

平成 4 年度九州工業大学修士論文

オブジェクト指向型有限要素解析 システムの開発

有限要素解析プログラムの部品化と プログラミングの方法論



指導教官

原田昭治	教授
荒井貞夫	客員教授
野田尚昭	助教授
秋庭義明	助教授
赤星保浩	助教授

九州工業大学
大学院工学研究科
設計生産工学専攻
博士前期課程 2 年
91341228
三村 泰成

平成 5 年 2 月 26 日提出

目次

第1章 序論	1
第2章 ソフトウェア開発・保守・改良の現状	2
2. 1. ソフトウェアの再利用	··· 2
2. 2. ソフトウェア再利用のための技術	··· 5
2. 3. オブジェクト指向技術	··· 7
2. 4. ソフトウェアの開発環境の現状	··· 15
第3章 技術計算における研究・開発環境	32
3. 1. 近年の技術計算分野の研究・開発における問題点	··· 32
3. 2. 技術計算の将来	··· 33
第4章 オブジェクト指向型有限要素解析システム	37
4. 1. ソフトウェア工学の有限要素解析への適用	··· 37
4. 2. 有限要素解析におけるオブジェクト指向の研究	··· 38
4. 3. 本システムの構成	··· 39
4. 4. 将来の展望	··· 41
第5章 有限要素法クラスライブラリの構築	46
5. 1. 本研究における設計方針	··· 46
5. 2. 有限要素法クラスライブラリの設計	··· 48
5. 3. 考察	··· 55

第6章 有限要素解析システムの構築例	81
6. 1. 热弹性解析	· · · 81
6. 2. 将来の解析プログラム	· · · 82
第7章 結論	85
参考文献	87
謝辞	90

第1章 序論

1968年ドイツのガルミッシュで開催されたNATOの国際会議で「ソフトウェアの危機」が問題になって以来、ソフトウェアの生産技術がソフトウェアエンジニアリングとして工学体系の1つに位置付けられるようになった。^[1] 構造化を用いたシステムの分析／設計、ソフトウェアの部品化（モジュール化）、設計とソフトウェアの再利用などの研究が盛んに行なわれ、最近ではこれらをさらに進めた、オブジェクト指向分析／設計（OOAD, OOA/OOD: Object-Oriented Analisys / Design），オブジェクト指向プログラミング（OOP: Object-Oriented Programming）が行なわれている。また、これらを一貫支援し自動化するCASE（Computer Aided Software Engineering）ツールなどについても事務処理、システム開発管理ツールを中心として研究・開発（R&D）が行なわれている。しかし、技術計算の分野ではソフトウェアの生産性と品質向上の技術に関する研究はほとんど行なわれていないのが実状である。

一般に数値解析では、手続き型言語であるFORTRANによってプログラムを書くのが主流となっている。しかし、FORTRANは単純な数式を機械語に翻訳することを目的として開発された言語であるため、プログラマは解析対象を単純なデータと数式の列へと変換しなければならない。このようにして作成されたプログラムは、プログラムが複雑化すると再利用性、モジュール性を配慮して作成することが難しくなる。一部を変更したいような場合にも、そのソースプログラムの内容を、全て理解しなければならなくなり、変更による影響が広範囲に及ぶためにデバッグに費やす時間／労力が増大する。このようなマンパワーの浪費を生じさせているにもかかわらず、FORTRANが使われる原因是技術計算における歴史が長く資産が多いためである。また、多くの大型計算機がFORTRANコンパイラしかサポートしていないた

め、大規模な計算を行なうためにはFORTRANを使う以外に有効な手段がなかったということも大きな理由となっている。今日までの技術計算の研究は、FORTRANを代表とする手続き型言語を用いたアルゴリズム中心のものが大半を占めている。それゆえ、この分野では大規模なプログラムの開発、保守、改良に費やすマンパワーの問題は、ほとんど放置されたままである。研究者には、プログラム言語、計算手法などの深い知識が要求され、大きな負担となる。このような状況は数値解析法の1つである有限要素法についても同様である。

手続き型言語だけではこれらの問題を回避することは容易ではない。少なくとも研究者がソースコードをデータベースとして参照、記憶、操作、拡張できるようなプログラミング環境が必要となってくる。このような環境を満たすためには、従来の手続き型の考えだけでは不可能であり、オブジェクト指向的な考え方の導入が必要である。オブジェクト指向法はソフトウェア・エンジニアリングの分野では多用されているが、技術計算の分野においてはオブジェクト指向法に関する研究はわずかしか行なわれていない。現在、18歳人口は減少しつつあり、将来若い研究者の人口が減少するのは明らかである。従って、研究者1人1人の負担はより深刻なものと予想され、次世代に研究資産を受け継ぐためにも、拡張性、再利用性に優れたプログラミング環境を開発することは、極めて重要なものと言える。

このような状況を踏まえて、本研究では有限要素法についてのプログラムをオブジェクト指向設計し、オブジェクト指向プログラミング言語(OOPL:Object-Oriented Programming Language)を用いて、その設計に基づいた有限要素法クラスライブラリを構築することによって、それらをデータベースとして、参照、記憶、操作、拡張できるようなオブジェクト指向型有限要素解析システムのプロトタイプを開発した。本論文では、このシステムの中で核となるクラスライブラリに着目し、クラスライブラリのオブジェクト指向設計およびその実装について論じる。以下に本論文の構成を示す。

第2章 ソフトウェアの開発、保守の現状を述べ、オブジェクト指向技術の必要性を論じる。また、技術計算以外の分野でのソフトウェア開発環境がどのような状態にあるかを述べる。

第3章 第2章を踏まえて、技術計算では、どのようにソフトウェアを管理したら良いかを考察する。

第4章 本システムの構成と将来の全体像について述べ、修士論文では、どの部分を論じるかを述べる。

第5章 有限要素法クラスライブラリの設計を行ない、有限要素法に必要と思われるオブジェクトとその構成を明らかにする。さらに、C++によって実装を行ない、数値解析プログラムソースの部品化、再利用性などを考察する。

第6章 有限要素法クラスライブラリを用いた熱弾性応力解析システムを構築し、クラスライブラリを用いたプログラム開発の有用性を示す。

第7章 結論

第2章 ソフトウェア開発・保守・改良の現状

2. 1 ソフトウェアの再利用^[2]

ソフトウェアの保守量は増加の一途をたどり、大きな問題となっている。アメリカにおける1980年代末の調査では、大中規模の企業のデータ処理(DP)部門の人的資源の約8割ぐらいが既存のソフトウェア資産の保守(修正・拡張)に使われているとのことである。このような状況では、アプリケーションの新規開発などの創造的な方面に割り当てる人員を減らさざるを得ない。また、短期間で他の職務に変わってしまう人材も少なくなく、18歳人口の減少なども考慮に入れると、人的資源の枯渇はますます深刻になるだろう。従って、ソフトウェア開発における今後解決せねばならない最大の課題の1つは、(修正・拡張を含む)保守作業の生産性の向上と保守量の軽減である。そして、生産性と品質を上げる(保守量を減らす)最上の方法は新たに作る量を減らすことである。すなわち、試験(test)済み、検証済みのコード(プログラム)の再利用(reuse)である。再利用の度合によっては、開発の生産性を10倍以上にすることも可能だと考えられ、その実現のためには、再利用を可能とするためのプログラミング・パラダイム(paradigm)やそれらをサポートするプログラム言語や開発技法やインターフェース、再利用部品の保管方法および新規需要に対応する部品の検索と修正・組み込み・合成の方法が必要である。

自動車や電気製品の生産などを見れば明らかのように、ソフトウェア以外の産業では、生産性、品質を向上するために、標準化された部品を組み立てることによって製品を製造する。従ってソフトウェアエンジニアリングの分野においても、再利用可能なソフトウェア部品により、ソフトウェアを構築するのは、必然的な流れだと言える。(ソフトウェアの部品化については2. 3節で詳しく述べる)

2. 2 ソフトウェア再利用のための技術^[2]

ソフトウェアの再利用性（reusability）とは、他のソフトウェアの中で使えるようにソフトウェアの部品が作られていることを意味し、再利用（reuse）とは、ソフトウェアの部品をどこかに格納しておいて、必要に応じて取り出して、そのままか多少手を入れて（カスタマイズして）、他の部品と組み合わせて使うことを意味する。

再利用性の一面である抽象化レベルは（要求）仕様、設計、コードの3段階があるとみなせる。（要求）仕様は実行環境とは無関係であり、同一仕様から異なる設計やコードを生じることがある。設計の再利用は、1つの設計を異なる環境用に使用することを意味し、同一設計から異なるコードを生じる。コードの再利用は目に見える利益をもたらす。しかし、それはコーディングの際に決められるので、局所的には適当であるように見えて、全体的に見て目的に合致しない恐れがある。また、変更なしに再利用できるコードの部分（部品）を見出だすことは容易ではない。

再利用部品が活用される度合は、新しい要求を満たすためにカスタマイズできるか否かによっても大きく左右される。その意味でソフトウェア部品は、白箱（white box）と黒箱（black box）の2種類に大別できる。白箱は内部構造を見て手を加えられるものであり、黒箱は明確に定義されたインターフェースを通じてのみアクセス可能でユーザーによって修正できないものである。白箱部品はカスタマイズの自由が大きいが、品質を保持するのが難しい。逆に黒箱部品は変更の自由がない代りに、品質の保持が白箱部品より容易である。

2. 2. 1 コードの再利用の推移（プログラミングの推移）

黒箱部品について見てみると、構造化プログラミングにより、FORTRAN数値計算ライブラリやUNIXのCライブラリのようにかなりの再利用と標準化を達成したも

のある。構造化プログラミングは、手続きだけに着目して階層化・抽象化し、段階的に詳細化することで大規模で複雑なプログラムを正確にする技法であった。これに対して、Adaなどの一部の言語では、手続きとデータをカプセル化して抽象データ型（abstract data type）の概念や情報隠蔽（information hiding）の概念が導入され、より部品化が進められるようになった。さらに、オブジェクト指向プログラミングでは、継承（inheritance）の概念が導入され、部品を新たな場面に順応させ、拡張を容易にし、継承によって黒箱部品のカスタマイズが可能となった。ソフトウェアの安全性と拡張性を両立させるためには、今後、オブジェクト指向プログラミングは不可欠である。

2. 2. 2 分析／設計

1970年代の後半から提案された構造化分析／構造化設計（SA/AD:Structured Analysis / Structured Design）の技法は、モジュールや手続きの整合性を増大させ、再利用性を高めている。データ流れ図（DFD:Data Flow Diagram）や状態遷移図（STD:State Transition Diagram）を使用した設計と、その設計に基づいて実装されたソースコードは、各プロセスやモジュールや手続きの変更や入れ替えを容易にするので、既存の再利用部品をはめ込みやすくしている。さらに再利用性を考慮するために、1980年代の後半以降には、オブジェクト指向分析／設計が注目されるようになった。（オブジェクト指向分析／設計については2. 3. 3で詳しく述べる）

分析／設計の技法は、ソフトウェアが満たすべき機能・構造を抽出するために用いられるとともに、より再利用性に優れたコーディングを行なうために用いられる。さらに、ソースコードでないダイアグラムで表現しておくことによってソースコードの理解を容易にし、開発・保守・改良などに非常に有用である。

2. 3 オブジェクト指向技術

2. 3. 1 オブジェクト指向プログラミング

ソフトウェア工学の分野で、最近常識となりつつある概念であるオブジェクト指向法は、プログラミングについて議論されることが多いが、そもそもとは実世界の物事をオブジェクトを用いてモデル化するための技法である。オブジェクトとは、メッセージを送れば、そのメッセージに対応する機能（メソッド）を実行するなんらかの物（オブジェクト）である。メッセージの解釈、それについての振舞いをオブジェクト自身が知っていて、オブジェクトを使う人（物）は、オブジェクトを一種のブラックボックスとして扱う。ここで例として、レストランの中をオブジェクト指向法でモデル化した場合を考えてみる。Fig.2.1のように客、コックという2種のオブジェクトを見つけることができる。客はウェイトレス（インターフェース）に”注文”と言うメッセージを送るだけで良く、ウェイトレスがどのような行動をとるかまったく気にしなくて良い。ウェイトレスは注文された料理の専門コックにメッセージを送るだけで、コックがどのように料理を作るかについては、知る必要もない。コックはメッセージに対して、それぞれの材料（データ）と料理方法（メソッド）で料理を作る。料理方法はコック（オブジェクト）だけが知っていれば良いのである。このように、それぞれのメソッドを持ったオブジェクトどうしが、メッセージを送りあうことによって、物事が進行するという考え方で実世界をモデル化することができる。^[3]

近年、特にプログラミングでオブジェクト指向法が注目を浴びているのは、オブジェクト指向法を導入するとプログラムを部品化（モジュール化）でき、拡張性、再利用性に優れた高効率なプログラミングが行なえるからである。現在では、OOP-（Object-Oriented Programings）という言葉も一般的になってきた。なぜ、部品化が必要なのかは、一般の工業製品の製造を考えれば容易に理解できるであろう。

例えば Fig.2.2(a), (b)のようなラジオと自動車の組立てを考えてみる。ラジオを組立てる場合、トランジスタや抵抗、ダイオードなどは部品とみなせる。入力に対してどのような出力をするのかさえわかっていれば、部品そのものの中身については、まったく気にすることなくブラックボックスとして扱える。ラジオの組立てにおいてはトランジスタなどの部品がオブジェクトとなる。さらに、自動車の組立てにあたっては、ラジオを部品として考えることができ、オブジェクトとして扱える。これに対して、従来のFORTRANのような手続き型プログラミングでは、このようにはなっていない。Fig.2.3のようにプログラムを外から見た場合、ブラックボックスとみなせるが、プログラム自体はデータとアルゴリズムが混沌としており、とても部品化されているとは言えない。OOPを使えばこれを部品化することができる。Fig.2.4のようにオブジェクト製造機の役目をはたすのがクラスと言うもので、ここから、データとアルゴリズムをパッキングしたオブジェクトを生成する。このオブジェクトを部品として組み立てることによりプログラムを構築することができる。また、部品の再利用、ブラックボックス化などにより、コーディング、デバッグに費やすマンパワーを大幅に低減することもできる。しかも、機能拡張についても、継承を使って、効率的に行なえる。結局、OOPとは、従来の手続き型プログラミングをFig.2.5のようにまとめなおすことによって部品化を行ない、継承によって拡張性を実現したものだと言える。

CPUの速度が向上してくるにしたがって、コンピュータは複雑な作業をこなせるようになった。しかしながら、作業が複雑になればなるほど、プログラムソースは巨大で複雑なものとなり、保守、改良に費やすマンパワーは膨大なものとなってきている。それゆえ、これまでの数値解析コードの開発においては、プログラム言語の文法が規格化されていれば良いと考えられてきたが、これからは、解析コードの部品化、規格化が必要である。このような要求を実現しうるのが、オブジェクト指向法である。

2. 3. 2 オブジェクト指向プログラミング言語

現在ではオブジェクト指向の概念を導入した言語が、多数開発されている。これらのオブジェクト指向プログラミング言語は、既存の言語にオブジェクト指向の概念を付加した「ハイブリッドなオブジェクト指向プログラミング言語」と設計段階からオブジェクト指向法の概念を導入して開発された「純粋なオブジェクト指向プログラミング言語」に大別することができる^[4]。現在では、オブジェクト指向プログラミング言語もコンパイル言語の物が多く、手続き型言語と比べても、計算効率で大きく劣るわけではない。従って保守性などを考えれば、すでにオブジェクト指向プログラミング言語は実用段階に入っていると言って良い。以下にいくつかのオブジェクト指向プログラミング言語を紹介しておく。

(a) ハイブリッドなオブジェクト指向プログラミング言語

< C++ >^[5, 9]

AT&T社の Bjarne Stroustrup によって設計されたもので、C言語にオブジェクト指向法の機能を付加した、強い型付け言語である。C言語は最も広く使われている言語の1つであり、プログラマがオブジェクト指向法に移行する際に比較的カルチャーショックが少ないとこと、主要なコンピュータベンダーがサポートしていること、g++(GNU C++) という無償で入手できるコンパイラが存在することなどから、もっとも使用されているオブジェクト指向プログラミング言語だと言える。

< Objective-C >^[5, 10]

Stepstone社（以前はProductivity Products International 社）のBard J.

Coxによって開発された言語である。ソフトウェアの生産性を上げることを主たる目標にしており、C言語のシンタックスの中に[]付きでメッセージ・センディングの機構を取り入れ、Smalltalkが持つメカニズムのほとんどを実現している。メッセージ式が[]でくくられていることから、C言語の記述とオブジェクト指向の表現を分離することができ、C++などに比べてシンタックスがわかりやすくなっている。Objective-Cではソフトウェア部品のことを「Softwar-IC」と呼び、特に部品化を強調している。

(b) 純粹なオブジェクト指向プログラミング言語。

< Smalltalk >^[5, 11]

XEROXのPARC研究所で開発されたもので、オブジェクト指向プログラミング言語として普及した最初のものであり、その成功が他の多くのオブジェクト指向プログラミング言語を生み出した。Smalltalkは単なる言語ではなく、オペレーティングシステムの機能もある程度含んだ開発環境である。

< Eiffel >^[12]

Bertrand Meyerによって設計された強い型付け言語である。多重継承、パラメータ化クラス（総称クラス）、メモリ管理、表明をサポートしている。カプセル化、アクセス制御、改名などのソフトウェア工学上優れた機能を持っており、技術的な観点からすれば、もっとも優れた商用のオブジェクト指向プログラミング言語だと言える。

以上の他、Object-Pascal, CLOSなどもあり、既存のプログラミング言語はオブジェクト指向法を取り込む方向にある。Table 2.1^[13]にC++, Smalltalk, CLOS, Eiffel, Objective-C の比較を示す。

2. 3. 3 オブジェクト指向分析／設計

プログラミング言語には一長一短があり、オブジェクト指向プログラミング言語を使ったからといって、必ずしも効率的なプログラミングが行なえるわけでは無い。どのように再利用、拡張性を考慮した構成のプログラムコードを作成するかが重要であり、その意味でも、オブジェクト指向分析／設計（OOA/OOD:Object-Oriented Analysis and Design）と併用してのソフトウェア開発が注目されている。OOA/OOD手法には、コード／ヨードン法^[1, 14]（OOA），シュレイア／メラー法^[11]（OOA）などいくつかあるが、ここでは現時点でもっとも完成された技法の1つであるOMT法^[1, 13]（OOAD）を取り上げる。本研究においてもシステムの分析／設計にはOMT法を採用した。

OMT(Object Modeling Technique) 法^[1, 13]

OMTはGeneral Electric社の研究開発センターで開発された手法であり、ほとんどの種類のアプリケーション開発に役立ったと主張されている方法論である。

Rumbaughらはオブジェクト指向開発の方法論とオブジェクト指向概念を表わす図形表記を提供している。この方法論をOMTと呼び、OMT方法論では次のような3つのモデル化表記法が用いられる。

・オブジェクトモデル (object model)

オブジェクトモデルはシステム内のオブジェクトの静的な構造とオブ

ジェクト間の関係を記述するもので、Fig.2.6(a), (b)のようなオブジェクト図（object diagram）によって表わされる。オブジェクト図はオブジェクト、クラス、それらの関係のモデル化のための形式的な表記法を提供する。オブジェクト図では以下のものが表現できる。

- ・ クラス (class), オブジェクト (object)
- ・ 属性 (attribute)
- ・ 操作 (operation)
- ・ 繙承 (inheritance), 況化 (generalization)
- ・ 関連 (association)
- ・ 集約 (aggregation)

- ・ 動的モデル (dynamic model)

動的モデルは時間とともに変化するシステムの状態を記述したもので、システムの制御を記述し、実装するのに利用される。動的モデルは Fig.2.7(a), (b)のような状態図（state diagram）によって表わされる。

- ・ 機能モデル (functional model)

機能モデルはシステム内でのデータ値の変換を記述したものである。

機能モデルはFig.2.8(a), (b)のようなデータフロー図 (DFD:Data Flow Diagram) によって表わされる。

OMTは以下のような手順で行なわれる。

1) 分析

ここではシステムが何をすべきかというモデルを作る。

- ・問題記述の最初の版の作成
- ・オブジェクトモデルの構築：オブジェクトのクラスの認識，データ辞書の作成，クラス間の結合の追加，属性の追加継承の導入，シナリオに基づくテストの繰り返し，クラスをグループ化してモジュールにする，といったことを行なう。
- ・動的モデルの作成：シナリオの作成，オブジェクト毎のイベントの認識とシナリオによるイベントのトレース，状態図の作成などを行なう。
- ・機能モデルの作成：入出力の認識，DFDの作成，各機能の記述などを行なう。
- ・3つのモデルの検証と改良：機能モデルで見つかった主要な操作のオブジェクトモデルへの反映，シナリオによるテストなどを行なう。

2) システムの設計

ここでは，システムの上位レベルの構造を決める。

- ・サブシステムへの分割
- ・問題固有の並行性の認識
- ・サブシステムのプロセッサとタスクへの割り当て
- ・データストアの実装方法の選択
- ・共通リソースの認識とそれらをアクセスする制御機構の決定
- ・ソフトウェア制御の実装方法を選択する：プログラム内に状態を持つか，状態マシンを直接実装するか，並行タスクを使うかを選択する。
- ・境界条件を考える。
- ・トレードオフの優先順位を決める。

3) オブジェクト設計

ここでは、分析モデルを磨き上げ、特定の言語やデータベースへの変換方法に依存しない、実装のための詳細を設計する。

- ・他のモデルで見つけた操作のオブジェクトモデルへの反映：機能モデルのプロセスから操作を見つけるか、動的モデルへのイベント操作を見つける。
- ・操作を実装するためのアルゴリズムの設計：操作のコストを最小にするアルゴリズムを選択する、アルゴリズムにあったデータ構造を選択する、必要であれば新たなクラスと操作を追加する、などを行なう。
- ・データへアクセスするためのパスの最適化：アクセスコストを最小化し便利さを最大化するため冗長な結合を追加する、効率化のため計算順序を変える、複雑な計算をやり直さないため計算結果を保存する、などを行なう。
- ・ソフトウェア制御を行なう。
- ・クラス構造を吟味して、継承を増やす：継承を増やすためのクラスと操作を再配置する、抽象クラスを抽出する、継承が意味的におかしいとき委譲（delegation）を使う、などを行なう。
- ・結合の実装方法を設計する：結合を1方向だけ実装する、結合をオブジェクトとして実装する、結合の両側のクラスへの属性として追加することによって実装する、などを行なう。
- ・オブジェクトの属性の表現法を決める。
- ・クラスと結合をモジュールにまとめる。

4) 実装

プログラミング言語で実現するか、データベースシステムで実現するか、コンピュータ以外で実現するかを決める。次に、クラス・操作・結合を対応する各言語

ごとに指針に従って展開する。

以上の1)～4)の作業を経た後に、運用、保守の段階に到る。OMTによる開発工程をFig.2.9に示す。

OMTは言語に依存しない方法論である。しかし、非オブジェクト指向プログラミング言語を実装言語として用いる場合、言語が支援していない機能はプログラマがそれぞれの言語の機能だけを用いてプログラムコードに翻訳せねばならず、大きな負担となる。また、将来的に継続してオブジェクト指向パラダイムを厳守するにはプログラマの自己統制が必要となり、長期あるいは大規模プロジェクトでのソフトウェアの品質を維持するのは困難である。従って、実装言語としてはオブジェクト指向プログラミング言語を選択することが望ましい。

2. 4 ソフトウェアの開発環境の現状^[1, 15]

1970年代ごろから自動車や半導体、電気製品などいわゆる物の生産の現場では、コンピュータで制御されたロボットやCAD/CAMシステムの導入で大規模かつ徹底した自動化が行なわれている。一方、ソフトウェア開発の現場では、1950年代にコンパイラが出現し、COBOL言語やFORTRAN言語で記述したプログラムから機械語への変換が自動化された。しかしながら、それ以降は、大きな進歩はなく、現在でも、ほとんどが手作りによるソフトウェア開発が続いている。

機器の電子化の進展や社会の高度情報化によるソフトウェアの需要は、今後ますます増大すると予想されている。しかしながら、人的資源の枯渇などを考慮に入れれば、プログラマ不足により供給が減少するのは明らかであり、大きな問題となっている。この問題を解決するためには、コンピュータによるソフトウェア開発の支

援が有効であると考えられ、ソフトウェアを少しでも工業的に生産しようとその開発形態をモデル化し、各工程の作業内容や目標や成果物を規定する試みが、今日まで多く行なわれてきた。主要なモデルは次のようなものである。

- ・ウォーターフォールモデル
- ・スパイラルモデル
- ・再利用モデル

3つのモデルをFig.2.10 (a), (b), (c)^[1]に示す。これらの開発工程を実現するために、構造化方法論やオブジェクト指向方法論などが生まれた。そして、開発工程をコンピュータにより支援・自動化するのが CASE (Computer Aided Software Engineering) ツールである。

ソフトウェア工学は、ホスト中心の開発を行なっていた時代までは、手作業を支援する方法論、紙ベースの仕様書体系、プログラミング段階だけの効率化、それらを支援する環境と発展していったと考えられる。1980年代中期になって、ワークステーションの発展に伴ない、図・表を用いたインターフェースが低価格で利用できるようになり、構造化分析を支援するビジュアルなツールがアメリカを中心に開発され利用されるようになるにつれて、これらをCASEツールとして統一的に分類するようになった。

初期のCASEツールは一連の仕様書の作成を支援する上流CASE (Upper-CASE: 分析工程および設計工程を支援する) であった。その後、プログラム開発、メンテナンスを支援する下流CASE (Lower-CASE: コードジェネレータなどを含む導入工程、メンテナンス工程を支援する) が登場し、上流CASE、下流CASEを使用しての開発が行なわれるようになった。さらに、一貫支援を行なう統合型CASE (Integrated-CASE: 一貫した方法論を適用して、全工程を支援する) へと発展しつつある。

現在ではデータベース、事務処理の分野を中心にして、構造化方法論を用いた上

流CASEツール、構造化プログラミングなどを支援する下流CASEツールが多数存在する。しかし、現段階では全ての工程をカバーしているツールはまだ開発されていない。また、オブジェクト指向方法論を用いた上流CASEツールは、現在は発展段階である。

オブジェクト指向技術の導入により、部品化、（要求仕様・設計・ソフトウェア部品の）再利用技術もさらに向上すると考えられ、今後はさらに工業的生産に近い統合化CASEへと発展するだろう。

Table 2.1 Comparison of OOPL

	C++ 2.0	Smalltalk-80	CLOS	Eiffel	Objective-C
Integration of classes with primitive types	hybrid	pure	Integrated	Integrated	hybrid
Strong type checking	Y	N	N	Y	Y
Ability to restrict access to attributes:					
Control of access from clients	Y	Y	N	Y	Y
Control of access from subclasses	Y	N	N	Y	N
Standard class library	N	Y	N	Y	Y
Parameterized classes	F	-	-	Y	N
Multiple inheritance	Y	N	Y	Y	N
Scoping of class name (packages)	N	N	Y	N	N
Messaging model:					
Single target object	Y	Y	N	Y	Y
Dynamic binding on multiple args	N	N	Y	N	N
Method combination features:					
SUPER concept	N	Y	Y	Y	Y
Before & after methods	N	N	Y	N	N
Assertions and constraints	N	N	N	Y	N
Metadata at run-time	N	Y	Y	N	Y
Garbage collection	N	Y	Y	Y	N
Efficiency:					
Static binding when possible	Y	N	N	Y	Y

Key to table entries :

Y = Yes, the feature is present.

N = No, the feature is not present in common current implementations.

F = Planned in a future release

- = Not Applicable: Parameterized classes are not needed in languages with weak typing.

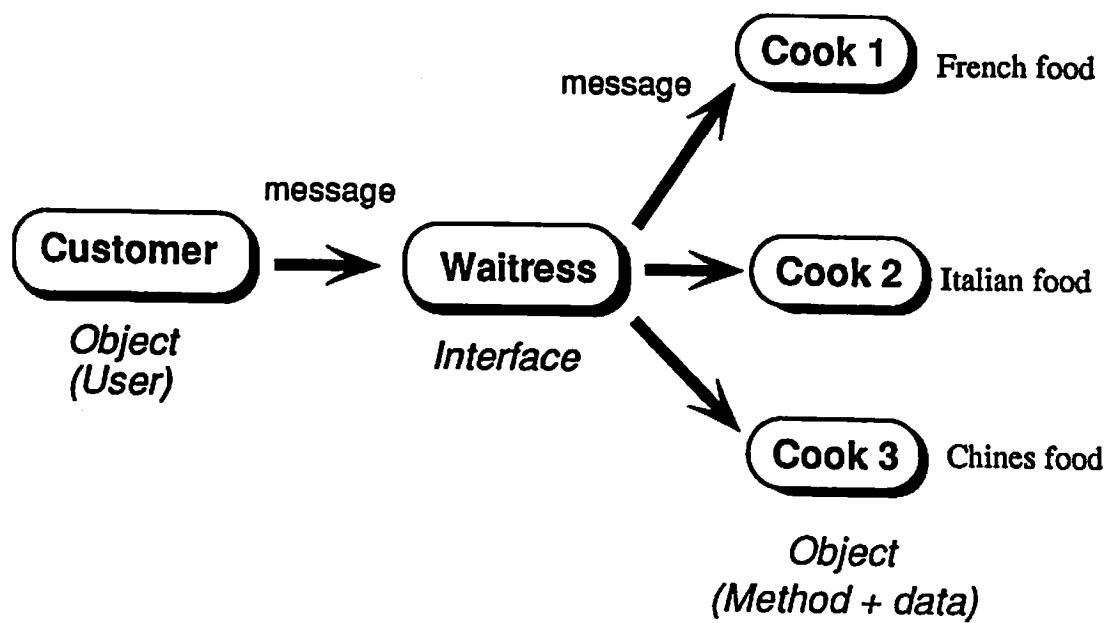


Fig.2.1. Object-Oriented Concept (Modeling of People in Restaurant)¹¹

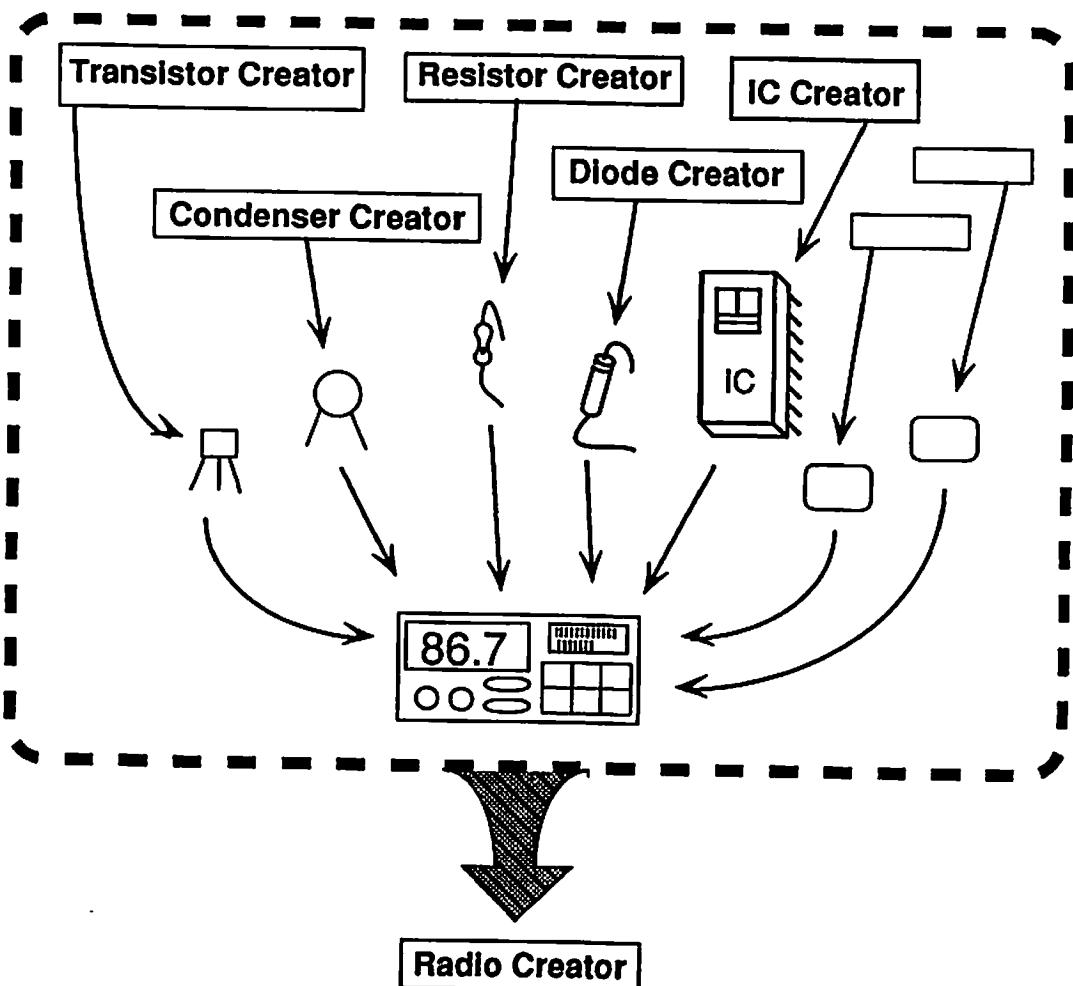


Fig.2.2(a) Object and Class in Production of Radio

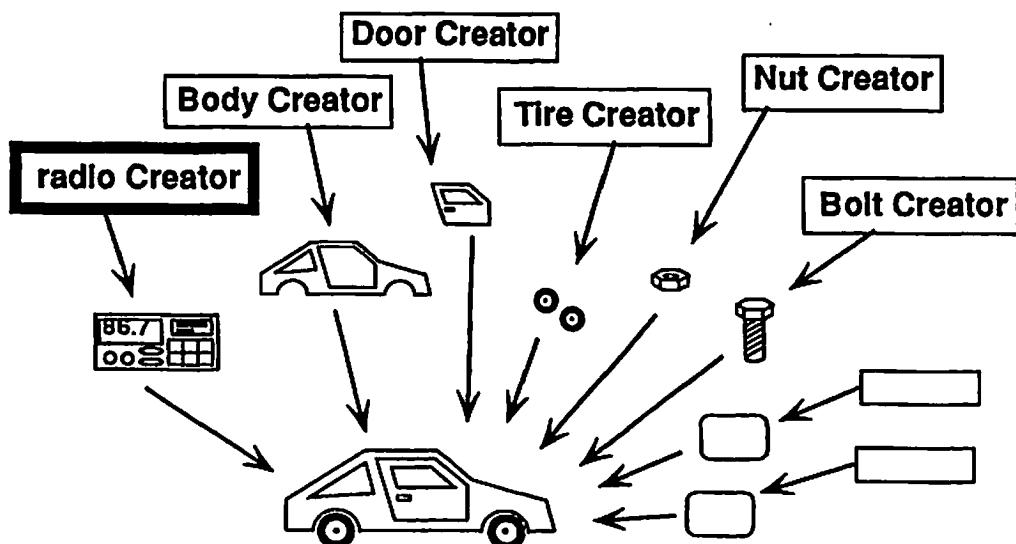


Fig.2.2(b) Object and Class in Production of Car

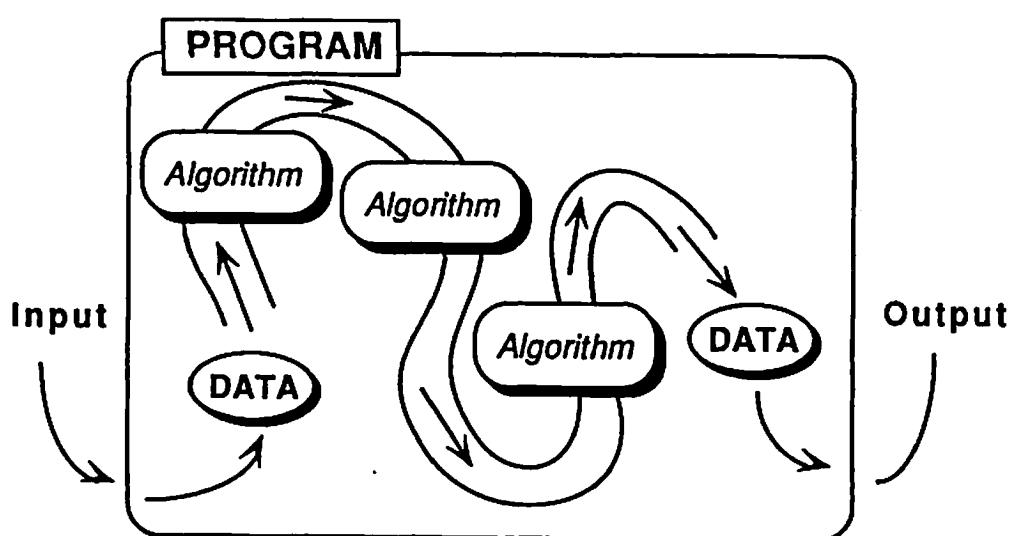


Fig.2.3. Conventional Procedural Programming

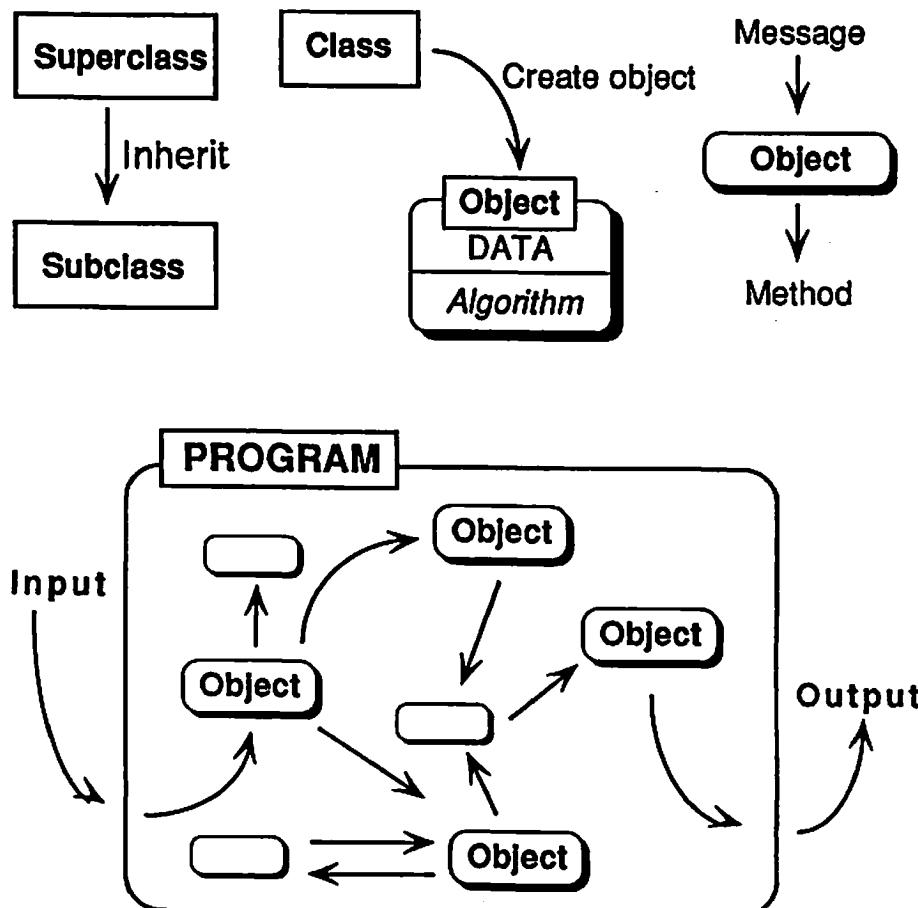


Fig.2.4. Object-Oriented Programmings

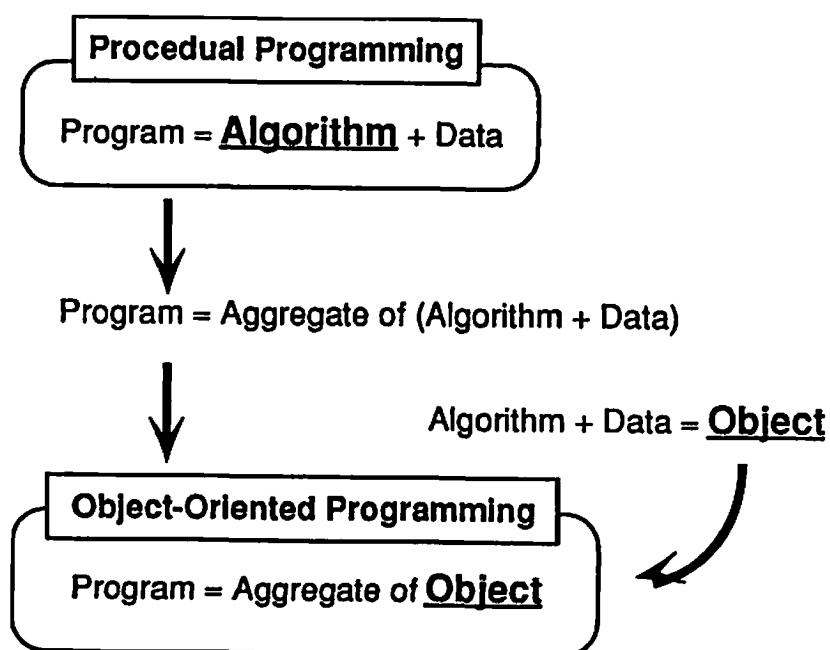


Fig.2.5. Readjustment of Programming

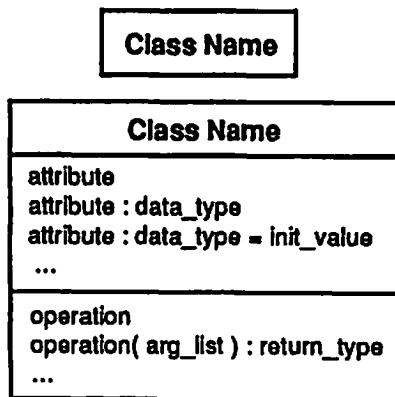
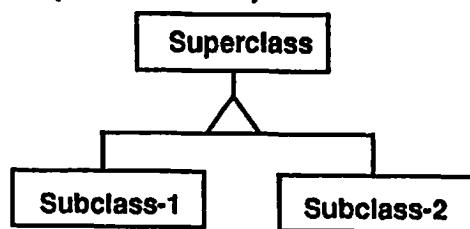
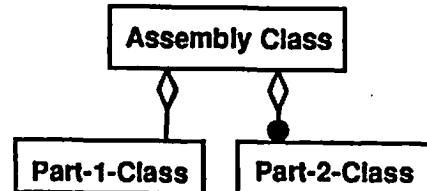
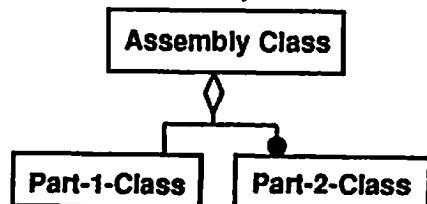
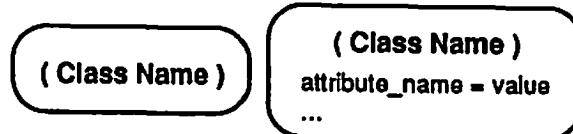
Class:**Generalization (Inheritance) :****Aggregation :****Aggregation (alternate form) :****Object Instances :**

Fig.2.6 (a) Object Model Notation Basic Concepts

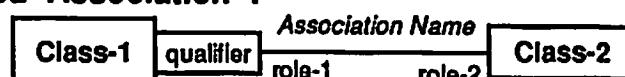
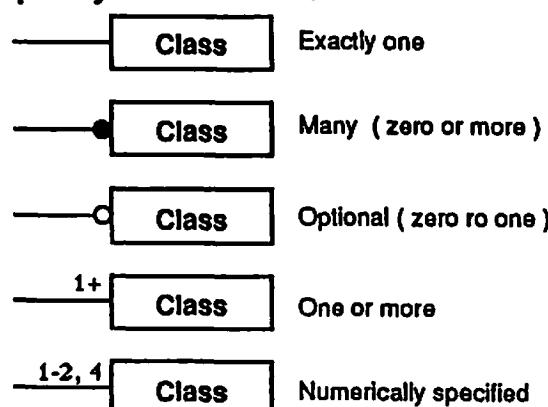
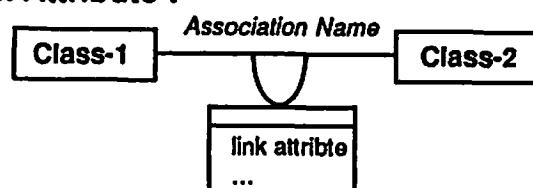
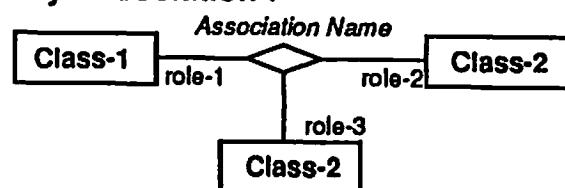
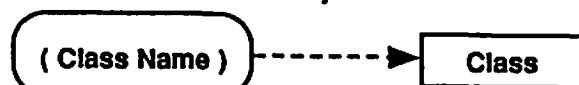
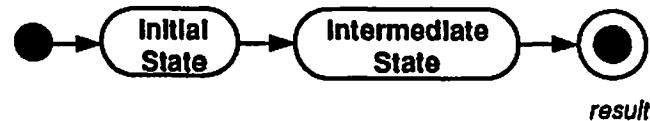
Association :**Qualified Association :****Multiplicity of Association :****Ordering :****Link Attribute :****Ternary Association :****Instantiation Relationship :**

Fig.2.6 (b) Object Model Notation Basic Concepts

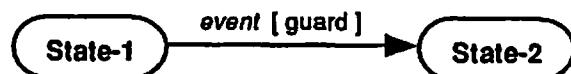
Event causes Transition States :



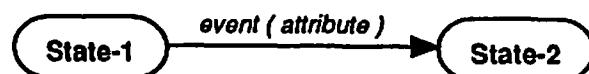
Initial and Final States :



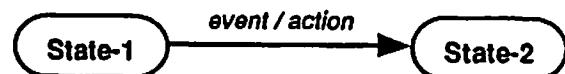
Guarded Transition :



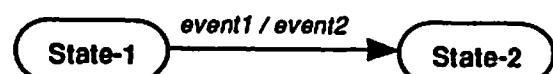
Event with Attribute :



Action in Transition :



Output Event on a Transition :



Sending an event to another object :

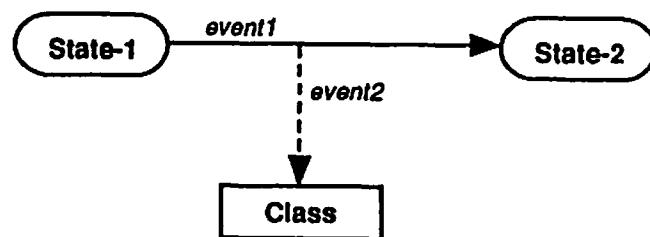
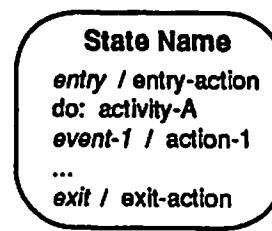
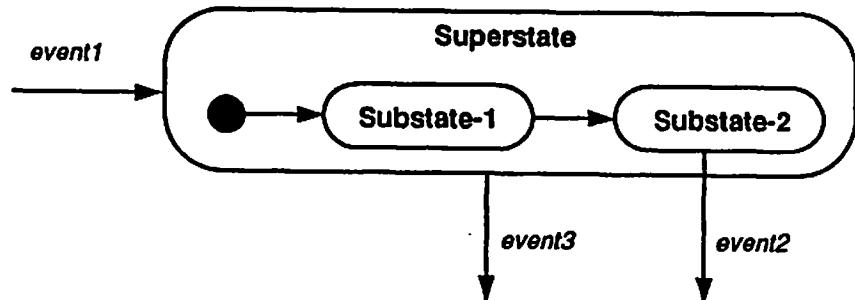


Fig.2.7 (a) Dynamic Model Notation

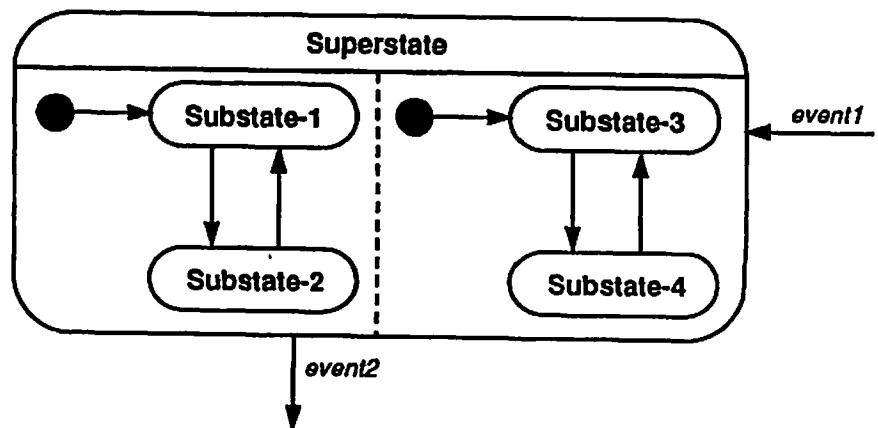
Action and Activity while in a State :



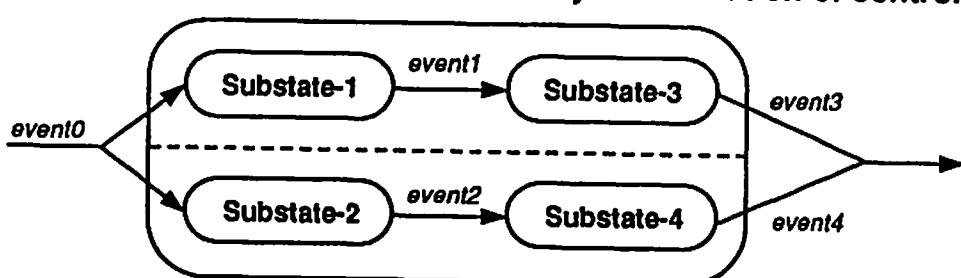
State Generalization (Nesting) :



Concurrent Subdiagrams :

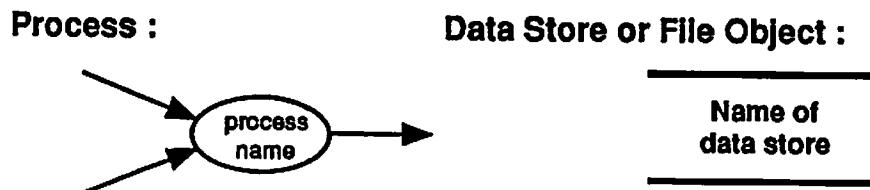


Splitting of control :



Synchronization of control :

Fig.2.7 (b) Dynamic Model Notation



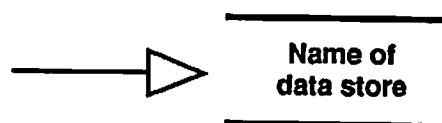
Actor Objects (as Source or Sink of Data) :



Data Flow between Processes :



Data Flow that Results in a Data Store :



Control Flow :

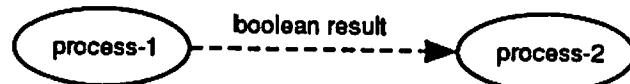


Fig.2.8(a) Functional Model Notation

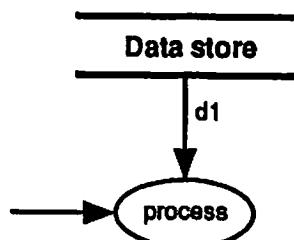
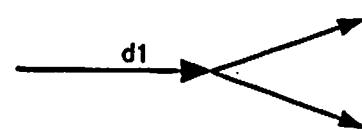
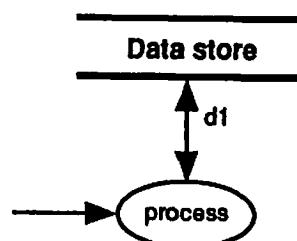
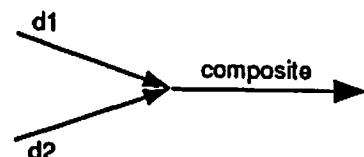
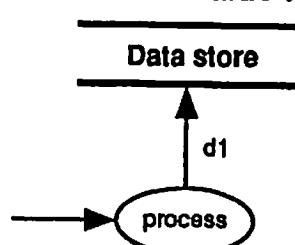
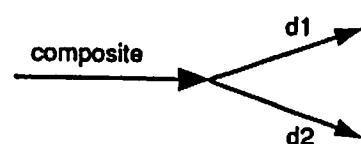
Access of Data Store Value :**Duplication of Data Value :****Access and Update of Data Store Value :****Composition of data Value :****Update of data Store Value :****Decomposition of Data Value :**

Fig.2.8(b) Functional Model Notation

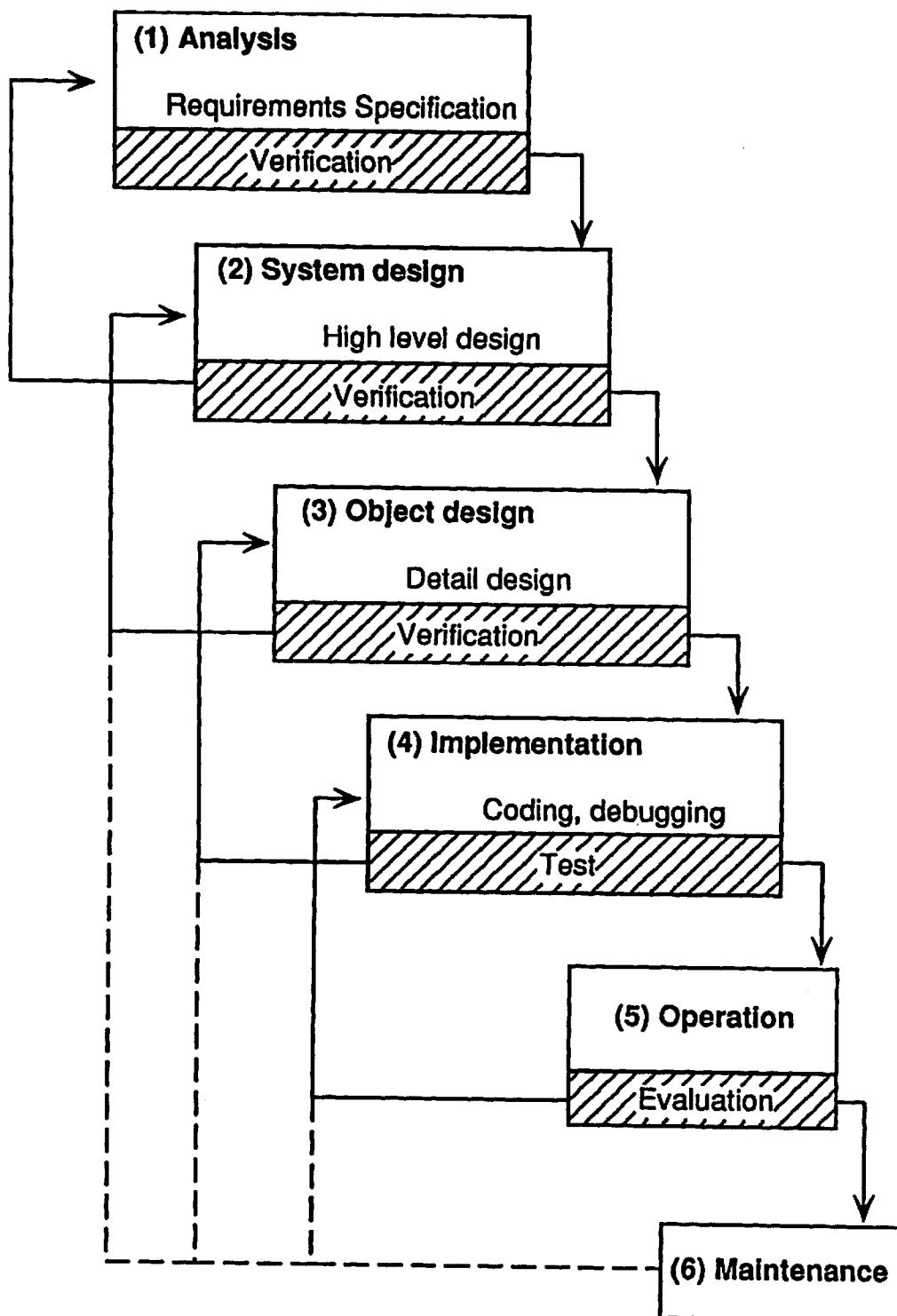


Fig.2.9. OMT's Production Process

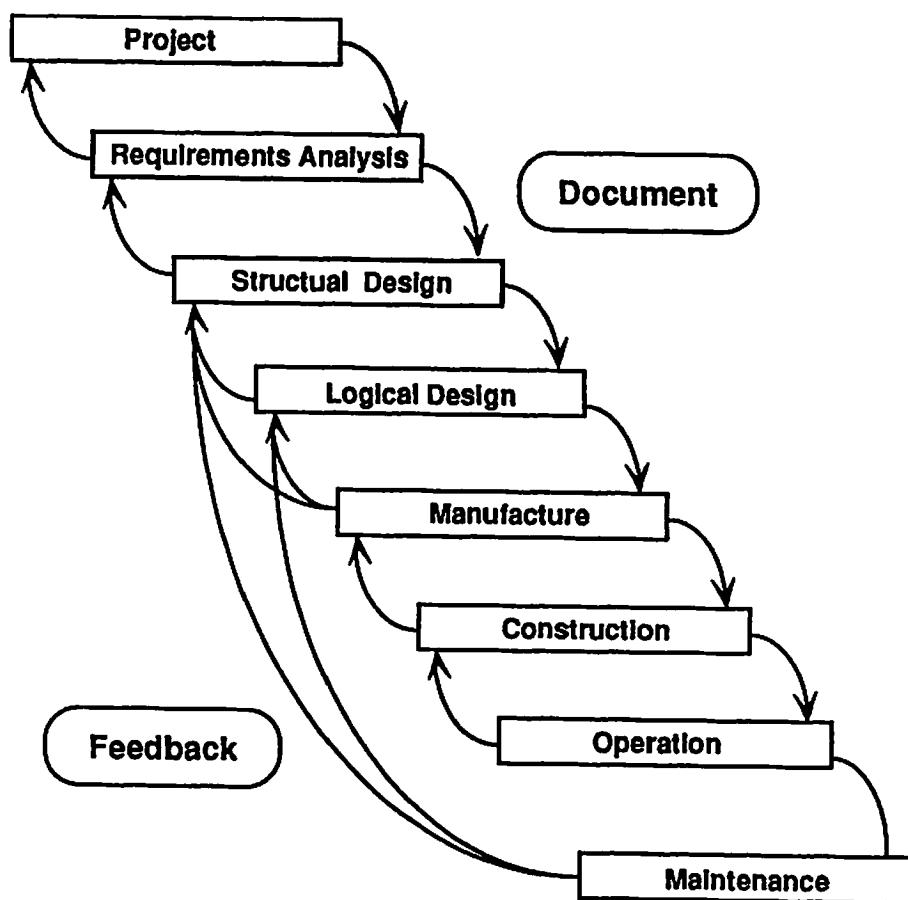


Fig.2.10(a) Water Fall Model

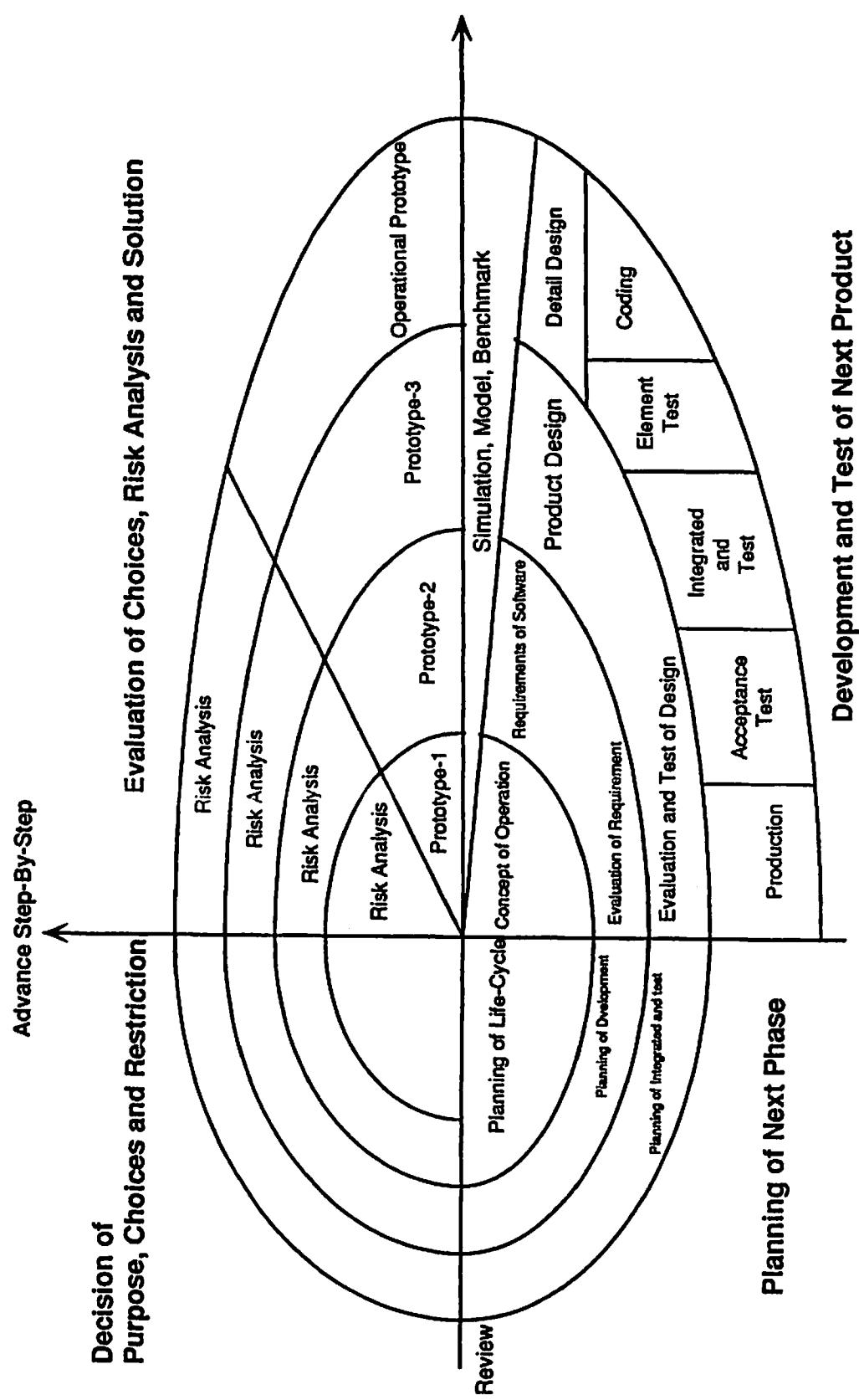


Fig.2.10 (b) Spiral Model

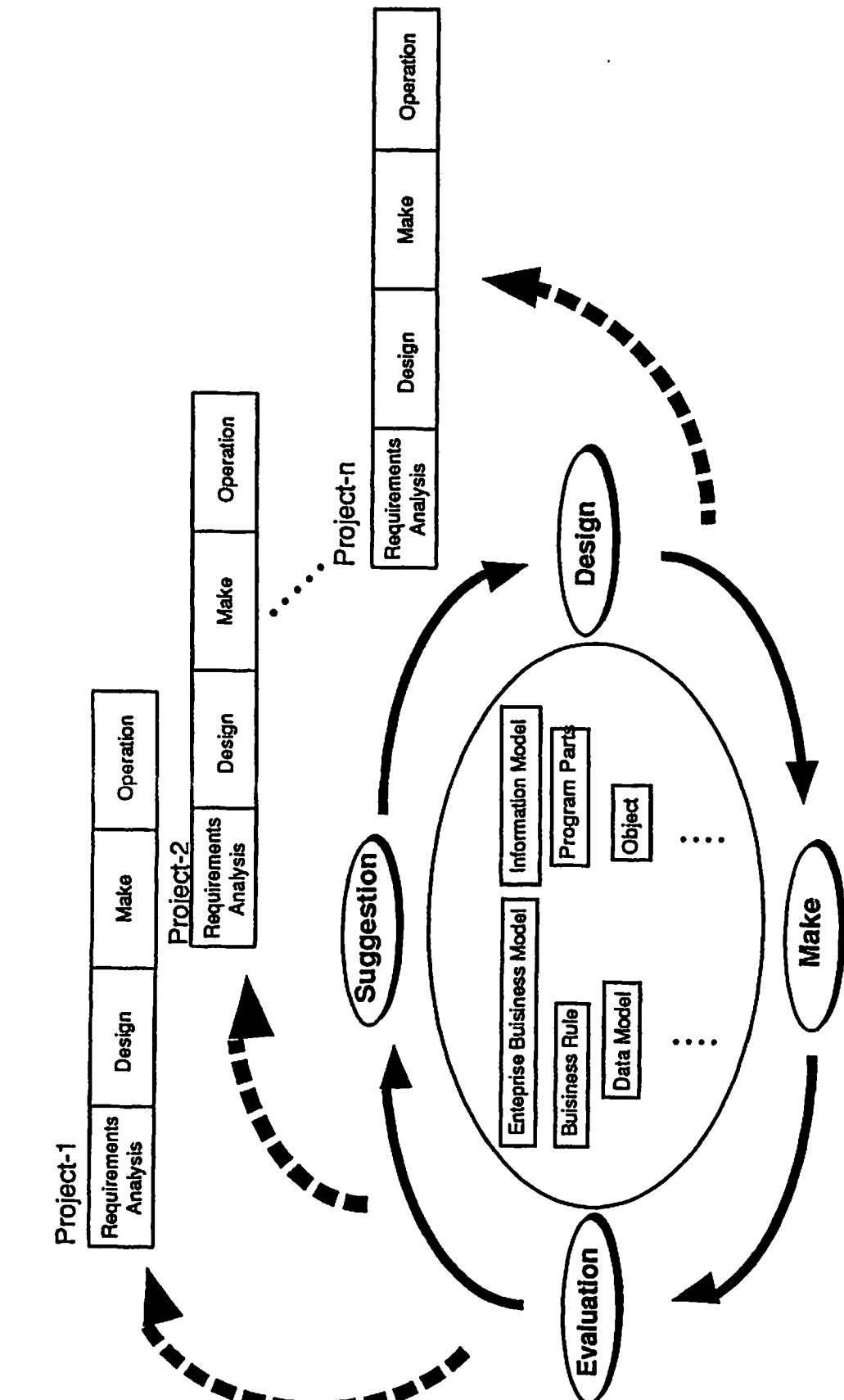


Fig.2.10 (c) Reuse Model

第3章 技術計算における研究・開発環境

3. 1 近年の技術計算分野の研究・開発における問題点

近年のハードウェアの発展は目覚ましものがあり、処理速度の増加、コストダウンのおかげで多様な分野でコンピュータが用いられるようになってきた。しかしながら視点を変えれば、コンピュータ科学・コンピュータ産業がいまだに幼児期にあると言え、技術の変化速度は他の分野とは比べものにならない。従って他分野の研究者は、日々刻々変化し続ける技術を理解しなければ、充分にコンピュータを使いこなすことができないという、非常に憂慮すべき事態に陥っている。これは技術計算の分野でも同様である。

一般的に技術計算の研究では、プログラムを構築し計算結果を考察するという作業が行なわれる。当然ここでは、理論や精度、アルゴリズムなどについて議論されるべきであるが、この作業の内、ツールの使用法の習得やコーディング、デバッグという、本論ではない部分に費やされる時間が少なくない。ソフトウェア工学の分野では、第2章で述べたようにソフトウェアをいかに効率良く、安全に構築するかについての研究・開発が行なわれているにもかかわらず、技術計算ではほとんど適用されることがないのが実状である。このようになっている原因としては、

- ・計算結果を出すのが先決であり、将来の拡張性や再利用性まで考慮している時間がない。
- ・一般的に言って、技術計算の分野では現在の環境に問題が多いということが認識されていない。
- ・現在の技術計算、ソフトウェア工学は高度なものとなっており、研究者

は計算理論についての深い知識を習得するだけで精一杯となり、ソフトウェア工学の内容をカバーするまでには到らない。

- ・他の工学と比べてソフトウェア工学の技術の変化速度は異常であり、そのことが他分野の研究者が内容を理解するうえで、大きな壁となっている。

などが考えられる。

データベースや事務処理、システム管理などの環境（CASEなど）のための研究・開発は、企業のニーズが大きいことから多数行なわれている。今後は技術計算でも、研究者をコーディング・デバッグなどの単純作業から開放し、できるだけ創造的な仕事に専念させることができるような環境作りをしていかなければならぬ。

3. 2 技術計算の将来

大学のように研究者（学生）が頻繁に入れ替わるような現場では、研究資産を継承しなければならないという問題に対処しなければならない。特にプログラムコードは自動車や電気製品のように規格化された部品を使い開発するようなものとは異なり、不定形の”記述”を扱わねばならず困難を極めている。他人の記述したプログラムを理解するのは難しく、2～3年はコードを改良して使うことはあるかもしれないが、新たに作り直すことがほとんどである。

このような状態を開拓するには、第2章で述べた技術を使って、標準のツールを用いてプログラム開発を継続して行なえるような環境を実現せねばならない。また、プログラム開発の際には、再利用性、拡張性のことを常に念頭に置き、“将来の研究者のためにソフトウェア資産を残す義務がある”ということを認識さ

せる思想教育も重要となってくる。

現在の「コンピュータ科学の分野」、「技術計算の分野」、「実験・機械設計などの現場」の関係では、相互の情報がすみやかに反映されにくい状況にある。 「コンピュータ科学の分野」からは多数の情報とツールが提供されているが、その情報を完全に消化できることができず、「技術計算の分野」からの情報がすみやかに現場に反映されにくい。

そこで Fig.3.1のように「計算環境供給グループ」をその間に加えたときのことを考えてみる。各分野の間には次のような相互関係が生まれ、分野の壁を緩和することに寄与できる。

- ・「計算環境供給グループ」はハードウェア・ソフトウェアの新しい技術を用いて、ソフトウェア部品やそれを扱う環境を供給するので、技術計算により早く新しい技術を投入できる。
- ・「コンピュータ科学の分野」にツールについてのバグレポートや数値解析にはどのようなツールが必要なのかをフィードバックできる。
- ・技術計算の研究開発には、供給されたツールを用いれば良く、新たな解析手法の開発などの創造的な研究に専念できる。
- ・数値解析の研究者は「計算環境供給グループ」に対して、ツールのバグレポートや要望をフィードバックできる。
- ・実験や設計の現場に新しい手法についてのプログラムコードがより早く供給される。
- ・数値解析の研究者に対して理論についてのバグレポートなどがフィードバックされる。
- ・理論的にある程度確立されたソフトウェア部品はデータベースに蓄えられて、必要時に参照される。
- ・「実験・機械設計などの現場」にも同じ計算環境が供給されることで、

理論と実験を簡単に検証できる。

以上のような環境を実現するためには「計算環境供給グループ」が重要なパイプ役を演じてくれる。

情報が公開されているにもかかわらず、技術計算分野はソフトウェア工学の観点から見ると鎖国状態に等しいと言って良い。種子島に火繩銃（FORTRAN）が伝来して以来国を閉ざしている間に外界（ソフトウェア工学）では産業革命（構造化、オブジェクト指向）が起こっており技術的に大きく遅れを取ってしまった。新しい技術を使うには確かに大きな負担を伴なう。しかしそのような技術は使ってみないと使えるかどうかさえもわからないし、また使わなければ成熟もしない。現在はもちろんのこと将来においても技術計算は、最もコンピュータを使用する分野の1つであるのは搖るぎ無い事実であり、コンピュータ科学分野との間を埋めるための計算環境を整備するための研究・開発がもっと盛んに行なわれるべきである。（計算力学の中に計算環境工学という分野を新しく作っても良いのではないだろうか？）

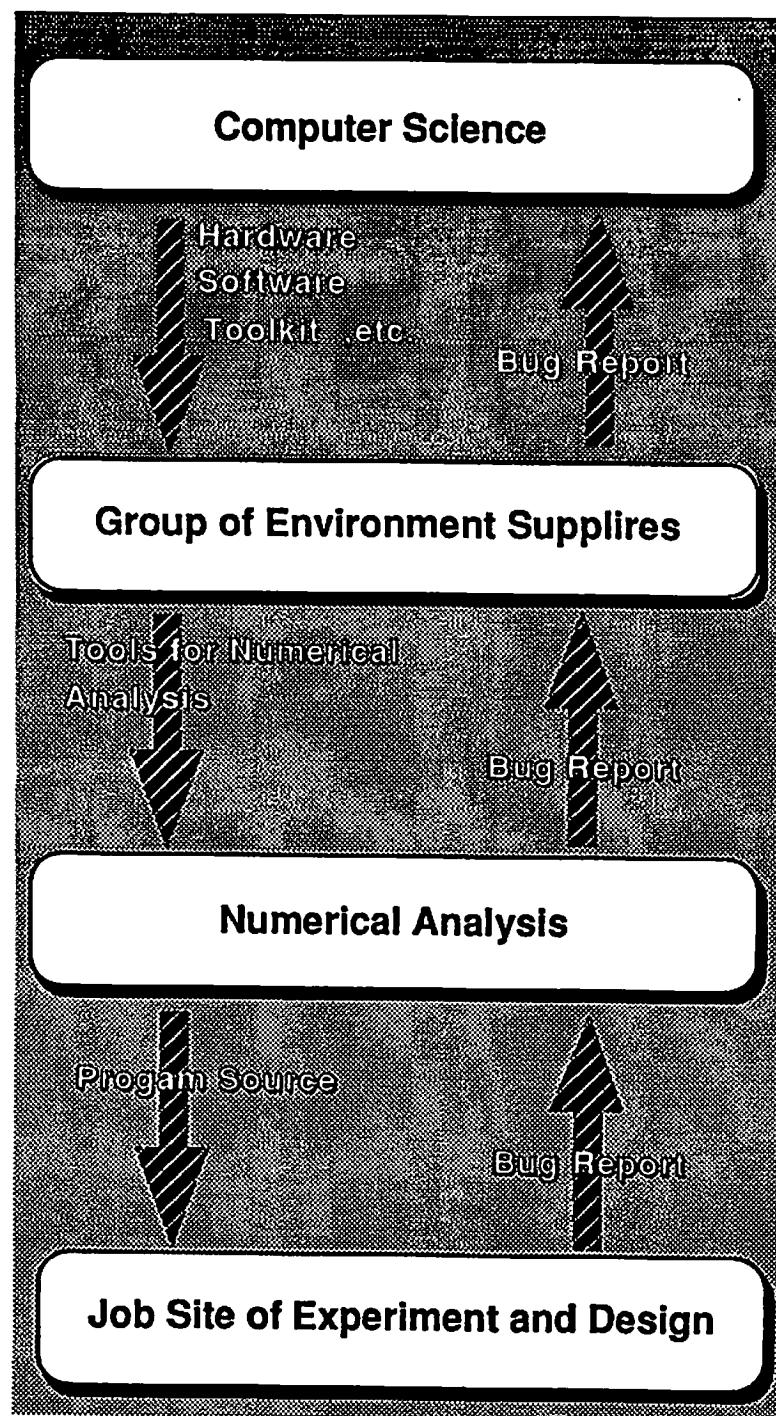


Fig.3.1. Future Environment of Numerical Analysis

第4章 オブジェクト指向型有限要素解析システム

4. 1. ソフトウェア工学の有限要素解析への適用

前述してきたとおりコンピュータ科学の技術が、すみやかに技術計算に適用されることは難しい状態にある。本研究では、このような状態を開拓するための試みとして、数値解析法の一つである有限要素解析のための環境を構築する。有限要素法を選択した理由としては次のようなものが上げられる。

- a) 構造解析を中心として広く使われている解析方法の一つであること。
- b) 歴史も長く、成熟期に達している解析方法であること。
- c) 広く使われているにもかかわらず、ソースとして公開されている汎用コードはADINAなど極少数であること。
- d) ADINAはFORTRANで記述された巨大で複雑なプログラムであり、内容把握には長い時間を必要とし、事実上書き換えは不可能に近いこと。
- e) 解析の前／後処理（プレ／ポストプロセッシング）についての研究は進められているが、解析本体についての開発環境は整備されていない。

オブジェクト指向法は本来変更に強いもので、新たな手法を摸索（プロトタイピング）するのに適していると一般的には言われている。その意味では、b) は理由として適していないようにも見える。しかしながら、本研究ではソフトウェア工学の技術（部品化、再利用など）をどのように適用できるかを最重要綱目に置いている。現時点ではソフトウェア部品の構築時に解析理論が不变である方が都合が良く、技術計算での実現方法を示す方が先決である。また、広く使われている手法であることから、場あたりしだいにプログラミングされたプログラムをオブジェクト指向によりソフトウェア部品として整理仕直し、それを資産として残すことのメリットも

大きい。

ソフトウェア工学自体もまだ発展途上にあり、新しい技術を適用するのは難しい。しかし、本研究のような研究・開発をソフトウェア工学と並行して多数行なうことによって、本当の意味で技術計算に必要な環境が実現できることを確信している。

4. 2. 有限要素法におけるオブジェクト指向の研究

近年、オブジェクト指向プログラミングが注目されるようになり、有限要素法に関する研究も、しだいに行なわれているようになっている。

Fordeらは既存のコード（彼らはこれをNAPと呼んでいる）にオブジェクト指向プログラミングの拡張を行なったObject NAPについて議論している。文献[16]では2次元弾性問題を扱っており、オブジェクト指向設計、設計の視覚的表現についても触れている。実装言語はオブジェクトリンクにポインタを使用したCとPascalの混合型で独自のスタイルを取っている。この論文ではコードがC言語100%使用時の約半分になり、拡張性が向上したと結論づけている。

Scholzは文献[17]で、Timoshenko beamの有限要素解析プログラムについてC++での実装を試みている。これに対しMackieは文献[18]でTurbo Pascalによる実装を試みている。これら2つの研究は有限要素法のコーディングにオブジェクト指向言語を用いたにとどまっている。

Remyらは技術計算プログラムにおけるオブジェクト指向プログラミング環境についてのプロジェクトを進めている。文献[19]ではオリジナルコードに大きな変更を加えないで流体、構造、熱伝導FEMプログラムをいかに構築するかを言及し、特定のマシンについてのウインドウ環境を提供している。この研究ではFEMはあくまで数値解析の例の1つであり、プロジェクト自体は技術計算のオブジェクト指向環境を目指していることから、コンセプト自体は他の研究より一步進んでいると言える。

Dubois-Pelerinらは有限要素法のオブジェクト指向プログラミングについての研究を続けている。文献[20]ではオブジェクト指向プログラミングの概念について述べており、手続き型言語（FORTRANなど）での実装についても議論している。[21]では簡単なトラス問題についてSmalltalkを用いた有限要素解析を行なうことでの検討を行なっており、さらに[22]ではSmalltalkを用いて有限要素法コードのプロトタイピングを行なっている。彼らの研究は、主にSmalltalkを用いた有限要素法のオブジェクト指向プログラミングの検討を行なったものである。我が国における有限要素解析コードについての研究は関東、赤星、青佐、三村のグループが、文献[23 - 25]においてC++によるクラスライブラリ構築の検討を行なっているが、論文の提出数は少なく、乗り遅れている感がある。それぞれの研究の比較をTable 4.1に示す。本研究ではオブジェクト指向方法論の一つであるOMTを適用し、有限要素法クラスライブラリのOOD、OOPにおける設計と実装ギャップ、多重継承の問題点についても考察した。

オブジェクト指向技術はプレ／ポスト、モデリングの分野ではある程度認知されつつある技術であるが、解析コード本体についてはオブジェクト指向パラダイムをどのように適用するかを摸索している段階と言える。

4. 3. 本システムの構成

本研究では次の項目を達成することを目標とした、有限要素解析を行なうための環境を構築する。

- ・（有限要素法クラスライブラリを拡張することによって作成された）過去のソースコードの再利用。
- ・プログラムの部品化（モジュール化）。

- ・部品のブラックボックス化によるプログラミングの効率化。
- ・プログラミングの共同作業を容易にすること。
- ・プログラミングにおける品質管理の達成。
- ・コーディング、デバッグの低減。
- ・ユーザーが情報を参照、記憶、操作、拡張できるプログラミング環境の実現。
- ・計算制御という目的よりも、プログラムの読みやすさ、生産性の高さ、操作性など人間にとって有益な部分の優先。

そこで本研究では次のような環境を提案する。

- a) オブジェクト指向プログラミングを用いて有限要素解析プログラムを部品化し、クラスライブラリとして提供する。プログラミングはFig.4.1のようにライブラリからオブジェクトを生成し、部品として組み立てるだけで行なえる。また、ソフトウェア部品の再利用によって、コーディング、デバッグの量を低減できる。
- b) 一般的に有限要素解析は現在Fig.4.2のような環境で行なわれており、有限要素解析本体とプレ／ポスト処理は別に取り扱われている。本システムではプレ／ポスト処理も有限要素解析の体系の一部と考え、Fig.4.3のようにシステムの中に取り込むものとする。
- c) ソフトウェア部品の資産を参照、登録できるようにする。（データベース機能）
- d) GUI (Graphical User Interface) を使用することにより、部品の選択を自動化する。
- e) 機能拡張時（プログラミング）の支援ツールを構築し、研究者の負担を少なくする。

本システムにおいて、有限要素法クラスライブラリは核となるものである。GUI-

による自動化を行なう場合にも、操作される部品の質が高くなればまったく意味の無いものとなってしまう。それゆえ、ライブラリの設計は充分に検討されるべきであり、再利用性、拡張性を考慮したオブジェクト構造になっていなければならぬ。そのためには従来のようなその場しのぎ的なプログラミングは許されず、一貫した方法論に基づいて行われる必要がある。そこで本修士論文では、OMTを用いて有限要素法クラスライブラリの構築を行うことによりソフトウェアの部品化を示し、さらに設計をも含めた再利用などを検討する。これについては第5章で述べる。また、このような方法論をシステム全体でも用いていかなければならない。

4. 4. 将来の展望

部品、支援ツールが洗練されれば、ネットワークを使った遠隔地からの利用を実現し、多くの研究者にデータを公開できる。また、多くの研究者に利用されれば、各環境でのソフトウェア部品をフィードバックすることにより、資産を充実することができ、Fig.4.4のように相互関係が生まれる。

最終的にはFig.4.5のような分散環境を実現が望まれる。それぞれの環境には個人のデータを蓄積でき、必要に応じてデータベース・サーバにソフトウェア部品を参照できる。また、負荷の高い計算は計算速度の速いコンピュータに任せ、ユーザーインターフェースはそれぞれのWSを持つことができる。

Table 4.1. Comparison Among Object-Oriented Finite Element Analysis Codes

	Language	OOD	OO Graphical Representation	Environment	Remarks
Ferde, Foschi & Stiemer ^[16]	C 32%, Pascal 68%	○	○	×	Development of Object NAP
Scholz ^[17]	C++	×	×	×	Implementation of Timoshenko beam
Mackie ^[18]	Turbo Pascal	×	×	×	Implementation of Class Element
Remy, Davico & Filho ^[19]	C++	×	×	○	Objective Computing Project
Dubois-Pèlerin, Zimmermann & Boome ^[20-22]	Smalltalk	×	×	×	Prototype program in Smalltalk
Kano, Akahoshi, Asai & Mimura ^[23-25]	C++	×	×	×	Construction of Class Library
Present Study	C++	○	○	TBD	Consideration of OOD, OOP in Numerical Analysis

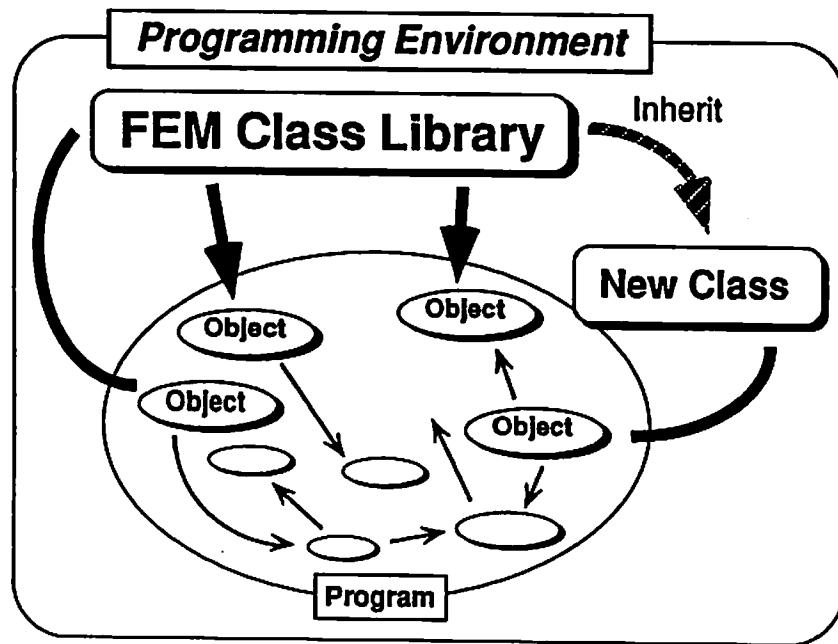


Fig.4.1. Programming Environment for FEM

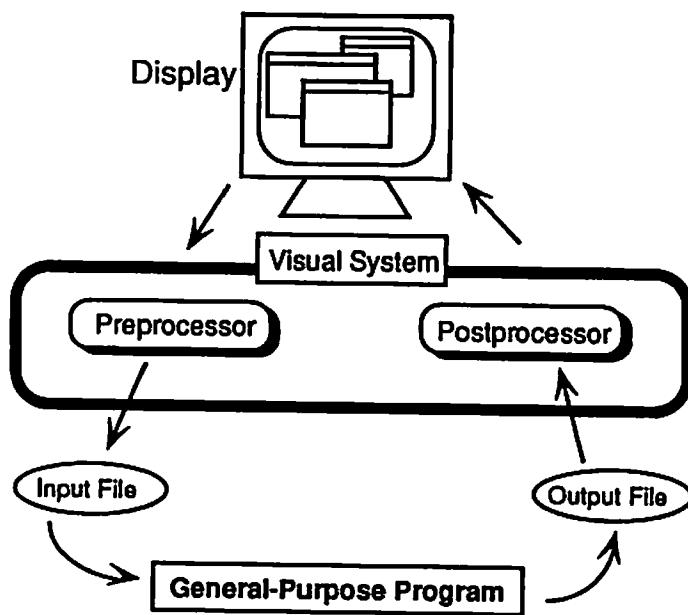


Fig.4.2. Conventional System for FEM

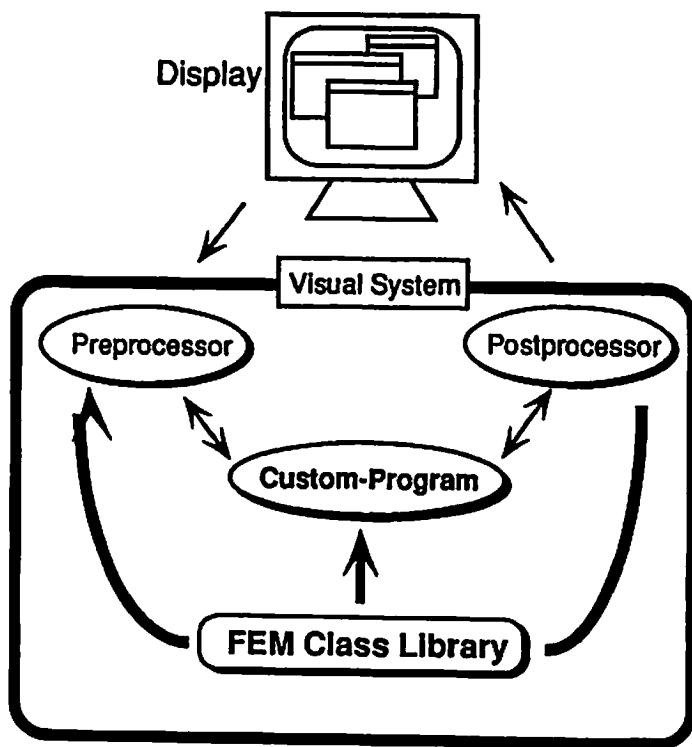


Fig.4.3. Over System for FEM

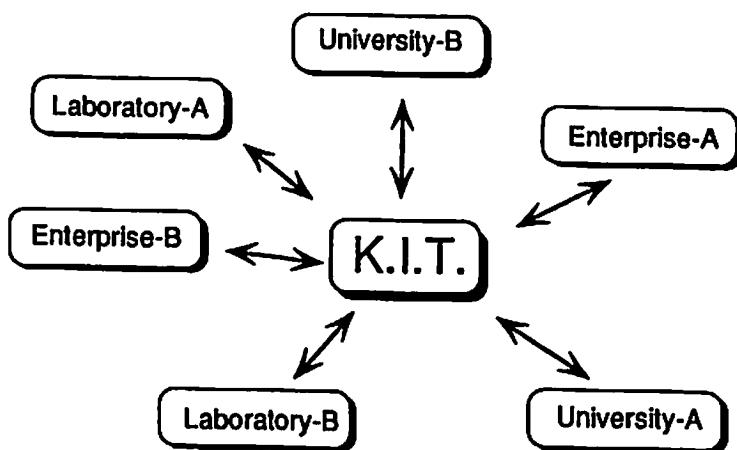


Fig.4.4. Mutual Relationship for FEM

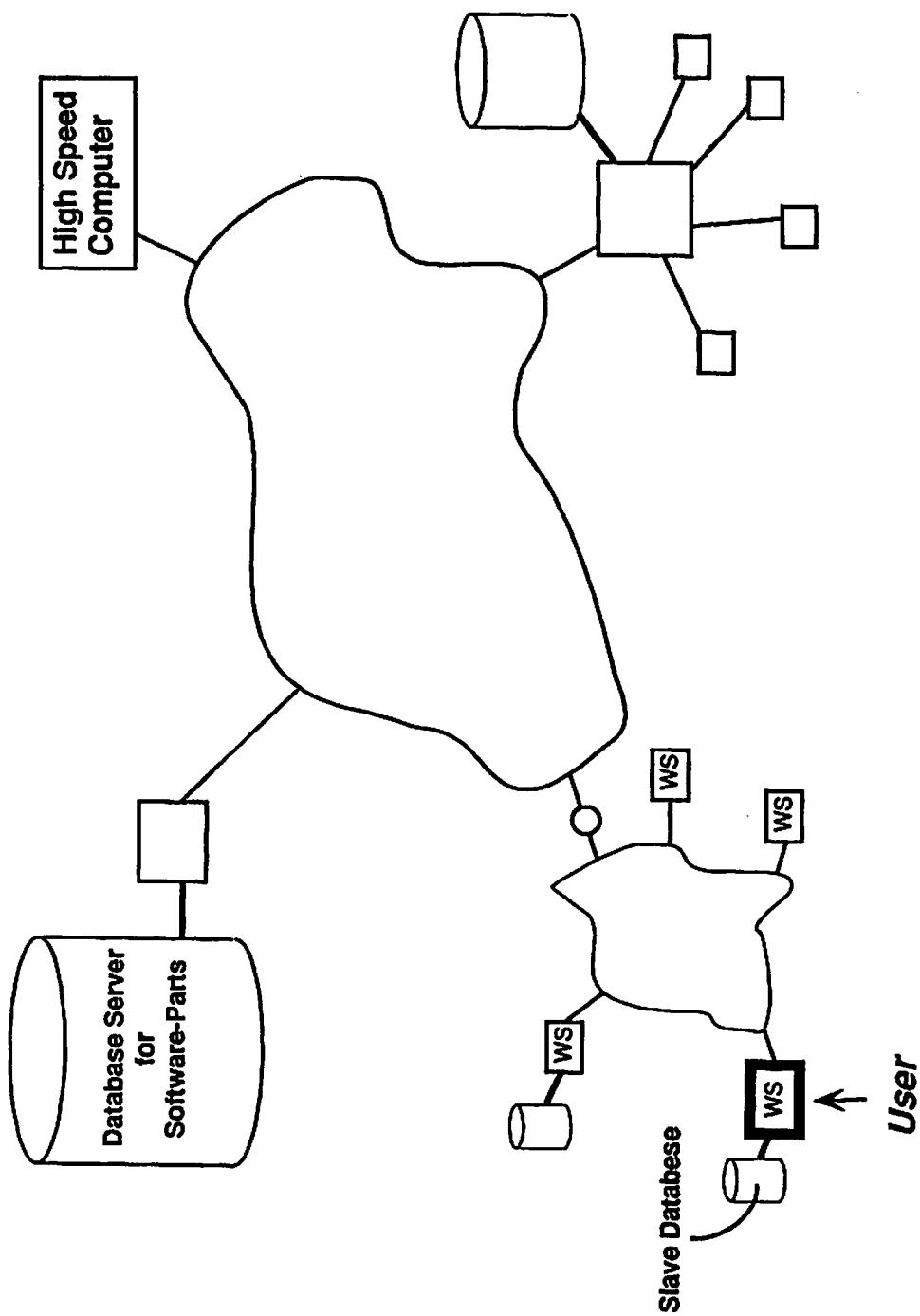


Fig.4.5. Decentralized Environment for Numerical Analysis

第5章 有限要素法クラスライブラリの構築

5. 1. 本研究における設計方針

プログラミングでオブジェクト指向設計を実践するためには、従来の手続き型プログラミングの考え方を大幅に変革しなければならない。しかしながら実際には「手続き型プログラミング」を自覚して行なっている研究者はほとんど存在せず、実現は容易ではない。しかも本研究では本来のソフトウェア生産のようにユーザーの要求をオブジェクト指向設計した後にそれをプログラミングに落とすという手順を踏まず、現在使用されている手続き型のプログラムや手法をオブジェクト指向設計により整理、部品化することになることから、どうしても手続き型プログラミングに影響されやすい。

オブジェクト指向法は曖昧な理論であり、そのポリシーを保って設計を行なうのは容易なものとはいえず、設計者、プログラマになんらかの統一された方針が必要とされる。そこで本研究では有限要素法クラスライブラリを構築するにあたり、次のことを実践することによって設計を進めた。

- 1) OMT法のObject Modelを使用することにより、データの抽象化、オブジェクトの相互関係を視覚的に明らかにし、クラス、オブジェクトの発見に役立てる。
- 2) オブジェクト指向設計を崩さないため、また将来の拡張時の負担を低減するために実装言語にはオブジェクト指向言語であるC++を使用する。
- 3) C++はC言語を拡張したハイブリッドなオブジェクト指向言語である。その上、言語仕様は手続き型と混在した形になっており、どのようにオ

プロジェクト指向するかはプログラマに委ねられる。本研究ではオブジェクトのメソッドの実装にはMayerが提唱するプロシージャとファンクション^[1,2]をできるだけ使用するようとする。C言語やFORTRANではFig. 5.1(a)のように変数のアドレスを引き数として落とし、サブルーチンの中で処理した値をそのまま同じアドレスに代入するという形式のプログラミングが多用される。これはちょうどFig. 5.1(b)のようにプロセスの中をデータが通ることによりなんらかの処理がされるという手続き型プログラミングそのものである。この手法ではプロセスの外でいつデータが壊されたかわからないという危険性があるとともに、流れるデータの型が変わればプロセスも変更せねばならないという保守性の悪いプログラムができてしまう。これに対し、プロシージャ、ファンクションはFig. 5.2(a)のようなオブジェクトに対する操作である。プロシージャはオブジェクトの内部状態を変更する操作でFig. 5.2(b)のようにオブジェクトにメッセージを送れば内部の状態が変更されたことを表わしている。ファンクションはオブジェクトの状態からなんらかの値を計算してくれるような操作で、Fig. 5.2(c)のようにメッセージを送れば計算値を返してくれることを表わしている。このようにプロシージャ、ファンクションによるアクセスを使うことで、オブジェクトにメッセージ（これをメッセージと呼べるかどうかは別にして）を送るという形式でプログラムを構築できる。これによって、オブジェクトの情報隠蔽、データと操作のカプセル化が容易になり、より部品化を進めることができる。ただし、C++でオブジェクトの参照（reference）と関連（association）を表わすにはポインタ（アドレス）を使用する以外に有効な方法が今のところないので、例外的に使用することにする。

- 4) "case"文の使用は可能な限り避けるものとする。"case"文で全ての場

合のメソッドを書いたとしても、もし対応していない操作が生じるときに、いちいちもとのプログラムを書き換えるべきではない。メソッドの選択は"case"ではなく継承されたクラスの選択（ポリモルフィズム：polymorphism）によってなされるべきである。

OOP言語は計算効率とオブジェクト指向法の実現との妥協点で設計されている。従って、現段階でのオブジェクト指向設計は言語に依存するのではなくプログラマのセンスによるところが大きい。また、オブジェクト指向法自体も万能ではなく、いかに再利用可能なクラスを抽出するかが問題となっている。逆に言えば場あたりしだいにクラスを決定してしまえば手続き型プログラミングより難解なプログラムになってしまうことも起こりえることから、今まで以上にプログラミングの設計段階での熟考が必要となってくる。

5. 2. 有限要素法クラスライブラリの設計

5. 2. 1. クラスElementの設計

クラスElementは剛性マトリックス、整合質量マトリックス、集中質量マトリックスを提供するものである。クラスElementは抽象クラスでありそれ自体からはオブジェクトを生成しない。実際の計算に使われるのは具象クラスであるElastic_2D4Nなどである。最終的なクラスElementのObject DiagramをFig.5.3(a), (b)に示す。図中のクラスIntegration, Connect, Material, E_NodeSetは以下に示すようなものを提供するクラスである。

<クラスInteguration> Fig.5.4に示すようなクラス構造を持つ数値積分を行なう

ための情報を提供するクラスである。それぞれの操作は次のことを行なう。

- **init()**

オブジェクトを初期状態にする。

- **next():int**

次の積分点に移る。積分終了時に零を返す。

- **weight():double**

積分点の重み係数を返す。

- **locate(loc[])**

積分点座標値を与える。

<クラス Connect> Fig.5.5に示すような形状関数に関する計算についてのサービスを提供するクラスである。それぞれの操作は以下のことを行なう。

- **Shape(locate[]):Matrix**

形状関数を返す。

- **ShapeDiffL(locate[]):Matrix**

局所座標系での形状関数の偏微分値を返す。

- **ShapeDifferential(local[]):Matrix**

全体座標系での形状関数の偏微分値を返す。

- **set_Node(nSet:E_NodeSet)**

節点の座標をセットする。

<クラス Material> Fig.5.6に示すようなクラスの構造を持つ材料についての情報を探するクラスである。主な操作は次のことを行なう。

- **Dmat():Matrix**

Dマトリックスを返す。

- `set_Dens(Dens:double)`

材料の密度をセットする。

- `Struct_Material::set_Young(Young:double)`

ヤング率をセットする。

- `Struct_Material::set_Poiss(Poiss:double)`

ボアソン比をセットする。

- `Heat_Material::set_Conduct(Conduct:double)`

熱伝導率をセットする。

- `Heat_Material::set_Spec(Spec:double)`

比熱をセットする。

`<E_NodeSet>` 1要素の節点（座標と全体節点番号）の情報を提供する。

以上のクラスを使うことによってクラス Element は次の機能を提供する。

- `Kmat():Matrix`

剛性マトリックスを作る。

- `Bmat():Matrix`

Bマトリックスを作る。

- `C_Mmat():Matrix`

整合質量マトリックスを作る。

- `L_Mmat():Matrix`

集中質量マトリックスを作る。

- `set_Node(nSet:E_NodeSet)`

1要素の節点情報をセットする。

上記のインターフェースはElementから継承によりできたクラス全てに共通であり、使う側は操作の実装がどのように行なわれているかについてはまったく知る必要がない。例えばElementで定義されたKmat()はFig.5.7のように実装でき、線形計算ではほとんどこの実装だけで事足りる。

Fig.5.3(a)中でT_Elas_2d4nは特別な例で以下のようないくつかの設計となっている。T_Elas_2d4nは熱弾性解析のための要素で最初はFig.5.8(a)のようにElastic_2D4NとThermal_Elas_Eから多重継承により実装すべきであると考えた。しかしながら名前、操作の重複などの問題によりFig.5.8(b)のような委譲(delegation)^[13]を使うことにした(delegationとは継承の代わりにオブジェクト内部に継承したいクラスのオブジェクトを属性として埋め込むことによって同じ効果を生む技法のことである)。C++はrename機構を備えておらず多重継承を行なう際に名前の衝突を避けることをできない。また、このような継承における問題はオブジェクト指向技術全般で解決されておらず、現段階では避けることはできない。それゆえ、プログラマが設計、実装時に手動で対処せねばならない。

操作についての設計についてOMTではDynamic Model, functional Modelが用意されていたが、本研究ではObject Diagramでクラスの関係、どのクラスにどの操作が含まれるかを明らかにし、操作の設計実装はC++で直接行なった。

5. 2. 2. クラスFEMの設計

クラスFEMは有限要素解析を行なうためのオブジェクトを生成するもので、Fig.5.9のようなクラス構造となっている。それぞれの操作は次のことを行なう。クラスMesh, Bound_Cond, matrix_eqの役割を以下に示す。

<クラスMesh> 有限要素解析のためのメッシュデータの情報を記憶、参照するためのサービスを提供するクラスである。

<クラス Bound_Cond > 境界条件の情報を記憶、参照するためのサービスを提供するクラスである。

<クラス matrix_eq > 連立一次方程式を解くためのサービスを提供するクラスである。クラスの構造は Fig.5.9 のようになっておりそれぞれの操作は次のようなものである。

・ `set_Size(size:int)`

係数マトリックスのサイズをセットする。

・ `component(i, j:int):double`

係数マトリックスの i, j 要素を取りだす。

・ `set_Mesh(me:Mesh)`

Mesh のオブジェクトをリンクする。

・ `alloc(size:int)`

係数マトリックスのメモリを確保する。

・ `add_elementK(EK:Matrix)`

要素剛性マトリックスを全体剛性マトリックスに足し込む。

・ `profile()`

連立一次方程式の解法に合わせた記憶リストを作る。

・ `decomp()`

係数マトリックスを分解する。

・ `solver(F:Vector):Vector`

連立一次方程式を解く。

・ `set_E_Num(i:int)`

現在、何番目の要素の剛性マトリックスを足し込むかをセット

する。

- `set_Dirichlet(BC:Bound_Cond, F: Vector):Vector`

ディリクレ条件を与える。

- `print()`

全体剛性マトリックスを標準出力に出力する。

継承による拡張を行なうことで、上記のインターフェースを統一的に使用することができるのでユーザは選んだ解法を意識せずにオブジェクトを使うことができる。

クラスFEMは次の操作を提供する。

- `formK()`

剛性マトリックスを作成する。

- `boundary_condition()`

境界条件を設定する。

- `solve()`

連立一次方程式を解く。

- `set_Mesh(me:Mesh)`

Meshのオブジェクトをリンクする。

- `set_Element(E:Element)`

Elementのオブジェクトをリンクする。

- `calc()`

有限要素解析を行なう。

- `print_result()`

解析結果を標準出力に出力する。

- `set_matrix_eq(K:matrix_eq)`

matrix_eqのオブジェクトをリンクする。

本研究の設計ではデータの標準入力をクラスFEMの中に組み込みます、Meshや

`Bound_Cond`などのオブジェクトをリンクし、それを参照する形式をとった。これはFig.5.12(a)のように異なるFEMのオブジェクトが同一のメッシュデータを使う場合などに対応するために必要である。Meshを属性としてFEMの内部データとして埋め込んでしまうとこのような場面では同じメッシュデータのMeshオブジェクトを複数生成しなければならない。また、このような設計にすることでFig.5.11(b)のようにプレ／ポストで使用するMeshオブジェクトをそのままFEMオブジェクトが参照することも可能となる。

5. 2. 3. 弾性応力解析と熱伝導解析のコードの比較

弾性解析と定常熱伝導解析をクラス `Elaetic_FEM` と `SteadyHeat_FEM` を用いて行なう場合のコードを比較してみる。解析を行なうために `Calc_Elastic` と `Calc_Heat` の2つのクラスを用意した。これらのリストを Fig.5.12(a), (b) ならびに Fig.5.13(a), (b) に示す。これらのクラスは標準入力からデータを読み込み、有限要素解析を行ない、標準出力に結果を出力するという簡単なものである。従って操作としては

```
void in_data();
```

```
void calc();
```

```
void out_data();
```

```
void run()
```

の4つしか持っておらず、外部に公開しているのは `run()` だけである。クラス `Calc_Elastic` と `Calc_Heat` では `run()` を起動すれば全てを行なうようにした。

`Elastic_FEM` と `SteadyHeat_FEM` はクラス `FEM` から継承により拡張していることから `set_Mesh(me:Mesh)`, `set_BC(BC:Bound_Cond)`, `calc()`, `print_result()` という同様のインターフェースを用いることができる。従って、`in_data()`, `calc()`, `out_data()` の実装を見れば明らかのようにほとんど同じ記述でプログラムを構築できる。

Calc_ElasticとCalc_Heatを用いたmain() プログラムを Fig.5.14(a), (b)に示す。プログラムを動かすにはオブジェクトを生成し、そのオブジェクトに対しrun()と言うメッセージを送るだけによく、Calc_ElasticとCalc_Heatを使う側は中の実装がどうなっているか気に止める必要はない。このようにOOPではオブジェクトを作ることにブラックボックス化でき、使いたいレベルに応じて最小限の情報だけを用いてプログラムを構築できる。

5. 3. 考察

5. 3. 1. OMTによる設計

本研究では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による実装を行なうためにOMTの適用を試みた。OMTが有限要素法クラスライブラリ構築のために有効であった点、気づいた点などを以下に示す。

a) Object Modelはクラス全体の関係を把握するのに非常に有用であった。

言語だけでは局所的な内容を把握することはできるがプログラムが大きくなるにつれてクラスの数も増え、全体を把握するのが難しくなる。オブジェクト図を用いれば継承 (Inheritance)、関連 (Association)などの記述法が提供されているため、クラスの関係のほとんどをチャートとして表現することができる。

b) 有限要素法クラスライブラリは通常のシステム（オペレーションシステムなど）とは異なり、動的に状態が変わることとはほとんどないのでDynamic Modelを使用する必要はない。一方、有限要素法クラスライブラリを使用して実際のプログラムを構築したときにはDynamic Model

を用いる必要がある。第6章の解析例のところでその使用法について触れる。

- c) プログラムを構築するうえでデータが受け渡されることも当然起こる。しかし、これはオブジェクトにメッセージを送った結果としてデータが受け渡される表現になっていなければならない。その意味ではData Flow Diagramの使用は避けることとした。Data Flow Diagramは構造化分析／設計に使われていたものであることもあり、オブジェクト指向設計への適用は不適切なようにも感じられた。
- d) Object Diagramはクラスの継承、リンク関係、どの操作がどこに属するかなどの静的な情報を記述でき、C++でのクラスの（特に~~.hの）表現をチャートで表わすことができる。これに対しオブジェクトの動的状態、操作の内容などを表わすためにDynamic Model、Data Flow Diagramが用意されている。しかしながらこれらの表現はC++の表現とはかけ離れており実装と設計との間に大きな隔たりを感じた。本研究でのクラスライブラリの構築を行なったかぎりでは、クラスの関係はObject Diagramを使用し、操作などは言語で表現して設計するほうが効率的であった。
- Dynamic Modelはシステムの状態を表現するもので、個々のオブジェクトの状態を表現するには到っていない。また、Data Flow Diagramはプロセスの中をデータが流れることを表わしており、オブジェクトの中の操作以外では使えそうにない。個々の動的なオブジェクトの状態とメッセージのやり取りを扱い、データの流れはあくまでもオプションでありオブジェクトに対するメッセージによって引き起こされることを表現できるようなOOD専用のダイヤグラムの必要性を感じた。
- e) 文献[13]ではOMTによる設計は言語によらない分析／設計ということを強調しており、非オブジェクト指向言語を用いてオブジェクト指

向プログラミングをした例も検討されている。しかし非オブジェクト指向言語による実装では多数の規約を定めねばならず、コードも冗長となりとてもオブジェクト指向しているとは言いがたい。また、それらのプログラミング規約を複数の人間に對し将来を通じて守らせることは不可能である。このようなオブジェクト指向パラダイムへの変換を自動化するためには今日では多数オブジェクト指向言語が研究開発されている。設計と実装のギャップを取り除くためにもオブジェクト指向言語を選択すべきである。

5. 3. 2. C++による実装

C++はC言語の構造体を拡張することで従来の手続き型のコンパイル言語にオブジェクト指向パラダイムの機能を取り込んだ実行効率とオブジェクト指向を両立させた言語である。しかし言語仕様はオブジェクト指向とC言語との区別が明確になされておらず、プログラマにオブジェクト指向の意識がないと部品化やプログラムの安全性を向上させることはできない。著者はオブジェクト指向の専門家ではないことからなんらかの一貫したオブジェクト指向プログラミング法を必要とした。そこで本研究の実装にはルーチン、プロシージャを導入し、できるだけオブジェクト指向設計を崩さないように心掛けた。実装した結果として、C++はかなりの部分でオブジェクト指向プログラミングによる部品化を実践できる言語であると言える。また、それだけ多方面に使用可能な高機能の言語である。しかし、標準ライブラリを持っておらず（好きなライブラリを選択できることからこれがC++の長所だとする見方も存在する）将来にわたってオブジェクト指向パラダイムを継続させるのは難しいように感じた。

オブジェクト指向言語のユーザは、UNIXユーザの多さから見てC言語からC++へ移行する者が多数を占めるものと考えられている。しかし技術計算のプログラマのほ

とんどはFORTRANユーザーでありC++の選択が最良かどうかは議論の余地がある。C言語とオブジェクト指向の二重の壁を越えねばならずC++への移行は困難を極めるであろう。C言語を使いこなすためにはアドレス、ポインタというメモリ管理を行なう低レベルの概念を把握しておかねばならない。また、C++でもオブジェクトの参照にはポインタを使うことからこの概念の理解は不可欠である。著者の大学で見る限り、機械系でFORTRANの引き数がアドレス渡しであることを認識してプログラムを構築している研究者はほんの一部の人間だけだと見受けられ、学生に到っては皆無に等しい。恥ずかしい話ではあるが著者自身これを知ったのは修士課程に入ってC言語を使い始めてからである。構造体についてもC++を使用するまでは存在さえ知らなかった。このような状況は大小の差はあるだろうが技術計算の現場で必ず見られるはずである。

どの言語を選択するにしても数値計算用の部品群と環境ツールの整備がなければオブジェクト指向の壁を越えさせるのは難しい。

5. 3. 3. 設計と実装における問題点

現在の工業製品では「電気の回路図から製品」「機械製図から製品」と言うように設計から実装までを一貫して行なう体系が出来上がっている。しかしながらソフトウェアでは「分析／設計から製品」というようには行なわれていないのが実状である。多くの分析／設計のための方法論は実装言語に依存しない方法論を強調している。従って分析／設計を行なう上流工程と言語による実装を行なう下流工程とに分断されている。これは分析というものがユーザーの要求を満たすための機能を抽出するための方法論として生まれたもので、基本的には実装についてはその機能を満たすものを作れば良いというアプローチを取っていることが原因と言え、実装の負担はプログラマに押しつけられる形となっている。オブジェクト指向的考え方が導入されることにより設計と実装のギャップは緩和されつつあるが、オブジェクト

指向分析／設計も基本的には実装言語に依存しないというアプローチを取っている^[13]。これに対し、Meyerはオブジェクト指向設計と実装を一つの言語（Eiffel）によって行なうことを提唱している^[12]。確かにこのアプローチを取れば設計と実装のギャップをなくすことができる。しかしながら、オブジェクト指向言語が従来の言語より表現力が豊かになったとはいえ、Diagramによる表現の方が比較にならないほど情報量が多いのも事実である。

本研究ではOMTを用い、C++により有限要素法クラスライブラリの構築を試みたが、実際にはOMTで示されているように要求分析、システム設計（高レベル設計）、詳細設計をOMTで行なった後に実装に入るというような工程にはならなかつた。本研究での分析、設計、実装工程はおおむね以下のようになり、図に示すとFig.5.15のようになる。

- a) 本研究の要求は有限要素コードを再利用可能な部品に整理することである。従って、要求分析の結果は有限要素解析に必要と考えられる機能とオブジェクトとなる。
- b) 設計段階ではまず低レベルのクラスの抽出を行ない、それを言語で表現する。初期段階はコーディングというよりも言語表現によって、抽象的なクラスの発見をするというものに近い。
- c) b) で発見したクラスをObject Diagramで表現し、継承関係、関連、属性、操作などをチェックする。
- d) c) での修正点を言語で表現し上のレベルのクラスを作る。この時点で部品として使われるクラス（サプライヤ）に新たに必要な操作などを加えたり、修正を加えたりする。
- e) 言語表現をObject Diagramで表現し、クラスの関係をチェックする。
- f) ~h) 上記の作業を繰り返す。

- i) 個々のクラスを作ることから始め、ボトムアップでまず設計を行ない、全体をObject Diagramでチェックし、トップダウンに設計を行なう、というような繰り返しを行なって最終的な設計を得ることができる。

結局のところソフトウェア開発で問題を解決するためには言語を無視するわけにはいかず設計の領域はFig.5.16のように言語とObject Diagramをまたぐ領域になる。しかしながら言語だけで全体のクラスの関係を把握するのは難しくObject Diagramでクラスの関係を表わすことは非常に有用である。ドキュメントとして残すことで第3者がソースを見るときにクラスの関係を一目で理解することができ、共同作業、メンテナンスにも威力を發揮するはずである。本研究ではOMTの設計を言語によって実装するというよりは、オブジェクト指向言語で作った設計をObject Diagramでドキュメントとして残すという使い方をした。

オブジェクト指向プログラミングを実践している研究者やプログラマにはOOAやOODは実装とのギャップが大きすぎて使い物にならず、それよりは表現力の抱負になったオブジェクト指向言語で問題の考察からプログラムの実装までを行なうべきだとする者も多い。確かにコーディングを行なってみないと明らかにならない面がほとんどであるのも事実であり、その意味では現在のOOADは不充分である。オブジェクト指向システム分析／設計の方法論はユーザの要求をいかに満たすかに力点を置き、「分析->設計----->実装」という上から下へのアプローチを取っており実際に動くものを作らねばならない現場の意見が反映しにくい。これに対し本研究ではある程度実装言語（ただしオブジェクト指向言語）に依存する「言語構造をドキュメントとして残す」という下から上へのアプローチを試みた。オブジェクト指向により、言語が部品を表わす表現力を持ち始め、それらの構成を見取り図として落とすことも可能になりつつある。このようなアプローチが設計と実装のギャップを埋めるものと本研究では考えている。また技術計算ではいくら設計ができても解析を

行なうためのプログラムが構築できなければ意味をなさない。それゆえ数値解析を行なえるプログラム部品を提供しながら数値解析のための方法論と環境の摸索を可能とするこのアプローチが技術計算にはもっとも適していると考えられる。

オブジェクト指向技法を使えばソフトウェア開発工程を職人のレベルから工業的生産へ簡単に移行できるわけではない。使う側から見れば効率的にプログラムが構築できるのであればオブジェクト指向でなくとも何の問題もない。しかしながら、オブジェクト指向技術はコンピュータサイエンスがたどらねばならない課程である。オブジェクト指向法に到るために構造化技法のステップを踏まねばならなかったのと同様に将来到来するであろう新たなパラダイムを使いこなすためには「オブジェクト指向法」というステップを踏まねばならない。将来の技術計算を志す研究者のためにも今このステップをのほる試みを怠ってはならない。

```

main(){
    int x;
    x = 1;
    *
    add(&x, 2);
    printf("x = %d\n", x);
}

void add(x, i)
int *x;
int i;
{
    *x += i;
}

```

PROGRAM MAIN
 IX = 1
 CALL ADD (IX, 2)
 *
 WRITE(6,10) X
 10 FORMAT (1H , 'X =', I5)
 STOP
 END
 *

 SUBROUTINE ADD (IX, I)
 *
 IX = IX + I
 *
 RETURN
 END

Example of C

Example of FORTRAN

result	x = 3
---------------	-------

(a) Example of Procedural Programming



(b) Data Flow Diagram

Fig.5.1. Conventional Procedural Approach

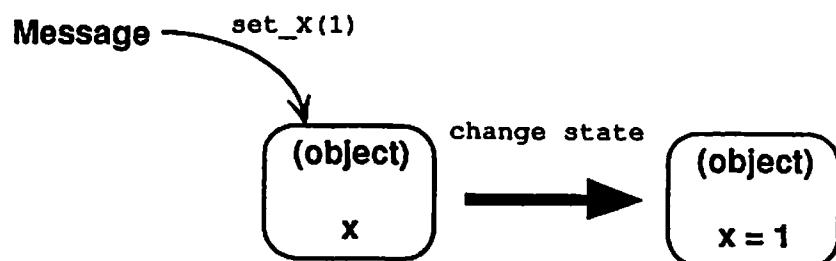
```

class A{
protected:
    int x;
public:
    set_X(int i){ x = i; }
    add(int i){ return x + i; }
};

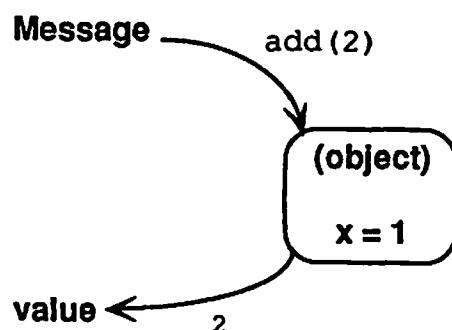
main(){
    A x;
    x.set_X(1);
    cout << "x =" << x.add(2) << "\n";
}

```

(a) Example of C++



(b) Procedure



(c) Function

Fig.5.2 Meyer's Object-Oriented Programming

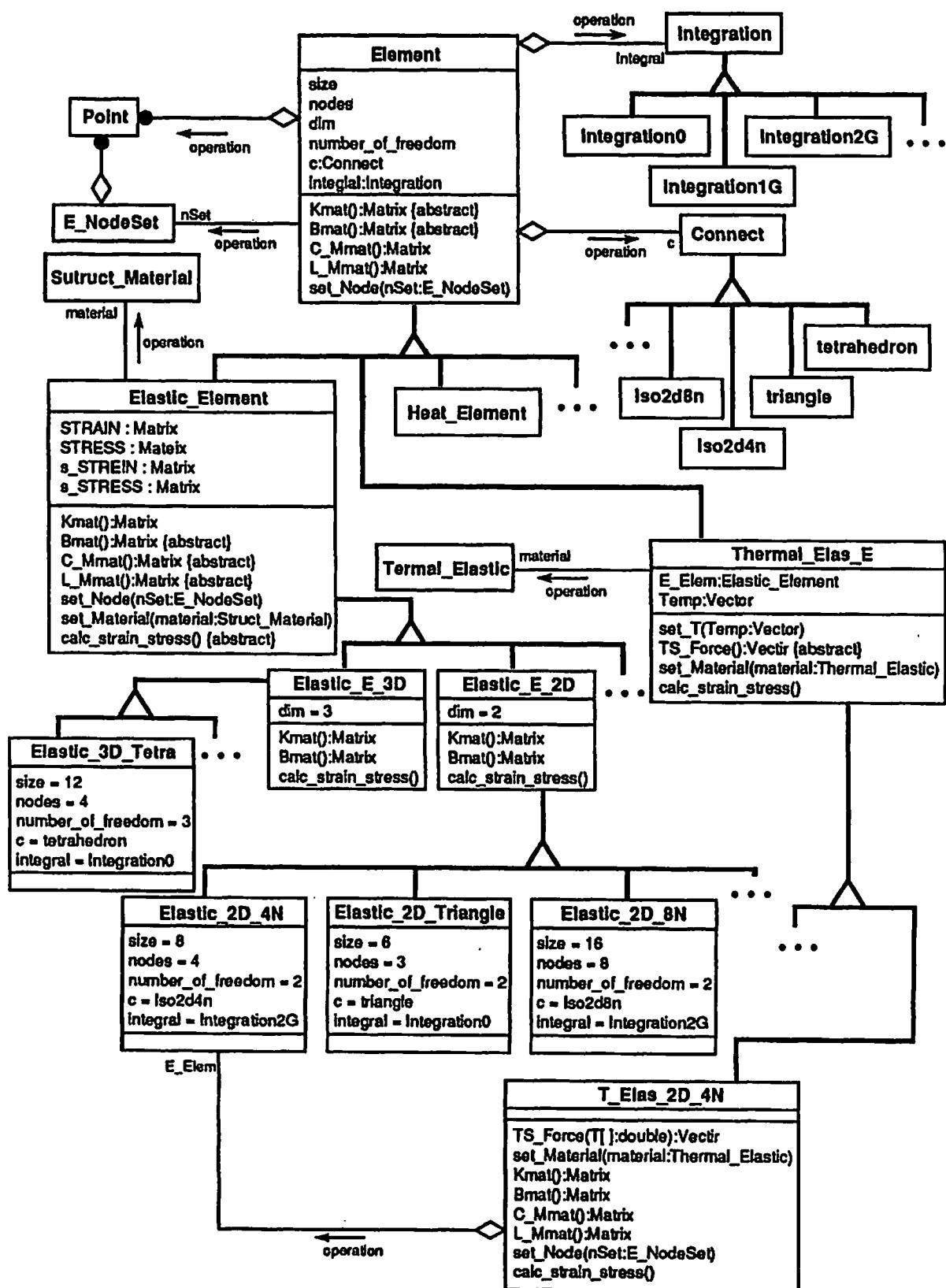


Fig.5.3(a) Object Model of Class Element (Elastic_Element)

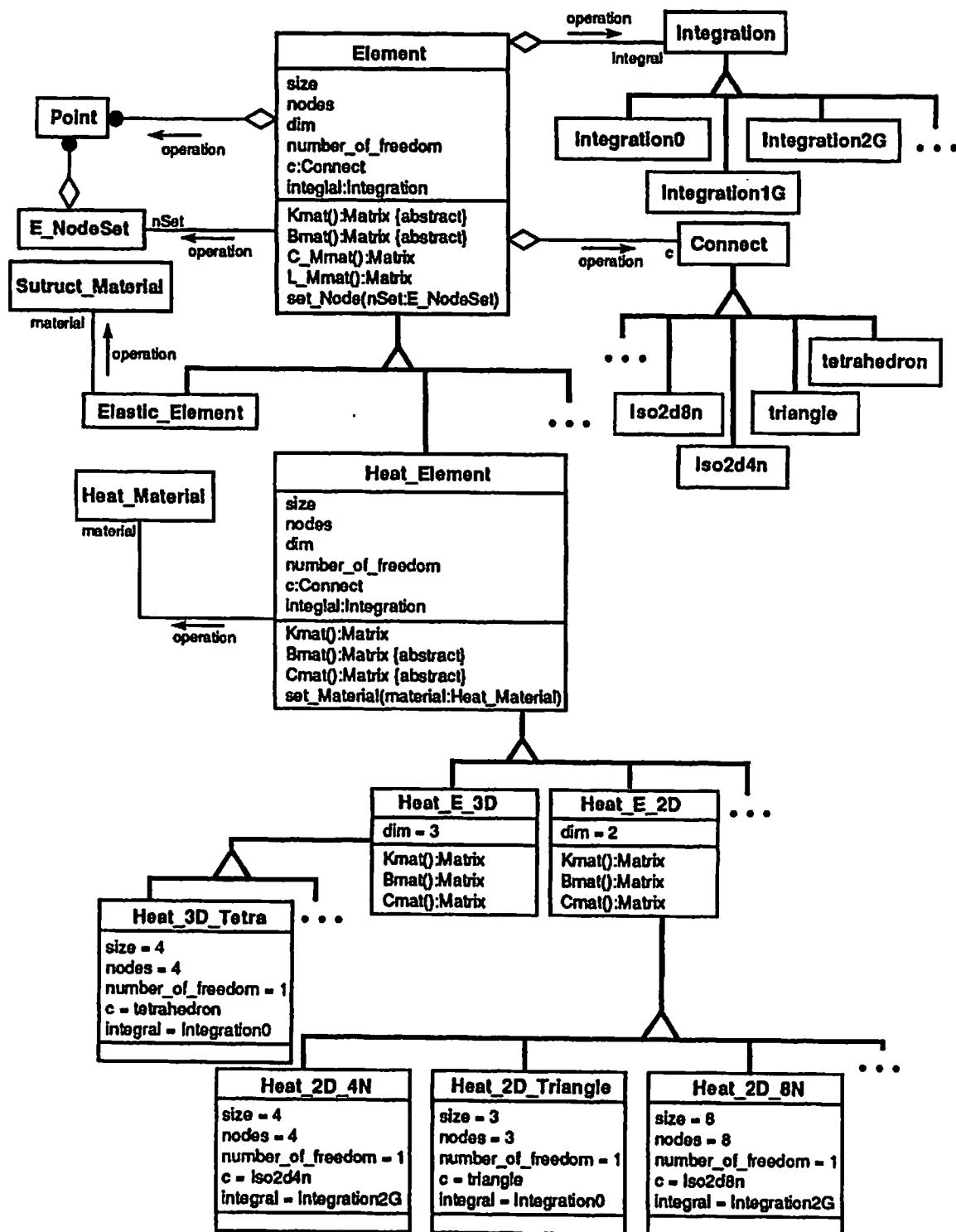


Fig.5.3(b) Object Model of Class Element (Heat_Element)

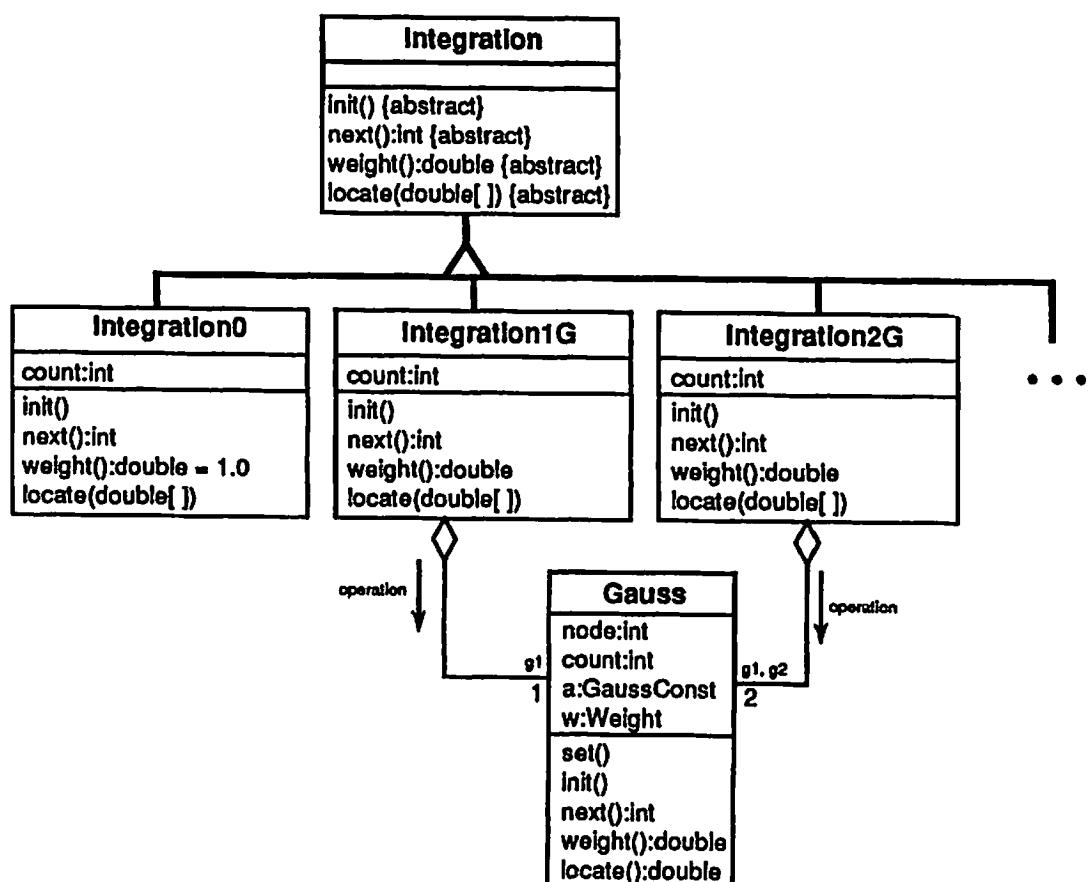


Fig.5.4 Object Model of Class Integration

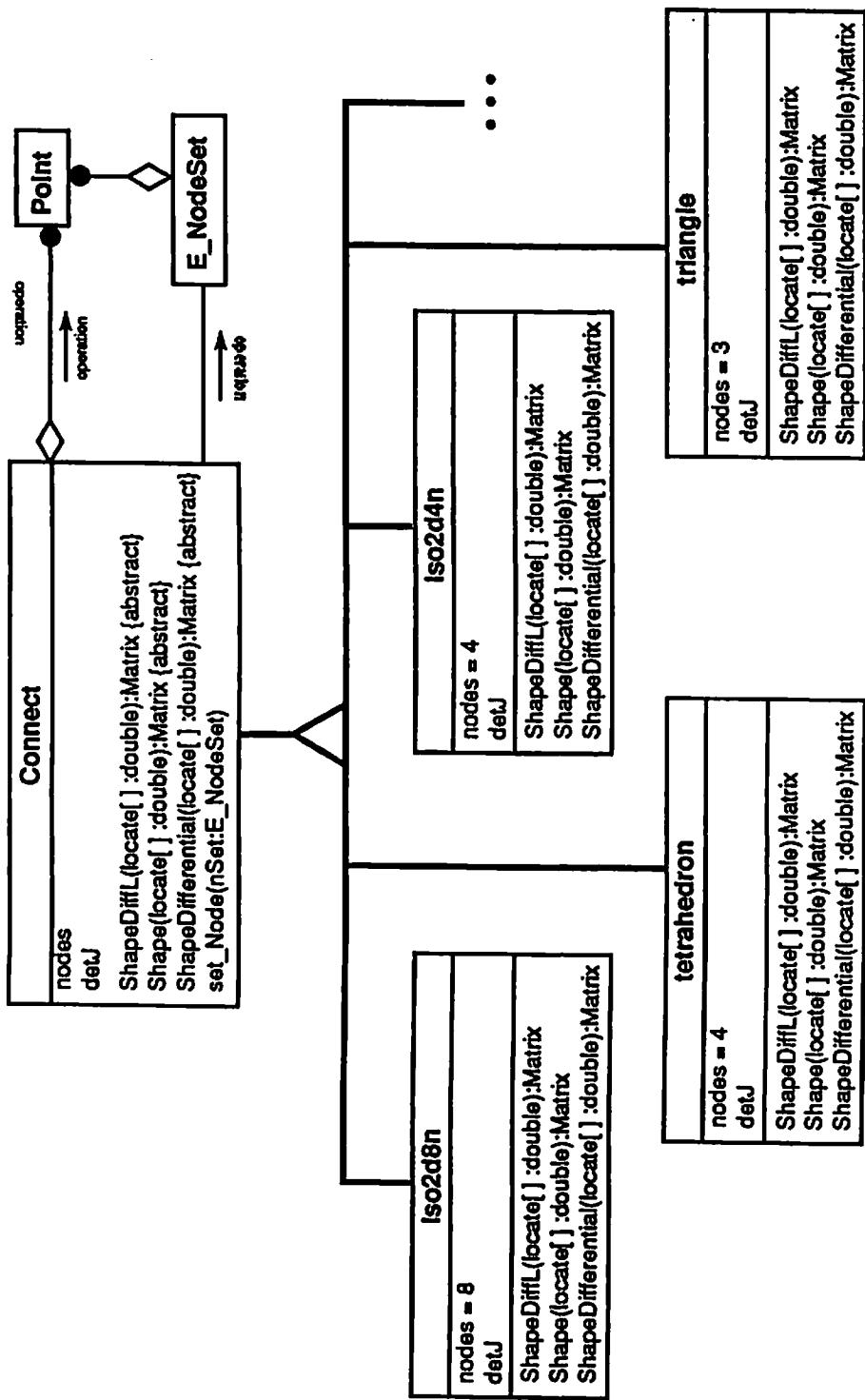


Fig.5.5 Object Model of Class Connect

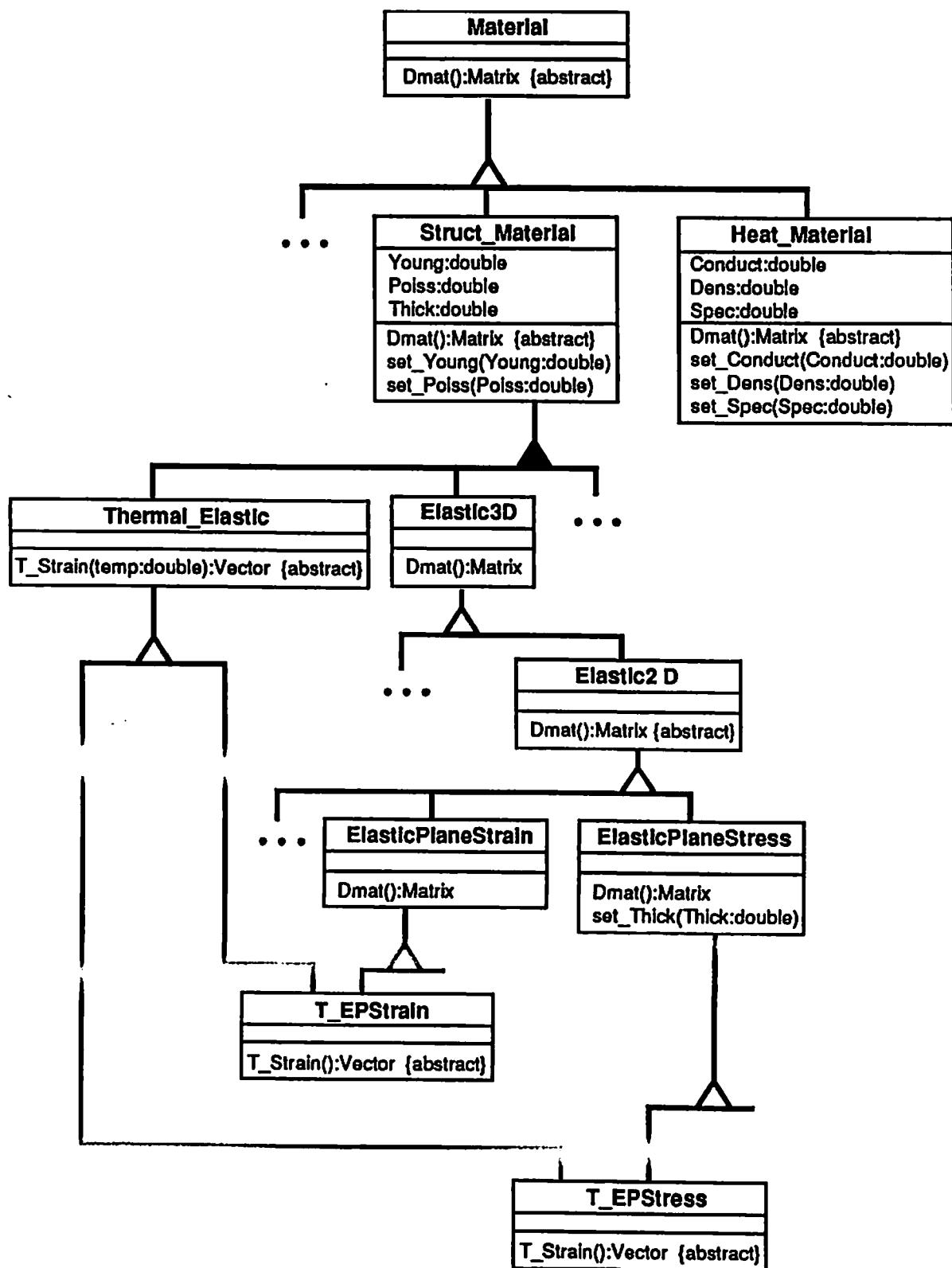


Fig.5.6. Object Model of Class Material

```
Matrix Element :: Kmat() {
    Matrix K(size,size);
    Matrix B,D;
    D = material->Dmat();
    detJ = 0.0;
    do {
        B = Bmat();
        K += c->DetJ() * integral->weight() * trans(B) * D * B ;
        detJ += c->DetJ();
    }
    while (integral->next());
    return K;
}
```

Fig.5.7 Implementation of Kmat()

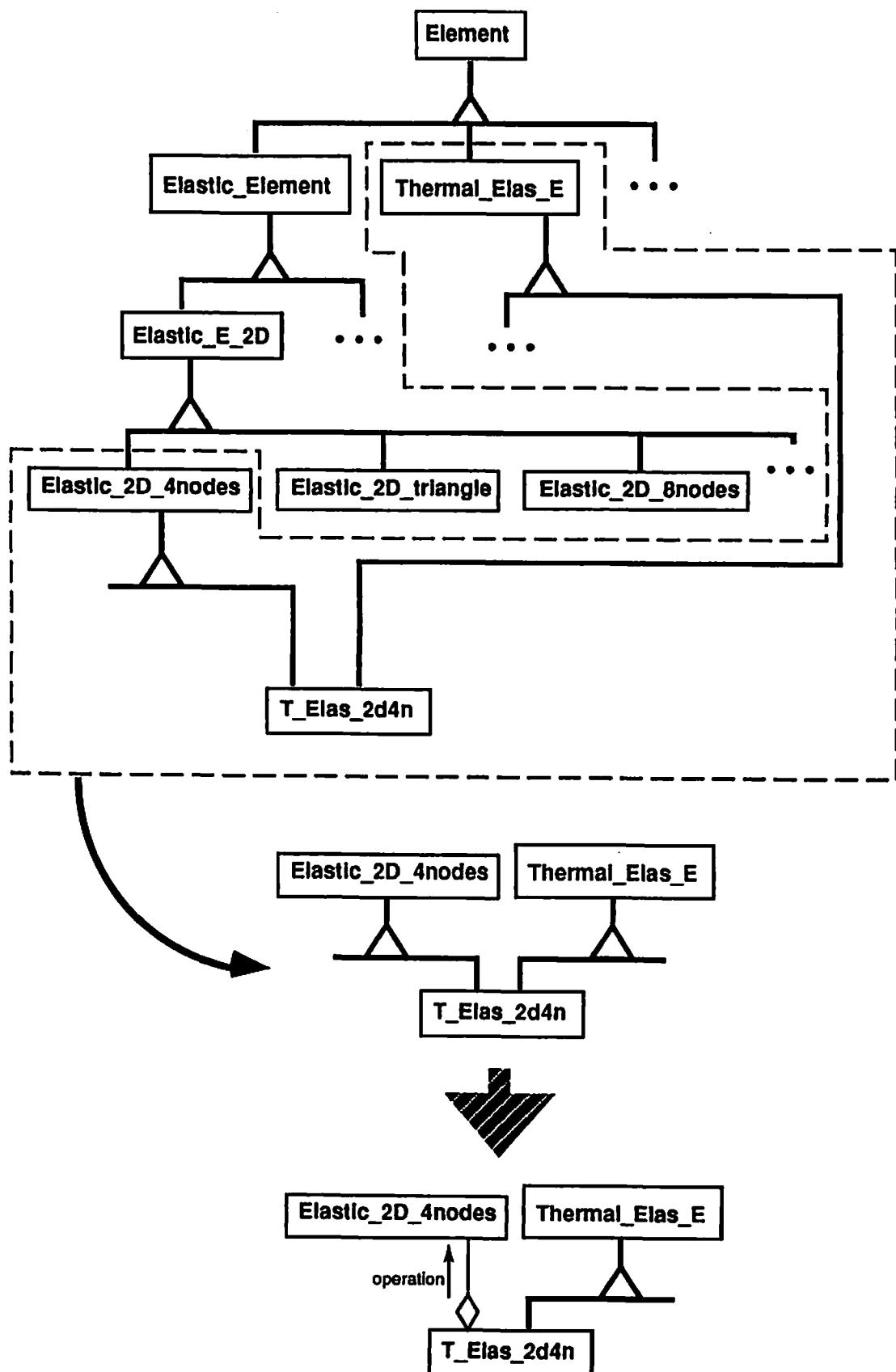


Fig.5.8 Using Delegation instead of Inheritance

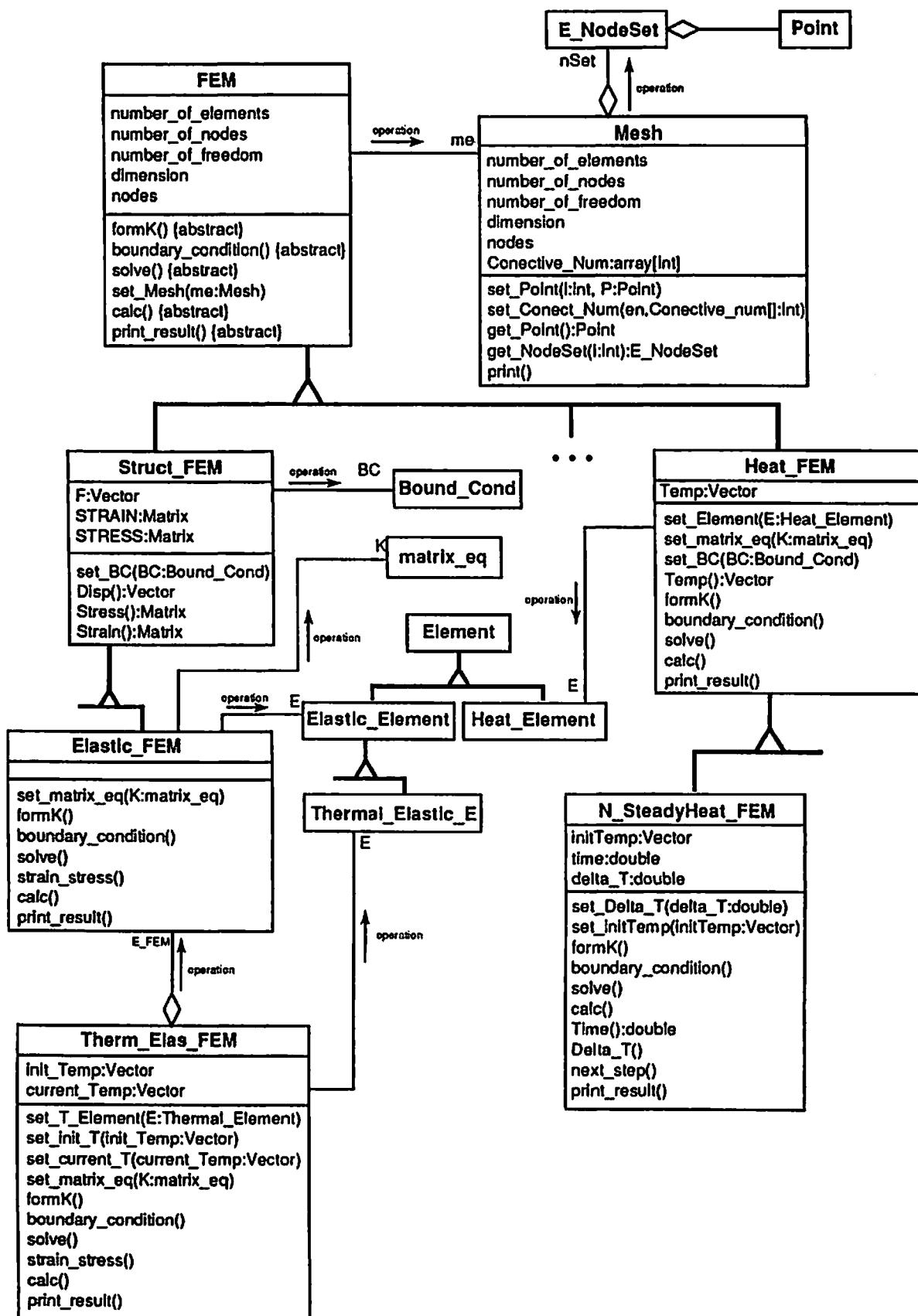
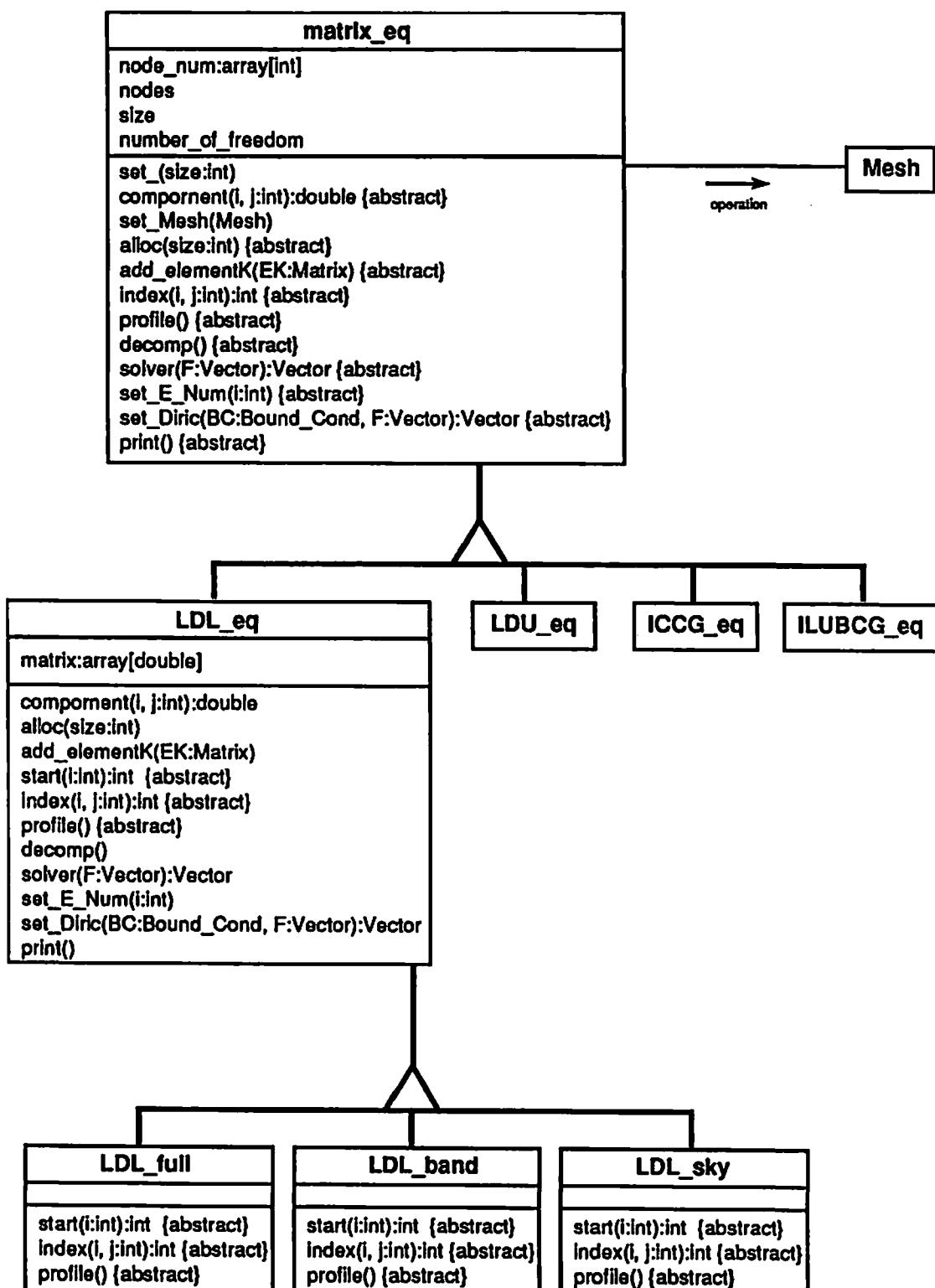
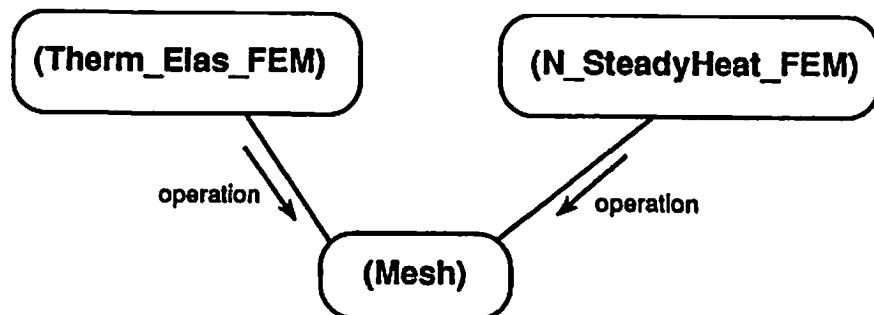
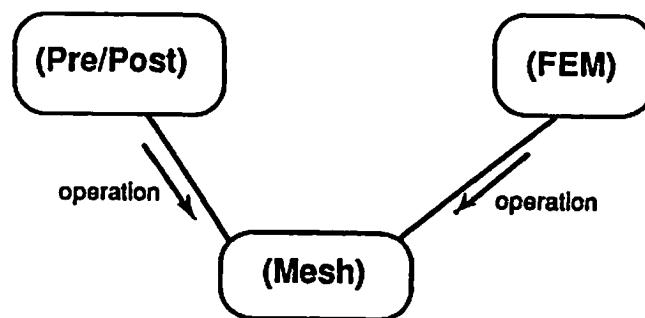


Fig.5.9 Object Model of Class FEM

Fig.5.10 Object Model of Class `matrix_eq`



(a) Reference from EEM



(a) Reference from Pre/Post

Fig.5.11 Reference of Mesh Object

```
// Definition of class Calc_Elastic;
// (User's define class);

class Calc_Elastic : public Analy_Only{
protected:
    Elastic_FEM* e_fem = 0;
    Struct_Element* E = 0;
    Struct_Material* material = 0;
    Mesh* me = 0;
    matrix_eq* K = 0;
    Bound_Cond bc;

    void in_data();
    void calc();
    void out_data();

public:
    Calc_Elastic(Struct_Element*, Struct_Material*, matrix_eq* );
    ~Calc_Elastic();
};
```

Fig.5.12(a) Definition of Class Calc_Elastic

```

// Implementation of class Calc_Elastic
//

#include "Calc_Elastic.h"

Calc_Elastic::Calc_Elastic(Struct_Element* elem, Struct_Material* m, matrix_eq* eq) {
    E = elem;
    material = m;
    K = eq;
}

Calc_Elastic::~Calc_Elastic() {
    if(me) delete me;
    if(e_fem) delete e_fem;
}

void Calc_Elastic::in_data(){
    int num_of_nodes, num_of_elem;
    cin >> num_of_nodes;
    cin >> num_of_elem;

    Create Mesh
    me = new Mesh(E->Dim(), num_of_elem, num_of_nodes, E->Nodes());

    double young, poiss;
    cin >> young;
    cin >> poiss;

    material->set_Young(young);
    material->set_Poiss(poiss);

    E->set_Material(material);

    me->input_data();
    bc.input_data();
    e_fem = new Elastic_FEM Create Elastic_FEM Object
}

e_fem->set_Mesh(me);
e_fem->set_Element(E);
e_fem->set_matrix_eq(K);
e_fem->set_BC(&bc);
}

void Calc_Elastic::calc(){ e_fem->calc(); }

void Calc_Elastic::out_data(){
    me->print();
    bc.print();
    cout << "\n\n ***** result ***** \n\n";
    e_fem->print_result();
}

```

Fig.5.12(b) Implementation of Class Calc_Elastic

```
// Definition of class Calc_Heat;
// (User's define class);

class Calc_Heat : public Analy_Only{
protected:
    SteadyHeat_FEM* sh_fem = 0;
    Heat_Element* E = 0;
    Heat_Material* material = 0;
    Mesh* me = 0;
    matrix_eq* K = 0;
    Bound_Cond bc;

    void in_data();
    void calc();
    void out_data();

public:
    Calc_Heat(Heat_Element*, Heat_Material*,
matrix_eq*);
    ~Calc_Heat();
};
```

Fig.5.12(a) Difinition of Class Calc_Heat

```

// Implementation of class Calc_Heat
//

#include "Calc_Heat.h"

Calc_Heat::Calc_Heat(Heat_Element* elem, Heat_Material* m, matrix_eq* eq) {
    E = elem;
    material = m;
    K = eq;
}

Calc_Heat::~Calc_Heat() {
    if(me) delete me;
    if(sh_fem) delete sh_fem;
}

void Calc_Heat::in_data(){
    int num_of_nodes, num_of_elem;
    cin >> num_of_nodes;
    cin >> num_of_elem;

    me = new Mesh(E->Dim(), num_of_elem, num_of_nodes, E->Nodes()); Create Mesh

    double cond, spec;
    cin >> cond;
    cin >> spec;

    material->set_Conduct(cond);
    material->set_Spec(spec);

    E->set_Material(material);

    me->input_data();
    bc.input_data(); Create SteadyHeat_FEM Object

    sh_fem = new SteadyHeat_FEM;

    sh_fem->set_Mesh(me);
    sh_fem->set_Element(E);
    sh_fem->set_matrix_eq(K);
    sh_fem->set_BC(&bc);
}

void Calc_Heat::calc(){ sh_fem->calc(); }

void Calc_Heat::out_data(){
    me->print();
    bc.print();
    cout << "\n\n ***** result ***** \n\n";
    sh_fem->print_result();
}

```

Fig.5.13(b) Implementation of Class Calc_Heat

```

// Main for Elastic Calculation;
// ;
#include "Calc_Elastic.h"

main () {

    Struct_Element* E = new Elastic_2D_4N;
    Struct_Material* m = new ElasticPlaneStrain;
    matrix_eq* K = new LDL_sky;

    Calc_Elastic* cal = new Calc_Elastic(E, m, K);

    cal->run();

    delete E;
    delete m;
    delete K;
    delete cal;
}

```

(a) Main for Elastic Calculation

```

// Main for Heat Calculation;
// ;

#include <stream.h>
#include "matrix_eq.h"
#include "LDL_eq.h"
#include "LDL_full.h"
#include "LDL_band.h"
#include "LDL_sky.h"
#include "Calc_Heat.h"

main () {

    Heat_Element* E = new Heat_2D_4N;
    Heat_Material* m = new Heat_2D;
    matrix_eq* K = new LDL_sky;

    Calc_Heat* cal = new Calc_Heat(E, m,
                                   m);

    cal->run();

    delete E;
    delete m;
    delete K;
    delete cal;
}

```

(a) Main for Heat Calculation

Fig.5.14 Example of FEM Calculation

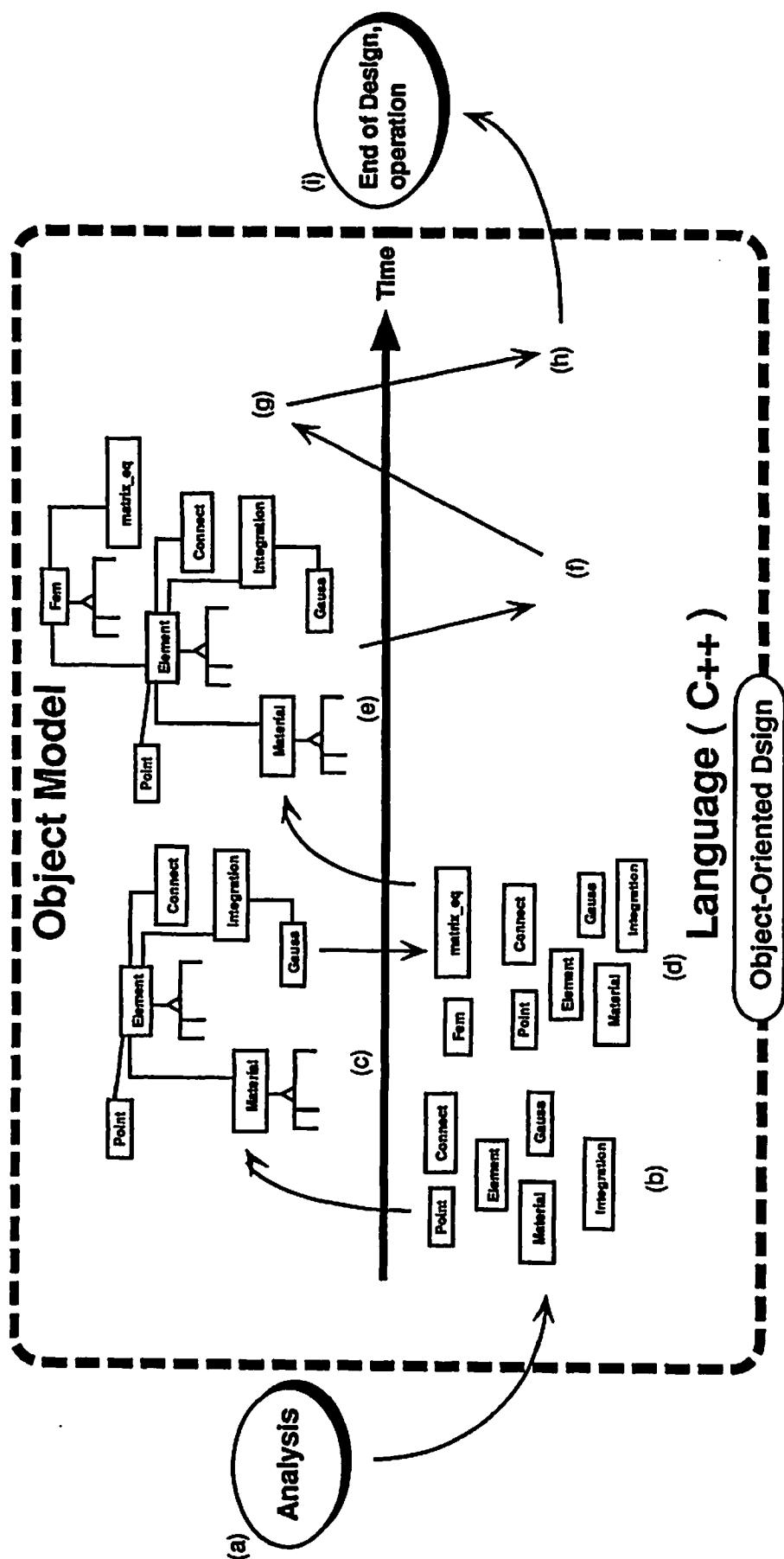


Fig.5.15 Present Process of OOD

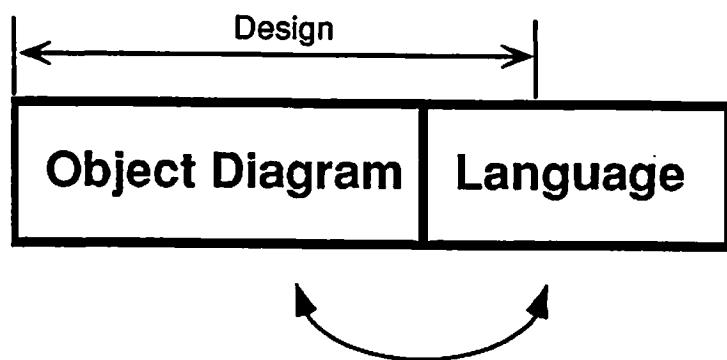


Fig.5.16 Field of OOD

第6章 有限要素解析システムの構築

6. 1. 热弹性解析

ここでは簡単な热弹性解析システムを構築することによって有限要素法クラスライブラリを用いた解析プログラムの構築について検討する。

热伝導解析、热弹性応力解析、その結果を表示するウィンドウの制御を行なう簡単なシステムを考えることとし、システムのState Diagramは Fig.6.1 のようになる。データ入力後、ボタンパネルを表示しアイドル状態に入る。パネル上のボタンを押すことにより計算、結果の表示などを制御する。State Diagramはシステムの動的な状態を表現することができ、また、このようなプログラムの機能をソースだけではなくドキュメントとして残すことができる。

従来の数値解析プログラムは一般的には

(データの入力) → (計算) → (結果の出力)

の手順しか存在せず、プログラム構造は静的なものである。これに対しウィンドウシステムのようなイベント駆動で進行するプログラムでは常に動的な状態が存在する。有限要素解析におけるプレ／ポスト処理は、このようなウィンドウを有するのが普通である。動的なプログラムは多様な状態を含むことになりプログラムが複雑にならざるをえない。それゆえ現在の研究ではプレ／ポストは解析本体とは区別されて研究されるのが普通である。しかしながらこれらのプログラムはプレ／ポストではなく本来は全てが一つの解析システムとなっているべきである。本研究で構築したものは”システム”と呼ぶには恥ずかしいほど単純なものである。しかし、こ

のようなものでも素人が作るにはかなりの労力を必要とする。グラフィック表示は数値解析が乗り越えねばならない壁でもあり、研究者を解析本体に専念させるためにもプログラム部品群が必要である。本研究はこれらの部品を蓄えるための環境、枠組みを実現するためのものである。

6. 2. 将来の解析プログラム

有限要素解析は汎用プログラムによって行なわれることが多い。汎用プログラムは多機能であるはあるがデータ入力でさえ多大な労力を必要とする欠点も持っている。これらを解消するためにプレ／ポスト処理の研究も進められているわけであるが、解析プログラム本体の保守低減については決定的な対策法は存在しないと言つて過言でない。研究の現場などで行なわれる有限要素解析は、汎用プログラムの一部の機能だけしか必要としないのがほとんどであり、不必要的機能に悩まされる結果となっている。これは1つのプログラムに機能を集中したことが原因であり、アプリケーションの肥大化に伴ない使用法が複雑になってくるからである。可能ならば必要な機能だけを取り出してプログラムを構築できる方が効率的に計算を行なうことができる。将来はFig.6.2に示すようにアプリケーションに機能を集中するではなく、周りに機能を分散させ、必要に応じてそれらを部品として組み立て、小さいが必要最小限の機能を満たす個人用のプログラムを自由に構築できる環境に移行して行くべきである。

汎用プログラムのような現在のアプリケーションは“人間がプログラムに合わせること”を強要してくれる。これに対し、個人用プログラムを開発する環境を作ることは“プログラムを人間に合わせること”に等しい。本研究は数値解析におけるこのような環境の実現のためのプロセスである。本研究のような環境についての地道な試みを続けることで、数値解析の専門家（コンピュータの専門家ではない）が計

算を行なうためにプログラミングを行なう必要が無くなる時代が到来することも夢ではないはずである。

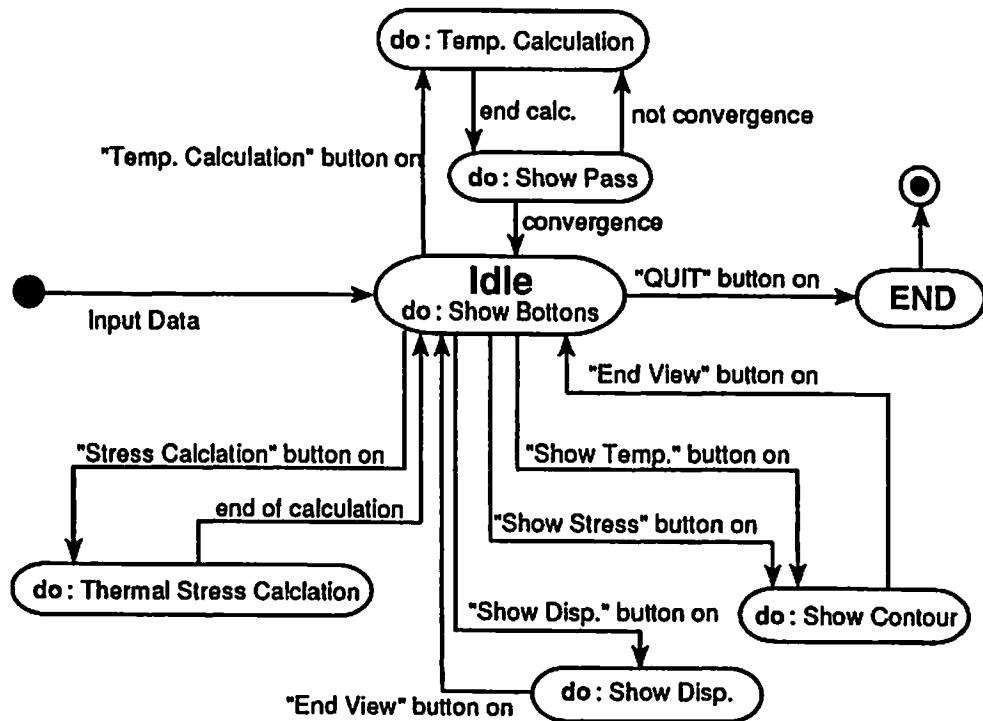


Fig.6.1 State Model for FE Analysis System

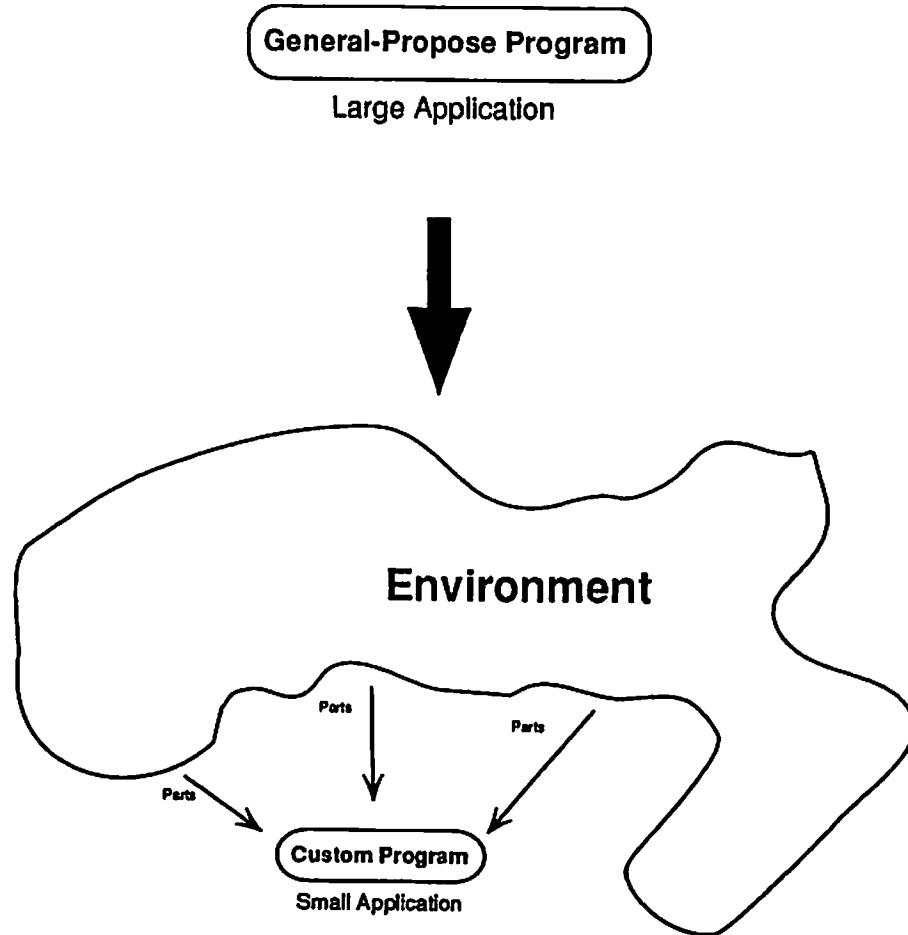


Fig.6.2 Future Environment for Numerical Analysis

第7章 結論

本研究では既存のオブジェクト指向方法論（OMT）と言語（C++）を用いて、有限要素法プログラムの部品化を行なうための方法論とその実践について論じた。以下に得られた結論をまとめると。

- (1) 有限要素法クラスライブラリ構築にOMTを適用し、数値解析プログラム構築に対するOMTの有用である部分と不十分な点を明かにした。
- (2) C++での実装を行なうことにより、数値解析におけるオブジェクト指向設計は下流工程から上流へのアプローチが適していることを明らかにした。
- (3) 本システムで使用するための有限要素法クラスライブラリの基本設計を得た。
- (4) C++による有限要素法クラスライブラリのプロトタイプを開発した。
- (5) 今日の数値解析のためのプログラミング環境における問題点や未解決部分を明らかにし、その解決策について検討した。以下に問題点を示す。
 - ・手続き型による開発環境は低効率であるにもかかわらずそれらが認識されていない。
 - ・それぞれの分野が高度な技術になっていることから、計算力学とソフトウェア工学との間では情報交換がすみやかに行なわれておらず、大きな解離が存在
 - ・汎用プログラムなどでは多数の機能を1つのアプリケーションの中に集中している。このようなプログラムは複雑になり、保守が困難になるだけでなく、ユーザが使用法

を把握することさえ困難にしている。将来は機能は周りの環境に分散し、小さいが必要最小限の機能を満たす個人専用のアプリケーションをユーザが自由に構築できるようにならなければならない。

- ・ウインドウシステムなどのVisual Systemは複雑なプログラムとなり、全てのユーザが恩恵を受けるのは難しい状況となっている。CPU速度の向上とともに数値解析プログラムでもVisual Systemを簡単に組み込める必要とする時代が必ず到来する。Visual Systemを使いこなすためには、オブジェクト指向、プログラムの部品化への対応は不可欠である。

参考文献

- [1] 原田実 監修, CASEのすべて, オーム社, 1991.
- [2] 竹下亨, ソフトウェアの保守・再開発と再利用, 共立出版, 1992.
- [3] 酒井博敬, 堀内一, オブジェクト指向入門, オーム社, 1989.
- [4] 吉田弘一郎, TURBO C++/BORLAND C++ によるオブジェクト指向狂詩曲, 技術評論社, 1992.
- [5] 木暮裕明, オブジェクト指向のすべて '90年代のプログラマの必須知識OOPSをマスターする, インターフェース, CQ出版, 1990.
- [6] S.C.Dewhurst, K.T.Stark, *Programming in C++*, Prentice-Hall International, 1989. (古山裕司 訳, C++言語入門, アスキー, 1990.)
- [7] 杉原敏夫, C++とオブジェクト指向, 工業図書, 1991.
- [8] 門内淳, 赤堀一郎, C++プログラミング, 日本ソフトバンク, 1989.
- [9] 工藤智行, 基礎編 CユーザーのためのC++入門, 技術評論社, 1992.
- [10] B.J.Cox, A.J.Novobilski, *Object-Oriented Programming : An Evolutionary Approach 2nd ed*, Addison-Wesley Publishing Company, 1991(1986). (松本正雄 訳, オブジェクト指向のプログラミング ソフトウェア再利用の新しい方法 改訂第2版, トッパン, 1992.)
- [11] 小林史典, オブジェクト指向とSmalltalk, CQ出版, 1989.
- [12] B.Meyer, *Object-Oriented Software Construction*, Prentice-Hall International, 1988. (酒匂寛, 酒匂順子 訳, Object-Oriented Software Construction オブジェクト指向入門, アスキー, 1990.)
- [13] J.Rumbaugh, M.Blaha, W.Premelani, F.Eddy, W.Lorensen, *Object-Oriented Modeling and Design*, Prentice-Hall International, 1991. (羽生田栄一 訳,

- オブジェクト指向方法論OMT モデル化とデザイン, トッパン, 1992.)
- [14] P.Coad, E.Yourdon, *Object-Oriented Analysis*, Prentice-Hall International, 1991.
- [15] 佐藤正美, CASEツール 機能解説と活用のノウハウ, ソフト・リサーチ・センター, 1989.
- [16] B. W. R. Forde, R. O. Foschi, S. F. Stiemer, *Object-oriented Finite Element Analysis*, Comput. Struct., 34(3), pp.355-374, 1990.
- [17] S.-P.Scholz, *Elements of An Object-Oriented FEM++ Program in C++*, Comput. Struct., 43(3), pp.517-529, 1992.
- [18] R. I. Mackie, *Object-Oriented Programming of The Finite Element Method*, International Jounal for Numerical Methods in Engineering, 35, pp.425-436, 1992.
- [19] P. Remy, B. Devloo, J. S. R. A. Filho, *An Object-Oriented Approach to Finite Element Programming (Phase I): A System Independent Windowing Environment for Developing Interactive Scientific Programs*, Advances in Engineering Software, 14, pp.457-467, 1992.
- [20] Y. Dubois-Pèlerin, T. Zimmermann, P. Bomme, *Object-Oriented Finite Element Programming Concepts*, New Advances Computational Structural Mechanics, pp.457-467, 1992.
- [21] T. Zimmermann, Y. Dubois-Pèlerin, P. Bomme, *Object-Oriented Finite Element Programming: I. Governing Principles*, Computer Methods in Applied Mechanics and Engineering, 98, pp.361-397, 1992.
- [22] Y. Dubois-Pèlerin, T. Zimmermann, P. Bomme, *Object-Oriented Finite Element Programming: II. A Prototype Program in Smalltalk*, Computer

- Methods in Applied Mechanics and Engineering, 9 8, pp.361-397, 1992.
- [23] 関東康祐, 赤星保浩, 青佐俊彦, 三村泰成, 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, 機械学会69期全国大会講演論文集, No910-62A, pp.636-637, 1991.
- [24] 青佐俊彦, オブジェクト指向プログラミングによる有限要素解析ライブラリの開発, 豊橋技術科学大学エネルギー工学系卒業論文, 1992.
- [25] 関東康祐, 赤星保浩, 三村泰成, 青佐俊彦, オブジェクト指向型有限要素開発システムの開発, 構造工学における数値解析法シンポジウム論文集第16巻, pp.601-606, 1992.

謝辞

本研究を進めるうえで、材料力学研究室の原田昭治教授、荒井貞夫客員教授、野田尚昭助教授、秋庭義明助教授、には多大な御助言をいただき、そして直接ご指導いただいた赤星保浩助教授には多忙であるにもかかわらず、貴重なお時間と知識をお貸しいただき心から感謝いたします。

研究活動を続けるうえで福島良博助手、川原忠幸技官、高瀬康技官には多大なご助力をいただきました。また、研究室の先輩である井上秀行氏には幾度となく研究についての相談にのっていただきました。

オブジェクト指向、C++によるプログラミングについて未熟な知識しか持ち合わせてていなかった著者に対し関東康祐助教授、青佐俊彦氏（豊橋技術科学大学エネルギー工学系）には貴重な資料と知識をお貸しいただきました。

最後になりましたが、材料力学研究室の諸氏の励ましのおかげで最後まで研究を終えることができました。

以上の方々にこの場をお借りして心から感謝の意を表します。