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

2004－MBL－31　(8)
2004－ＩＴＳ－19　(8)
2004／11／11

MANETアプリケーション向けのシミュレータ MobiREALの実装に関する検討

小西一樹 † 内山彰 †　廣森聡仁 †　山口弘純 †　安本慶一 ‡ 東野輝夫 †　谷口健一 †

我々の研究グループでは，移動ノードを含むネットワークシステムやプロトコルのより現実的かつ詳細な評価を可能とするために，人や車に相当する各移動ノードが周辺物体の存在状況や携帯端末などから得られる情報を元にその行動を決定する様子を容易に記述可能であり，その記述に基づき移動ノードを含むネットワークシステムやプロトコルのシミュレートが可能なMobiREALネットワークシミュレータの設計及び実装を行っている．本稿ではその研究動機や有効な活用事例を検討するとともに，その実現に関する詳細設計を行う．

A Study on a Realistic Network Simulator for MANET Applications

Kazuki Konishi †, Akira Uchiyama †, Akihito Hiromori †, Hirozumi Yamaguchi †,
Keiichi Yasumoto ‡, Teruo Higashino †, Kenichi Taniguchi †

Our research group has been designing and developing a real network simulator for mobile network systems or protocols, called MobiREAL. MobiREAL simulator enables developers to specify the realistic behavior of mobile nodes; persons and cars that correspond to mobile nodes may change their behavior and location from time to time depending on their surroundings, and MobiREAL provides a framework to simulate both the behavior of mobile nodes and network systems interactively. In this paper, we present our motivation in details and enumerate the required functionalities for MobiREAL. Then we describe the design details.

1　はじめに

近年の計算機システムのユビキタス化や無線技術の普及に伴い，移動通信端末（以下，移動ノード）が無線LANなどを介して通信するモバイルネットワークの研究が盛んに行われてきている．モバイルネットワークシステムやプロトコルの設計や実装の際には計算機によるネットワークシミュレーションが不可欠であるが，従来のネットワークシミュレータでは，移動ノードの移動は完全なランダム移動やランダムな目的地選定に基づく移動などに限定しているものが多い．例えば，ns-2[1]ではrandom mobility, random waypoint mobility [2]に従った動作のトレースを出力する独立したプログラムを提供している程度であり，GloMoSim[3]でもそれらに加えtrace mobility, random drunken mobility, reference point group mobilityなどのサポートにとどまっている．

ネットワークエンジニアにとっては，より現実世界に即したシミュレーションシナリオ（現実的な移動ノードの行動）のもとでネットワークシステムの評価を行うことが望ましい．しかし，歩行者や車などに相当する移動ノードの行動を従来のシミュレータで使われているモビリティモデルで表現することは容易でない．例えば，通過地点座標と通過時刻のシーケンスを指定できるようなモビリティモデルを用いれば，各移動ノードの軌跡を指定することは原理的に可能であるが，移動ノードが周囲環境から得られる情報に応じて行動を動的に変化させる様子などが表現できない．また，あらかじめ通過地点を静的に指定することで，非現実的な行動（例えば多数の歩行者で混雑して前方に移動できないような場所に無理に突入するなどの行動）を再現してしまう可能性もある．

そこで本研究では，各移動ノードが自身の周辺環境に応じてその行動を変化させる様子を記述可能であり，その記述に基づいた移動ノードの行動決定プロセスのシミュレートと，各移動ノードの現在座標を反映したネットワークシステムのシミュレートが可能なネットワークシミュレータ MobiREALの設計を行う．MobiREALシミュレータは，行動シミュレータ部，ネットワークシミュレータ部，アニメータの3つの主要コンポーネントより構成される（図1）．行動シミュレータは端末を保持する移動ノードの行動そのものをシミュレートし，アプリケーション実装を備えるネットワークシミュレータに対し，アプリケーションユーザの現在位置をシミュレーションの経過に従い通知する．また，アプリケーションユーザの入力をネットワークシミュレータに通知し，ネットワークアプリケーションからのデータ出力を受け取る．

‡大阪大学 大学院情報科学研究科 · Graduate School of Information Science and Technology, Osaka University
‡奈良先端科学技術大学院大学 情報科学研究科 · Graduate School of Information Science, Nara Institute of Science and Technology

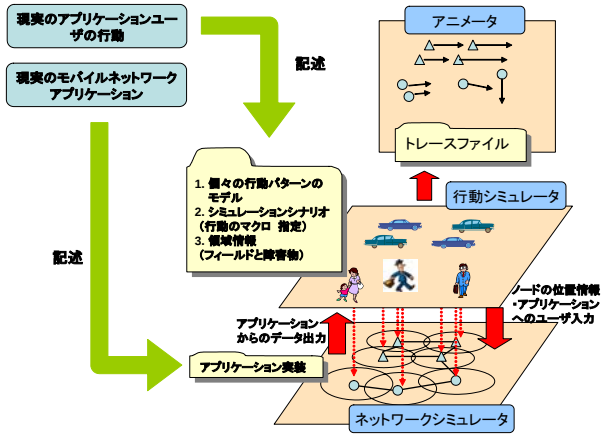

図 1: MobiREAL シミュレータ全体図

2 研究の動機

本章では，既存シミュレータのモビリティモデルにおいて現実的な評価が難しい状況，及び，MobiREAL シミュレータを実現することでネットワークシミュレーションの現実性が向上する状況を具体的なアプリケーション例をもとに検討し，MobiREAL の必要性及び有用性について論じる．

我々は，MobiREAL が必要あるいは有用とされる状況は評価対象とするネットワーク階層ごとに異なると考えている．また，複数の端末の移動パターン（モビリティモデル）に影響を受ける層はネットワーク層以上であると考えてよい．以下ではそれぞれの層における状況例を挙げ，MobiREAL の必要性と有効性を議論する．

ネットワーク層　現実問題として，街頭では地理的にはごく近くにいる者同士で簡単に連絡を取り合いたい場合（混雑する駅前での待ち合わせなど）がしばしばある．そのような場合，広域インフラに頼ることなく，近隣の携帯端末やショートレンジの安価な基地局を介したホップバイホップ通信で 2 端末間のトランシーバ的なリアルタイム通信を行えることが理想であるが，その際には AODV や DSR などの MANET ルーティングプロトコルが必要となる．しかし，このようなアプリケーションを想定してそれらルーティングプロトコルの適性を評価する場合，パケット到着率，経路切断率，経路回復率や回復遅延，経路維持のためのメッセージ総数などのメトリックはモビリティモデルの現実性に大きく依存する．例えば，歩行流にある程度沿って経路が確立された場合，経路を維持する中継端末はおおよそ同方向に移動するため，経路切断率は下がる傾向にあると考えられるが，逆に歩行流に直行する経路であれば経路切断率は大きく上昇する．このような様々な環境においてもルーティングプロトコルがアプリケーションに対し十分な性能を提供できるかなどが評価できる．

また，MobiREAL では個々の移動ノードの行動そのものを詳細に記述した上でシミュレート可能であるため，例えば歩行者の街路上での位置分布なども，個々の歩行者の位置を手動で指定する必要なく，歩行流の特性（歩行者の出現パターン）をシミュレーションシナリオとして指定し，定常状態になるまで移動ノードのシミュレーションを進めるだけで再現可能である．したがって，単純なモビリティモデルでは現れることのなかった移動ノード分布を自然に実現でき，結果としてノード密度分布に偏りのある実環境などが容易に再現できる．このため，例えばオンデマンド型 MANET ルーティングプロトコルにおける経路発見率など，個々の手続きは短時間で終了するためモビリティには直接影響を受けないが，ノード密度などノードの位置分布に影響を受けるメトリックの評価における現実性を向上させることもできる．

また，無線を含む広域通信では，Mobile IP や携帯電話網のセル間ハンドオフなども現実のモビリティモデルでの性能評価が必要であると考えられる．さらに，既存のプロトコルの性能評価のみならず，移動ノードの密集分布，電波の到達状況などに依存したトポロジデザイン（ショートレンジ基地局の配置場所の検討など）に利用できるとも考えられる．これらに対する適用可能性および有用性も検討したい．

トランスポート層　トランスポート層プロトコルは，下位層の経路特性によりその性能を大きく左右される．例えば，ハンドオフに起因するパケット損失などは TCP の輻輳制御に影響を与えると考えられるが，前述のようにそれらはモビリティモデルに大きく影響を受ける．したがって，ネットワーク層のみならず，現実環境での具体的なアプリケーションを想定した場合のトランスポート層プロトコルの適性評価をする場合にも MobiREAL は有用であると考えられる．

アプリケーション層及びアプリケーションユーザ　ここでのアプリケーションとは，ネットワーク層及びトランスポート層で実現されない（あるいはできない）固有の機能を指し，位置情報を用いたコンテキスト依存アプリケーションなどが相当する（位置情報ルーティングはコンテキストアウェアルーティングとしてネットワーク層の機能としてデザインされる場合もある）．そのようなコンテキスト依存アプリケーションは，コンテキストに応じて行動を変える移動ノードの行動を再現することが，評価の現実性に直結する．例えば，いくつかのショートレンジ基地局がスポット情報（例えばイベント開催情報やタイムセールなどの街頭広告）を配布しているとする．各移動ノードは自身の位置情報

履歴を考慮し，例えば移動方向先にあるスポットの情報はその移動ノードにとって有用であるなどの基準で有用な情報のみをフィルタリングし，近隣端末に一定時間間隔（例えば数分）でブロードキャストするような情報管理アプリケーションが各端末に実装されているとする．この際，情報配布を行う基地局の選択とそれに対する移動ノードへの情報の格納率の差などを評価できるなど，コンテキスト依存アプリケーションの現実的な評価が可能である．

さらに，情報の活用効率（実際にその情報を見てスポットに向かった歩行者の割合）など，アプリケーションユーザの行動そのものも評価でき，アプリケーションの有効性を評価する際に役立つ．同様の事例として，広域無線通信網を用いた災害時の避難誘導システムにおいて，誘導情報の妥当性や避難誘導システム自身の有用性を確認，論議できる（例えば誘導システムを使う場合と使わない場合で無事に避難できる人数の違いを評価できる）などの利点が考えられる．

3 行動シミュレータ

行動シミュレータは，障害物や道路などのシミュレーション領域情報を読み込み，移動ノードの行動パターンを記述した C++ のクラス（行動クラス）のオブジェクトを生成する．また，それらの位置情報をシミュレーション時刻の進行に従い更新し，ネットワークシミュレータに通知する．さらに，行動クラスのオブジェクトとネットワークシミュレータのアプリケーション実装とのデータの受け渡しを行う．行動シミュレータ全体は C++ で実装される．

3.1 シミュレーション領域情報指定

行動シミュレータに対し指定するシミュレーション領域の構成要素は，(i) 折れ線分による塀などの障害物及び閉多角形による侵入不能領域，(ii) 侵入可能ポイント指定付きの閉多角形による広場などの自由移動領域，(iii) 交差点を表す交差点ポイント，及び (iv) ポイント間の接続関係を表す幅指定付きの線分による通路，である．この領域情報指定を解釈する関数が後述する行動シミュレータの main 関数で呼び出され，後述する行動管理クラスにおいて衝突判定などが可能な形式のオブジェクトに自動変換される．

3.2 行動クラス

移動ノードの行動を記述する行動クラスは，C++のクラスで指定する．ここで定義された行動クラスは，後述する行動管理クラスにおいて指定された条件のもとでオブジェクト化される．

行動クラスは以下のメンバを保持するクラスとして規定する．これらのメンバはすべてクラス外から参照可能なように public アクセス指定子を指定する．

- 現在位置を示す変数 `coordinate` L（`coordinate` は 2 つの整数メンバからなる構造体）
- 速度ベクトルを示す変数 `vector` V（`vector` は 2 つの実数メンバからなる構造体）
- アプリケーションへのデータ送信キュー及びアプリケーションからのデータ受信キューへのポインタ `app_data` $*Ain$, $*Aout$（`app_data` はネットワークシミュレータでのアプリケーション実装に依存するデータ型）
- 周辺ノード情報へのポインタ `neighbors` $**N$（`neighbors` は `coordinate` と `vector` をメンバとする構造体）
- シミュレーション領域情報へのポインタ `field` $**F$
- 周辺オブジェクト（障害物）情報へのポインタ `field` $**J$
- 速度ベクトル計算を行い，L 及び V の値を更新する関数 `vector` $calcVector()$ の実装

なお，ポインタで指定されるメンバは，この行動クラスのオブジェクトを管理する行動管理クラス（後述）からデータが与えられると仮定している．

上述の規定条件を満足する限り基本的にはその他は自由にユーザ定義可能であるが，ユーザ親和性向上の観点から，本稿では，街頭などでいくつかの目的地を持って移動する典型的な歩行者に対する歩行者リファレンスクラスを提案する（図 2）．

歩行者リファレンスクラスでは，速度ベクトル計算がそれぞれ異なる状態（通常移動状態，混雑回避移動状態，障害物回避移動状態，減速移動状態，滞在状態，ランダム移動状態）を定義し，状態間の遷移もしくは状態中の遷移（自己ループ遷移）において，内部変数の値を更新したり，アプリケーションキューに対する読み書きを行う．`coordinate` の配列である変数 $dest$ は，配列要素の先頭から順に経由すべき経由地を，最後尾要素に目的地を格納する．また $dest$ に対応し，各地点に対応する興味の程度（興味度）を表す整数値配列 $interest$ 及び到着予定時刻を表す整数値配列 $deadline$ を保持する．また，アプリケーションからの情報をどの程度活用するかを示す実数変数 IFO，滞在時間を示す整数変数 PS も保持する．列挙型変数 S は現状態を保持し，状態遷移は関数 $transState$ に記述している．

状態遷移の例として，例えば通常移動状態から混雑回避移動状態への遷移は，周辺ノード情報（周辺の端

```
class Walker : Node {
  typedef enum {通常移動, 混雑移動, 障害物回避移動,
                減速移動, 滞在, ランダム移動} state_t;
private:
  list<coordinate>  dest;
  list<int>         interest;
  list<int>         deadline;
  float             IFO;
  int               PS;
  state_t           S;

  /*情報に対する状態遷移*/
  void transState() {
    if (S == 通常移動){
      int rush = 混雑時間予測関数 (N);
      if ( 障害物判定関数 (J,N)) {
        S = 障害物回避移動;
      } else if (rush <= 1){
        S = 混雑回避移動;
      } else if (rush < 5) {
        S = 減速移動状態;
      }
    } else if (S == 滞在) {
      if (PS == 0) {
        要素削除関数 (0);
        S = (dest.size() > 0) ? 通常移動 : ランダム移動;
      } else {
        PS--;
      }
    } else if (L == dest[0]){
      S = 滞在;
      PS = 滞在時間計算関数 (interest[0])
    } else if (混雑時間予測関数 (N) >= 5 &&
               !障害物判定関数 (J)){
      S = 通常移動;
    }

    while (Aout->Size() != 0) {
      app_data appout = Aout->deque();
      q = 興味度計算関数 ( appout.pos, appout.time);
      /*時間優先で挿入*/
      for (i=0;i<deadline.size();i++){
        if (appout.time < deadline[i]) break;
      }
      要素挿入関数 ( i, appout.pos, appout.time,q);
    }
  }

public:
  /*状態によって速度決定*/
  void calcVector() {
    transState();
    if (S == 通常移動){
      V = (dest[0] - L) / (deadline[0] - current);
    } else if (S == 混雑回避移動) {
      V = 速度ベクトルライブラリ::衝突回避処理関数 (L,N);
    } else if (S == 障害物回避移動) {
      V = 速度ベクトルライブラリ::衝突回避処理関数 (L,J);
    } else if (S == 減速移動状態) {
      V = (dest[0] - L) / ((deadline[0] - current) * 2);
    } else if (S == 滞在中) {
      V = coordinate(0,0);
    } else {
      V = coordinate(random(-2,2),random(-2,2));
    }
    L = L + V; //位置更新
  }

private:
  /*現在の速度で進行した場合何秒後に混雑状況になるか*/
  int 混雑時間予測関数 (neighbors**);

  /*進路上に障害物があるかどうか*/
  bool 障害物判定関数 (field**,field**,neighbors**);

  /*IFO, L を利用して興味度を計算*/
  int 興味度計算関数 (coordinate,int);

  /*興味度から滞在時間を計算*/
  int 滞在時間計算関数 (int);

  /*dest, deadline, interest を一括して扱う*/
  void 要素削除関数 (int);
  void 要素挿入関数 (int,coordinate,int,int);

public:
  /*基底クラスで定義されてるから書かなくてもいい？*/
  coordinate        L;
  vector            V;
  app_data          *Ain,*Aout;
  neighbors         **N;
  field             **F,**J;
}
```

図 2: 歩行者リファレンスクラス

末の位置と速度ベクトル集合）と現在の速度ベクトルからこのまま進んだ場合に現速度での進行が不可能であると判断される場合に遷移するような遷移条件などを定義する．また，自己ループ遷移の例としては，例えばアプリケーションキュー $Aout$ を介して得られる座標と時刻の組（例えば店舗の位置とセール時刻）に対し，自身の情報活用度に応じた興味度を設定し，配列 $dest$，$interest$ 及び $deadline$ の適当な位置に挿入する動作や，経由地の順序を興味度にしたがって入れ替える動作など，状態変更を伴わず変数値のみを更新する動作が挙げられる（$transState$ 内の while ループ）．

$calcVector$ では，速度ベクトル計算の前に $transState$ を呼び出し，(可能な状態遷移があれば）状態遷移を行った上で，状態に対応した速度ベクトル計算を行う．速度ベクトル計算は，配列 $dest$ の先頭座標に対して，(もし未計算であれば到着経路計算アルゴリズムを適用してシミュレーション領域情報 F から経路を算出した上で）ベクトルの方向を計算し，さらにその座標に対応する到着予定時刻と距離からベクトル長を算出する．なお，周辺ノードや障害物が存在する状態ではそれに対応する衝突回避行動を速度ベクトル計算に含めなければならないが，それらを逐一記述することは容易でないため，MobiREAL ではいくつかの状況で利用できる速度ベクトル計算の基本ルーチンをライブラリ化している．例えば，対面して近づいてくるノードに対しては，文献 [4] で紹介されている衝突回避処理方式に基づいた速度ベクトル計算補助関数を提供している．この関数は，周辺情報に含まれる周辺ノードの速度ベクトルと自身の速度ベクトルから，現在の速度ベクトルで進んだときに衝突すると予想される領域を求め，その

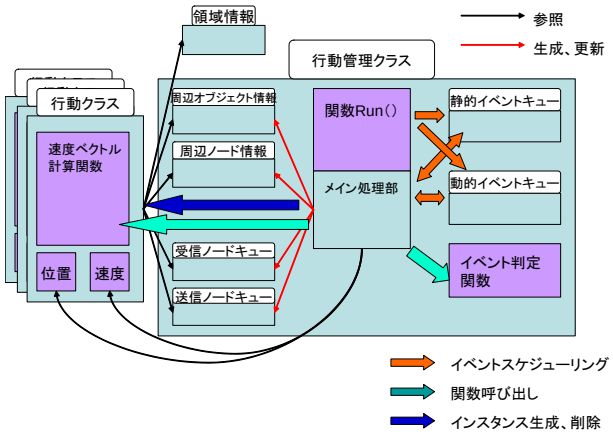

図 3: 行動シミュレータクラス関係図

領域を回避するような速度ベクトルを計算する．同様に，壁などの障害物に対しても，障害物を動かないノードの集合とみなすことで速度ベクトル計算を行うことができる．

3.3 行動管理クラス

同様に C++のクラスとして記述する行動管理クラスは，個々の移動ノードに対応する行動クラスのオブジェクトの生成や削除の指定に従いオブジェクトを管理する，行動シミュレータの主要動作を記述する．さらに，シミュレーション時刻の進行を管理し，各行動クラスオブジェクトの $calcVector$ 関数を呼び出してそれらの位置及び速度ベクトルを更新するとともに，各行動クラスオブジェクトに対しシミュレーション領域情報，周辺ノード情報，周辺オブジェクト情報を提供する．また，ネットワークシミュレータとのインタフェースを介し，各行動クラスオブジェクトのアプリケーションキューに対するデータ受け渡しを行う（図 3）．

行動管理クラスは，Run メンバ関数にその振る舞いを記述する．利用者は，ノード生成及び削除などのイベント指定のためにあらかじめ用意された 2 つのメンバ関数 $putStaticEvent$ または $putDynamicEvent$ を用いてイベントの種類（ノード生成，削除）と共に指定する．なお，$putStaticEvent$ はイベント発生時刻があらかじめ指定できる場合に用い，$putDynamicEvent$ はイベントの発生条件を指定することで動的なイベント生成を実現する．また，それらの指定後にシミュレーション進行ルーチンを指定し，シミュレーション時刻をインクリメントしながら各行動クラスオブジェクトの $calcVector$ を呼び出すメンバ関数 $updateVectors()$ を呼び出す．またそれに伴い周辺ノード情報や周辺オブジェクト情報を更新するメンバ関数 $updateEnv()$，及びイベントキューの更新を行うメンバ関数 $updateQueue()$ も呼び出す．

なお，$putDynamicEvent$ で利用するイベント条件は，イベント判定を行うメンバ関数を実装することで実現する．この関数はシミュレーション領域情報やノードの現在位置や速度ベクトルなど，行動管理クラスがアクセス可能なデータを利用して記述することができる．$putDynamicEvent$ により設定されたイベントは動的イベント専用のキューに挿入され，シミュレーション時刻のインクリメントと共に，そのイベント条件が成り立つか否かが $updateQueue$ 関数により検査される．

なお，行動管理クラスは比較的プリミティブなレベルでの記述が可能である反面，全てのノードの生成や内部変数の設定などを個々に行うのは大規模シミュレーションでは非効率的であるため，抽象レベルでのシナリオ指定を支援するライブラリを提供することが望ましい．このうちの一つとして，各ノードごとの生成だけではなく，一定の出現パターンでノードを生成するノード生成器の提供を考えている．ノード生成器は，ノード生成頻度，及びノードインスタンス化の際の初期値の分布関数が与えられた場合に，移動ノード群を自動生成するため，個々のオブジェクト生成を指定する手間を軽減できる．

3.4 main クラス

main クラスでは，シミュレーション領域情報を読み込んでフィールドオブジェクトを生成した後，行動管理クラスオブジェクトを生成し Run 関数を呼び出す．また，シミュレーションを終了させる時刻の指定などもここで行う．

4 ネットワークシミュレータ

ネットワークシミュレータ部は，アプリケーション層以下のシミュレートを行う．ただし，行動シミュレータからシミュレーションの進行に伴い得られる移動ノードの位置情報に基づき，ネットワークシミュレータが管理する移動ノードの位置情報を更新しながらネットワークをシミュレートする．また，アプリケーションユーザに相当する行動シミュレータからアプリケーションへのユーザ入力を受け取り，ユーザへのデータ出力値を渡す動作をインタラクティブに行いながらシミュレーションを進行させる．

```
Run(int stoptime){
  [ユーザがシナリオを記述する]
  for (i=0;i<100;i++){                          //時刻 100 まで毎秒ノードを一つ生成
    putStaticEvent(i,"create",Walker,initparam);  //initparam は Walker 生成時に与えるパラメータリスト
  }
  putStaticEvent(200,"delete",10);              //時刻 200 でノード ID が 10 のノードを削除
  putDynamicEvent(0,"create",Walker,initparam,0); //動的イベント No.0 のイベントを作成
  putDynamicEvent(1,"delete",1,10);             //動的イベント No.1 のイベントを作成（条件成立後 10 秒後に発生）


  [メイン処理部]
  sendInitializeData();                         //ネットワークシミュレータに初期情報を与える
  for (clk=0;clk<stoptime;clk++){
    updateEnv();                                //各ノードに適切な変数値を設定
    updateVectors();                            //状態遷移，速度計算
    updateQueue();                              //動的イベントの発生判定チェック
    if (clk % t == 0) {
      rcvdata = rcvFromNetsim();                //アプリケーションの出力を受け取る
      sendToNetsim(senddata);                   //位置情報，アプリケーションのユーザ入力を渡す
    }
  }
  sendStopData();                               //終了通知
}

[ユーザが定義する動的イベント判定関数]
bool checkEvent(int evNo){
  switch(evNo){
    case 0: if (node.size() < 50) return false;   //動的イベント 0 の発生条件式;
            else return (node[50]->S == Walker::滞在);
    case 1: return (node[1]->L.x < 100);          //動的イベント 1 の発生条件式;
  }
}
```

図 4: 行動管理クラス記述例

4.1 ネットワークシミュレータと行動シミュレータの連携

ネットワークシミュレータと行動シミュレータはそれぞれシミュレーション時間 t ごと独立にシミュレーションを実行し，その結果得られるデータを他方に送信する．t シミュレーション時間におけるシミュレーション結果を相互に交換し終えるまで次の時間のシミュレーションを停止する．

具体的には以下のように実行される．シミュレーション開始時には，オブジェクトや移動ノードの初期配置を含む領域情報とともに，シミュレーション時刻 0 における各ノードの座標と速度ベクトル，アプリケーションへの入力が行動シミュレータからネットワークシミュレータへ渡される（図 5 中 (a) の矢印）．その後，各シミュレーション時刻 $n*t$ ($n = 0, 1, ...$) において，ネットワークシミュレータは移動ノードごとに，$[(n-1)*t, n*t]$ のネットワークシミュレーション中において得られたアプリケーションからのデータ出力を行動シミュレータに渡す（図 5 の (b) の矢印）．同様に，各シミュレーション時刻 $n*t$ ($n = 0, 1, ...$) において，行動シミュレータは各移動ノードごとに，$[(n-1)*t, n*t]$ の行動シミュレーション中において得られた，(i) アプリケーションへのユーザ入力，(ii) 時刻 $n*t$ における座標及び速度ベクトル，をネットワークシミュレータに渡す

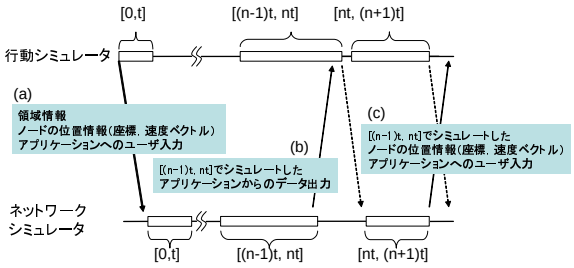

図 5: 行動シミュレータとネットワークシミュレータの相互連携

（図 5 中 (c) の矢印）．このように，行動シミュレータとネットワークシミュレータは互いに独立にシミュレーション時間 $[(n-1)*t, n*t]$ のシミュレーションを行い，シミュレーション時刻 $n*t$ に一旦停止する．その際に，他方にその結果を送信し，他方からの結果を受け取り，シミュレーション時間 $[n*t, (n+1)*t]$ のシミュレーションを行う．

t の大きさの決定は，アプリケーションに求められるシミュレーション精度とシミュレーションに要する時間とのトレードオフに基づきシミュレータ使用者が設定する．

4.2 GTNetS を利用したネットワークシミュレータ部の実装

MobiREAL では，ネットワークシミュレータとして，米 Georgia Institute of Technology で開発された GTNetS[5] を使用する．GTNetS は，スケーラビリティに重点をおいて設計されており，標準で有線ネットワークの並列実行環境を備えている．MobiREAL では，行動シミュレータ部との連携部分（データ交換処理部分）を独自クラス（インタラクションクラス）として実装し GTNetS に組み込む．この連携部分を含めたネットワークシミュレータ部の構成を図 6 に示す．

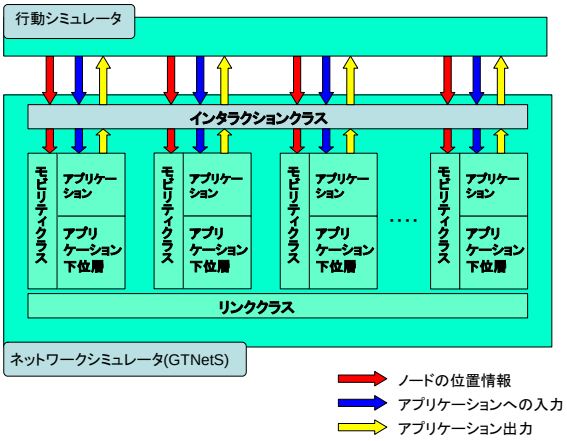

図 6: ネットワークシミュレータ部構成図

4.2.1 インタラクションクラス

GTNetS は discrete-event タイプのネットワークシミュレータとして実装されており，パケットの送受信等のシミュレーションイベントは，その実行時刻と共にイベントキューに挿入される．その後，イベントキューに挿入されたイベントは，シミュレーション時間順に取り出され処理される．これらイベントキューへのイベント挿入，処理の繰り返しによりシミュレーションが構成される．MobiREAL では，行動シミュレータとのデータ交換と，行動シミュレータから渡されるノード情報，アプリケーション情報に対する処理を行うために，インタラクションイベントを作成し，シミュレーション時間 t ごとに処理する．

まずシミュレーション前にインタラクションイベントを時刻 t にスケジューリングする．このスケジューリングにより，$[0, t]$ のシミュレーションを行った後，時刻 t でインタラクションイベントが処理され，行動シミュレータとのデータの受け渡し，受け取ったデータに対する処理が行われる．インタラクションイベントの最後には，次のインタラクションイベントを時刻 $2t$ にスケジューリングし，次回の行動シミュレータとのデータ受け渡しに備える．このような一連の処理を，行動シミュレータによってシミュレーション終了の情報が渡されるまで繰り返す．

4.2.2 ノードの生成，削除に対する拡張

GTNetS は，ネットワークトポロジの変化に乏しい有線ネットワークを主な対象としているため，シミュレーション中の新規ノードの追加，既存ノードの削除は行わない．使用者がシミュレーションシナリオとして，ネットワーク通信を行うノードの生成やノード上で動作するアプリケーションの設定等をシミュレーション開始前に行うことを想定して設計されている．

一方，無線ネットワークにおいては，ノードの移動だけでなく，ノードの発生や消滅も頻繁に発生するため，これらのノードの変化に対し GTNetS で適切に対処する必要がある．

ネットワークシミュレータ部では GTNetS におけるノードの管理方法の変更と，ノード間のリンクを表現する Link クラス（図 6 リンククラス）の拡張を行い，柔軟にノードの生成，削除が出来るようにしている．しかしながら，現時点ではこの拡張により GTNetS の機能の一部が使用できなくなっている．アプリケーション開発者が GTNetS が提供するライブラリを活用できるようにするため，この問題への対策を行う予定である．

4.2.3 ノードの移動に対する拡張

GTNetS ではノード移動を計算, 管理するモビリティクラスが用意されており，各ノードの移動は，ノードに割り当てられたモビリティクラスのインスタンスに依存する（図 6 参照）．MobiREAL では，行動シミュレータから受け取ったノードの速度ベクトルをネットワークシミュレータに反映させるため，速度と変化時刻の組を蓄積出来るようなモビリティクラスを作成して，GTNetS に組み込み，インタラクションクラスから速度ベクトルを受け取る．

モビリティクラスは，速度ベクトルを受け取った時点では，ノード位置を更新せず，蓄積情報の更新のみを行う．無線の到達計算等でノードの現在位置が必要になった場合のみ，ノード位置と速度，蓄積情報によって位置計算を行う．また，行動シミュレータとネットワークシミュレータは独立に位置情報を管理するため，ネットワークシミュレータは，行動シミュレータから速度ベクトルのみならず，絶対座標を適宜受け取ることで位置を補正する．このような絶対座標を受け取った場合は，過去の速度ベクトルは不要となるため，蓄積情報の初期化を行う．

4.2.4 アプリケーションの実装

GTNetS におけるアプリケーションは，クラス Application の派生クラスとして定義される．設計者は GTNetS が提供するトランスポート層，ネットワーク層等のライブラリを使用し，独自のアプリケーションを組み込む．

ネットワークシミュレータ部では，MobiREAL 用アプリケーションのインターフェイスとして機能させるために，Application クラスに，行動シミュレータからのユーザ入力や行動シミュレータへの出力を扱うための関数を加えた MobiREAL 用アプリケーション基底クラスを用意する．行動シミュレータから受け取ったアプリケーションへのユーザ入力は，インタラクションクラスによって文字列としてアプリケーションに渡される．クラス内の入力文字列の解釈部分を記述することで，ユーザ入力に対するアプリケーションのふるまいを自由に設計することができる．また，アプリケーションからの出力もアプリケーション設計者が文字列として定義する．この文字列は，シミュレーションによってアプリケーションが情報を受信したとき，インタラクションクラスに渡される．シミュレーション時間 $t*n$ におけるインタラクション時に，$[t*(n-1), t*n]$ にインタラクションクラスに集められたデータが，一つの文字列として行動シミュレータに渡される．

5 アニメータによる視覚化

アニメータは移動ノードの動きやネットワークの可視化を行うことで，シミュレーション結果の直観的理解を助けることを目的としている．現在，Windows 上で動作する MobiREAL アニメータを処理速度，描画能力の観点から DirectX9 を採用して実装を進めている．現時点でのアニメータは俯瞰視点での移動ノードの行動の可視化を実装しており，シミュレーション領域を表す地図画像の上で，移動ノード（人）を表す小さい円と，人が持つ端末の無線範囲を表す半透明の円が，ノードの行動に従って移動する．アニメータのスナップショットを図 7 に示す．

地図画像の上には，ビルなどのオブジェクトを表示することができ，無線伝達範囲を表す大きい円とあわせて，おおまかにではあるが各ノード間における通信の可否を確認することができる．

アニメータの機能として，レイヤごとに可視化するか否かを指定できる機能が重要であると考えられる．例えば，アプリケーション層であれば，実装したアプリケーションによる情報伝達状況が，移動ノードを表す円の色を変化させることで見分けられる機能，トランスポート層であれば，エンド間通信がどのノードを経由しているかなどを表示する機能，ネットワーク層であれば，ネットワークトポロジをノード間線分で補助表示する機能，リンク層であれば特定リンクのパケットレベルの可視化，物理層であれば，電波の減衰の程度が色濃度の違いで判断できる機能，などアプリケーション開発者の目的に応じた可視化を実現することを目標としている．

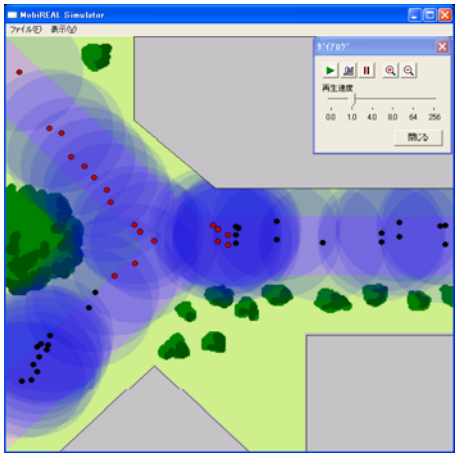

図 7: MobiREAL アニメータのスナップショット

6 まとめ

本稿では現実世界の人物行動などを記述でき，それに基づいたモバイルアドホックネットワークアプリケーションの評価を可能とするネットワークシミュレータ MobiREAL の活用事例と詳細設計を中心に述べた．今後は引き続き MobiREAL シミュレータの開発を進め，その実用性の評価を行っていく予定である．

参考文献

[1] ns-2. http://www.isi.edu/nsnam/.

[2] T. Tracy, B. Jeff, and Vanessa D. A survey of mobility models for ad hoc network research. Wireless Comm. & Mobile Computing (WCMC): Special Issue on Mobile Ad Hoc Networking: Research, Trends, and Applications, Vol. 2, No. 5, pp. 483 – 502, 2002.

[3] X. Zeng, R. Bagrodia, and Gerla M. GloMoSim: A library for the parallel simulation of large-scale wireless networks. In *Proc. of ACM Parallel and Distributeal Simulation (PADS'98)*, pp. 154 – 161, 1998.

[4] 岡田公孝, 和田剛, 高橋幸雄. 個人行動をベースにした歩行モデルと歩行流シミュレーション. 日本オペレーションズ・リサーチ学会, 春季研究発表会, 2003.

[5] G. F. Riley. The Georgia Tech network simulator. In *Proc. of the ACM SIGCOMM Workshop on Models, Methods and Tools for Reproducible Network Research*, pp. 5 – 12, 2003.