有限要素プログラム開発へのオブジェクト指向設計の導入と Java 言語による実装

佐藤匡史
法政大学大学院工学研究科建設工学専攻修士課程

草深　守人[1)]　　竹内　則雄[2)]　　武田　洋[3)]
1)、2）法政大学工学部土木工学科　3)法政大学工学部システム制御工学科

本研究はオブジェクト指向プログラムの設計、実装、有効性について考察することを目的としている。有限要素解析を Unified Modeling Language を用いて設計を行い、オブジェクト指向言語 Java を用いて実装した。そのプログラムを手続き指向言語 FORTRAN により実装されたプログラムと比較した。その結果 Java により実装されたプログラムは FORTRAN で実装されたプログラムとほぼ同等の性能であった。このことから、有限要素解析のオブジェクト指向プログラムは技術計算の分野においても十分に有効である。

１． はじめに

現在、ハードウェアの進化や高級言語の使用により、いっそう複雑な問題にコンピュータが応用されている。その中で、技術計算分野では、オブジェクト指向プログラミングが応用されている例がいくつか示されている。しかし、その設計段階、実装、有効性を具体的に示している例は少ない。従って本研究では、技術計算へのオブジェクト指向設計、実装、有効性を明確にすることを目的としている。具体例として、有限要素解析の UML による設計を行った。Java による実装後、手続き指向言語である FORTRAN で実装されたプログラムとの比較を行った。

２． 有限要素解析のオブジェクト指向設計

オブジェクト指向プログラミングを有限要素解析に応用した例は幾つか存在するが、その目的が曖昧なものが多い。本研究では、オブジェクト指向プログラミングの骨格である「カプセル化」、「抽象データ型」、「クラス」、「継承」の利点を活かしたプログラムの開発を目的としている。

２．１　設計の手段

オブジェクト指向設計では、対象とするシステムの分析が十分に行なわれたとしても、その結果を具体的な設計として誰にでも解りやすい標準化された実体として表現できなければならない。本研究では、このような問題を解決するために UML を利用して**図 1** の手順で設計を行なった。UML はオブジェクト指向設計を目で見て解りやすい実体として表現するためのビジュアル言語である。

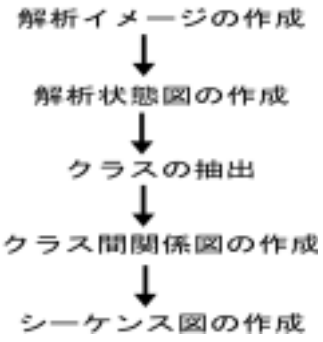

図 1　UML による設計手順

２．２　解析イメージ図

有限要素解析のイメージを図化すると**図 2** のようになる。中央の楕円体の部分は有限要素解析の計算を行なう部分であるが、その詳細は外からは全くわからない。

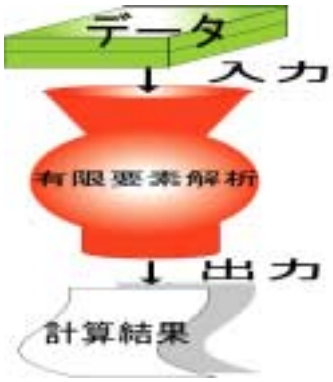

図 2　オブジェクト指向による有限要素解析

２．３　解析状態図

図3は**図2**の球体の内部で行なわれる作業を表している。この図は、有限要素解析をいくつかの主要な作業ごとに分割し、その分割された部分が互いに通信をしながら作業が進む様子を表現したものである。図中の四角で囲まれている部分が分割された作業の内容であり、矢印は通信の内容を示している。

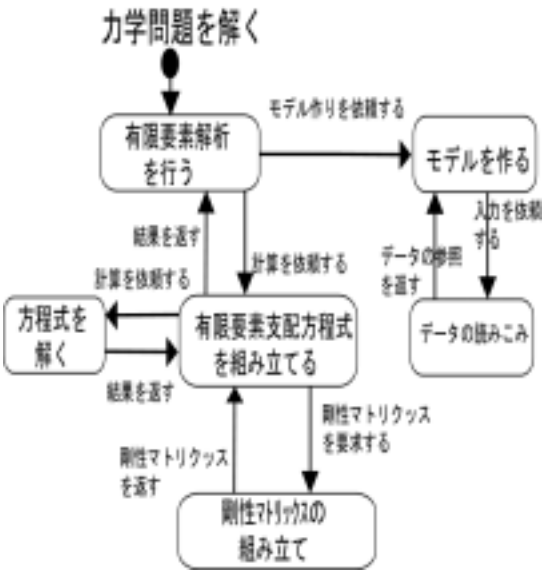

図3　有限要素解析内部の状態

２．４　クラス

図 3 の四角で囲まれている部分をクラスとして抽出を行った。その際、クラス名はクラスの担当する作業を明確に表現できるものを選んでいる。**図3**内の右上に描かれた「モデルを作る」部分を例として**図4**に示す。図はUMLに基づいてクラスを表現したもので、図の上段部分にクラス名、中段部分にこのクラスから生成されるインスタンスの持つ変数、すなわちインスタンス変数、下段部分にはこのクラスの振る舞いを決定するメソッドを記述している。

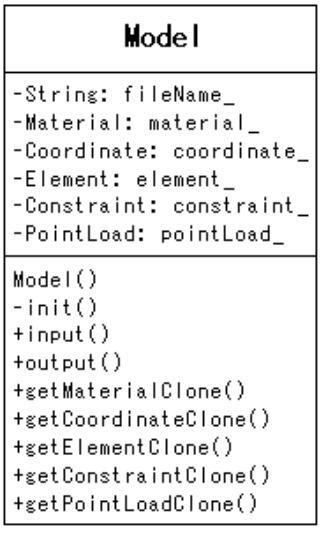

図4　Model クラス

２．５　クラス間の関係図

（１）　全体クラス間関係図

図3から抽出した個々のクラスの間にある関係を見やすい形で表現したのが**図5**に示すクラス間関係図である。この図では、主要なクラスのクラス間の合成関係や、継承関係などをUMLを用いて表現している。

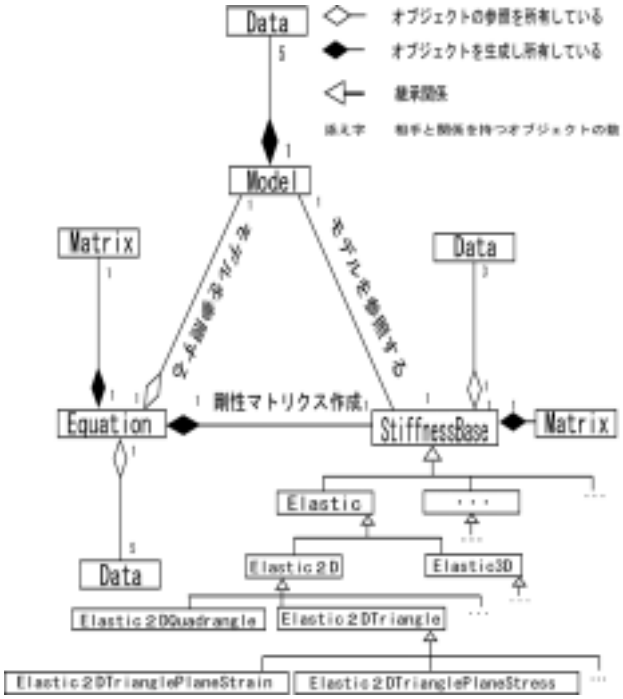

図5　主要クラス間の関係図

（２）　Model クラス周辺関係図詳細

図5に示した全体クラス間関係図内のModel クラス周辺関係図を**図6**に示す。ここでは、Model クラスが他のクラスとどのような関係にあるかを表現している。中でも**図 4** には表現されていなかった、Data クラスとの関係がより詳しく表現され、個々のクラスから生成されるオブジェクトの名前を記述し、その関係にどのような特徴があるかがコメントにより記述してある。

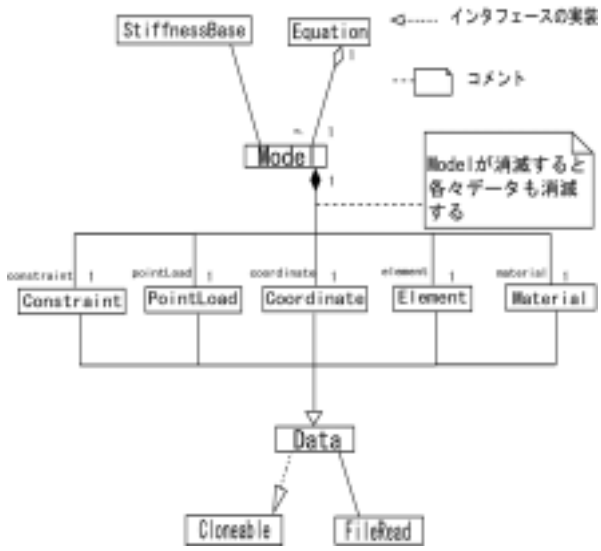

図6　クラス間関係の詳細図（Model クラス周辺）

２．６　時間推移に伴うオブジェクトの動き

個々のクラス、またそのクラス間の関係を設計した後に、実際に生成されるオブジェクトの動きを表現する必要がある。シーケンス図は、このようなクラスから生成されるオブジェクトの時間推移に伴う動きを表現するためのものである。シーケンス図では、縦方向に時間をとり、図下方に向かって処理が進む。また横方向にはオブジェクト間の通信が表現される。**図 5** のクラスから生成されるオブジェクトの動きを**図7**に示す。図中では、処理の流れをよりわかり易くするために作業内容として左の端にその時点での作業を記述している。また、図中の四角の帯はそのオブジェクトが実際に作業を行っている状態を表し、この帯が途切れる場合にはオブジェクトは存在しているが作業を休止している状態を表現している。

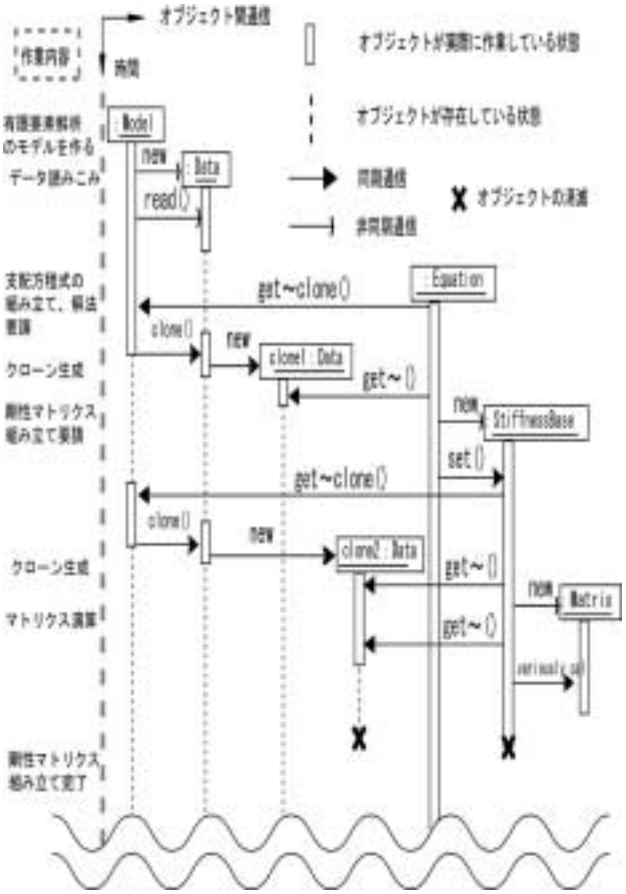

図7　シーケンス図

３． Java による実装

UML により設計された有限要素解析はオブジェクト指向言語である Java を用いて実装した。その具体的な例として、Model クラスの Java による実装を以下に示す。

```
public class Model{

   private String fileName_;

   private Material material_;
   private Coordinate coordinate_;
   private Element element_;
   private Constraint constraint_;
   private PointLoad pointLoad_;

   Model(String fileName){
      fileName_ = fileName;
      init();
   }


   private void init(){
      material_   = new Material(fileName_, "material");
      coordinate_ = new Coordinate(fileName_, "coordinate");
      element_    = new Element(fileName_, "element");
      constraint_ = new Constraint(fileName_, "constraint");
      pointLoad_  = new PointLoad(fileName_, "pointload");
   }

   public void input(){

      Data[] data =  new Data[5];

      data[0] = material_;
      data[1] = coordinate_;
      data[2] = element_;
      data[3] = constraint_;
      data[4] = pointLoad_;

      for(int i = 0; i < date. length; i++){
         data[i]。read();
      }
   }

   public void output(){

      Data[] data =  new Data[5];

      data[0] = material_;
      data[1] = coordinate_;
      data[2] = element_;
      data[3] = constraint_;
      data[4] = pointLoad_;

      for(int i = 0; i < data. length; i++){
         data[i]. write();
      }
   }

   public Material getMaterialClone(){
      return (Material)material_. clone();
   }

   public Coordinate getCoordinateClone(){
      return (Coordinate)coordinate_. clone();
   }

   public Element getElementClone(){
      return (Element)element_. clone();
   }

   public Constraint getConstraintClone(){
      return (Constraint)constraint_. clone();
   }
```

```
    public PointLoad getPointLoadClone(){
        return (PointLoad)pointLoad_.clone();
    }

}
```

4．実行時間の比較

4．1 Just In Time Compiler

FORTRAN では、ソースコードはコンパイラによりバイナリコードに変換され、そのバイナリコードを配布、実行する。このバイナリコードは、各プラットフォームごとに最適化され、違うプラットフォームでこれを利用しようとしても実行はできず、このことからこのような言語はプラットフォームに依存していると言える。しかし、Java ではソースコードはコンパイラにより中間コードにコンパイルされる。この中間コードはバイトコードと呼ばれ、インタプリタである Java Virtual Machine（JVM）で実行されるコードであり、この中間コードを利用することで、JVM が構築されている環境ならば、どこでも実行が可能であり、このことからプラットフォームには依存していないと言える。このような利点がある反面、JVM がインタプリタであるため、FORTRAN に比べ明らかに実行速度が遅くなるという問題もある。この問題を解決するために、本研究では Just In Time Compiler（通称 JIT コンパイラ）を用いている。JIT コンパイラでは、Java の中間コードをあるまとまりごとにネイティブコードに変換しながら実行が行われる。インタプリタとは異り、実行とコンパイルを同時に行うという仕組みにより、Java の実行速度を高速化している。

4．2 実行環境

プログラムを実行した環境を**図 8** に示す。なお、基本的な数値計算、有限要素解析は同じ環境で計測を行った。

Javaの実行環境

OS		Windows 98
CPU		Celeron 433MHz
メモリ		128 MB
使用環境		JDK1.2
実行コマンド	JIT on	java
	JIT off	java -Djava.compiler=NONE

FORTRANの実行環境

OS	Windows 98
CPU	Celeron 433MHz
メモリ	128 MB
使用環境	MSFortran PowerStation4.0

図 8 計測したプログラムの実行環境

4．3 基本的な数値計算の計測

本研究では、オブジェクト指向プログラミングを実現するために Java を用いている。そこで、Java の基本的な数値計算（内積計算）における性能を FORTRAN との比較により示す。なお、計測は以下に示すプログラムの計算に入る直前と計算終了直後の時刻の差を取ることにより行った。

［FORTRAN］

```
do  i  =  1, 100
   itemp  =  0
   do  j  =  1, 3000000
      a(j) = j
      b(j) = j
      itemp  =  itemp  +  a(j) * b(j)
   end do
end do
```

［Java］

```
for(int  i  =  0;  i  <  100;  i++){
   temp  =  0;
   for(int  j  =  0;  j  <  3000000 ;  j++){
      a[j]  =  j;
      b[j]  =  j;
      temp  +=  a[j] * b[j];
   }
}
```

4．4 有限要素解析実行時間の計測

UML を用いて、有限要素解析をオブジェクト指向設計し、Java により実装したプログラムの有効性を評価するために、これまで我々が行った様々な解析において、最も使用頻度が多かった FORTRAN により実装した有限要素解析プログラムを用いて実行速度の比較を行なった。その際、増分荷重を 1000 増分としているため、実際に有限要素解析を 1000 回繰り返しているのと同意である。解析に利用したモデルを**図 9** に示す。

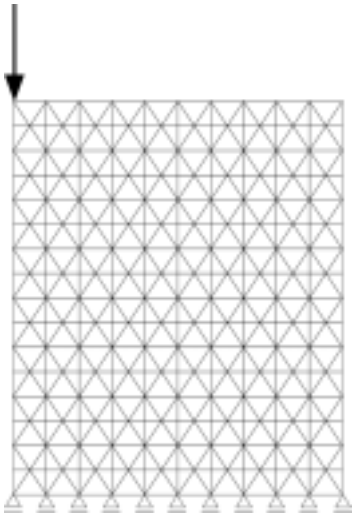

図 9 有限要素解析に利用したモデル（480 要素）

5．結果と考察

5．1 計測結果

（1）基本的な数値計算の計測結果

基本的な数値計算を FORTRAN、Java（JIT on）、Java（JIT off）で行い、その実行時間の比較を**図 10** に示す。縦軸は実行時間、横軸は内積計算に利用する配列の要素数を表している。

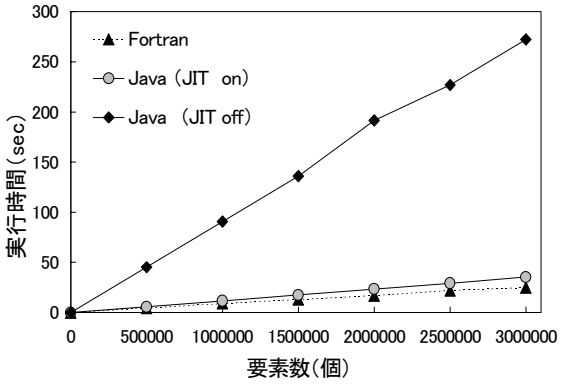

図 10　内積計算実行時間の比較

（2）有限要素解析の実行時間の比較

Java によって実装された有限要素解析プログラムと FORTRAN によって実装された有限要素解析プログラムの実行時間の比較は、いくつかの異なる条件でで行った。1 つ目に、我々が頻繁に使用しているによる FORTRAN による BANDMATRIX 法を用いた有限要素解析プログム、2 つ目に FORTRAN による SKYLINE 法を用いた有限要素解析プログラム、3 つ目に Java（JIT　off）により SKYLINE 法を用いた有限要素プログラム、4 つ目に Java(JIT　on)による SKYLINE 法を用いた有限要素プログラムのそれぞれを計測した。**図 11** にその比較を示す。

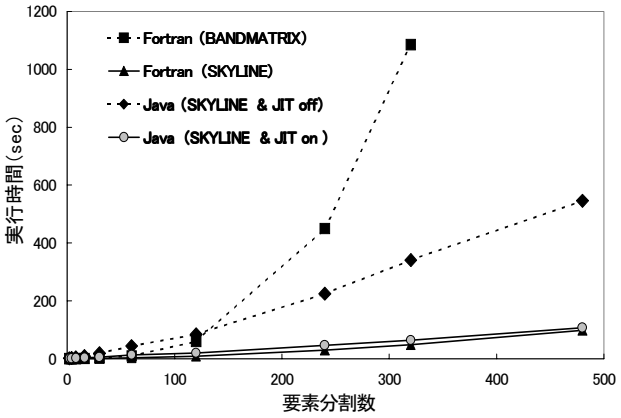

図 11　有限要素解析実行時間の比較

5．2　考察

（1）基本的な数値計算の考察

図 10 から内積計算では、Java の JIT　off の場合 FORTRAN のおよそ 1/11 の性能である。しかし、JIT　on の場合は FORTRAN のおよそ 3/4 の性能を発揮し、JIT　off に比べ処理速度がおよそ 8 倍に向上している。このようなことから、基本的な数値計算では、Java の開発環境に用意された技術により十分な性能が得られることがわかる。また本研究では、JDK1. 2 の JVM 上のみでの計測であったが、現在では多くの JVM が開発され、さまざまな最適化や、その他の高速化の技術が用意されている。そのため、今後は本研究の計測を上回る性能が十分に期待できる。

（2）　有限要素解析の考察

有限要素解析プログラムの実行時間の比較では、頻繁に利用していた BANDMATRIX 法を用いた有限要素解析を FORTRAN で実装したプログラムでは要素分割数が少ない場合には、Java の JIT　off に勝る速さであったが、要素分割数が増えるに従って急激に速度が低下し、100 要素を超えると実行速度が Java の JIT　off よりも遅くなり、その後両者の差は拡大している。このことから、処理速度の速い言語を用いたとしても、その計算手法の選択によってその性能を著しく低下させてしまうことがわかる。SKYLINE 法を用いたプログラムでは、Java における JIT　on の実行速度が JIT　off の実行速度のおよそ 5 倍に向上している。このことから、Java により実装されたプログラムを実行する際には、基本的な数値計算だけでなく、多数のインスタンスの生成を行うオブジェクト指向プログラムでも JIT が有効であることがわかる。また、FORTRAN との比較でも、平均的には FORTRAN の 7 割程度の実行速度であるが、最終的に実行速度は 9 割程になった。このような結果は、Java で実装した有限要素解析プログラムでは、要素数が少ない場合には全体の計算時間に対して、インスタンスの生成に伴う生成時間の割合が大きく、その結果としてインスタンスの生成を行わない FORTRAN よりも実行時間がかかると考えられる。実際に要素分割数が 100 以下の場合には、Java プログラムでは FORTRAN プログラムの 5 倍以上の実行時間がかかっている。

6．結論

基本的な数値計算を扱う場合には FORTRAN が有効であることは疑う余地がない。しかし、現実には技術計算において、このような基本的な数値計算だけを扱うことはほとんどない。また、有限要素解析においては、我々が頻繁に使用している FORTRAN で実装された BANDMATRIX 法を用いたプログラムは、オブジェクト指向で設計し、Java で実装したプログラムの JIT　off よりも遥かに遅いという予想外の結果となった。さらに SKYLINE 法を用いたプログラムによる比較でも、JIT　on であれば、オブジェクト指向によるプログラムであってもほぼ FORTRAN と同等の実行速度が得られることがわかった。このことから、オブジェクト指向プログラムの弱点とされていた実行速度の問題は、例えば数値計算手法を適切に選定するなどのごく単純な対策でほとんど解決できることを示唆しているものと思われる。さらに Java の開発環境を上手く利用することで、今後、技術計算へのオブジェクト指向導入はその多大な恩恵を十分に受けることが出来ると考える。

参考文献

[1]　浅海　智晴：Java Super Tips、 pp28-79、 2000
[2]　三木、 中井：計算工学におけるオブジェクト指向とエージェント指向、 計算工学 Vol. 1、 No. 2、 pp. 5-13、 1996
[3]　関東、 三村、 赤星：FEM におけるオブジェクト指向の応用、計算工学 Vol. 1、 No. 2、 pp15-20
[4]　三木、 杉山、 内田：オブジェクト指向によるはりの変形解析、 日本機会学会論文集 A 編、 Vol. 57、No541、 pp. 2154-2159、 1991
[5]　三木、 杉山、 内田：オブジェクト指向によるトラスの構造解析、 日本機会学会論文集 A 編、 Vol. 57、No541、 pp. 2160-2165、 1991
[6]　轟、 渡辺、 小林、 中村：複合材料積層構造剛性のオブジェクト指向有限要素解析手法を用いた最適化、 日本機械学会論文集A編、 Vol. 60、No571、 pp860-865、 1994
[7]　ハーベイ･M･ダイテル、 ポール･J･ダイテル、 小島　隆一（訳）：Java プログラミング Vol. 1 Vol. 2、 Prentice Hall Japan
[8]　矢川　元基、 関東　康祐：オブジェクト指向計算力学入門、 pp78-134、 1999
[9]　http://www . knt . mech . tut . ac . jp/research/fmm/ohp/19990927/java/

キーワード.
オブジェクト指向,Unified Modeling Language, Java, FORTRAN, 有限要素解析

……

Summary.

Introducing Object Oriented Design to Finite Element Method Program and implementing in Java

Masashi SATO
Department of Civil Engineering, Faculty of Engineering, HOSEI University

Morito KUSABUKA Norio TAKEUCHI
Department of Civil Engineering, Faculty of Engineering, HOSEI University

Hiroshi TAKEDA
Department of System Control Engineering, Faculty of Engineering, HOSEI University

This study aims at discussing design, implementation, and usefulness in the Object Oriented Program. The Finite Element Method Program was designed by The Unified Modeling Language and was implemented in Java. The program was compared with another program implemented in FORTRAN. As a result, The program had nearly equality performance. The Finite Element Method Program by Object Oriented has enough usefulness in technological computation.

Key words.
Object Oriented, Unified Modeling Language, Java, FORTRAN, FEM