ハイパフォーマンス
コンピューティング 67－18
（１９９７．８．２０）

自動分散型オブジェクト指向シミュレーションシステムの生成

上原 均　畠山正行
茨城大学工学部情報工学科
E-mail: {uehara,masayuki}@cis.ibaraki.ac.jp

要旨

オブジェクト指向数値シミュレーション・システムの短所である実行速度がやや遅いという問題の解決法の一つとして数値計算システムの分散化があるが、従来の分散型システムの構築では専用ライブラリなどの知識が必須であった。本研究では逐次型オブジェクト指向シミュレーション・システムのソースコードから、分散処理が自動化された自動分散型オブジェクト指向シミュレーション・システムのソースコードを生成する自動生成システムを構築することで解決を試みた。本研究の自動生成システムを用いることで、ユーザに分散処理記述の負担をかけること無く分散型オブジェクト指向シミュレーション・システムを構築できるようになった。

Automatic Source Code Generator of Automatically Distributed Object-oriented Simulation System

Hitoshi Uehara　Masayuki Hatakeyama
Department of Computer & Information Sciences, Ibaraki University
E-mail: {uehara,masayuki}@cis.ibaraki.ac.jp

Abstract

The aim of this study is to develop an automatically distributed computing system based on the object-oriented paradigm for numerical simulation without the user programming. In this study, we have developed an automatic source code generator that automatically generate the source code of the distributed computing system from the source code of the sequential computing system for the numerical simulation. Since we have perceived that the object-oriented paradigm is suitable for the distributed computing, we have developed this automatic source code generator. The development of the distributed OO-simulation systems leads a realization of the distributed-computing without any user's load.

1　はじめに

オブジェクト指向パラダイムに基づいて構築された数値シミュレーション・システム [1](以下、OOシミュレーションと略) は実行時の柔軟性などの幾つかの長所を持っているが、逆に実行速度がやや遅くなる場合があるという短所も持っている。この短所の克服方法の一つとしてシステムの分散/並行化がある。しかし現状では、分散/並行型システムの構築には PVM[2] などの分散/並行型システム構築用ライブラリの利用が事実上必須であり、ユーザにはそれらの技術の習得が強いられている。

そこで本研究では分散型 OO シミュレーションシステム構築時におけるユーザの負担軽減を目的として、**自動分散型 OO シミュレーション・システムのソースコード自動生成システム**を開発する。この生成システムでは、C++で記述された単一ホスト上で駆動する逐次 OO シミュレーション・システムのソースコードから自動分散型 OO シミュレーション・システムのソースコードを生成することを

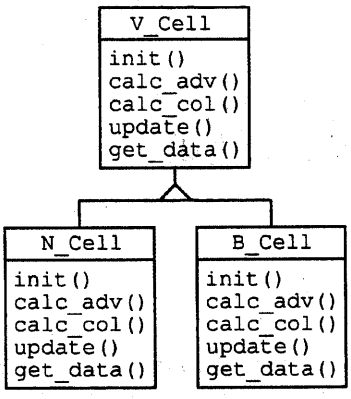

図 1: OO シミュレーションシステムのクラス図

目指す。

2 オブジェクト指向シミュレーション・システムの概要

OO シミュレーションでは、(その粒度に多少の差異があるが) 離散的な計算単位であるオブジェクト群から構成され、各々のオブジェクトは計算パラメータなどをその属性として持ち、初期化、計算、データ取り出し、値更新などのメソッドを持っている。この具体例として、離散速度座標法 (DVO 法) [3] の OO シミュレーション・システムでのクラスの概略を図 1に示す。

この例の V_Cell は抽象化された上位クラスであり、N_Cell は通常のセルを、B_Cell は境界領域のセルを各々モデル化したクラスである。N_Cell と B_Cell は共に V_Cell を継承しているためにインタフェース (=メソッド) は同一であるが内部処理は異なるため、各々のクラスで各メソッドを再定義している。

3 自動分散型オブジェクト指向シミュレーションの実現

3.1 自動分散型 OO シミュレーション・システムの定義とその構成

本研究では逐次型システムのソースコードから分散型システムのソースコードを自動生成することを目的としている。逐次型システムではオブジェクトの分散配置や分散環境での通信処理などを考慮する必要がないため、そのソースコードにはそれらの分散処理に関する記述がない。そのため本研究では、従来はユーザが逐一記述していたそれらの処理を、自動的に処理するソースコードを自動生成する。この自動化された分散処理から、本研究で構築する自動生成システムによって生成される分散型システムのことを自動分散型システムと呼ぶ。

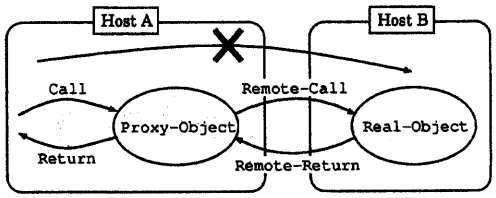

図 2: ProxyObject の概念

この自動分散型システムの構成には、PVM[2] などで用いられているマスタ・スレイブ方式を取り入れる。マスタはユーザが記述した逐次型システムのメイン部分の記述と自動生成した ProxyObject (次節で述べる) の定義から構成され、スレイブはユーザが定義したオブジェクトのクラス定義と自動生成されたメッセージ処理部分などから構成される。

3.2 分散オブジェクトの実現：ProxyObject の導入

分散型オブジェクト指向システムを実現する上での問題点の一つとして、分散環境上でのオブジェクト間通信があげられる。本研究では、その解決策として ProxyObject を用いる。ProxyObject とは本来の通信対象であるオブジェクトと同じインタフェースを持つオブジェクトであり、その内部において分散環境での通信の詳細を隠蔽する (図 2参照)。この ProxyObject を仲介にすることで、分散環境における通信処理の詳細を意識しない分散透過性が実現できる。

この ProxyObject の生成/削除はそれに対応するユーザ定義のオブジェクトの生成/削除と同義である。

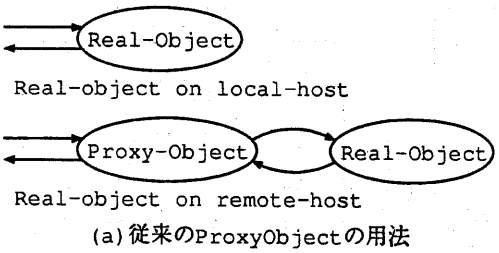

図 3: ProxyObject の用法

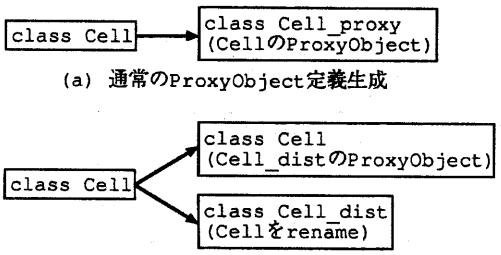

図 4: ProxyObject の定義生成

3.3 分散透明性の実現：ProxyObject との「入れ替え」

ProxyObject を用いて記述するには、そのオブジェクトがリモートホストに存在するか否かをプログラマは明確に意識しなければならない（図 3(a) 参照)。よって ProxyObject を用いただけでは、オブジェクトが分散している事自体を意識しない「分散透明性」は実現できない。3.1節で述べたように逐次型システムの記述には分散処理の記述がないため、この分散透明性を自動生成システムにおいて実現することが不可欠である。

そこで本研究では、通信の仲介役であった ProxyObject を本来の通信対象のオブジェクトと「入れ替え」ることで分散透明性を実現する。

通常はユーザが定義したクラス定義から別個に ProxyObject の定義を生成し（図 4(a) 参照)、リモートホストにオブジェクトが存在する場合のみ、ProxyObject が用いられる。しかし本研究では、ユ

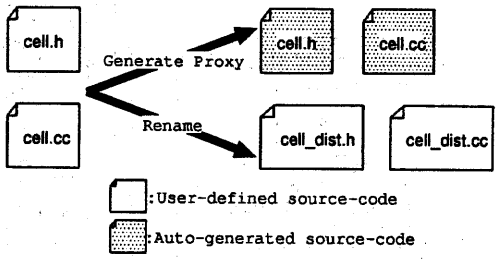

図 5: ProxyObject ファイルの生成

ーザが定義したクラスの名前を別のクラス名に変更し、ProxyObject のクラスを先のユーザが定義したクラス名で再定義する（図 4(b) 参照)。これはその定義が行われたファイル名などにも及ぶため（図5参照)、本来はユーザ定義のクラス定義が読み込まれていたプログラムでは、その代りに ProxyObject のクラス定義が読み込まれる（図6参照)。ProxyObject は本来のオブジェクトのインタフェースと同じインタフェースを備えているため、記述面での破綻は生じない。

このクラス定義の「入れ替え」により、全てのユーザ定義オブジェクトへのアクセスはリモート/ローカルの区別無く、ProxyObject を介して行われることになる（図 3(b) 参照)。そして、この ProxyObject においてリモート/ローカルホストに関する通信の分岐処理を隠蔽することで、分散透明性を実現する。

3.4 メソッドの並行処理化：並行処理メソッド

C++は逐次オブジェクト指向モデルに基づいているため、処理は全て逐次的に処理され、メソッドの並行処理はできない。そこで自動分散型システムにおける処理の並行化を実現するには、C++の記法で許される範囲でメソッド処理の並行化を実現しなければならない。

C++でのメソッドの呼び出しは以下のように戻り値を扱う場合と扱わない場合の二つに大別される。

1. `method(...)`
2. `value = method(...)`

この 2. の記法では、戻り値を待つ間は呼び出し側の処理が停止するため、メソッドの並行処理を記述できない。よって、1の記法でのみ呼び出される

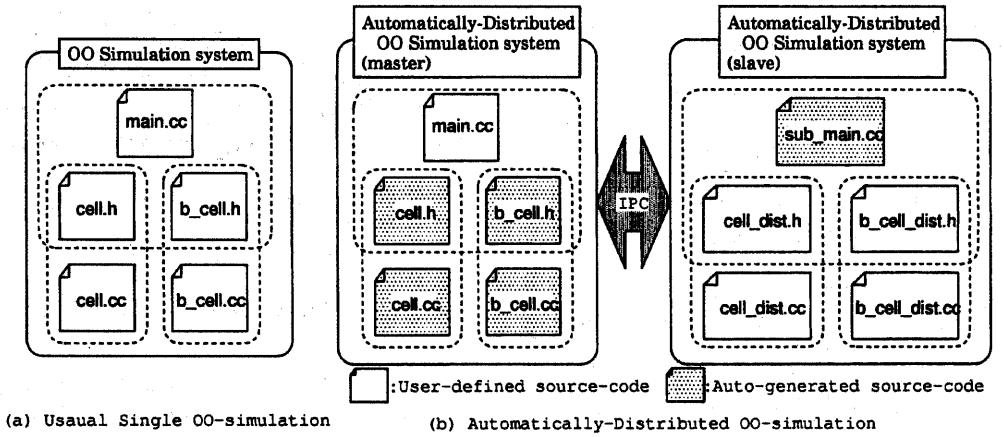

図 6: 自動生成システムによるシステム構成

メソッドが並行動作し得るメソッドとして扱える。C++では戻り値を持たない関数がこれに該当する。

戻り値を返さない関数を並行処理されるメソッドとするため、関数自体の戻り値は元々存在しないが、その処理自体が終了したかどうかを示すための戻り値を処理する必要はある。この処理は自動分散型システム内部で処理するものとする。

本研究では、ユーザが定義したクラス定義を解析して前述の並行処理化の条件に合うメソッドがある場合には、それを並行処理メソッドの候補とする。ただし実際にそのメソッドを並行処理すべきかについては、システムの処理内容に大きく依存するため、ユーザに最終的な決定権を持たせるものとする。

3.5 同期制御

数値計算システムではシステム全体における計算値の同期性が非常に重要である。

本研究では以下のような三種類の同期制御の方法を用いて自動分散型システムの同期制御を行う。また、3番目の同期制御のために、並行処理されるメソッドに対して**分類コード**（非負の整数）を全ての並行処理メソッドに付与する。この値はユーザが最終的に決定するものとする。

1. **各 ProxyObject での同期**
   ある ProxyObject を介して並行処理メソッドが実行中の場合、その ProxyObject を介するメッセージ通信は先の並行処理メソッドが終了するまで遅延する。

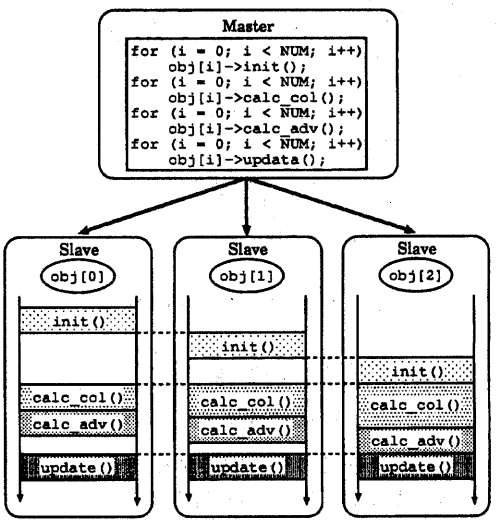

図 7: 本研究での三種類の同期制御方法

2. **並行処理メソッド実行時の同期**
   (一つ以上の) 並行処理メソッドが実行されている場合、全ての ProxyObject を介した逐次処理メソッドの実行は、並行処理メソッドが全て終了するまで遅延する。
3. **並行処理メソッドの分類コードによる同期**
   分類コード値が同じメソッドは並行に処理される。逆に分類コード値が異なる場合には同時には実行されない。

この三種類の同期制御の例を図7に示す。この例で

は、メソッド init() を逐次処理メソッド、calc_col() および calc_adv(), update() を並行処理メソッドとしている。なお、calc_col() および calc_adv() には同一の分類コードが与えられているが、update() は異なる分類コードが与えられている。そのため、calc_col() および calc_adv() は並行処理され得るが、update() は分類コードの異なる上記の二つのメソッドの処理が全て終了するまで、その実行が遅延される。

4 自動生成システムの設計

本章では前章で述べた自動分散型オブジェクト指向シミュレーション・システムのソースコードを自動生成するシステムについて述べる。

本研究ではユーザが C++で記述した逐次型システムのソースコードから前述の自動分散型システムのソースコードを生成することを行う。そのため、自動生成システムはユーザが記述したソースコードの解析部分と ProxyObject の定義生成を含めた自動分散型システムのソースコード生成部分の二つから構成される。

4.1 ユーザ記述ソースコードの解析

本研究では 3.2節で述べたように ProxyObject を中心として自動分散型システムを構成するため、ソースコードの解析はユーザが記述したしたクラス定義の解析が主になる。

この定義解析ではクラス名やその継承関係、外部に公開されるメソッド定義を中心に解析を行う。

4.2 自動分散型システムのソースコード生成

自動分散型システムのソースコード生成では、前節で述べた定義解析により得られたユーザ定義クラスの情報から、その ProxyObject クラスのソースコード及び自動分散型システムのスレイブにおいて必要なメッセージ処理部分などのソースコードの生成を行う。

ProxyObject クラスの定義では、その継承関係は元のクラス定義におけるクラスの継承関係と同様に定義される。また、メソッドの仮想関数指定などもこれに準ずる。

4.3 基底クラス

本研究では、ユーザが定義する全てのクラスの上位クラスとして DObject クラスを定義する。本研究で構築する自動生成システムを利用する場合には、このクラスを継承してクラス定義しなければならない。このクラスでは、分散処理に必要な要素が記述・隠蔽されている。しかし、これらの記述は自動生成されたソースコードにおいて利用され、ユーザが逐次型 OO シミュレーション・システムを記述する上では殆んど意識する必要がない。

5 実装

自動生成システムを Solaris 2.5 上において試作した。自動分散型システムにおける並行処理はスレッドにより実現し、異機種間分散環境での通信はソケットと XDR ライブラリの併用により実現した。

また今回の試作では解析する逐次型システムのソースコードに幾つかの制約を設けている。これは主に解析部分の実装の簡略化と、分散環境上でのほぼ実現不可能な機能を制限するためである。具体的な制約項目としては多重継承や大域変数の禁止があげられる。

6 関連研究

分散オブジェクト指向技術としての関連研究としては、CORBA[4] や HORB[5] があげられる。これらは本研究で用いている ProxyObject と同じ、あるいは等価な概念によって分散透過性を実現している。しかし本研究で行っているような分散透明性については実現していない。

CORBA 関連の研究として C++で記述したオブジェクトを分散オブジェクトへ変換するような研究例などがあるが、最終的にはユーザがそれらを用いてプログラミングするため、本研究で実現しているような分散透明性は実現できていない。

7 考察

本研究の目的は、ユーザに分散処理記述の負担をかけずに分散型 OO シミュレーション・システムを構築することである。この目的は 3.2節で述べたユーザ定義クラスの ProxyObject の生成および本

来のクラス定義との入れ替え、必要な関数などの自動生成による分散透過性および分散透明性の実現、分散処理を自動化したソースコードの生成の実現により達せられたと判断する。

また分散型システムにおける並行処理を実現するために、メソッド定義を解析することによるメソッドの並行処理化を実現した。この並行処理化ではユーザとの対話的なメカニズムを提供して、ユーザが分散/並行処理の記述をせずに分散型システムの挙動を制御できるようにしている。

また同期制御については3.5節で述べたように三種類の同期制御メカニズムを実現して問題の解決を図っている。実際のOOシミュレーションのシステム構成および計算手順などから考えて、この三種類のメカニズムによる同期制御は必要十分と思われる。またデッドロックについても、自動生成されるソースコードがデッドロックを回避する記述を含んでいるため、この問題も解決したものと判断する。

本研究の実装では実装の簡略化を計るため、解析する逐次型システムのソースコードに幾つかの制約を課している。これらの制約が本研究の対象である数値計算システムを構築する上で直接障害となる可能性は低いと判断するが、プログラミングの自由度などの観点から実装をより本格化して制約を緩和することを考えている。ただし、これらの制約には分散型システムでは実現困難な機能（大域変数の使用など）の制限も含まれているが、これらに関しては今後の研究が必要である。

本研究の関連研究のサーベイも行ったが、直接的な関連研究の事例は見いだせなかった。これは自動分散化処理の困難さを示すものと考える。本研究では離散的計算単位としてのオブジェクトとオブジェクト間メッセージ通信という分散化に適したオブジェクト指向パラダイムの特異性に着目して、OOシミュレーション・システムの自動分散化とその自動生成を実現した。

8 結論と今後の展望

本研究では、OOシミュレーションシステムを対象とした、自動分散型オブジェクト指向シミュレーションの自動生成システムを提案し、その試作を行った。これにより、ユーザが分散処理の記述を行わずに分散型OOシミュレーション・システムが構築できることになった。

現在実装されている自動生成システムは試作システムではあるが、自動分散型システムのソースコード生成は十分可能である。

今後は実際に自動分散型OOシミュレーション・システムを生成し、その性能評価などを行い、より効率的なソースコードの生成を図る予定である。

参考文献

[1] Hatakeyama, M., Kaneko, I., and Uehara, H. DSMC Analyses for Highly Complicated and Interactive Flow Based on the Object-Based Mechanism and GUI Environments. In *Rarefied Gas Dynamics 19*, pp. 1175–1181, 1994.

[2] Beguelin, A., et al. A User's Guide to PVM: Parallel Virtual Machine. Technical Report TM-11826, Oak Ridge National Laboratory, 1991.

[3] 畠山正行, 進藤裕希, 宍戸実. 離散ボルツマン方程式のネットワークDVO解法. 第10回数値流体力学シンポジウム講演論文集, pp. 278–279. 日本数値流体力学会, 1996.

[4] 小野沢博文. 分散オブジェクト指向技術CORBA. ソフトウェアリサーチセンター, 1996.

[5] Hirano, S. HORB: Distributed Execution of Java Programs. In *WWCA'97 - Worldwide Computing and Its Applications*, pp. 29–42, 1997. URL: http://ring.etl.go.jp/hirano/horb_wwca97.ps.