C++に基づく拡張可能な離散型シミュレーション言語

4J-3

森川拓次，三角貞治，越田一郎

東京工科大学

1．はじめに

あるシステムの定量的な特性を調べる方法として対象と等価なモデルを作りシミュレーションを行う方法が定着している．このようなシミュレーションを行う場合，対象とするシステムのイメージに近いモデルが組めること，そのモデルの表記が容易であること，十分な統計量が得られること（精度も含む），が重要であるが今回報告するシステムは，イメージに近いモデルが容易に組めることを目的としている．

本システムはC++のクラスセットからオブジェクトを生成し，それを組み合わせてモデルを組む方法を採用している．

2．特徴

本シミュレーターはシミュレーションの対象物に対応するノードを宣言しそれを組み合わせてモデルを作り上げている．そのため複雑で自由度の高いプログラミングが可能である．また，C++のクラスとして各ノードが記述されているため導出クラスを作ることにより，容易に拡張が出来る．各基本ノードには拡張用に

startExp（開始時に呼ばれる）
activateExp（動作時に呼ばれる）
reciveExp（ﾒｯｾｰｼﾞを渡される時に呼ばれる）
endExp（終了時に呼ばれる）

の4つの仮想関数が用意されている．

利用者は，導出クラスを作るとき，上記のそれぞれの名前で各動作を拡張できる．

3．シミュレーターの基本モデル

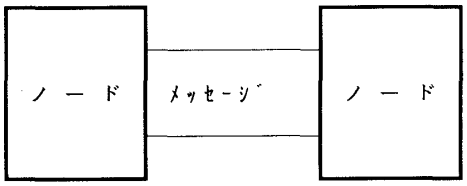

図1

本シミュレーターは基本的なモデルとして図1のようなものを想定している．モデルの基本構成要素であるノード（Node）間をﾒｯｾｰｼﾞ（Message）が流れることにより，処理を表現している．

4．シミュレーターの動作原理

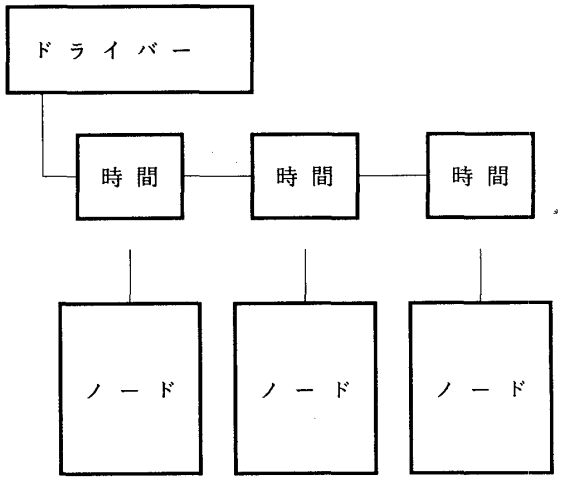

図2

Extendablue simulation language on C++
Tokyo Engineering University

Takuji MORIKAWA, Sadaharu MISUMI, Ichiro KOSHIDA

シミュレーターの動作原理は図２のとおりである．

各ノードが次に生起するイベントの発生予約時間を持ち，その時間の順番に，ドライバーが各ノードを活性化することにより作動する．ドラーバーの持つ予約時間のリストがイベントリストとなっている．

５．記述方法

```
double lambda1, lambda2;
double Src(){
    return Poisson( lambda1 );
}
double Srv(){
    return Poisson( lambda2 );
}
main(){
    Driver   world;
    Source   source1;
    FIFO     queue1;
    Server   server1;
    Sink     sink1;

    world.setNode( &source1 );
    world.setNode( &queue1 );
    world.setNode( &server1 );
    world.setNode( &sink1 );
    source1.setOutput( &queue1 );
    queue1.setOutput( &server1 );
    server1.setOutput( &sink1 );
    source1.setDistribution(Src);
    server1.setDistribution(Srv);

    cout << "Lambda1,Lambda2t:";
    cin  >> lambda1 >> lambda2;
    queue1.setEndCount( 10000 );

    world.run();
    queue1.print();
}
```

リスト１

リスト１はメッセージを発生するソース（Source）から待ち行列キュー（この場合FIFO）そして処理をするサーバー（Server）を経てメッセージの消去のためのシンク（Sink）へとつなげたモデルが対象である．

プログラムは以下のとおりである．宣言の後，ドライバーのworldに関数setNodeで宣言したノードを登録し，各ノードのメッセージ出力先を関数setOutputで指定する．リスト１の中で宣言したポアソン分布関数を関数setDistributionでそれぞれsource1，server1に設定する．メッセージを何個受け取ったら終了にするかをqueue1に設定し，ドライバーに関数runでシミュレーションの実行を指示する．終了後，結果をqueue1に出力させて終了する．

６．おわりに

現在用意されているクラスだけでは複雑なシミュレーションモデルをイメージどおりに組み立てる事が難しいため，シミュレーションの用途に特化した各種ノードの整備が必要と考えられる．

またシミュレーションに必要な事以外は，なるべく記述しないですませる方法として，リスト１で，本質的に必要でない部分（ノードの宣言，ドライバーへの登録など）は記述しないで，なんらかのトランスレイターを通してC++のメインプログラムとすること，視覚化インターフェースを使用してのモデルの構築などが考えられる．

〈参考文献〉

［１］越田一郎：C++に基づくシミュレーション言語の試作，1989年電子情報通信学会春期全国大会講演論文集，1989