時空間構造を導入したオブジェクト指向モデル

3G－5

阿部一裕 瀬尾和男 尾崎敦夫 清原良三 中島克人
RWCP† 超並列三菱研究室††

1 はじめに

筆者らは,RWCP プロジェクトの一環として, 実世界シミュレーション, 及びシミュレーションプログラムの記述性の向上を目的とし時空間構造を導入した超並列オブジェクト指向モデルの研究を進めている. 実世界シミュレーションでは, 超並列マシンがもたらす膨大な計算パワーを用い, 対象を単純なモデルになおすことなく, 個々の要素の動作を詳細にシミュレーションすることから系全体の性質を把握することを目指している.

すでに文献 1) において, 不特定なオブジェクト間の通信の媒介, および楽観的分散時刻管理を支援する場の概念を導入したモデルの提案をおこなった.

本稿では, 実世界シミュレーションで対象とする問題のモデル化に必要な時空間構造の内, オブジェクト間の同時相互参照に着目し, デッドロックをふせぎ, かつシミュレーション時刻に関する因果関係を満たしたオブジェクト間の同時相互参照を安全かつ簡潔に記述するためのオブジェクトモデルの提案をおこなう. また, このモデルを実現し CM5,Paragon など高並列計算機をターゲットとするプロトタイプ処理系の実装について報告する.

2 従来のオブジェクトモデルの問題点

実世界シミュレーションで対象とする問題では, 同一時刻に要素間で相互参照がおこなわれる場合をモデル化する必要がある. 例えば図 1にしめすように異なる方向からきた二人が交差点を同時に渡ろうとする場合である. このような場合, オブジェクトの状態を更新するメソッドと, 他のオブジェクトに自分の状態を参照させる被参照メソッドを同時に実行する必要がある.

並列オブジェクト指向モデルには, 同時に複数のメッセージの実行を許し, オブジェクト内に複数個のアクティビティを許す多重スレッドモデルと, メッセージの実行は逐次化され, オブジェクト内のアクティビティはたかだかひとつである単一スレッドモデルがある.

オブジェクト間で同時相互参照をおこなう問題を多重スレッドモデルで記述した場合, 状態更新メソッドと被参照メソッドが同時に実行されるので, 双方のオブジェクトが返答待ちとなるデッドロックは生じない. しかし

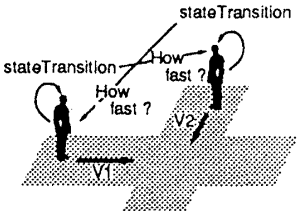

図 1: オブジェクト間の相互参照

ふたつのメソッドの実行のタイミングにより, 被参照メソッドが参照する値が, 状態更新メソッドにより更新される前の状態か更新された後の状態かは保証されない.

単一スレッドモデルで記述した場合, 双方のオブジェクトがともに被参照メソッドからの返事待ちの状態になりデッドロックが生じる危険性がある. 速達モードメッセージの機構がモデルに備えられていれば, 速達モードメッセージで被参照メソッドを実行することによりデッドロックを回避することが可能である. しかし, この場合でも, 参照する値が, 更新前の値か更新後の値かは保証されない. またプログラムが複雑になることが報告されている [2].

3 提案するオブジェクトモデル

状態を格納する領域を, 参照用の領域と更新用の領域に分離し, 多重スレッドモデルを提供することにより, オブジェクト間の同時相互参照によるデッドロックをふせぎ, かつシミュレーション時刻に関する因果関係を満たした参照を実現するオブジェクトモデルを提案する.

本モデルでは, シミュレーション対象中の動作主体を ENTITY と呼ぶオブジェクトでモデル化する. ENTITY は, 図 2にしめすように stateVariables,methods, stateSelector から構成される.

stateVariables としては,ENTITY が起動する時刻 (time), 現時刻の状態 (stateT$_j$), 更新した状態を書き込む領域で次回起動時刻の状態となる領域 (stateT$_i$), 過去の時刻における状態 (stateT$_k$,..) を持つ. 各状態にはタイムスタンプが付けられており, 例えばタイムスタンプが T$_j$ である stateT$_j$ は, 時刻 T(T$_k$<T<=T$_j$) における ENTITY の状態をしめす.

methods には,stateTransiton メソッドひとつと, 複数個の reference メソッドがある. stateTransiton メソッドは,ENTITY の主体的な動作をしめすメソッドである. スケジューラのみが実行でき, 状態の更新をおこなうことができる. reference メソッドは, 他のオブジェクトからの参照要求に返答するメソッドである. 参照のみ可能で状態を更新することはできない.

An Object Oriented Model with Time and Space Management Mechanism, Abe,K.,Seo,K.,Ozaki,A.,Kiyohara,R.,Nakajima,K. RWCP† Massively Parallel Systems Mitsubishi Lab.††.

† RWCP:Real World Computing Partnership

(新情報処理開発機構) †† 三菱電機 (株) 内

メッセージには, 送信元 ENTITY のメッセージ送信時の時刻が付加される. 受信した ENTITY は, stateSelector で, メッセージに付加された時刻における状態を選択し, その環境の下でメソッドを実行する. メッセージの時刻が ENTITY の時刻より未来であるメッセージ (msgC) の処理は, ENTITY がメッセージの時刻になるまでサスペンドする. そうでない場合 (msgA,msgD) メソッドを即時に実行する多重スレッドモデルを提供している.

stateTransiton メソッドでは, 現時刻の状態 ($stateT_j$) と, 参照要求メッセージ (msgB) を送信して得た他の ENTITY の時刻 T_j の状態から, 状態の更新値を求める. 更新値を次回起動時刻の状態となる領域 ($stateT_i$) に書き込む. ENTITY 自身が次回起動する時刻を, このメソッドで決定し, 実行終了後, 変数 time の値を次回起動する時刻に書き換え ENTITY の時刻を進める.

同一時刻にある, ふたつの ENTITY 間で同時相互参照がおこった場合,stateTransiton メソッドと reference メソッドが多重に実行され, 返答待ちによるデッドロックは生じない. また stateTransiton メソッドは状態の更新値を次回起動時刻の状態となる領域に書き込むので, reference メソッドは状態更新前の状態を参照することが保証される.

状態変数を何時刻分もつかは, 分散時刻管理手法により変化する. 保守的分散時刻管理手法を採用した場合, 過去の状態が参照される可能性がなくなった段階で, スケジューラが ENTITY に stateTransition メソッドを起動させるメッセージ (msgA) を送信するので, 2 時刻分の状態 (stateTransition メソッド実行時は, 現時刻と次回起動時刻の状態, 実行終了後は前回起動時刻と次回起動時刻の状態) があれば十分である. 一方楽観的分散時刻管理手法を採用した場合, 時刻のロールバックをおこなうために可能なかぎり多くの状態を保存する必要がある.

4 処理系の実装

前節でのべた機能を実現するためのプロトタイプ処理系を作成した. この処理系は,GNU C++ 処理系と, ENTITY, マルチスレッド, 演算ノード間通信などの機能を実現するクラスライブラリから構成される.

ENTITY は, 図 3にしめすように, クラス stateSelector のオブジェクトと, クラス Entity(図 4) のオブジェクト複数個からなる複合オブジェクトとして実現している. Entity クラスの導出クラスを通常の C++ の new 演算子で生成し, 図 6の構文にしたがいシステムに登録すると, Entity オブジェクトのコピーが規定値個生成され, Entity で定義された変数 next,before に Entity オブジェクトへのポインタが循環的に代入される. 次に処理系で定義された stateSelector オブジェクトが生成され図 3にしめす構造が形成される.

Entity は ENTITY の状態を保持するオブジェクトであり, シミュレーション中の動作主体は,Entity の導出クラスとして定義する. 1 時刻分の状態変数, 1 時刻でおこなう ENTITY の振舞いを記述する stateTransition メソッド 1 つと, 被参照用の reference メソッドを複数個記述する (図 5). stateTransition メソッド実行時に,stateSelector により現時刻の状態を保存した Entity の内容が, next 変数によりさされている次回起動時刻の状態を保存する Entity にコピーされる. 状態の更新は, 図 5の stateTransition メソッド中にしめすように next 変数をもちい, 次回起動時刻の状態を保持する Entity に書き込むことによりおこなう.

5 おわりに

今後, 本稿で提案したオブジェクトモデルの有効性を CM5,Paragon などの高並列計算機上で検討していく予定である.

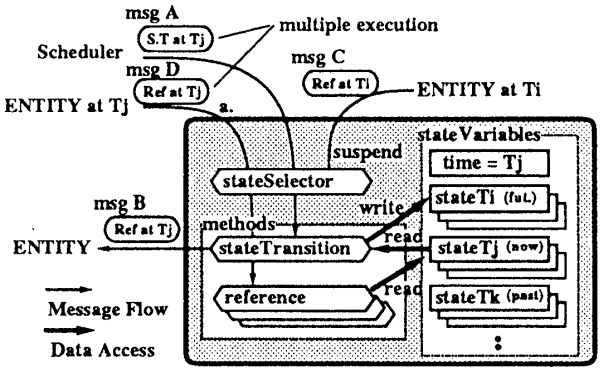

図 2: ENTITY の内部構造

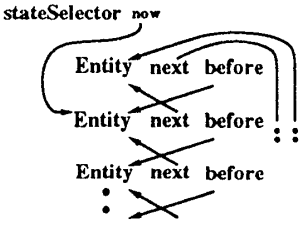

図 3: ENTITY の実装

```
class Entity{
protected:
 Entity*  next;
private:
 int      timestamp;
 Entity*  before;
};
```

図 4: Entity クラス

```
class Vehicle :public Entity{
private:
 int x,y,vx,vy;          // 状態変数
 int stateTransiton(){ // 状態更新
   next->x = x + vx; ...};
public:
 int refX() const{return x;};
 ...                     // 状態参照
};
```

図 5: クラス定義例

```
Entity* anEntity
  = new Vehicle(....);
scheduler->
  in(anEntity,initTime);
```

図 6: ENTITY の生成

参考文献

1) 尾崎敦夫 他: 時空間オブジェクトモデルの基本設計 - ミクロ交通シミュレーションへの適用検討 -, 情報処理学会第 47 回全国大会, 1B-2,1993.

2) Ishikawa,Y.: Communication Mechanism on Autonomous Objects, OOPSLA92,pp.303-314,1992.