

- [34] [http://www.nersc.gov/doc/Comp\\_Research/CCSE/highlight.html](http://www.nersc.gov/doc/Comp_Research/CCSE/highlight.html)
- [35] <http://www.roguewave.com/products/products.htm>
- [36] [http://source.asset.com/WSRD/abstracts/ABSTRACT\\_699.html](http://source.asset.com/WSRD/abstracts/ABSTRACT_699.html)
- [37] <http://www.unikarlsruhe.de/~ae64/cxsc/cxsc.html>
- [38] <http://www.ohriki.t.u-tokyo.ac.jp/~nanakorn/feapoop.html>
- [39] <http://cwis.auc.dk/phd/fulltext/hededa/html/rep.html>
- [40] 平尾隆行、関係データベースシステム、近代科学社、(1986)
- [41] <http://www.ontos.com/ontos-db.html>
- [42] <http://www.objectivity.com/aboutodb/aboutodb.html>
- [43] [http://www.odi.com/prodinfo/os/object\\_store.html](http://www.odi.com/prodinfo/os/object_store.html)
- [44] <http://www.gemstone.com/Products/products.htm>
- [45] M.Wooldridge and N.R.Jennings, Intelligent Agents, Springer-Verlag, (1995)
- [46] 木下哲男、菅原研次、エージェント指向コンピューティング、ソフトリサーチセンター、(1995)
- [47] 人工知能学会、特集「エージェントの基礎と応用」、人工知能学会誌、Vol.10、No.5、(1995)
- [48] 中島秀之編、マルチエージェントと協調計算 I ~ IV、日本ソフトウェア科学会、(1992 ~ 1995)
- [49] 中島秀之、松原仁、本位田真一編、協調プログラミング例題集、bit 別冊、共立出版、(1996)
- [50] M.Wooldridge, J.P.Muller and M.Tambe (Eds.), Intelligent Agents II, Springer, (1996)
- [51] R.E. フィルマン、D.P. フリードマン、協調型計算システム、雨宮他訳、マグロウ・ヒル・ブック、(1986)
- [52] L.Gasser and M.N.Huhns, Distributed Artificial Intelligence, Vol.II, Morgan Kaufmann, (1989)
- [53] G.Agha, Actors: A Model of Concurrent Computation in Distributed Systems, MIT Press, (1986)
- [54] R.Engelmore and T.Morgan (Eds.), Blackboard Systems, Addison-Wesley, (1988)
- [55] B.Elbert and B.Martyna, Client/Server Computing, Artech House, (1994)
- [56] 小野沢博文、分散オブジェクト指向技術 CORBA、ソフトリサーチセンター、(1996)
- [57] <http://www.omg.org/>
- [58] <http://www.next.com/>
- [59] <http://www.apple.co.jp/OpenDoc/>
- [60] <http://www.microsoft.com/devonly/>
- [61] <http://java.sun.com/>
- [62] <http://www.genmagic.com/Telescript/index.html>
- [63] <http://www.genmagic.com/Telescript/awt/>
- [64] 中井、藤井、計算システムにおけるエージェント指向の応用、計算工学、Vol.1、No.2、pp.26-30、(1996)

## FEMにおけるオブジェクト指向の応用

関東 康祐  
三村 泰成  
赤星 保浩

### 1 はじめに

一時のオブジェクト指向のブームは、過ぎ去ったようだが、そもそも、オブジェクト指向がFEMに適用されるようになったのは、いつごろからであろうか。最初のオブジェクト指向有限要素法プログラムのひとつとして、Fordeら<sup>[1]</sup>のObject Pascalで書かれたものがある。同時に、Baughら<sup>[2]</sup>の研究が有り、ほどなく、日本でも石田ら<sup>[3]</sup>、関東ら<sup>[4]</sup>の報告がなされている。いずれにしろ、オブジェクト指向が提唱されてから、約10年の月日が経っている。

オブジェクト間のメッセージパッシングで解析を行う先進的な研究<sup>[5,6]</sup>もあるが、初期の研究には、従来の有限要素法のプログラムをオブジェクト指向プログラム言語で書き直しただけのものも多かった。それでは、従来のプログラムとの差別化は十分ではなく、徐々に問題解析、プログラム設計へと興味が移っていったようである。

一方、入力データ作成<sup>[7,9]</sup>や、解析結果の出力<sup>[10]</sup>には早くから有効性が認められ、現在では、オブジェクト

指向による設計・プログラミングが当然のことのように受け入れられている。

以上のことから、FEM解析プログラムへのオブジェクト指向の適用は、現在のところ、そんなにうまくいっていないように思える。それは、一体なぜなのだろうか。

そもそも、オブジェクト指向の基本的考え方は、データとそれを扱う関数の局所化であった。プログラミング手法の発展の歴史は、一貫して局所化への道であったとも言える。サブプログラムによって、われわれは局所変数を得た。構造化プログラミングにより、モジュール化が推進された。そして、オブジェクト指向により、われわれは局所的なデータと関数のセットを手に入れた。われわれの脳には一度に多くのことを考える能力は無いらしい。他との干渉をなるべく排した単純なものの組み合わせとして、プログラミングを考える方向に突き進んでいる。

われわれは、オブジェクト指向によって、環境の状態に従い、刻々と変化する「オブジェクト」を効率良くモ

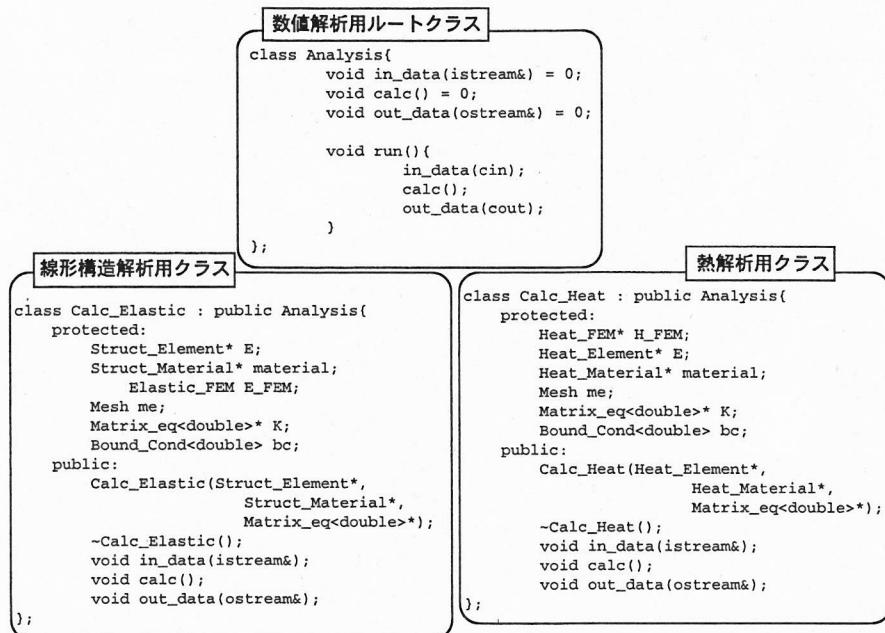


図1 有限要素解析用クラスのソースリスト

モデル化することができるようになった。しかし、FEM解析は、扱いにくい複雑怪奇なオブジェクトではないようである。すっきりときれいに定式化された理論が教科書の中で展開されている。それでは、FEM解析はオブジェクト指向を振り回す必要のないほど、単純な問題なのだろうか。

現実を見れば、そうでもないことがわかる。われわれは膨大な要素モデルが有ることを知っている。さまざまなものモデルが提唱されている。解くべき問題は山のようにある。そんな状況で、われわれは既存のプログラム

をつかって、すぐに答えが出せるだろうか。そうでなければ、必要とするFEM解析プログラムを苦もなく作ることができるだろうか。それは、必要な機能を持っているだろうか。バグは無いのだろうか。これらの質問に明確に答えることができないとしたら、われわれはどうにしたら良いのだろうか。

著者らは、技術計算分野におけるオブジェクト指向アプローチとして、

- ・有限要素法(FEM:Finite Element Method)クラスライブラリの構築<sup>[4]</sup>
- ・OOA/OOD(Object-Oriented Analysis/Design)の適用<sup>[11]</sup>
- ・分散オブジェクトを用いた分散協調環境<sup>[12]</sup>

を研究してきた。ここでは文献[4,11]の研究の一部を例にとり、オブジェクト指向技術の最大の利点のひとつであるソフトウェア資産の蓄積、再利用に焦点を絞った議論を行う。さらに、将来の分散協調環境についての検討も行い、最後にコンピューティング環境の現状分析とその将来について述べたい。

## 2 オブジェクト指向によるFEMプログラムの開発

### 2.1 OOPLによるクラスライブラリ

オブジェクト指向を用いたソフトウェアの再利用の方法で、もっとも典型的なのがOOPLを用いたクラスライブラリの構築である。ここでは、著者らが作成した、FEMクラスライブラリ<sup>[4]</sup>を例にとり、OOPLを使ったプログラム言語レベルでの再利用について述べる。図1に数値解析を行うためのルートクラスであるクラスAnalysisと線形構造解析用のクラスCalc\_Elastic、熱伝導解析用のク

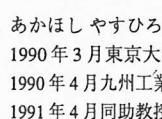
### 筆者紹介



かんとう やすひろ  
1985年3月東京大学大学院原子力工学専攻博士課程修了。1985年4月豊橋技術科学大学技術開発センター助手。1987年2月同工学部助手。1989年5月同講師。1992年4月同助教授。



みむら やすなり  
1993年3月九州工業大学大学院博士課程修了。1996年4月大分大学工学部助手。



あかほし やすひろ  
1990年3月東京大学大学院原子力工学専攻博士課程修了。  
1990年4月九州工業大学工学部講師。  
1991年4月同助教授。

ラス Calc\_Heat のプログラムリストを示す。同図では C++<sup>[13]</sup> による定義のみを載せているが、他の OOP を使用しても、ほぼ同様のクラスライブラリが構築できる。クラス Analysis は抽象クラスであり、クラスインターフェースのみを定義しており Calc\_Elastic、Calc\_Heat は共にクラス Analysis を継承している。図 2 にクラスの使用例を示す。クラス Analysis を継承することで構造解析、熱伝導解析を全く同じインターフェースを通して使うことができる。クラスの供給者と使用者が、「オブジェクト指向」と「クラスの仕様」を共通概念として把握していれば、スムーズに共同作業を行うことができる。従来行われていたソースの書き換えや、サブルーチンなどよりも、安全で自由度の大きい再利用環境を実現できる。技術計算用のクラスライブラリが多数存在すれば、技術計算の専門家でないプログラマでも容易にアプリケーションの一機能として組み込むことも可能となる。

## 2.2 OOA/OOD の適用

ここでは、ソフトウェアの分析／設計の方法論の一つである OMT (Object Modeling Technique)<sup>[14]</sup> を用いて、有限要素法クラスライブラリについてのクラス図を示し、その静的構造のドキュメント化<sup>[11]</sup>について述べる。OMT のクラス図の基本的な表記法を図 3 に示す。

剛性マトリックスに関するデータやアルゴリズムを管理するクラス Element についてのクラス図を図 4 に示し、それに対応する C++ のプログラムリストを図 5 に示す。クラスライブラリが複雑になると、そのプログラム構造を第三者に伝えるのが困難になってくる。OMT などの表記法を共通概念として把握していれば、クラスラ

イブライの構造をスムーズに共同者に伝えることができる。

OOA/OOD は発展段階の技術であるが、ソフトウェア開発の現場では、「顧客ープログラマ」、「プログラマープログラマ」間における情報交換ツールとして導入されつつある。

## 2.3 OOA/OOD/OOP の適用範囲

### (1) 有効な場合

まず OOA/OOD/OOP の適用可能な範囲を考える。

以下の条件下では、クラスライブラリという利用形態はかなりの威力を発揮する。

1) クラスのインターフェースがある程度決定できるとき

2) クラスの約束事を利用者に伝達可能なとき

3) 利用者(開発者)に対して特定のツールを指定できるとき

上記の条件を満たすためには、比較的小規模なグループでの利用、あるいは、特定の分野に絞った資産共有などに適用範囲を限定しなければならないが、プラットフォーム依存のアプリケーション開発や商用アプリケーション開発にはかなりの利用価値がある。また、クラスライブラリと共に「ブラウザ」などのツールの利用ができるれば、クラスの利用方法(インターフェース)を全て覚える必要がなく、開発者の負担を軽減することもできる。OOA/OOD についても同様のことが言える。もともとソフトウェア分析法は、「顧客ープログラマ」間の意志伝達のためのツールとして考えられたものであり、OOA/OOD も限られたグループ内で使用されるのが普通である。したがって、適用についても「約束事を利用者に伝達可能な」範囲に限定される。OOA/OOD は、クラスライブラリ構築のガイドラインを定め、複雑になりがちな

```
main() {
    Struct_Element* E = new Elastic_2D_4nodes;
    Struct_Material* m = new ElasticPlaneStrain;
    Matrix_eq<double>* K = new LDL_sky<double>;
    Calc_Elastic* cal = new Calc_Elastic(E, m, K);
    cal->run();
    delete E;
    delete m;
    delete K;
    delete cal;
}

main() {
    Heat_Element* E = new Heat_2D_4N;
    Heat_Material* m = new Heat_2D;
    Matrix_eq<double>* K = new LDL_sky<double>;
    Calc_Heat* cal = new Calc_Heat(E, m, K);
    cal->run();
    delete E;
    delete m;
    delete K;
    delete cal;
}
```

図 2 有限要素解析法クラスの使用例

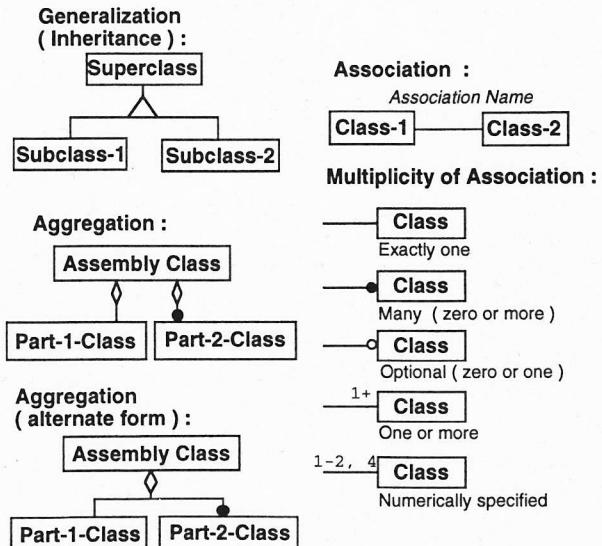


図 3 OMT におけるクラス図の基本表記法

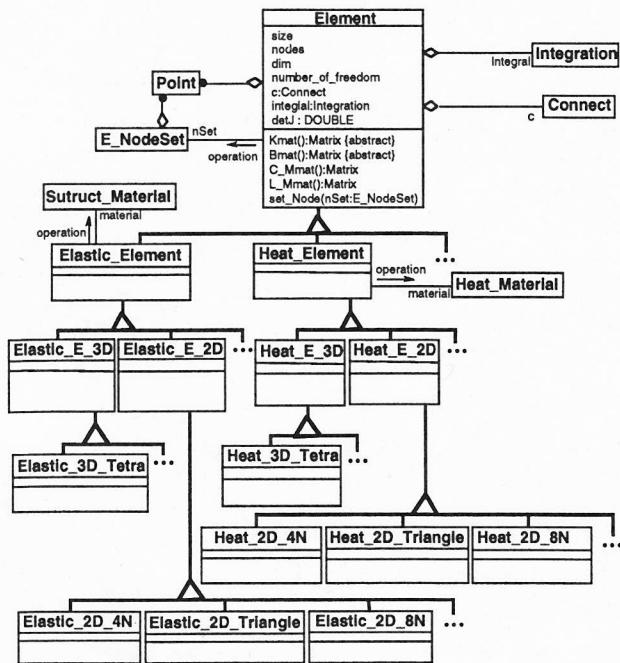


図4 クラス Element のクラス図

設計を整理するためのツールであり、ソフトウェア開発の現場では、ドキュメント化の標準ツールとして使われることは、ほぼ間違いない。

上記のようにOOA/OOD/OOPは、小グループによる共同開発時にもっとも威力を發揮するが、たとえ自分で作ったものであっても、過去にコーディングしたプログラムの内容を把握することは、非常に退屈な作業であり、個人の資産を「クラスライブラリ」という形でまとめておくことも、将来的に非常に有用なことである。また、技術計算分野でのクラスライブラリ利用は十分行われておらず、それゆえ、資産の蓄積も少なく、適用方法さえほとんどの定まっていない。これらの方針性を示すことも、研究者の仕事といえる。

## (2) 有効でない場合

次にOOA/OOD/OOPだけで対応することが不可能であると考えられる場合、すなわち

- 1) クラスのインターフェースを容易に決定できないとき
- 2) クラスの約束事を利用者に容易に伝達できないとき
- 3) 利用者(開発者)に対して特定のツールの使用を強要できないとき

の条件下での資産共有を考える。研究過程でのプログラム開発や技術計算全般という広範囲をカバーするソフトウェア開発などの場合に上記のような状況が起こる。広い範囲をカバーする单一のクラスライブラリの設計が可能かどうかは疑問であるし、研究段階のソフトウェアのインターフェース(rootクラス)を決定するのは容易な

ことではない。このような状況下ではクラスライブラリという利用形態で対応するのは困難である。

結局、広い分野をカバーする単一ソフトウェアを設計することは実質上、不可能と言っても過言ではない。それでも、知的資産を共有したいという要求は当然存在する。次章では、広範囲の分野をカバーするようなソフトウェア共用環境についてのアイデアについて述べる。

## 3 分散／協調環境

### 3.1 実現のための条件

広範囲の分野をカバーするためには、当然のことながら、一つのチームでは対応できず、できるだけ不特定多数の人間が協調できるような環境でなければならない。このような環境の実現のためには以下のことを考慮する必要がある。

- (a) 「再利用」の観点からは、環境を固定すべきであるが、コンピュータ利用の可能性を考えれば、できるだけ多様性を保てるように自由な環境を選択できた方がよい(マルチ・プラットフォーム)。それゆえ、どこに妥協点を持ってくるかが重要である。
- (b) コンピュータは自動プログラミングはできるが、創造はできない。結局、プログラミングの最も創造的なところは、人間がやる以外にない。また、創造は個性そのものである。それゆえ、それぞれの人が作ったもの

```
template <class T>
class Element {
protected:
    int size, //剛性マトリックスのサイズ
        nodes, //1要素の節点数
        dim, //次元数
        num_of_freedom; //自由度
    Array<int> x; //節点番号
    Array<Point> p; //座標
    Connect* c; //クラスConnectのポインタ
    Integration* integral; //クラスIntegrationのポインタ
    double detJ;

public:
    Element();
    virtual ~Element();
    virtual Matrix<T> Kmat() = 0;
    virtual Matrix<double> Bmat() = 0;
    virtual Matrix<double> C_Mmat(); //分布質量マトリックス
    virtual Matrix<double> L_Mmat(); //集中質量マトリックス
    void set_Node(E_NodeSet&);

    int Node_num(int i){ return x[i]; }
    Point get_Point(int i){ return p[i]; }
    Connect* get_Connect(){ return c; }
    Integration* get_Integral(){ return integral; }
    int Dim(){ return dim; }
    int Freedom(){ return nfree; }
    int Nodes(){ return nodes; }
    int Size(){ return size; }
    double DetJ(){ return detJ; }
};
```

図5 Element のソースリスト

を、どのように分散協調させるかが重要である。

(c)あらかじめインターフェースを決定するのは容易ではないので、似て否なるものが複数存在し、自然淘汰的に決定されて行くべき環境が必要とされる。

ソフトウェアの共用環境として、以下のような環境が必要であると考えられる。

#### (1) 知的資産の最小単位（分散オブジェクト）の決定

まず、知的資産を貯蔵するための「単位」と最低限の「約束事」を決定する。開発時にはできるだけ自由な環境を選択でき、再利用の時には、複雑なプログラミングなしで利用できるようなものが望ましい。そこで著者らは、「分散オブジェクト」<sup>[15]</sup>と呼ばれているものを知的資産の最小単位として採用し、計算力学用のプログラム部品<sup>[12]</sup>を構築した。分散オブジェクト自体は、プロセス間通信、ネットワーク通信を伴うようなプログラミングをサポートする技術であり、プログラムの中から、他のプロセスのサービスを呼び出せる。

図6に、NEXTSTEP上のObjective-C<sup>[16]</sup>を使って、乱数を発生する分散オブジェクトの簡単な例を示す。コンピュータ上で実行可能な「エージェント」とみなせ、この「エージェント」をプログラム部品の単位とする。COBRA<sup>[15]</sup>という標準規格案も存在し、将来的にはOS非依存、言語非依存で利用できる可能性がある。現段階でも、FORTRANなどの関数をリンクして使うことは可

能である。

ソフトウェア部品の管理、インターフェースの決定方法などのアイデアは(3)で述べる。

#### (2) 分散オブジェクトの利用環境

(1)で述べたプログラム部品を自由に組み合わせる環境として、図7に示すような「疑似電子ボード」という利用形態を提案する。部品の組み替えはドラッグ&ドロップで行え、部品のインターフェースチェックもボードが行う。将来的には、オブジェクトを簡単に操作できるようなスクリプト言語も必要とされるだろう。分散オブジェクトを操作するスクリプト言語として、X11R6のcontribに含まれている「dish」<sup>[17]</sup>という言語も存在し、技術的には可能になりつつある。

#### (3) 知的資産の貯蔵

ここでは、人間の記憶モデルからヒントを得た、知的資産の貯蔵方法のアイデアについて述べる。人間の記憶モデルの一つである二貯蔵庫記憶モデル<sup>[18]</sup>を図8に示す。このモデルでは、環境から刺激が入力されると、まず刺激に対するデータが短期貯蔵庫に記憶され、長期貯蔵庫のデータを使って比較検討（リハーサル）され、必要であれば長期貯蔵庫に記憶され、不必要であれば、少しの間だけ短期貯蔵庫にとどまり、その後、消去されるというモデルである。記憶されるデータの最小単位はchunk

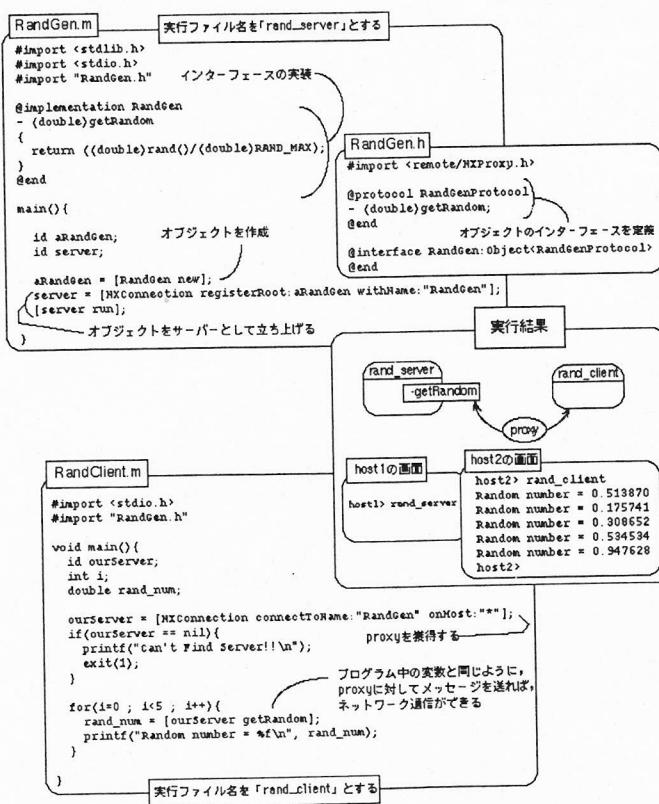


図6 Objective-Cを使った、分散オブジェクトのコーディング例

と呼ばれる何らかの固まりである。このモデルでは、認知過程の解明も chunk の定義もできないという問題を含んでいるが、コンピュータの利用と合致するので、このモデルを参考にすることにした。

図 9 に著者らが提案する知的資産共有環境を示す。Internet 上にデータベースサーバが複数存在し、それらを合わせたものを長期貯蔵庫と考え、ユーザの目の前のローカルなコンピュータを短期貯蔵庫と考える。当然、記憶されるデータの最小単位は、分散オブジェクトである。図 10 にシステムのインターフェースの概念図を示す。データベースサーバは、実際には各地に点在しているが、インターフェース上では、リンク情報のみを扱い、どのマシンからアクセスしようとも、長期貯蔵庫は一つであるかのように見せることができる。他人が作ったオブジェクトをコピーすることも可能であるし、自分の作成したオブジェクトを長期貯蔵庫に記憶させることもできる。これは、不特定多数の人間が、一つの脳を共有し、認知過程（問題認識、プログラミング）については、それぞれの人間が行うことになる。このような環境が存在すれば、R.Dawkins の言うところのミーム（文化の自己複製子）<sup>[19]</sup>のようなものが生まれると考えられる。つまり環境に適応したものが、数多く複製され生き残るのである。オブジェクトが生き残る条件としては、

- ・高機能であること
- ・インターフェースがシンプルであること
- ・機能過多でないこと
- ・ライセンスフリーであること

などが考えられる。また、オブジェクトが進化できる条件としては、

- ・ソースが公開されていること。
- ・分かりやすいコーディングであること。
- ・ソースに関するドキュメントがあること。

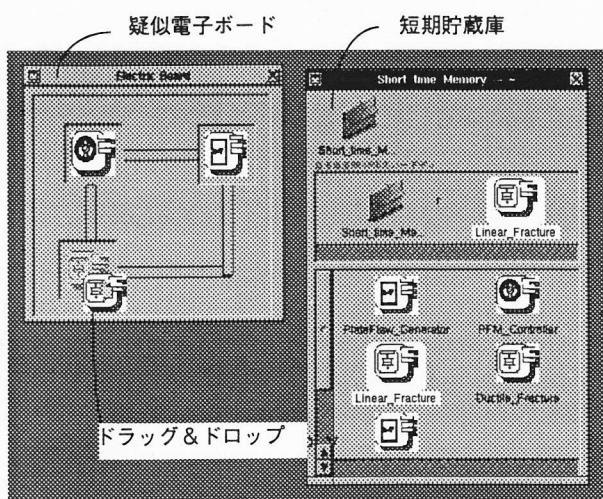


図 7 疑似電子ボードによる利用形態

などが考えられる。特に、ソースが公開されていない場合、いくらかはオブジェクトであったとしても、進化することはほとんどできないだろう。結局、この環境では、問題を認知するのも、知的データを記憶するのも、知的データを進化させるのも、淘汰圧をかけるのも全て人間の仕事であり、コンピュータはそれらの知的活動が発生するためのフィールドを提供する。このような環境を実現できれば、オブジェクトのインターフェース決定についても時間が解決するだろう。

実際には上記のような環境が現在も存在する。Internet 上に接続されているコンピューティング環境そのものである。優れた PDS (たとえば Emacs や TeX など) は、各地のローカルマシンに複製され、同じような過程を経て、進化してきた。ただ、ソースが巨大になると、進化させる当事者となるためには、高度なプログラミング技術が必要とされ、限られた人間しか参加できなかっただけである。進化する複製子の粒度を細かくすることで、多くの人間が進化の過程に参加することができるようになると考えられる。

### 3.2 コンピューティング環境の現状分析

「ソフトウェアの共有、再利用」というのは、結局、ある特定のソフトウェアについての内容（知識）をお互いに交換するための「約束事」を決めることがある。これは、人と人がコミュニケーションをするための約束事を決めるのと同等である。

ゲーデルの証明<sup>[20]</sup>が示唆しているとおり、このような約束事（ルール）を決定する際に、ルール自体が論理的に証明可能かどうかを議論するのは、不毛であるように思われる。現状は、「コンピュータ」というツールで知識を交換するための「基本命題」を模索しているところなのである。コンピュータ上の「共通認識」を構築していくための努力をする方が建設的であると考える。

現在、CORBA<sup>[15]</sup>や Java<sup>[21]</sup>のような、将来、共通認識

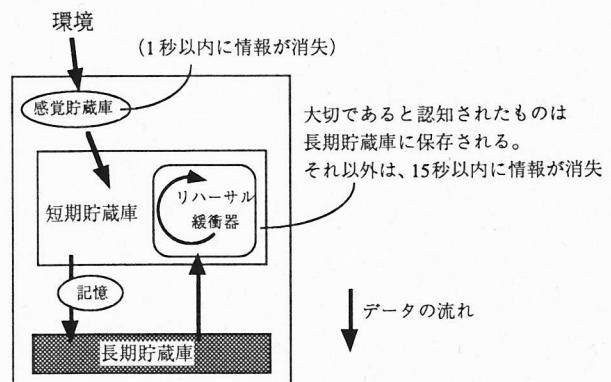


図 8 二貯蔵庫記憶モデル

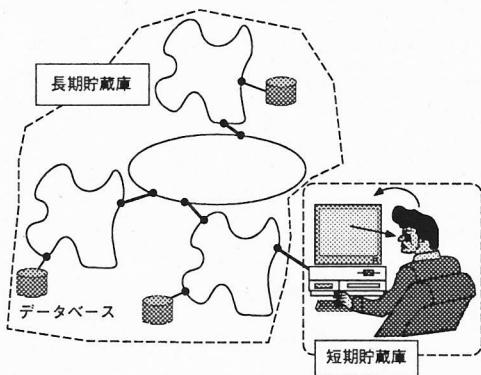


図9 知的資産共有環境

となる可能性のある技術が提案されている。本論文で示したのは、あくまで著者らの考える将来の利用形態であり、もっと多数の利用形態が提案され、環境自体も自然淘汰的に選択されていくべきものである。技術計算分野は、歴史も長く、コンピュータを「使う」ためのノウハウをかなり蓄積しているはずであるし、それらをオープンにする努力をすべきである。われわれは、知識の「受け手」ではなく「送り手」であることを認識すべきである。オブジェクト指向はそのための基本語として機能することだろう。

#### 4 おわりにかえて

恐竜の時代は過ぎ去ろうとしている。恐竜の足元には小さなネズミのような動物がちょろちょろと走り回っている。彼らは群れをなして、生活をしているようだ。前の世代の栄養を十分に吸収した彼らは、新しい環境の中で、どのように分化し、進化していくのだろうか。

注) 「演绎的体系では、証明しえない命題が必ず存在する」ということを証明した。图形問題における「平行」や、Newton力学における「 $f=ma$ 」は証明しえない基本命題であり、これらを元にして、理論が組みあがられる。コンピュータは、論理演算の固まりであり、基本命題を捜すことが重要なのは言うまでもない。プログラミング言語自体が、基本的な予約語と文法ルールを元にして、全てを組みあげている。

#### ■参考文献

- [1] B.W.R.Forde, R.O.Foschi and S.F.Stiemer, Object-Oriented Finite Element Analysis, Comput.& Structures, Vol.34, pp.355-374, (1990)
- [2] J.W.Baugh Jr. and D.R.Rehak, Object-Oriented Design of Finite Element Programs, Comput.Util.Struct.Eng.Proc.at Structures Congress'89, ASCE, pp.91-100, (1989)
- [3] 石田、新美、福和、中井、多賀、オブジェクト指向による構造解析に関する研究：その1 構造解析へのオブジェクト指向の導入、日本建築学会大会学術講演梗概集情報、A、pp.1553-1554、(1992)
- [4] 関東、赤星、三村、青佐、有限要素解析コード開発におけるクラスライブラリの構築、日本機械学会69期全国大会講演論文集、No.910-62A、pp.636-637、(1991)
- [5] 三木、杉山、内田、オブジェクト指向によるトラス構造解析、日本機械学会論文集A編、Vol.57、No.541、pp.2160-2165、(1991)
- [6] 藤、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、No.571、pp.860-866、(1994)
- [7] S.Nagasaki, H.Sakuta and M.Goto, Development of Finite Element Analysis Support System based on the Hybrid Knowledge Model, Computers in Engineering, Proc.Int.Comput.Eng.Exhibit, Vol.2, pp.53-59, (1992)
- [8] J.-F.Ballay, F.Nuwendam and J.-C.Kieny, A Tool using an Object Oriented Language for Field Computation in a CAD Prospect, COMPUMAG IEEE Trans.Magnetics, Vol.28, pp.1774-1777, (1992)
- [9] H.Ohtsubo, Y.Kawamura and A.Kubota, Development of the Object-Oriented Finite Element Modeling System - MODIFY, Eng.with Comput., Vol.9, pp.187-197, (1993)
- [10] Y.Zheng, R.W.Lewis and D.T.Gethin, FEView: an Interactive Visualization Tool for Finite Elements, Finite Elements in Analysis and Design, Vol.19, pp.261-294, (1995)
- [11] 三村、赤星、原田、関東、青佐、熱・構造解析におけるオブジェクト指向アプローチ（設計と実装）、第6回計算力学講演会講演論文集、Vol.930-71、pp.47-48、(1993)
- [12] 三村、赤星、原田、関東、分散オブジェクトを用いた計算力学ソフトウェア共用システムの開発、日本機械学会第8回計算力学講演会講演論文集、No.95-4、pp. 281-282、(1995)
- [13] S.B.Lippman, C++ プライマー、トッパン、(1993)
- [14] J.Rumbaugh, Object-Oriented Modeling and Design, Prentice Hall, (1991)
- [15] Common Object Services Specification, Vol.1, The Object Management Group, (1994)
- [16] 林檎郎、NEXTSTEPパワーガイドブック、pp.260-275、技術評論社、(1994)
- [17] 嶋嶋、Fresco入門、Software Designe、6月号、pp.161-168、(1995)
- [18] G.R.Loftus、E.F.Loftus、人間の記憶 認知心理学入門、pp.10-14、(1980)
- [19] R.Dawkins、利己的な遺伝子、pp.306-321、紀伊国屋書店(1991)
- [20] E.Nagel、J.R.Newman、数学から超数学へ ゲーデルの証明、白揚社、(1968)
- [21] 溝口文雄、大和田勇人、入れたてJava、共立出版、(1996)

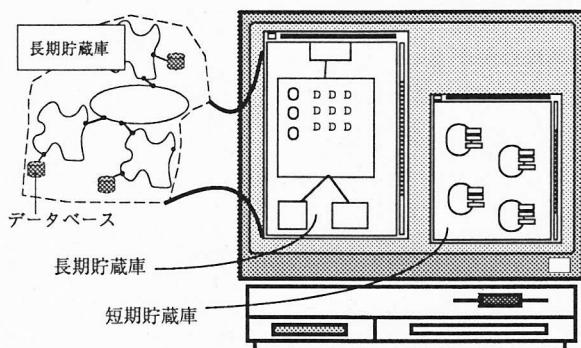


図10 知的資産共有環境のインターフェース