1. 操作対象となる車両の指定
   ユーザインタフェース上で交差点を指定することにより行う．指定された交差点に最初に進入した車両が操作の対象となる．
2. 進行方向の指定
   ユーザインタフェース上でアバターの予定進路上の交差点を選択し，その

交差点につながっている道路の中から次に進入する道路を指定する．

3. 車両への反映
   操作対象の車両がその交差点に進入すると指示に従い，指定された道路に進入する．

シミュレーション内にユーザの操作対象となっている車両が存在する場合，どの縮尺においても操作している車両の位置が分かるように，道路状況が混雑度で表示されている場合（縮尺が大きい場合）でも操作対象の車両のみは表示されるようにする．

道路の閉鎖と開放

道路の閉鎖・開放を行うユーザは，メインウィンドウ上に表示されている道路を選択し，その道路の閉鎖や開放を行う．道路を閉鎖した場合，閉鎖されていることが分かるように，その道路には閉鎖マークを表示する．閉鎖マークが表示されている道路を選択した場合は，その道路を開放することができる．道路を開放すると，閉鎖マークは外れる．

3.3 システムの設計

交通シミュレータとして，日本 IBM 東京基礎研究所と京都大学の共同研究により開発している交通シミュレータを用いる．このシミュレータに対して，シミュレーションの実行をリアルタイムに観測し，エージェントや環境を操作することの出来る機能を付加する．

3.3.1 設計方針

シミュレータには前述の交通シミュレータを用いる．シミュレーション実行中の情報を外部に提供するため，ログの出力機能を拡張し，TCP/IP 通信によって外部に送信できるように修正する．また，外部からの操作を反映させるため，各エージェントの速度変更，経路選択の部分を外部で定義できるようにする．これにより，ユーザによる車両の操作が可能になる．一方，現在の交通シミュレーションではシミュレーションの環境を実行中に変更することは困難である．しかし，エージェントの経路選択を外部定義することができれば，閉鎖した道路を常に選択の対象外とすることで，仮想的に道路の閉鎖機能を実現することができる．したがって，交通シミュレータに関しては，ログの通信機能とエージェントの意思決定の外部定義を行えるように修正すればよい．

そして，シミュレータとクライアントソフトウェア間に通信サービスを配置

し，シミュレーションが送信するログや道路を選択するロジック，クライアントからの要求，クライアントに提供する情報を管理する．

3.3.2 アーキテクチャ

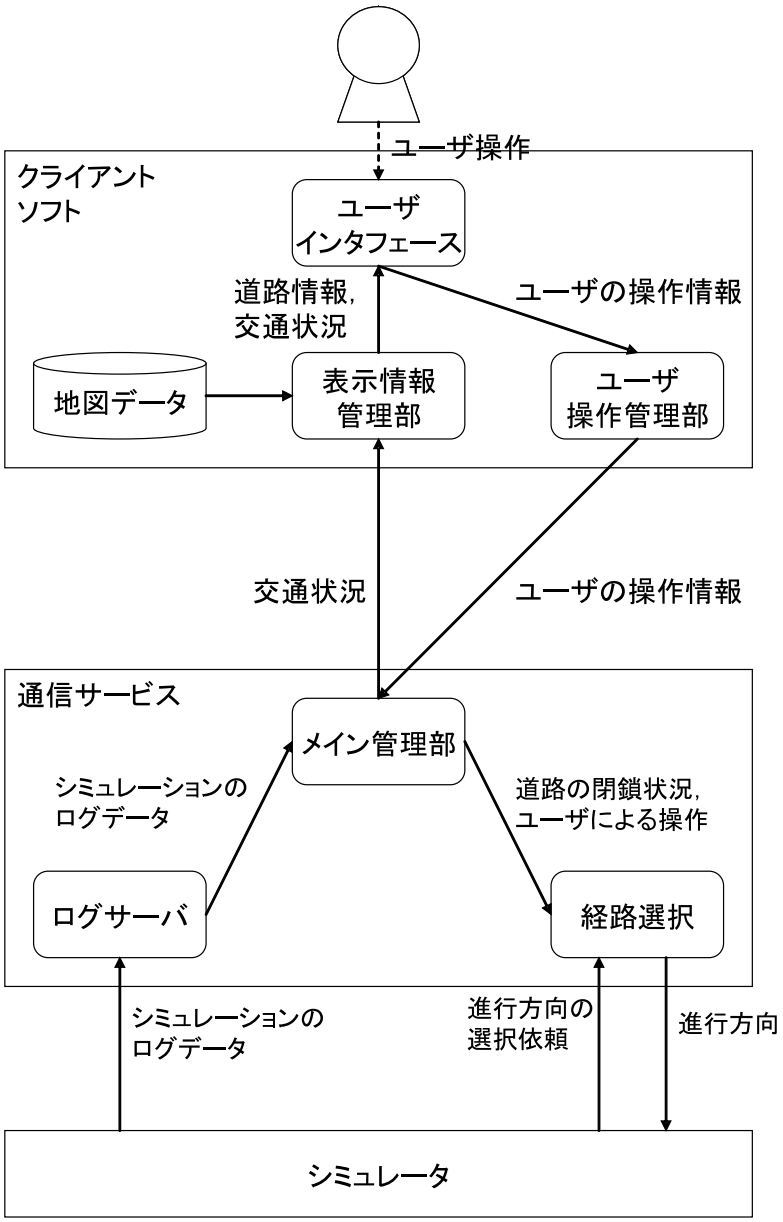

図 2: 可視化ツールとシミュレーションサーバの構成

交通シミュレータの可視化を行うためのアーキテクチャを，図 2 に示す．クライアントとシミュレータの間に，ユーザからの操作情報を管理し，シミュレータに反映させるための通信サービスを配置する．

クライアントソフトは以下のように構成される．

ユーザインタフェース　表示情報管理部から，表示範囲内のシミュレーションの状況を取得，描画する．一方で，ユーザからの操作を受け付ける．ユーザから入力された道路の閉鎖や車両の進行方向といった情報は，ユーザ操作管理部へ渡される．

表示情報管理部　道路情報，車両の位置情報，道路の混雑度情報といった，ユーザに表示するための情報を管理する．道路情報はローカルに保存した道路データから，車両や混雑度の情報は，通信サービスから得る．

ユーザ操作情報管理部　ユーザから入力された情報を管理し，通信サービスに送る．

通信サービスは，以下のように構成される．

メイン管理部　クライアントソフトに対するインタフェースとなる．クライアントからの要求に応じて，ログサーバからシミュレーションのログデータを取得してクライアントに送信したり，クライアントから送られてきた道路の閉鎖状況やユーザの操作を経路選択部に渡したりといった役割を持つ．

ログサーバ　交通シミュレータから定期的に送信されるログデータの管理を行い，メイン管理部からの要求に応じてデータを渡す．

経路選択部　交通シミュレータにおけるエージェントの意思決定を外部定義したもの．交通シミュレータ内の各車両が経路選択を行う際，現在の状況をこの経路選択部に送信し，経路選択部が進入する道路を選択，シミュレータに送る．このとき，もしその車両がユーザに操作されているものであり，かつ，ユーザが進路を指定していれば，それに従う．また，ユーザによって閉鎖されている道路については，選択対象から除外することによって，その道路を通らないようにする．

3.3.3　処理の流れ

クライアントの情報の取得におけるシーケンス図を図 3 に示す．まず，クライアントは通信サービスに対して，表示範囲内の車両と混雑度情報を要求する．メイン管理部がその要求を受信し，ログサーバから車両と混雑度の情報を得る．そして，得られたログデータから，範囲内の車両および道路の混雑度を選別し，クライアントに返す．シミュレータからのログデータは，定期的にログサーバに送られ続けている．

車両の進行方向を行うときのシーケンス図を図 4 に示す．ユーザがユーザイ

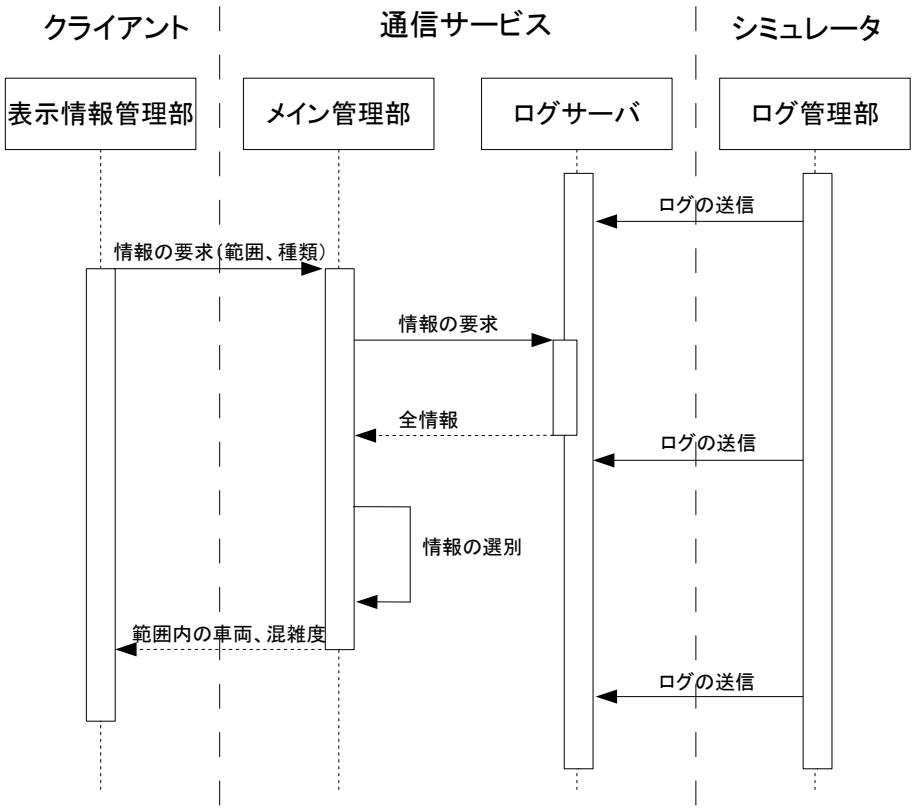

図 3: 情報の取得におけるシーケンス図

ンタフェースを介して指定した進行方向の情報は，ユーザ操作管理部を経由して，通信サービスのメイン管理部に送られる．その情報を受け取ったメイン管理部は，その情報を経路選択部へと伝える．シミュレータのエージェントは，交差点に到達するたびに，どの経路を辿ればよいかを決定するため，経路選択部に現在の状況を送る．経路選択部はその情報を基に，次に進むべき道路を決定し，エージェントへ返す．このとき，ユーザによってその交差点での進行方向が指定されていれば，周囲の状況に関わらずその方向の道路を返す.

図 5 に，道路の閉鎖，開放を行う場合のシーケンス図を示す．車両の進行方向を行う場合と同様に，閉鎖された道路の情報は通信サービスの経路選択部へと送られる．道路の閉鎖の場合，エージェントが経路を決定するために現在の状況を送ってきたとき，閉鎖された道路を選択しないようにする．これにより，シミュレータ内の環境が変更できない状態でも，仮想的に道路の閉鎖を実現することが出来る.

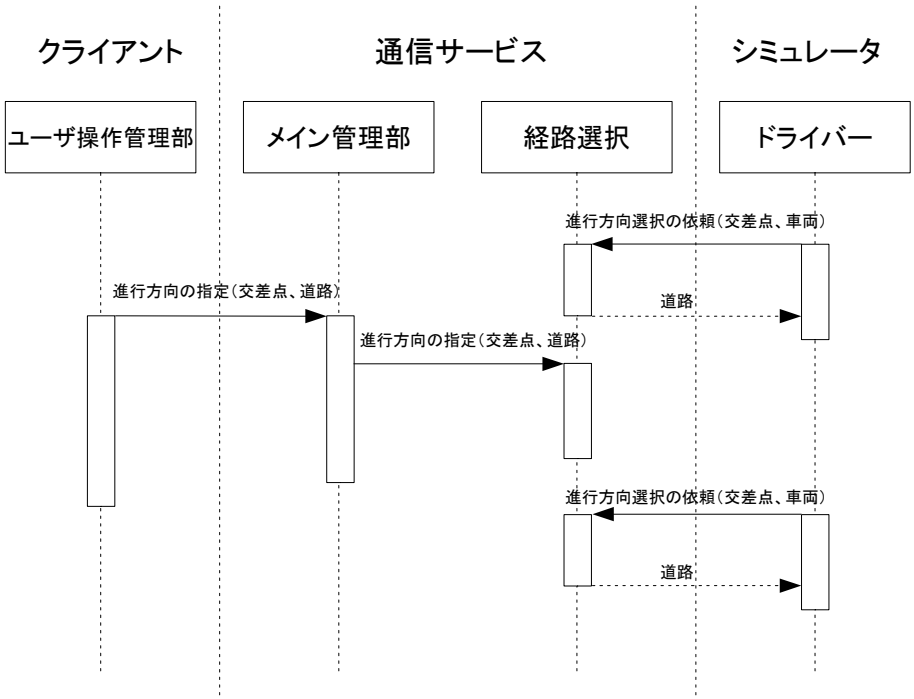

図 4: 車両の操作におけるシーケンス図

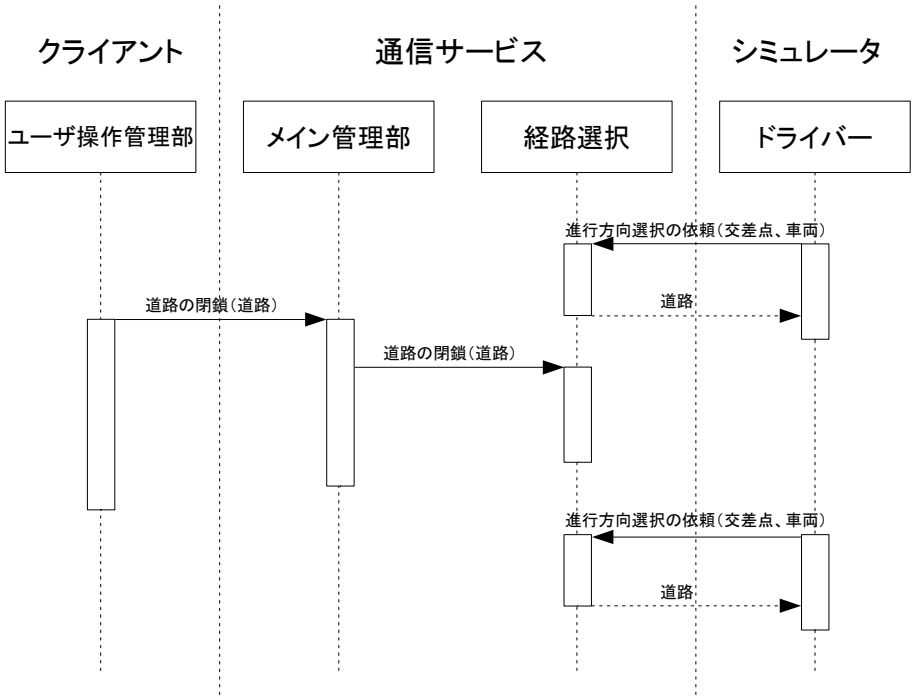

図 5: 道路の閉鎖，開放におけるシーケンス図

第4章　可視化ツールの実装

シミュレータは既存のものを使用した．ただし，ログの出力機能を拡張し外部に送信できるように修正し，外部からの操作を反映させるため，各エージェントの速度変更，経路選択の部分を外部で定義できるようにする．また，クライアントソフトウェアは所属する研究室で実装が行われ，本研究ではシミュレータとクライアントソフトウェアを結合する，通信サービスの実装を行った．通信サービスの各部の実装について以下に述べる．

4.1　メイン管理部

通信サービスの本体である．実行されるとログサーバを起動しシミュレータからのログを待ち受ける状態になる．またクライアントからの情報の要求や，シミュレーションの操作の指示などはメソッドの呼びだしという形で実現される．このオブジェクトはそのための RMI インタフェースを持つ．

クライアントの情報の要求に対応し，ログサーバに車両の情報や道路ごとの混雑度の情報を要求する．通信するデータ量を減らすため，ここで得られた情報からクライアントが必要としている情報への選別を行う．ユーザが表示している領域内の情報のみを要求する．領域については長方形であり，東西南北それぞれの端の座標を指定する．この情報はメソッド呼び出しの引数として渡される．また，詳細表示時は各車両の情報を要求し，広域表示は各道路の混雑度を要求する．

ここで，クライアントに送信する情報を選別するのは，通信するデータ量を減らすことで，データの送信にかかる時間を短縮するためである．そのため，選別に要する時間が，データ量を減らすことで短縮された送信にかかる時間を上回ってはいけない．そこで，まず情報の選別に要する時間を計測した．今回使用したシミュレータではひとつの Java 仮想計算機上で 10 万から 20 万のエージェントが保持可能であり，複数の環境をつなげて動作することが可能である．そのため，選別の対象とするデータは 10 万台，50 万台，100 万台の車両のデータとした．この場合の情報の選別は 2 次元座標の範囲検索に相当する．そのため，効率的に選別を行うためには情報のソートを行う方法や，R 木を作成する方法などが考えられる．しかしクライアントからの要求の間隔は，通信サービスが保持するシミュレーションの情報が更新される間隔より長い．すなわち 1 度の

表 1: 情報の選別に要する時間

車両数 (台)	10 万	50 万	100 万
所要時間 (ms)	2.34	14.12	27.97

表 2: クライアントと通信サービスの間の通信に要する時間

該当件数 (台)	1000	3000	5000	1 万	5 万	10 万
所要時間 (ms)	6.87	14.03	19.73	41.78	224.32	407.83

検索を効率するために，１回以上ソートを行うことや木を作成することになり，かえって処理が増大することになる．そのため，選別方法は全データを走査する方法を用いた．計測結果を以下の表 1 に示す．結果はそれぞれ 30 回計測した値の平均値である．時間の単位はミリ秒である．

所要時間は車両数に比例している．また，クライアントの情報の更新間隔は最短で 100ms であるため 200 万台程度までででは十分な速さであると考えられる．現状シミュレータの性能から車両数が増大した場合処理速度が落ちることから 200 万台の検索で十分な処理速度が得られれば問題はない．また，さらに車両数が多くなる際には，シミュレーション環境が分割されるため，車両データも領域ごとに分割される．そこで，分割された集合ごとに同時に検索を行うことで，検索に要する時間はこれ以上にかかることはない．次に，RMI 通信を用いて，クライアントがデータの要求してからデータの取得するまでにかかる時間を，情報の選別を行い，該当した情報のみ送信するの場合について計測した．送信される情報の量が，通信に要する時間に大きな影響を与えると考えられるため，該当件数を変化させて計測した．選別方法は，上記と同様に，全データを走査する方法を用い，対象とするデータは，10 万台の車両のデータとした．これは送信する情報が，クライアントに表示する車両であるためである．10 万台以上の車両を表示した場合は，明らかにユーザがその情報を処理しきれない．そのため，それ以上の情報を表示することはない．そこで送信するデータの最大値を 10 万台とした．計測結果は表 2 に示す．結果はそれぞれ 30 回計測した値の平均値である．所要時間とは，クライアントが情報を要求してから，情報を受け取るまでに要する時間である．時間の単位はミリ秒である．

クライアントの情報の更新間隔は，表示領域に対応して，最短で 50m 四方で

100ms，そこから 250ms，500ms，1s，･･･と長くなる．また，各車両の位置を表示する領域は，1km 四方程度までである．ここで表示する車両の大きさを 2m × 5m と考えると，画面上に車両を敷き詰めても 50m 四方で 250 台である．また，全面積に対する道路の面積である面積率は，東京都の 23 区でも 25 %を超えるものはない．すなわち，1km 四方でも範囲内に存在し得る車両は，25000 台を越えない計算である．1km 四方を表示時には，更新間隔は 1s 以上となり，5 万台の情報の送信に要する時間が 224.32ms 程度であることから，更新間隔に対して，これらの所要時間は許容範囲内であると考えられる．10 万台程度の車両を用いた場合でも，送信する情報を選別し減らすことにより，所要時間の短縮に大きな効果があることがわかる．

以上により情報の選別の有効性が分かった．よって，送信する情報は表示する領域内の車両，混雑度に限定する．また，選別方法は全データを走査し，該当するデータを選別する方法を用いる．

クライアントのシミュレーションへの操作の指示に対しては，その情報を経路選択オブジェクトに反映させる．指示に関する情報は，クライアントからメソッド呼び出し時に引数として渡される．この情報は経路選択オブジェクトで管理され，操作の実現も経路選択オブジェクトにて行う．

4.2 ログサーバ

クライアントに表示するための情報の受け渡しを行う．通信における遅延を軽減するために，ログサーバは最新の情報を保持し，クライアントはその値を参照する形をとる．情報の更新は，シミュレータからのログの送信によって行われる．このとき，シミュレータからは任意のタイミングでログが送信され，クライアントからも任意のタイミングで最新の情報を要求される．シミュレータから送られてくるログは解析し，クライアントが使用可能な形式に変換する必要がある．また，クライアントの要求に応えるため，ある時間のデータを解析中にも，その時点で完全かつ最新のデータを提供できる必要がある．そのため，ログサーバはスレッドを用いて，常に送信されてくるログを解析し，同一時間の情報の解析が終了した時点で，最新の情報として上書きする．クライアントは，メイン管理部を通じて，このオブジェクトのメソッドを呼び出すことにより，最新の情報を参照する．すなわち時間 t のログを解析中には，クライアントは時間 t-1 の情報を参照し，ログサーバが時間 t のログを解析し終えた時点で，

時間 t の情報が参照可能となる．ログサーバが保持する情報は，各車両の位置と方向，および各道路の混雑度の情報である．それぞれの集合をリストとして保持する．

4.3 経路選択部

シミュレータ単体の機能としては，道路の通行止めは実装されておらず，本来進行方向の決定もシミュレータ内で閉じている．そこで進行方向の指定や，道路の閉鎖を実現するため，車両の進行方向の選択のロジックを，外部で記述できるように変更した．

このオブジェクトは，車両の進行方向の選択のロジックを管理する．そのため，進入する道路の選択に必要な情報である，車両に対する進行方向の指定に関する情報，および道路の閉鎖，開放に関する情報をこのオブジェクトで管理する．車両に対する進行方向の指定に関する情報は，指示を行う交差点，進入する道路の組の集合で表現する．道路の閉鎖，開放に関する情報については，閉鎖された道路の集合として表現する．それぞれリストの形で保持する．これらの情報は，ユーザの操作によってのみ変更される．ユーザの操作は，メイン管理部を通して，このオブジェクトに伝えられる．また，閉鎖された道路の情報は，クライアントの画面描画に必要であるため，クライアントからメイン管理部を通して要求され，このオブジェクトが持つ情報が参照される．

シミュレータ内のドライバーは，交差点到達時に進行方向の決定を依頼する．この依頼はシミュレータ側のタイミングで行われ，経路選択オブジェクトは依頼を待ち受ける形になる．そのため，この依頼は，クライアントが RMI 通信を用いて，このオブジェクトのメソッドを呼び出すことで実現する．このメソッド呼びだしを可能にするため，経路選択オブジェクトは RMI インタフェースを持つ．進行方向の決定を依頼された際，車両が存在する交差点に進行方向の指示や周囲に閉鎖された道路がなければ，進行方向の決定を行わない．交差点に進行方向の指示が存在すれば，その道路を選択する．また，交差点の周囲に閉鎖された道路が存在すれば，それ以外の道路を選択することにより，道路の通行止めを実現する．ドライバーは，経路選択オブジェクトから道路を指定されれば，その指示に従って行動し，指示がなされなければ，シミュレータのデフォルトの経路選択ロジックを用いて進行方向を決定する．

4.4　実行例

本ツールの実行例を以下に示す．サーバ側で通信サービスを起動した後，シミュレータを起動しシミュレーションを実行する．その後，クライアントを起動することで，シミュレーション状況が画面上に表示される．対象とした地域は大阪府である．その様子を図 6 に示す．

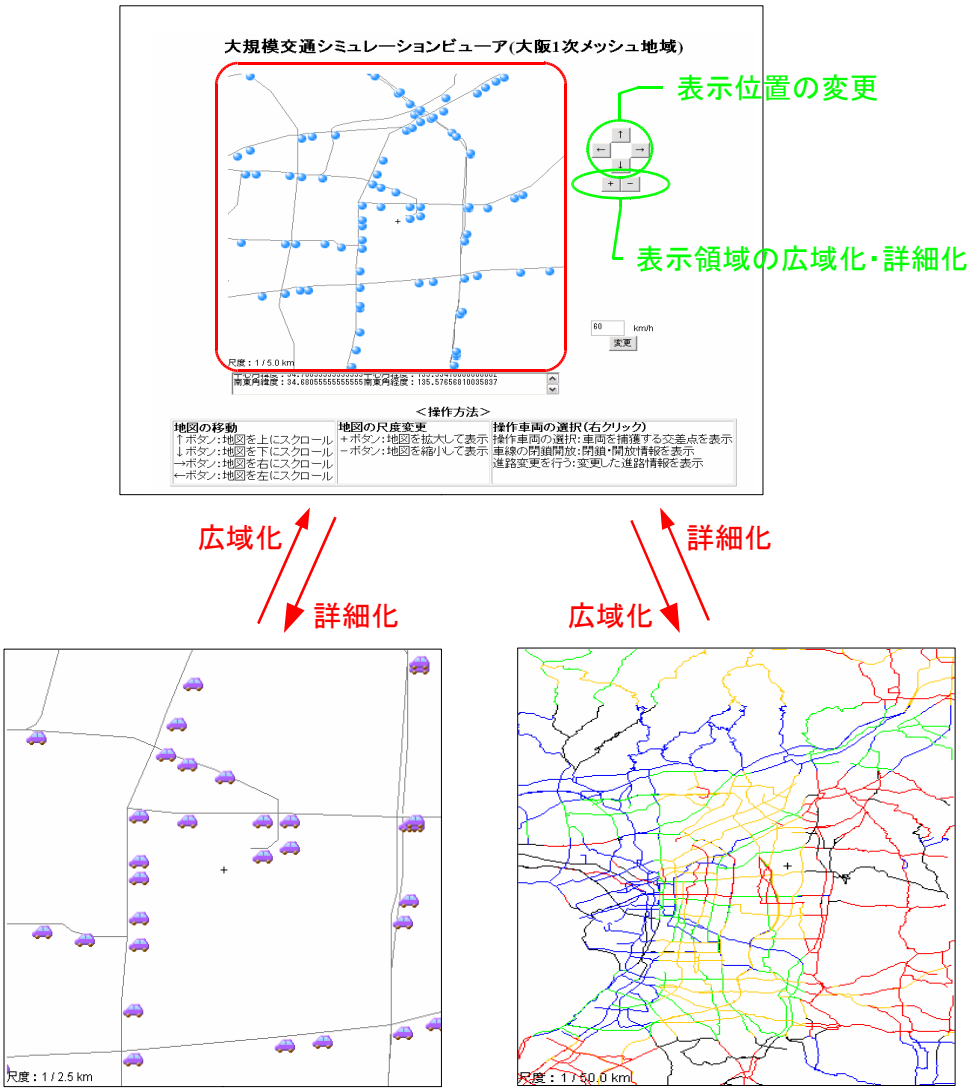

図 6: 表示

図 6 の左下部が，車両一台一台が表示される詳細表示時の画面であり，右下部が，各道路の混雑度が色で表示される広域表示時画面である．上部は，その

中間であり，車両が点で表示される画面である．ユーザインタフェースの右上部の矢印が，表示領域を移動するボタンである．デフォルトではアバター周辺の状況が表示されるが，表示する位置は任意に変更可能である．その下に位置する＋，－のボタンは表示領域の詳細化，広域化を行うボタンである．

次に道路を閉鎖・開放する様子を，図 7 に示す．道路の閉鎖は，閉鎖する道路

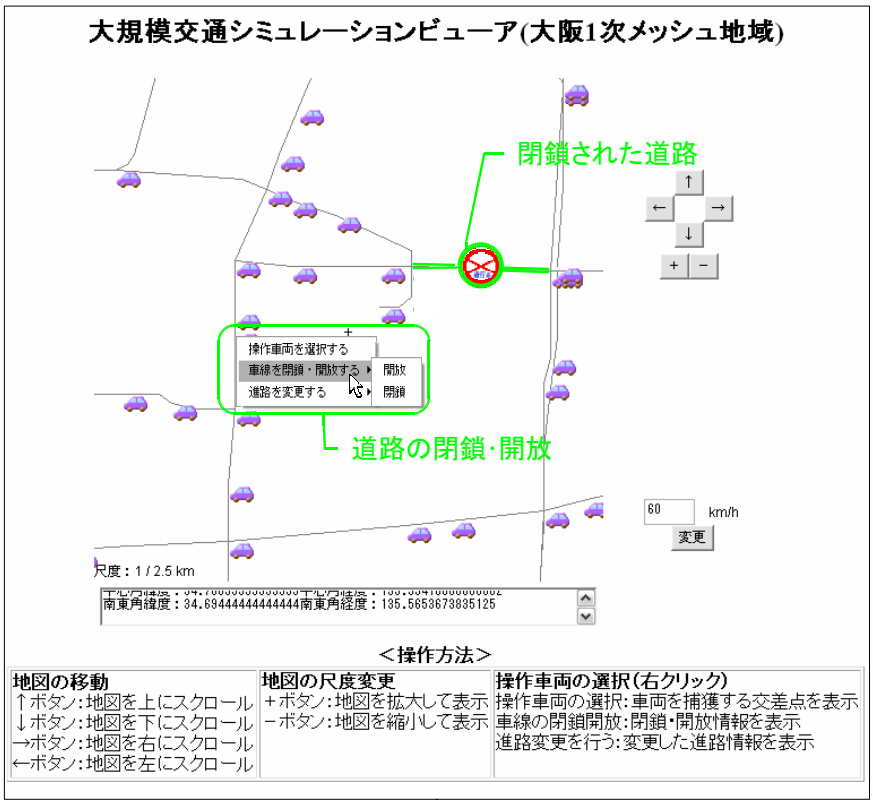

図 7: 道路の閉鎖

にカーソルを合わせ，右クリックで表示されるポップアップメニューから，「車線を閉鎖・開放する」を選択し，その後「閉鎖」を選択することで実行する．開放する場合も同様にし，最後に「開放」を選択する．閉鎖された道路は，道路上に通行止めの道路標識が表示される．閉鎖された道路には車両は進入できない．

次に車両を操作する様子を，図 8 に示す．ユーザの操作の対象となる車両は赤く表示される．また，他の車両を操作対象にする場合は，交差点にカーソルを合わせ，右クリックで表示されるポップアップメニューから，「操作車両を選

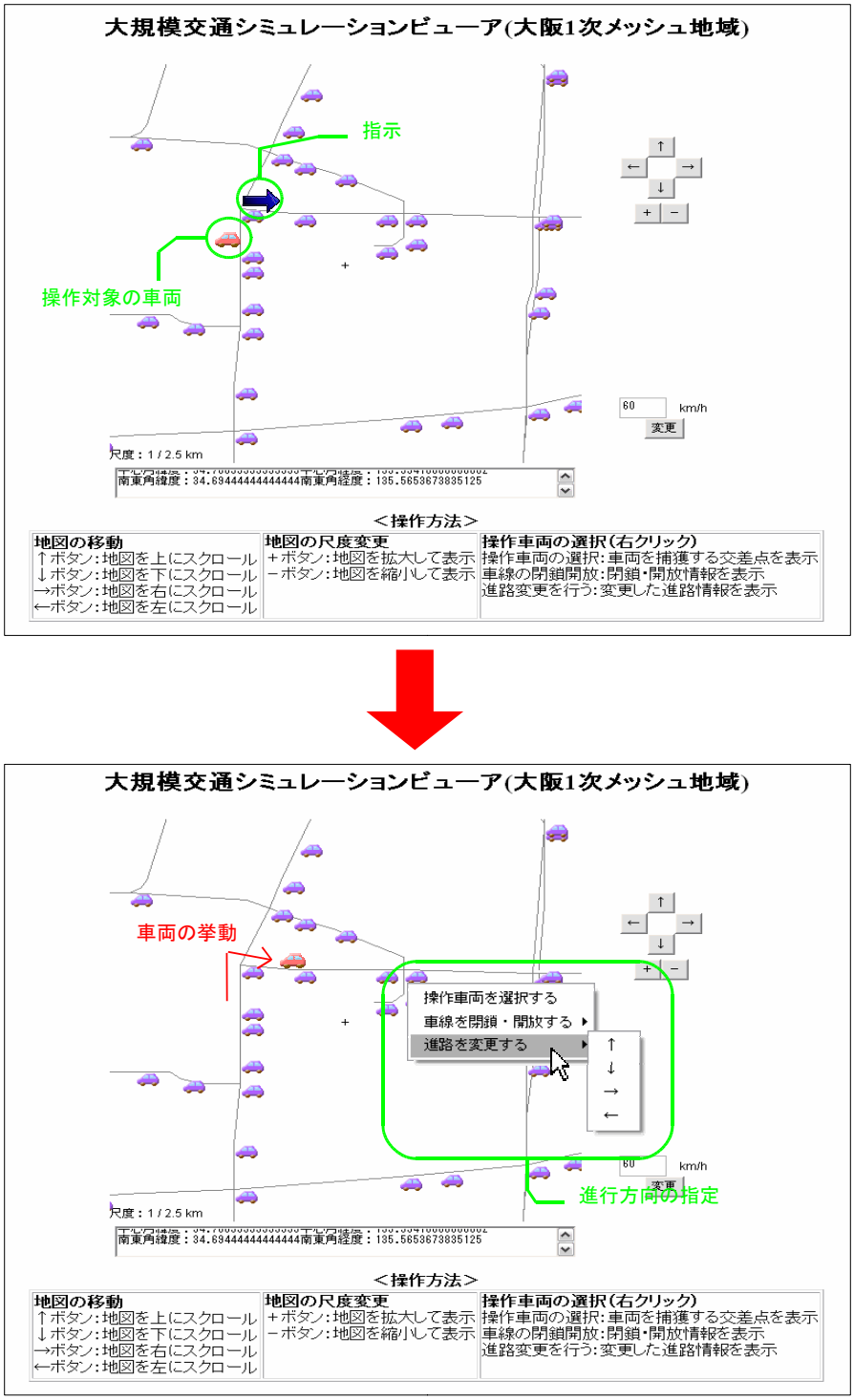
大規模交通シミュレーションビューア(大阪1次メッシュ地域)
指示
操作対象の車両
60
km/h
変更
尺度：1 / 2.5 km
南東角緯度：34.694444444444南東角経度：135.56536738835125
＜操作方法＞
地図の移動
↑ボタン：地図を上にスクロール
↓ボタン：地図を下にスクロール
→ボタン：地図を右にスクロール
←ボタン：地図を左にスクロール
地図の尺度変更
＋ボタン：地図を拡大して表示
－ボタン：地図を縮小して表示
操作車両の選択(右クリック)
操作車両の選択：車両を捕獲する交差点を表示
車線の閉鎖開放：閉鎖・開放情報を表示
進路変更を行う：変更した進路情報を表示
大規模交通シミュレーションビューア(大阪1次メッシュ地域)
車両の挙動
操作車両を選択する
車線を閉鎖・開放する
進路を変更する
60
km/h
変更
進行方向の指定
尺度：1 / 2.5 km
南東角緯度：34.694444444444南東角経度：135.56536738835125
＜操作方法＞
地図の移動
↑ボタン：地図を上にスクロール
↓ボタン：地図を下にスクロール
→ボタン：地図を右にスクロール
←ボタン：地図を左にスクロール
地図の尺度変更
＋ボタン：地図を拡大して表示
－ボタン：地図を縮小して表示
操作車両の選択(右クリック)
操作車両の選択：車両を捕獲する交差点を表示
車線の閉鎖開放：閉鎖・開放情報を表示
進路変更を行う：変更した進路情報を表示

図 8: 車両の操作

択する」をクリックする．その後，指定した交差点を通過した車両が，操作対象の車両となる．進行方向の指定は，操作対象の車両の予定進路上の交差点にカーソルを合わせ，右クリックで表示されるポップアップメニューから,「進路を変更する」を選択し，その後表示される矢印から指定する方向を選択し，クリックする．進行方向を指定すると，指定した交差点に進行方向が，青色の矢印で表示される．その交差点に操作対象の車両が進入すると指示に従い，進行方向を変更する．

第5章　おわりに

本稿では，マルチエージェントシミュレーションに基づく交通シミュレーションに関連して，シミュレートする交通状況の理解・把握を容易化し，かつ分析のためのシミュレーション操作の機能を提供可能な，マルチエージェント交通シミュレーション可視化ツールのを開発し，その機能，実装について述べた．また，クライアントソフトの情報の取得に関わるパフォーマンスを計測した．その結果，通信サービスが情報を選別することにより，大規模シミュレーションの可視化を実用的な実行速度で実現できることがわかった．

今回，シミュレーション状況に操作を行う機能として，交差点における車両の進行方向の指定，道路の閉鎖と開放を行う機能を実装した．異なった状況で発生する交通現象の確認や，交通現象の再現性の確認，各車両が周囲の状況に与える影響の確認を行うためには，状況に様々な変化を与え，その後の状況を観測し，変化による影響を確認することが，求められている．これらの機能を実装したことにより，簡易にシミュレーションに参加し，シミュレーションの状況に様々な変化を与えることが可能となった．また，シミュレーション状況を提示する機能により，その後の状況を容易に観測することも可能となった．しかし，現時点では，与えた変化による交通状況への影響を観測することは困難である．これは使用した交通シミュレータで，現在個々の車両の相互作用による影響を効果的に反映させる機能が備わっていないことにある．しかし，シミュレータは開発中であり，研究室では車両，運転者の振る舞いに関するモデルを作成する研究がなされている．本ツールでシミュレーションの状況を変化させ，状況を観測する機能は実現されているため，シミュレータに車両の相互作用による影響が反映されるようになることで，状況の変化による影響を確認するこ

とができ，異なった状況で発生する交通現象の確認や，交通現象の再現性の確認，各車両が周囲の状況に与える影響の確認といった目的に使用することが可能となる．

実際の車両の操作として，進行方向の決定のほか，車線の変更，速度の変更，停駐車などといった操作が挙げられる．また環境の操作としては道路の閉鎖，開放のほか，車線単位の通行止めや信号の設置，道路の設置，制限速度などの交通規制の変更などが挙げられる．これらの操作は，現時点では，シミュレータの機能として実装されていないので，本ツールにも実装されていない．しかし，本ツールは，シミュレータの機能の拡張を想定して設計されており，シミュレータの開発が進み，機能が拡張されると，車両の進行方向の指定，道路の閉鎖と開放の実装のように対応することで，これらの操作もユーザに提供することが可能である．

本ツールは，上記のようにシミュレータの機能の拡張を見据えた，プロトタイプとして設計，実装した．シミュレータの機能の拡張に伴い，ユーザに提供する操作は増加する．それにより，シミュレーションに多様な変化を与えることが可能となる．また，ユーザに豊富な操作を提供することが可能になれば，本ツールは，複数の参加者を運転者として参加させる参加型シミュレーションを行う環境として利用することが可能であり，運転者のモデル化のための情報収集やモデルの妥当性の確認に役立つものである．

謝辞

本研究を行うにあたり，熱心なご指導を頂きました石田亨教授に深謝致します．また，日頃より多くの助言や議論をしてくださいました，松原繁夫准教授，服部宏充助教，山根昇平氏をはじめとする石田研究室の皆様に感謝致します．本研究は，日本学術振興会科学研究費基盤研究 (A) (18200009, 平成 18 年度～20 年度)，総務省戦略的情報通信研究開発推進制度地域情報通信技術振興型研究開発（平成 17 年度～平成 19 年度）の助成を受けて行われた．

参考文献

[1] Espie, S., Saad, F. and Schnetler, B.: Microscopic traffic simulation and driver behavior modeling: the ARCHISIM, *Proceedings of the Strategic Highway Re-*

search Program and Traffic Safety on Two Continents (1994).

[2] Balmer, M., Cetin, N., Nagel, K. and Raney, B.: Towards Truly Agent-Based Traffic and Mobility Simulations, *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, Washington, DC, USA, IEEE Computer Society, pp. 60–67 (2004).

[3] Paruchuri, P., Pullalarevu, A. R. and Karlapalem, K.: Multi agent simulation of unorganized traffic, *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, New York, NY, USA, ACM, pp. 176–183 (2002).

[4] Doniec, A., Mandiau, R., Espie, S. and Piechowiak, S.: Dealing with multi-agent coordination by anticipation: application to the traffic simulation at junctions, *EMSS'2005 European Modeling Simulation Symposium* (2005).

[5] 飯田克弘, 隅本雄一, 巽義知, 安時亨: VR 技術の適用による合流部付加車線延伸効果の検討, 土木計画学研究・論文集, Vol. 22, pp. 925–932 (2005).

[6] 大口敬, 飯田克弘: 高速道路サグにおける追従挙動特性解析におけるドライビング・シミュレータ技術の適用性, 交通工学, Vol. 38, pp. 41–50 (2003).

[7] Weiß, G.(ed.): *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, MIT Press, Cambridge, MA, USA (1999).

[8] Nakanishi, H., Yoshida, C., Nishimura, T. and Ishida, T.: FreeWalk: A 3D Virtual Space for Casual Meetings, *IEEE MultiMedia*, Vol. 6, No. 2, pp. 20–28 (1999).

付録：通信サービスのソースコード

A.1　メイン管理部

A.1.1　RMIService.java

```
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.List;

public interface RMIService extends Remote {
List<VehicleInfo> getVehicles(double north, double west, double south, double east)
 throws RemoteException;
long[] getClosedRoads(double north, double west, double south, double east)
 throws RemoteException;
List<Congestion> getCongestionInfo(double north, double west, double south, double east)
 throws RemoteException;
void closeRoad(long roadID) throws RemoteException;
void openRoad(long roadID) throws RemoteException;
void selectVehicle(long crossPointID) throws RemoteException;
void setNextRoad(long vehicleID, long crossPointID, long roadID) throws RemoteException;
AvatarInfo getAvatar() throws RemoteException;
}
```

A.1.2　RMIServiceObj.java

```
import java.rmi.Naming;
import java.rmi.RMISecurityManager;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.ArrayList;
import java.util.List;
import java.lang.Thread;

public class RMIServiceObj extends UnicastRemoteObject implements RMIService{

static LogServer logServer;
static SelectRoad selectRoad;
static SelectRoadImpl selectRoadImpl;

public static void main(String args[]) {
    if (System.getSecurityManager() == null) {
      // セキュリティマネージャーを設定します
      System.setSecurityManager(new RMISecurityManager());
    }
    try {
      // サーバー側のリモートオブジェクトを生成します
      RMIServiceObj obj = new RMIServiceObj();
      selectRoadImpl  = new SelectRoadImpl();
      selectRoad = selectRoadImpl;
      // リモートオブジェクトに新しい名前を関連付けます
      Naming.rebind("MyObject", obj);
      Naming.rebind("SelectRoad", selectRoadImpl);
    } catch (Exception e) {
```

```
      e.printStackTrace();
    }
    logServer = new LogServerImpl();
Thread thread = new Thread((Runnable) logServer);
thread.start();
}

protected RMIServiceObj() throws RemoteException {
super();
// TODO 自動生成されたコンストラクター・スタブ
}

/**
 *
 */
private static final long serialVersionUID = 1L;

public List<VehicleInfo> getVehicles(double north, double west, double south, double east)
 throws RemoteException {
ArrayList<VehicleInfo> vehicles = (ArrayList<VehicleInfo>) logServer.getVehicles();
ArrayList<VehicleInfo> vehiclesTemp = new ArrayList<VehicleInfo>();
for(int i=0;i<vehicles.size();i++){
VehicleInfo vehicle = vehicles.get(i);
if(vehicle.getLat()<north && vehicle.getLat()>south
&& vehicle.getLng()>west && vehicle.getLng()<east)
vehiclesTemp.add(vehicle);
}
return vehiclesTemp;
}

public long[] getClosedRoads(double north, double west, double south, double east)
 throws RemoteException {
long[] closedRoads= selectRoadImpl.getClosedRoads();
return closedRoads;
}

public List<Congestion> getCongestionInfo(double north, double west, double south, double east)
 throws RemoteException {
ArrayList<Congestion> congestionInfo = (ArrayList<Congestion>) logServer.getCongestionInfo();
ArrayList<Congestion> congestionInfoTemp = new ArrayList<Congestion>();
for(int i=0;i<congestionInfo.size();i++){
Congestion congestion = congestionInfo.get(i);
if((congestion.getSLat()<north && congestion.getSLat()>south
&& congestion.getSLng()>west && congestion.getSLng()<east)
||(congestion.getELat()<north && congestion.getELat()>south
&& congestion.getELng()>west && congestion.getELng()<east))
congestionInfoTemp.add(congestion);
}
return congestionInfoTemp;
}

public void closeRoad(long roadID) throws RemoteException {
selectRoadImpl.addClosedRoad(roadID);
```

```
}

public void openRoad(long roadID) throws RemoteException {
selectRoadImpl.removeClosedRoad(roadID);
}

public void selectVehicle(long crossPointID) throws RemoteException {
}

public void setNextRoad(long vehicleID, long crossPointID, long roadID) throws RemoteException {
selectRoadImpl.setNextRoad(vehicleID, crossPointID, roadID);
}

public AvatarInfo getAvatar() throws RemoteException {
return logServer.getAvatar();
}

}
```

A.2 ログサーバ

A.2.1 LogServer.java

```
import java.util.List;

public interface LogServer {
public List<VehicleInfo> getVehicles();
public List<Congestion> getCongestionInfo();
public AvatarInfo getAvatar();
}
```

A.2.2 LogServerImpl.java

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.List;

import org.omg.CORBA.portable.InputStream;

public class LogServerImpl implements LogServer{
ArrayList<VehicleInfo> vehicles, vehiclesTemp;
ArrayList<Congestion> congestionInfo, congestionInfoTemp;

LogServerImpl(){
vehicles = new ArrayList<VehicleInfo>();
```

```
vehiclesTemp = new ArrayList<VehicleInfo>();
congestionInfo = new ArrayList<Congestion>();
congestionInfoTemp = new ArrayList<Congestion>();
}

private void set(){
vehicles = (ArrayList<VehicleInfo>)vehiclesTemp.clone();
congestionInfo = (ArrayList<Congestion>)congestionInfoTemp.clone();
}

private void setVehicle(double lat, double lng, double direction){
VehicleInfo vehicle = new VehicleInfo(lat, lng, direction);
vehiclesTemp.add(vehicle);
}

private void setCongestion(long roadID, int congestion,
 double sLon, double sLat, double eLon, double eLat){
Congestion c = new Congestion(roadID, congestion, sLon, sLat, eLon, eLat);
congestionInfoTemp.add(c);
}

public List<VehicleInfo> getVehicles() {
return vehicles;
}

public List<Congestion> getCongestionInfo() {
return congestionInfo;
}

public AvatarInfo getAvatar() {
return null;
}

static class SockListener implements Runnable {
Socket sock;

SockListener(Socket s) {
sock = s;
}

public void run() {
try {
InputStream in = (InputStream) sock.getInputStream();
ObjectInputStream oin = new ObjectInputStream(in);
while(true) {
StringBuffer sb = new StringBuffer();
String serverName = oin.readUTF();
int id = oin.readInt();
int len = oin.readInt();
for(int i=0;i<len;i++) {
int t = oin.readInt();
ColumnType type = ColumnType.getFromInt(t);
if (type == ColumnType.BOOLEAN) {
```

```
boolean v = oin.readBoolean();
sb.append(v);sb.append(" ");
} else if (type == ColumnType.BYTE) {
byte v = oin.readByte();
sb.append(v);sb.append(" ");
} else if (type == ColumnType.DOUBLE) {
double v = oin.readDouble();
sb.append(v);sb.append(" ");
} else if (type == ColumnType.FLOAT) {
float v = oin.readFloat();
sb.append(v);sb.append(" ");
} else if (type == ColumnType.INT) {
int v = oin.readInt();
sb.append(v);sb.append(" ");
} else if (type == ColumnType.LONG) {
long v = oin.readLong();
sb.append(v);sb.append(" ");
} else if (type == ColumnType.SHORT) {
short v = oin.readShort();
sb.append(v);sb.append(" ");
} else if (type == ColumnType.STRING) {
String v = oin.readUTF();
sb.append(v);sb.append(" ");
} else if (type == ColumnType.OBJECT) {
Object v = oin.readObject();
sb.append(v);sb.append(" ");
}
}
System.out.println("[" + serverName + ":" + id + "] " + sb);
}
} catch(Exception e) {
e.printStackTrace();
}
}

}


/**
 * @param args
 */
public static void main(String[] args) {
try {
ServerSocket sock = new ServerSocket(Integer.parseInt(args[0]));
while(true) {
Socket s = sock.accept();
Thread th = new Thread(new SockListener(s));
th.start();
}
} catch(Exception e) {
e.printStackTrace();
}
}
```

```
}
```

A.3 経路選択部

A.3.1 SelectRoad.java

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface SelectRoad extends Remote {
public long selectNextRoad(long vehicleID, CrossPointInfo crossPoint) throws RemoteException;
}
```

A.3.2 SelectRoadImpl.java

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.ArrayList;
import java.util.Random;

public class SelectRoadImpl extends UnicastRemoteObject implements SelectRoad {
SelectRoadImpl() throws RemoteException {
super();
}

/**
 *
 */
private static final long serialVersionUID = 1L;
private ArrayList<Long> closedRoads = new ArrayList<Long>();
private ArrayList<long[]> nextRoads = new ArrayList<long[]>();

public void addClosedRoad(long roadID) {
closedRoads.add(roadID);
}

public void removeClosedRoad(long roadID) {
closedRoads.remove(roadID);
}

public void setNextRoad(long vehicleID, long crossPointID, long roadID) {
long[] nextRoadSet = new long[3];
nextRoadSet[0] = vehicleID;
nextRoadSet[1] = crossPointID;
nextRoadSet[2] = roadID;
nextRoads.add(nextRoadSet);
}

public void resetNextRoad(long vehicleID, long crossPointID) {
for(int i=0;i<nextRoads.size();i++){
if(nextRoads.get(i)[0]==vehicleID && nextRoads.get(i)[1]==crossPointID){
```

```
nextRoads.remove(i);
break;
}
}
}

public long selectNextRoad(long vehicleID, CrossPointInfo crossPoint) throws RemoteException{
for(int i=0;i<nextRoads.size();i++){
long[] nextRoadSet = nextRoads.get(i);
if(nextRoadSet[0]==vehicleID && nextRoadSet[1]==crossPoint.getID()){
nextRoads.remove(i);
return nextRoadSet[2];
}
}
long[] road = crossPoint.getOutRoadIDs();
for(int i=0;i<closedRoads.size();i++){
long roadID = closedRoads.get(i);
for(int j=0;j<road.length;j++){
if(road[j]==roadID){
Random r = new Random();
return road[r.nextInt()%road.length];
}
}
}
return -1;
}

public long[] getClosedRoads() {
long[] closedRoad = new long[closedRoads.size()];
for(int i=0;i<closedRoads.size();i++){
closedRoad[i] = closedRoads.get(i);
}
return closedRoad;
}

}
```

A.4 その他

A.4.1 VehicleInfo.java

```
import java.io.Serializable;

public class VehicleInfo implements Serializable{
/**
 *
 */
private static final long serialVersionUID = 1L;
private double lat, lng, direction;

public VehicleInfo(double lat, double lng, double direction){
this.lat = lat;
```

```
this.lng = lng;
this.direction = direction;
}

public double getLat(){
return lat;
}

public double getLng(){
return lng;
}

public double getDirection(){
return direction;
}
}
```

A.4.2 AvatarInfo.java

```
import java.io.Serializable;
import java.util.List;

public class AvatarInfo implements Serializable{
/**
 *
 */
private static final long serialVersionUID = 1L;
private double lat, lng, direction, speed;
private List trip;

public AvatarInfo(double lat, double lng, double direction, double speed, List trip){
this.lat = lat;
this.lng = lng;
this.direction = direction;
this.speed = speed;
this.trip = trip;
}

public double getLat(){
return lat;
}

public double getLng(){
return lng;
}

public double getDirection(){
return direction;
}

public double getSpeed(){
return speed;
}
```

```
public List getTrip(){
return trip;
}
}
```

A.4.3 Congestion.java

```
import java.io.Serializable;

public class Congestion implements Serializable{
/**
 *
 */
private static final long serialVersionUID = 1L;
private long roadID;
private int congestion;
private double sLng, sLat, eLng, eLat;

public Congestion(long roadID, int congestion, double sLng, double sLat,
 double eLng, double eLat){
this.roadID = roadID;
this.congestion = congestion;
this.sLng = sLng;
this.sLat = sLat;
this.eLng = eLng;
this.eLat = eLat;
}

public long getRoadID(){
return roadID;
}

public int getCongestion(){
return congestion;
}

public double getSLng(){
return sLng;
}

public double getSLat(){
return sLat;
}

public double getELng(){
return eLng;
}

public double getELat(){
return eLat;
}
}
```