NC-ZJ1064 97-001 Vol.1

限定資料

分置01

本資料は 11 年 7 月25 日付けで
登録区分変更する。 [技術展開部技術協力課]

地層処分ヴァーチャル・エンジニアリング・システムの開発（Ⅰ）

対話技術の研究

（動力炉・核燃料開発事業団　研究委託内容報告書）

技術資料

開示区分	レポートNo.	受領日
3T	J1064 97-001 Vol.1	1998.4.14.

この資料は技術管理室保存資料です
閲覧には技術資料閲覧票が必要です
動力炉・核燃料開発事業団　技術協力部技術管理室

１９９７年３月

日商岩井株式会社

この資料は，動燃事業団の開発業務を進めるため，特に限られた関係者だけに開示するものです。ついては複製，転載，引用等を行わないよう，また第三者への開示又は内容漏洩がないよう管理して下さい。また今回の開示目的以外のことには使用しないよう特に注意して下さい。

本資料についての問合せは下記に願います。
〒107　東京都港区赤坂１－９－13
動力炉・核燃料開発事業団
技術協力部　技術管理室

限　　定　　資　　料
PNC ZJ1064 97-001 Vol. 1
１９９７年３月

地層処分バーチャル・エンジニアリング・システムの開発（Ⅰ）

—対話技術の研究—

井ノ口　稔雄１）、 Larry BAIRD２）、
G.MEGSON３）

要　旨

対話システム技術の研究開発では、地層処分バーチャル・エンジニアリング・システムと利用者との対話（情報交換）技術である情報の視覚化表示、データ入力の為のボタン、プルダウン・メニュー等のユーザー・インターフェイス技術の予備設計検討を行った。作業にあたり、個別に開発する技術の評価が必要な項目については評価用試作プログラムを開発し評価を行った。

評価プログラムの試作開発にあたっては、従来より研究が進められている、処分システムの性能評価に関する方法論を基盤としたシナリオ解析や、関連する個別現象の連成シミュレーション等の数値計算を効率的に統合して実行し、かつ、作図、作表、動画化等の後処理にかかわる操作を解かりやすく効率的に行えるよう考慮して検討した。

研究開発作業の結果、対話機能は地層処分バーチャル・エンジニアリング・システムの重要な機能であると判断された。又、開発にあたっては、オブジェクト指向プログラミング技術、ネットワーク並列処理技術、MIT/X-Windowシステムに関連する技術、等、多くの新しいソフトウエア開発技術が要求されるものと判断した。これらの技術開発課題は、次年度より開始する設計開発作業により詳細に開発蓄積することが必要である。

本報告書は日商岩井株式会社が動力炉・核燃料開発事業団の委託により実施した研究内容結果である。
契約番号：080D0330
事業団担当部課室および担当者：環境技術開発推進本部　処分研究グループ
梅木　博之

１）日商岩井株式会社
２）Global Technology Assosicates, Inc. U.S.A.
３）Department of Computer Science. University of READING. U.K

COMMERCIAL PROPRIETARY
PNC ZJ1064 97-001 Vol. 1
MARCH 1997

Developmemnt of the Virtual Engineering System for High Level Radioactive Waste Geological Disposal Performance Assessment

Graphic User Interface Development

Toshio INOGUCHI 1), Larry BAIRD 2),
G.MEGSON 3)

Abstract

A Graphic User Interface (GUI) is a "window" to connect computer systems and users. It needs to be easy for users to operate a system and to understand the information which the system displays. As research of in the development of a Interactive Graphic User Interface for the Virtual Engineering System for Geological Isolation, the following fields are explored; 1) data visualization methodologies, 2) interactive GUI design, 3) peripheral technology.

During the work, several software prototypes were developed for evaluation and study. Although the total system architecture is not designed for practical use, the prototype was constructed to smoothly and logically achieve scenario analysis for performance assessment and simulation of natural phenomena via the GUI. As a result of the research it is found that, since the system handles a range of operations and information, a well developed interactive GUI is essential for the Virtual Engineering System for Geological Isolation. It is also considered that further development research will be required into technologies for Object Oriented Programming, network parallel processing and MIT/X-Windows. These technologies will be acquired during the detailed development study starting from next fiscal year.

Work perfoormed by Nissho Iwai corporation under contract with Power Reactor and Nuclear Fuel Development Corporation.
PNC Liaison : Radioactive Waste Management Project, Isolation System Research Program
Hiroyuki Umeki

1) NISSHO IWAI corporation
2) Global Technology Associates, Inc. U.S.A.
3) Department of Computer Science. University of READING. U.K

【 目 次 】

1 はじめに …… 1
1.1 対話システム開発技術基盤の検討 …… 2
1.2 ユーザー・インターフェイスの試作 …… 3
1.3 ユーザー・インターフェイスの高度化検討 …… 10
2 対話システム開発技術基盤の検討 …… 12
2.1 ＵＮＩＸ系対話システム開発技術評価 …… 14
2.1.1 Ｘプログラミング開発技術 …… 14
2.1.2 IRIS OpenGLプログラミング開発環境 …… 21
2.2 ＰＣ系対話システム開発技術評価 …… 40
2.2.1 新しいコモンコントロール …… 40
2.2.2 新しいコモンダイアログボックス …… 44
2.2.3 長いファイル名のサポート …… 44
2.2.4 ショートカット …… 44
2.2.5 ファイルビューア …… 45
2.2.6 ユーザーインターフェイス拡張 …… 45
2.3 アップル・マッキントッシュ対話システム開発技術 …… 46
2.3.1 イベント駆動型プログラミング …… 47
2.3.2 リソース …… 50
2.3.3 Toolbox …… 53
2.3.4 オペレーティングシステム …… 55
2.3.5 システムソフトウェア …… 56
3 ユーザーインターフェイスの試作 …… 60
3.1 ユーザーインターフェイス設計に考慮した事項 …… 60
3.1.1 ディスプレイ画面上の形の配列 …… 60
3.1.2 空間的メタファ、ディスプレイ、ツールのデザイン …… 66
3.2 マルチ・コンピュータ環境構築のために採用した画面制御技術 …… 71
3.2.1 ウィンドウ・システム …… 71
3.2.2 ウィンドウ・システムの概観 …… 73
3.2.3 ウィンドウ …… 77
3.2.4 メニュー …… 81
3.2.5 コントロールとコントロール・パネル …… 86
3.2.6 マウス／キーボード・インターフェイス …… 90
3.2.7 長所と短所 …… 100
3.3 マルチ・エージェント／マルチ・プラットホームの対話機能 …… 102
3.3.1 システム操作とマルチ・エージェント処理 …… 104
3.3.2 マルチ・エージェント・ユーザーインターフェイス …… 109
3.3.3 階層構造十階層構造によるユーザーインターフェイス …… 115
3.3.4 逆階層構造十階層構造によるユーザーインターフェイスの構造手順 …… 119
3.3.5 逆階層構造十階層構造方式の利点 …… 126
3.3.6 協調計算における逆階層構造十階層構造の導入について …… 127

3.4 試作基本対話機能 …… 130
3.4.1 操作用ユーザーインターフェイス …… 130
3.4.2 試作対話システム制御機能概念 …… 136
4 ユーザー・インターフェイスの高度化検討 …… 148
4.1 地層処分性能評価シミュレーションとシステムの操作 …… 148
4.1.1 安全性評価作業の基本的な進め方 …… 148
4.1.2 システム操作から見た評価期間の考え方 …… 149
4.1.3 システム操作から見た多重バリア方式の考え方 …… 150
4.1.4 システム操作から見た評価シナリオの考え方 …… 151
4.1.5 システム操作から見た評価モデルの考え方 …… 151
4.1.6 システム操作から見たパラメータの不確定性について …… 153
4.1.7 システム操作から見た総合評価の考え方 …… 154
4.2 システム操作から見た作業の不確かさについて …… 155
4.3 対話機能から仮想環境へ（ヴァーチャル・エンジニアリングの将来） …… 158
4.3.1 複雑系シミュレーション …… 158
4.3.2 マルチエージェント・シミュレーションによる仮想世界の観察 …… 163

1　はじめに

平成８年度は、地層処分ヴァーチャル・エンジニアリング・システムの対話技術について予備的検討と試作を行った。対話技術は、地層処分ヴァーチャル・エンジニアリング・システムを利用して性能評価にかかわる操作を行う場合に、直接の操作の対象となるコンピュータ操作画面を制御するソフトウェアのことである。この操作画面は、メニュー、作図、３Ｄオブジェクト表示、動画表現等コンピュータ・グラフィクスの機能を最適に利用すると供に、地層処分性能評価作業にかかわる一連の操作について、使いやすく、かつ適当と判断される情報の組に整理されて開発する必要がある。従って、地層処分ヴァーチャル・エンジニアリング・システムの対話技術は、コンピュータ・グラフィックス技術、性能評価にかかわる一連の評価フレーム、そして、関連するサブシステムであるシミュレーション・システム、データベース・システム等との効率的なインターフェイスという、性能評価技術一般と広範な情報処理技術の複合技術から組み立てられるものとした。

そのような背景から、本研究作業では具体的に地層処分ヴァーチャル・エンジニアリング・システムの対話プログラムを試作することとした（以下、この対話プログラムをGUI:Graphics User Interfaceと呼称する。）。図１－１に平成８年度に実施した作業を示す。

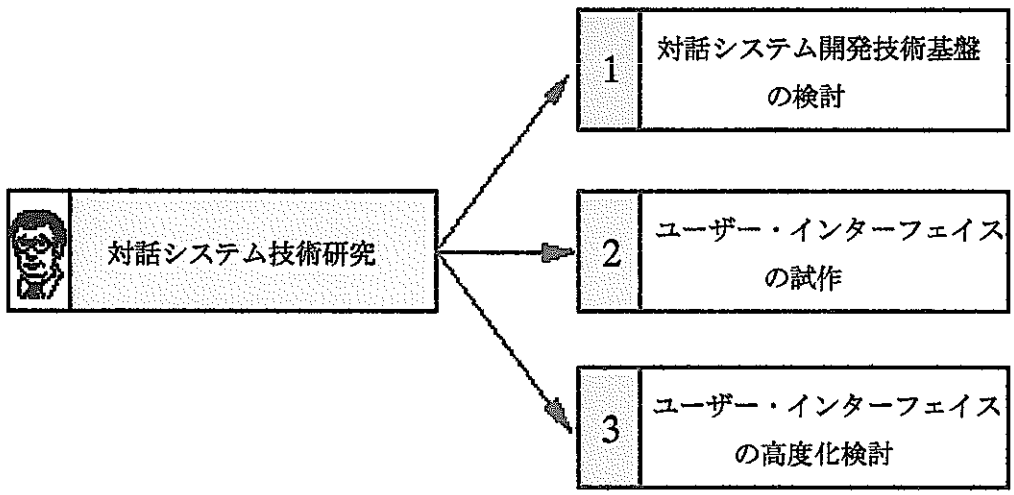

図１－１　平成８年度に実施した作業

試作作業は、視覚化表示、データ入力の為のボタン、プルダウン・メニュー等、エンド・ユーザーと直接情報を交換する機能を受け持つソフトウェアについて実施した。試作したGUIプログラムは、目標となるシステムの機能を画面上に示すとともに、複数に渡るGUIプログラム相互の関連の評価等を行えるように相互の呼び出し機能（コールバック）を組み込むことにより、GUIシステム全体の操作を評価した。GUIの開発にあたっては、従来より研究が進められている処分システム性能評価に関する方法論を基盤としたシナリオ解析や、関連する個別現象シミュレーション等のモデル開発作業を効率的に統合し、作図、作表、動画化等の後処理にかかわる操作を解かりやすく行える操作を合わせて考慮した。

平成8年度の作業では、地層処分ヴァーチャル・エンジニアリング・システム全体の対話環境構築の為の基本概念をプロトタイプとして試作した。試作システムは、“シミュレーション・システム”、“データベース・システム”、“ソフトウェア管理システム”等の各システムで必要となる対話機能及び動画処理等の技術について、ライブラリーやサブシステムとして標準化し、次年度より本格的に開発する、地層処分ヴァーチャル・エンジニアリング・システムの対話環境技術基盤として整理した。

1．1　対話システム開発技術基盤の検討

対話機能から想定する地層処分ヴァーチャル・エンジニアリング・システムは、処分の安全性評価に係わる各種のシナリオについて、計画する処分システムとその環境を構成するオブジェクト、解析コード、検証・確証用試験データ等、複雑に組み合わせ、複雑な現象を数値的に再現（シミュレート）するシステムとした。システム構成面から地層処分ヴァーチャル・エンジニアリング・システムを考えた場合、それは下記に示す機能別に分散したハードウェアをネットワークにより統合し、シミュレーションにおいてはそれらが並列的に協調して動作する、並列ネットワーク分散処理技術により一元統合化されたシステムとして考えた。

シミュレーション用システム（Runtime Visual Simulation System）
操作用システム（Operator Console System）
データベース用システム（Data Base System）
パラメータ・ディスプレイ用システム（Status Display System）
オーディオ・ビデオ用システム（Audio Video System）

対話システム開発技術基盤の検討作業では、地層処分ヴァーチャル・エンジニアリング・システムのGUI機能開発に必要な技術の評価を行なった。検討の結果、計画するシステムが異機種のネットワーク並列処理機能（マシン・パラレル）により統合化されるため、GUI開発環境については完全な一元化はハードウェアをドライブする基本システムの相違等から難しいと考えられた。

従って、システム・ハードウェアをUNIX系とＰＣ系に二分し、UNIX系は、MIT/X11を基盤としたXネットワーク・プロトコールによりグラフィック・ウインドウ制御を統一すると供に、Zバファを画像フレーム・メモリとして有するハードウェアについては、シリコン・グラフィクス社が開発したIRIS 4D/GLプロトコールを採用し、拡張することが現在入手可能な技術として適当であるとした。

また、X11のウインドウ管理については、OSF/Motif xm intrinsicを採用することが適当であるとした。なぜならば、地層処分ヴァーチャル・エンジニアリング・システムのUNIXワークステーションが異機種のマルチ・コンピュータ環境となるため、メーカーが共同して開発したOSF/Motifがシステムの統合化に適しているためである。ＰＣについては、MacintoshとIBM PC系の２種類のアーキテクチュアが市場で入手可能である。地層処分ヴァーチャル・エンジニアリング・システムのＰＣとしては、仕様の公開されているIBM/PC系を採用し、GUI開発環境はUNIXシステムとサーバー・クライアントの構成が容易なアーキテクチュアを提供するWindows95を採用した。

1．2　ユーザー・インターフェイスの試作

ユーザー・インターフェイスの試作作業は、地層処分ヴァーチャル・エンジニアリング・システムのユーザー・インターフェイスについて、操作機能面からの要求を分析するための試作作業とした。作業はプロトタイピング技法を用いて、具体的な対話システムの骨子をプログラミングし、動作条件等を評価しながら進めた。

（1）GUI基本動作概念

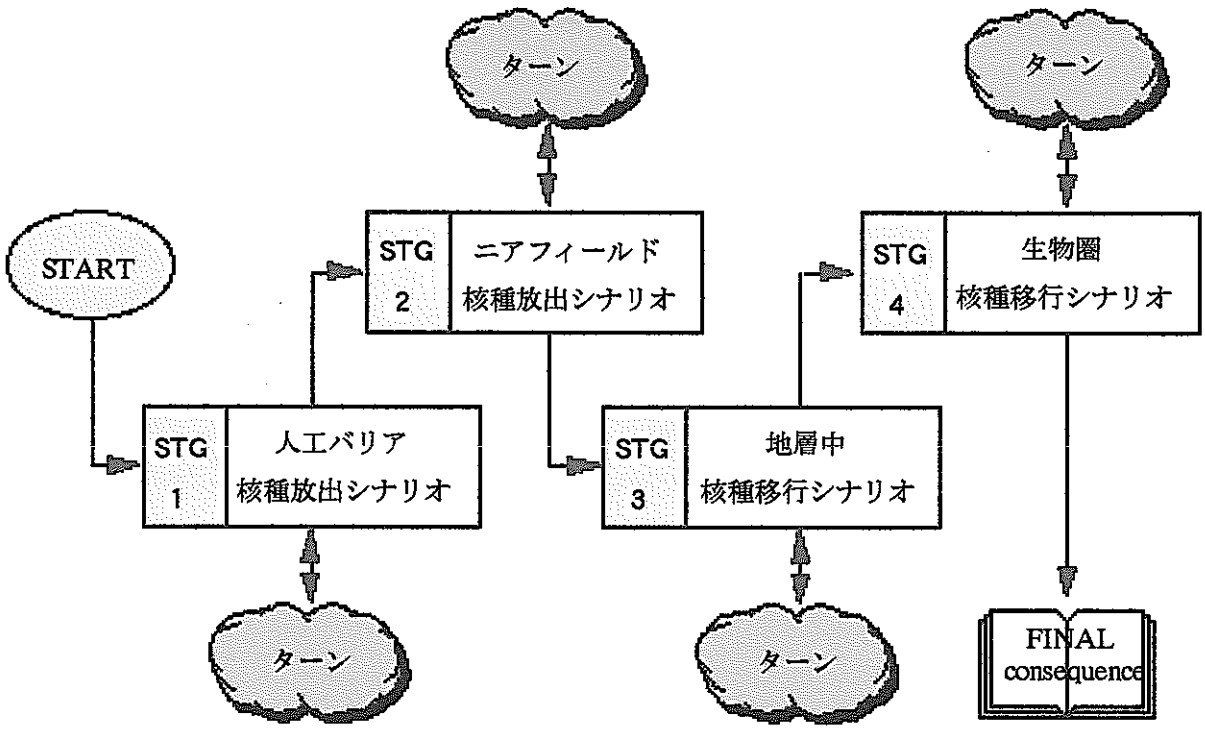

図1．2－1　ステージの概念

GUIプログラムの基本部分を試作するにあたり、図1．2－1に示す**ステージの概念**を試験的に考案した。ステージの概念は、地層処分ヴァーチャル・エンジニアリング・システムのシミュレーション機能を操作する基本的な考え方とした。それは、多重バリア方式を評価するシナリオのサブセットについて分割したものとした。図1．2－1に試作システムの4段階ステージを示す。図1．2－1に示すように、シミュレーションのシナリオのサブセットを多重バリアに従って準備していると考える。これにより、評価はこれらの多重バリアごとの個別事象についてシナリオに沿って行なうものとした。この様な考え方は、全体シナリオから考えた場合、各ステージ間のフィードバックや相互影響を含まない点が指摘されるが、個別の現象解析と全体シナリオ解析の中間に位置する手法として、それらを補足することが期待できるため導入した概念である。

操作にあたっては、このステージを単位として、それに含まれる個別シナリオについてFEP等を参考に評価シミュレーションを行ない、シミュレーション結果をターンという概念で保存することとし

た。各ステージにおいては、ステージ毎のシナリオの組みを**ステージ・シナリオ**と呼称することにし、それらの評価結果を定量的に把握できるよう、単純な定量化指標として**ターンの概念**を試験的に考案した。

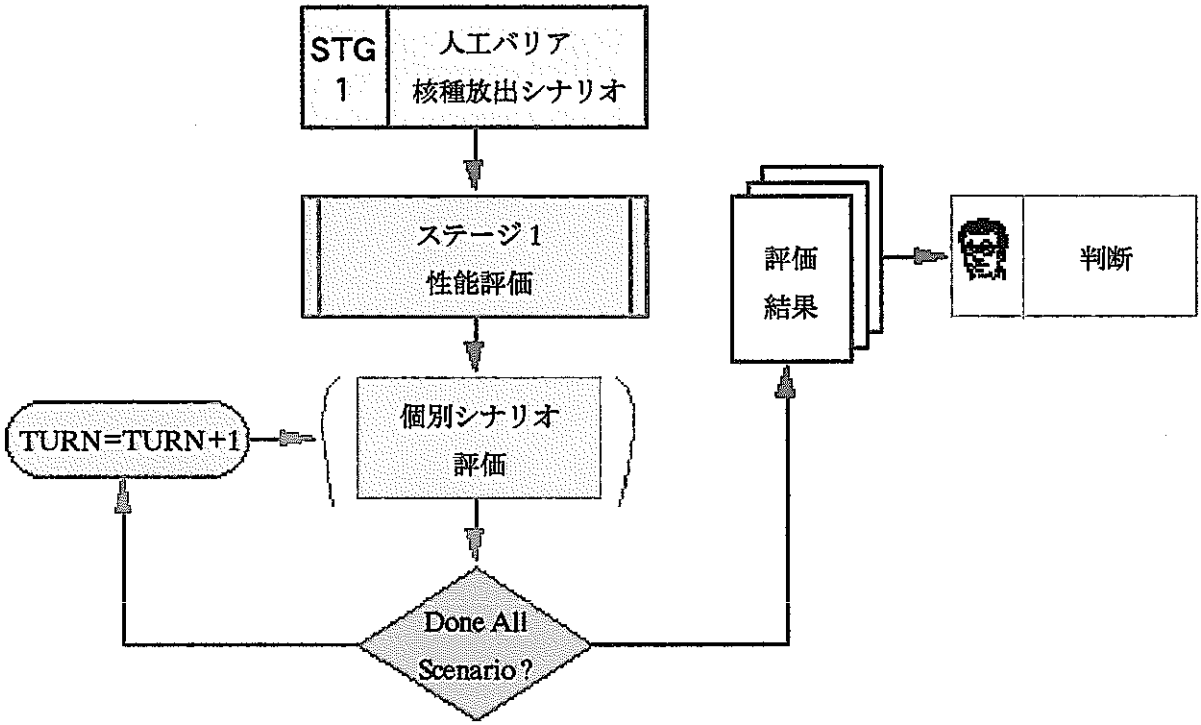

図１．２－２　ターンの概念

ターンは、各ステージにおける評価の指標とした。それは、各ステージの評価シナリオにかかわる評価の満足度を示すひとつの目安である。各ステージは、個別シナリオから構成されるサブ・グループとした。例えば、ステージ１（人口バリアの核種放出シナリオ）においては、ガラス固化体、オーバーパック及びベントナイトについての個別シナリオが、現在整備中の階層ＦＥＰマトリクスから抽出すると１８のサブ・グループに分割される。これらのサブ・グループは、内部に現象レベルの個別シナリオを含むサブ・グループと考えた。例えば、ベントナイト中の化学的現象・特性と言うサブ・グループの中には７種類の個別シナリオが現在提案されている。

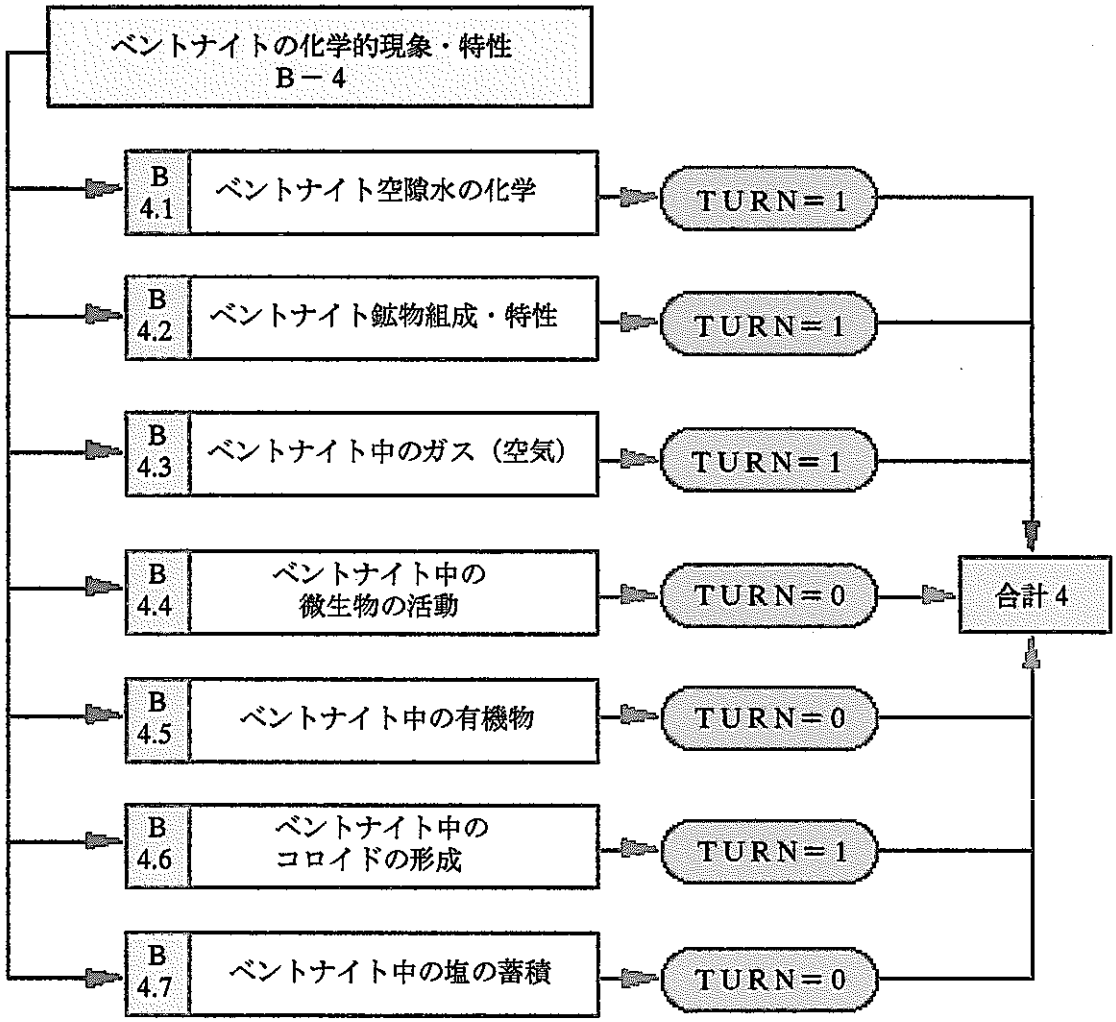

図１．２－３　サブ・グループ・ターンの合計概念

このように、ターンとは個別シナリオごとの評価指標とした。例えば、ベントナイト中の化学的現象・特性と言うサブ・グループについてはターンの値は７である。GUI設計において、ターンは個々のシナリオに組み合わされた下記の状態を示す指標（評価作業の進行を示す状態変数）であり、地層処分ヴァーチャル・エンジニアリング・システムを利用して評価が行われたか否かを表すこととした。

（完了ターン　）ターン＝０　：　シナリオは発生したが評価は行っていない状態
（未完了ターン）ターン＝１　：　シナリオについてなんらかの評価がなされた状態

現象解析レベルでは、個別シナリオそのものにかかわる評価手法について各種の議論が考えられると予想されたが、本研究は、情報処理技術の応用としてシステマチックな性能評価を目標とした情報システム環境開発技術が目標のため、検討対象から除外した。従って、前記のベントナイト中の化学的現象・特性にかかわる評価は、現在のシナリオからは７ターンで評価を完了すると考えた。それにより各サブ・グループのシナリオ評価の状態を示す指標を式１）の通りとし、結果をGUI表示し性能評価全体について視覚的に把握するための１指標とした。今後詳細なターン変数表現が必要と考えら

れる。

サブ・グループ・ターン完了値＝（サブ・グループの完了ターン数）／
（サブ・グループのターン数）＊１００．０　　式　1）

図１．２－３にサブ・グループ・ターン完了値の例を示す。この例では、完了値は５７％である。この５７％の完了値を視覚的に把握しやすいよう、カラーでGUI上に表示するものとした。

（２）シナリオ評価GUI

各ターンを実行するためのGUIをシナリオ評価GUIとした。シナリオ評価GUIは、個別シナリオに対応して、１ターンの評価を行う作業を支援する機能を提供するものとした。評価を行う作業の支援とは、個別シナリオに記載された評価に関する記述にかかわるモデル、シミュレーション結果、試験結果との検証結果等、シナリオ解析の論拠となる具体的な情報を提供する作業とした。

図１．２－４に示すとおり、シナリオ評価においては、ＦＥＰリストの評価に関する記述を出発点として、モデルの選択、シミュレーションの実施、結果の検証までを対話方式で行うものとした。ターンについては、動作の妥当性を評価することを目的としたため、簡素化して、図１．２－４に示す一連の評価作業の完了をもって１ターンとした。この点については、今後、モデル、検証、試験データ等の多様化を考慮して指標をたてる必要があると考えられる。図１．２－４から解かるように、その初期の段階では、ターンは従来から研究開発されてきた個別評価コードを利用してシナリオに示される個別評価事象を、数値計算で解析することにより全体を終了する還元論的手法を一元管理するアプローチとしている。

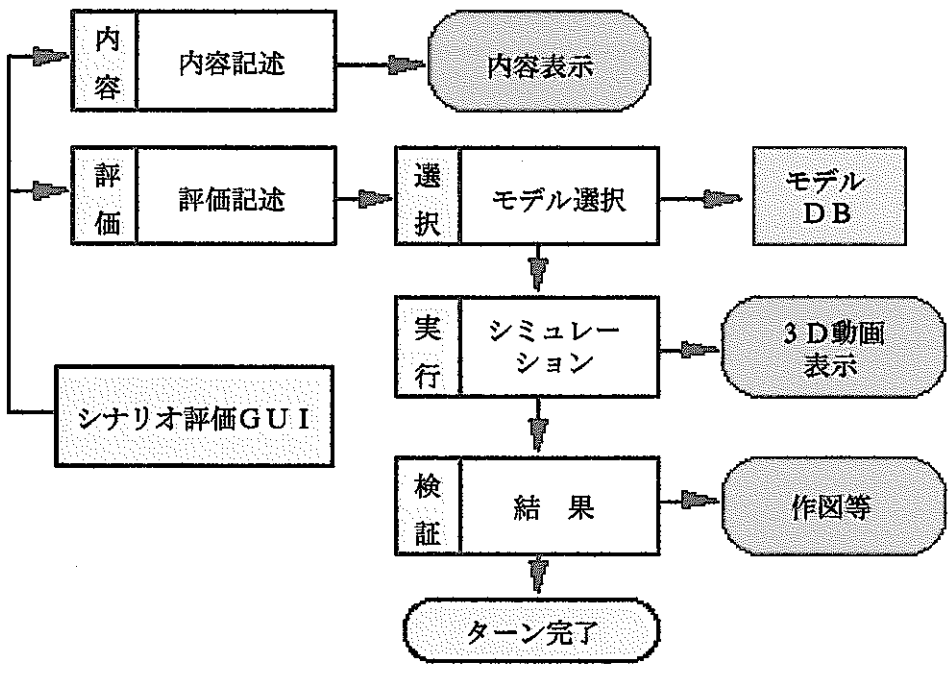

図１．２－４　シナリオ評価GUI動作機能概念

（3）シミュレーション

シミュレーションは、図１．２－４に示すシナリオ評価GUIから起動される機能とした。今回の試作では、シミュレーション・コードは、個別にデータベース管理されているものとして起動し、計算結果をシステムに共通な３次元動画像表示プログラムにより再現するものとした。この場合のシミュレーションの考え方は、個別シナリオ評価GUIに含まれる機能としている。

また、連成モデルについては、連成するシナリオを関係付け連成シミュレーションであることを確認できるよう配慮した。これについては今後詳細な研究が必要である。シミュレーションは、評価シナリオの対象となるオブジェクトが４つのステージについて共通であることから、３Dオブジェクトについて、時刻及び物理変量という２つの変量次元を加えて５次元オブジェクト（X、Y、Z、t、物理変量）として考え、共通の視覚化サブ・システムに表示するものとした。図１．２－５に起動時の操作画面を示す。

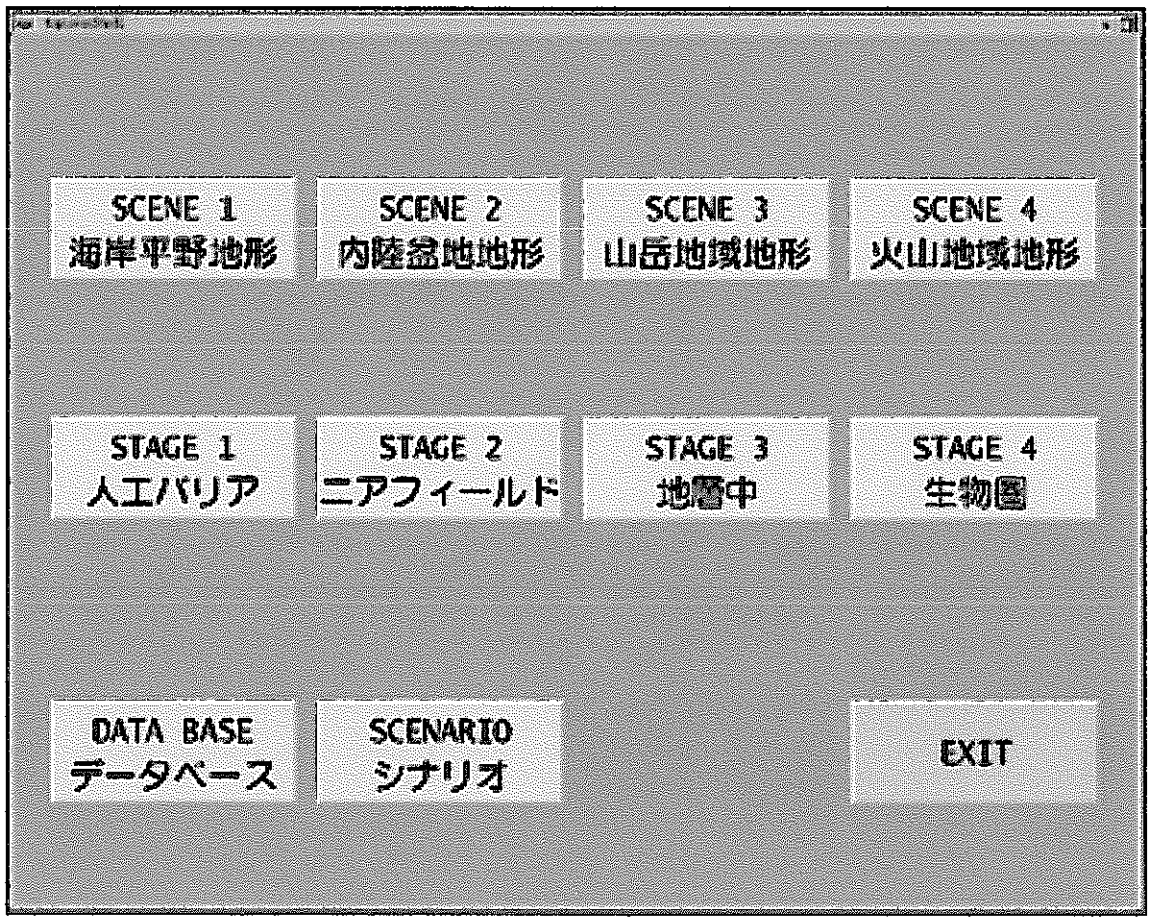

図１．２－５　初期設定画面の例

図１．２－５に示すように、システムは起動されると全ての画面を初期設定し、主操作用画面にステージ等のメニューを表示する。図中の海岸平野、内陸盆地等は、将来のシナリオとして考えられる天然地形に関する情報を格納するデータベースについて選択する操作を行なうために予備的に準備した。図１．２－６にその例を示す。

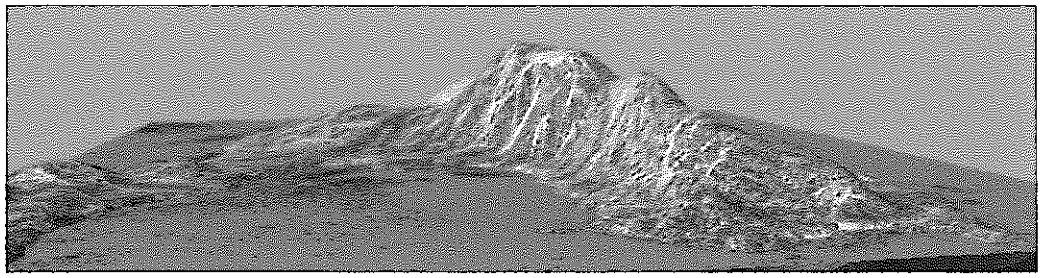

図１．２－６　天然地形オブジェクトの例

図１．２－７はステージを選択操作した画面の例である。図では、ステージ１の人工バリア核種放出シナリオを選択している。図に示すように、ステージを選択すると当該ステージ内の個別シナリオを選択することができる。例えば、ガラス固化体の滲出やオーバーパックの腐食等である。

このように、ステージ内ではシナリオはより単純な物理化学過程に還元されるものとした。

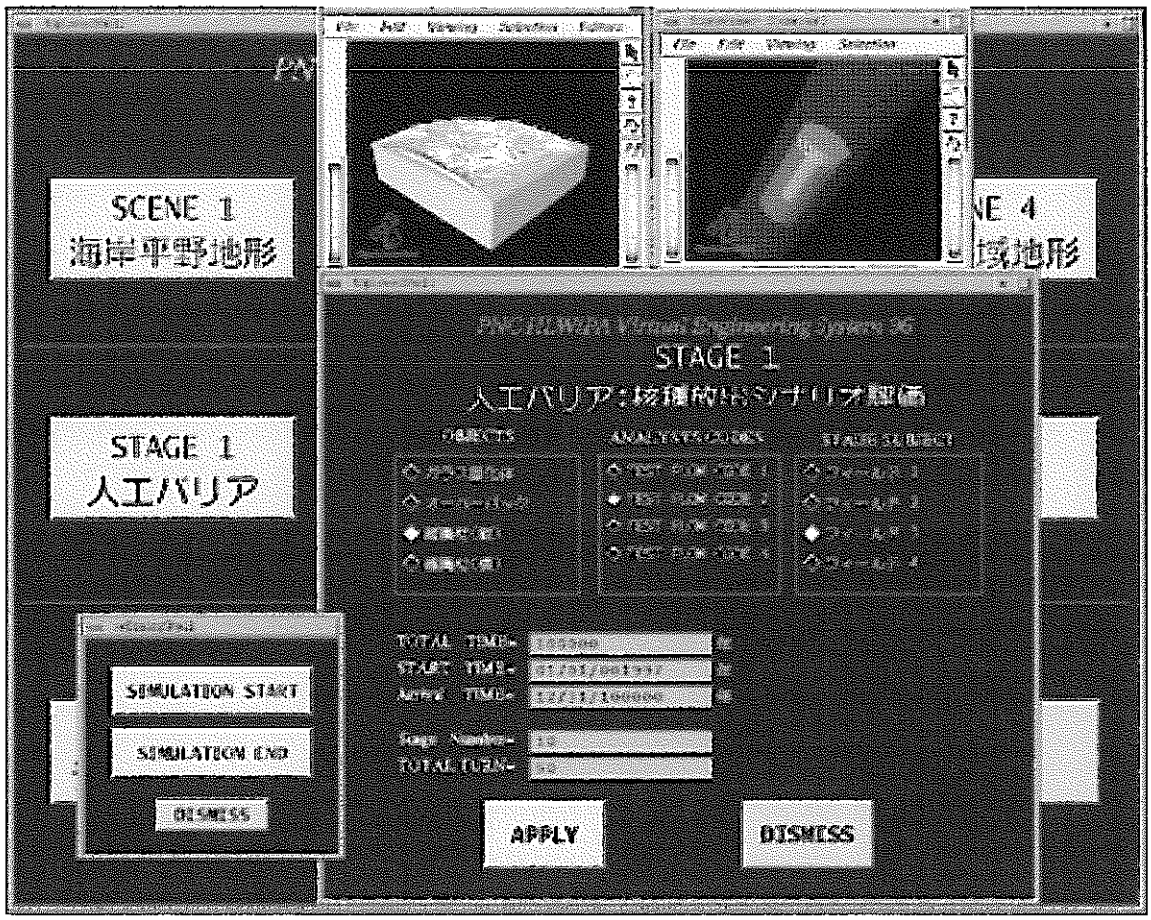

図１．２－７　シミュレーション条件選択例

図１．２－８は、以上のような操作により選定された個別事象の評価計算コードを実行した時点で起動される、３次元の動画化プログラムによる作図表現である。このプログラムは、個別コードから物理化学変量の値と、ｘ、ｙ、ｚ空間座標の組を時刻とともに受け取ることにより３次元で動画表示するプログラムである。これは、動作中でも、拡大・縮小、回転、塗りつぶし、ベクトル表現、等高線表現等を任意の断面について行なえるようにしている。

以上のように、試作した地層処分ヴァーチャル・エンジニアリング・システムの対話機能は、シミュレーションの対象となる問題領域をシナリオに沿って実行し、３次元動画化する機能を提供する。

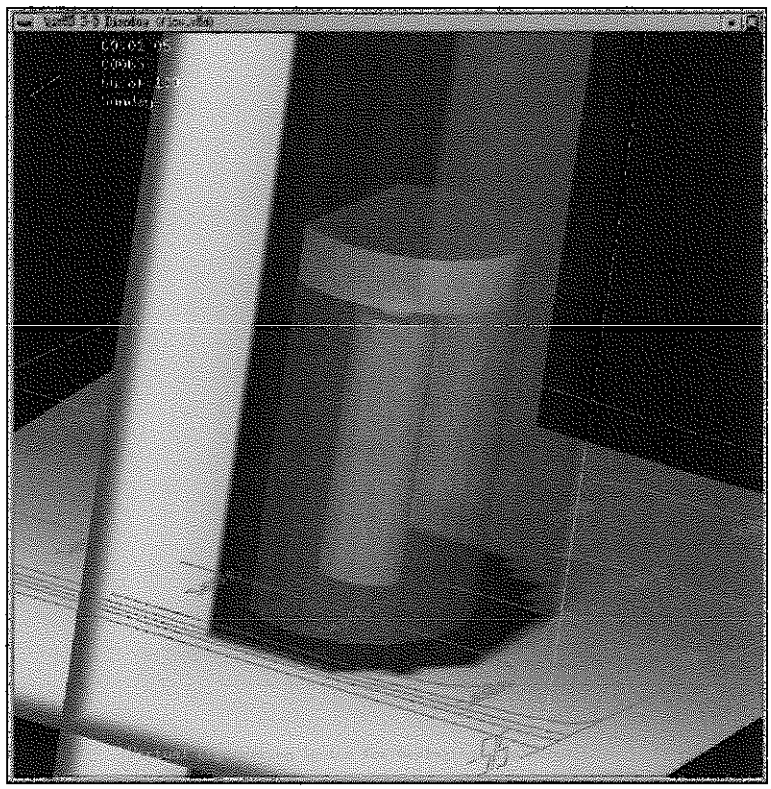

図１．２－８　３次元シミュレーション動画表現例

ソフトウェア管理は、地層処分ヴァーチャル・エンジニアリング・システムを開発運用する上で必要となる多くの試験・検査データ、技術文献、プログラム、解析結果等の情報について、その品質を保守管理することを主目的として開発するシステム運用保守ツールとした。試作においては、下記の観点からの情報の管理に大別して実施した。

数値データの管理　（例）熱力学データベース等の試験データ
文章データの管理　（例）モデル記述データベース
解析コードの管理　（例）プログラム・データベース

図１．２－９は、地層処分ヴァーチャル・エンジニアリング・システムを構成する情報（ソフトウェア）のインフラ・ストラクチュアを示したものである。実際には、このような単純２分岐木構造にはならないが、管理面からは管理の容易さを考え、可能な限り図１．２－９に示す様な体系になるよう情報のインフラ・ストラアクチャを構成するものとした。GUIの試作に於ては、図１．２－９に示す情報構造に向けて、データ・ベース等の構築を進める上で必要となると考えられる技術を利用する為のGUIを試作した。例えば、熱力学データベース分析試験データ等の解析や管理を行う機能、地下水のモデル記述と関連プログラム情報に関する記述等を管理する機能、実際のソースプログラムとその複雑度等の品質管理指標を管理する機能に関するものである。

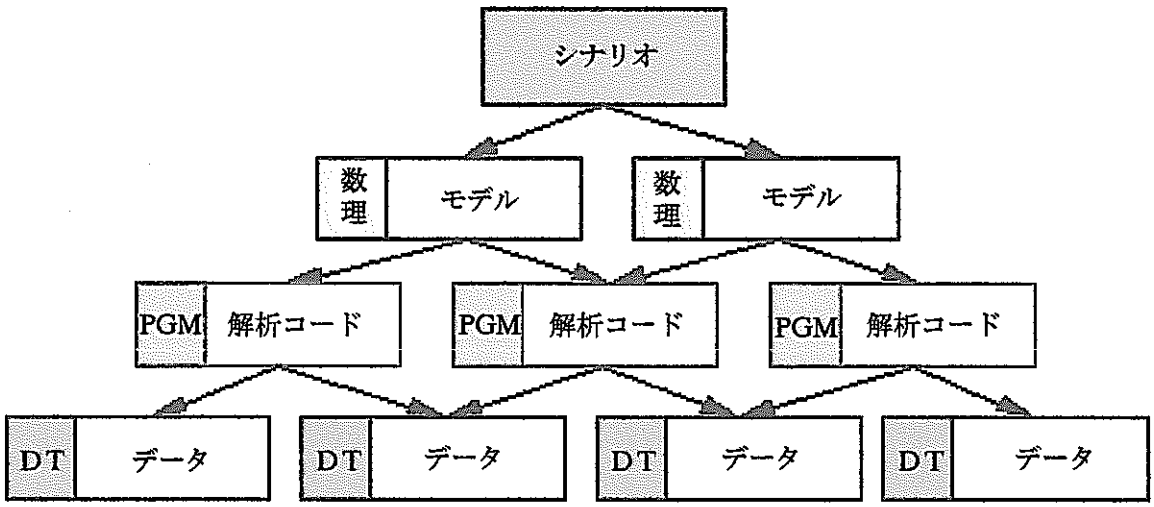

図１．２－９　情報のインフラ・ストラアクチャ

１．３　ユーザー・インターフェイスの高度化検討

ユーザー・インターフェイスと言う情報処理概念は、８０年代に確立してきたビットマップ・ディスプレイ装置と、それをネットワーク・レベルで透過的にサポートするMIT X11グラフィック・ディスプレイ制御システム・ソフトウェアの開発成果にその基盤を置いている。

エンド・ユーザーからみた場合、それは大型画面に並べられたいくつもの画面であり、恰も机上に並べた本やノートの様に参照、書き込み等の作業を行うことを可能としたコンピュータ画面制御技術である。このことは、ユーザーの画面情報に対する思考過程が、ユーザー・インターフェイス出現以前のメイン・フレーム・コンピュータに接続された端末装置に比較して、機能は飛躍的に高度化したが、反面複雑かつ試行錯誤的な作業を増加させる原因を提供したと考えられる。これは、作業の円滑性に関しては負の要因と考えられる。なぜならば、ユーザーは画面上に散在した複数のソフトウェアに対応する判断を行わなければならず、多くの場合、それらのソフトウェアの間に直接の関連が無い場合が考えられるためである。例えば、シナリオを評価する画面、解析結果をプロットする画面、さらにネットワーク通信の画面等がユーザーのコンピュータ画面に展開し、ユーザーはそれらについて適宜判断し、作業をすすめなければならないため、心理的、生理的ストレスに起因する環境であると考えられる。従って、エンド・ユーザーから見た場合、ユーザー・インターフェイスの使い勝手は、

仕事を進める上で重要な要因と考えられる。そのような背景から、ユーザー・インターフェイスは、解かり易く“気が利いた”ものとして設計されなければならないと判断し、より使いやすいユーザーインターフェイスを開発する為の技術について予備的に検討した。

作業はユーザーとの対話処理操作をより効率化すると供に、ユーザーによる意思決定を支援する為の参考となる情報を提供する機能等について予備的に検討した。地層処分ヴァーチャル・エンジニアリング・システムとユーザーとの対話は、評価シナリオの選択、シミュレーションの実行、実行結果の整理等、シミュレーションに直接関連する操作に始まり、シミュレーション結果の妥当性等を検証・確証するため、関連情報をデータベースより検索し、解析結果と比較評価する等の様々な操作が複雑に関連するものであると考えられた。

2　対話システム開発技術基盤の検討

対話機能から想定する地層処分ヴァーチャル・エンジニアリング・システムは、処分の安全性評価に係わる各種のシナリオについて、計画する処分システムとその環境を構成するオブジェクト、解析コード、検証・確証用試験データ等、複雑に組み合わせ複雑な現象を数値的に再現（シミュレート）するシステムとした。

システム構成面から地層処分ヴァーチャル・エンジニアリング・システムを考えた場合、それは下記に示す機能別に分散したハードウェアをネットワークにより統合し、シミュレーションにおいてはそれらが並列的に協調して動作する並列ネットワーク分散処理技術により、一元統合化されたシステムとして考えた。

シミュレーション用システム（Runtime Visual Simulation System）
操作用システム（Operator Console System）
データベース用システム（Data Base System）
パラメータ・ディスプレイ用システム（Status Display System）
オーディオ・ビデオ用システム（Audio Video System）

対話システム開発技術基盤の検討作業では、地層処分ヴァーチャル・エンジニアリング・システムのユーザーインターフェイス機能開発に必要な技術の評価を行なった。

検討の結果、計画するシステムが異機種のネットワーク並列処理機能（マシン・パラレル）により統合化されるため、ユーザーインターフェイス開発環境については、完全な一元化はハードウェアをドライブする基本システムの相違等から難しいと考えられた。

従って、システム・ハードウェアをUNIX系とＰＣ系に二分し、UNIX系はMIT/X11を基盤としたXネットワーク・プロトコールによりグラフィック・ウィンドウ制御を統一すると供に、Zバファを画像フレーム．メモリとして有するハードウェアについては、シリコン・グラフィクス社が開発したIRIS4D/GLプロトコールを採用し、拡張することが現在入手可能な技術として適当であるとした。また、X11のウィンドウ管理については、OSF/Motif xm intrinsicを採用することが適当であると判断した。なぜならば、地層処分ヴァーチャル・エンジニアリング・システムのUNIXワークステーションが異機種のマルチ・コンピュータ環境となるため、メーカーが共同して開発したOSF/Motifがシステムの統合化に適しているためである。

ＰＣについては、MacintoshとIBM PC系に２種類のアーキテクチュアが市場で入手可能である。地層処分ヴァーチャル・エンジニアリング・システムのＰＣとしては、仕様の公開されているIBM/PC系を採用し、ユーザーインターフェイス開発環境はUNIXシステムとサーバー・クライアントの構成が容易なアーキテクチュアを提供するWindows95を採用した。

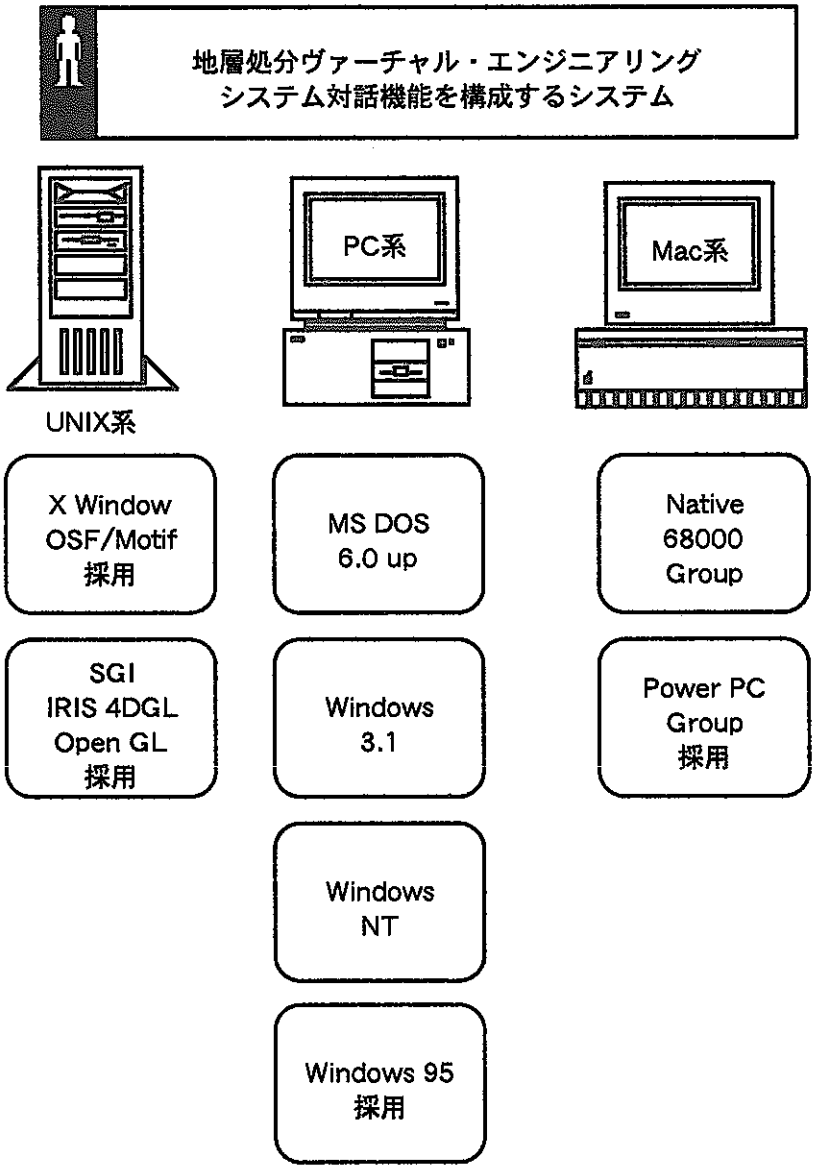

図２－１　入手可能な対話システム開発環境

図２－１に現在、市販されているUNIX、PC、Macintoshコンピュータが提供するGUI制御ソフトウェアの種類を示す。図から判るように、８種類のGUI制御技術によるGUIが普及している。地層処分ヴァーチャル・エンジニアリング・システムのユーザー・インターフェイス開発に当たっては、選択したハードウェアに搭載するこれらの基本ウィンドウ制御システム・ソフトウェアの仕様に従って対話プログラムを設計開発しなければならない。また、開発するシステムは、異なった機種のコンピュータがネットワークで通信しながら資源を共有し、対話システムを同期的に実行する機能が要求されるため、ユーザー・インターフェイス開発にあたっては、マルチ・プラットホームに透過的な制御プログラムとする必要がある。従って、ここでは、それらの要求を満足させるためのシステムの設計の準備として、図２－１に示す対話制御システムの基本技術を整理した。作業にあたっては、予想されるハードウェア・アーキテクチュアについて、グラフィック・ユーザー・インターフェイス・プロ

グラムを開発するための技術、環境基盤を整備し、次年度より開始する予定の統合的ソフトウェア開発のための準備を行なうこととした。従って、各プラットホームに関する対話システムの開発にかかわる基本技術をとりまとめた。

またマルチ・プラットホーム（ネットワークに分散接続された、異なったハードウェア、異なったオペレーティング・システム等）環境において、効率良く地層処分ヴァーチャル・エンジニアリング・システムの対話システムを開発するための技術等について検討した。

２．１　UNIX系対話システム開発技術評価

UNIX系対話システム開発を進めるため、技術評価作業は、MIT X11（X Window System Version 11）を基本とした、対話システムの設計開発技術と開発環境（開発用のツールソフトウェア、ライブラリ、コンパイラー等）について技術的な評価を行った。また、X11は仕様上２次元の対話制御システムであるため、３次元の対話制御システムとして標準になりつつある、シリコン・グラフィクス社の4DGL系についても評価対象とした。

２．１．１　Xプログラミング開発技術

Xウィンドウ・システムは、MITで設計されたネットワーク透過なウィンドウ・システムである。Xディスプレイ・サーバは、モノクロームまたはカラーのビットマップ・ディスプレイ・ハードウェアを備えたコンピュータ上で稼働する。サーバは、同一マシン上またはネットワーク上の他のマシンにある様々なクライアント・プログラムに、ユーザからの入力を配分し、クライアント・プログラムからの出力リクエストを受け付ける。Xlibは、C言語のサブルーチン・ライブラリである。アプリケーション・プログラム（クライアント）はXlibを使用して、ウィンドウ・システムとのストリーム接続によるインターフェイスをとる。クライアントは接続しているXサーバと同じマシン上で動作するのが普通であるが、必ずしもそうである必要はない。

Xlibとは、Xウィンドウ・システム・プロトコルへの抵水準C言語インターフェイスである。ここでは、システム開発に必要なライブラリ中の各関数の仕様と関連情報を整理する。高水準の概念、たとえばXツールキットが提供するものは、Xlibライブラリの上に構築する。

（１）Xウィンドウ・システムの概要

本項で使用する用語の中にはXに固有のものがある。また、他のウィンドウ・システムでもよく使われる用語の中にも、Xでは異なる意味を持つものがある。Xウィンドウ・システムは、重なり合うウィンドウや、サブウィンドウを含む、1つまたは複数のスクリーンをサポートするシステムプログラムとその実行環境である。

スクリーンは、１つの物理的モニタとハードウェアで、カラーでもグレー・スケールでもモノクロームでもよい。各ディスプレイ、またはワークステーションには複数のスクリーンがあってもよい。１つのXサーバは、任意の数のスクリーンに表示や描画のサービスを提供することができる。１人のユーザのための１台のキーボードと、１つのポインタ（通常はマウス）を備えた一群のスクリーンをディスプレイと呼ぶ。

Xサーバ内のすべてのウィンドウは厳密な階層構造を形成している。すべての階層の一番上にある

のがルート・ウィンドウで、ディスプレイの各スクリーンを覆っている。各ルート・ウィンドウは、子ウィンドウによって、その一部または全部が覆われる。ルート・ウィンドウ以外のすべてのウィンドウは親を持つ。通常、１つのアプリケーション・プログラムにつき、少なくとも１つのウィンドウがある。

子ウィンドウもまた、その子ウィンドウを持つことができる。このようにして、アプリケーション・プログラムは、各スクリーン上で任意の深さの木を作成することができる。Xは、ウィンドウに対するグラフィックス操作、テキスト操作、及びラスタ操作を提供する。

子ウィンドウが、その親ウィンドウより大きいこともある。つまり、子ウィンドウの一部または全部が親の境界外にはみ出すことがある。ただし、ウィンドウへの出力は、その親によってクリップされる。

あるウィンドウの複数の子ウィンドウが重なり合う位置にある場合は、それらの子ウィンドウのうちの１つが他のすべての子ウィンドウの上にあるものと見なされ（あるいは一番上に引き上げられ）、他のウィンドウを隠す。他のウィンドウによって覆われている領域への出力は、そのウィンドウにバッキング・ストアがない限り、ウィンドウ・システムによって抑止される。あるウィンドウが別のウィンドウによって隠されている場合、隠した方のウィンドウはその祖先のウィンドウだけ（隠された方の祖先でもある）を隠すことになる。

ウィンドウには、幅が0ピクセルから数ピクセルのボーダがある。ボーダは、任意のパターン（ピックスマップ）または色に設定することができる。多くの場合（常にではない）、ウィンドウには背景パターンがあり、そのウィンドウが可視状態になったときにウィンドウ・システムによって再描画される。子ウィンドウは親を隠す。また、親ウィンドウ内のグラフィックス操作は、普通は子ウィンドウによってクリップされる。各ウィンドウとピックスマップはそれぞれの座標系を持つ。座標系はX軸が水平で、Y軸が垂直であり、原点[0、0]は左上である。座標はピクセルを単位とした整数であり、ピクセルの中心と一致する。ウィンドウの原点はボーダより内側の左上隅である。

Xは、ウィンドウの内容の保持を保証しない。ウィンドウの一部または全部が隠された後で再びスクリーンに表示されたときに、その内容が失われている場合がある。その場合、サーバはイベントを送って、ウィンドウの一部、または全体を再描画する必要があることをクライアント・プログラムに知らせる。プログラムは、要求があればウィンドウの内容を再生成できるようになっていなければならない。Xはまた、ピックスマップと呼ばれる画面外の記憶領域を用いるグラフィックス・オブジェクトを提供している。１プレーン（深さ１）のピックスマップは、ビットマップと呼ばれることもある。ピックスマップは、ほとんどのグラフィックス関数においてウィンドウと同じように扱うことができ、様々なグラフィックス操作において、パターンやタイルを定義するために使用される。ウィンドウとピックスマップをまとめてドローアブルと呼ぶ。

Xの環境ライブラリであるXlibの関数のほとんどは、出力バッファにリクエストを追加するだけである。これらのリクエストは後でXサーバ上で非同期的に実行される。サーバに格納されている情報の値を返す関数は、明示的応答を受け取るかエラーが発生するまでは復帰しない（つまりブロックする）。

プログラマは、エラーが報告されたときに呼び出されるエラー・ハンドラを提供することができる。リクエストが非同期的に実行されることをクライアントが望まないならば、リクエストの直後にXSyn

cを呼び出せばよい。するとリクエストは、先にバッファに入れられている非同期イベントがすべて掃き出されて処理されるまでブロックされる。重要な副作用として、Xlibの出力バッファは、サーバからの値を返す関数、あるいは入力を待つ関数を呼ぶことにより常に掃き出されるというものがある。

多くのXlib関数は整数の資源識別子を返す。この識別子を使って、ユーザはXサーバに格納されているオブジェクトを参照することができる。資源の型としては、Window、Font、Pixmap、Colormap、Cursor及びGContextがあり、<X11/X. h>ファイルの中で定義されている。

これらの資源はリクエストによって作成され、リクエストによって、または接続がクローズされるときに破壊（または解放）される。これらの資源のほとんどは、基本的にアプリケーション間での共用が可能で、実際、ウィンドウはウィンドウ・マネージャによって明示的に操作されている。フォントとカーソルは複数のスクリーンの間で自動的に共用される。フォントは必要に応じてロード及びロード解除され、複数のクライアントによって共用される。フォントはサーバの中にキャッシュされることが多い。Xlibは、グラフイックス・コンテクストをアプリケーション間で共用するためのサポートは提供していない。

クライアント・プログラムはイベントを通知される。イベントは、リクエストの副作用である場合もあれば（たとえば、ウィンドウのスタック順序を変更するとイベントが生成される）、完全に非同期の場合もある（たとえば、キーボードからのイベント）。クライアント・プログラムは、イベントの通知を要求する。他のアプリケーションがユーザのアプリケーションにイベントを送ってくる可能性もあるので、プログラムにはすべての型のイベントを処理（または無視）する備えがなければならない。

入力イベント（キーの押下やポインタの移動など）は、サーバから非同期に到着し、明示的な呼び出し（XNeoxtEventやXWindowEventなど）によって要求されるまで待機させられる。また、いくつかのライブラリ関数（XRlise Windowなど）は、ExposeイベントやConfigureRequestイベントを生成する。これらのイベントも非同期に到着するが、クライアントは、サーバにイベントを生成させる可能性のある関数を呼び出した後でXSyncを呼び出すことによって、これらのイベントを明示的に待つこともできる。

（2）エラー

関数にはステータス（status：エラーを示す整数）を返すものがある。そのような関数は、異常終了すると0（ゼロ）を返す。関数がステータスとして0を返した場合には、返される引数は更新されない。Cは複数の値を返すようにはなっていないので、多くの関数はクライアントが渡した記憶領域に結果を書き込んで返すという方法をとらなければならない。省略時の場合、エラーは標準ライブラリ関数かユーザが提供した関数によって処理される。文字列へのポインタを返す関数は、文字列が存在しない場合にはNULLポインタを返す。

Xサーバはプロトコル・エラーを検出するとそれを報告する。１つのリクエストについて複数のエラーが発生した場合には、サーバがそのうちのどれを報告するかはわからない。

通常Xlibはリクエストをサーバに即座には送信しない（いったんバッファに入れる）ので、エラーは実際に発生してからかなりの時間が経ってから報告される可能性がある。しかし、Xlibはデバッグのために動作の同期化を強制するための機構を提供している。同期化機構を有効にすると、エラーは

発生した時点で報告されるようになる。Xlibはエラーを検出すると、プログラムのエラー・ハンドラを呼び出す。エラー・ハンドラがない場合は、エラー・メッセージが出力されてプログラムは終了する。以上が制御の概要である。

（3）標準ヘッダ・ファイル

下記のインクルード・ファイルはXlib標準の一部である。

<X11/Xlib.h>

これはXlib の主ヘッダ・ファイルである。Xlibのシンボルの大部分は、このファイルをインクルードすることにより宣言される。このファイルは、プリプロセッサ・シンボルXlibSpecificationReleaseも含んでいる。このシンボルの値としては、標準の本リリースでは5が定義されている（従来のリリースのXlibにはこのシンボルはない）。

<X11/X.h>

このファイルは、アプリケーションによって使用されるXプロトコルのためのすべての型と定数を宣言する。このファイルは <X11/Xlib.h> で自動的にインクルードされるので、アプリケーションのコードでこのファイルを直接参照する必要はないはずである。

<X11/Xcms.h>

このファイルは、色管理機能の大部分のシンボルを含む。プレフィックス Xcms の付くすべての関数、型、シンボル及び色変換コンテクスト・マクロがこのファイルで宣言されている。このファイルは <X11/Xlib.h> で自動的にインクルードされるので、アプリケーションのコードでこのファイルを直接参照する必要はないはずである。

<X11/Xutil.h>

このファイルは、クライアント間通信関数及びアプリケーション・ユーティリティ関数で使用する様々な関数、型、シンボルを宣言する。このファイルは <X11/Xlib.h> で自動的にインクルードされるので、アプリケーションのコードでこのファイルを直接参照する必要はないはずである。

<X11/Xresource.h>

このファイルはリソース管理機能のためのすべての関数、型、シンボルを宣言する。このファイルは <X11/Xlib.h> で自動的にインクルードされるので、アプリケーションのコードでこのファイルを直接参照する必要はないはずである。

<X11/Xatom.h>

このファイルは、すべての定義済みアトムを宣言する。シンボルにはプレフィックスXA_が付いている。

<X11/cursorfont.h>

このファイルは、標準カーソル・フォントのために「付録B」に列挙されているカーソル・シンボルを宣言する。どのシンボルにもプレフィックスXC_が付いている。

<X11/keysymdef.h>

このファイルは、すべての標準キーシム値とプレフィックスXK_の付いたシンボルを宣言する。キーシンボルはグループに分けられ、プリプロセッサ・シンボルが各グループをインクルードするか否かの制御を行なう。プリプロセッサ・シンボルは関連する値を取得するためにファイルをインクルードするのに先だって定義されていなければならない。プリプロセッサ・シンボルには以下のものがある。XK_MISCELLANY、XK_LATIN1、XK_LATIN2、XK_LATIN3、XK_LATIN4、XK_KATAKANA、XK_ARABIC、XK_CYRILLIC、XK_GREEK、XK_TECHNICAL、XK_SPECIAL、XK_PUBLISHING、XK_APL、XK_HEBREW。

<X11/keysym.h>

このファイルは、プリプロセッサ・シンボルXK_MISCELLANY、XK_LATIN1、XK_LATIN2、XK_LATIN3、XK_LATIN4、XK_GREEKを定義し、<X11/keysymdef.h>をインクルードする。

<X11/Xlibint.h>

このファイルは、エクステンションのために使用するすべての関数、型、及びシンボルを宣言する。このファイルは<X11/Xlib.h>で自動的にインクルードされる。

<X11/Xproto.h>

このファイルは、エクステンションを実装するために使用するXプロトコルのための基本的な型とシンボルを宣言する。これは<X11/Xlibint.h>で自動的にインクルードされるので、アプリケーション及びエクステンションのコードでは、このファイルを直接参照する必要はないはずである。

<X11/Xprotostr.h>

このファイルは、エクステンションを実装するために使用するXプロトコルのための基本的な型とシンボルを宣言する。これは、<X11/Xproto.h>で自動的にインクルードされるので、アプリケーション及びエクステンションのコードでは、このファイルを直接参照する必要はないはずである。

<X11/X10.h>

このファイルは、X10互換関数で使用されるすべての関数、型、シンボルを宣言する。

（4）一般的な値と型

下記のシンボルはXlibによって定義されており、本稿を通じて使用される。

＊Xlibは型Boolおよび論理値TrueとFalseを定義している。
＊Nonoは普遍的なヌル資源識別子またはヌル・アトムである。
＊型XIDは汎用の資源識別子として使用される。
＊型XPointerは“chaf*”として定義され、データへの汎用の不透明ポインタ（opaque pointer）として使用される。

（5）命名と引数に関する規則

Xlibの関数の命名と構文はいくつかの規則に従っている。関数がどのような情報を必要とするかを覚えていれば、これらの規則によって、関数の構文が予測しやすくなるであろう。

主な命名規則は以下の通りである。

＊Xのシンボルを他のシンボルと区別するために、ライブラリは外部シンボルには大文字と小文字の混在する名前を使用する。従来通りに、変数名は小文字だけ、ユーザ・マクロは大文字だけで表記する。
＊Xlib関数名は、すべて大文字のXで始まる。
＊関数名とシンボルは、すべて大文字で始まる。
＊ユーザが見ることのできるデータ構造体の名前は、すべて大文字のXで始まる。さらに一般化すると、ユーザが内容を参照する可能性のあるものはすべて大文字のXで始まる。
＊マクロ及び他のシンボルは、大文字のXでは始まらない。これらをすべてのユーザ・シンボルと区別するために、マクロ名を構成する各語の先頭は大文字になっている。
＊データ構造体のすべての要素名または変数名は小文字である。名前が複数の語からなる場合には、各語がアンダースコア（_）でつながれている。

主な構文規則は以下の通り。

＊display 引数がある場合には、常に引数リストの先頭に現れる。
＊資源オブジェクト引数は、常にdisplay 引数の直後または引数リストの先頭に現れる。
＊グラフィックス・コンテクストが他の型の資源（ほとんどはドローアブル）と組になっている場合、組になっている資源を引数リストの中で前に置く。ドローアブルは他のすべての資源よりも前になる。
＊引数リスト中では、常に source（ソース）引数が先にあり、destination（宛先）引数が後にある。
＊引数リスト中では、常に x（x 座標）引数が先にあり、y（y 座標）引数が後にある。
＊引数リスト中では、常にwidth（幅）引数が先にあり、height（高さ）引数が後にある。
＊引数 x、y、width、height を一緒に使用する場合、常に x 引数と y 引数が先にあり、width 引数と height 引数が後にある。
＊マスクを構造体とともに使用する場合、引数リスト中では常にマスクを先に置き、構造体へのポインタを後に置く。

（6）プログラミングに関する留意点

プログラミングに関する主な留意点は以下の通り。

＊Xにおいては、座標と大きさは16ビットで表わされる。この決定は、所定の性能水準を得るために必要な帯域幅を小さくするためになされた。Xlib では座標を int として宣言するのが普通である。16ビットより大きな値は、エラーも警告もなしに切り捨てられる。大きさ（幅と高さ）は、符号なしの数値である。
＊キーボードは、異なるメーカのワークステーション間で最も相違の大きな部分である。したがって、移植性の高いプログラムを書きたいのであれば、キーボードについては特に保守的にプログラミングすべきである。
＊一般にディスプレイ・システムの画面外メモリの量は限られている。したがって、ピックスマップとバッキング・ストアの使用は極力減らすべきである。
＊ユーザは、スクリーンを地所のように管理できなければならない。したがって、スクリーン全体を制御しようとするのではなく、ウィンドウ管理に反応するようにユーザ・アプリケーションをプログラムしなければならない。ただし、ユーザがその最上レベル・ウィンドウの内部で何をするかは、ユーザ・アプリケーション次第である。

（7）文字セットとコード表現

いくつかのXlib関数は特定の文字セットと文字コード表現を参照する。以下に示すものが最も一般的である。

X Portable Character Set

97文字の基本集合。Xlibによってサポートされるすべてのロケールにあるものと想定される。この文字集合には、以下の文字が含まれる。

a..z A..Z 0..9
!"#$%&' ()*+,-./:;<=>?@[＼]^＿ ‘{|}~
<間隔>,<タブ>,<改行>

これは、ISO8859-1 図形文字集合の左/下半分に <間隔>,<タブ>,<改行> を加えたものである。これはまた、7ビットのASCII図形文字集合に同じ３つの文字を加えたものでもある。これらの文字のホストにおける実際のコード表現は、システムに依存する。

Host Portable Character Encoding

ホストにおけるX Portable Character Set のコード表現。コード表現自体は、この標準では定義されていないが、コード表現はホストのXlibによってサポートされるすべてのロケールで同一でなければ

ならない。ある文字列がHost Portable Character Encodingであるという場合には、その文字列はそのホストでのコード表現によるX Portable Character Setの文字しか含んではならない。

Latin-1 ISO8859-1規格に従ってコード化された文字集合。 STRING encoding Latin-1にタブと改行を加えたもの。

POSIX Portable Filename Character Set

POSIX準拠のホスト上でファイルの命名に使用することのできる65文字からなる文字集合であり、すべてのロケールで正しく処理される。この文字集合には以下の文字が含まれる。

a..zA..Z0..9．__-

＊各関数の解説は、まず概説によって他の関数との違いを明らかにし、関数宣言と各引数の説明を続ける。ANSI C関数プロトタイプ構文は使用されていないが、Xlibのヘッダ・ファイルは通常ANSI C環境の関数プロトタイプを使用して関数を宣言する。必要があれば、引数リストの後に関数の説明を続ける。説明の最後には、その関数が生成する可能性のあるXlibエラー・コードが存在すればそれを列挙する。

＊ユーザが渡す引数と関数が返す引数との混同を避けるために、ユーザが渡す引数の説明は「を指定する」で終える。関数が返す引数の説明は「が返される」で終える。ユーザが渡して関数が返す引数の説明は「～を指定する。～が返される。」と表記する。

＊値を返すために使用される構造体へのポインタの名前は、__returnで終える。これらの関数に渡される他のポインタは、すべて読み取り専用である。いくつかの引数は入力と出力の両方に使われる構造体へのポインタである。そのような引数の名前は__in__outで終える。

2．1．2　IRIS OpenGLプログラミング開発環境

OpenGLは、グラフィックス・ハードウェアに対するソフトウェア・インターフェイスである。このインターフェイスは、およそ120種類のコマンドから成り立っており、それらを使用してオブジェクトやインタラクティブな3Dアプリケーションを作成するために必要な操作を指定する。

OpenGLは、作成したグラフィックスを表示するコンピュータが、グラフィック・プログラムを実行するコンピュータとは異なる場合でも正常に動作するよう設計されている。これには、ネットワーク化されて、相互にデジタル・データを転送可能な複数のコンピュータを含む作業環境が考えられる。

この場合、プログラムを実行させてOpenGLの描画コマンドを発行するコンピュータを“クライアント”、それらのコマンドを受け取って実際に描画を実行するコンピュータを“サーバ”と呼ぶ。クライアントからサーバへOpenGLコマンドを転送するためのフォーマット（“プロトコル”と呼ぶ)

は常に同一のものであり、仮にクライアントとサーバが異機種のコンピュータであっても、プログラムはネットワークを通して正常に動作する。

また、OpenGLプログラムがネットワーク上で使用されない、つまり、スタンドアロン（単独）で作業する場合、そのコンピュータはクライアントであると同時にサーバである。

（1）OpenGL仕様概要

OpenGLは、多種の異なるハードウェア・プラットフォーム上で実行可能な効率の高いハードウェア“非依存型”のインターフェイスとして設計されている。この特色を活かすため、タスクのウィンドウ化やユーザのコマンド入力などの機能は用意していない。このため、実際に使用するハードウェアを制御するウィンドウ・システムの操作性に問題があっても、そのシステムの環境を利用して作業しなくてはならない。さらに、OpenGLには3Dオブジェクトのモデルを作成するための上位レベルのコマンドも用意されない。この種のコマンドがあれば、自動車、人体の一部、飛行機、分子モデルなどの比較的複雑な形状を作成できるが、OpenGLを使用する場合には、必要なモデルを点、線分、多角形（ポリゴン）などの、“プリミティブ（幾何学的基本形状）”の組み合わせで作成することが必要となる。

図２．１－１はOpenGLを利用した一例である。コンピュータによって、より複雑な方法で連続して描かれた（つまり“レンダリング”された）画面である。以下にこの画像の作成手順を解説する。

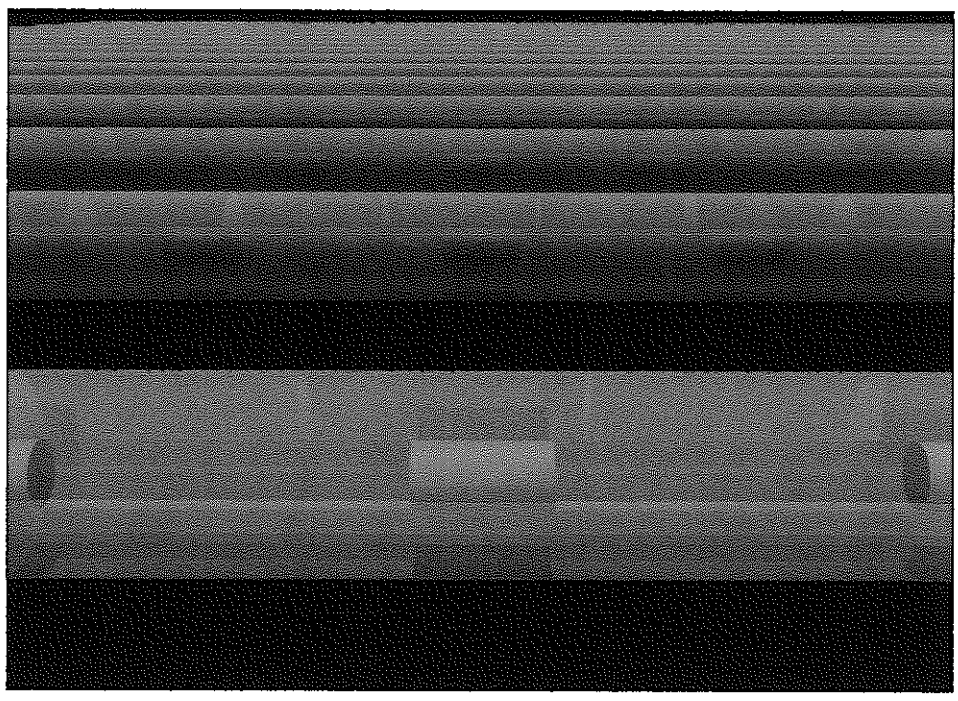

図２．１－１　OpenGL作図例

図２．１－２は、ワイヤフレーム・モデルを表示した画面で、すべてのオブジェクトは針金で作成したように見える。針金の各線はそれぞれプリミティブ（典型例として多角形）の輪郭に対応する。

例えば、オーバーパックの表面は三角の多角形により構築される。

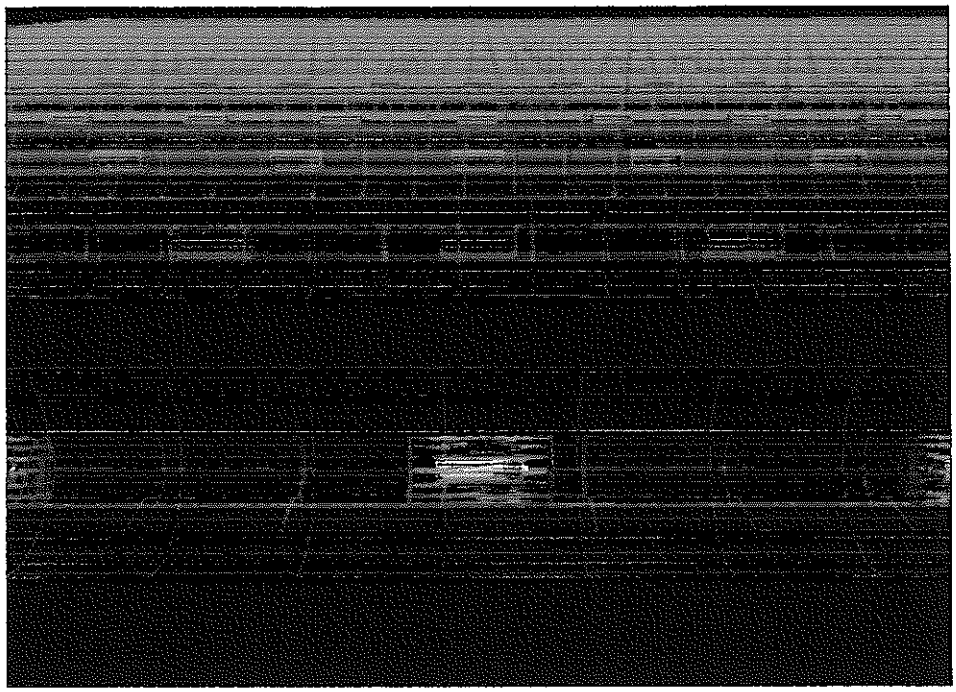

図２．１－２　ワイヤ・フレーム

オブジェクトがワイヤフレームではなく、ソリッド（中身の詰まったもの）な場合でも、隠れるべき部分のオブジェクトの一部が表示される。例えば、ガラス固化体はオーバーパックに隠されているにもかかわらず、そのオブジェクトが表示される。ベントナイトのオブジェクトは何百ものブロックで構成されているため、ほとんどソリッド化されて表示され、ベントナイトの裏側を形成するものまで含めて、全ブロック全ての輪郭線を作成するワイヤフレームが表示される。このベントナイト・オブジェクトの構築方法を参照するのが、低レベルのオブジェクトの組み合わせでかなり複雑なオブジェクトを作成することができる。

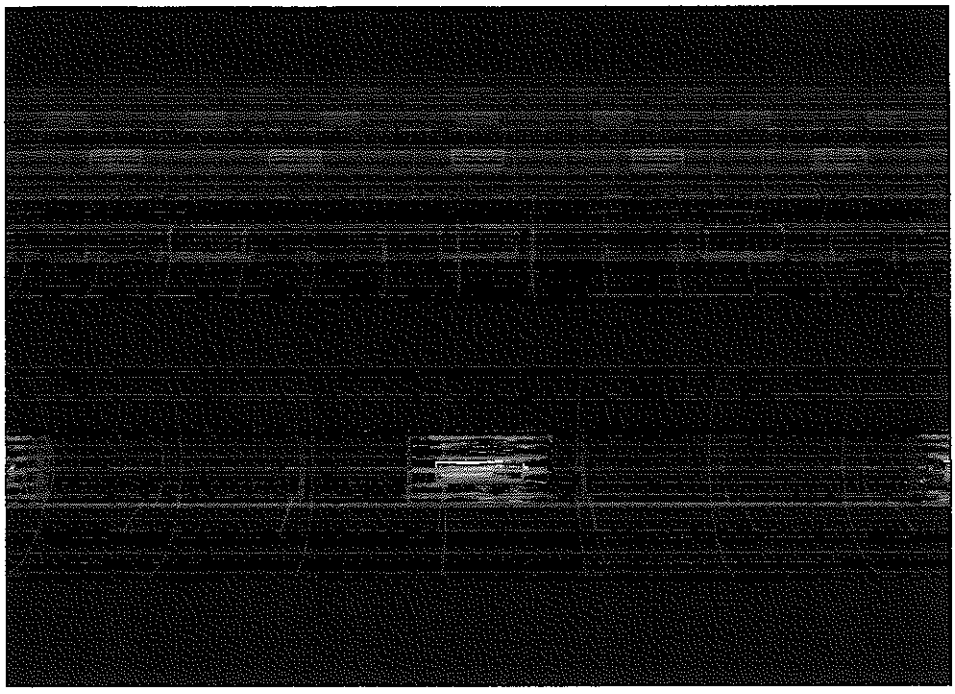

図２．１－３　デプス・キュー

図２．１－３に、同じワイヤフレーム画面の“デプス・キュー”バージョンを示す。実世界で見られるように、眼点から離れたラインはぼけており視覚的な奥行きが与えられている。

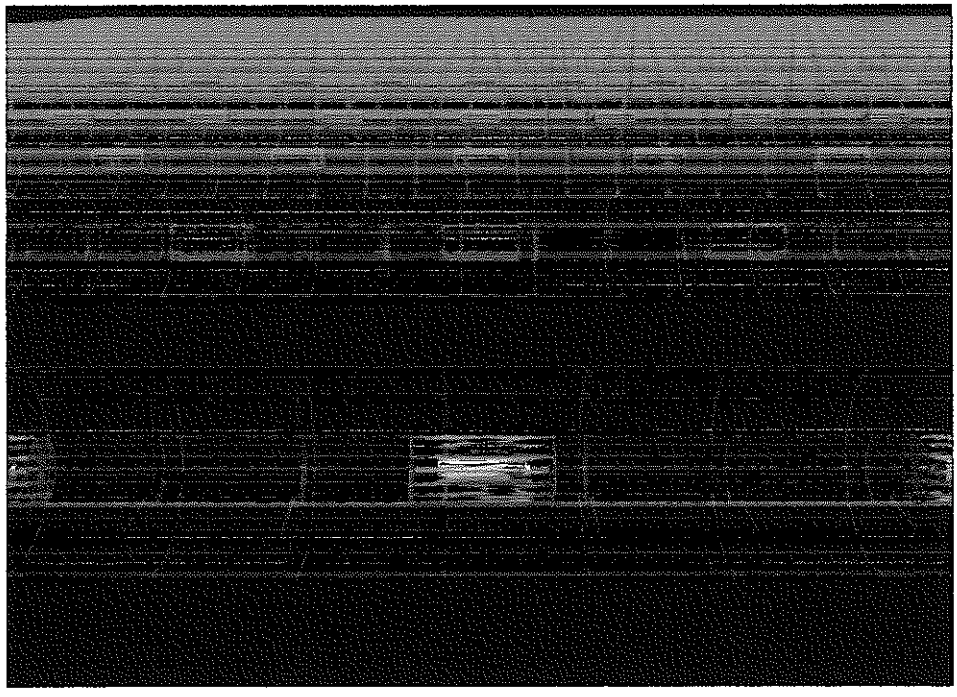

図２．１－４　アンチエイリアシング

図２．１－４にワイヤフレーム画面に“アンチエイリアシング”を施したものを示す。アンチエイリアシングとは、隣接するピクセルの一部分が描く画像からはみ出した場合に生成されるギザギザ（ジャギー）の状態を除去する技法である。このようなギザギザは、通常はほとんど水平、または垂直な線に生じる。

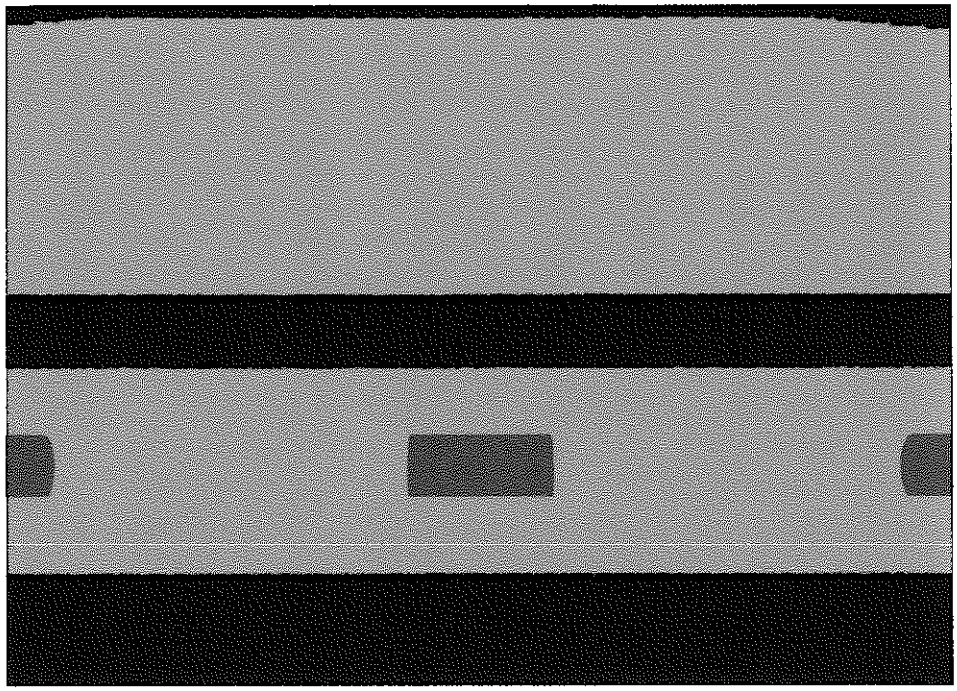

図２．１－５　フラット・シェーディング

図２．１－５は、画面の“フラット・シェーディング”バージョンである。画面内のオブジェクトは単色のソリッドとして表示されている。照光処理条件には対応していないため平坦に表示され、滑らかな曲線としては描かれていない。

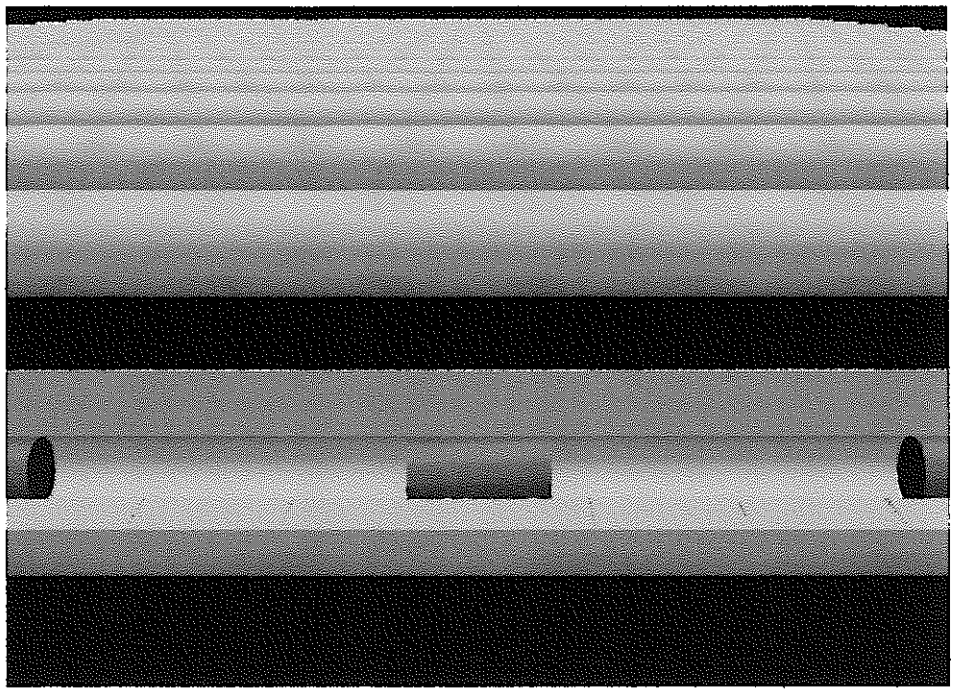

図２．１－６　照光されたスムース・シェーディング

図２．１－６は、“照光されたスムース・シェーディング”バージョンである。オブジェクトが光源に対応して濃淡処理（シェーディング）されると、画面がリアルで立体的に見えることが理解できる。オブジェクトの表面も滑らかに丸みを帯びている。

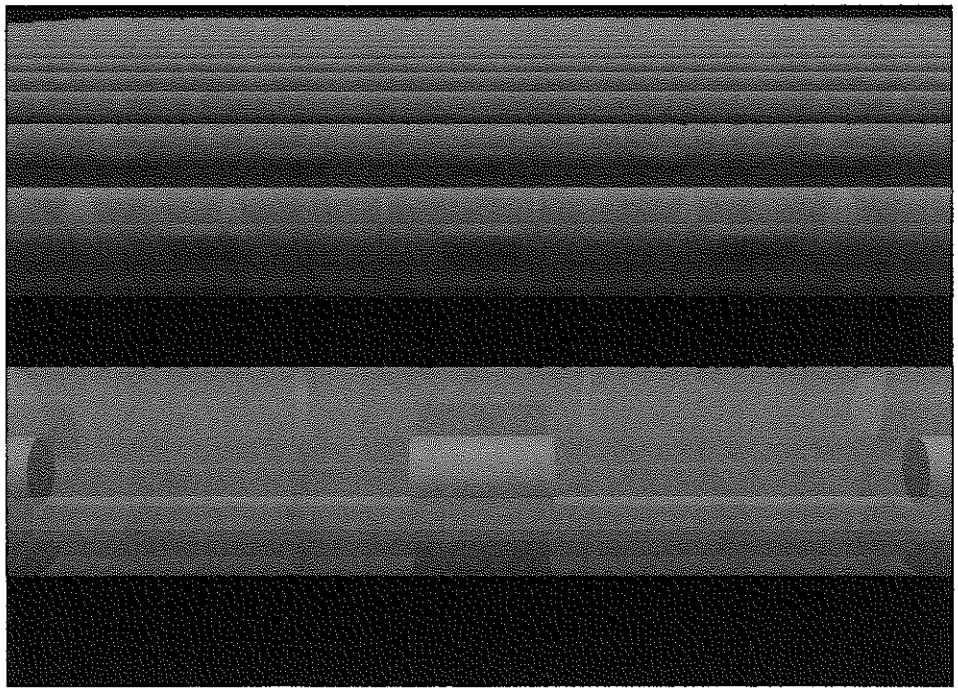

図２．１－７　陰影とテクスチャ

図２．１－７は、前の画面に“陰影とテクスチャ”を加えたものである。OpenGLは陰影処理を基本的な機能として提供しない。“テクスチャ・マッピング”を使用すると３次元オブジェクトに２次元のテクスチャを貼り付けられる。この画面の場合、テクスチャ・マッピングの最も顕著な例は坑道の表面に見られる。オーバーパックの表面はすべてテクスチャ・マッピングされたものである。

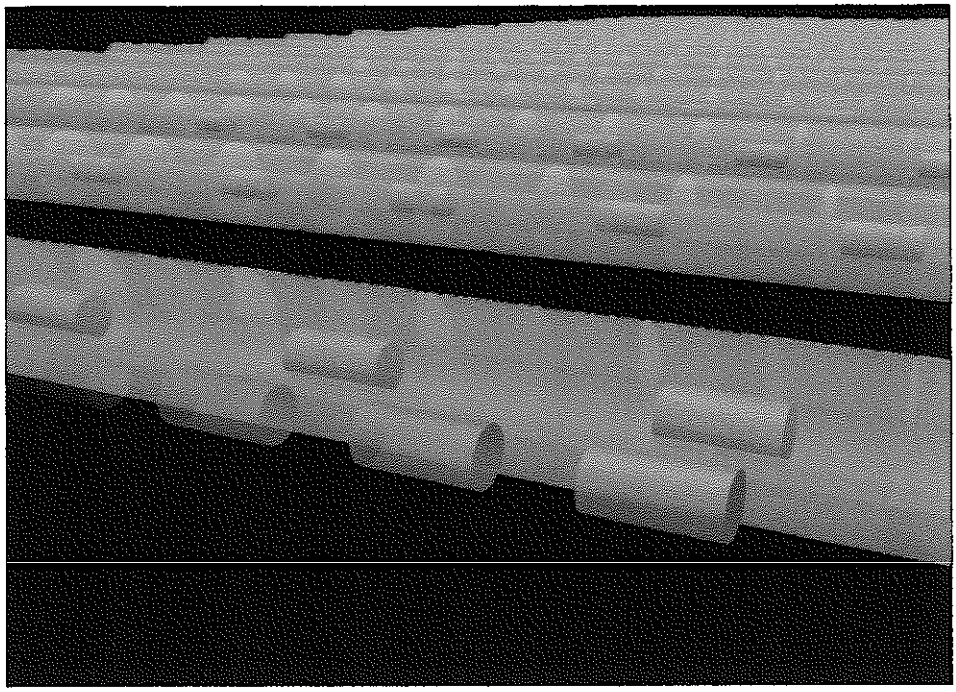

図２．１－８　動きのボケたオブジェクト

図２．１－８に画面内の“動きのボケた”オブジェクトを示す。オブジェクトは、進行に伴って移動の軌跡をぼかして見える。

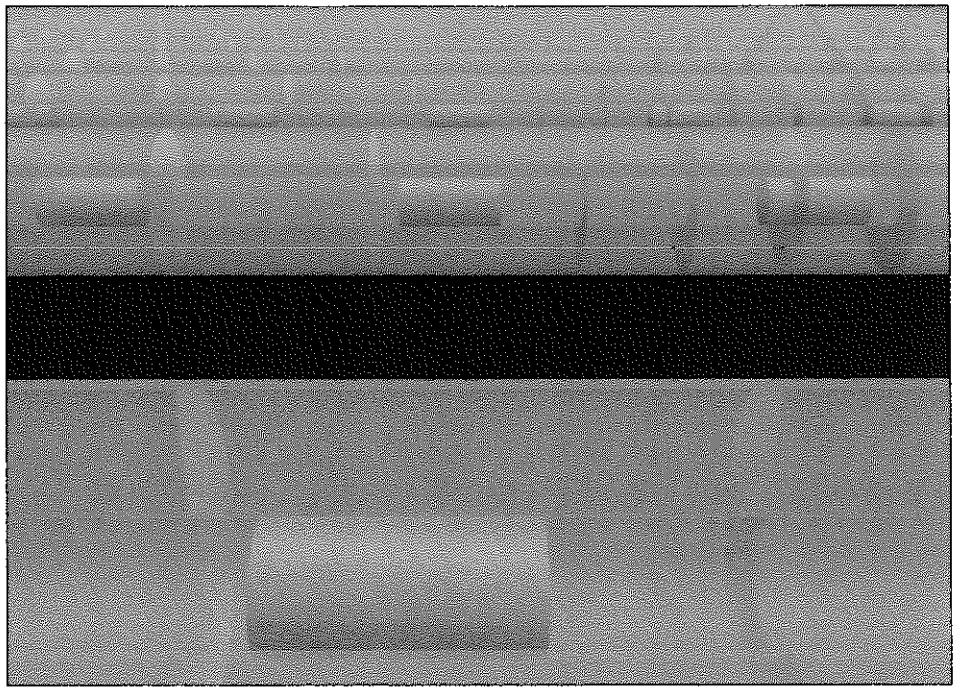

図２．１－９　動きのボケた異なる視点からのスナップショット

図２．１－９は、画像がまさに3Dオブジェクトの“スナップショット”モデルであることを示している。次の２つのカラー画像では、OpenGLで表現可能な複雑な視覚効果を紹介する。

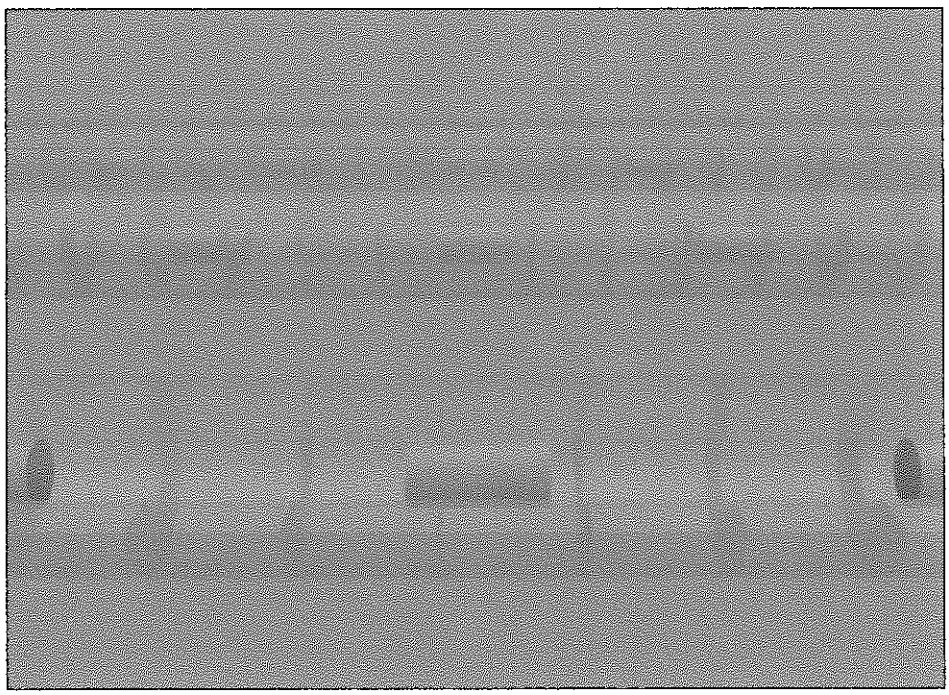

図２．１－１０　大気効果

図２．１－１０は、大気効果を使用して、空気中に存在する微粒子を表現したものである。

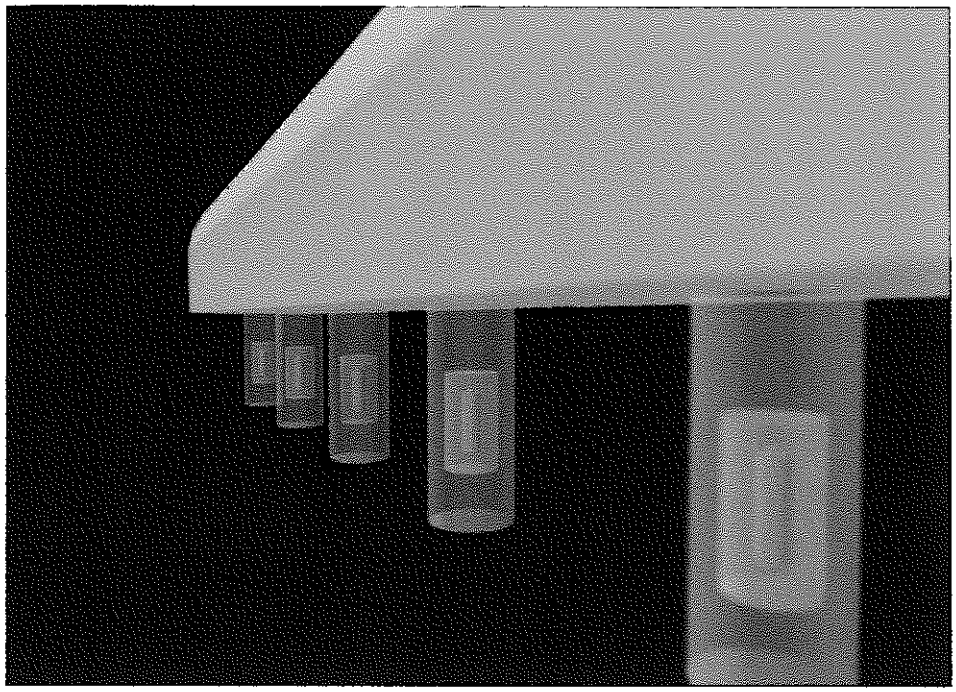

図２．１－１１　被写界深度効果

図２．１－１１は、“被写界深度”効果を表し、カメラのレンズが、被写画面内に存在するモデルすべてに対して焦点を合わせられない状況を再現したものである。

カメラが画面内のある1点に焦点を合わせると、その点より手前や奥にあるオブジェクトはボケて表示される。以上の図よりOpenGLで画面に画像をレンダリングするために必要な、主なグラフィック操作は下記のステップの通りである。

ステップ１：プリミティブでオブジェクトの形状を決定し、オブジェクトを数学的に記述する。
ステップ２：３D空間にオブジェクトを配置し、構成した画面をどの位置から眺めるかを選択する。
ステップ３：すべてのオブジェクトのカラーを計算する。カラーはアプリケーションで具体的に割り当て、指定した照光処理、またはオブジェクトに貼り付けるテクスチャで確定される。
ステップ４：オブジェクトの“数学的な記述”と、対応するカラー情報を画面上でピクセルに変換する。この処理を“ラスタライズ”と呼ぶ。

OpenGLは、この作業の間に、他のオブジェクトに遮られたオブジェクトの一部を削除する（遮られた部分は作業の無駄を省くため描かれない）などの操作も実行する。さらに、画面をラスタライズした後、画面にそれを表示する前であればピクセルデータも操作可能である。

（2）OpenGLプログラミング

多数の機能を持つOpenGLグラフィック・システムのプログラムは複雑だと思われがちであるが、実際のプログラムはいたって単純な基本構造となっている。その作業内容は、OpenGLがどのようにレンダリングするかを制御するステータスの初期化とレンダリングするオブジェクトの指定である。

OpenGLのプログラムを検討する前にいくつかの用語について検討する。すでに“レンダリング”は登場しているが、これはコンピュータがモデルから画像を作成する処理を意味する。これらの“モデル”、つまりオブジェクトは、点、線分、多角形の“頂点”によって形成されるプリミティブによって作成される。最終的にレンダリングされた画像は、画面に描かれた“ピクセル”で構成される。

この“ピクセル”とは、“画素”の略であるが、これはディスプレイ・ハードウェアが画面上で配置可能な視覚要素の最小単位である。ピクセルに関する情報（カラーなど）は、システム・メモリの形で“ビットプレーン”に含まれる。ビットプレーンとは、画面上の各ピクセルに関する1ビット分の情報を持つメモリ領域のことである。

例えばビットは“ある特定のピクセルがどのくらい赤いか”などの意味を示す。ビットプレーン自体は、グラフィック・ディスプレイが画面上の全ピクセルの濃度を制御するために必要なすべての情報をもつ“フレーム・バッファ”に含まれる。

図２．１－１３は、図２．１－１２で示した黒色の背景上にある白色の方形をレンダリングするためのものである。

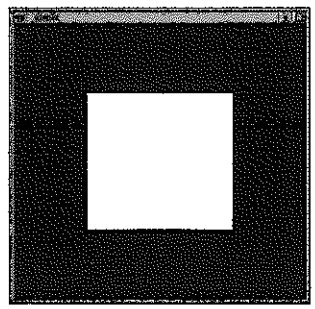

図２．１－１２　黒色の背景上にある白色の方形

```
#Include <whateverYouNeed. h>
main () {
   OpenAWindowPlease();
   glClearColor(o.o,o.o,o.o,o.o);
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(1.0,1.0,1.0,);
   glOrtho(-1.0,1.0,-1.0,1.0,-1.0,1.0);
   glBegin(GL_POLYGON);
      glVertex2f(-0.5,-0.5);
      glVertex2f(-0.5,0.5);
      glVertex2f(0.5,0.5);
      glVertex2f(0.5,-0.5);
   glEnd();
   glFlush();
   KeepTheWindowOnTheScreenForAWhile();
}
```

図２．１－１３　単純なOpenGLのプログラム

図２．１－１３において、“main()”ルーチンの最初の行で画面上にウィンドウを開く。

“OpenAWindowPlease()”は、ウィンドウ・システム指定ルーチン用の置き換え記号、次の2行はウィンドウを黒色にするOpenGLコマンドである。“glClearColor()”でウィンドウのカラーを設定し、“glClear()”でそれを実際に実行する。一度、消去カラーを設定すると、“glClear()”をコールする度にそのカラーでウィンドウを消去する。再度“glClearColor()”をコールすると消去カラーを変更できる。

同様に“glColor3f()”は、描くオブジェクトに使用するカラーを設定するが、この例では白色を使用している。これ以降に描かれるすべてのオブジェクトは、再度設定するまでこのカラーを使用する。次のコマンド“glOrtho()”は、OpenGLが最終画像を描く際に採用する座標系を指定し、画像をどのように写像するかを決定する。

“glBegin()”と“glEnd()”の間に挟まれたコールは、描くオブジェクト（この例では4つの頂点をもつ多角形）を決定する。多角形の“角”は、“glVertex2f()”で設定する。（x、y）の座標で示した引数からわかるように、この多角形は方形である。最後に“glFush()”は、描画コマンドをOpenGLコマンド用格納バッファに保存せず、直ちに実行することを強制する。

早き換え記号の“KeepTheWindowOnTheScreenForAWhile()”ルーチンは、画像を直ちに消去せず、強制的に画面上に残す。

（３）OpenGLコマンドの構文

前項の単純なプログラムを見てもわかるとおり、OpenGLコマンドは“gl”で始まり、コマンド名の各単語の先頭を大文字（例えば“glClearColor”）にする。同様に定数は、“GL_”で始まり、すべて大文字で表記して単語間には下線を加える（例えば、“GL_COLOR_BUFFER_BIT”）。また、いくつかのコマンド名に付随する一見無関係な文字（“glColor3f()”の“3f”など）がある。

確かに現在のカラーを設定する定義としては、コマンド名の“Color”の部分だけで充分なのであるが、異なる型式の引数を使用可能にするため、それらのコマンドが複数定義されている。特に、接尾子の“3”は引数を3つ使用することを示すが、“Color”コマンドの別のバージョンでは4つの引数を使用する場合もある。“f”は引数が浮動小数点値であることを表している。

引数に8種類もの異なるデータ型式を受け入れるOpenGLコマンドもある。OpenGLのANSI C動作環境用のデータ型式を指定する接尾子の文字を対応するOpenGL形式の定義と併せて表2．1－1に示す。

使用するOpenGLの特定の動作環境は必ずしもこの表に従う必要はない。例えば、C++やAdaの動作環境にはその必要がない。

表2．1－1　コマンドの接尾子と引数のデータ型

接尾子	データ型	対応する標準的なC言語の型	OpenGLの型定義
b	8ビット整数	signed char	GLbyte
s	16ビット整数	short	GLshort
i	32ビット整数	long	GLint、GLsizei
f	32ビット浮動小数	float	GLfloat、GLclampf
d	64ビット浮動小数	double	GLdouble、GLclampd
ub	8ビット符号なし整数	unsigned chaf	GLubyte、GLboolean
us	16ビット符号なし整数	unsigned short	GLushort
ui	32ビット符号なし整数	unsigned long	GLuint、GLenum、GLbitfield

次の2つのコマンドは、

```
glVertex2i(1,3);
glVertex2f(1.0,3.0);
```

前者が頂点の座標を32ビット整数で指定するのに対し、後者は単精度の浮動小数点値で指定するという点を除いて同等である。OpenGLには末尾に“v”を付加したコマンドもあるが、これはコマンドが一連の個別引数ではなく、数値のベクトル（または配列）のポインタを持つことを示す。

多くのコマンドには、ベクトル、非ベクトル双方のバージョンが存在する。ただし、個別引数のみを受け入れるコマンドもあれば、最低何個かの引数をベクトルとして指定が必要な場合もある。次の例は、現在のカラーを設定するコマンドのベクトル・バージョンと非ベクトル・バージョンをどのように使用するかを示したものである。

```
glColor3f(1.0,0.0,0.0,);
float color_array[] = {l.0,0.0,0.0};
glColor3fv(color_array);
```

本書では、実際のサンプル・プログラムを除いてOpenGLコマンドをその基本名のみで表し、アス

ダリスク（＊）を付加したものは、そのコマンド名がさらに続くことを示す。例えば、“glColor*()”は、現在のカラーを設定するために使用するコマンドの全バリエーションを表す。特定のコマンドのひとつのバージョンを指定したい場合は、そのバージョンを指定するために必要な接尾子を追加する。例えば、“glColor*v()”は頂点を決定するのに使用するコマンドすべてのベクトル・バージョンを表す。最後に、OpenGLでは定数“GLviod”を定義しているが、C言語でプログラムする場合、これを“void”の代用として使用できる。

（4）ステート・マシンとしてのOpenGL

OpenGLはステート・マシンである。様々な変数（またはモード）を使用し、その変数は変更されるまで有効となる。これまで見てきたように“現在のカラー”も状態変数である。現在のカラーを白色、赤色、または他のカラーに設定可能であるが、別のカラーに変更するまで各オブジェクトは現在のカラーで描かれる。現在のカラーは、OpenGLのもつ多数の状態変数のひとつに過ぎない。他に、現在の視界や射影変換、線と多角形の破線パターン、多角形の描画モード、ピクセル・パックの法則、光の位置や特性、描くオブジェクトの材質特性などを制御するものもある。多くの状態変数は、コマンド“glEnable()”や“glDisable()”で有効、無効を決定するモードを設定する。各状態変数、またはモードは初期値を持ち、どの時点においてもシステムの各変数の現在値を確認できる。

これには通常、次の４つのコマンド、“glGetBooleanv()”、“glGetDoublev()”、“glGetFloatv()”、“glGetIntegerv()”のいずれかを使用するが、受け取るデータ型によって使用するコマンドは異なる。

状態変数には、より特化された確認コマンド（例えば、“glGetLight*()”、“glGetError()”、“glGetPolygonStipple()”など）を持つものもある。

さらに、コマンド“glPushAttrib()”と“glPopAttrib()”を使用して、一連の状態変数の数値を属性スタックに保存や取り出しが可能である。

（5）OpenGL関連のライブラリ

OpenGLでは、単純、かつ強力なレンダリング・コマンドを用意しており、上位レベルの描画はすべてこれらコマンドの実行が必要である。そのため、プログラミングの作業を単純化するために、OpenGL上に独自のライブラリ作成の必要性も予想される。また、OpenGLプログラムをウィンドウ・システム上で簡単に動作できるようなルーチンも必要になる。

以下に挙げる特別機能を持ったライブラリやルーチンがすでに作成されている。最初の２つは、いずれのOpenGL動作環境にも提供されており、３番目は本書用として書かれたもので“ftp”で利用する。４番目は、OpenGLに基づく単独の製品である。

“OpenGL Utility Libraly（GLU）”は、視界の方向や射影、多角形のタイリング、表面のレンダリング用の行列の設定などのタスクを実行するために使用する低レベルのOpenGLコマンドを使用したルーチンを提供する。このライブラリは、OpenGLの動作環境の一部として供給している。

“OpenGL Extension to the X Window System（GLX）”は、OpenGLのコンテクストを作成し、X Windowを使用する機器で描画可能なウィンドウにそれを結び付けるためのものである。

“Open Inventor”は、OpenGLに基づいたオブジェクト指向のツールキットで、インタラクティブな３Dグラフィック・アプリケーションを作成するためのメソッドやオブジェクトを提供する。

C＋＋で記述した、この“Open Inventor”は、Silicon Graphics社より入手可能である。ユーザ・インタラクション用の半完成オブジェクトや内蔵イベントモデル、３D画面の作成、編集用の高度なアプリケーション要素を機能としてもち、さらにオブジェクトのプリントや他のグラフィック・フォーマットとデータ交換も可能である。

（6）プログラミング・ガイド用の補助ライブラリ

OpenGLは、レンダリング・コマンドを備えているが、ウィンドウ・システムやオペレーティング・システムからは完全に独立した設計となっている。つまり、ウィンドウの開閉や、キーボードやマウスでイベントを取り込むためのコマンドを提供しない。ウィンドウを開かずに完全なグラフィック・プログラムの記述は不可能で、目的のプログラムを作成するためには多少のユーザ入力、オペレーティング・システムやウィンドウ・システムのサービスが必要となる。

多くの場合、プログラムが完全でなければ目的の画像の作成は不可能であるため、ウィンドウのオープンや入力の検知などの作業を単純化するために補助ライブラリを使用する。

OpenGLの描画コマンドは、単純なプリミティブ（点、線分、多角形）の作成のみに限定されるが、補助ライブラリには、球、トーラス、ティーポットなど、より複雑な３Dオブジェクトを作成するルーチンを用意している。

OpenGLの動作環境があり、システム内にこの補助ライブラリが存在する場合には、本書のサンプルは変更を加えずにリンク、実行させる。補助ライブラリは意図的に単純化しているため、その上に高度なアプリケーションを構築するのは困難が予想される。

ウィンドウの管理

３つのルーチンが初期化とウィンドウを開くために必要なタスクを実行する：

＊“auxInitWindow()”が画面上にウィンドウを開く.
　プログラムの実行を中断するエスケープ・キーを有効とし、ウィンドウの背景色を黒色に設定する。
＊“auxInitpositio()”は、画面上にウィンドウを配置する位置を“auxInitWindow()”に指示する。
＊“auxInitDisplayMode()”は、RGBA、またはカラー指標ウィンドウのどちらを作成するかを
“auxInitWindow()”に指示する。また、シングルかダブルのバッファ・ウィンドウが指定可能.
（カラー指標モードで作業する際に、カラーマップに特定のカラーを取り込む場合は、
“auxSetOneColor()”を使用する）このルーチンを使用して、関連するデプス、ステンシル、
アキュムレーションの各バッファをウィンドウに持たせることも示す。

入力イベントの操作

これらのルーチンを使用して、特定のイベントが発生した場合に実行するコールバック・コマンドを登録する。

＊“auxReshapeFunc()”は、ウィンドウのサイズ変更、移動、アクティブ化などに対する応答動作を指示する。
＊“auxKeyFunc()”と“auxMouseFunc()”は、キーやマウス・ボタンの押し、離しで実行するルーチンに、キーやマウス・ボタンをリンクするために使用する。

3Dオブジェクトを描く

補助ライブラリには、次の3Dオブジェクトを描くためのルーチンがいくつか準備されている：

球体　八面体
立方体　十二面体
トーラス　二十面体
円柱　ティーポット
円錐

これらのオブジェクトは、表面の法線を定義したソリッド・シェーディングされたオブジェクト、またはワイヤフレームとして描くことができる。例えば、球とトーラスのルーチンは、図2．1－14のようになる。

```
void auxWireSphere (GLdouble radius) ;
void auxSolidSphere (GLdouble radius) ;
void auxWireTourus (GLdouble innerRadius,GLdouble outerRadius) ;
void auxSolidTourus (GLdouble innerRadius,GLdouble outerRadius) ;
```

図2．1－14　球とトーラスのルーチン

これらのモデルすべては原点を中心に描かれる。単位スケール係数を使用して描く場合は、モデルは“-1”から“1”の全座標の枠内に収る。オブジェクトを拡大縮小するためにはこれらのルーチンに引数を使用する。

（7）バックグラウンド処理の管理

もし他に未処理のイベントが存在しない（例えばイベント・ループが停止する等）場合、“auxIdle Func()”で実行する機能を指定できる。このルーチンは、その関数のポインタを引数として持つ．関数の実行を無効にする場合はゼロに設定する。

（8）プログラムの実行

“main()”ルーチンで画面にオブジェクトを描き直すルーチンの名前を引数として“auxMainLoop()”をコールする。図2．1－15では、補助ライブラリの使用方法を示す。

```
#include <GL/gl.h>
#include "aux.h"
int main(int argc,char** argv)
{
   auxInitDisplayMode(AUX_SINGLE | AUX_RGBA);
   auxInitPosition(0.0,500.500);
   auxInitWindow(argv[0]);
   glClearColor(0.0,0.0,0.0,0.0);
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(1.0,1.0,1.0);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho(-1.0,1.0,-1.0,1.0,-1.0,1.0);
   glBegin(GL_POLYGON);
      glVertex2f(-0.5,-0.5);
      glVertex2f(-0.5,0.5);
      glVertex2f(0.5,0.5);
      glVertex2f(0.5,-0.5);
   glEnd();
   glFlush();
   sleep(10);
}
```

図２．１－１５　補助ライブラリを使用した単純なOpenGLプログラム：simple.c

（９）動画処理

グラフィック・コンピュータを使用して得られる視覚的効果のひとつが動画の作成である。自らが設計した機構の動作を検討するエンジニア、シミュレータを使用して飛行訓練を受けるパイロット、単なるコンピュータゲームのマニアのいずれにとっても、アニメーションがコンピュータ・グラフィックの重要な要素である。映画の場合、画像を連続（1秒間に24枚）させて動きを表現し、それをスクリーンに毎秒24枚の速度で映写する。レンズの裏側でフィルムが回り、シャッターが開くとその位置のフレームが映写される。このシャッターは、次のフレームに送る間は一時的に閉じ、フレームを次々に映写する動作を繰り返す。

このようにして、1秒間に24枚もの独立した画像を見ているにもかかわらず、脳はそれらを滑らかな動作のアニメーションとして認識する。現在の最も優れた映写機では、さらにその倍の毎秒48フレームという速度で各々の画像を映写してちらつきを抑えている。

コンピュータ・グラフィックの画面では、画面の描き直しを1秒あたりおよそ60～76回実行し、なかには毎秒120回の速度を持つものもある。もちろん、毎秒30回よりも60回の方がより滑らかになり、120回の方が60回よりも見た目は優れている。ただし、120回を超える速度では、人間の目の収穫逓減点（投資に見合う収穫が得られる限界）を超えるため無意味となる。

動画の映写を実行するための基本条件は、それを表示した時点で各フレームの内容が完全なものであることである。以下のプログラムを使用して、何万ものフレームからなる映画をコンピュータ・アニメーション化する。

```
open_window();
  for(i=0;i<1000000;i++){
  clear_the_window();
  draw_frame(i);
  wait_untill_a_24th_of_a_second_is_over();
}
```

図２．１－１６　アニメーション・プログラム例１

システムが画面を消去し、実際にフレームを描くために要する時間を加えると、このプログラムでは消去と描画に必要な1／24秒に近づくほど状態は悪化する。描画に約1／24秒を要すると仮定した場合、最初に描くオブジェクトは1／24秒の間を通して表示するため、画面上にソリッドな画像として表示される。しかし、1／24秒の終わり頃に描くオブジェクトについては、次のフレームの描画に移るためプログラムによって直ちに消去されるため、幽霊のようにぼんやりした残像として表示されるのが精一杯である。これは、最後に描かれるはずのアイテムに代わって消去された背景を、ほぼ1／24秒の間見続けているために生じる。問題点は、このプログラムが描画の完成したフレームを表示するのではなく、描画を実行する過程をそのまま表示することにある。

これを解決する簡単な方法は“ダブル・バッファ処理”、つまり２組の完全なカラー・バッファを供給するハードウェアやソフトウェアを利用する。

これは、一方を描画する間にもう一方を表示する。フレームの描画が終了すると双方のバッファを交換し、表示していた側は描画、描画していた側は表示される。これは、フレームを２コマのみ持つフィルムの輪に例えられる。一方のフレームをスクリーンに映写する間にアニメータが表示されないもう一方のフレームを消去して描き直す．このアニメータの作業が速ければ、映写機は単にそれらを次々に映写することになり、見る側は、準備段階とすべてのフレームがすでに描がれた段階との相違に気がつかない。

このようにダブル・バッファ処理を使用すると、描画が完成された時点でフレームが表示されるため、一部分だけを描画したフレームが表示されることはない。アニメーション化したグラフィックスを滑らかに表示するために先のプログラムを変更すると以下のようになる：

```
open_window_in_doble_buffer_mode();
for(i=0;i<1000000;i++){
    clear_the_window();
    draw_frame(i);
    swap_the_buffers();
}
```

図２．１－１７　アニメーション・プログラム例２

表示可能なバッファと描画可能なバッファを単に交換するだけではなく、“swap_the_buffers()”ルーチンは、前のバッファが完全に表示されるように現在の画面の消去が終了するまで待機する。

また、このルーチンにより、新しいバッファを最初から完全な状態で表示できる。利用するシステムが、画面を毎秒60回消去すると仮定した場合、最高速度は毎秒60フレームとなり、すべてのフレームを1／60秒以内で消去、描画さえ可能ならば、アニメーションはその速度でも滑らかに表示される。

しかし、実際のシステムでは、描くフレームの内容が複雑すぎて1／60秒に収まらない状況が頻繁に発生することも考えられるため、各フレームを２回以上連続して表示することになる。

例えば、あるフレームを描くのに1／45秒を要すると、結果的に毎秒30フレームとなってしまうため、フレーム毎にグラフィックスの動作しない1／30-1／45＝1／90秒間が発生する。1／90秒では大した損失と思えないが、1／30秒毎にそれだけの時間が無駄になる訳であるから、実際には全体の1／3が無駄になる。さらに、ビデオの消去間隔は一定であり、予期しない実行結果となることもありえる。例えば、1／60秒毎にモニタ消去し、一定速度でフレームを消去する場合、毎秒60フレーム、30フレーム、20フレーム、15フレーム、12フレーム…（60／1、60／2、60／3、60／4、60／5…）となる。

これは、例えばアプリケーションを書いて徐々に要素を追加する場合（処分システムの景色を追加する場合など）、最初は、追加する要素は影響せずに毎秒60フレームを維持するが、これに要素を追加すると、システムが1／60秒の間にすべてを描くことができずに、最初に可能となるバッファ交換のタイミングを逸して効率は半分となる。フレーム毎の描回時間が1／30秒を越える場合にも同様の状況が発生する。効率は毎秒30フレームから毎秒20フレームまで低下し、その33パーセントしか実行されない。

もうひとつの問題は、画面の複雑さによって“魔法の時間”（この例では1／60秒、2／60科、3／60秒など）のいずれかに近接する場合、バリエーションが多様なため、あるフレームは、わずかにその時間を超え、あるものはわずかに足りないなどフレーム速度にばらつきが生じ、滑らかに表示されないことが考えられる。この場合、全フレームに充分な速度を持たせるように画面を単純化することが不可能ならば、意図的に速度を遅らせてフレームの速度を一定にすることを推奨する。

フレームの内容が非常に複雑な場合には、さらに高度な解決法が必要となる。

興味深いことに、実際のアニメーション・プログラムの構造は、これまで説明した内容と大差がない。ほとんどの場合で各フレームに対してバッファ全体が下絵から描き直されるが、この理由は、描き直しが必要な部分を検出する方法よりも簡単なことによる。

これは、特に飛行機の方向をわずかに変更するだけで窓の外に見えるものの位置がすべて変化する３Ｄのフライト・シミュレーションなどのアプリケーションを使用する場合に該当する。

ほとんどのアニメーションの場合、視点の移動、車が道路を少し走る、オブジェクトがわずかに回転するなどの異なる変換によって画面内のオブジェクトが描き直される。

複雑な再計算が必要なフレームで大きな変更があると、一般にフレーム速度は遅くなる。ただし、“swap_the_buffers()”ルーチン後の空き時間をその計算に使用できる。

OpenGLには“swap_the_buffers()”の機能はないが、これは、その機能がすべてのハードウェアで使用可能ではなく、さらに、どのような場合でもウィンドウ・システムに依存するウェイトが非常に大きいためである。

ただし、GLXにはそのコマンド（void glXSwapBuffers(Display *dpy, Window window);）が用意され、X Windowを使用する機器では使用可能である。

```
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glx.h>
#include "aux.h"

static GLfloat spin = 0.0;
void display (void)
{
  glClear ( CL_COLOR_BUFFER_BIT ) ;
  glPushMatrix ( ) ;
  glRotatef ( spin,0.0, 0.0, 1.0);
  glRectf ( -25 . O , -2S. O, 25. O, 25. O) ;
  glPopMatrix ( ) ;
  glFlush ( ) ;
  gIXSwapBuffers (auxXDisplay ( ) , auxXWindaw( ) ) ;
}

void spinDisplay (void)
{
  spln = spin + 2.0;
  if (spin > 360.0)
    spin = spin - 360.0;
  display ( ) ;
}

void startIdleFunc (AUX_EVENTREC *event)
{
  auxIdleFunc (spinDisplay) ;
}
```

図２．１－１８　ダブルバファプログラム例

図２．１－１８では、正方形を一定速度で回転させるという例で、“glXSwapBuffers()”を使用する。

２．２　PC系対話システム開発技術評価

PC系対話システム開発の技術評価は、マイクロソフト社の提供するDOS、Windowsの２種のオペーレティング・システムを基本とした対話システムの設計開発技術と開発環境を対象として行った。

Microsoft Windows95は、初心者にとってもパワーユーザーにとっても、より簡単により速く仕事ができるように設計し直された新しいユーザーインターフェイスである。それは、インターフェイスのために実現された新しい機能をアプリケーションに取り入れることができる機能である。従って、開発者はインターフェイスに使用されているコントロールをリバースエンジニアリングする必要はない。

それらは、ソフトウェア開発キット自身にコントロールが含まれている、オペレーティングシステム自身が長いファイル名をサポートするようになった、インターフェイスのコードを何とか解明しようと苦労する必要もなくなった拡張性が組み込まれているなどである。本項では、Windows95のユーザーインターフェイスを活用するために必要な、以下の話題を中心にプログラミング技術を検討する。

- 新しいコモンコントロール
- 新しいコモンダイアログボックス
- 長いファイル名のサポート
- ショートカット
- ファイルビューア
- ユーザーインターフェイス拡張

以下、これらの機能を検討するとともに、Windows95のレジストリの使用、ユーザーインターフェイスDLL（SHELL32. DLL）の使用、シェルのネームスペースのブラウズなど、関連する重要な項目についても言及する。

２．２．１　新しいコモンコントロール

Windows95の新しいコモンコントロールには以下のものがある。

- ステータスバーとツールバー
- トラックバー、プログレスバー、アニメーションコントロール、アップダウンコントロール
- イメージリスト、リストビューコントロール、列見出し、ツリービューコントロール
- プロパティシート、タブ、ウィザード
- リッチエディットコントロール

これら多くの新しい要素を概観するにはエクスプローラがよい。エクスプローラは、ファイルマネージャとプログラムマネージャに代わるものである。図２．２－１には、新しいコモンコントロールのいくつかが示されている。Microsoftは、これらのコントロールをエクスプローラのために設計し、実装したが、あらゆるWindowsアプリケーションで使用できるようにエクスポートされている。

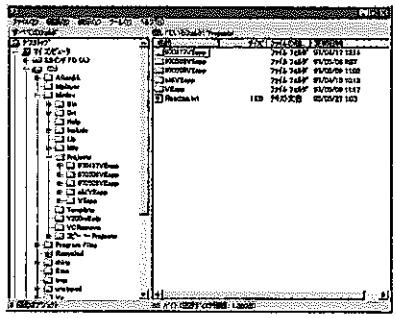

図２．２－１　エクスプローラ

（１）ステータスバーとツールバー

ステータスバーとツールバーは、組織化コモンコントロールである。すなわち、他のコントロールを組織化したりグループにまとめるために使用される。ステータスバーとツールバーは数年前からすでにアプリケーションに登場しているが、Windows95以前はオペレーティングシステムによるサポートはなかった。

ステータスバーは、通常アプリケーションのメインウィンドウの最下部に位置する。現在のドキュメントの名前、ドキュメントのサイズ、日付、時刻などのステータス情報がテキストまたは図形で表示される。ステータスバーは、アプリケーション内のオブジェクトに関するユーザー情報を提供する便利な手段である。

ツールバーには、ファイルのコピー、ドキュメントの印刷、書式の適用などの一般的な作業をすばやく実行するためのボタンやコンボボックスなど、他のコントロールがある。

たとえば、多くのアプリケーションでは、メニューから保存コマンドを選んだり、キーの組み合わせを覚えなくても、ツールバーのボタンをクリックするだけで、すぐに［保存］コマンドを実行することができる。ツールバーはアプリケーションのメインウィンドウの最上部に位置することが多いが、ユーザーがツールバーをカスタマイズしたり移動できる場合もある。ツールバーとステータスバーはともに先ほどの図に示されている。

（２）トラックバー、プログレスバー、アニメーションコントロール、アップダウンコントロール

強度の設定、値の変更、及びシステムで何が行われているかをユーザーがわかるようにいくつかの新しいコモンコントロールが設計された。たとえば、トラックバーを使えば、明るさ、ボリューム、色調などの強度を示す目盛りマークの範囲に沿ってスライダを動かすことができる。また、図２．２－２に示す垂直トラックバーを使うと、左右のスピーカのボリュームを調整することができる。

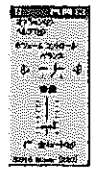

図２．２－２　トラックバー

アプリケーションが時間のかかる処理を実行しなければならないときには、プログレスバーで処理の進み具合をユーザーに知らせることができる。処理が進むにつれて、プログレスバーが強調色で左から右へ塗りつぶされていく。プログレスバーは、セットアッププログラムや大量のデータをある場所から別の場所にコピーするときによく目にする。たとえば、Windows 95で大きなファイルをコピーするときには、図２．２－３に示すダイアログボックスが表示される。このダイアログボックスにはプログレスバーのほかにアニメーションコントロールが含まれ、ユーザーのためにAVIクリップを再生することができる。

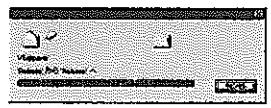

図２．２－３　作業中のプログレスバーとアニメーションコントロール

もう１つの汎用コントロールはアップダウンコントロールである。これを使うと、その相棒となるコントロールの項目をスクロールしたり、項目を変更することができる。通常、アップダウンコントロールはエディットコントロールと組み合わせてスピンボックスを作る。スピンボックスは、数、曜日、月などを回転して変更するためのものである。図２．２－４に例を示す。

図２．２－４　スピンボックス

（３）イメージリスト、リストビューコントロール、列見出し、ツリービューコントロール

リストを管理するユーザーインターフェイスの作成は込み入った作業になることがある。しかし、Windows 95の新しいリスト管理用コモンコントロールにより、これが非常に簡単になる。イメージリストはリストビューコントロールとツリービューコントロールとともに使用するもので、アイコン及びビットマップのリストを簡単に管理することができる。リストビューコントロールでは、データのリストをいろいろな方法で表示することができる。リストビューコントロールの列見出しによって、

各列の情報が何に関するものかを示すことができる（見出し項目をクリックして情報を並べ替えることもできる）。ツリービューコントロールは、データのリストを「ツリー」階層で表示する。これらのコントロールは、エクスプローラのユーザーインターフェイスの基本となっている。

（4）プロパティシート、タブ、ウィザード

プロパティシートは、項目のプロパティを通常は何枚かのプロパティシートページを使って表示する。たとえば、図２．２－５に示す［画面のプロパティ］プロパティシートでWindows95の画面の現在のプロパティを確認したり変更したりすることができる。ラベルの付いたタブコントロールのどれかをクリックすると、対応するプロパティシートページに切り替わる。すなわち、［スクリーンセーバー］タブをクリックすると［スクリーンセーバー］ページが表示され、現在のスクリーンセーバーの名前を確認したり、別のスクリーンセーバーを選択することができる。

ウィザードはプロパティシートの一種であり、ユーザーは一連のダイアログボックスの質問に答えることで処理を完了させることができる。ウィザードは、セットアップやインストールの際にとくに有用である。たとえば、Windows95で新しいハードウェアデバイスをインストールするときには、図２．２－６に示すハードウェアウィザードでセットアップを行うことができる。

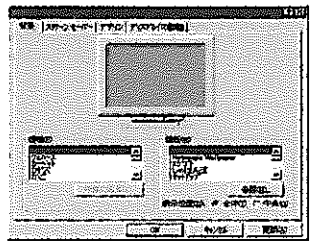

図２．２－５　タブのあるプロパティシート

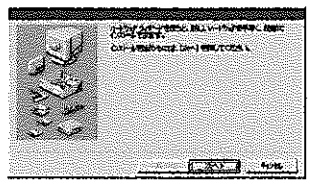

図２．２－６　ハードウェアウィザード

（5）リッチエディットコントロール

リッチエディットコントロールは、基本的にはテキスト編集ボックスであるが、それ以上のものである。複数行エディットコントロールに匹敵する機能を備えながら、文字と段落の両方の書式をテキストに適用することができ、ドラッグアンドドロップを行うことができ、OLEオブジェクトをテキストに埋め込むこともできる。リッチエディットコントロールを試してみるには、Windows95のワードパッドのテキスト編集エリアで試してみるとよい（図２．２－７）。

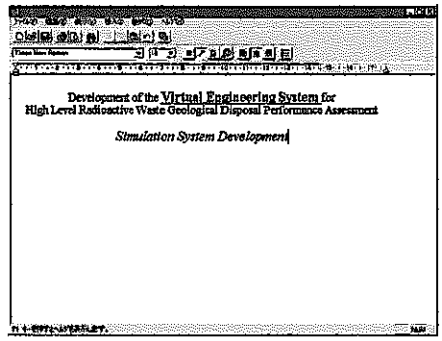

図２．２－７　ワードパッド

２．２．２　新しいコモンダイアログボックス

Microsoftは、Windows 95でコモンダイアログボックスのアップデートを行った。最大の変化はダイアログボックスの外見にあるが、いくつかの新しい機能にも気づくことだろう。図２．２－８に示す新しい［ファイルを開く］コモンダイアログボックスは、アップデートされたスタイルを示している。

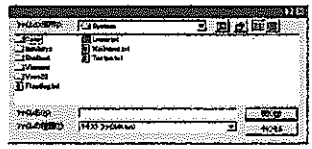

図２．２－８　新しい［ファイルを開く］コモンダイアログボックス

２．２．３　長いファイル名のサポート

［ファイルを開く］コモンダイアログボックスを見るとわかるように、扱いにくく、わかりにくい８文字のファイル名による時代は終わった。意味があり、覚えやすい名前をファイルに付けることができるようになった。

たとえば、「SINVCKOS.DOC」ではなく、「Summer Inventory Checklist and Order Sheet」というように。長いファイル名は、VFAT（プロテクトモードFATまたは仮想FAT）という Windows95の新しいファイルシステムによって実現されている。VFATシステムは古いFAT（File Allocation Table）システムと互換性があり、拡張されたFAT構造を使用して長いファイル名とファイル情報を格納する。

２．２．４　ショートカット

よく使うプログラム、ドキュメント、プリンタのショートカットを追加したり、仕事の進め方や個性に合わせて、見た目を変えてデスクトップをカスタマイズすることができる。

ショートカットにより、よく使う項目へ簡単にアクセスできるようになる。たとえば、ドキュメントのショートカットをセットアップすれば、ショートカットをダブルクリックするだけでドキュメントを開くことができる。ドキュメントの完全な名前やシステム上の正確な位置を覚える必要はない。プリンタへのショートカットをデスクトップに作成すると、ファイルアイコンをプリンタアイコンに

ドラッグするだけでファイルを印刷することができる。

Windows95では独自のショートカットを簡単に作成することができ、アプリケーションに含めることもできる。図２．２－９は、デスクトップにあるショートカットである。これを使用すると、筆者が最も頻繁に実行するアプリケーションの１つ、レジストリエディタに簡単にアクセスすることができる。

図２．２－９　ショートカット

２．２．５　ファイルビューア

ファイルビューアを使うと、ファイルを作成したアプリケーションを開くことなく、ファイルの内容を参照することができる。ファイルビューア自体が、メニュー、ツールバー、ステータスバー、及び標準のウィンドウコントロールといった要素を含んだインターフェイスを提供する。ファイルビューアを表示するには、ファイルを選択してから、エクスプローラの［ファイル］メニューから［クイックビューア］を選択する。あるいは、ファイルを選択してから、右クリックでコンテキストメニューを表示し、そこから［クイックビューア］を選ぶ。独自のファイルビューアを作成して、アプリケーションに含めることができる。

２．２．６　ユーザーインターフェイス拡張

インターフェイス拡張は、ファイルオブジェクトを操作するための追加の手段を提供したり、追加の情報を提供して、ユーザーインターフェイスの基本機能を拡張する。また、インターフェイスの拡張によって、アプリケーションからのWindows95のファイルシステム及びネットワークのブラウズが容易になる。Windows95には、以下の７つのユーザーインターフェイス拡張が組み込まれている。

コンテキストメニューハンドラ。
ドラッグアンドドロップハンドラ。
アイコンハンドラ。
プロパティシートハンドラ。
コピーフックハンドラ。
ドロップターゲットハンドラ。
データオブジェクトハンドラ。

２．３　アップル・マッキントッシュ対話システム開発技術

Apple Macintosh系対話システム開発の技術評価は、アップル社の提供するMac OS（我が国では漢字トーク）を基本とした対話システムの設計開発技術と、開発環境を対象としたマッキントッシュのソフトウェア開発にかかわる固有で基礎的な事項を整理する。

マッキントッシュを始めとして、グラフィック・ユーザーインターフェイスを持つシステムでは「ビットマップ」を使って画面上の表示を行う。ビットマップとは、画面上に示されている、すべての画素一つ一つに対応してメモリ内に（１つあるいは複数の）ビットが用意されているということを意味する。対応するメモリ内のビットが各ピクセルの状態を決める。白黒しか表示できないシステムでは、各ピクセルのオン、オフがメモリに記録される。カラーのシステムでは、各ピクセルの色が複数のビットを使って記録される。これとは対照的に「キャラクタマップ」のシステムでは、プログラムは画面上のピクセルは制御せず、文字単位で制御する。文字は決まった位置にだけ表示され、通常は１画面に80文字×25行表示される。

ビットマップシステムでは、ピクセルを（20、75）といった座標を使って指定する。１番目の座標はピクセルの水平方向の位置、２番目の座標は垂直方向の位置を示す。画面の左上の隅を点（0、0）として各ピクセルの座標が決められ、水平方向の値と垂直方向の値を決めることによって、画面上の任意のピクセルを指定できる。

画面上に何かを描く（描画する）には、まず開始位置を指定し、続いて描画操作を実行する。以下のその例を示す。

```
MoveTo（30、50）；　　／＊　点（30、50）への移動　　＊／
Line（0、100）；　　／＊　下方向へ長さ100ピクセルの線を描く　＊／
```

テキストベースのシステムと違って、ビットマップのシステムでは画面上の任意の位置にテキストを描くことができる。テキストについて「描く」という表現を使った。マッキントッシュにとっては、テキストを書くことと図形を描くことはほとんど同じ意味を持っている。どちらの場合も、望みのものを描くようピクセルが点灯するだけのことである。図２．３－１はテキストと図形、そしてそれを拡大して、使われているピクセルの様子を表したものである。

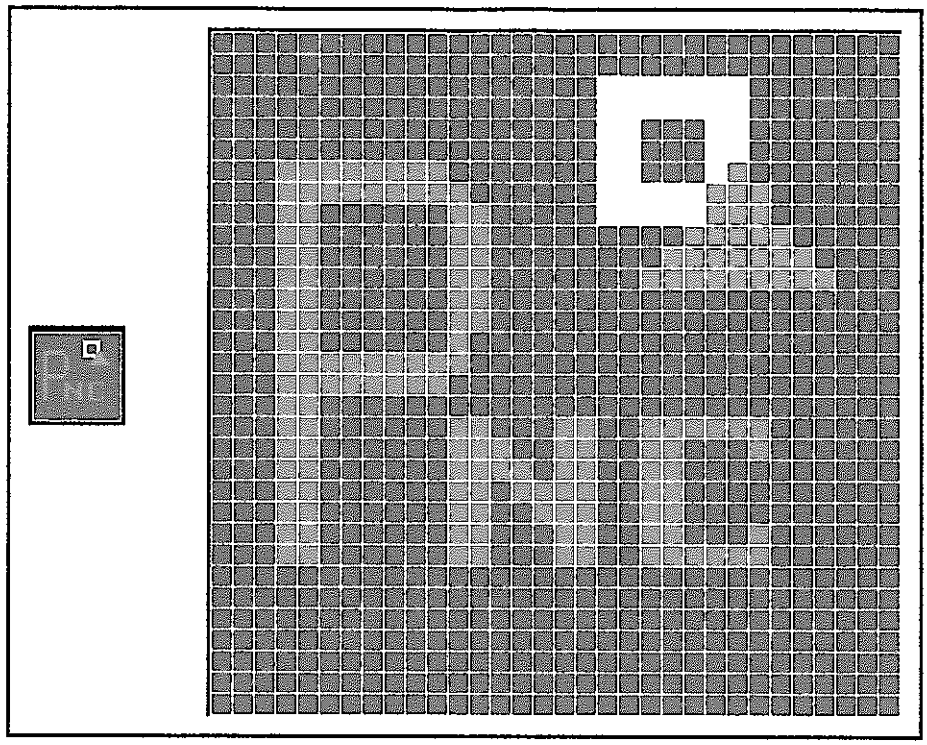

図２．３－１　ビットマップグラフィックス

図２．３－１を見れば、ビットマップを使うと文字と図形を混在させることが簡単になり、画面上の好きなところに図形やテキストが置けることがわかる。

２．３．１　イベント駆動型プログラミング

グラフィック・ユーザーインターフェイスを使わないプログラムは、いつも同じような順序で逐次的に実行される。たとえば４つの画面を表示する図２．３－２のようなプログラムの場合、４つの画面をあらかじめ決まった順序で表示するのが普通である。

図２．３－２　従来型のプログラム構造

この方式のプログラムとマッキントッシュ流のプログラムとの一番大きな違いは「事象：イベント」という概念である。キーを押したりマウスのボタンをクリックしたりするとイベントが生成される。イベントが生成されるとシステムソフトウェアは、自動的にそのイベントに関する情報を「イベントレコード」に記録する。具体的には、これらの情報がいくつかの「フィールド」に記録されるのである。マウスクリックであれば、まずそのイベントのwhatフィールドにマウスのクリックであるという情報が記録される。つまりwhatフィールドにはイベントの種類が記録されるのである。同じようにwhereフィールドにはマウスがクリックされた画面上の位置が記録される。

グラフィック・ユーザーインターフェイスを使うプログラムでは、必ずしも一定の順序で物事を行う必要はないし、情報を表示するのにいつも画面全体を使う必要もない。そのかわりにウィンドウを使うのである。ユーザはウィンドウを好きな順番で眺めることができる。マッキントッシュでは、ウィンドウの選択にキーボードやマウスなどの指示装置が使われる。図２．３－３にマッキントッシュのプログラムの構造を示す。

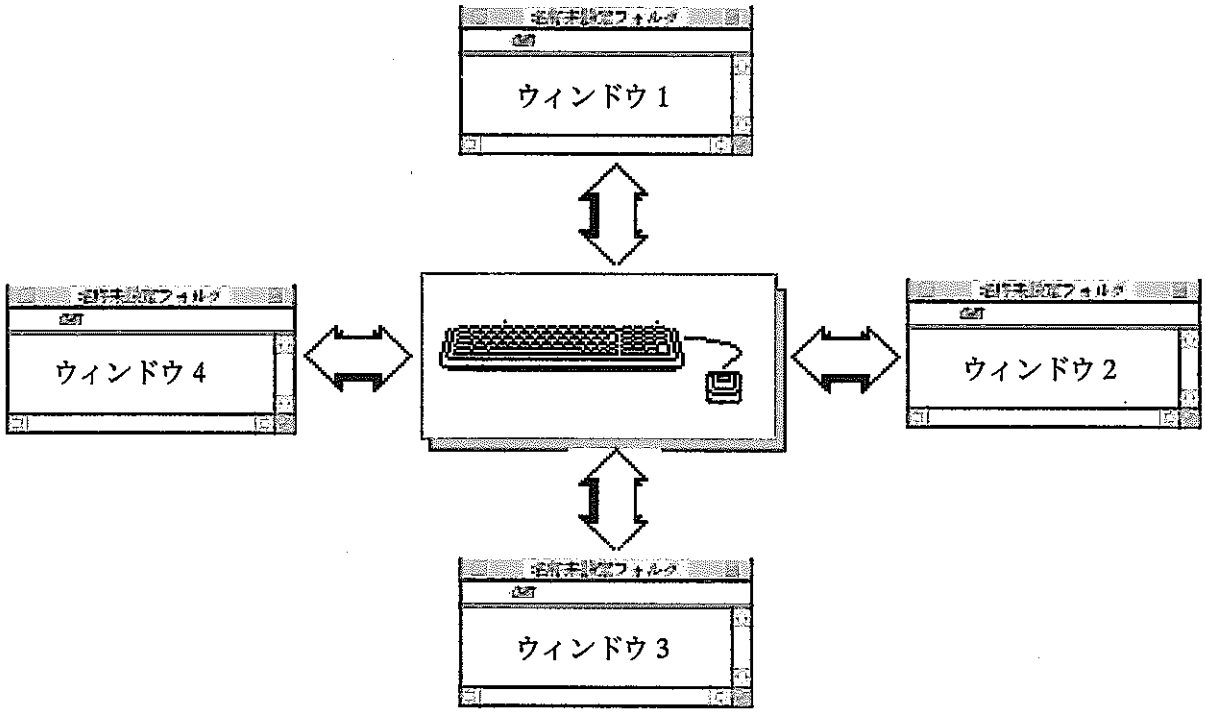

図２．３－３　マッキントッシュのプログラム構造

マッキントッシュのプログラムはすべてメインループによって制御される。このメインループは、休むことなくイベントを取り出しては処理するものである。イベントが生じると、それは「イベントキュー」に入り、キューに入ったイベントは「イベントループ」によって処理される。次の例はあるイベントループを表したものである。

```
EventRecord The Event;
Boolean N ot_Done = TRUE;
while (Not_Done)
{
        GetNextEvent( everyEvent, &The_Event );
        switch (The Event.what)
        {
                case mouseDown:
                    Handle Mouse Down();
                    break;
                case keyDown:
                        Handle_Keystroke();
            }
    }
```

GetNextEvent()によってイベントを１つ取り出し、それをイベントレコードを表す変数The_Eventに代入する。それからイベントの型（EventRecordのwhatフィールド）に従ってそのイベントを、処理する。このサンプルプログラムは、マウスのクリックとキー入力という２つのイベントだけを処理するものである。それぞれのイベントを扱う関数、HandleMouseDown()あるいはHandle_Keystroke()を呼び出して処理している。このような「イベント処理ルーチン」はプログラマが作成する。

例えば、MSWindowsのプログラマにとっては、ループの中でCetNextEvent()を呼んでイベントを取り出す手法は一般的でと考えられる。Widowsでは、ループの中でCetMessage()を呼び出してメッセージの有無を調べる。大きな違いは、マッキントッシュではすべてのアプリケーションが１つのイベントの列を扱うのに対して、Windowsでは各ウィンドウが自分専用のメッセージストリームを処理するという点である。図２．３－４はこれを表したものである。

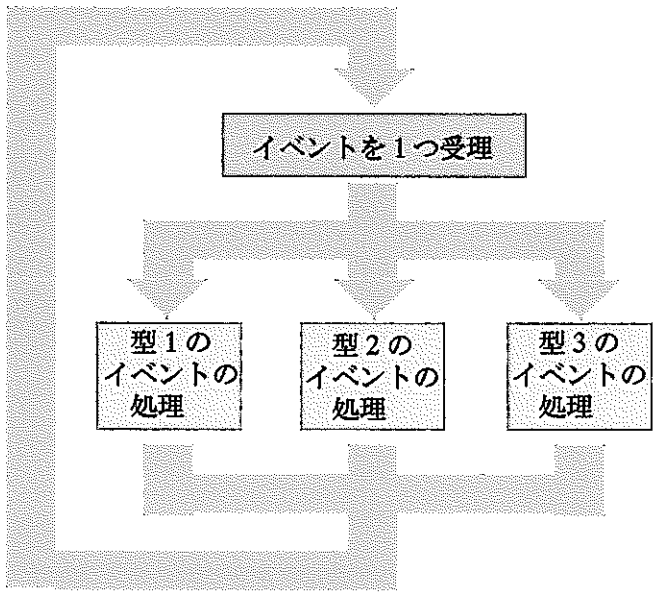

図２．３－４　イベント駆動型プログラムの構造

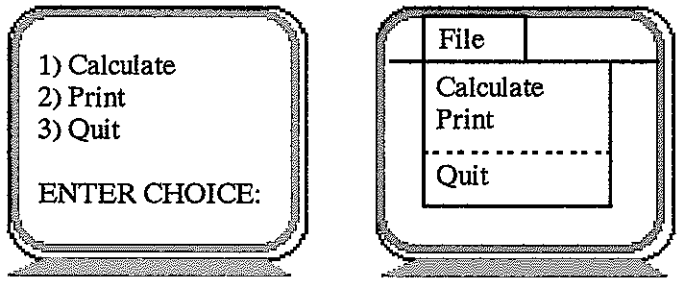

図２．３－５　イベント駆動型でないプログラムとイベント駆動型のプログラム

このようなプログラムでは、次のようにscanf() 関数を使ってキーボードの入力を取り出してから、その入力に応じた処理を行うループを使う。例えば下記の通りである。

```
Boolean All_Done = FALSE;
int cho i ce ;
while (All_Done == { FALSE )
{
        scanf ( "'-=.d", &choice) ;
        switch (choice)
        {
            case 1:
                    Do_Calculations ( ) ;
                    break;
            case 2:
                     Print_Results ( ) ;
                    break;
            case 3:
                    All_Done =  TRUE ;
                    break;
        }
}
```

図２．３－５の右側は、マッキントッシュによる選択肢の提示方法を示している。scanf（）を使う場合ユーザの入力を待ってそれに反応するが、これは本当の意味でイベント駆動型ではない。限られたメニュー（選択肢）だけを表示して、ユーザをその画面で待たせてしまっているからである。

それに対し、マッキントッシュのプログラムではキーボードからの入力やマウスのクリック、ディスクの挿入などすべてのイベントに気を配っている。もっとも重要なことは、ユーザの動作によってのみ、イベントの種類とそれが生じる時刻が決定されるということである。このような自由をユーザに保証するため、マッキントッシュのプログラミングではイベントとイベントループがたいへんに重要な要素となっている。

２．３．２　リソース

メニュー、ウィンドウ、ダイアログ、アイコンなど、プログラムのインターフェイスを構成するものはすべてリソースと呼ばれる。リソースはソースコードの一部分ではない。ソースコードではリソースの存在を意識しており、それに依存することはあるが、リソースはソースコードには含まれない。従って、インターフェイスの各部分をリソースとして作っておくと次のような利点がある。

＊リソースは、プログラミングの知識がなくてもグラフィカルに作成、編集ができる。
そして、プログラムの完成した後でも編集できる。
＊リソースを別のプログラムにコピーして再利用できる、

リソースを作るには、AppleのResEditなどのリソースエディタを用いる。リソースは「リソースファイル」に保存する。ResEditと、ResEditを使って作ったリソースを持つファイルのアイコンを図２．

3－6に示す。

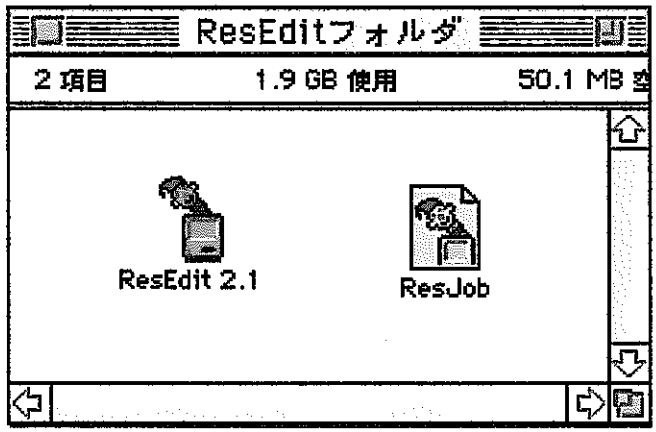

図2．3－6　ResEditとResEditファイルのアイコン

プログラムのインターフェイスを構成している要素は、それぞれ独自の型を持っている。リソースのタイプは長さ4の文字列で表し、大文字と小文字は区別される。たとえば、メニューのリソースタイプは、‘MENU’である。ResEditなどのプログラムを使うと、アプリケーションのインターフェイスを構成する各要素に対応するリソースを1つ1つ視覚的に作っていくことができる。図2．3－7に‘MENU’リソースの作成の様子を示す。メニュー項目を定義するソースコードを書かずに、ResEditを使って‘MENU’リソースを作ることができる。

MSWindowsではこの考え方は、BorlandのResource WorkshopやMicrosoftのAppStudioなどのリソース編集用ツールによる。ResEditはAppleのリソース編集プログラムである。Windowsの場合と同様、リソースを視覚的な方法でも、テキストを使って表現したものをコンパイルする方法でも作れる。マッキントッシュではResEditを使って視覚的に作るプログラマが多いという点がWindowsとは異なっている。従って、マッキントッシュの画面上にあるものはすべてリソースからできている。

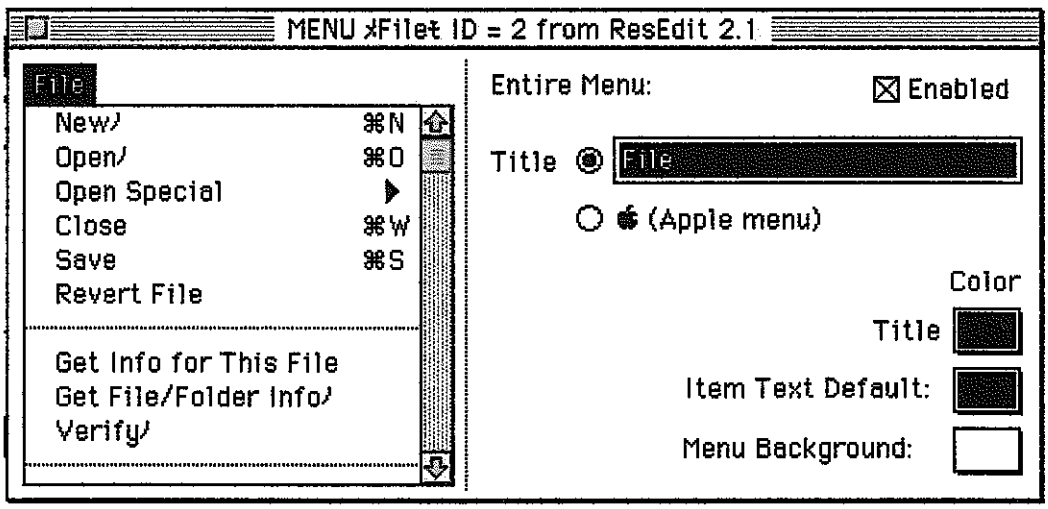

図2．3－7　ResEditを使った‘MENU’リソースの編集

＊メニューバー用には‘MBAR’リソースがあり、これでメニューを指定する。
＊各メニュー用にはそれぞれ‘MENU’リソースがあり、これがメニュー項目を定義している。
＊ウィンドウ用には、‘WIND’リソースがあり、その大きさと初期位置を定義している。
＊ダイアログ用には‘DLOG’リソースがあり、大きさと初期位置を定義している。
＊ダイアログ用にはもう１つ別のリソースもあり、ダイアログボックスに表れるボタンなどの項目を定義している。この様子を図２．３－８に示す。

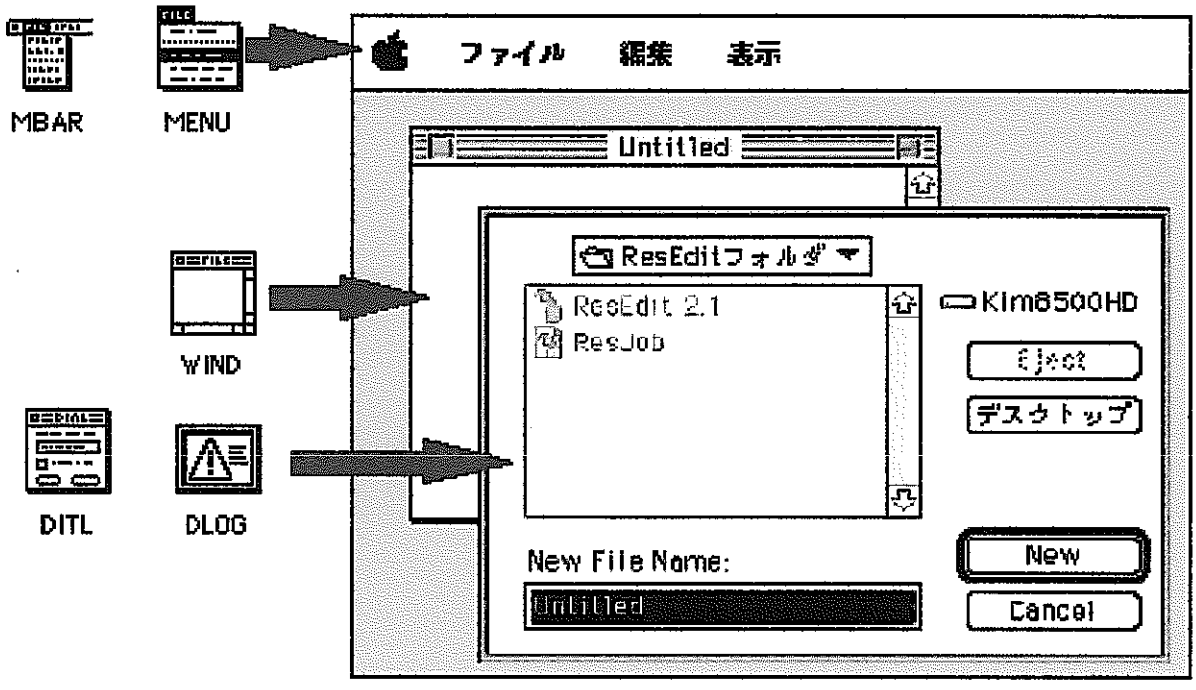

図２．３－８　画面の構成要素とリソースの関係

プログラムを作る際には、画面を構成する要素のリソースを作っておき、これをソースコードから読んで利用する。

THINKCやSymantecC＋＋のコンパイラでは、他の多くのコンパイラ同様、１つの環境の中にソースコードエディタ、コンパイラ、そしてリンカが統合されている。ソースコードを独立したアプリケーションに変換するには、THINKCやSymantecC＋＋でコンパイルし、このコンパイルされたコードをリソースファイルのリソースと結合する。その結果としてアプリケーションができあがるわけである。図２．３－９にこの様子を示す。

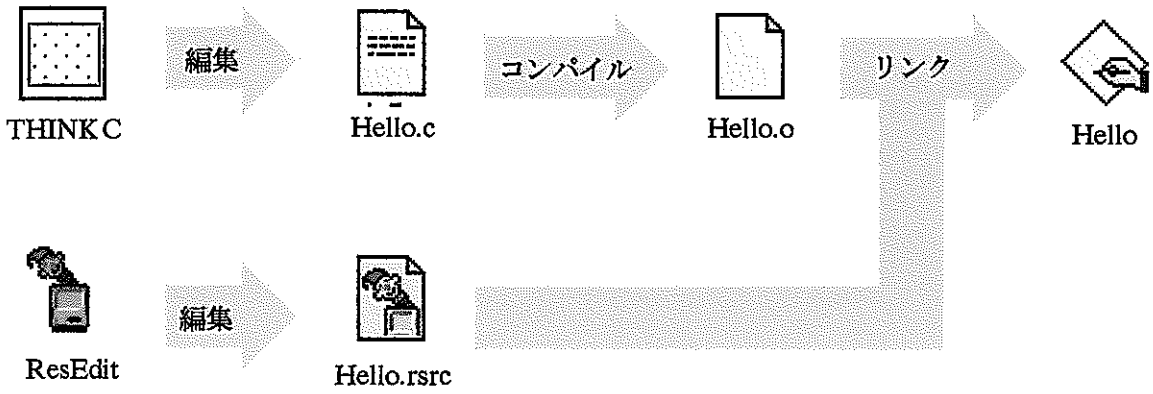

図２．３－９　ソースコードとリソースの関係

THINKCなどの開発環境では、図２．３－９のHello.oのようなファイルはFinder上には現れない。「プロジェクトファイル」にオブジェクトコードをすべて保存する。ソースコードと最終的なアプリケーションさえあればよいので、オブジェクトファイルを別に作る必要はないわけである。

プログラムをリンクする際に、リンカがオブジェクトコードを‘CODE’リソースに変換し、これを別に作ったリソースといっしょにして最終的なアプリケーションを完成する。つまり、マッキントッシュのプログラムは大部分がリソースから構成されているといっても間違いではない。リソースとソースコードが最終的にいっしょになるのなら、なぜ別々のファイルに入れておくのかについては、ソースコードはテキストエディタで作られるが、リソースはリソースエディタで作られるからである。

２．３．３　Toolbox

ResEditのようなリソース編集ツールを使ってメニュー、ウィンドウ、ダイアログを作るのは簡単である。しかし、リソースの中身はインターフェイスを断片的に記述しただけで、それだけでは何も処理しない。たとえば、ResEditでメニュー項目を作るのは簡単であるが、そのメニューを表示するためには、つまりメニュー上のマウスの動きを追跡し、メニューをプルダウンし、その中の項目を表示するにはソースコードを書く必要があるのである。このようなメニューの扱いは、すべてのマッキントッシュプログラムで行われていることである。そして、これを実現するコードも、どのマッキントッシュプログラムでも似たりよったりのものが使われる。すでにあるものを「再発明」する必要はない。というのはプログラミングにも通じるもので、マッキントッシュではこれが極限まで押し進められている。アップル社のプログラマは、マッキントッシュのプログラムで共通に行われるような動作を扱う数千個に及ぶルーチンを作成し、それを開放している。

アップル社では、プログラミング言語でよく行われるようにライブラリとして提供するのでなく、ルーチンのコードをROMチップに焼き付けて、マッキントッシュの内部に組み込んだ提供形態としている。こういったルーチンの集合をマッキントッシュ・ユーザインターフェイス・ツールボックス、または略してツールボックス（Toolbox）と呼んでいる、

パソコンやメインフレームのプログラマで、ウィンドウ環境でのプログラム経験のない人でも、このような考え方に惑わされる必要はない。メインフレームやパソコンでも、他の人が書いたルーチン

を使うことはある。Cの標準ライブラリのstrlen() やprintf() などがこの例である。

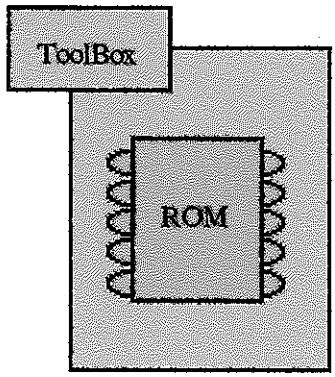

図２．３－１０　ROM内のToolbox

図２．３－１０は、Toolboxのルーチンのコードが ROMに入っていることを図に示したものである。MS - DOSのパソコンにもROMに組み込みまれたソフトウェア（ROMBIOS）がある。ただし、マッキントッシュのToolboxの方が簡単に使え、グラフィックスの表示の手段や洗練されたユーザインターフェイスを実現するための機能が整理して組み込まれていると言える。

前項までリソースを作り、次にリソースを使うソースコードを書く手法を検討した。このためにはソースコードが何らかの形でリソースとやり取りする必要がる。Toolboxルーチンがこの役割を果たすものである。

```
WindowPtr  the_Window;
the_Window  =  GetNewWindow(128,0L,(WindowPtr)-1L);
```

ルーチンGetNewWindow()は、IWINDリソースの場所を探して、それをメモリにロードするToolbox関数である。GetNewWindow() のコードはROMの中にありソースコードでGetNewWindow() を呼び出すと、ユーザプログラムの実行が一時中断されてROM内のコードが実行される。

GetNewWindow() を呼び出すと、そのウィンドウを指すポインタWindowPtrが返される。

このWindowPtr型の変数を使って、特定のウィンドウを識別するわけである。ウィンドウの移動やウィンドウ内へのグラフィックスの描画等の操作をするときには、そのウィンドウを指すウィンドウポインタを指定する。

ROM中にあるGetNewWindow() のコードは、プログラムの‘WIND’リソースを見てウィンドウの型と大きさを決める。例では、GetNewWindow() に渡されている引数がある。最初の引数は、ウィンドウのIDである。リソースファイルが複数の１WINDリソースを持つ可能性があるため、この引数でどの‘WIND’リソースを使うかを指定する。Toolboxルーチンは、ウィンドウについての情報を探してリソースファイルを見にいくと、リソースファイルから必要な情報がToolboxに渡される。

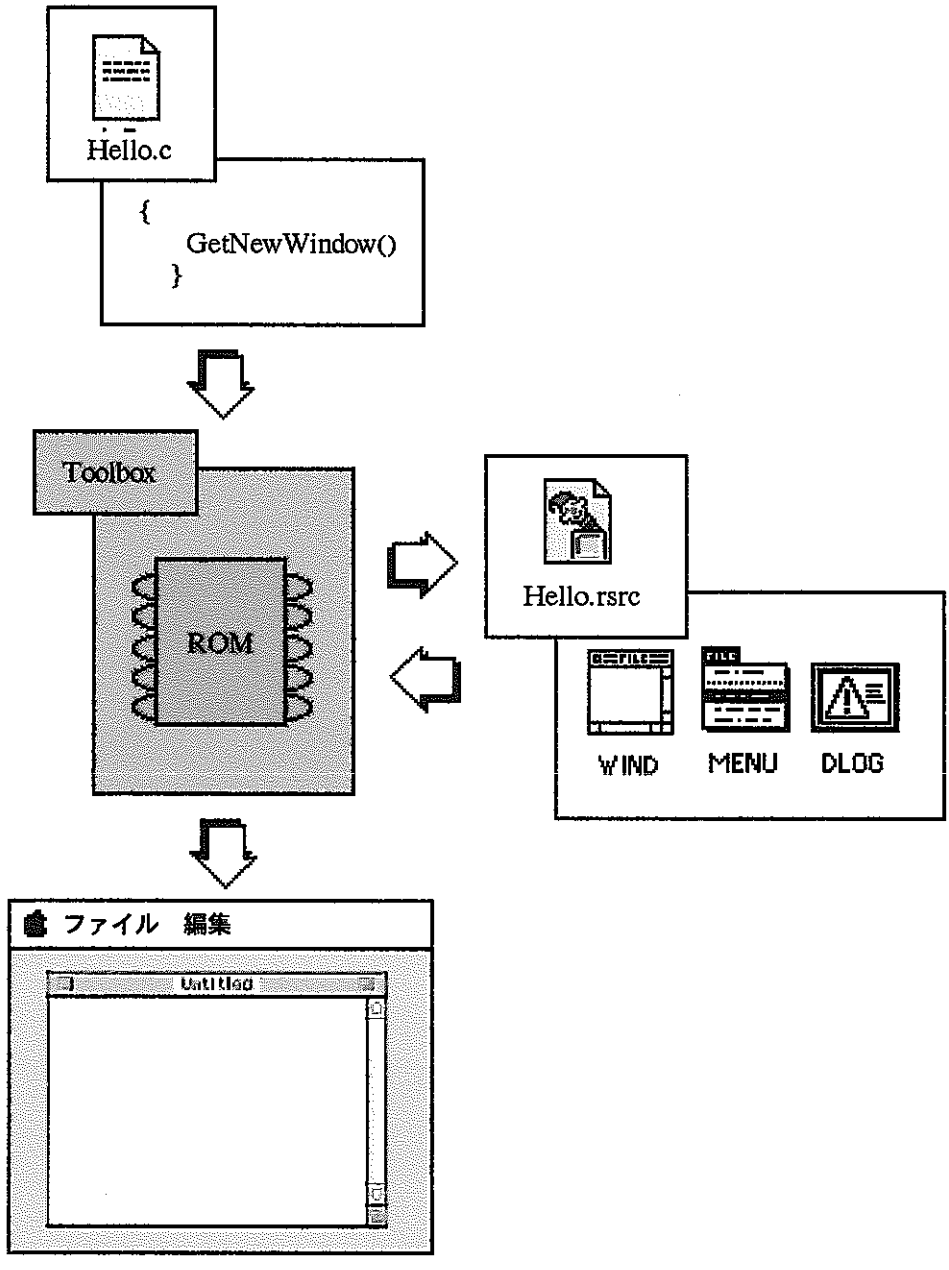

図２．３－１１　ソースコード、リソースとToolbox

図２．３－１１はソースコード、Toolbox、そしてリソースの間の関係をまとめたものである。Get New Window() の呼び出しによってToolboxがアクセスされ、次いでToolboxからプログラムのリソースを見にいって、ウィンドウの型や大きさに関する情報を得る。この情報は‘WIND’リソースの中に入っている。GetNewWindow() ルーチンはToolboxにある何千という関数の一つに過ぎない。

２．３．４　オペレーテイングシステム

Toolbox同様、マッキントッシュのオペレーティングシステム（OS）を構成しているコードもROMの中にある。図２．３－１２はこれを示したもので、図２．３－１０に似ている。OSは、次の項で説

明するシステムフォルダの中にあるシステムファイルとは別のものである。

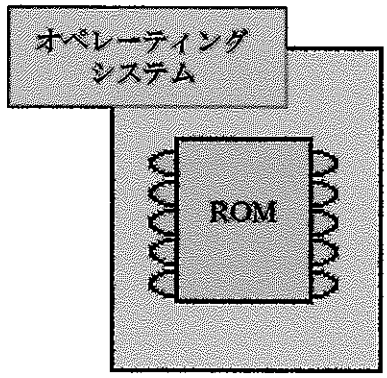

図２．３－１２　ROM内のオペレーティングシステムのコード

OSはToolbox同様、ソースコードから関数呼び出しとしてアクセスできるルーチンで構成されている。ToolboxルーチンとOSルーチンは、実行するタスクのレベルが違う。OSルーチンはキー入力やディスクの挿入といった低レベルのタスクを扱うが、Toolboxルーチンはより高いレベルのタスクを扱うのである。ウィンドウの表示、ウィンドウ内の図形や絵の描画などの高レベルの作業の方がユーザの注目度は高くなる。

Toolboxのタスクを実行するのと同じように、OSのタスクも適当な関数を呼び出すことで実行する。Eject() はOS呼び出しの例で、フロッピーディスクドライブからディスクを物理的に取り出すものである。

PCやメインフレームのプログラマにとっては、マッキントッシュではOSへのアクセスが簡単だということわかるであろう。必要なタスクを実行するには、呼び出すべきOSのルーチンを知ってさえいればよいのである。ジャンプ命令や割り込みを使って直接メモリ番地を指定する必要はない。

２．３．５　システムソフトウェア

ここまででToolboxのルーチンとOSのルーチンについて検討した。この２つはまとめてシステムソフトウェアと呼ばれる。マッキントッシュでは、このようにシステムソフトウェアは大きく分けて、ToolboxルーチンとOSルーチンの２つに分けられることになるが、機能的に関連したルーチンのまとまりごとに「マネージャ」と呼ばれるグループも形成している。

ウィンドウマネージャはマネージャの例で、GetNewWindow() やMoveWindow() など、ウィンドウに関連するルーチンからできている。この他のマネージャとしては図２．３－１３のようなものがある。図２．３－１３から、Toolboxのマネージャはユーザインターフェイス、すなわちウィンドウ、ダイアログ、メニューなどを取り扱うことがわかる。一方、OSを構成するマネージャは、メモリ管理やモニタなどのデバイスの制御を始めとする低レベルのタスクを処理する。

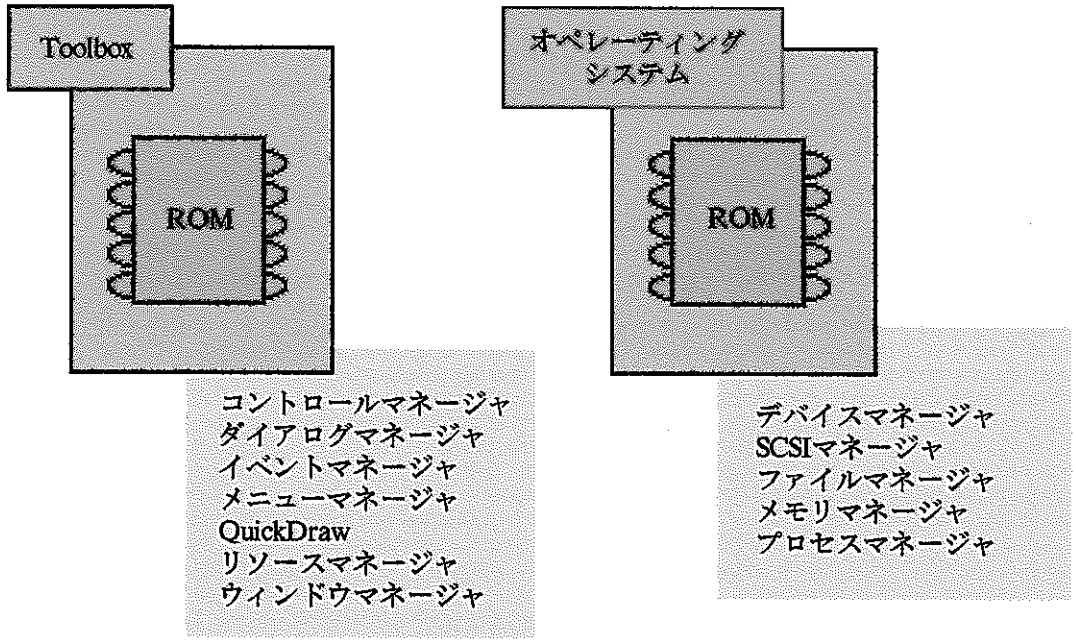

図２．３－１３　マッキントッシュのマネージャ

（１）SystemファイルとFinder

どのマッキントッシュにもシステムフォルダがあり、その中には必ずSystemファイルとFinderの２つが入っている。

SystemファイルはROMにあるOSとは別のもので、すべてのプログラムからアクセスできるリソースによって構成されている。新しい機種のROMには以前の機種にはないルーチンが入っている場合がある。古い機種を使っているユーザは、Systemファイルを更新すれば新しいToolboxやOSのルーチンにアクセスできるようになる。

Systemファイルには「パッチ」が入っており、このパッチの部分に新しい機種のROMに入っているルーチンの代わりをするコードが置かれている。古いROMが新しいシステムルーチンの呼び出しを受けた場合、Systemファイルのコードを実行するわけである（図２．３－１４を参照）。新しいROMを持っているときは、Toolboxが直接新しいルーチンを呼び出す。これに対し、古いROMの場合は新しいSystemファイルにアクセスすることになる。

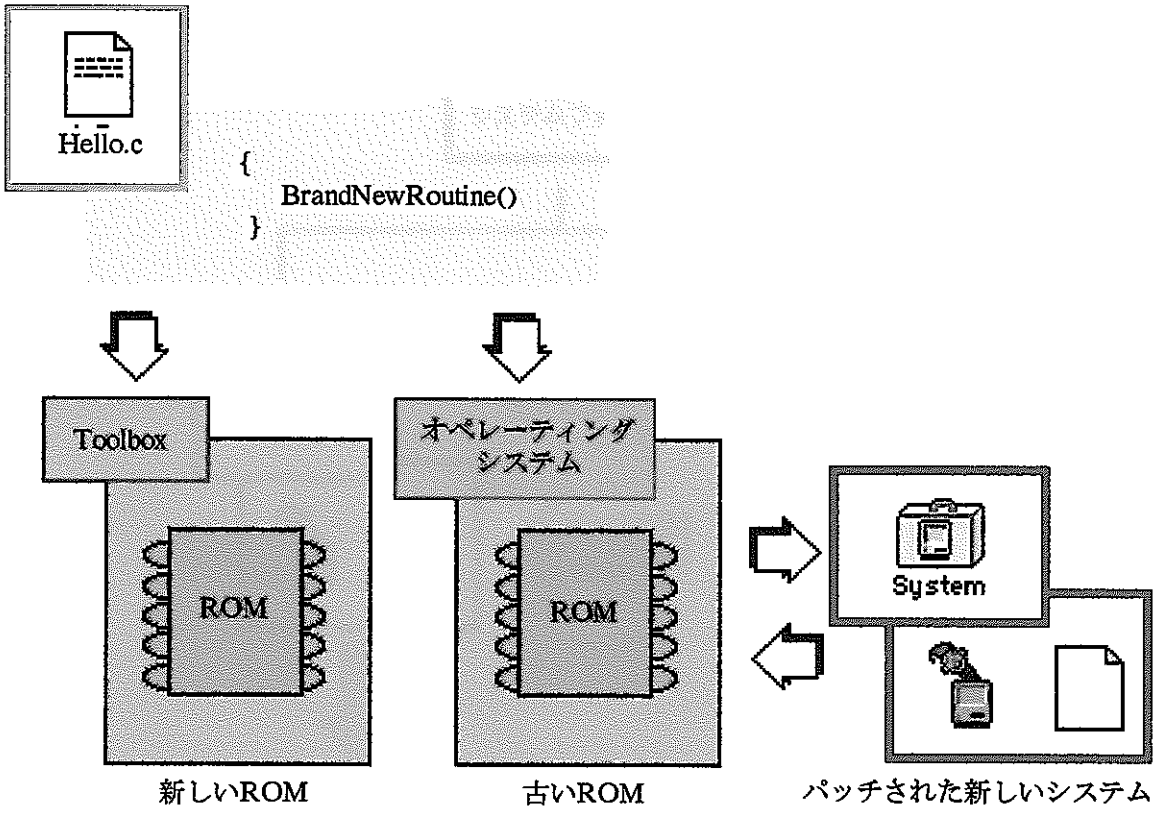

図２．３－１４　ROMルーチンの役目をするパッチ

古いROMと古いSystemファイルを使っているときに、新しいルーチンを呼び出そうとすると、そのアプリケーションはクラッシュする（動かなくなってしまう）ことになる。すでに数多くのマッキントッシュが市場に出回っているためシステム開発上の問題を生じる。

（２）Finder

Finderは、普通のアプリケーション同様、コードとリソースからできているプログラムである。

Finderは、ユーザがコンピュータの電源を入れると、メモリにロードされて動き始める。デスクトップのパターンやその上に表示されるゴミ箱、ファイル、フォルダなどのアイコンを表示する役目をする。また、ファイルの移動、コピー、削除などの際にも、Finderが必要な作業を行う。

FinderはSystemにある共有のリソースを使って画面表示を行う。図２．３－１５は、SystemファイルとFinderの関係とFinderの役目を示したものである。

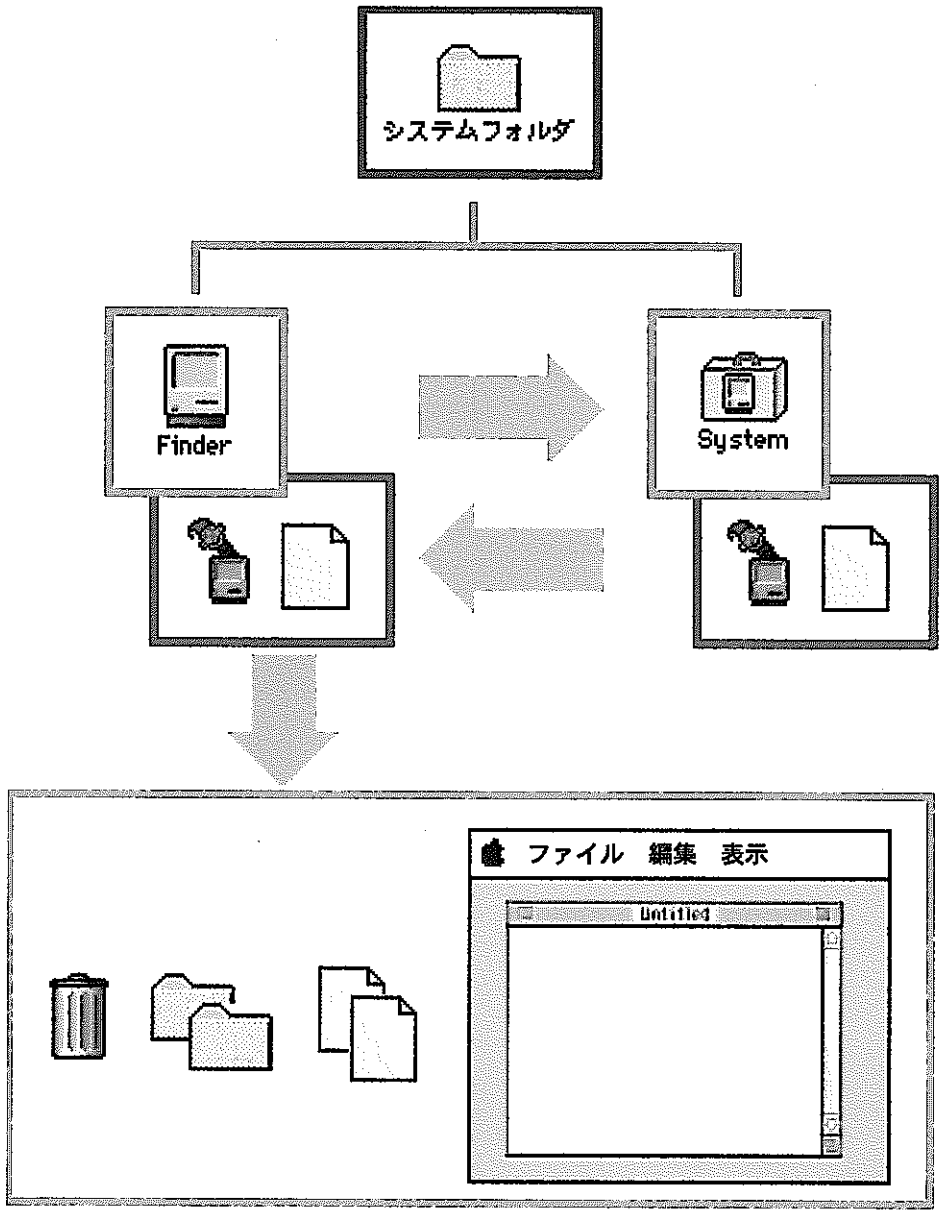

図２．３－１５　FinderとSystemファイル

3　ユーザーインターフェイスの試作

地層処分ヴァーチャル・エンジニアリング・システムのユーザー・インターフェイスについて試作を行った。試作にあたっては、ユーザーインターフェイスの操作性について基礎的な考え方をとりまとめた。

3．1　ユーザーインターフェイス設計に考慮した事項

ユーザーインターフェイスの様な人が意識する空間、特に二次元的な平面上に要素をどのようにレイアウトするかということは、効果的なビジュアル・コミュニケーションをめざす設計者にとっての基本的な考慮事項である。また時には、設計者は三次元空間におけるレイアウトをプランしなければならない場合もある。

グラフィック・ユーザー・インターフェイスで用いられる重なりのあるウィンドウやコントロール・パネル、ダイアログ・ボックスなどのリアルな影や、ハイライト、テクスチャのある表面などは三次元的なレイアウトの例といってよいだろう。三次元的なレイアウトのもっと端的な例には、VPLリサーチ社（カリフォルニア／レッドウッドシティ）やAutoDesk社（カリフォルニア／サウサリート）の人工現実感（Artificial Reality）、仮想現実感（Virtual Reality）と呼ばれるシステムがある。これらは、三次元空間に直接コンピュータ・グラフィックスのディスプレイ環境を構築している。

これらの技術は、今後仮想的な地層処分システムと天然地層環境をコンピュータの中に構築することを目指す地層処分ヴァーチャル・エンジニアリング技術の開発に有益な方向を与えると考えられる。

3．1．1　ディスプレイ画面上の形の配列

コンピュータのディスプレイ上であれ、通常のDTPなどによって印刷された紙面であれ、表示する情報が複雑になる程、よいレイアウトをするためには「グリッド」が欠かせないものになる。このグリッドは、通常あまり意識されないものだが、レイアウト上は非常に有効なものである。

レイアウトのためのグリッドとは、ある情報表示領域を視覚的、あるいは概念的にまとまりのある単位に区切るための水平、垂直線の集まりである。たとえば、本書のページは本文やイラストレーションの入る部分と、ヘッダーと呼ばれるページの上方に位置する本の名前やページ数を入れる部分とに大きく分けることができる。

グリッドは、紙面やスクリーンの構成を決める際に、グラフィック設計者やスクリーン表示を担当するプログラマをおおいに助けてくれる。これらのページやスクリーンを見る人は、普段それの存在にすら気がつかないものだが、視覚的な表現のすべてに見やすさ、読みやすさをもたらすものであるといってよい。

四辺形には、それぞれの年代や文化によって好まれるプロポーション（縦横の比率）の関係というものがある。例えば、アクロポリスにあるギリシャのパルテノン宮殿の幅と高さの比率はギリシャ古典建築における美しさの理想を表現している。しかし、ヨーロッパ、日本、ヒンドゥーなどの古典的な建築における美しさを構成している比率はギリシャのものとは異なっている。

また、ヨーロッパの建築様式の変遷をみれば、美しさそのものも時代によって変化して行くということがわかるだろう。とはいえ、建築や絵画から文字の設計やページレイアウトに至るまで、多くの異なる文化がつくる環境の中に、さまざまな視覚形式をとおして、好まれるプロポーションの傾向に

引き継がれている表現も見いだすことができる。図３．１－１に基本的なプロポーションのいくつかをあげておく。

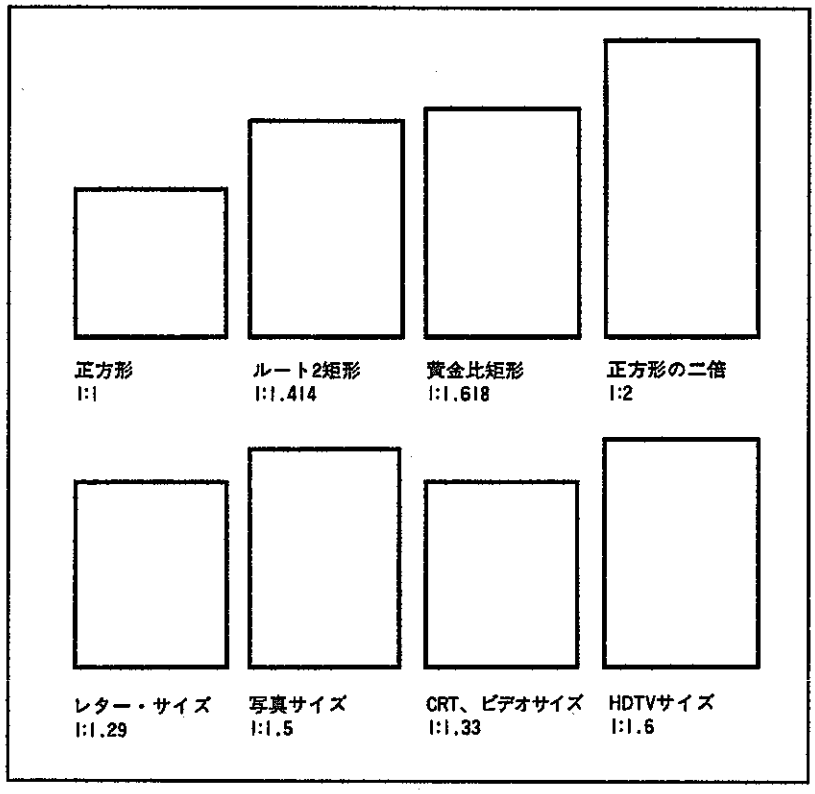

図３．１－１　伝統的なプロポーションと現代的プロポーション

この図の上列の長方形は、いろいろな文化や多くの年月を通してグラフィック設計や工業設計、建築設計で典型的、古典的プロポーションとして使われてきたものである。また、２番目の列は、現在工業的な慣習で用いられているプロポーションの例である。

正方形は最もシンプルな比率を持つ四角形である。正方形は、どのような視覚的な状況におかれても注意を引きやすい形で、安定と永続性を表す。また、これが斜めに回転した形、すなわちダイアモンド型は非常に動的な形となり、「動き」と「緊張」を表す。いまのところ、正方形のスクリーンというのはあまり生産されていないが、レイアウトのためのフォーマットとしては正方形は確立されたものといえる。また、正方形の紙面というのも経済的な理由から頻繁には使われないが、全体的なビジュアル・フィールドとしては、堂々とした存在感を与えるフォーマットといってよいだろう。

1対1.414（＝2の平方根）という比率の長方形（ルート2矩形）は、視覚上、内部に正方形を内接しており、四角形を自然に分割したい場合によく用いられる。この長方形の長手のほうを半分になるように折り畳むと、元の長方形と同じ1対1.414の比率を持つ長方形になる。この性質はこの長方形に独特のものだが、本のページ用のフォーマットとしては理想的なものといえる。この長方形でできた本を開くと、開いた本の全体とそれぞれのページの比率とが等しくなる。このテクニックは伝統的なヨーロッパや日本の書籍設計で用いられている。実際、この比率はヨーロッパ、日本では標準的な紙のサイズとしても採用されているので、通常の本や雑誌、手紙、ポスターなどは、皆この比率で作成され

ている。

1対1.618という比率の長方形は黄金比として知られており、古代から芸術家、設計者、哲学者などの間で好まれてきた。この比率には数学的な関係があり、植物の成長パターンや動物の生活にも関係している比率である。ギリシャ建築にはこの比率が用いられている。また、20世紀初めのフランスの建築家ル・コルビジェはこれを彼の建築要素のためのモジュラ・システムの基本比率とした。黄金比を持つ長方形は、内接する最大の正方形を切り離すと、残った長方形がやはり黄金比を持つという特徴がある。この特徴は黄金比を持つ長方形だけのもので、こうした操作を永久に繰り返しても損なわれない。

日本人には畳の比率としてなじみ深い正方形を二つ合わせた1対2の長方形は、西欧人にはあまり用いられてないプロポーションである。1対2以上に長い長方形は、辺の比率による形の識別という点からみると、次第に識別しにくいものになる。

これらの基本的な長方形の辺の比率は、視覚フィールドの外形やその主要な構成要素、ならびにイラストレーションやテキストブロックなどの比較的小さいエリアに用いられる。典型的な視覚要素の例には、チャート、フォーム、テキスト、イラストレーションなどがあるが、これらは紙の上やプレゼンテーション用スライドのレイアウト、また、スクリーン・ディスプレイ用のユーザー・インターフェイスを成立させるためのものでもある。

以上の有効なプロポーションを用いるに際には、ビジネスにおける情報表示用のディスプレイに用いられる四角形の多くが、必ずしも上で述べたような比率を持っていないことを知っておく必要がある。図３．１－２のフォーマットは、一般的に用いられる商業的、工業的製品の視覚フィールドの外形を決めるためのものである。

この中で、ハイビジョン・テレビが新しく採用した幅広のフォーマットだけは、伝統的なプロポーションに近い値に設定されていることは注目に値する。設計者は非伝統的なプロポーションを使うときにも、伝統的なプロポーションを使ってフォーマットを分割するということを行う。たとえば、横長のスライドを使ってグラフなどを表示する場合、画面を等分に分割するのではなく、右から正方形の領域を確保してそのエリアにメインタイトルとデータを表示し、残った左側に凡例やグラフの単位などを表示するなどである。

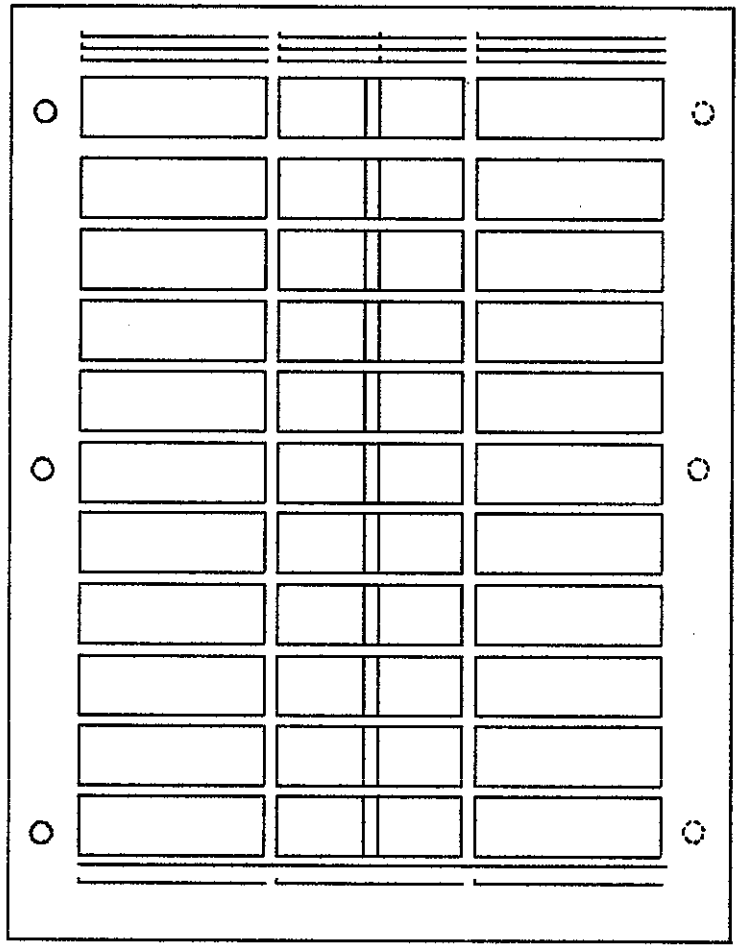

図３．１－２　ページレイアウト・グリッド

現在有るディスプレイのプロポーションの不統一は、周囲に十分なマージン（余白）を持っていなければならないことを意味している。また、異なるメディアに汎用的に使用するような視覚的レイアウト構成とする時にはなおさらである。四周のマージンを表す線は、実際に平面上の多くのレイアウト・グリッドの基準線となるものである。マージンの外に色をのせたり、記号類を表示できないというわけではない。これは、作ったものが当初とは異なるメディアに移されたときに、マージンの外においた要素が表示面から切られてしまう可能性があるという意味である。

本や雑誌、レポートなどのマージンを決める際には、綴代も考慮しなければならない。（図３．１－２参照）たとえば、数ページにわたる文書やレポートなどで、３穴のパンチを使用するのであれば、少なくとも16ミリ程度の余裕を見ておく必要があるだろう。

スクリーン・レイアウトを考えるに当たって、オフィス・オートメーションや電子出版用のワークステーションに採用されている、60から150ppi（Pixel per inch）の高解像度ディスプレイは標準的なレター用紙を表示できなければならない。多くの場合、縦型ページの幅方向は実際のサイズを維持することができるが、高さに関しては多少蹴られてしまう。これはディスプレイの解像度の限界によるものである（100ppiのディスプレイにおける1024ピクセルの高さは、米国での標準的な紙の高さである1

1インチではなく10.24インチになってしまう。）。標準的には、横型ディスプレイの残った右側のエリアはコマンドやコントロール用のメニュー、ないし状態表示などに使われる（訳注：日本における標準的なサイズであるA4用紙は、210×297ミリであるから、これを表示するには100ppiのディスプレイを用いたとすると、827×1170ピクセル必要ということになる）。

スクリーンやページ・レイアウト用のグリッドの多くは複数のコラムを持っている。ここで、コラムの幅はどのように決めたらよいのだろうか。まず初めに、使用する文字のサイズと一行に何文字入るようにすればよいかを決めなければならない。英文の場合、ジャスティファイ（訳注：単語間のスペースの幅などを調節して各行の右端が揃うように処理すること）されているいないにかかわらず、一般的には40から60文字が一行に入れる最大の文字数であるとされている。コラム間（カッターという）は、やはりある程度スペースをとる必要があるが、最低でも2文字分の幅をあける。また、印刷物では1から2picaあけておく。（1picaはおよそV6インチ＝4.2ミリ）そして、ページやスクリーンの標準的なコラム数は2から5である。これは縦型のときも横型のときも同じである。ときによっては、主コラムをフォーマットの右端に配置し、コメントやサブタイトルや注書きのためのやや狭いコラムを左に配置するということも行われる。

コラムは水平のグリッド・ラインによって、欄外の見出しや主タイトル、メインとなる本文そして脚注などに分断される。これらの水平線に関連して、テキストをどこに配置するかを決めるにはいくつか方法があるが、もっとも一貫性のある方法としては、各水平線がテキスト行の位置を示していると仮定することである。この場合、空間的に強調したいようなときは、たとえばサブヘッドの前を一行あきにするなどの処理をする。この方法は、全てのページないしスクリーンを通してすべてのテキストがきちんと一貫性をもって揃うやり方である。

テキストの入るエリアの周囲には適度な「息抜き」のスペースが必要である。特にマルチウィンドウのディスプレイなどのレイアウトにおいて見落とされがちな点として、しばしばテキスト・エリアのまわりにマージンが取られていない場合がある。そのようにすると、テキストはウィンドウの枠を越えて次のウィンドウのテキストとつながって見えてしまう。このようなレイアウトはスクリーンを不必要に狭苦しく、乱雑な感じに見せてしまう。簡単な解決方法としては、ウィンドウの表示内容とウィンドウの枠との間に１文字分のスペースをあけることである。これは固定幅のキャラクタ・ディスプレイにおいても同様である。また、ページ・レイアウトの場合と違って、スクリーン上では、通常、他のウィンドウとの境界線には、間に空間のある二重の線を使うより、一本の線を使うほうが複雑に見えないようである。この効果は、スクリーン上ではあまり細い線を表示できないことに起因している。

通常のイラストレーションのレイアウトを考えるに当たっては、コラムの幅や、そのエリアを決めるための水平垂直線の配置を参考にする。これは、そこに入るものが図や写真、ダイヤグラム、テキストであっても同じである。水平線は通常、テキストの一行ごとのラインによって決まるものである。実際、レイアウト・グリッドのもっとも重要な点はイラストレーションのサイズ、プロポーション、位置に変化と一貫性を与えることである。

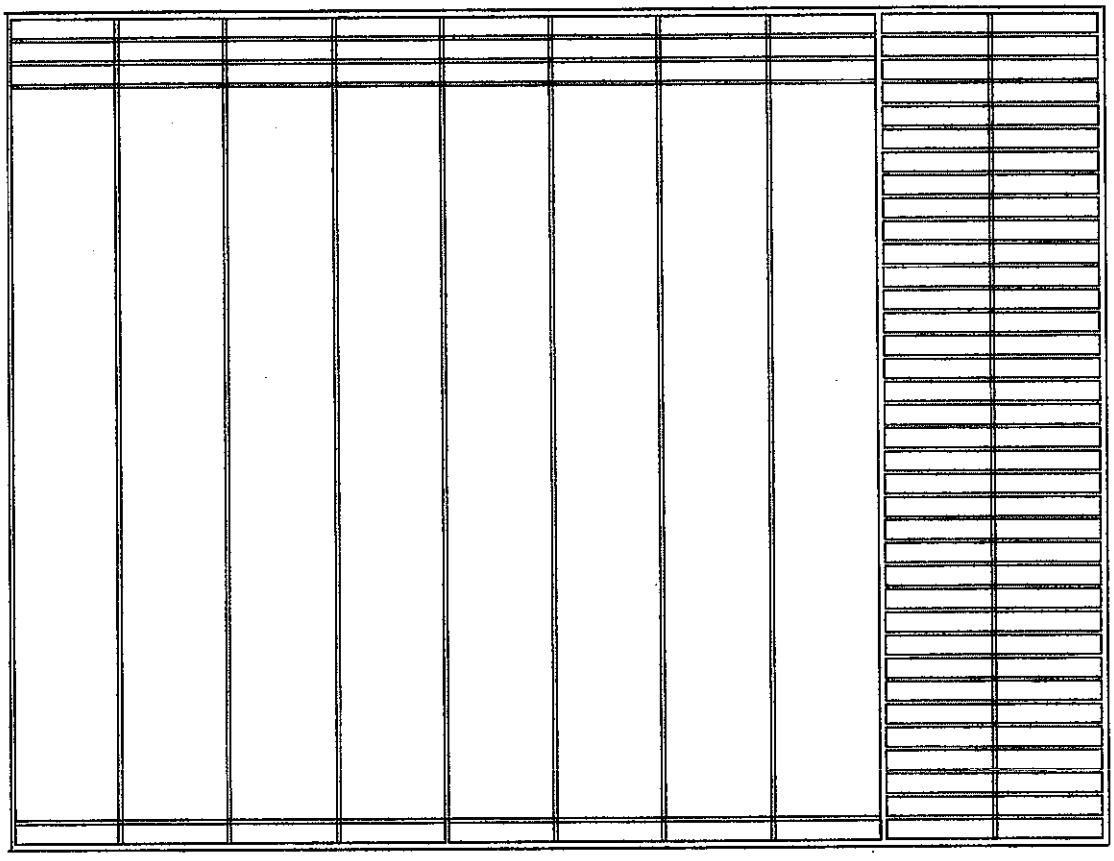

図３．１－３　スクリーン・レイアウト用グリッド

垂直なコラム・ラインと水平線によって決まる、いくつかの小さな長方形のエリア（図３．１－３参照）を組み合わせることによって、視覚的によいリズム感を出すことができる。このリズム感がそれぞれのイラストのスペースの実際的な要求を満たす。図それ自体はグリッドに沿った細い長方形の線で囲うか、ページやスクリーンの一部を表す淡いグレーのバックを付ける。

紙のページやスライドやスクリーンにおけるレイアウト・グリッドがもたらす視覚的な関係は、プロポーション、フォーマット、テキストタイプ、イラストレーションが個々ばらばらに関わっているわけではない。このことは、アイコンやシンボルにおいてもいえることである（図３．１－４参照）。

通常、視覚言語のプログラマまたはグラフィック設計者は、テキストや図やフォーマットなどの異なる要求を満たすために、これらのファクタのバランスをうまく取る必要がある。これらに関しては、合理的な解決策がとられるべきではあるが、かといって個人的な好みを反映する余地がないわけではないし、同時に情報表示を改善することも可能である。

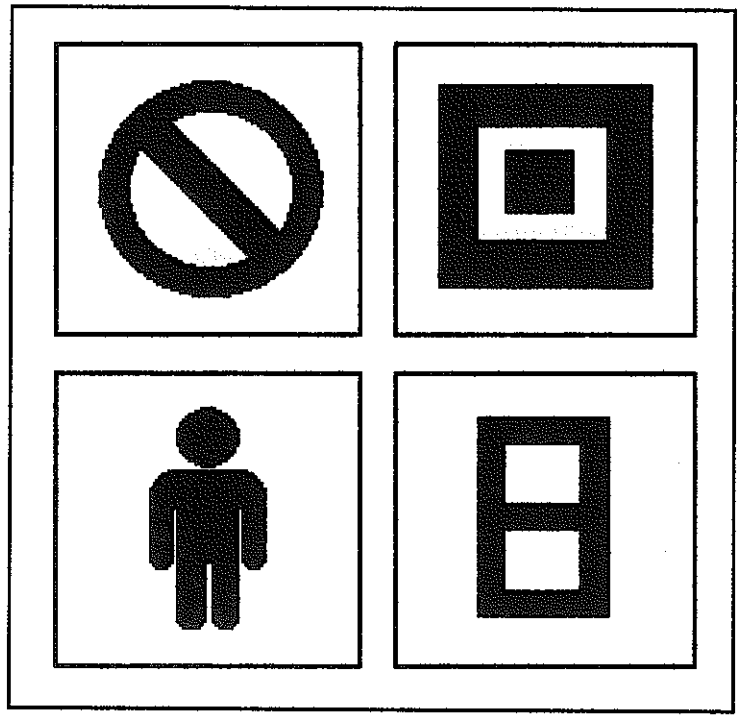

図３．１－４　グリッドベースの設計

３．１．２　空間的メタファア、ディスプレイ、ツールのデザイン

コンピュータ技術の非常に初期の頃より、電子出版とユーザー・インターフェイスには空間的なイメージが使われていた。当初はまだ、それらの空間的な特徴は必要以上に複雑ではなく、ユーザーにも明確に理解可能な範囲のものであった。仮想的な空間に表示されるすべてのテキストや、言葉でない記号は、単一の平面的シンボルとして解釈されていたといってよい。

近年、高い能力をもったワークステーションの技術によって、三次元空間上に動きのある三次元的なカラーの記号を表示することも可能になってきた。現在、そして将来にわたるユーザー・インタフエースとコンピュータ・グラフィックスの設計者にとって、複雑な外観をもったユーザーとディスプレイ間のインタラクションの設計は最大の考慮事項といってよいだろう。特に、オブジェクト、構造、プロセス、ナビゲーション、セレクション、操作などを効果的にコミュニケートするためには、空間的な表現が多用されるであろう。次に、効果的なビジュアル・コミュニケーションのための空間的特性の最適化に関連した事柄について示す。

1980年代初頭におけるXerox StarとApple Macintoshの登場によって、マルチ・ウィンドウ・デスクトップという空間的なメタファが、外観とインタラクションの基礎となることをコンピュータ・グラフィックスのプログラマは力説してきた。使用者は、平面的な背景とその上に展開された複数の矩形のウィンドウを見ている。ウィンドウは、全面にタイル上に敷き詰められている場合もあれば、さまざまな方法によってオーバーラップしているときもある。オブジェクトやプロセス、構造、データなどを表すアイコンや、その他の記号類はバックグラウンドやウィンドウ上に表示される。空間的に奥行きのあるウィンドウに加えて、いろいろなメニューやダイアログ・ボックスが、ウィンドウ内やあるいは全てのウィンドウの上に表示される。そして、これら全ての表示要素のさらに前面には、カー

ソルがビジュアル・フィールドを横切るように浮いている。全てのウィンドウ、全てのバックグラウンドの部分がグラフィックの要素であり、それらは三次元的な空間を表している。

ここでいう、空間の前面の層、中間層、そしてバックグラウンドなどは、近代絵画における伝統的な空間構成を思わせるものである。このマルチ・レイヤーの構成は、2.5次元などともいわれている漫画アニメーションのセル画を思わせる。この空間へのオブジェクト表示における、ある種の視覚効果は、見るものに対して空間的な構成として認識させるのに使われる。この効果の一例として次のようなテクニックがある。

影を付ける
エッジを斜めにする
ハイライトとローライト
綿みと伸張

たとえば、通常影付けは、表示の右下に向かって付けられ、ウィンドウ・レイヤーやプルダウン・メニューやポップアップ・メニュー、そしてダイアログ・ボックスなどに使われる。

また、あるインターフェイスでは、アイコン、ボタン、スイッチ、メニューの要素ないし、メニュー全体を囲う矩形、ダイアログ・ボックス、ウィンドウに対して、見るものに向かって飛び出してくるような斜めの辺を付けたものもある。時には、それらの辺は三次元的な形状を強調するために何段階かの濃さのグレーによって色付けされたり、表示物の左上のコーナーに明るい面を持ってくることもある。

さらに、ウィンドウ全体やスクリーンの一部のエリアが、目立つようにハイライト表示される場合もあるし、それを強調するために、その他のウィンドウなどが空間内でより後ろにあることを表すようにローライトされる場合もある。表示要素はまた、そのサイズや外見を変化させることもある。たとえばアイコンがウィンドウへと拡大することなどである。この変化はしばしば二次元的に広がるように表現されるが、これは表示要素がオーバーラップしているような錯覚を与えるものである。

これらは、設計者がチャート、マップ、ダイヤグラムのような、情報オリエンテッドなグラフィックスを強化するのに用いたのと同様なテクニックである。これらの空間的なクオリティは、グラフィック設計の観点から言えば、きわめて価値を持つものであり、次のようなことを達成する。

スクリーン上のさまざまな要素を明快に区別する
見るものに対して特定のオブジェクトの分類を認識させる
ユーザー・インターフェイスの設計、スタイルに魅力と訴求力を加える
コーポレート設計、プロダクト設計に規則をつける

伝統的なデスクトップ以外にも、あるユーザー・インターフェイスでは「コントロール・パネル」

のイメージも使用される。これは、スイッチやノブやその他のコントロール・デバイスなどを配置した平面的なパネルでスクリーン上に複数表示されることもある。デスクトップのバリエーションとして「ジャイアント・デスクトップ」などというものもある。これは、使用者が見ているものはビューポートを通した大きなバックグラウンドの一部であり、その他のエリアを見るためにはスクロール・デバイスなどを使用するものである。

また、ほかのデスクトップのバリエーションとしては、マルチ・デスクトップと呼ばれるものもあり、ここでは使用者はズームやカットやポップなどのビジュアル・エフェクトのテクニックによって、デスクトップ間を移動する。

空間的な役割を補助するためにサウンドを使用したアプローチではMIT Architecture Machine Groupの1970年の空間的データベース管理システム（Spetial Database - Management System）があり、これはバックグラウンド・プレーンが使用者に向かって開かれるときには“whoosh”、また下のレイヤーに降りるときにはホッピング・サウンドとして“pop”などというサウンドを付けたものである。近年のハイパーテキスト製品は、通常、スクロール、パンニングできるようなプレーンないしカードといった概念を用いている。

近年、プログラマはヒューマン・コンピュータ・コミュニケーションに空間的なメタファを利用する経験をもちはじめてきた。その一つとして建築のメタファがある。たとえばThe Learning Company社は1980年代の初期にRocky's Bootsという子供用のゲームに入口と出口をもつ、いくつかの部屋というメタファを使用者に与えるものを制作した。スクリーン・ディスプレイに、リンクされた空間を表現するために、私たちが経験的によく知っている建物の構造を用いている。

三次元的な現実感を操作できる、より大きな能力をもつワークステーションでは、その他のアプローチも可能である。Adobe社のDisplay Post Scriptというピクチャー定義言語を採用したSunやNeXTなどの複雑なイメージ表示をサポートしたディスプレイの登場によって、ユーザー・インターフェイスの空間的なコミュニケーションの基礎として、視覚的により洗練された建物や都市環境に表示メタファを用いることも可能となってきた。ここで必要とされるものは、親しみやすいシンボル、なじみのある空間構成、そしてシンボルとのインタラクションに必要な作法である。ここ数年のビデオ・ゲームには空間使用法のバリエーションとして、部屋、ビルディング、景観などを動作フィールドとして採用している。

将来における空間的表現方法の方向性は、エンターテイメントと現在のユーザー・インターフェイスの分野ですでに実現化されつつあるといってよい。この２つの分野では、宇宙の表現や三次元物体の表現が重要な要素になっている。

商用のケーブル・テレビやテレビ放送、映画産業などにとって、宇宙を表現するということは常に魅力的である。映画Star Warsのタイトル・バックでは毎回同様なものが引き続き使われている。最近では、夜のニュース番組や天気予報、映画のタイトル、局自体のアイディフィケーションに、宇宙空間を表現した映像上に写真や文字やその他の要素を回転させるなどというのはよく使われるテクニックである。ここ数年、NASAではヘッド・マウンテッド・ディスプレイによる完全な三次元空間の実験が行われている。最近では仮想三次元空間を体験できる商用の製品版も売り出されている。

現在のところ、動的な物体や、光の当たった表面やそのものの影などを表現できるような製品は非常に高価で、莫大な開発費、開発時間、人材そして装置が必要である。しかしながら、カリフォルニ

ア・サンタバーバラのWavefront社のような洗練されたアニメーション・ソフトウェアの制作会社は、使えるハードウェアやユーザー・グループを拡大しつつあり、より強力な三次元ディスプレイやイメージ・レンダリングなどの能力を実現する、コンピュータ・グラフィックス産業が育成されつつあるといってよいだろう。最終的には、これらの能力はユーザー・インターフェイスを構成する部品の表現において広く一般的に使われるようになるだろう。高価なワークステーションでなくとも、三次元的なユーザー・インターフェイスを構成するオブジェクトを表示することは可能である。たとえば、1980年代後半のCommodole Amiga用の音楽編集用のパッケージ・ソフトの楽譜の表示の場面には、左右には本物のような柱、そして上部には舞台用の梁などが表現されている。

ユーザー・インターフェイスや情報グラフィックスにおいては、データ・オブジェクトやプロセス、それらのステータス、そしてさまざまな種類の構造を表現する必要がある。加えて、設計者はユーザーがいろいろなやり方でそれらのオブジェクトをナビゲートし、選択し、操作する方法を決定しなければならない。設計者にとって、空間の認知方法を確立したGibsonの視覚的な役割についてのリストは参考になるかもしれない。以下の遠近感に関する経験則は、Edward Hallの「かくれた次元」より整理した。

位置（Position）

テクスチャ：後方へ傾いた面のテクスチャは遠くへ行くにしたがって密度が増していく
サイズ　　：物体は後方へ行くほど大きさが小さくなる
線の遠近感：平行線は消失点に向かって収束する

視差（Parallax）

双眼鏡の遠近感：両眼に映るイメージは、物体の位置が多少ずれている
動きの遠近感　：形状の変化しない動く物体のスピードは、遠くにあるほど遅く見える

観察者の位置や動きによらない役割

空気遠近法　　：遠ざかるほど霞んでいき、青みが増える
不鮮明さ　　：焦点のあった面より近すぎても遠すぎても物体はぼやけて見える
垂直方向の位置　　：低い位置のものは近く、高い位置のものは遠くに見える
テクスチャや間隔の変化：間突な変化は奥へ移動したように見える
イメージの重なり　　：遠くから見ると、近づいた物体同士は重なって見える
動きの比率の変化　　：近くの物体は遠くのものより大きく動く
外形の完全性と継続性　：より近くの物体は後ろにあるその他のものに重なる
明暗の変化　　：唐突な変化はエッジが際だち、緩やかな変化は完全に見える

現在すでに、これらのうちのいくつかは、ユーザー・インターフェイス、および情報グラフィクスの中で、空間をイメージさせるのに用いられている。ユーザー・インターフェイスや動的な情報グラフィクスは、ますます視覚的に複雑になってきているので、設計者はこれらの空間的奥行きの役割をさらに利用していくようになるだろう。また、最終的にはシステマティックなやり方で上記の空間的

な表示の性質を決めていく必要もあるだろう。

一つの空間上で、オブジェクトを統合できるような空間メタファがもっと複雑になり、空間的イメージがより洗練されてくれば、ユーザー・インターフェイスの個々の要素によって、複雑な内部空間的構造を表現できるようになる。たとえば、ウィンドウやメニュー、ダイアログ・ボックス、アイコン、カーソルなどのようなユーザー・インターフェイスの典型的な構成要素は、意味のある三次元的なクオリティを獲得することができ、以下のような可能性が期待できる。

期待1：タイトル・バーやスクロール・バーがソリッドな形で出っ張ったウィンドウ。

期待2：現在見えている部分を表すソリッドなチューブを持つ半透明の曲がったコラムのように見えるスクロール・バー。

期待3：すべての面にそれぞれ本当に意味のあるものを配置できるような、正面が長方形に見える直方体でできたウィンドウ。

期待4：三次元的なブロック上のアイコンで、内部に動くパーツをもち、メタリック、ラフ、ウォームといった属性やアイコンの特徴を表すような表面をもつもの。

期待5：大きな三次元的な人間の顔を持つカーソルで、それが指す指先は特定のスクリーンの構成要素にユーザの注意を向ける。またその人間の顔の表情は内容の重要性を暗示する。

この点に関しては、ユーザー・インターフェイスの設計者はグラフィック設計、建築設計、工業設計、プロダクト設計の専門家の歴史や現在の仕事を調べてみるとよい。たとえば現代の工業設計の世界でも、すべてのオブジェクトは互いに密接に関係し合っていて、表現の選択は限られたものであるとするミニマリスト的なアプローチと、包括的でエキゾチックでさまざまな表現を許す、より華麗なアプローチとの間での議論が存在している。こういった観点からみると、設計とはエンジニアリングの分野より、美的なスタイリングの世界に属しているともいえる。

また、設計が世界的に通用するような製品の市場性に寄与しているといってもよい。ユーザー・インターフェイス、情報グラフィクスの世界も、やはり工芸品のように設計を考慮すべき対象である。そして、コーポレート設計や立体的な設計を行うプロダクト設計の標準も、これらに少なからぬ影響を与えるだろう。

ユーザー・インターフェイスおよび情報グラフィクスは、どんどん空間的な性格を帯びてきている。たとえば、仮想現実感（Virtual Reality）、人工現実感（Artificial Reality）といった分野では、空間に配置されるシンボルは、文化的な特色を備えており、これはエルゴノミクスのような直接的な問題よりはるかに複雑である。ユーザー・インターフェイスや情報グラフィクスの設計者が、ユーザーとコンピュータ間の空間についての指針として、個人間の空間の科学であるプロクセミクス（Proxemics／Edward Hanを先駆とする）の規律を考慮することはきわめて合理的なことである。

テクノロジーの進展に加えて、増加するデータ量と機能を表現しなければならない必要性、またマーケット的な興味などに刺激されて、ユーザー・インターフェイス、情報グラフィクスは、より空間的な特徴を帯びてきている。コミュニケーションの効果を上げるには、ユーザー・インターフェイス、情報グラフィクスの設計者は、イメージを構成するすべての要素に対して、非常に広範な空間的性質の階層構造を調整、統合、最適化する必要があるだろう。

幾何学的な問題として、そして文化的な人工物として、複雑な空間構造を扱う他の分野の専門家の考慮事項と業績を調べることによって、この課題を研究していく必要がある。ユーザー・インターフェイスと情報グラフィクス設計によって変化する範囲と程度は、三次元的なコミュニケーション上の形態を作るという技術を試行する重要な課題を与えるものである。

３・２　マルチ・コンピュータ環境構築のために採用した画面制御技術

本項では、計画する試作システムの構成を機能分散型マルチ・コンピュータとして考え、個々のコンピュータ固有の対話機能設計に必要な事項をとりまとめる。それらは、次にあげるウィンドウ・システムの機能的な能力と使い勝手上の特徴について触れる。

Apple Macintosh、OpenLook、Microsoft Windows、OSF/Motif Window Management Toolkitである。

ファイル・システムやCPUサイクルのかわりに、ウィンドウ・システムはスクリーン・スペースや人力デバイスといったリソースを管理している。グラフィック・ユーザー・インターフェイスにおいてウィンドウ・システムは、キーボード指向のコマンド・ランゲージの抽象的な、ときには混乱した文法や言葉からエンド・ユーザーを守るために、オペレーティング・システムのフロント・エンドの役割をする。

３．２．１　ウィンドウ・システム

ウィンドウ・システムは、グラフィック・ユーザー・インターフェイスの歴史上ユニークな位置を占めるものである。ここでは簡単にそれぞれのシステムのユニークな特徴を見る。

（１）Macintosh

Apple社のMacintoshは、1984年に高解像度のビットマップ・グラフィック・ディスプレイとダイレクト・マニピュレーション（直接操作）というインタラクション・スタイルを持ち、最初のマス・マーケットへ向けてのコンピュータとして発表された。そのウィンドウ・システムは、内蔵のオペレーティング・システムとMacintosh ROM内のユーザー・インターフェイス・ツールキット・ルーチンの上に構築されている。

初めに取り上げたMacintoshは、高いレベルの応答性と非常にシンプルなユーザー・モデルを持ったシングル・タスクのシステムである。サード・パーティーのソフトウェア開発者の間で、その標準的なヒューマン・インターフェイスの構成要素が広く受け入れられることにAppleは成功した。その結果、親しみやすいMacintoshのアプリケーションに関する知識は、次の新しいアプリケーションの学習という仕事を容易にしてくれるものである。Macintoshは、グラフィック・ユーザー・インターフェイスをもって広く市場に受け入れられ、そして商業的にも大成功を収めた初めてのコンピュータ・システムであるといってよい。

特に、それまでコンピューティングということに対して、伝統的に否定的だった非技術分野の市場における人気は、製品の一貫性とユーザー支援のヒューマン・インターフェイスというAppleの努力によるところが大きい。時代的な先行性と市場への浸透力によって、Macintoshはグラフィック・ユーザー・インターフェイスのインタラクションの標準を確立したといってよいだろう。ユーザーのアクションに対する応答性のよさがインタラクションの質を決定する。また、そういったインタラクショ

ンは、ウィンドウ・システムが特定のハードウェアとソフトウェア環境にタイトにインテグレートされたときに可能となるものである。

（2）OpenLook

OpenLookグラフィック・ユーザー・インターフェイスは、UNIX System V4の標準的オペレーティング環境として、Sun MicrosystemsとAT&Tによって共同開発されたものである。このグラフィック・ユーザー・インターフェイスは、イメージング・モデル、およびネットワーク・コミュニケーション・サービスを提供する、基本ウィンドウ・システムの最上層のレイヤーにあたる。さまざまなバージョンのOpenLookが、XWindow SystemおよびNeWS（Sun Network extensible Window System）の最上層に移植されてきた。

OpenLookは、開発者のための模範的な機能仕様とスタイル・ガイドによってガイダンスが規定されている。OpenLookの設計者は、斬新な外観や特徴的な動作をさせることによって、法的な訴訟を避けることを目標にしたようである。この結果、採用された多くの規約は産業標準からは大きく外れたものになっている。

OpenLookは、際だったアプローチをとっている。提供される多くの状況依存型のモード仕様のオペレーションをみると、機能性を最大限に目指していることがわかる。UNIXの世界では、あまり経験のないユーザーにとっても比較的受け入れられてはいるが、OpenLookの拡張された機能性自体は、NextStepやMacintoshのヒューマン・インターフェイスには見られない、複雑な側面をもっているといってよいだろう。

（3）OSF／Motif

OSF／Motifは、DECとHewlett Packardによって、OSF（Open Software Foundation）のために開発されたウィンドウ・マネージャおよびユーザー・インターフェイス・ツールキットである。Motifは、標準的なUNIXのOSFバージョンにリンクしており、OpenLookに代わるものである。

Motifのウィンドウ・マネージャは、OpenLookと同様、ネットワーク・オリエンテッドなXWindowシステムの最上層のレイヤーに位置するものである。外観的には、最終的なシステムの機能的特性によって個々に変更することができ、ベンダーは、自分達のウィジェット（Widget：インターフェイスの構成要素）を使って機能的なシェルにカスタマイズするように奨励されている。

Motifウィンドウ・マネージャの標準的な外観、動作は、OS／2のPresentation Managerに基づいたものである。OS／2やMicroSoft Windowsになじんだ PCのユーザーが、ワークステーション・レベル・マシン上のMotifに慣れるのは、やや難しいかも知れない。標準的なMotifのウィジェットは、疑似3D的な外観をしている。

OSF用Motifは、ハイ・エンドのネットワーク・ベースのコンピュータ環境のためのグラフィック・ユーザー・インターフェイスであるが、その外観と動作はMicrosoft Windowsとのグラフィック・ユーザー・インターフェイスと一貫性を持つものである。これらは、IBMとそのコンパチブルなプラットフォームにおける実質的な業界標準なので、これらのオペレーティング環境が、PCベース・システムのグラフィック・ユーザー・インターフェイスを目指す動きの中では優位を占めるものと見られている。OSFはWindowsの知識がMotifに容易に移行でき、PCユーザーがワークステーション・プラットフ

ォームにアップグレードするときに、それがウィンドウ・システムとして選択されると予測している。

Motifは、ネットワーク・トランスペアレントなXWindowシステムの上層へ移植できることから、ワークステーション環境での標準になる可能性がある。Motifは、PCとワークステーション環境の間の橋渡しとなるウィンドウ・システムである。Motifは、異なるシステム間の移行が容易なので、高性能ワークステーションとPCネットワークを統合している使用している会社にとっては、非常に魅力的なものといえる。

（4）Microsoft Windows

Microsoft Windowsは、PCコンパチブル・システムのMS-DOSによるキャラクタ・ベース環境に代わるマルチ・タスクのグラフィック指向のシステムとして1985年に作られた。Windowsのビットマップ・ディスプレイとマウス操作のメニューが、PC上にグラフィック指向のソフトウェアの扉をはじめて開いた。

Windowsは、DOS環境上に構築されており、初期の頃は、640Kのアドレス空間、低解像度のディスプレイなどのさまざまなデザイン上の制約があった。しかし、近年の改良により、特に80386ベースのマシンでは、Windowsの応答性とグラフィック・クオリティはかなり向上してきた。

Microsoft Windowsは、ビジネスから技術的な環境において広く行き渡ったMS-DOSユーザーの非常に異なる市場を満足させなければならない。キャラクタ・ベースのDOSインターフェイスの資産を反映して、これらの製品はキーボード・ベースの広範な制御をサポートしている。メニューやダイアログ・ボックスからの選択に使われるキーストローク・コンビネーションなどに、このことがはっきりと現われている。

3．2．2　ウィンドウ・システムの概観

ウィンドウ・システムは、ディスプレイ・スクリーンを複数の機能的なエリアに分割したものである。このエリアは、複数のアプリケーションを管理、制御したり、グラフィック環境上の複数のデータを操作するためのものである。ドキュメントやアプリケーションを表現するウィンドウは、適切に定められた視覚的、機能的環境を提供するものだが、この環境は、マウスやキーボードなどの入力デバイスや、物理的に限定されたディスプレイ・スペースを時間管理して、多重な処理を可能としたものである。ウィンドウに関するプログラム・コード（**ソフトウェア・アーキテクチャ、ウィンドウ管理の方法、基本となるウィンドウ・システムのイメージング・モデルなど**）は、ディスプレイの品質やユーザーによって体験されるインタラクションのレベルに大きく関係する。この項ではこれらの要素について見ていくことにする。

（1）ソフトウェア・アーキテクチャ

ウィンドウ・システムを実現するコードの置き場所と構成方法は、応答性、デバイスの独立性、そして最終的に要求されるシステムのリソースなどに影響を与える。たとえば、カーネル・ベース・アーキテクチャでは、高いレベルのインタラクションが可能となるが、一つの機械のアーキテクチャやリソースにしか対応できない。クライアント・サーバー型のアーキテクチャでは、異なる機械間のネットワーク全体で、一つのウィンドウ・システムの実体を共有することが可能になるが、反応時間は

おそらくネットワークのコミュニケーション・バンド幅によってかなり制限を受けてしまうだろう。

（2）カーネル・ベース・アーキテクチャ

カーネル・ベース・アーキテクチャでは、ウィンドウ・サービスはオペレーティング・システム自体の一部か、あるいはオペレーティング・システムのRAMまたはROMベースのライブラリに常駐している標準のアド・オン・モジュールによって提供されることになる。カーネル・ベースのウィンドウ・コードやオペレーティング・システムは、同一の物理的メモリー・スペースを共有し、本質的に同じ方法でアクセスされる。

（3）クライアント・サーバー・アーキテクチャ

クライアント・サーバー・アーキテクチャは、複数のプロセス、マシン間でウィンドウ・システムのリソースを共有できるようになっている。サーバーとは、接続されているマシンのネットワークの中で、特定の機能を持つソフトウェアが走っているコンピュータである。またクライアントとは、サーバーによって提供される機能を要求したり利用したりできる、同じネットワーク上のソフトウェアの一部である。

最良のクライアント・サーバー方式でも、多大なコミュニケーション・オーバーヘッドによって、カーネル・ベース・ウィンドウ・システムに比べたら明らかなパフォーマンスの低下が見られる。また、カーネル・ベース・システムは、より高いパフォーマンスが達成できるが、それらはデバイスに依存しており、それぞれのマシンで同じコードを実行するという冗長性がある。

ウィンドウ管理の機能としては、ウィンドウが移動されたり、サイズを変えられたり、またその重なりを変えられたりといった、ウィンドウ間の空間的な関係をシステムが維持するようにすることである。ウィンドウを自動的にアレンジするような機能を備えたウィンドウ・コントロール・メニューには、いくつかの選択肢がある。特に重要なのは、タイリング、オーバーラップ、カスケーディンク・ウィンドウなどである。タイリング方式とオーバラップ方式のそれぞれのメリットに関するこれまでのさまざまな論争は、オーバーラップ・ウィンドウ管理の方に軍配があがったようだが、タイリング・ウィンドウも、大きな高解像度のディスプレイには依然有効である。

（4）タイリング・ウィンドウ

タイリング・ウィンドウは、ウィンドウ・システムによって、有効ディスプレイ空間（これはディスプレイ・スクリーン全体のときもあるし、ウィンドウの内容エリア全体のときもある）を完全に敷き詰めるように自動的に調整され、ウィンドウは、お互いにオーバーラップしないようになっている。どれかのウィンドウの大きさが変わったときには、ほかのウィンドウは、それにつれて逆方向にサイズ変更しなければならない。

（5）オーバーラップ・ウィンドウ

オーバーラップ・ウィンドウは、見るものからの距離を表す奥行き方向の値が組み込まれている。それぞれのピクセルに関して、表示部分に重なっている最も手前にあるウィンドウの内容だけが表示される。このようにして、一番手前にあるウィンドウは、同じディスプレイ空間を占めるほかのウィ

ンドウの内容を覆い隠し、本当に重なり合っているような錯覚を起こさせる。最終的なウィンドウの重なり具合は、ユーザーの机上の紙の束のようなもので、ユーザーは生まれつき持っている空間管理能力といった有利さを活用することができる。

（6）カスケード・ウィンドウ

カスケード・ウィンドウは、オーバーラップ・ウィンドウの特殊な場合である。どのウィンドウも、完全に覆い隠されることがないように、規則的な順序で自動的にウィンドウを配置するものである。それぞれの連続したウィンドウの原点（左上の角）は、ディスプレイ空間を大事に使うために、垂直、水平方向に少しだけずらしてあり、これによって簡単にどのウィンドウでも重なりの一番手前に持ってくることができる。

イメージング・モデルは、ウィンドウ・マネージャが使用するディスプレイ・デバイスの物理的なバッファ上で、グラフィック要素の数学的表現からピクセル・イメージに変換する一連の機能群として具体化されている。２つの主だったイメージング・モデル・クラスは、アウトプット・デバイスの解像度に直接対応するようなラスター・イメージ（ビットマップ）か、あるいは抽象的な手続きによるイメージ記述からアウトプット・デバイスの物理的な座標にマッピングする、PostScriptのようなベクター・ベースの描画言語に基づいている。

イメージは、手続き的に定義されていて、ウィンドウ・システムのソフトウェアによって解釈しなければならないので、Display PostScriptのようなベクター・ベース言語の描画時間は、含まれるイメージの複雑さによって決まってくる。これに比較すると、ラスター・ベースモデルでの描画時間はイメージの複雑さには影響を受けない。

座標システム、カラー化の能力、イメージング・モデルなどによって決まるイメージの変換方式などは、ウィンドウ・システムのパフォーマンスや、それがサポートするディスプレイの品質に非常に大きく影響を与える。

デバイス依存型のイメージング・モデルは、最高のパフォーマンスを発揮するが、プラットフォームが代わったときには、印刷の品質が制約されたり、表示にあたってプログラムを作り直すことが必要になる場合がある。洗練されたデバイス独立なイメージング・モデルは、どのようなアウトプット・デバイスの解像度にもマッチするようにスケールを調整することができるが、これに要する変換は、描画スピードを著しく低下させる。

MITとDECによって共同開発されたXWindowSystemは、急速にワークステーション・コンピューティングや、ネットワーク・コンピューティング環境における事実的な標準になってきた。X11は、クライアント・プログラムとウィンドウ・サーバー間のコミュニケーションを可能にする、高いスループットとネットワーク・トランスペアレントなメカニズムを提供している。

クライアントとサーバーのどちらか、または両方のローカル・マシン上に存在するXWindowSystemは、ネットワーク・コミュニケーションや、グラフィック・ユーザー・インターフェイス・レイヤーの外観や動作とは完全に独立した、さまざまな表示に関係するイメージングの土台となるものである。MotifやOpenLookは、X11の外観や動作のバリエーションを作れる能力を利用して、X11の基本機能の上に構築されている。

ユーザーがそこで体験するウィンドウ・システムの外観や動作は、少数の標準的な構成要素から成

り立っている。それぞれのシステムは、本質的には同じようなヒューマン・インターフェイスのオブジェクトを使用しているが、各ベンダー間での、これらの構成要素の呼び方は非常にバラつきがある

（7）ウィンドウ

ウィンドウ・マネージャから見ると、ウィンドウは、ディスプレイ・スクリーン上での固定的なエリアで個々に移動、サイズ変更および描画が可能なビジュアル・ディスプレイである。ほとんどの構成要素は、実際にはシステムによってウィンドウとして扱われ、管理されているわけだが、私達はユーザーの視点からウィンドウを捉えたいと思うであろう。したがって、私達の採用したウィンドウの定義では、ユーザーがサイズ変更や、スクロール、編集などのようなテクニックを使って、その内容の見え方を変更できるディスプレイ・オブジェクトだけを指すことにした。

（8）メニュー

メニューは、コマンドを思い出して、それをタイピングする代わりに、コマンドを表示してそれを指し示すことによってユーザーに指定させる方法を与えるものである。メニュー・システムは、筋肉的な労力を多少減らすとともに、人間の記憶量の限界によって生じる問題を非常に減少させてくれる。特に、通常使われるアプリケーションの数や複雑さが、ユーザーが覚えなければならない個々のコマンド・セットを制限しているようなときには、メニュー・システムの効用は大きい。

（9）コントロール

マウスやキーボードによって直接操作することのできる視覚的なウィンドウ構成要素は、すべてコントロールである。それぞれのウィンドウ・システムは、標準的なコントロールのセットを定義しており、各製品にまたがる一貫したインタラクションの規約を規定したもので、各アプリケーションに組み込むことができる。

（10）ダイアログ・ボックス

ダイアログ・ボックスは、ユーザーが選択することのできるオプション群を表示して、視覚的、機能的な状況を作るものである。限られた空間内において生じるユーザーとシステムの間のインタラクティブな情報交換は、すべてダイアログと考えてよい。

ダイアログ・ボックスの種類としては、コントロール・パネル、質問ボックス、メッセージ・ボックスという、はっきりとした3つのクラスが考えられるが、これらは互いにかなりオーバーラップしてもいる。どのダイアログ・ボックスも、システムの状態によってその効果が決まり、引き続く操作が限定されるといった、明確に規定された有効範囲というものが決まっている。

（11）モードレス・ダイアログ

モードレス・ダイアログ・ボックスは、有効な範囲が決まっており、ユーザーの次の操作には制約を与えない。モードレス・ダイアログは、サイズや位置の変更などの基本的なウィンドウの機能が組み込まれている場合もある。ユーザーは、必要であれば、これに答えることなく仕事を続けることもできるし、また、これに答えた後でも、モードレス・ダイアログをディスプレイに表示したままに

しておくことができる。

（１２）モーダル・ダイアログ

モーダル・ダイアログ・ボックスでは、ユーザーはこれに答える前にはその他のアクションを起こすことができない。アプリケーションのモーダル・ダイアログでは、そのダイアログが満足される前には、ユーザーはすべてのアプリケーション機能を実行することができない。これに対して、システムのモーダル・ダイアログでは、ユーザーはシステム内のすべての操作を実行できなくなってしまう。

（１３）コントロール・パネル

コントロール・パネルは、ユーザーの明示的または非明示的な要求に対して表示され、現在のシステムの状況に応じた多くのパラメータ情報を提示する。このパラメータは、パネルが表示されている間は、ユーザーがインタラクティブに変更することができる。システムの状態の変更は、ユーザーが新しい設定を明示的に決定するまでは効果を発揮しない。

（１４）質問ボックス

質問ボックスは、ユーザーのアクションに対して表示されるが、ユーザーが明示的にこれを要求することはできない。質問ボックスは、はい、いいえ、で答えられるような簡単な質問情報をユーザーに与え、その中で必要な情報が得られるような状態を作り出す。コントロール・パネルと同じように、質問ボックスでは、ユーザーは質問されたアクションをキャンセルすることもできる。

（１５）メッセージ・ボックス

ユーザーに危険などを知らせることがメッセージ・ボックスの基本的な機能だが、ユーザーが直接それを要求する類のものではない。また、典型的には、システムがリカバーできないような危険な状態になった、あるいはそうなる可能性がありそうなときに表示されるものである。ユーザーの選べる選択肢は、通常は、はい、いいえ、で決まるような限られたものであるか、あるいは、システムの状態が逆行できないような場合には、単にメッセージの承認を得るだけである。

（１６）マウス／キーボード・インターフェイス

システムは基本的なインタラクション・デバイスとして、マウスやキーボードを用いる。それぞれのデバイスは、ある種のインタラクション・タスクにうまく適合したものである。マウスはメニュー操作や、ウィンドウのサイズ／位置変更といった空間的な操作を必要とする仕事をするためには大変便利なものである。キーボードは、文章入力やウィンドウの切り替えのような連続的な仕事により向いている。

３．２．３　ウィンドウ

本項で述べているウィンドウ・システムは、すべて標準的なウィンドウ構造を持っており、ユーザーは、この中にアプリケーション・プログラムやドキュメントを表示、制御することができる。これらのシステムでは、アプリケーションが必要に応じて変更することのできる、いくつかの標準的なウ

ィンドウのタイプを規定している。このセクションでは、比較のために、他ウィンドウ・システムの提供している基本的特徴を持った標準ウィンドウについて示す。

（1）アプリケーション・ウィンドウ

アプリケーション・ウィンドウは、プログラムが起動されたときに、最初にワークスペースに表示されるものである。アプリケーション・ウィンドウは、データ自体のこともあれば、実際に処理されるデータが表示された、いくつかの「子」ウィンドウ（いわゆる、独立したウィンドウまたは「親」ウィンドウによって作られる従属的なウィンドウ）の最上位の状況を表す場合もある。ユーザーが自由に変えることのできるデータを表示する子ウィンドウは、通常、ドキュメント・ウィンドウと呼ばれる。これは、それらが普通はテキストや絵のファイルのような、単一のデータ・オブジェクトを表示したものだからである。

（2）ドキュメント・ウィンドウ

ドキュメント・ウィンドウの構造は、そのアプリケーション・ウィンドウと似たものか、あるいは全く同じものである。ドキュメント・ウィンドウは、今作っている一つのオブジェクト・データを蓄えたり、表示したりする環境を作りだすものである。ほとんどのアプリケーションは、ユーザーが一度に複数のファイルで仕事ができるように、マルチ・ドキュメント・ウィンドウをサポートしている。親ウィンドウのアプリケーション・メニューは、現在、ユーザーの入力を受け付けているドキュメント・ウィンドウだけに有効である。

（3）ウィンドウ・アイコン

ウィンドウ・アイコンとは、ユーザーが入力する必要のない間、ウィンドウ・サイズを縮めてディスプレイ空間を節約するためのものである。ウィンドウ・アイコンは、アクティブなアプリケーションやドキュメントを視覚的に思い出させることによって、ユーザーに全体の状況を把握させるのに役に立つ。これらのアイコンはまた、視覚的オブジェクトとの直接的なインタラクションを通して、ただちに以前の状態に戻ることができる。

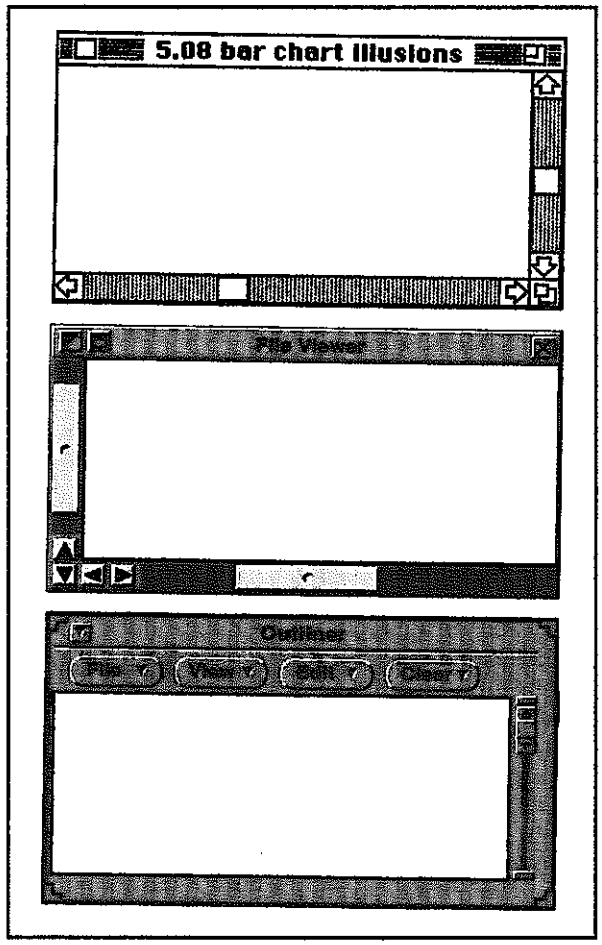

図３．２－１　ウィンドウの例１

グラフィック・ユーザー・インターフェイスにおける最近の傾向は、ウィンドウやウィンドウの構成要素の形状が、自動車のダッシュボードやオーディオ機器といった親しみ深いデバイスのような実際の世界に似せて三次元的に見えることである。三次元的に立ち上がったエリアは、周囲との差を容易に区別でき、掴めるような、あるいは押せるようなオブジェクトとして認識されるので、このスタイル上の扱いは機能的な有用な意味があるといってよい。

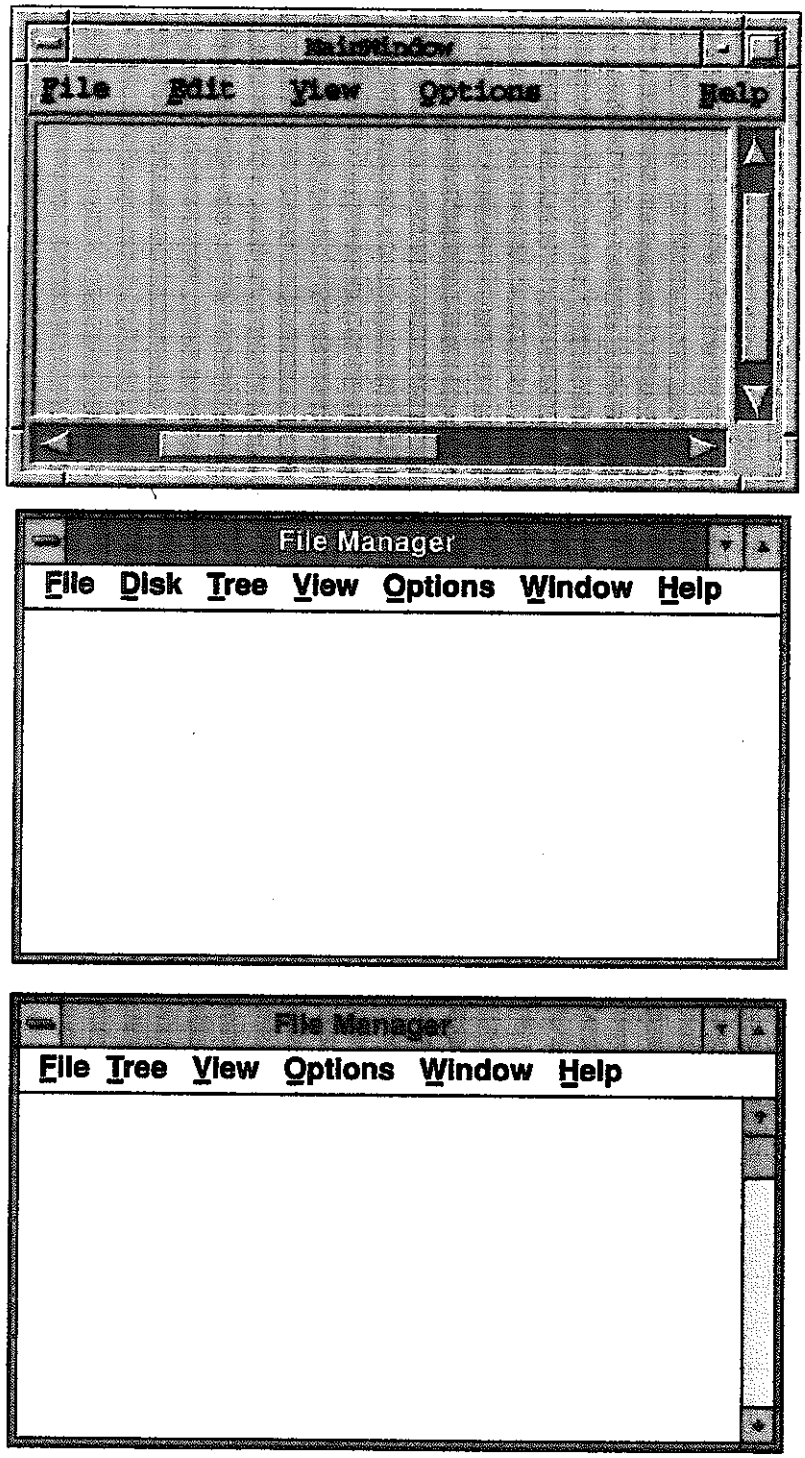

図３．２－２　ウィンドウの例２

アプリケーション・ウィンドウやドキュメント・ウィンドウの構造は、各システム間でかなり異なっている。ほとんどのウィンドウは、ウィンドウ・システムによってコントロールされた周辺の領域と、アプリケーションが自由に変更できるコンテント（内容）領域を持っている。

図３．２－１、図３．２－２のウィンドウ・システムの一覧は、６つのシステムにおける、フル装

備の基本ウィンドウの全体構造と主要構成要素の例である。

ウィンドウ・システムの技術は、ほぼ完成の域に達しており、ウィンドウ管理、操作のためのユーザーの標準的なコントロール類は、ほぼ固まってきている。（図３．２－３参照）

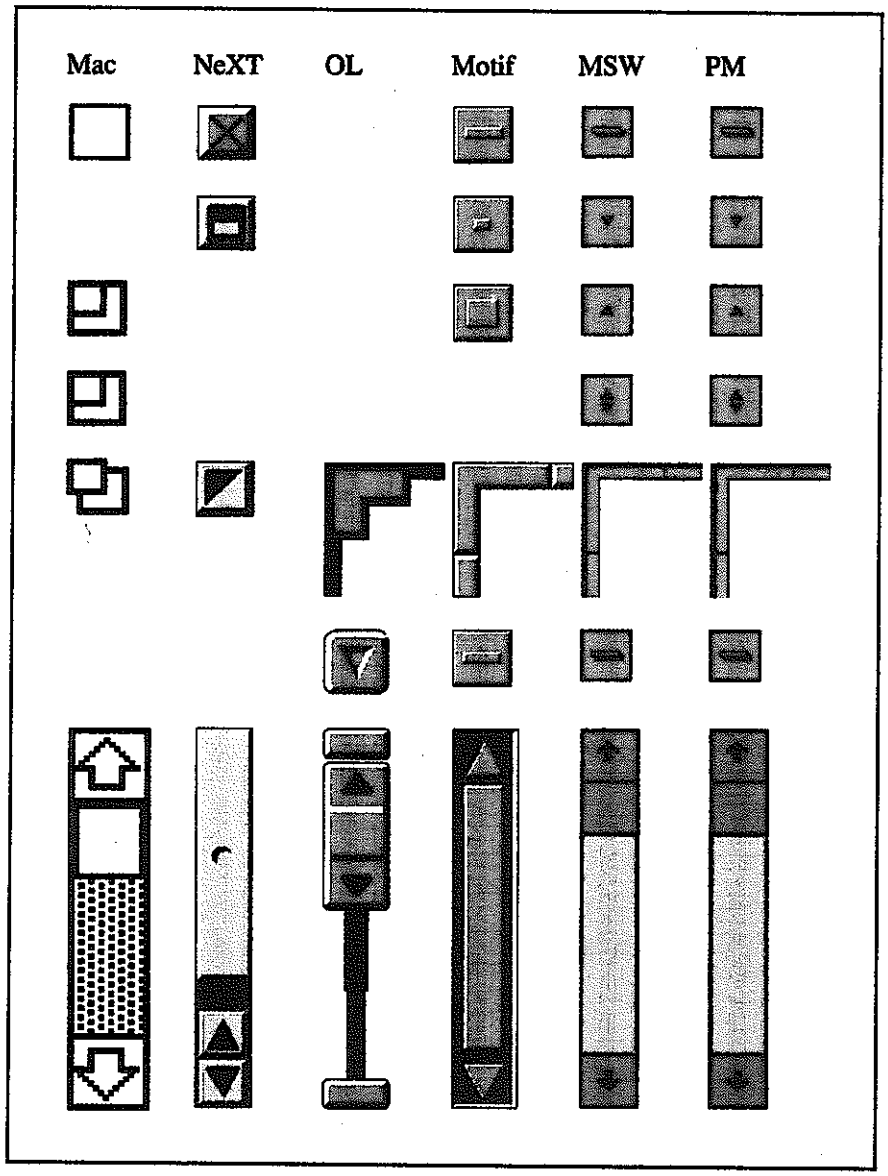

図３．２－３　ウィンドウ・コントロールの例

ウィンドウは、アプリケーションに関連するデータを表現するための手段である。データを変更するためには、適切な機能を選ぶためのメニューが必要になってくる。

３．２．４　メニュー

メニューは、名前やそのコマンド特有な文法を覚えることなく、ユーザーがコマンドを発行することのできる手法である。ここで比較してきたウィンドウ・システムは、選択／実行型のコマンド発行パラダイムを使っている。ユーザーは、はじめにオブジェクトや操作が適合するオブジェクト群を選

択し、その後にメニューから適切なコマンドを選ぶ。

このウィンドウ・システムは、それぞれ、それらの能力を活かすようなメニュー・インターフェイスを提供している。もっとも一般的なレベルでは、すべてのシステムはコマンドのリストを縦に配列したメニューを表示する。しかし、それ以上のメニューの外観と動作ということになると、各システムでかなり違ってくる。図３．２－４、図３．２－５に各システムの持っている標準的メニューの代表例を示す。

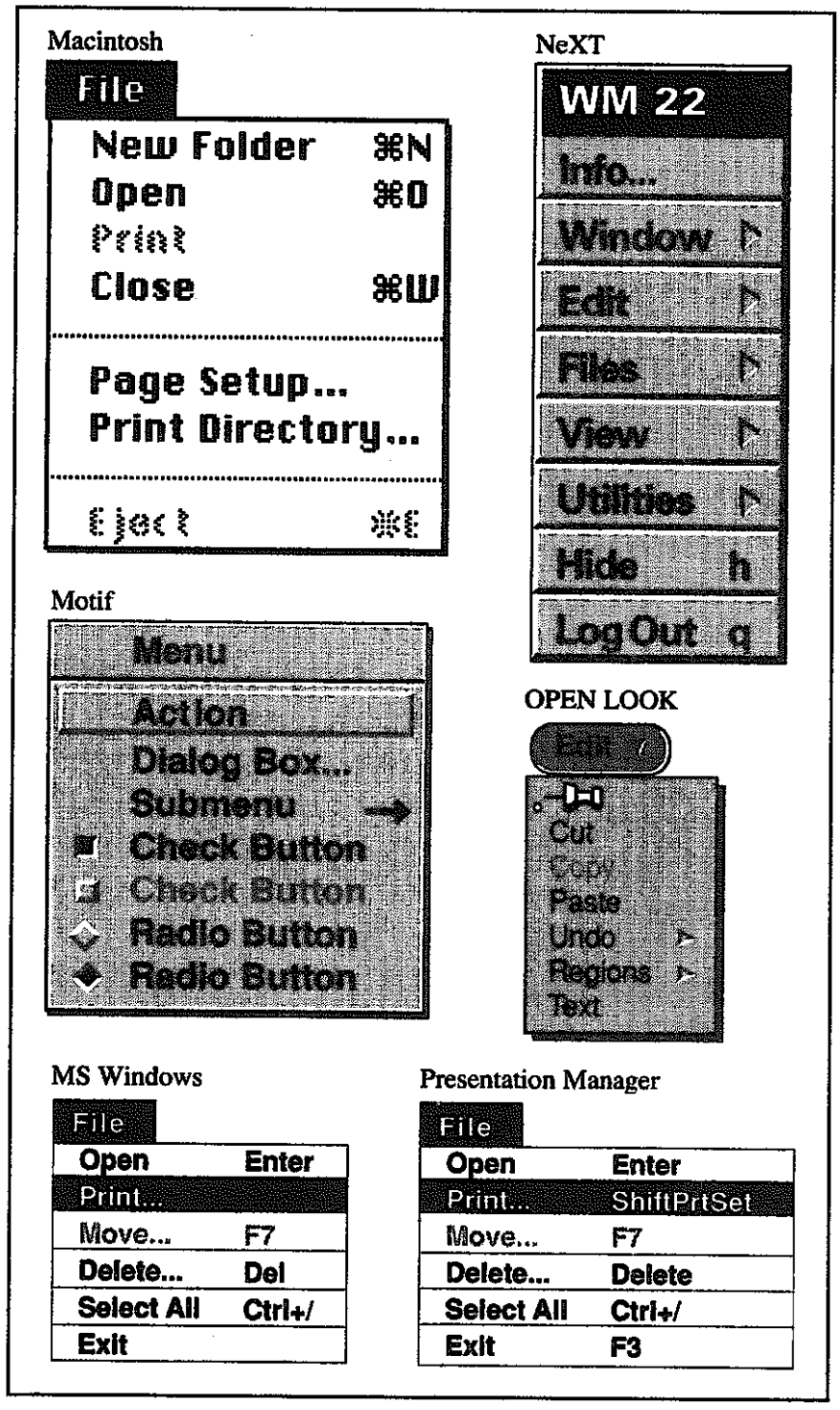

図３．２－４　メニュー例

Mac	NeXT	OL	Motif	MSW	PM
ファイル 編集	インフォ ウインドウ 編集	ファイル ビュー 編集 プロップス	システム	コントロール ファイル 編集 ビュー ウインドウ ヘルプ	システム ファイル 編集 ビュー ウインドウ ヘルプ

図３．２－５　メニュー表示技術の比較

ここでは、メニューの外観、動作およびアプリケーションにわたる標準化などの主要な違いについて検討する。このセクションでは、基本的なメニューの特性に基づく、いくつかのメニュー・クラスを定義する。まず、表現、表示期間（デュレーション）、ナビゲーションといった違いがある。

（１）表現

明示的でないポップ・アップメニューは、マウスのメニュー・ボタンをディスプレイの特定の領域で押すと開かれる。この領域はスクリーン全体のこともあれば、ウィンドウやウィンドウ境界のような定義された領域のこともあるが、メニューの存在をユーザーにあらかじめ知らせるようなフィードバックはない。明示的なメニューでは、メニューを受け付けるエリアは、特定のシンボルや標準的なコントロールにあった視覚的な表現で表されている。プルダウン・メニューは、その特殊な場合で、メニューはディスプレイやウィンドウの上部にあるメニュー・バーについている。

（２）表示期間

メニューの表現は、一定の位置にずっと表示され続けているような静的なものと、ユーザーが選択している間だけ表示される動的なものとがある。ステイ・アップ・メニューは、これらの動作の組み合わせである。ステイ・アップ・メニューは、使いやすい場所に置かれて、選択された後でもディスプレイに残っていて、ユーザーは表示されている選択肢から連続的に選んでいくことができる。

（３）ナビゲーション

メニューの階層構造は、マウスのメニュー・ボタンを使って移動できる。これには、押して（そのまま）／他のものを差し／離すという方法と、「クリック（押して離す）して／他のものを差し／またクリックする」という方式がある。Macintoshを除く、他の５つのウィンドウ・システムでは、両方の選択方法をサポートしている。

メニューという視覚言語の特性は、ウィンドウ・システムによって提供されるグラフィック・ユーザー・インターフェイスの概念モデルを強化することができる。メニュー項目の状態を反映して用いられる視覚言語には、

ア．複数の設定を持てる項目や、項目グループから現在選択されている状態、
イ．項目が選択可能かどうか、
ウ．結び付けられているサブ・メニューやダイアログ・ボックスがあるか、
エ．項目にアクセスするのにキーボード・ショートカットが使えるか、

などを表示すべきである。また、現在選択されているメニュー項目を特定するために、メニュー・システムを通じて、マウス・ポインタやキーボード・カーソルなどのユーザー・ナビゲーションに対して適切な視覚的フィードバックを返さなければならない。

それぞれのウィンドウ・システムは、選択できない項目を選択できるものよりも非強調表示している。サブ・メニューのある項目には、矢印や三角形が、そしてダイアログ・ボックスを持つものは省略符号が付けられている（図３．２－４参照）。

OpenLookだけは、メニューにキーボード・ショートカットを明示していないが、他のシステムは、「アクセラレータ・キー」または「ホット・キー」などと呼び、関係するメニュー項目の右端にその修飾子が付けられている。

メニューの動作という面では、ユーザーの操作方法と、ユーザーのインタラクションに対する反応の方法を限定している。個々のメニューのクラスによって、その表現、デコレーション、ナビゲーションの骨性などが規定されている。各システムは、個々のアプリケーションやドキュメントに対応したプルダウン・メニューをサポートしている。Macintoshではスクリーンの最上部にそれを設けている。また、それぞれのシステムは、明示的あるいは非明示的なポップアップ・メニューを持っている（図３．２－６参照）。

一項目システムごとに定義されている標準的なメニュー項目を図３．２－６に示す。ウィンドウ・システムのコマンドのメニューは、一般的にはシステム自体によって与えられる。他のメニューはアプリケーションによって定義され、表現、動作ともシステム・メニューとは異なる場合もある。それぞれのタイプのメニューは、ユーザーが欲するコマンドがどこにあるのかを知るのを助けるために標準化されている。

（１）ウィンドウ・コントロールのメニュー

タスク管理、ウィンドウ管理の機能は、複数のウィンドウにディスプレイ空間を共有させることを可能にするものである。MacintoshとNextStepでは、これらのシステム・レベルの機能のいくつかをアプリケーション・プログラムによる同じセットのメニュー内に置いている。

これらのシステムのユーザーは、ほとんどのウィンドウ管理タスクのためのコントロールがウィンドウ境界にあることを知っていなければならない。その他のシステムでは、アプリケーション・プログラムで使われるメニューと物理的、概念的に区別されたメニューの中にウィンドウ・コントロールの機能を置いている。

Mac	NeXT	OL	Motif	MSW	PM
	インフォ	ファイル	システム	コントロール	システム
ファイル	ウインドウ	ビュー		ファイル	ファイル
編集	編集	編集		編集	編集
		プロップス		ビュー	ビュー
				ウインドウ	ウインドウ
				ヘルプ	ヘルプ

図３．２－６　標準メニューコマンド

（２）コンポーネント・メニュー

OpenLookは、テキスト・フィールド、スクロール・バー、スクロール・リストのようなコントロール類と同様に、ウィンドウのさまざまな構造的な領域に非明示的なポップアップ・メニューを設けている。OpenLookによって順守するようにいわれているコンポーネント・メニューについては、付録Bで詳細に述べてある。

（３）アプリケーション・メニュー

アプリケーションには、通常、それ自身のメニュー・セットが必要となるが、それは、アプリケーションとドキュメント・ウィンドウの両方において、システム・メニューと分離あるいは統合されて表示される。いくつかのアプリケーション機能は、ユニバーサルなもので、仮想的にすべてのプログラムのメニューに現われるものである。これらの共通する機能のメニュー構造上の位置と名前が大きくばらついていたら、システムは習うにも使うにも不必要に難しいものになってしまうだろう。

したがって、多くのウィンドウ・システムでは、多くのアプリケーションを通して一貫するようなメニューの設定となるようなリコメンデーションや標準を設けている（図３．２－７参照）。

メニューによって直接行うことのできないアプリケーション・ウィンドウの付加的な調整などは、データのそれ以上の調整が必要となる。データのパラメータとその意味は、次のセクションで述べられるコントロールによって与えられる。

	Mac	NeXT	OL	Motif	MSW	PM
移動				●	●	●
サイズ		●		●	●	●
ミニマイズ		●	●	●	●	●
マキシマイズ			●	●	●	●
リストア		●	●	●	●	●
アプリケーション終了				●	●	●
前面のウインドウを全ての背後に置く			●	●		
表面のリフレッシュ			●			
ウインドウ属性の変更			●			

図３．２－７　コントロール・メニュー

３．２．５　コントロールとコントロール・パネル

効果的にデザインされたコントロール類は、グラフィック・ユーザー・インターフェイスに具体的、物理的オブジェクトで理解可能なインタラクションのイリュージョンを生じさせる。コントロール類は、ボタン、ノブ、スイッチと外観も動作も似ているので、ユーザーは、コントロール類に対して経験済みの知識を活用することができる。このダイアログが生じる標準的なコンテクストの例は、図３．２－８の代表的なコントロール・パネルに見られるものである。

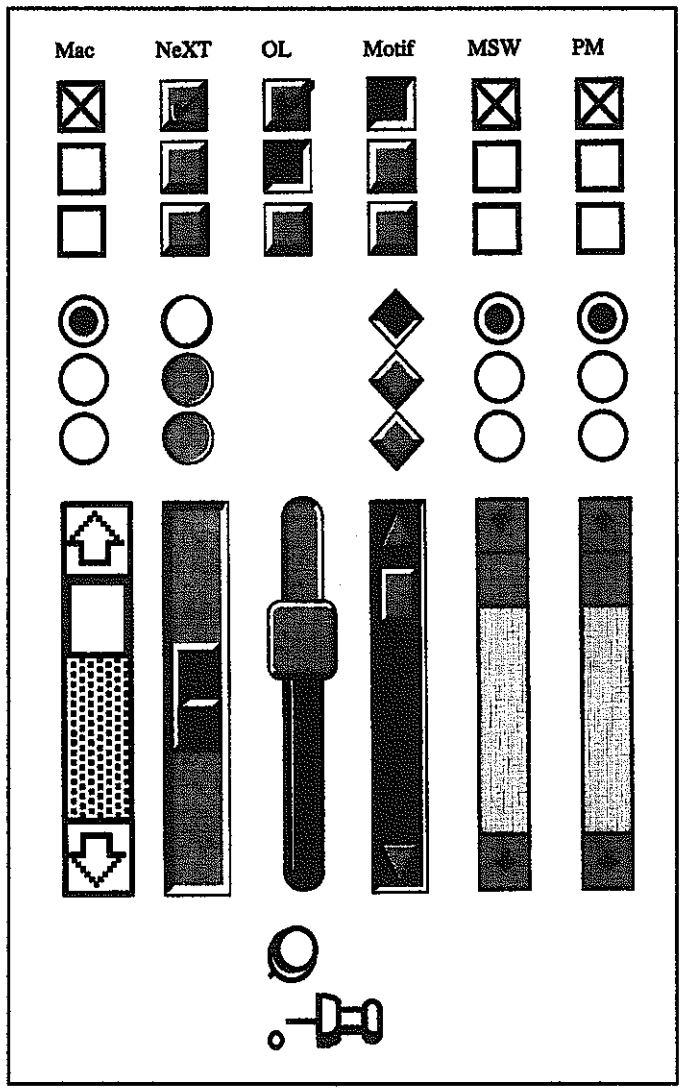

図３．２－８　標準コントロールの例

一般的に、あるタイプのコントロールの動作は、各システムを通して同じようなものであるが、その視覚的な表現はそれぞれかなり異なっている（図３．２－９、３．２－１０参照）。

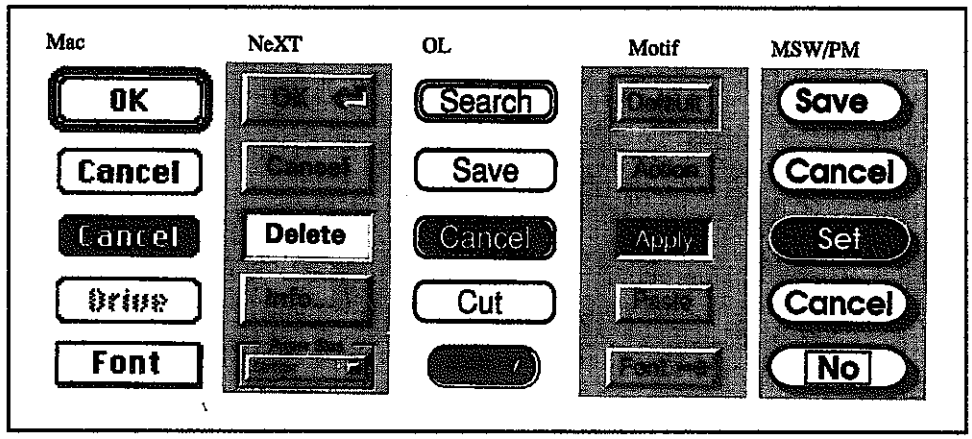

図３．２－９　標準ボタンの例

６つのウィンドウ・システムごとに、いくつかの標準的なコントロールのタイプが定義されており、アプリケーション開発者がつくるコントロール・パネルの基礎となっている。各システムの標準コントロール類のセットは、次のような情報の５つの基本的なクラスを見たり、修正したりする方法を与えるものである。５つのクラスとは、排他的な設定、非排他的な設定、比例的なスライダ・コントロール、テキストフィールドとリスト、そしてコマンド・ボタンである。

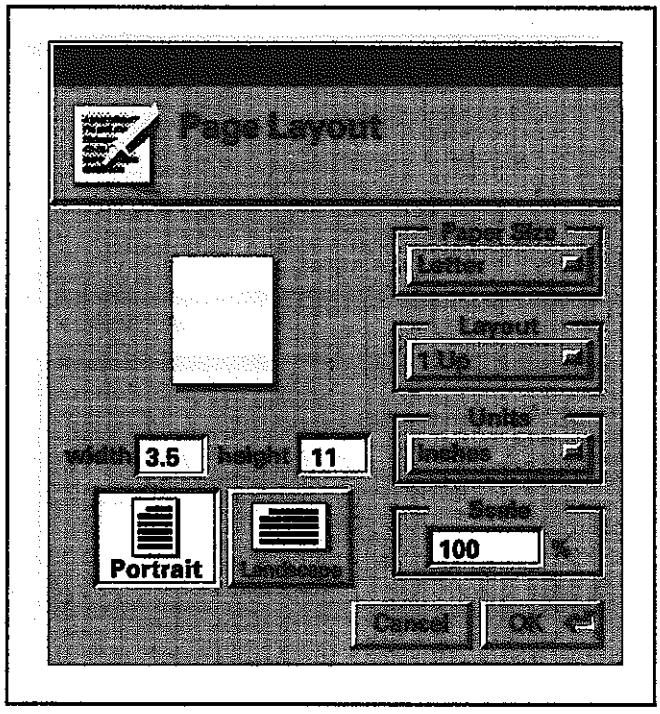

図３．２－１０　コントロールパネルの例

（１）排他的設定

排他的な設定とは、選択肢のグループからただ一つの値を選び出す方法である。よくある自動車のラジオに見られるようなプリセット・ボタンは、ユーザーに対する概念の説明としてしばしば使われている。

（2）非排他的設定

非排他的な設定とは、ユーザーが個別にセットしたりクリアしたりできる属性（アトリビュート）や骨性（プロパティ）のリストである。非排他的設定は同時にいくつでも表示することができる。

比例的なスライダは、定義された範囲の中で、連続的に変化させることのできる変数を表現するのに使われる。スライダは、パラメータの数にかかわらず有効で、特に多くの選択肢を持つ場合には排他的設定よりも楽に表示ができる。ダイアルやゲージも、ユーザーがその値を変更しない受動的な形で、同様な種類の情報を扱うことができる。

（3）テキスト・フィールドとリスト

テキスト・フィールドは、ユーザーがシステムに文字を入力できるようにするためのもので、多少の編集機能を持つ。リストは、編集能力を待つものもあるが、通常はすでにあるオブジェクトのグループからの選択のために使われるリード・オンリーの仕掛けである。スクロール・リストは、リストの置かれる空間よりも多くの項目を表示するためのグループとして使われる。標準的なウィンドウのスクロール・バーや、より一般的な比例スライダは、リストの表示されている部分を知るために使われる。

（4）コマンド・ボタン

コマンド・ボタンは、特定のシステムの動作を実行する指令を表すが、他のコントロール類と違い、システムの状態をただちに変更してしまう。また、コマンド・ボタンは、ユーザーがメニュー・システムを通してナビゲーションはできないが、即座に実行されてしまうという点では、メニュー・コマンドと同じものである。

コマンド・ボタンは、システムが即座に対応するという点で他のコントロール類とは違いがある。たとえば、コントロール・パネル上で「キャンセル」を押すか、クリックすると、パネルは消えてシステムは以前の状態に戻る。メニュー項目と同様、コマンド・ボタンも、選択が可能か、付加的メニューやダイアログ・ボックスが表示されるか、といった情報を持つこともある。各ウィンドウ・システムから出されるコマンド・ボタンの視覚的な扱いを図３．２－９に示す。

いわゆるコントロール・パネルやダイアログ・ボックスには、通常、キーボードのリターン・キーのような単純なアクションによって受け付けられる、予め選ばれたデフォルトの反応がユーザーに与えられている。デフォルトの反応は、そのコマンドが後戻りのきかない、あるいは危険性を帯びたオペレーションになるような場合を除いて、普通もっともよく使われるコマンド・ボタンに割り付けられる。デフォルト・ボタンは、より目立つように付加的な視覚情報（囲みの枠を太くするなど）によって特定させる（図３．２－９、図３．２－１０）。

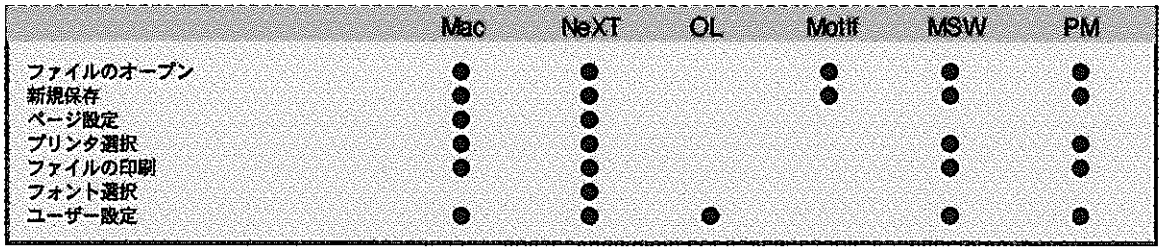

図３．２－１１　標準コントロール・パネル

よく用いられる標準的なコントロール・パネルが、それぞれのウィンドウ・システムで定義されている（図３．２－１１）。標準的なパネル・デザインによって、アプリケーションを通したユーザーのインタラクションの一貫性を増すことができる。個々のアプリケーションで、デフォルトのコントロール・パネルを多少変更しなければならないとしても、一貫した基本構造が存在することは、互いに異なるプログラム間でさえ、基本的なタスクにある程度の親しみやすさを獲得することができる。標準的なデザインが広く受け入れられるものであるならば、システム同様、いろいろな新しいアプリケーションの学習課程は簡易化されるだろう。

コントロール類やコントロール・パネルは、ユーザーをコンピュータと直接に対話させる方法だが、その他のコマンド・コントロールの状態表示のやり取りも、しばしば必要となる。さまざまな質問ボックス、メッセージ・ボックスが、この対話を支える方法としてある。

質問ボックス、メッセージ・ボックス（図３．２－１２、図３．２－１３）は、ユーザーが出すものではなく、アプリケーションが出すものである。これらのボックスは、コントロール・パネル同様、より一般的なダイアログ・ボックス・クラスの特別なケースである。質問ボックスは、通常、やはりモーダルであるが、ユーザーがディスプレイ上のその他の情報を確認するために、質問中もボックスを移動できるようになっているものもある。メッセージ・ボックスは、常にモーダルな状況で出され、その中の有効な選択肢がユーザーによって明確に選択されないうちは、それ以上のオペレーションをとることができない。

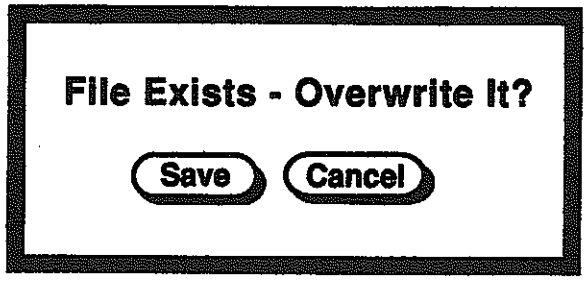

図３．２－１２　質問ボックスの例

図３．２－１３　メッセージ・ボックスの例

質問ボックスによって、システムはユーザーから特定のちょっとした情報を引き出すことができる。これは、通常、ユーザーに対して必要な情報を質問という形で問うものである。たとえば、「次のファイル“最終セクション”を消去してよいですか？」など。

メッセージ・ボックスは、危険性のある、あるいは不正な状態に入ったことに注意を促すものである。時には、リカバーのチャンスもなく、ユーザーは単に二次的なダメージを最小限に食い止めるために警告されるだけのような時もある。また、ユーザーには、いくつかの選択肢が残されることもあるが、たいていの場合ユーザーは、システムが問題を処理しようとする前にメッセージの受領を確認するだけ、ということが多い。それぞれのシステムでは、メッセージ・ボックスは、ユーザーの注意を引きつけるためにビープ音とともに表示される。

ダイアログ・ボックスは、ひとまとまりの意味のある認知的タスクを具体化したものであるが、これは、ユーザーの視点から見ればひとつの工事がなされることを意味する。これらの正しいデザインは、しばしば製品デザインのキー・ファクタとなる。

３．２．６　マウス／キーボード・インターフェイス

このセクションでは、グラフィカル・インターフェイスのマウスによるインタラクションの基本的特性について述べる。各ウィンドウ・システムは、同じような基本的コマンド・セットを使用している、すべてのウィンドウ・システムでサポートされている。基本的な機能は３つ（選択、メニュー、調整）ある。ボタンの数が３つ以下のマウスでは、ユーザーが行おうとしているオペレーションを決めるために、キーボード修飾子やポインタの位置を使われなければならない。

（１）マウスの機能

選　択

ディスプレイ内でコントロール類やデータ・オブジェクトを操作したり、引き続くオペレーション（いわゆる、メニュー・コマンド）に関係するオブジェクトの選択などに使われる選択機能は、この３つの基本機能の内ではもっとも重要性の高いもので、全部のボタンの中で一番使いや

すいマウス・ボタンが割り当てられる。

メニュー

メニュー機能は、通常、非明示的なポップアップ・メニューの表示に使われる。メニュー・ボタンに割り当てられたボタンは、ディスプレイのどこででもポップアップ・メニューを開くことができる。いいかえると、メニュー機能はポインタの位置によってではなく、イベントを生じるボタンと設定されていることによって、実行されるということである。

調整

調整機能とは、選択された項目の数を増やしたり、減らしたりすることである。メニューや選択機能よりも必要とされる頻度は少ない。したがって、マウス・ボタンへの割当は最後に行われる。

（2）マウスの動作

どのマウス・ボタンによっても、実行される動作は限られたものである。5つのオペレーション（ポイント、クリック、タブル・クリック、プレス、ドラッグ）が基本単語を形成し、マウスを使用したシステムとのインタラクションのベースとなる。

ポイント

マウスのピクセル・レベルの選択ポイント（「ホット・スポット」）を、ディスプレイ、スクリーン上の目的の場所にくるようにマウス・ポインタを合わせる。

クリック

目的の項目をポイントし、動かないようにしながらマウス・ボタンを押し、ただちに離す。

ダブル・クリック

目的の項目をポイントし、動かないようにしながらマウス・ボタンを押して離すことをすばやく2度行う。

プレス

目的の項目をポイントし、マウス・ボタンを押し続ける。

ドラッグ

目的の項目上でマウス・ボタンをプレスする。そしてボタンを押し下げたまま、マウスを適切な方向に動かす。

キーボードをよく使うアプリケーションにおいて、あまりマウスを使わないユーザーは、マウスとキーボード入力の組み合わせが必要な操作は非常に不便に感じるかもしれない。手をキーボードから

離して、マウスを探し、操作し、再びキーボードの正しい位置に手を戻すという、比較的高い労力は、マウスのもつ便利さを相殺してしまう可能性がある。

しかしながら、キーボード入力があまり必要でない状況では、論理的一貫性があり、理解しやすい方法であれば、少数のキーボード修飾子によってマウス・ファンクションを拡張することも可能である。また、ある場合には、キーボードとマウス操作のコンビネーションは避けられないものでもある。たとえば、ワン・ボタン・マウスを用いたときのOpenLookでは、キーボード・メニューの修飾子を使用しなければ、非明示的なさまざまなポップアップ・メニューによって得られるパワーと柔軟性をあきらめなければならない。

マウスは、ウィンドウやコントロールの基本的な操作を与えるものだが、各ウィンドウ・システムは、キーボードからも同様にウィンドウ・システムの機能にある程度のアクセスすることができる。Microsoft Windowsでは、すべてのシステム・ファンクションに対して、キーボードによる等価の操作方法を与えることが明確な目標となっている。

状況に応じてマウス・ポインタの図柄を変えることによって、システムの現在の状態を意味のある視覚フィードバックとしてユーザーに返すことができる。ポインタは、通常、ユーザーの注意がいくところなので、予期しない動作が実行された時に、ただちにユーザーに警告を与えることができる。また、ポインタの図柄は、現在のオペレーション・モードを常に表示することによって、ユーザーがどこにいるのかを示すことができる。各システムの標準的なポインタ・イメージを図３．２－１４に示す。

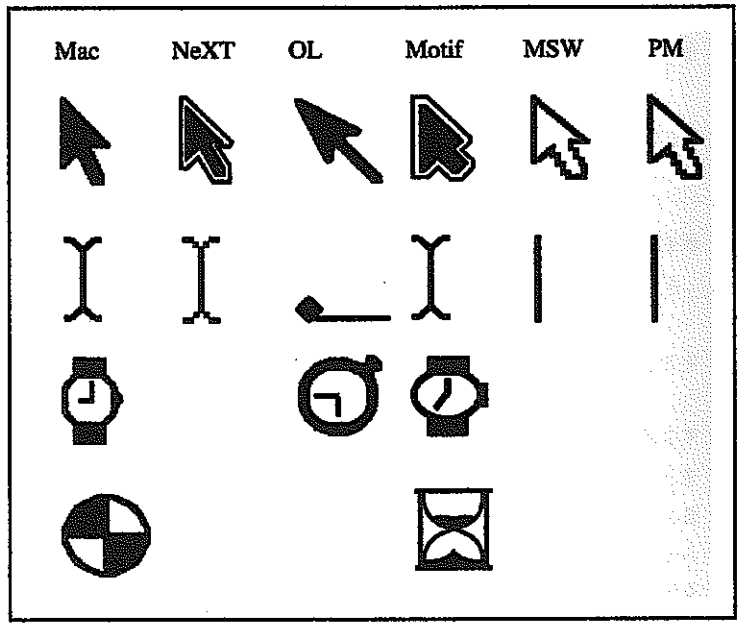

図３．２－１４　マウス・ポインタのイメージ

（３）ポインタのジャンプ

OpenLookの一般的な骨性として、ユーザーがポインタを置きたい地点をシステムが「知っている」時には、マウス・ポインタを自動的に動かす、というものがある。この動作はポインタ・ジャンプと

呼ばれる。

（4）マウス・ボタンのマッピング

Macintoshを除いたウィンドウ・システムでは、3個のマウス・ボタンを使用することができる。1つ以上のボタンが使えるとき、ユーザーはボタンのクリックやプレスがどのような結果を生じるのかを覚えておかなければならない。この問題は、マウス、ファンクションが配置されている論理的カテゴリ（選択、メニューなど）をしっかりと一致させることによって簡易化される（図3．2－15参照）。

	Mac	OL	NeXT	Motif	MSW	PM
ワンボタン	選択	不可	選択	選択	選択	選択
ツーボタン（左）	不可	選択	選択	選択	選択	選択
つーぼたん（右）	不可	調整*	メニュー	メニュー	キャンセル	キャンセル
スリーボタン（左）	不可	不可	選択	選択	選択	選択
スリーボタン（中）	不可	不可	調整	メニュー	キャンセル	キャンセル
スリーボタン（右）	不可	不可	メニュー	自由	自由	自由

図3．2－15　マウスボタンの割当例

（5）キーボード・フォーカス

アクティブなウィンドウとは、キーボードのフォーカスを待っているものと考えてよい。なぜなら、ユーザーによって他のウィンドウがアクティブにされるまで、そのウィンドウがすべてのキーボード入力を受け付けるからである。ウィンドウは通常、キーボードを使う必要のないオペレーションの場合でも、ユーザーによって操作される前にキーボード・フォーカスを得る必要がある。OpenLookでは、この考え方の例外として、キーボード・フォーカスを得ずにウィンドウを選択、操作することができる。

ほとんどのウィンドウ・システムは、ウィンドウのタイトル・バーの外観を変えることによって、アクティブなウィンドウがわかるようになっている。Motifでは、キーボード・フォーカスのあるウィンドウによって決まるパレットにマッチするように、全体のディスプレイの色を変化させることができるようなグローバルなカラー・コードを持っている。

（6）キーボードによるウィンドウ管理

キーボードからのウィンドウ管理コマンドによって、大さな高解像度のディスプレイでマウスを動かし、どこかに置くといったことを行わなくてすむ。また、キーボードによるウィンドウ管理は、マウスでは直接実行できないようなオペレーションも実行できる。

キーボードによるコントロールは、オーバーラップ・ウィンドウで、大きなウィンドウが小さなウィンドウを完全に隠すことを許しているような場合には特に有効になってくる。隠れたウィンドウをドラッグして表に出すかわりに、キーボード・ユーザーはウィンドウの順序をずらして、自分の欲しいウィンドウが前面にくるまでウィンドウの重なりを1枚づつめくり、手前に持ってくることができ

る。

マウスを優先使用したいという嗜好と、シンプルで一貫性のある概念モデルを提供するという目的によって、Macintoshはシステム・レベルのファンクションに対するキーボード・アクセスを制限している。その他のシステムでは、より広い選択肢を残している。

各ウィンドウ・システムは、メニュー・コマンドで共通に使われるキーボード・ショートカットを備えている。特にマウスの使用が重要でないアプリケーションにおいて、キーボードから手を離すことによる人間の動作的なコストが生じるので、この機能はタッチ・タイピストにとって重要なものとなる。

キーボード・ショートカット、または、アクセラレータはキーボード・ショートカットをサポートしていないOpenLookを除いたシステムでメニュー項目の中に明確に表示されている。

Motif、Windowsでは、キーボードによってすべてのメニューやコントロール・パネル項目にアクセスできる方法を持っている。メニュー・ショートカットによる単一のキー・コンビネーションの代わりにユーザーは、実行キーとスペース・バーを用いたキーボード・ナビゲーションによってメニュー・システムをブラウズすることができる。質問ボックスやコントロール・パネルも、スペース・バーやタブ、エンター・キーを使用することによってナビゲートできる。

OpenLookは、キーボードによってマウス・オペレーションをエミュレートすることにより、マウスなしのハードウェア設定をサポートしている。矢印キーがポインタを動かすのに使われ、その見え方と動作はマウスがあるときと同じである。すべてのマウス・ボタン・ファンクションは、ユーザー・プリファレンス・ダイアログ・ボックスを使って、どのようなキーの組み合わせにも設定できる。ユーザーは、このダイアログ・ボックスによって、すべての必要とされる動作の決まりを定義できる。

ウィンドウ・システムでは、いずれも選択用のマウス・ボタンで、タイトル・バーをドラッグすることによってユーザーがウィンドウを移動できる。OpenLookでは、ウィンドウ上部のヘッダーによるのと同様、サイドの領域によってもドラッグできる。この方式は高解像度ディスプレイで、望みのマウスの動作量をかなり減らせる可能性がある。また、Motif、Windows、のユーザーは、キーボードだけを使ってウィンドウを移動することができる。

ウィンドウ・システムによって、サイズをコントロールするオペレーションや表現方法、それにサイズ変更時の視覚的フィードバックなどはかなり異なっている（図３．２－１６参照）。

	Mac	NeXT	OL	Motif	MSW	PM
キーボードからのサイズ・ウインドウ				●	●	●
左辺（1D）からのサイズ・ウインドウ				●	●	●
右辺（1D）からのサイズ・ウインドウ				●	●	●
上辺（1D）からのサイズ・ウインドウ				●	●	●
下辺（1D）からのサイズ・ウインドウ		●		●	●	●
左上コーナーからのサイズ・ウインドウ			●	●	●	●
右上コーナーからのサイズ・ウインドウ			●	●	●	●
左下コーナーからのサイズ・ウインドウ		●	●	●	●	●
右下コーナーからのサイズ・ウインドウ	●	●	●	●	●	●

図３．２－１６　ウィンドウの拡大・縮小機能

Macintoshは、ウィンドウの右下のコーナーからだけしかサイズ変更ができない。ウィンドウをスクリーンの方向や左方向へ拡大するためには、ユーザーは、はじめにウィンドウのサイズを大きくする方向にドラッグしてからサイズ変更コントロールを使って、下辺を目的の方向へドラッグしなければならない。

OpenLookは、すべてのコーナーからサイズ変更できるメカニズムを備えており、またユニークな機能として、ウィンドウとその内容を同一形状でスケーリングすることができる。スライダー・コントロールを使ってウィンドウを拡大、縮小した後に、新しいパラメータを適応させるためのボタンを押すことによって、ユーザーはその結果を見ることができる。スケール機能は、ディスプレイを節約したり、ドキュメント・ウィンドウに優先順を付けたり、また非常に小さいサイズの文字を含むテキストを編集しやすくするのに使われる。

Motif、Windowsはリサイズ・コントロールにもっとも柔軟性があり、すべてのコーナー、すべての辺にハンドルを持っている。考えられるすべてのリサイズ・オペレーションの動きの軸に添ったハンドルがつくという、一貫性のある理由があるので理解しやすいという良さがある。ウィンドウ・コントロール・メニューからサイズ・オプションを実行することによって、ウィンドウの周囲を調整するのにキーボードと矢印キーを使うことができる。

また、マウスでサイズ・コントロールがうまく選択できているときに、Motif、Windowsだけがポインタのイメージを変化させる。ポインタをどのサイズ・コントロール上で止めても、ポインタのイメージは、デフォルトの選択矢印の形から辺を自由に動かせる方向に対応した２つまたは４つの矢印を持った形に変化する。ポインタ・イメージを変化させるという視覚的なフィードバックは、ユーザーに境界の持つ意味を知らせるとともに、比較的狭いコントロールを選択しやすくすることができる。

キーボードは、同時に１つのウィンドウにしか入力できないので、ユーザーは文字入力を受け付けるウィンドウを変更したり、特定できなければならない。Macintoshのシンプルな概念モデルでは、ウィンドウがディスプレイの一番手前に持ってこられた時にのみフォーカスを持つようになる。ウィンドウにキーボード入力するためには、ユーザーはそのウィンドウの境界内のどこかを単純にクリックする。

OpenLookのウィンドウは、マウスの選択または調整ボタンがテキストの挿入されるエリア内のどこかでクリックされたときに、自動的にキーボード・フォーカス（「入力エリア」と呼ぶ）を受け付ける。オプション的な設定でマウスがウィンドウ内に入れば、フォーカスが変わるというようにもできる。フォーカスはキーボードによっても変えることができる。ユーザーは、個々のウィンドウの重なりの順序を変えることなく、重なったウィンドウ間でフォーカスを循環させることができる。循環の順序は、ウィンドウを使った順またはその逆が可能である。キーボードは、また、もっとも最近使った２つのウィンドウを交互にフォーカスすることができる。

Motifは、ユーザーがウィンドウの前にポインタを動かすだけで、そのウィンドウに自動的にフォーカスが当たるように設定することもできる。しかし、デフォルトの設定では、ユーザーがウィンドウの境界内でクリックしたときにのみキーボード・フォーカスが変わる。

また、キーボード・フォーカスは、alternateキー、shitキー、tabキーを使用することによって、重なったウィンドウ順か、その逆方向に循環させることができる。Motifのデフォルトでは、キーボード・フォーカスを受け付けたときに、自動的にそのウィンドウが重なりの一番手前に来るようになってい

ないが、システムをそのように設定することもできる。また、ユーザーまたはアプリケーションが定義したディスプレイのウィンドウやアイコンのサブセットの中で、キーボード・フォーカスが循環するのはMotifだけのものである。

Windowsでは、ウィンドウは重なりの手前にきたときにのみキーボード・フォーカスを受け付ける。ウィンドウの順序とフォーカスを同時に変えるたくさんのキーボード・オペレーションがあり、これについては、次のサブセクションで述べることにする。

オーバーラップ・ウィンドウの順序を変えることは、ウィンドウの移動やサイズ変更よりも頻度の高い仕事である。したがって、この仕事を達成するメカニズムの善し悪しは、システムの使い勝手を決めるのに決定的な要因であるといってよい。どのシステムでも、ユーザーが部分的に覆われたウィンドウの領載内でマウスの選択ボタンをクリックすれば、そのウィンドウは、ディスプレイの前面に来る（図３．２－１７）。Macintoshのヒューマン・インターフェイスでは、これが唯一のウィンドウの順序を変えるメカニズムである。その他のシステムは、この基本的な順序変更のテクニックのたくさんのバリエーションを提供している。

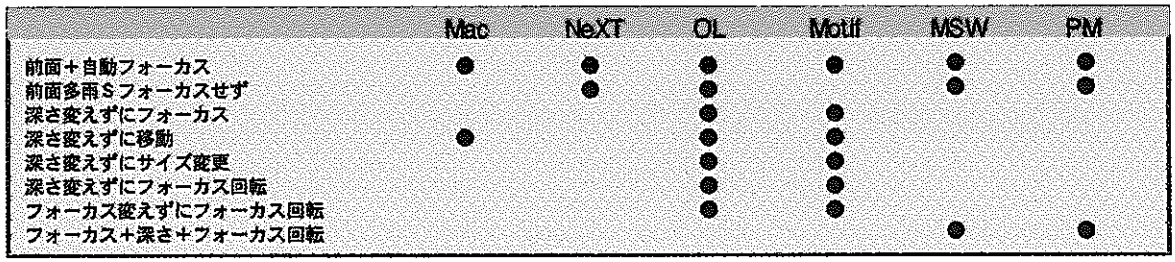

図３．２－１７　ウィンドウ・フォーカスと重なり

OpenLookでも、ウィンドウのテキスト・フィールド以外の領域でマウスの選択ボタンをクリックすることによって、ウィンドウにキーボード・フォーカスを与えずに前面に持ってくることができる。OpenLookは、ウィンドウ・コントロール・メニューの選択肢によって、ウィンドウをディスプレイの後方に送り出すことができる。このテクニックは、大きなウィンドウが後方にあるウィンドウへのアクセスを邪魔しているようなときには有効である。

Motifでも、OpenLookのようにウィンドウ・コントロール・メニューにウィンドウを後ろに送る（「Lower」と呼ぶ）選択肢があり、手前のウィンドウを重なりの一番後に送り、２番目にあったウィンドウを手前に持ってくることができる。キーボード・コマンドで、この機能は拡張され、ユーザーはフォーカスを変えずに、前から後ろ、または後ろから前にウィンドウをシャッフルすることができる。この機能を実行する論理的なキーボード・イベントは、ユーザーによって物理的なキーに好きに割り当てることができる。

Windowsのユーザーは、ウィンドウの重なりを順方向、逆方向に循環させたり、手前にある２つのウィンドウを入れ替えたり、また、ウィンドウをアクティブにしたり、自動的に前に持ってくる前に残りのウィンドウの順序を変えずにウィンドウを前から後ろにプレビューすることができる。このプ

レビュー・モードでは、ウィンドウやアイコンはフルサイズに開かれるが、それぞれのウィンドウの境界とタイトルバーだけが表示される。ウィンドウの内容は描画されないので、このプロセスは、ユーザーが目的のウィンドウを見つけるまで高速で繰り返すことができる。これらのオペレーションは、alternateキー、エスケープキー、シフトキー、スペースバーの組み合わせによってキーボードから実行することができる。

ドキュメントや、見えている物理的なウィンドウ・サイズより大きいスペースをスクロールすることは、すべてのウィンドウ・システムでかなり頻繁に発生するタスクである。スクロール・メカニズムのデザインや、それがサポートする動作のタイプは、このクリティカルなタスクのパフォーマンスに大きな影響を持っている。

（7）スクロール・バー

表示しているデータのサイズが、有効な表示スペースより大きいときにスクロール・バーが必要になってくる。典型的なスクロール・バーは3つの基本的な要素を持っている。スクロール・コンテナ、スクロール・ハンドル、スクロール・アローがそれである（図3．2－18参照）。

（8）スクロール・コンテナ

スクロール・コンテナとは、スクロールハンドルが移動するための道である。スクロール・ハンドルの動きは、通常、垂直または水平軸方向に限られているが、両方向を同時に調整するような二次元的なスクロール・コントロールも可能である。

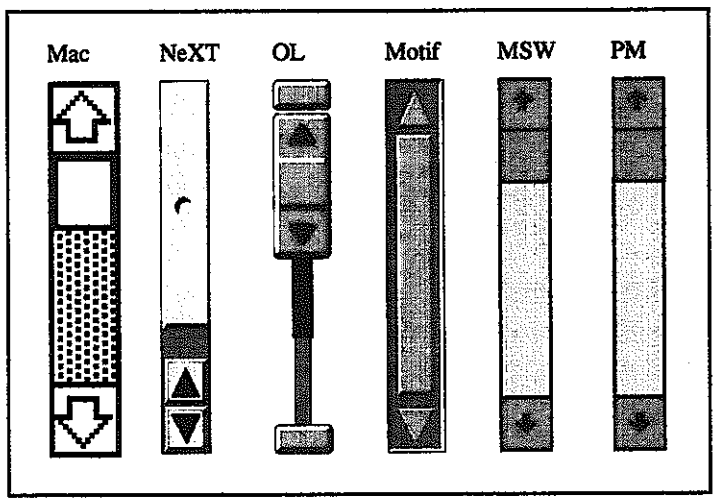

図3．2－18　標準ウィンドウ・スクロール・バー

（9）スクロール・ハンドル

スクロール・ハンドルは、表示されているデータ内での現在の位置を特定するためのオブジェクトである。マウス・ポインタでスクロール・ハンドルをドラッグすることによって、ユーザーはウィンドウ上に現われるドキュメントの部分を変えることができる。

スクロール・バー上の2つのスクロール・アローは、データ・オブジェクトの境界の外方向へ一行

づつスクロールさせるものである。「ウィンドウ移動」という考え方のもとでは、スクロール・アローの示す方向は、それがクリックやプレスされたときに、データに対してウィンドウの枠をその方向へ動かすことを意味する。したがって、データ領域はアローの向きとは逆方向に動くように見える。

ウィンドウ・システムでは、スクロール・バーはマウスの選択ボタンによって操作される。定義によれば、スクロールバーによって、ユーザーはマウスでスクロール・ハンドルを選択し、スクロール・コンテナ内の好きな位置にドラッグすることができる。また、スクロール・アローの動作はどのシステムでも一貫している。スクロール・アローのクリックは、ウィンドウの内容を一行分移動する。アローが押し続けられれば、ウィンドウの内容は、マウス・ボタンが離されるまで連続的にスクロールされる。

どのシステムも「データ移動」ではなく、「ウィンドウ移動」に基づくスクロール・モデルを採用している。ウィンドウ移動モデルでは、スクロール・アローのプレスや、スクロール・ハンドルのドラッグは、アローの向きや、スクロール・ハンドルの移動方向とは逆向きにウィンドウ内容をスムーズに移動させる。この対応は不自然に感じるが、テストによって、ユーザーにはこのモデルがはっきりと好まれることが証明されている。

Macintoshのヒューマン・インターフェイスで採用されているスクロールの動作が、これらのシステムの中ではもっともシンプルである。Macintoshユーザーは、スクロール・コンテナ内でマウス・ボタンをプレスまたは繰り返しクリックすることで、ドキュメントのページをめくる（ウィンドウの大きさ分のスクロール）ことができる。しかし、この機能は基本的なスクロール・テクニックを強化したにすぎない。

Macintoshのスクロール・アローは、スクロール・コンテナの両端にある。この配置は、スクロール・アローやスクロール・ハンドルが操作されたときに、同じ方向（現在の位置から離れる方向）にマウスが動かされるという概念モデルの明快さを最大限活かしたものである。しかし、小さいウィンドウを除いて、スクロール・アローが大きく離れていることによって、ウィンドウ内容をアジャストするために、スクロールの方向を切り替える労力が著しく増している。

OpenLookのスクロール・バーは、比較的複雑な概念モデルに基づいている。これは拡張された機能と、オペレーションに複数のモードを持つものである。たとえば、OpenLookのスクロール・バーは、常に非明示的なポップアップ・メニューが組み込まれており、これによって、ユーザーはドキュメントの初めまたは終わりに、またドキュメント内の以前いた位置に直接移動することができる。ユーザーはスクロール・バー・メニューでウィンドウを分割したり、分割された部分を結合したりすることができる。

OpenLookのスクロール・バーは、キーボードからも操作できるライン、ページ、またドキュメントの終端やサイド方向へのスクロールが論理的にキー配列できる。ユーザーは、スクロール・コンテナの両端の直立したバーをクリックすることによって、ドキュメントの始めか終わりに直接移動することができる。スクロール・コンテナ自体の内側での位置クリックの動作はしない。この領域内でのクリックは、ウィンドウ内容のページをめくり、スクロール・ハンドルをポインタの方へ動かす。

ほかのシステムと違い、OpenLookは、対応する方向への動きが不可能なとき、スクロール・アローは暗くなる。ウィンドウ内にドキュメントが全部見えているときは、両方のアローが暗くなっている。ドキュメントの一部分だけが見えているときのみ、スクロール・バーの動きの軸を表す黒いラインは、

ウィンドウ内に見えているのが全体のドキュメントのどれくらいの分量にあたるのかを示すために、部分的に太くなる。ウィンドウのサイズが小さくなるにつれ、スクロール・バーの構成要素は2つのスクロール・アローが残るまで省かれていく。

OpenLookは、スクロール・アローをスクロール・ハンドルの両端に置いている。これは、構成パーツと動きの方向が、空間的に好ましい関係を維持するために、分離を最少限に押さえようとしたものである。スクロール・ハンドルの長さは、ドキュメントの見えている部分に対応して変化するわけではないので、スクロール・アローは、どの方向にもスクロールしやすいように十分近付けて置かれている。

	Mac	NeXT	OL	Motif	MSW	PM
前面＋自動フォーカス	●	●	●	●	●	●
前面多雨Sフォーカスせず	●	●	●	●	●	●
深さ変えずにフォーカス	●	●	●	●	●	●
深さ変えずに移動			●	●		
深さ変えずにサイズ変更		●				
深さ変えずにフォーカス回転						

図3．2－19　スクロールの動作

スクロールし続けている間、スクロール・アローはスクロール・バーと一緒に動いていく。このプロセスの間、ユーザーがマウスを動かしてしまうのを避けるために、OpenLookは自動的にマウス・ポインタをスクロール・アローに添って動かす。システムではなく、ユーザーが全てのオペレーションのコントールを行使すべきであるということは広く受け入れられていることであり、ポインタのジャンプ（マウス・ワープと呼ばれる）を行うことは、ユーザー中心のデザインの基本的な原則に明らかに違反している。

このケースでは、ポインタ・ジャンプの利益がコストより勝っているかもしれないが、その結果である方向感覚が失われることは、この動作が推奨される前に注意深い評価を行わなければならない。スクロール・アローの配置は、表現の明快さと人間の実行に関する効用という基本的なトレードオフを反映したものとなっている。3つのデザイン・ソリューションの相対的な効用を述べてきたわけだが、これは実験によって解決されるべき問題であろう。

Motif、Windowsは、スクロール・バーの両端にスクロール・アローを持ち、スクロール・コンテナのマウス・クリックによってスクリーンいっぱいのページめくりを行うという、Macintoshで採用されているようなスクロールに関する概念モデルにしたがっている。また、これらのシステムでは、キーボードでも目的の方向にスクロール・ハンドルを動かすことができる。Motifは、これにドキュメントの両端に直接移動できる機能を加えている。これを行うには、スクロール・コンテナをクリックするときにシフト・キー（先頭）またはコントロール・キー（終端）を押す。

OpenLookで採用されている拡張されたオンライン・ヘルプでは、ウィンドウやコントロールをポイントしてヘルプ・キーを押せばいつでも実行できる。これによって、ポップアップ・ウィンドウが開かれ、マウス・ポインタで指されたオブジェクトに関しての状況に応じた情報が表示される。ヘルプ

機能は、ユーザーがどのキーにも割り当てることができる。OpenLookの基本オブジェクトに関する標準的なヘルプの文章は機能に関する仕様についてのものである。ユーザーは、アプリケーションの仕様に関する付加的な情報を、ヘルプウィンドウ内の「More」ボタンを押すことによって得ることができる。

現在、Motifは標準的なオンライン・ヘルプ機能を持っていない。これに対して、Windowsは標準のヘルプ・メニューとブラウザ・ウィンドウの動作と、標準ウィンドウ構造の中でそれらを表示する方法を規定している。ユーザーは、マウスかキーボードを使って、ヘルプ・メニューをプルダウンし、チュートリアル情報やヘルプ・システム自体に関するヘルプ、また、利用可能なトピックスの索引などにアクセスすることができる。メニューから項目を選択すると、スクロール可能なドキュメント・ウィンドウが開かれ、テキストによるヘルプ情報が表示される。

ヘルプ・メニュー（Fl＝ヘルプ）のラベルで、ユーザーはF1のファンクション・キーを使ってヘルプ・ウィンドウが開けられることがわかる。また、ユーザーはマウスのメニュー・ボタンでヘルプ・メニューのタイトルをダブル・クリックすれば、ヘルプ・ウィンドウを開くことができる。すべてのダイアログ・ボックス上には、ヘルプ・ボタンを置くことができ、これによって同じウィンドウに対して状況に応じた（context - sensitive）アクセスを与える。シフト・キーとFlキーを押すと、マウス・ポインタがクエスチョン・マークに変わり、ユーザーをヘルプ・モードにする。ユーザーは、それからオブジェクトをクリックし、関連するヘルプ情報を見ることができる。

それぞれのウィンドウ・システムで基本的なウィンドウ管理のタスクを達成する方法が異なるので、各システムは長所も短所も持っていることになる。

3．2．7　長所と短所

これまで述べてきたウィンドウ・システムは、さまざまな洗練のレベルで、ユーザーにとっての長所も短所も持っている。

Macintoshが用いたインタラクティブなコントロールの直接性は、全てのウィンドウ・システムの目標とすべきものである。マウスは、スムーズに均一に動き、オブジェクトがドラッグのオペレーション中、ポインタに遅れるようなことはない。より早い、より強力な機械でも、このシングル・タスクのMacintoshの応答性とインタラクティビティに匹敵するシステムはまだない。カーネル・ベースのMacintoshのヒューマン・インターフェイスの効率は、より複雑なオペレーティング環境でのヒューマン・インターフェイス上の未来の改善点を評価する有用なベンチマークとなると思われる。

しかし、Macintoshのその他の多くの性能的な特徴は、あまりぱっとしたものではない。シンプルなユーザー・モデルはシステムを学びやすいものにしてはいるが、エキスパートのユーザーに必要とされるような柔軟性には限界がある。従って、以下のような点に大きな限界がある。

ウィンドウがたった１つのコーナーからしかサイズを変えられない
スクロール・オペレーションが拡張されていない
ウィンドウを後ろに送る機能がない
ウィンドウをミニマイズする機能がない
マウスなしでのオペレーションができない
キーボード・コントロールに制限がある
拡張性に制限がある
キー・マッピングの設定に制限がある

Windowsが、MS - DOS環境への後ろ向きなコンパチビリティによって拡張に対する制限を受けているのに対し、Macintoshは、オリジナルのウィンドウ制御やシンプルなユーザー・モデルとのコンパチビリティの必要性から制限を受けている。その結果、多くの有用なコントロールの改良（たとえば、Windowsによる、より洗練されたサイズ変更のコントロールのような）がMacintoshにはなされないままである。

OpenLookのグラフィック・ユーザー・インターフェイスは、評価してきたシステムの中では最も拡張された機能を持つものである。この強力なパワーと柔軟性は、エキスパート・ユーザーために与えられたものだが、初心者にとっても、その習熟曲線はよいものである。OpenLookの多くのユニークな特徴が、次のようにヒューマン・インターフェイスを非常に複雑なものにしている。

マウスの選択ボタン、メニュー・ボタンに対する複雑な機能の割り当て、複数の状況に応じた（手掛かりのない）ポップアップ・メニューからフォーカスを外さずにウィンドウを選択できる、ウィンドウ、メニュー、ボタンの状態による拡張されたフィードバック、Motifのウィンドウ・マネージャおよびToolkitは、OpenLookのようにXWindowシステムによる効率的なクライアント・サーバーが移植されていれば利用することができる。

Motifは、三次元的なメタファを採用している。OSFは開発者に向けて、基本となるMotif Widgetセットを自分達の拡張に作り替えるように奨励している。この柔軟性は、Motifのウィンドウ・マネージャのデフォルトの動作でも引き継がれている。Motifは比較してきたシステムの中では、もっとも受容度の高いものである。その中でも最も価値のある特徴は次のとおりである。

リサイズ・コントロールの柔軟性、キーボードから全ての機能にアクセスできる、マウス・ボタンを完全に設定できる、キーの割当がユーザーによって容易に変更できる、ウィンドウとウィンドウ・アイコンのサブセットを循環させられる、ポインタのあるウィンドウに自動的にフォーカスが行く。Motifには欠点が少なくシステム自体を再設定できることによって救われている。Motifは、XWindowシステムのコミュニケーションと、表示プロトコルをサポートしている大半のUNIXアプリケーションとの互換性を持っている。このことから、DOSとUNIXベースの機器を結ぶコンピュータ環境ではMotifは理想的な選択になるだろう。

もとになるオペレーティング・システムや、標準ユーティリティの違いを別にすれば、Windowsの外観と動作はほとんど区別がつかない。Windowsの共通のスタイル・ガイドが、これらのシステムの収斂の度合いを示している。これらのシステムは、（Motifも含めて）拡張されたキーボード・コントロールの能力によって区別される。キーボード・コントロールに必要となるキー・ストロークと修飾

子の数多い組み合わせが、ヒューマン・インターフェイスの複雑性を増しているが、Windowsはヘルプ・システムへの容易なアクセス形態によってこれを埋め合わせている。これらは、ヘルプ機能が明確に標準的なウィンドウ構造内に表示できる唯一のシステムである。

Windowsの標準的なコントロール類は比較的理解しやすく、あまり経験のないオペレータでもすぐに使えるシステムになっている。また同時に、キーボード・ショートカットの拡張された使用は、もっとも極端なDOSユーザーにとっても満足のいくものだろう。Windowsのヒューマン・インターフェイス上の強力さは次のような点である。

- 柔軟性のあるリサイズ・コントロール
- 完全なマウスレス・オペレーションのサポート
- メニュー項目へのキーボードによる直接アクセス
- 強力なオンライン・ヘルプ機能
- すばやいプレビューオプションを使用したウィンドウ・シャッフル
- 前面のウィンドウと、2番目ないし最後のウィンドウとのすばやい切り替え

しかし、次のような多くのネガティブな特徴が、２つのシステムの機能を制限している。

- 記憶不可能なキーボード割当
- キー割当が、ユーザーによって簡単に変更できない
- いくつかのアプリケーションが、ウィンドウ・システムを無視して全画面を占有している

この項では、グラフィック・ユーザー・インターフェイスを基本とするウィンドウ・システムの例を比較してきた。これらの製品には、さまざまな効用と限界がある。これらは市場でお互いに競合しているので、常に変化しその特徴を改良している。今後10年間でどの製品がもっとも成功するかを予測することはむずかしいが、それらの本質的な特性をよく理解することがグラフィック・ユーザーインターフェイスの設計にとって重要である。

３．３　マルチ・エージェント／マルチ・プラットホームの対話機能

科学のさまざまな分野で計算し、シミュレートするための新しい方法が発見されるごとに、その分野が進歩するということがくり返されてきた。例えば、原子力工学、プラズマ物理学、宇宙物理学、分子生物学、地質学などがある。また、音楽や音声処理の分野でも、コンピュータは重要な役割を果たしてきた。

医学では、念入りに設計されたコンピュータ・プログラムとシステムが医療診断を補助している。例えば、ソルトレイク市復員軍人運営病院の精神科医は、このシステムを使って精神病を診断している。患者は、まず人間の受付係の挨拶を受けたあと、コンピュータ端末のある個室に通される。その

後は、コンピュータ・プログラムが一連のテストを行ない、20分後に職員の精神科医が質問を再検討し、患者の状態に関する最終判断を行なう。コンピュータ・プログラムによる診断は、患者にとって時間と費用の多大な節約になるばかりではない。もっとも重要なことは、コンピュータによる診断が、驚くほど正確であると実証されていることである。コンピュータは疲れないし、偏見をもつこともない。要するに、非人間性がここでは役立っているのである。その非人間性は、また、核燃料サイクル施設や列車運転等の遠隔監視でも役だっている。おそらく、コンピュータと医学の分野の最大の進歩は、体の各部の内部を表示し再構成してみせることのできる断層撮影法であろう。現在の磁気大脳造影法、磁気による心搏記録法を使えば、脳や心臓を発生源とする微弱な磁場を表示することができる。

要約すると、計算におけるコンピュータ利用によって、現在、科学者が問題を追求し概念化するスタイルが根本的に変わってきている。また、コンピュータ・モデルが、まったく新しい研究領域を開拓しつつもある。実際、科学的な手法のあらゆる変化の中で、コンピュータ利用による変化ほど重要なものはないと思われる。

実験から得られるデータはきわめて膨大であり、コンピュータの助けなしには、比較したり結論を引きだしたりすることはほとんど不可能であろう。例えば、すでに膨大な量のDNA塩基配列が発見されているが、コンピュータを利用してはじめて生態の遺伝物質の中にある、この塩基配列に隠れている相関の発見が可能になるだろう。同様に、地層処分性能評価では、膨大な量の地質条件についてその地球化学的特性と研究しなければならないが、これも前者と同様に調査により得られた固有の熱力学データベースやＫｄデータベースをコンピュータで効率よく利用することにより、予測手法の開発等が可能になるであろう。

教育の分野でも、コンピュータ化からは多大な恩恵を受けている。個人用ワークステーションの大学内ネットワークは、適切なソフトウェアと組み合わされ、補修学習に役立ってきた。学生たちはまた、計算物理学を学習するための道具としてコンピュータを使っている。コンピュータに接続されたビデオディスクは、コンピュータとビデオ技術を組み合わせた教育教材であり、化学、物理、工学の教授法を変えつつある。同様に、これらの技術はこれからの放射性廃棄物処分問題等の社会問題を適切に、一般大衆や地域住民に理解してもらうための重要なメディアとなるであろう。

コンピュータ応用の進歩は現在も続いている。コンピュータは、他のコンピュータの設計、ビデオ解析システム、タンパク質構造の決定、暗号システム、ロボット工学、分子進化の研究などで重要な役割を果たしている。小児マヒ・ウイルスなどのウイルスをコンピュータを使って３次元構造表示する技術は、新しい治療法の発見につながる可能性がある。雷雨の最新モデルでは、一片の雲が成長して竜巻きを起こす嵐にまで成長する過程のシミュレーションを行なうこともできる。巨大な地震検知システムは、コンピュータ、計測装置、自動計測電送、データ圧縮などの相互作用から構成されている。将来のコンピュータでは、おそらく、日中の各時刻での陽光の流れをシミュレーションすることができるようになるであろう。

原子力発電所やエイズ・ウイルス、経済やヒト・ゲノムのような複雑系にひそむ問題点は、そのメカニズムについて信頼できるモデルや理論を作るために必要な実験ができないという点である。例えば、原子炉メルトダウン（炉心融解）が起こる限界を知るという目的で実験することは許されない。従来は、そのような実験は不可能であったが、現在、私たちはこの類の複雑系に関する実験が行える条件を持っている。また、全体系を理解するために、現実のシステムを細分化し各部分の知識の統合

を目指してその部分部分を別々に研究する必要もない。

現在、信頼度の高い高品質のコンピュータが使えるので、複雑な現実世界のプロセスを再現する仮想世界を実際にコンピュータ内部に作ることができる。このような仮想世界を実験室として用い、物理や社会とか生物や行動に関する複雑なプロセスについて発展的な理論を作るための実験が行えるのである。

以上のようにコンピュータの進歩は、科学の発展を支える重要な基盤となっている。そして、人は、より深くコンピュータと係わることにより、その能力を高めた結果を得ることができる。そのためには、人とコンピュータとの情報交換（対話機能）が期待通りのコンピュータシステムを開発するための重要な技術となっている。地層処分ヴァーチャル・エンジニアリング・システムは、地層処分された高レベル放射性廃棄物が多重バリア方式により、長期に亙り生活圏より隔離されることをコンピュータを利用しながら確認していく機能を提供することを目的に開発するシステムである。それは、コンピュータ性能の飛躍的な発展に伴い、仮想環境をコンピュータ内部に構築し、考えられるシナリオに添って多方面に亙るシミュレーションを可能にするものである。そのためには、仮想環境と対話機能に係わる情報処理技術の開発が必要である。それにより、人工世界を構築し、幾つものモデルを複雑に組み合わせた複雑系によるシミュレーションを将来実現することを主目標としている。そのようなシステムに於ては、対話機能の質が直接シミュレーションの質に影響するであろう。

計画する試作システムは、複数のワークステーションとＰＣがネットワークに接続され、相互に協調して性能評価シミュレーションを進めるものである。対話システムは、前述してきたように採用するシステムの環境により技術が異なっている。このようなソフトウェアは利用者にとって使いやすい反面、ユーザインターフェイス部分の開発が難しい。特に、Motifベースの対話システムとWindowsベースの対話システム相互の通信や、各対話システムの背後で個々に独立したタスクとして動作するオブジェクト（以下、このような実行形態とエージェントと記す）の協調等込み入った機能の開発が必要となる。ここでは、そのようなシステムをマルチ・エージェント・ユーザーインターフェイスとして呼び、情報構造を検討する。

３．３．１　システム操作とマルチ・エージェント処理

地層処分ヴァーチャル・エンジニアリング・システムを利用した性能評価は、凡そ、以下のような質問を動機としてなされるであろう。評価にあたっては、試験データやFEP、そしてシミュレーション・コードや視覚化機能等をランダムに操作して試行錯誤的に作業が進められると考えられる。従って、対話機能は、それらのランダムな操作要求に答えるため、メニューやボタン、スライダ等を装備したグラフィック・ユーザー・インターフェイスにより提供する必要がある。

質問０１：地層処分システムは、最終閉鎖が完了するまで、放射能被爆、放射能レベル及び非管理区域への放射性物質の放出が、環境保護機関等の定める環境規制等の許容値以内にいつでも納まるように設計がなされているか？

質問０２：地層処分システムの地質は、予想される熱応力の影響下で放射性核種移行の抑制能力が増加するような鉱物質に変化したり、またはそのままで変わらずにいる鉱物質を有するか。

質問０３：地層処分システムの地層は、過去において処分システムが重大な影響を受けるような地震が発生したことがあるか？

問題０４：地層処分システムの地層は、地下水流れに悪影響を及ぼす隆起、陥没、摺曲、断層のような構造変形があるか？

問題０５：地層処分システムの地層は、動水勾配、割れ目での平均流速、蓄電率、導伝率、自然再充電、電位差レベル、放電位置のような、人の近づきうる環境への放射性核種移行に影響を及ぼす水理学上の変化の可能性を有するか？

質問０６：処分区域は、放射性核種の吸着を減少させたり、岩場のもつ強度を弱めたり、人工バリアシステムに悪影響を及ぼしたりする地球化学的過程を有するか？

質問０７：処分区域は、地下施設や坑道やたて坑の封鎖機構の設計、施行で複雑な工学手法を必要とする岩盤状態や地下水状態を有しているか？

質問０８：処分区域は永久閉鎖中、安定な地下開孔部を設計できないような地盤力学的特性を有するか？

質問０９：地層処分システムの方位、形態、レイアウト、深さや地下施設の一部である人工バリアは放射性核種の格納や隔離に有効か？

質問１０：地層処分システム中の通路は、作業が安全に実施され回収ができるように設計されているか？

質問１１：地層処分システム中の通路の掘削方法は、人の近づきうる環境へ地下水や放射性廃棄物が移行する優先通路の発生を抑えるような方法か？

質問１２：地層処分システムは、水やガスの侵入を制限できるように設計されているか？

質問１３：地層処分システムは、最大勢負荷、破砕帯やその他の不連続の条件、及び母岩、周囲の地層、及び地下水の熱輸送特性、地盤力学的、帯本地質学的、地球化学的システムの予想応答を考慮して目標を満足するように設計されているか？

以上の質問は、予想される質問のほんの一部であると考えられる。計画されている地層処分システムの性能評価シミュレーションのための技術は、このように非常に多岐に亙る専門領域の問題が複雑に組み合わさったものであり、特にランダムな試行錯誤的問題解決に陥りやすいと考えられる。

地層処分システムの性能評価シミュレーションには、およそ３種類のソフトウェア・モデルが必要である。

第１は、処分システム施設および地表の景観の物理的な記述、処分システムに処分される廃棄体の位置、種類および量に関する数値的な記述、それにシミュレーション対象地域の地層の特性記述等からなる物理モデルである。

第２は、廃棄物から漏れ出た放射性核種が、処分場近傍、及び地層中のそれぞれの場に及ぼす効果を予測する汚染物質移行モデルである。

第３は、地震や火山の噴火など、サイトの通常の操業にとっては外的要因である気候変動等の効果を計算に入れるための一連のモデルである。

それぞれの種類のモデルは、空間的、時間的に幅広いスケールで利用されるものでなければならない。また、空間的には、廃棄体スケール、処分ピット周辺スケール、処分場全体スケール、地層スケ

ールの４つのスケールを考えると良いであろう。

シミュレーションの時間区分に関しては、短期間の定義として100年という期間がしばしば使われるが、これは制度的な規制をそれよりも長い期間に亙って保証することはできないと考えるためである。長期間については100年～1万年の期間を想定することができる。これは、これまでに行われた研究によって、放射性核種、および環境上重要な意味を持つ微量元素の漏出の大部分は1万年以内に起こり、1万年以降は気候学的変動が廃棄物管理工学よりも大きな意味を持つことになる可能性が高いと考えられているためである。

（１）処分システムの情報

処分システムの情報には、処分システムの物理的構造、および処分システム内の廃棄物の分布を記述するのに必要なすべての情報が含まれる。物理的構造の記述には、処分システム内のすべての廃棄体、ボアホール、立坑、通路および室の位置ならびに寸法が含まれている必要がある。また、地表の景観情報も含まれている必要がある。

（２）地質学的情報

地質学的情報は、シミュレーション対象地域の地層の特性を記述するものである。このモデルには、地山の地層の他、その地域の鉱床や人工的な構造物の位置や規模についてのデータが含まれている必要がある。地山の地層の特徴は、厚さ、深さ、水平方向の広がり、熱伝導率、粒間地下水流、収着の性質、地山と地山に含まれる地下水、および処分した廃棄体のパッケージの間での化学的相互作用の程度などのパラメータに関して記述される必要がある。

（３）物質移行情報（性能評価情報）

汚染情報は、処分システムからの漏出核種の拡散を予測するために使われる情報である。それは、主に個別の物理化学現象仮定の記述をモデルにより開発されたシミュレーション・コードと関連するデータである。この種のモデルは、処分システムのために岩盤、およびその間隙流体に生じる外乱による水文学的、熱・力学的な力の間の相互作用を考慮に入れたものでなければならない。

各種の力の間における相互作用の内容を理解するためには、まず熱的な力の発生源から行うことが一般的操作として考えられる。熱的な力は、処分される廃棄体から発生する大量の熱によって生じる。この熱入力は、相当の期間続き、地下水の対流ならびに母岩の膨張を引き起こす。比較的わずかな温度上昇が大きな体積で長期にわたって起きた場合には、かなりの対流を生じることになる可能性がある。

また、処分システムの建設においては、地盤に大きな空洞を作ることで当初の応力分布を変えることになる。この応力分布の変化によって、母岩に力学的な変形が生じ、既存の割れ目が開いたり、閉じたりする現象が考えられる。このような割れ目パターンの変化は、流体の流れや地盤中の溶質の輸送に大きな影響を与える。

更に、廃棄物の拡散ならびに関係する水文学的、熱力学的な相互作用について的確な予測をするためには、人工バリアと天然バリアの両方の空間的スケールを対象とした別々のモデルが必要となる。

ア．人工バリア

人工バリアは、次の３つの事項に関する廃棄物パッケージの達成能力のシミュレーションを行うものである。

- 長期にわたって放射性核種を封じ込めること
- 地山への放射性核種の漏出速度を低減すること
- 廃棄体への水の接近を最小限に抑えること

人工バリアには、処分場の廃棄物処分孔に収容するすべてのもの（すなわち、固化体、キャニスタ、オーバパック、緩衝材、埋戻し材等）を含めるものとする。人工バリアは、オーバパック設計寿命、当初からの欠陥よるオーバパックの早期破損率、破損したオーバパックからの部分浸出率、オーバパックの破損から浸出期間の最後までの時間などのパラメータを使ってオーバパックの物理的性質の特徴を記述する。

イ．天然バリア

天然バリアに対しては、地下水と人工バリアの間の相互作用のほか、それに続いて核種の地下水中への移行が起きる場合の様子をモデル化することが要求される。天然バリアは、対象とする区域全域にわたって流れの経路と流速を特定するとともに、処分場に関係する水文学的、熱的および、力学的な力の相互作用を考慮に入れたものでなければならない。天然バリアで特に必要となるパラメータとしては、以下が考えられる。

- 処分システムの建設によって生じる岩盤の機械的変化
- 処分する廃棄物による熱負荷の増大
- 天然の動水勾配および透水率
- 地下水流の流速および水量
- 地下水のｐＨ、酸化電位、イオン強度、含有酸化剤、処分した廃棄物が関係する化学的な変化などの化学的性質。

ウ．生物圏情報

生物圏事象モデルは、処分システムの通常の操業にとっては外的な要因である。それは生物圏における事象の影響をシミュレーションするモデルである。例えば、地震、火山の噴火、大洪水、隕石の衝突、地図にない大きな割れ目、ボアホールを塞いだ栓の破損の他、爆弾の爆破、火災または処分サイトへの侵入などの人為的な事象を挙げることができる。

（4）基本対話機能要件

地層処分ヴァーチャル・エンジニアリング・システムの操作においては、利用目的に応じて主として2つの形式があると考えられる。第1に、多数の観衆にシミュレーション画像が提供され、観衆がフロアから「～ならばどうなる」式の質問を行いながらシミュレーションの経過の規定に参加する形式がある。第2の形式として、性能評価研究のための利用があり、研究者が現象の局所的および広域的な影響について評価する場合である。

これらの利用方法では、シミュレーション結果を視覚的な直観に訴える単純化された描写にまとめるコンピュータ・グラフィックスによって、対話方式で自由に操作できる大型のビジュアルな環境をサポートすることができるとともに、結果の詳細な分析と科学的な視覚化によって、あらかじめプログラムされた複雑なシミュレーション・シナリオの実行をサポートすることができるシステムのアーキテクチャ開発が必要である。

個別シナリオ等の評価においては、シミュレーションは場合により数時間に渡ることも考えられるため、必ずしも完全な対話速度で動作する必要はないと思われる。例えば、小スケールの結果を高い精度で得ることに研究者の関心がある場合は、シミュレーションは対話速度よりも、ゆっくりと動作して詳細な計算結果を作り出してもよく、その上で、その結果に対して研究者が対話方式で質問を与えるようにすることができるようにすることが必要である。これをアーカイブ処理と言う。

将来、テラフロップス級の超高速コンピュータの利用により、全面的な対話方式シミュレーションが可能なシステムを作り上げることができるのであれば、このような動作機能は不要と思われるかも知れないが、実際にはそうではない理由として互いに関連し合った2つの要因がある。

第1の要因は、モデルのデータ空間のイメージを捉えるのに必要な大量のデータを記録することは、たとえ95%以上のデータ圧縮ができたとしても、それによって記録されるシミュレーションの規模（すなわち持続時間）に厳しい制約が課される可能性があるということである。従って、実時間性が保証されないであろう。

第2の要因は、たとえデータ空間のイメージを保持するのに十分な2次記憶装置があったとしても、そのような装置の入出力（I／O）速度の遅さのために、2次記憶装置への転送がシステム性能における重大なボトルネックとなるということである。これも前記と同様に、実時間性を保証しない要因である。地層処分の性能評価問題に係わる詳細なシミュレーションを行う場合は、シミュレーション・システムが、「実時間」システム、すなわち外部からの割り込み、ないしはイベントに対してマイクロセカンド、ないしはミリセカンドの時間枠で応答するシステムとして動作する必要はないと考えられる。

対話機能に対しては、目立つ休止のない迅速な動作が要求されるが、本来的な要求としての「実時間」要件は存在しない。このように考えることによって、シミュレーション・システムのハードウェア、およびソフトウェアの「実時間」特性の指定にかかわる問題は、システム設計面から回避することができる。

以上の様に、目標とする地層処分ヴァーチャル・エンジニアリング・システムは、高度な対話機能を通じて、複数の独立したプログラムが、場合により協調して要求に答えるための処理を行うシステムとなると考えられる。そのようなシステムの動作は、従来の逐次的にコードを組み合わせて計算する手法と本質的に異なったものであると考えられる。

以上のようなシミュレーション手法（数値実験）は、複雑な現象の理解にとって非常に大切なものと考えられる。数値実験では、コンピュータは素過程として論理的に動いている。しかし、コンピュータ内で起こる現象を理解したり、定式化したりするとき必要になることは、それら論理の筋をすべて追うことではなく、コンピュータ内の現象を通して養われた新しい直観にもとづくことであると考えられる。

このとき最初に重視すべきことは、素過程での論理の正当性である。これを信じた後、今までの物理学的直観とは異なるかもしれない新しい直観を養う必要がある。たとえ、ある事実がわれわれの物理的直観に反したとしても、このような意味で「数学的事実」であれば、それを真理とみなすことは計算科学的方法と考えられる。数学は、それ独自の自律運動を遂げているのであるが、それとは別に地層処分性能評価のような、複雑な現象に対する数学的定式化をコンピュータの中で形式化することが計画するシステムに期待できる。

そのようなシミュレーションは、複数の独立したタスクとして動作するオブジェクトである“エージェント”の協調により実現すると考えることができる。従って、試作する対話システムは、マルチ・エージェントな環境に対応するユーザーインターフェイスとなる。

３．３．２　マルチ・エージェント・ユーザーインターフェイス

理想的な地層処分ヴァーチャル・エンジニアリング・システムの対話機能の基礎的考え方として、一個のユーザーインターフェイスを複数のエージェントの集まりとして構成する、マルチ・エージェント・ユーザーインターフェイスの制御について検討する。

マルチ・エージェント・ユーザーインターフェイスにおいては、個々のユーザーインターフェイス・エージェントは単独で動作し、ユーザとの対話が可能である。マルチ・エージェントを用いる利点は、各ユーザーインターフェイス・エージェントを個別にプログラミングできること、テストできることである。ユーザーインターフェイス・エージェントの協調は二通り必要である。

協調１：複数のユーザーインターフェイス・エージェントが一体となって一個のユーザーインターフェイスを構成するには、それらが協調する必要がある。

協調２：複数のユーザーインターフェイス・エージェントが同一のものに対するユーザインタフェースを提供しているとき、それらの表示の一貫性を保つためにも協調が必要である。

（１）マルチ・エージェントモデル

ユーザーインターフェイス管理システム（ウィンドウ・マネジャと呼ばれる事もある）とは、ユーザーインターフェイスを構築するためのソフトウェアシステムである。ユーザーインターフェイスの開発においては、このユーザーインターフェイス管理システムの機能を検討することが重要な課題となる。ユーザーインターフェイス管理システムは、ウィンドウシステムの上位に位置し、ウィンドウシステムの提供する入出力機能と、アプリケーションプログラム本体との間の橋渡しを行なう。ユーザーインターフェイス管理システムを用いて構築されたユーザインターフェイス部とアプリケーション本体とが合わさって、ウィンドウシステム上で動作するユーザーインターフェイスを構成する。

ユーザーインターフェイス管理システムの実装制御アルゴリズムに関する有力な考え方として、シ

ーハイムモデルがある（図３．３－１参照）。シーハイムモデルでは、対話部はプレゼンテーション部、対話制御部、アプリケーション・インターフェイスの３つの要素から構成される。各要素はコンパイラにおける字句解析部、構文解析部、意味解析部にそれぞれ対応する。ユーザの入力は、３つの構成要素によって順に解析され計算部への入力となる。逆に計算部の出力は逆順で，３つの構成要素によって処理されユーザへの出力となる。シーハイムモデルの問題点の一つは、ダイレクトマニピュレーションが実現困難なことである。

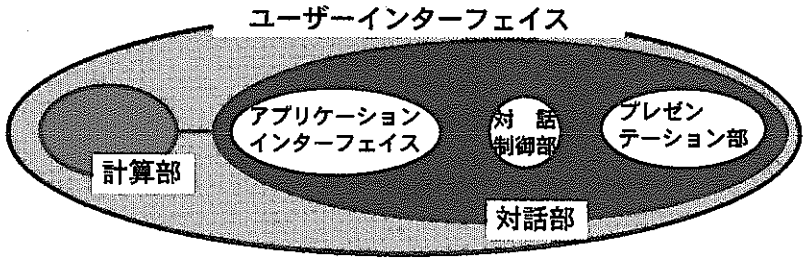

図３．３－１　シーハイムモデル

ダイレクトマニピュレーションを行なうには、対話部の末端であるプレゼンテーション部を管理する情報から、計算部の管理する情報まで、全ての構成要素の情報を必要とする。プレゼンテーション部と計算部との間で、何度も制御を行き来させれば不可能ではないが、現実的には実現困難である。

例えば、グラフィックス・エディタにおけるラバーバンディングはプレゼンテーション部単独で実現することはできず、計算部の助けによって、ラバーに囲まれた範囲の図形を特定させなければならない。同様に、視覚しながら計算を進めるには、マウス・カーソルの情報をプレゼンテーション部が取り込むとともに、計算部にコールバックをかけて結果を待たなければならない。そこで、同期問題が生じる。同じことは、性能評価に係わる個別コードをインフルエンス等に関連させて実行する場合も言える。

マルチ・エージェントモデルは、シーハイムモデルへの反省から提案された。機能の層別化によってユーザーインターフェイスを分解するシーハイムモデルと異なり、マルチ・エージェントモデルでは、画面領域の分担によってユーザーインターフェイスを分割する。

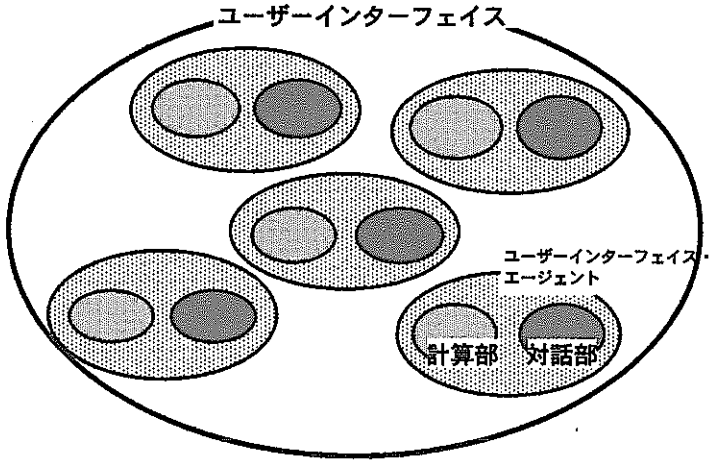

図３．３－２　マルチ・エージェントモデル

マルチ・エージェントモデルでは、一個のユーザーインターフェイスの占める画面領域は、いくつかの小領域（例：テキストサブウィンドウ）に分割される。各ユーザーインターフェイス・エージェントは、小領域の一つにおける対話操作を担当する。個々のユーザーインターフェイス・エージェントは単独で動作し、ユーザとの対話が可能である。これらのユーザーインターフェイス・エージェントが集まって領域全体での対話操作を提供する。直観的にいうと、シーハイムモデルが横割りであり、マルチ・エージェントモデルは縦割りである。個々のユーザーインターフェイス・エージェントは、さらに対話部と計算部に分割される。対話部は、そのユーザーインターフェイス・エージェントの担当領域における対話操作を提供する部分である。計算部は、アプリケーション全体のうちの一部であって、そのユーザーインターフェイス・エージェントが操作すべき部分である。

このように、対話部と計算部を組みにしたエージェントを考えることにより、従来型の同期関係に支配されたシミュレーションから、並列的に動作するマルチ・エージェントによる複雑な協調計算を行うアーキテクチュアを実装することが期待できる。

すなわち、ユーザーインターフェイス全体の計算部（アプリケーション全体）も、ユーザーインターフェイス・エージェントの計算部の集まりとみなすことができる。マルチ・エージェントモデルには次のような利点がある。

利点１：個々のユーザーインターフェイス・エージェントを独立に作成できる
　　個々のユーザーインターフェイス・エージェントを独立にプログラミングすると、対話操作を行なってテストすることができる。

利点２：ユーザーインターフェイス全体の構成を柔軟に変えられる
　　ユーザーインターフェイス・エージェントは他の部分と独立しているので、ユーザーインターフェイス・エージェントの追加や削除が容易である。

利点３：ダイレクトマニピュレーションが容易に実現できる

対話部と計算部の分離は、シーハイムモデルのように全体で一括して行なわれるのではなく、ユーザーインターフェイス・エージェント単位で無理なく行なわれる。あるユーザーインターフェイス・エージェントがダイレクトマニピュレーションを実現するのには、そのユーザーインターフェイス・エージェントの対話部と計算部の情報を用いればよい。

利点４：マルチスレッド・ダイアログが容易に実現できる

マルチスレッド・ダイアログとは、ある系列の操作の途中で、他の系列の操作を実行することである。例えば、文書を格納するファイル名をエディタで指定している途中で、他のウィンドウでファイル名の一覧を表示する操作を実行することである。

マルチ・エージェント・モデルでは、あるユーザーインターフェイス・エージェントが一つの系列の操作を行なっているのと並列に、他のユーザーインターフェイス・エージェントにもう一つの系列の操作を実行させることによって、マルチスレッド・ダイアログを自然に実現することができる。ただし、ユーザーインターフェイス・エージェントを並列に動作させるためには、並列オブジェクト指向での実装を要する。例えば、X11のXtなどのツールキットの提供するwidget（ボタン、メニュー、スクロールバーなど）は、ユーザーインターフェイス・エージェントと似ている点がある。どちらも画面上である範囲の領域を担当する。しかし、個々のwidgetが単独で動作しないという点において、widgetはユーザーインターフェイス・エージェントと決定的に異なる。従って、ツールキットでは上述の利点は得られない。

（２）ユーザーインターフェイス・エージェントの協調

ユーザーインターフェイス・エージェントが、それぞれ独立して動作するだけでは十分でなく、それらが協調して動作する必要がある。本研究では、必要となる協調を一体化のための協調と一貫性のための協調の二つに分ける。一体化のための協調とは、複数のユーザーインターフェイス・エージェントが一体となって一個のユーザーインターフェイスとして振舞うための協調である。

例えばスクロールバー（ユーザーインターフェイス・エージェント）とテキストウィンドウ（ユーザーインターフェイス・エージェント）からなるユーザーインターフェイスにおいては、スクロールバーでスクロールを行なうとテキストウィンドウでの表示範囲が変わる。

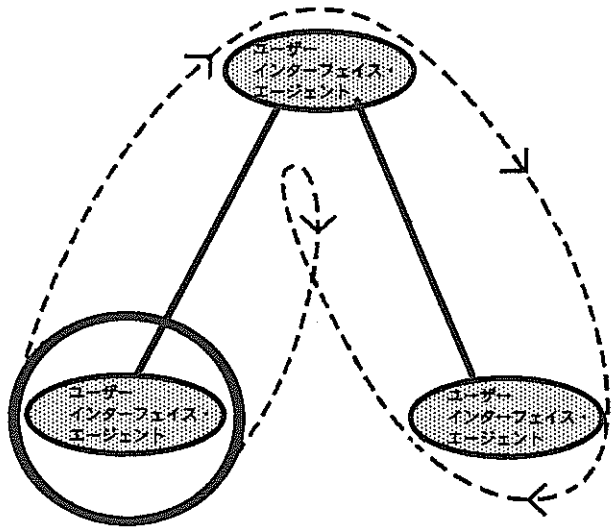

図３．３－３　階層構造による協調

一貫性のための協調とは、複数のユーザーインターフェイス・エージェントが、同一のものに対するユーザインターフェイスを提供しているとき、それらの表示を一貫させるための協調である。

すなわち、一個の計算部を共有する複数のユーザーインターフェイス・エージェントは、協調する必要がある。例えば、同一のテキストに対して２つのテキストウィンドウが存在するとき、一方で編集を行なうと、もう一方でも表示を行なわなければならない。

同様に、例えば、地球化学コードPHREEQUEの熱力学データベースを整理する対話機能設計を考えた場合、PHREEQUEは検証されたものを一つに計算部として扱い、同一の熱力学データベースについて、２つ以上のウィンドウを表示しながらデータ解析作業がすすめられることが考えられる。いずれにしろ、このような一貫性のためのエージェントの協調機能は、今後、システム開発のさまざまな局面で生じるであろう。

マルチ・エージェントを用いた従来の研究では、この二つの協調を実現するのに、階層構造による方法か逆階層構造による方法のいずれか一方のみを用いている。

（３）階層構造による実現方法

階層構造による方法では、ユーザーインターフェイス・エージェントを階層的に構成し、上位のユーザーインターフェイス・エージェントに下位のユーザーインターフェイス・エージェントを制御させる（図３．３－３参照）。

この方法では、一体化のための協調を容易に実現することができる。下位のユーザーインターフェイス・エージェントの挙動のうち、協調を必要とするものは、上位のユーザーインターフェイス・エージェントによって検出される。上位のユーザーインターフェイス・エージェントは、関連するユーザーインターフェイス・エージェントに通知し協調を行なわせる。

例えば、スクロールバー（ユーザーインターフェイス・エージェント）とテキストウィンドウ（ユーザーインターフェイス・エージェント）とを持つファイルエディタ（上位ユーザーインターフェイ

ス・エージェント）を考える。スクロールバーの挙動は、ファイルエディタによってテキストウィンドウに通知されスクロールを引き起こす。

各ユーザーインターフェイス・エージェントは、上位ユーザーインターフェイス・エージェントや兄弟ユーザーインターフェイス・エージェントに依存しない。上位や兄弟がすべて削除されても、そのユーザーインターフェイス・エージェントは単独で動作しうる。言い換えると、ユーザーインターフェイス・エージェントは、上位ユーザーインターフェイス・エージェントや兄弟ユーザーインターフェイス・エージェントからの入力を必要とせず、これらに対する出力も行なわない。上述の例では、テキストウィンドウ（兄弟ユーザーインターフェイス・エージェント）とファイルエディタ（上位ユーザーインターフェイス・エージェント）が削除されても、スクロールバー（ユーザーインターフェイス・エージェント）が単独で動作が可能である。

階層構造による方法の欠点は、一貫性のための協調に向かないことである。一貫性を保つべき複数のユーザーインターフェイス・エージェントは、一個の計算部を共有することによってのみ関連づけられている。したがって、協調を制御すべき上位のユーザーインターフェイス・エージェントは存在しない。一貫性の保持の目的で上位のユーザーインターフェイス・エージェントを設けることによって、この問題を解決することは可能ではある。この上位ユーザーインターフェイス・エージェントは、一貫性を保つべきユーザーインターフェイス・エージェントすべての上位に位置付けられる。

いずれかのユーザーインターフェイス・エージェントが、計算部の状態を変更するような挙動を行なうと、上位ユーザーインターフェイス・エージェントが、他のユーザーインターフェイス・エージェントに通知して一貫性を保たせる。

しかし、この解決方法で一貫性を保つためには、どのようなユーザーインターフェイス・エージェントにより、どんな操作が行なわれるのかをあらかじめ想定しておく必要がある。したがって、新たなユーザーインターフェイス・エージェントが追加されるたびに一貫性を保たせるための変更を上位ユーザーインターフェイス・エージェントに行わなければならない。この作業は、煩雑であるため、実際には行なわれないことが多い。またユーザーインターフェイス・エージェント以外のプログラムによって計算部の状態が変更されると一貫性は保たれない。

（4）逆階層構造による実現方法

逆階層構造による方法では、計算部を共有する複数のユーザーインターフェイス・エージェントの協調を行なう。逆階層構造とは、階層構造の逆転したものであり、共有される計算部と、各ユーザーインターフェイス・エージェントの対話部とが形作る構造である（図３．３－４参照）。

この方法は、一貫性のための協調を容易に実現することができる。あるユーザーインターフェイス・エージェントの対話部が、ユーザとの対話の結果、計算部の状態を変更すると、どのユーザーインターフェイス・エージェントの対話部も、この変更を検出して再表示を行なう。

例えば、一個のテキスト（計算部）を共有する複数のテキストウィンドウ（対話部）を考える。あるテキストウィンドウからテキストを編集すると、全てのテキストウィンドウがこの変化を検出して再表示を行なう。

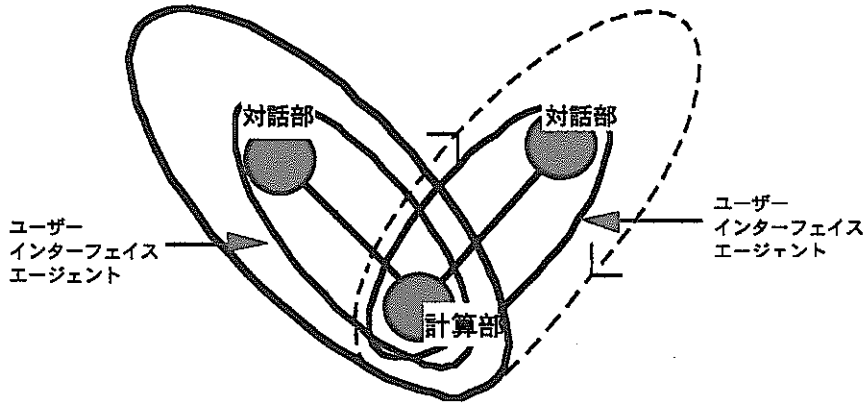

図３．３－４　逆階層構造による協調

この方法の欠点は、一体化のための協調に向かないことである。一体となって振舞うべき複数のユーザーインターフェイス・エージェントは、上位のユーザーインターフェイス・エージェントを構成することによってのみ関連づけられている。協調の中心となるべき共有される計算部は存在しない。

一つの解決方法は、全てのユーザーインターフェイス・エージェントの状態を表すデータを設け、各ユーザーインターフェイス・エージェントの計算部として共有させることである。その結果、各ユーザーインターフェイス・エージェントの挙動は、この計算部の状態変化として現れる。他のユーザーインターフェイス・エージェントは、状態変化を検出し、必要な協調を行なう。この計算部は、アプリケーションの一部ではなく、他ユーザーインターフェイス・エージェントについての情報であるので厳密には計算部とはいえない。この解決方法では、ユーザーインターフェイス・エージェントの独立性が損なわれる。実際、各ユーザーインターフェイス・エージェントの計算部が、他ユーザーインターフェイス・エージェントの情報を多く含んでいる。

３．３．３　逆階層構造十階層構造によるユーザーインターフェイス構築

従来方式の問題点を解決するため、逆階層構造と階層構造を併用したユーザーインターフェイス構築方式が良いと考えられる（図３．３－５参照）。一個のユーザーインターフェイスは、複数のユーザーインターフェイス・エージェントの階層構造として構成される。これらの統合化のための協調は、階層構造中の上位ユーザーインターフェイス・エージェントによって行なわれる。

また、異なるユーザーインターフェイスに属するユーザーインターフェイス・エージェントが、計算部を共有して逆階層構造をなすことがある。逆階層構造中の対話部は、計算部の状態変化を検出し一貫性のための協調を行なう。

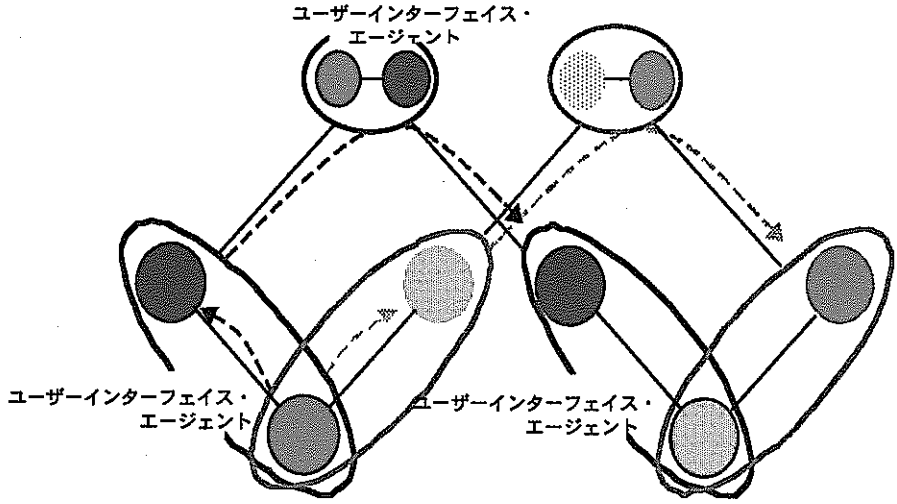

図３．３－５　逆階層構造十階層構造による協調

（１）ブラウザ

ブラウザとは、一般にプログラム作成、分類、検索を支援するツールである。ブラウザは、カテゴリ・リスト・ビュー、クラス・リスト・ビュー、プロトコル・リスト・ビュー、セレクタ・リスト・ビュー、テキスト・ビューの５つのサブビューで構成される（図３．３－６参照）。各サブビューでは、それぞれ表示されているクラスやメソッドに対する追加や変更、削除、ファイルへの書き出しなどの操作機能がメニューコマンドとして提供されている。

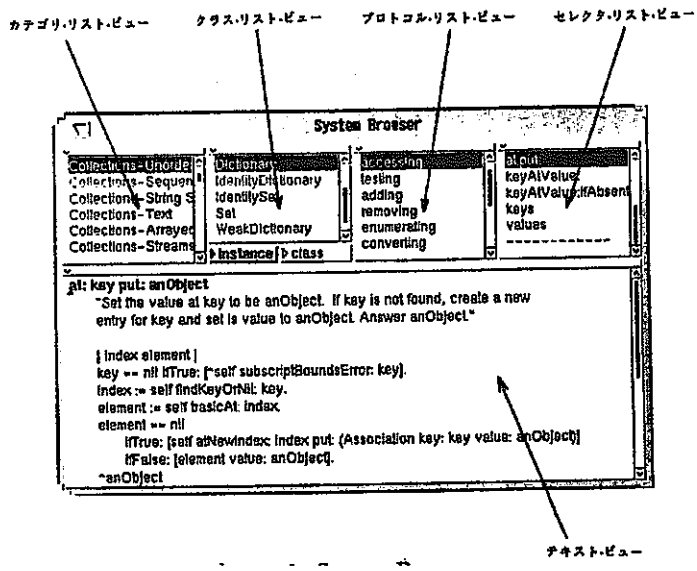

図３．３－６　ブラウザ

各サブビューでは、それぞれ表示されているクラスやメソッドに対する追加や変更、削除、ファイルへの書き出しなどの操作機能がメニューコマンドとして提供されている。

各サブビューは、一体化のための協調を行なう。すなわち、各リストビューに表示されている要素を順次選択していくと、選択されたカテゴリや、クラスに属するクラスのリストや、プロトコルのリストが隣のサブビューに表示され、テキスト・ビューには、クラス定義やメソッド定義などが表示される。

ブラウザが単独で使用される場合には、上記の協調により、各サブビューは一体となってブラウザの機能を実現する。しかし、複数のブラウザが同時に存在するとき、これらは同一のプログラム空間をアクセスするにもかかわらず、一貫性のための協調は行なわれない。

例えば、あるブラウザからクラスの名前を変更しても、他のブラウザは元の名前を表示したままである。また、ファイルからクラス定義を読み込むなどしてクラスを再定義したときも、ブラウザは新しいクラス定義を表示しない。updateコマンドが実行されるまでは、古いクラス定義を表示したままである。

（2）ブラウザの従来の実装（逆階層構造）

従来のブラウザの実装は、逆階層構造による構築方式に基づいている。各サブビューは、ビューオブジェクトとコントローラ・オブジェクトのペアからなるユーザーインターフェイス・エージェントによって実現され、それらが一個のモデル・オブジェクト（クラスブラウザのインスタンス）を共有することによって逆階層構造を形成する（図３．３－７参照）。

このとき、各ビューオブジェクトとコントローラオブジェクトには、汎用の部品として予め用意されているものが用いられる。各サブビューのメニューコマンド等のブラウザに固有の機能は、すべてモデルオブジェクトの機能として記述され、モデルオブジェクトによって実行される。一体化のための協調は、サブビューが共有するモデルオブジェクトを介して行なわれる。

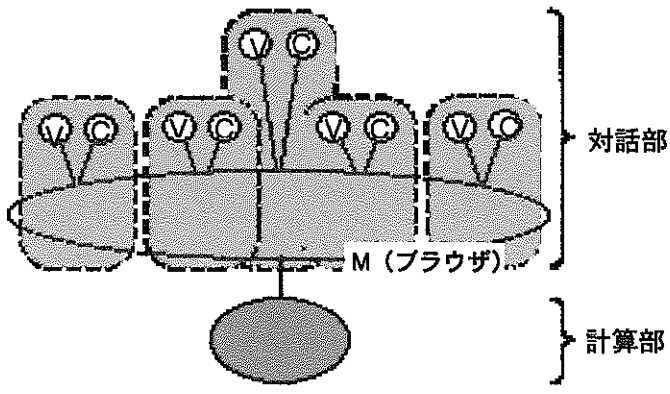

図３．３－７　従来のブラウザ

例えば、クラス・リスト・ビューにおいて、あるクラスを選択すると、クラス・リスト・ビューはモデルオブジェクトに選択されたクラスの名前を伝える。モデルオブジェクトは、これを一旦インス

タンス変数classNameに格納する。

あわせて、メッセージchanged:を自分自身に対して送り、dependencyfieldの機構を通じて、メッセージupdate:を全てのサブビューにブロードキャストする。このupdate:に応えて、プロトコル・リスト・ビューとテキスト・ビューは、表示すべきプロトコルとテキストをモデルオブジェクトから新たに求め、それらを表示する。

この実装方法は、統合化のための協調を本来は不向きな逆階層構造を用いて行なったものである。このようにして作成されたブラウザには、次の問題点がある。

問題１：サブビューが独立したユーザーインターフェイス・エージェントでなくなる。
ブラウザの機能が全てクラスブラウザに一括して記述される。そこでは各サブビューの記述が複雑に交錯し、独立していない。すなわち、サブビューはもはやユーザーインターフェイス・エージェントではない。したがって、マルチ・エージェントモデルの利点は成立しない。例えばクラス・リスト・ビューを他のサブビューと独立に作成することはできない。

問題２：モデルが巨大になり理解し難い。
ブラウザのほとんど全ての機能がモデルオブジェクトの機能として実装されるため、モデルオブジェクトは必然的に巨大になり理解し難い。

（３）逆階層構造＋階層構造による実装

逆階層構造と階層構造とを用いたブラウザの実装を図３．３－８に示す。ブラウザの各サブビューは単独で動作するユーザーインターフェイス・エージェントであり、ブラウザ全体もユーザーインターフェイス・エージェントである。

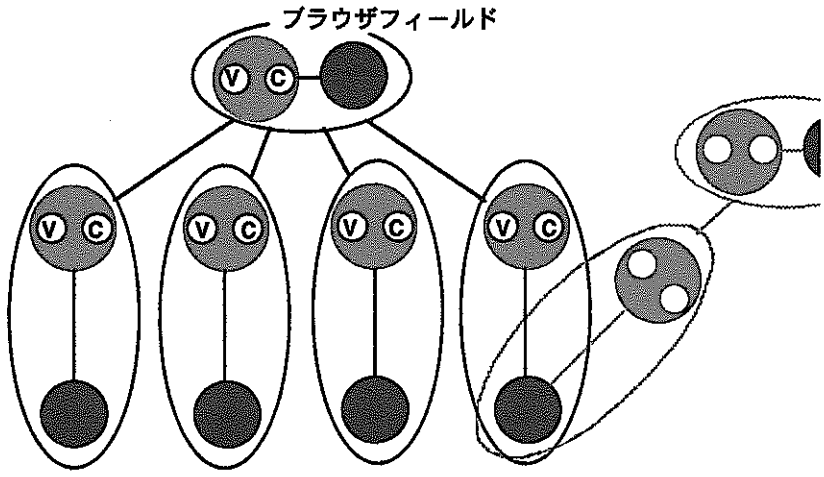

図３．３－８　ブラウザの協調と実装

ブラウザ全体である（全体）ユーザーインターフェイス・エージェントと、サブビューである（部分）ユーザーインターフェイス・エージェントとは階層構造をなす。統合化のための協調は、全体ユーザーインターフェイス・エージェントによって行なわれる。

例えば、クラス・リスト・ビューにおけるクラス選択は、全体ユーザーインターフェイス・エージェントによって監視され、プロトコル・リスト・ビューとテキスト・ビューに通知される。プロトコル・リスト・ビューは、選択されたクラスの持つメソッド・プロトコルを表示し、テキスト・ビューは、そのクラスの定義を表示する。複数のブラウザがあるとき、それらのサブビューであるユーザーインターフェイス・エージェントは、計算部を共有することによって逆階層構造をなす。各ユーザーインターフェイス・エージェントは、計算部の変化を監視し、必要に応じて再表示を行なう（一貫性のための協調）。

例えば、２つのブラウザのクラス・リスト・ビューは、クラスを分類したオブジェクトを共有する。このオブジェクトへのクラスの登録をクラス・リスト・ビューは監視し、登録が行なわれるとクラス・リストの再表示を行なう。このブラウザは、従来のブラウザと異なり、複数のブラウザが一貫性を保つ。

例えば、あるブラウザからクラスの名称を変更すると、そのクラスを表示していたすべてのブラウザは表示をやり直す。また、ファイルからクラス定義を読み込んだときや、他のユーザーインターフェイスでクラスを定義したときも同様である。

３．３．４　逆階層構造十階層構造によるユーザーインターフェイスの構築手順

逆階層構造十階層構造に基づくユーザーインターフェイスを実際に構築するための手順を検討する。前者と同様にブラウザを例とすると構築の手順は下記の通りとなる。

手順１：ユーザーインターフェイス・エージェントの作成
　各サブビューをユーザーインターフェイス・エージェントとしてクラスの定義を行い、それぞれのユーザーインターフェイス・エージェントが単独で実現できるコマンド群をメソッドとして定義する。

手順２：一貫性のための協調の実現
　計算部を共有することによって逆階層構造をなす複数のユーザーインターフェイス・エージェントに、一貫性のための協調を実現するメソッドを定義する。

手順３：統合化のための協調の実現
　全体ユーザーインターフェイス・エージェントのクラス定義を行う。この全体ユーザーインターフェイス・エージェントが持つインスタンス変数には、各サブビューを部分ユーザーインターフェイス・エージェントとして格納する。これにより、階層構造が構築される。全体ユーザーインターフェイス・エージェントには、一体化のための協調を実現するためのメソッドの定義を行う。

（１）ユーザーインターフェイス・エージェントの作成

ブラウザのサブビューをそれぞれユーザーインターフェイス・エージェントとして実現する。各ユ

ーザーインターフェイス・エージェントは、クラスCategoryListView、ClassListView、ProtocolListView、SelectorUsrView、BrowserTextViewとして定義する。例として、図３．３－９にClassListViewのクラス定義を示す。次に各サブビュー・ユーザーインターフェイス・エージェントに対しメソッドの定義を行う。これはポップアップメニューに表示される各種コマンドの機能やリスト・ビューにおけるリスト要素の選択の機能を実現するためのものである。このような機能は次の３種に大別できる。

```
ListView2 sub class : #ClassListView
   instanceVariableNames: 'organization category className meta systemDictionary'
   classVariableNames: 'ClassMenu'    クラス辞書の格納
   poolDictionaries: ''
   category: 'New Browser'!
```

図３．３－９　クラス・リスト・ビューの定義

```
removeClass
   ...
   do: [nonMetaClass := self nonMetaClass.
        (self confrm:
          'Are you certain that you want to remove the class ', nonMetaClass name, ' ?')
            ifTrue :              ユーザーへの確認
              [nonMetaClass subclasses isEmpty not       サブクラス有無の確認
                 ifTrue: [self notify: nonMetaClass name , ' has subclasses'].
              nonMetaClass removeFromSystem]] !      クラス削除の実行
```

図３．３－１０　クラス削除の実行メソッド

（a - 1）単独のビューで実現可能なもの。計算部に影響を与えない。

（a - 2）単独のビューで実現可能なもの。計算部に影響を与え、それに伴ってビューの表示の更新が必要となる（一貫性のための協調を必要とするもの）。

（b）　複数のサブビューの協調によってはじめて実現可能となるもの（一体化のための協調を必要とするもの）。

本ステップでは（a - 1）、（a - 2）の機能を実現するためのメソッドを定義する。ただし、（a - 2）については表示の更新のための記述は行わない。表示の更新の記述は、次ステップの一貫性のための協調の実現において与える。クラス・リスト・ビューを例にとれば、file out（外部ファイルへの書き出し）、print out（プリンタへの出力）などが（a - 1）に、remove（クラスの削除）、rename（クラス名の変更）が（a - 2）に、find method（メソッドの探索）、comment（コメント文の参照）などが（b）にそれぞれ属する。（a - 2）の例として、図３．３－１０にremoveコマンド（クラスの削除）を実行するremoveClassなるメソッドの定義を示す。表示の更新の記述は存在せず、クラスの削除を実行することによって計算部の状態（systemDictionaryの内容）を変更するだけのメソッドとな

っていることが分かる。

（2）一貫性のための協調の実現

複数のクラス・リスト・ビューは、計算部としてsystemDictionaryを共有することにより、逆階層構造をなす。本ステップでは、（a - 2）に属するコマンドの実行によって必要となる表示の更新についての記述を行う。この記述により、逆階層構造をなしているユーザーインターフェイス・エージェント間に一貫性のための協調を実現する。

各サブ・ビューは、計算部の状態が変更されたとき、自らが自動的に再表示を行うものとする。こうすることによって、いかなるユーザーインターフェイス・エージェントが、計算部の状態を変更しようとも再表示がかかる。上記の設計方針を実現するためにオブジェクト指向モデルにデーモンの概念を導入する。

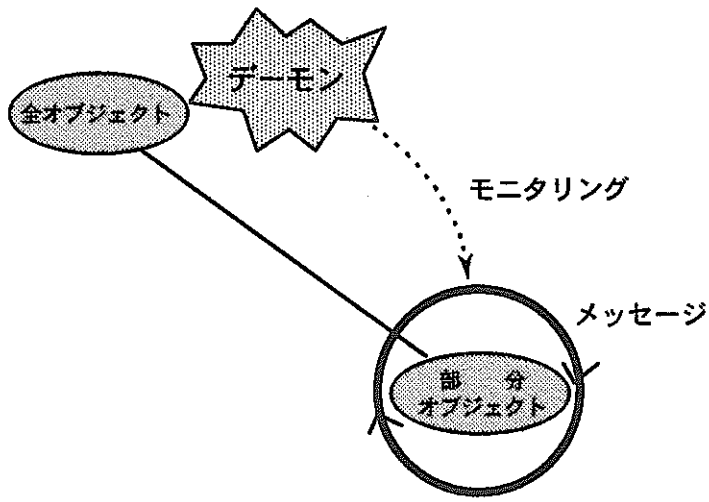

図３．３－１１　デーモンの概念図

```
!ClassListView daemonMethodAt: 'systemDictionary' monitoring: #removeClassNamed:!
systemDictionaryRemoveClassNamed: aSymbol
  (Smalltalk at: aSymbol) category = category
      ifTrue:
        [callNextMethod.        本来のメソッドの実行
        self updateForListOntemsChange]    表示の更新
      ifFalse: [callNextMethod].!!
```

図３．３－１２　クラスの削除を監視するデーモンメソッド

図３．３－１１にデーモンの概念図を示す。全体オブジェクトに定義されたデーモンは、部分オブ

ジェクトへのメッセージを監視し、メッセージの到着を確認すると、予め登録されたメソッドを起動する。各サブビューには、共有する計算部を監視するデーモンメソッドが定義される。計算部へ状態の変更を行うメッセージが到着すると、このデーモンメソッドが発火して、サブビューの表示を更新する。このデーモンの機能によって、一貫性のための協調が実現されている。

図３．３－１２は、クラス・リスト・ビューに定義されたデーモンメソッドの一例である。これは、計算部であるsystemDictionaryを監視して、クラスの削除（systemDictionaryにremoveClassNamed：なるメッセージが送信されると）が行われると自らの表示の更新を行うものである。

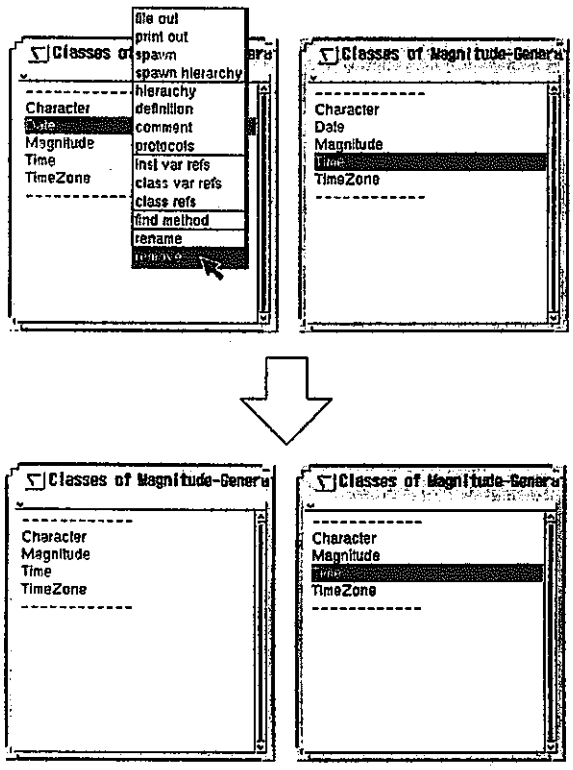

図３．３－１３　サブビュー間の一貫性のための協調

図３．３－１３に、クラス・リスト・ビューを単独で２つ開いた状態で、クラスの削除を行ったときの協調の様子を示す。左側のクラス・リスト・ビューにおいて実行されたクラスの削除に伴ない、右側のクラス・リスト・ビューの表示も更新されていることが分かる。図３．３－１４は図３．３－１３の協調のメカニズムを模式的に示したものである。

ユーザーが、ポップアップメニューからremove（クラスの削除）コマンドを指定すると、クラス・リスト・ビューは、removeClass（図３．３－１０参照）なるメッセージを受け取る（図３．３－１４中の（1））。removeClassを受け取ったクラス・リスト・ビューは、その後systemDictionaryを書き換えるためにremoveClassNamed：なるメッセージを送信する（図中（2））。双方のクラス・リスト・ビューに定義されているデーモン・メソッドsystemDictionaryRemoveClassNamed：（図３．３－１２参照）はこのメッセージを確認すると表示の更新のためのメッセージ（updateForListOfItemsChange）

を送信し（図中（3））協調が実現する。

（3）統合化のための協調の実現

ここでは、一体化のための協調（（b）に属する機能）を記述する。これらの記述を行うための場としてBrowserFieldなるクラスを新たに定義する。図３．３－１５はBrowserFieldのクラス定義である。各サブビューを格納するためのインスタンス変数を持つことが分かる。

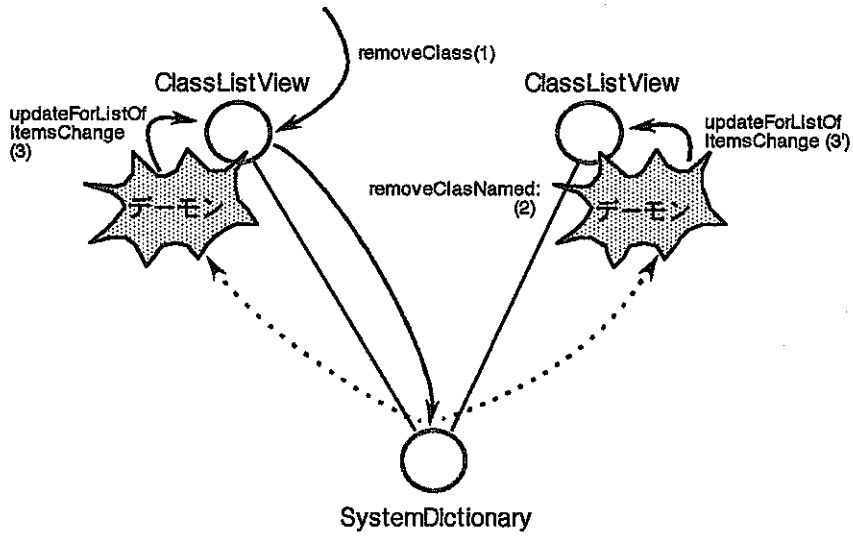

図３．３－１４　サブビュー間の一貫性のための協調の模式図

```
CompositeView subclass : #BfowserField
  insanceVariableNamcs : 'categoryListView classListView protocolListView
                          selectofListView metaSwitch codeView'
    classVariableNames:'
    poolDictionaries:'
    category:'New Browser'!
```

図３．３－１５　BrowserFieldの例

これは、BrowserFieldが全体ユーザーインターフェイス・エージェント、各サブビューが部分ユーザーインターフェイス・エージェントとなって階層構造が成立することを意味する。協調の記述は、全体ユーザーインターフェイス・エージェントであるBrowserFieldにデーモンメソッドとして定義する。

これらのデーモンメソッドは、インスタンス変数に格納された部分ユーザーインターフェイス・エージェントであるサブビューを監視する。協調を必要とする状態を引き起こすメッセージが、このサブビューに到着すると、このデーモンメソッドが他のサブビューに対して協調の指示を与える。

図３．３－１６に、BrowserFieldに定義されクラス・リスト・ビューを監視するデーモンメソッドの例を示す。これは、ユーザーによってクラス・リスト・ビュー中でリスト要素の選択が行われると（classListViewにselectItem：が送信されると）起動され、プロトコル・リスト・ビューとブラウザ・テキスト・ビューに協調の指示を与える。

クラス・リスト・ビューにおいてクラス要素の選択を行ったときのSystem Browserの動きを図３．３－１７に示す。プロトコル・リスト・ビューとブラウザ・テキスト・ビューが協調し表示が変更されていることが分かる。

```
!BrowserField daemonMethodAt: 'classListView' monitoring: #selectItem:!
classListViewSelectItem: className
    callNextMethod.
    codeView className: className.
    className isNil & categoryListView selectedItem notNil
      ifTrue: [codeView textMode: #classDefinition]
      ifFalse: [codeView classMode iftrue: [codeView updateForTextChange]].
protocolListView className: className.!!
```

図３．３－１６　クラス選択を監視するデーモンメソッド

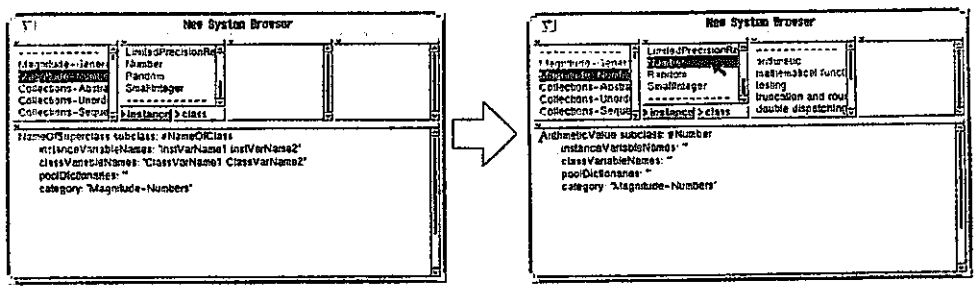

図３．３－１７　サブビュー１体化のための協調

図３．３－１８に図３．３－１７の協調のメカニズムを模式的に示す。クラス・リスト・ビューにselectItem：が送信されると（図３．３－１８中の（1））、BrowserField後定義されたデーモンメソッドclassListViewSelectItem：（図３．３－１５参照）が起動され、プロトコル・リスト・ビューとブラウザ・テキスト・ビューに表示の変更を指示するメッセージ（className：、updateForTextChange）をそれぞれ送信する（図中（2）、（3））。

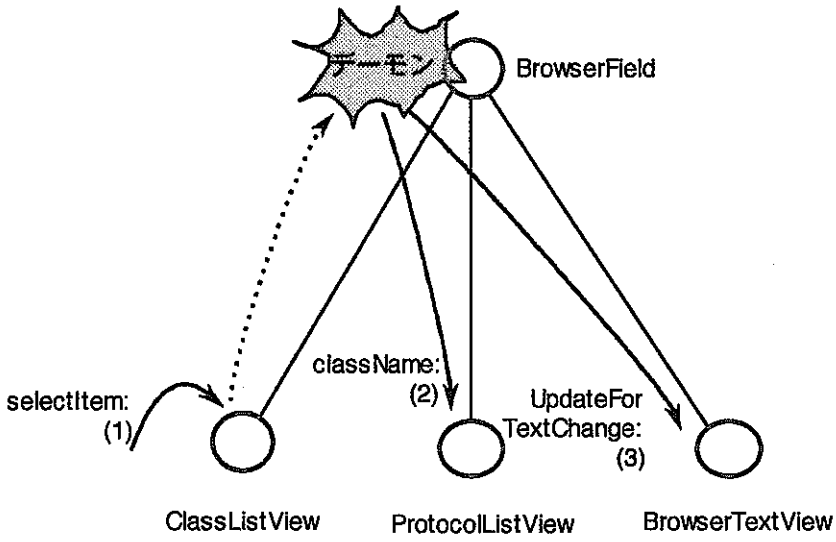

図３．３－１８　サブビュー１体化のための協調の模式図

（４）逆階層構造十階層構造によるブラウザの動作

本項では、前項までの手順で完成したブラウザの動作を確認するために、２つのブラウザを開き、クラスの削除を例としてその協調の仕方をみる。２つのブラウザを上下に並べ、上方に位置するブラウザのクラス・リスト・ビューで、ポップアップメニューからremoveコマンドを実行したときの動作を図３．３－１９に示す。

下方のブラウザのクラス・リスト・ビューにおいても表示は更新され、さらに上下のブラウザ共にプロトコル・リスト・ビュー、ブラウザ・テキスト・ビューとの協調が実現されている。

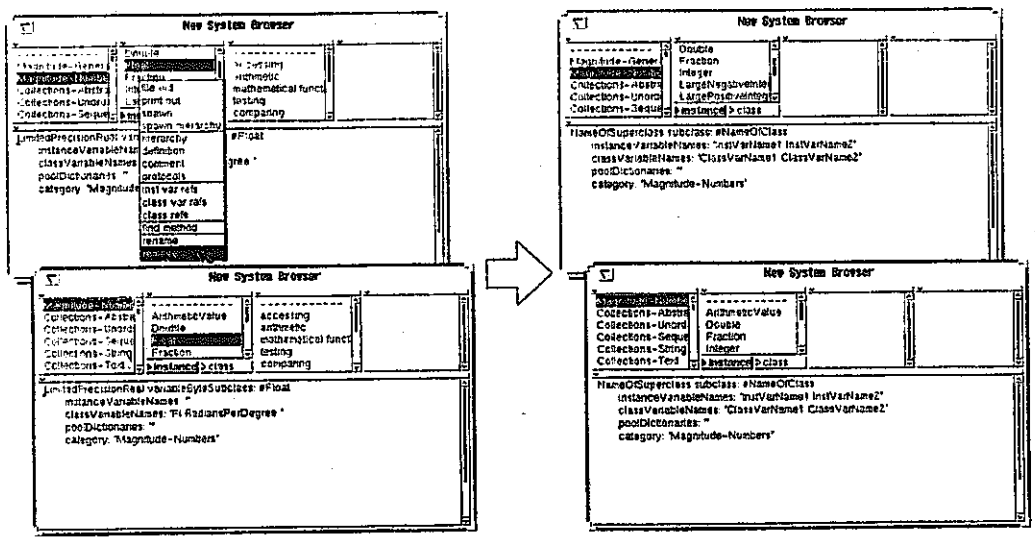

図３．３－１９　複数ブラウザ間の協調

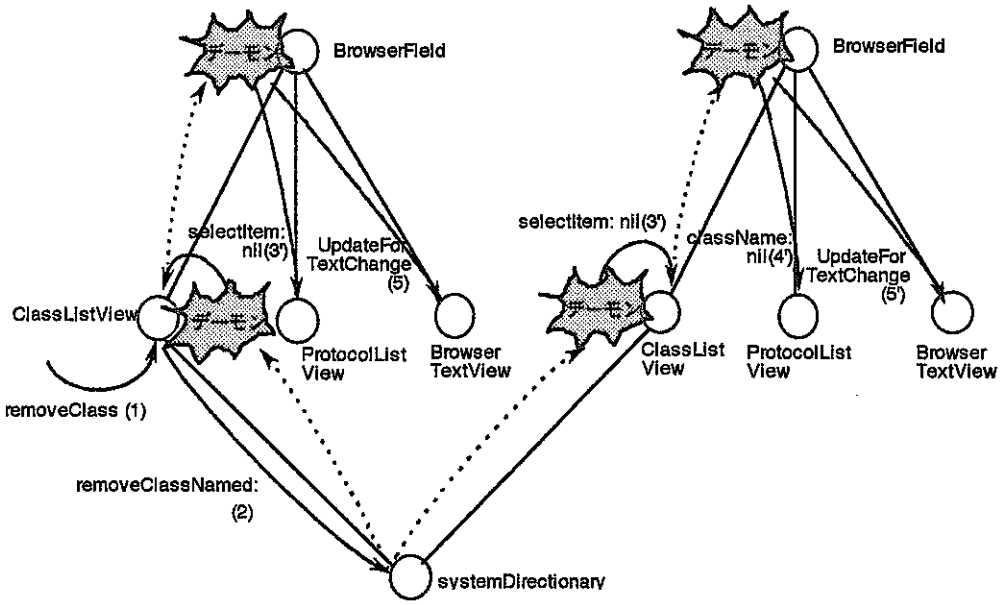

図３．３－２０　複数ブラウザ間の協調の模式図

図３．３－２０に図３．３－１９の協調のメカニズムを模式的に示す。まず、図３．３－１８と同様に、ユーザーの操作を受けたブラウザのクラス・リスト・ビューが、removeClass（図３．３－１０参照）なるメッセージを受け取り（図３．３－２０中の（1））、引き続いて計算部であるsystemDictionaryを書き換えるためにremoveClassNamed：なるメッセージを送信する（図中（2））。双方のブラウザのクラス・リスト・ビューに定義されているデーモンメソッド（図３．３－１２参照）は、このメッセージを確認すると表示の更新のためのメッセージ（updateForUsrOfItemsChange）を自らに送信し、ブラウザ間の協調が実現する。

次に、表示更新メソッドupdateForListOfItemsChangeの中でselectItem：なるメッセージが呼び出される（図中（3）（3'））。その後は図３．３－１４と同様である。すなわち、双方のブラウザ中で、クラス・リスト・ビューにselectItem：が送信されると、BrowserFieldに定義されたデーモン・メソッド（図３．３－１５参照）が起動され、プロトコル・リスト・ビューとブラウザ・テキスト・ビューに表示の変更を指示するメッセージ（className：、updateForTextChange）を送信する（図中（4）（4'）（5）（5'））。これによって、ブラウザ内の協調が実現する。

３．３．５　逆階層構造＋階層構造方式の利点

逆階層構造＋階層構造方式の応用には、以下の利点を期待することができる。

利点１：複雑なユーザーインターフェイスの一貫性を保つことができる

今回作成したブラウザは、同一のシステム環境に複数存在しても、完全に一貫性を保つ。計算部を共有するユーザーインターフェイスの一貫性をとる例は、従来いくつか報告されているが、ほとんどは、テキストなどの簡単な計算部をもつユーザーインターフェイス

に限られている。Smaltalkのクラス／メソッド定義のように複雑な計算部をもち、ブラウザのように複雑な機能をもつユーザーインターフェイスにおいて一貫性が実現された例はなかったと思われる。

利点２：想定したユーザーインターフェイス以外のものによって、計算部の状態が変更されたときもユーザーインターフェイスは一貫性を保つことができる

今回作成したブラウザが開かれている状態でファイルからクラス定義を読み込むと、ブラウザの表示は自動的に更新される。各ユーザーインターフェイス・エージェントは、クラス定義の変更を検出するのであり、他のユーザーインターフェイス・エージェントには依存しないのがその理由である。ブラウザの作成時には、ファイルからのクラス定義の読み込みを考慮していなかったにもかかわらず一貫性は保たれている。

利点３：各ユーザーインターフェイス・エージェントを独立に作成し実行できる

この利点はマルチ・エージェントモデル自体のものである。今回のブラウザの例では、クラス・リスト・ビューなどのサブビューをそれぞれ完全に独立して作成し、動作確認を行なうことができた。従来のブラウザの構成では、常にブラウザ全体の動作を考慮しなければ、作成することも動作確認を行なうこともできない。

利点４：各ユーザーインターフェイス・エージェントの記述が適度な大きさである

記述はユーザーインターフェイス・エージェントに分散して行なわれ、各ユーザーインターフェイス・エージェントの記述量は数百行程度である。ユーザーインターフェイス全体を理解するのには、これらのユーザーインターフェイス・エージェントの記述をそれぞれ独立に理解すればよい。ほとんど全ての記述が、クラスブラウザに集中していた従来の実装と比べ、プログラムの理解が大きく容易になる。

３．３．６　協調計算における逆階層構造十階層構造の導入について

複数のサブシステムが何らかの形で相互作用を及ぼしながら計算を進めて行くシステムの構成は、凡そ以下の２通りの手法に分類できる。

手法１：元来１つの問題を複数のサブシステムへと分割し、作業の分散化を図るタイプ。トップダウン的な設計が行なわれるため、どのようなサブシステムによってシステムが構成されるかは固定されている。また、各サブシステムの作業結果を元々の問題に対する作業結果へと統合する必要があるため、すべてのサブシステムの運用、制御を司りシステム全体の整合性を実現する集中管理ノードの存在が前提となる。

手法２：基本的に独立した処理を行う自律的なエージェントが、必要に応じた情報のやり取りを行い、それによってシステム全体に協調的動作を実現しようとするタイプ。通常、すべての処理を一元的に管理するエージェントは存在せず、システムを構成するエージェントも予め特定することはできない。

このように、手法１のタイプにおいては、本来１つのシステムで解かれている問題をいかに複数のサブシステムへと分割するかが、その興味の中心であるのに対して、手法２のタイプでは、複数のエ

ージェントをいかに協調させるかに重点が置かれる。従って、システムの動作としては、共に複数のサブシステムで並列（並行的）に処理が行われるのであるが、問題解決の方向性は逆である。

手法１のタイプは、いわゆる並列型システムと呼ばれるものである。このタイプは、集中管理ノードの存在によって、スケジューリング、および解の最適化が期待できるシステム構成である。しかし一方では、新たなサブシステムをシステムに追加するような場合に、集中管理ノードの変更も同時に必要となるので、柔軟性、拡張性の点では不利である。手法１のタイプのシステム構成に対して与えるべきシステムの構造は、本質的に階層構造である。

ある１つの集中管理ノードは、その管理下にある複数のサブシステムに対し、上位の階層に位置し、そのような集中管理ノードをさらに集中管理するノードは、その上位に位置する。このようにして階層構造が成立する。この構造によって、手法１のタイプのシステム構成に対して、確実な管理、制御が可能なシステムが実現できる。

本研究では、１つのユーザーインターフェイスを、いくつかのユーザーインターフェイス・エージェントへと分割した。この分割は、手法１のタイプのシステム構成である。したがって、一個のユーザーインターフェイスをまとめる必要が生じ、集中管理を行なう全体ユーザーインターフェイス・エージェント（BrowserField）を新たに作成することによって階層構造による統合化のための連動を行なった。この場合、ブラウザに対し、新たなユーザーインターフェイス・エージェントを追加することはないので、階層構造による連動が適切である。

しかし、集中管理は通常の集中管理とは異なる。通常の集中管理では、サブシステムは上位に隷属する。すなわち、入力は上位から与えられ、結果も上位に対して出力する。これに対し、本研究の集中管理では、サブシステムは上位から独立である。

ユーザーインターフェイス・エージェントは、上位ユーザーインターフェイス・エージェントとの間の入出力のやりとりを必要としない。上位は必要に応じて介入するのみである。すなわち、各ユーザーインターフェイス・エージェントが、上位ユーザーインターフェイス・エージェントに全く依存しない。これは、マルチ・エージェントの指向性に沿った協調型の階層構造であるといえる。

一方、手法２のタイプは分散（協調）型システムと呼ばれるものである。このタイプは、自律的エージェントの相互作用によって、システム全体の協調が実現されるため、柔軟性、拡張性に優れる。しかし、集中管理エージェントが存在しないため、最適な制御を行なうことは困難である。手法２のタイプのシステム構成は、本質的に逆階層構造となる。

自律したエージェントが協調するためには、協調の媒介となる共有物が不可欠である。その共有物を介して複数のエージェントが逆階層構造をなすブラウザの例では、自律的なユーザーインターフェイス・エージェントであるサブビュー間の連動を実現した。これは（２）のタイプのシステム構成である。

したがって、計算部を共有物として逆階層構造とした。本研究では、各ユーザーインターフェイス・エージェントが、この計算部の状態を監視し、それに対応して自らの状態を変更することにより、ユーザーインターフェイス・エージェント間の協調を実現した。これによって、いくつのユーザーインターフェイス・エージェントが連動し合うのか予め決定することができない状況に適応することが可能となった。逆階層構造による一貫性のための連動は、各ユーザーインターフェイス・エージェントが、集中管理ユーザーインターフェイス・エージェントの存在なしに自律的に行なうものである。こ

の連動は従来から協調問題解決と呼ばれてきたものである。

ユーザーインターフェイスの例からも明らかなように、一般に協調計算には上記２種のタイプのシステム構成が混在する。したがって、手法１のタイプのシステム構成に対しては階層構造を、手法２に対しては逆階層構造を適切に用いることが重要である。これによって、与えられた問題に対して無理のないシステムの構築が可能となる。

ここで、逆階層構造と階層構造をいかに共存させるかという問題が生じる。従来は、両者が共存するシステムはみられない。これは次のような理由によるものと思われる。

逆階層構造は構成要素として自律的なエージェントを前提とし、それらの協調的動作によってシステム全体の機能を実現するものである。一方、階層構造は集中管理によってはじめて動作する隷属的サブシステムを構成要素としている。したがって、階層構造の隷属的サブシステムを逆階層構造の構成要素として用い、自律的な動作をさせることは本質的に不可能である。

本研究で両者が共存できたのは、階層構造中のエージェントが隷属的でなく、独立しているからである。自律的振るまいは、隷属的サブシステムには追加できないが、独立したエージェントには追加できる。すなわち、階層構造中のエージェントが逆階層構造にも加わって、そこで必要な協調を行うことができる。

（２）逆階層構造十階層構造におけるデーモンの役割

本研究では、階層構造における集中管理と逆階層構造における協調の両方を、デーモンによって行なっている。階層構造における集中管理は、下位エージェントを監視するデーモンを上位エージェントに設けることによって実現されている。上位のエージェントに通知するためのコードを、下位のエージェントに埋めこむ必要がないのがデーモンの利点である（下位が上位から独立する）。監視のためのコードは、上位エージェントにのみ記述されている。

逆階層構造における協調は、共有される計算部を監視するデーモンを各ユーザーインターフェイス・エージェントの対話部に設けることによって実現されている。ユーザーインターフェイス・エージェントに関する情報を計算部に埋めこむ必要がないのがデーモンの利点である（計算部が対話部から独立する）。計算部の状態変化を検出するためのコードは、ユーザーインターフェイス・エージェントの対話部にのみ記述されている。今回の実装で重要なのは、この２種類のデーモンが逆階層十階層構造中に共存し、しかもそれらが連係して動くことである

（３）まとめ

マルチ・エージェントによるユーザーインターフェイス構築技法は、階層構造に基づく方法と逆階層構造に基づく方法が従来より提案されている。しかし、いずれの方法によってもユーザーインターフェイスの開発は容易であるとは言い難い。

本研究では、エージェントの協調を一体化のための協調と一貫性のための協調とに分けた。階層構造に基づく方法、逆階層構造に基づく方法は、それぞれ一体化のための協調、一貫性のための協調の記述に適し、もう一方の記述には適さない。現実のユーザーインターフェイスには、いずれのタイプの協調も含まれることから、両方の構造を用いてユーザーインターフェイスを記述すべきである。この仮説から下記のユーザーインターフェイス制御機能等を期待できる。

・複雑なユーザーインターフェイスの一貫性を保つことができる。
・ユーザーインターフェイス以外によって計算部の状態が変更されたときも、ユーザーインターフェイスは一貫性を保つことができる。
・各ユーザーインターフェイス・エージェントを独立に作成し、実行できる。
・各ユーザーインターフェイス・エージェントの記述が適度な大きさである。

という利点を持つことを確認した。階層構造または逆階層構造の単独の適用では、これらの利点は同時に得ることはできない。１ユーザーインターフェイスに限らず、互いに影響し合う複数のエージェントで構成されるシステムは、元来１つの問題を複数のエージェントへと分割したタイプのシステムと、独立した処理を行なうエージェントが必要に応じて情報をやり取りしてシステムの協調動作が行なわれるタイプのシステムの両方の特徴を合わせ持つ。

これらのタイプのシステムには、それぞれ階層構造と逆階層構造による記述が適している。従って、階層構造と逆階層構造の併用によるシステム構築法は、マルチ・エージェントシステム一般に対する構成法をも与えている。階層構造と逆階層構造におけるエージェントの協調は、いずれもデーモンを用いて実現している。階層構造において、下位のエージェントをその上位エージェントから独立させること、逆階層構造において計算部を対話部から独立させることが、デーモンによって始めて可能になった。

ユーザーインターフェイスは、システム開発の成功の鍵を握る最も重要な要素の一つである。システムが使い易いか使い難いかを決めるのは、このサブシステムであるためである。いかにそのシステムが高度なシミュレーション機能を有していようと、また、いかに多くの特別なオプションを有していようと、操作する者にとって、そのシステムが使いづらければ、システムが最大限に活用されることはない。地層処分ヴァーチャル・エンジニアリング・システムの対話機能用として検討すべき対話機能は、マルチ・エージェント・ユーザーインターフェイス方式によることが将来への可能性の高い開発手法と考えられる。

３．４　試作基本対話機能

試作したグラフィック・ユーザー・インターフェイスは、将来の地層処分ヴァーチャル・エンジニアリング・システムの中核となる並列複合事象シミュレーションの実行を操作するためのグラフィック・ユーザー・インターフェイス機能を想定した。

機能は大別して、シミュレーション・シナリオの決定、シミュレーションの準備、シミュレーションの実行、シミュレーション結果の整理という４種類とした。尚、システム構成については、異機種分散型対話方式としてマシン・パラレルな環境を試行した。

３．４．１　操作用ユーザーインターフェイス

図３．４－１に試作システムの構成を示す。試作システムは、４台のコンピュータと６台の大小モニタ画面を１システムとして協調動作できるよう統合化したものである。

図３．４－１　試作システムの構成

図３．４－１に示すとおり、本システムは６台のディスプレイを接続、それぞれのディスプレイは個別の機能を提供する独立した機能セグメントである。それらは下記の機能を提供する。

（１）主操作機能

図３．４－２に主操作画面を示す。これは、システムを初期設定した時表示される画面である。

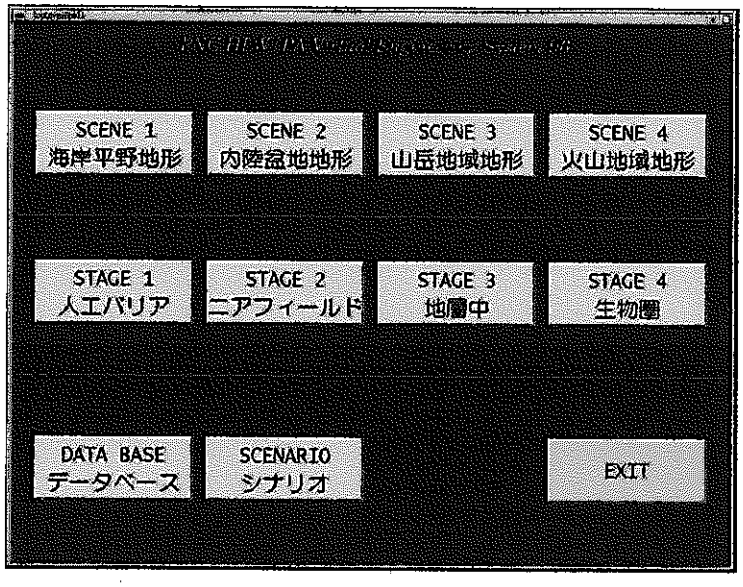

図３．４－２　主操作画面

図３．４－２に示すとおり、主操作画面は簡単なメニューから構成した。メニューは下記の分類に添ってボタン・ウイジェッドを実装した。

天然地形シーン想定ボタン・グループ

シーン１：海岸平野地形選択（３次元海岸平野地形オブジェクトを構成する情報）

シーン２：内陸盆地地形選択（３次元内陸盆地地形オブジェクトを構成する情報）

シーン３：山岳地域地形選択（３次元山岳地域地形オブジェクトを構成する情報）

シーン４：火山地域地形選択（３次元火山地域地形オブジェクトを構成する情報）

図３．４－３に天然地形選択画面を示す。

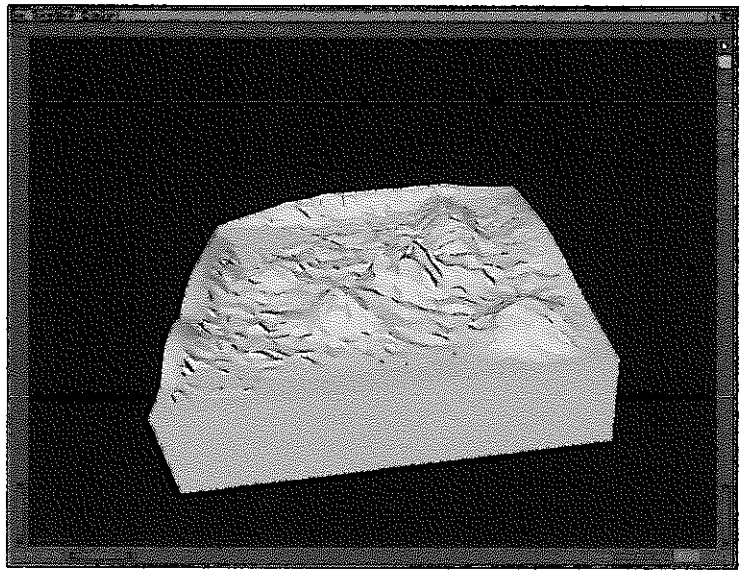

図３．４－３ 天然地形情報の例

ステージ選択ボタン・グループ

ステージ１：人工バリア核種放出シナリオ

人工バリア性能評価シミュレーション操作を開始するボタンウイジェッド

ステージ２：ニアフィールド核種放出シナリオ

ニアフィールド性能評価シミュレーション操作を開始するボタンウイジェッド

ステージ３：地層中核種移行シナリオ

地層中核種移行シミュレーション操作を開始するボタンウイジェッド

ステージ４：生物圏核種移行シナリオ

生物圏核種移行シミュレーション操作を開始するボタンウイジェッド

データベース保守等の補助ボタン

データ等の保守操作を開始するボタンウイジェッド

システムの終了ボタン

本システムの操作を終了するボタンウイジェッド

図３．４－４にステージ１：人工バリア核種放出シナリオ選択画面を示す。

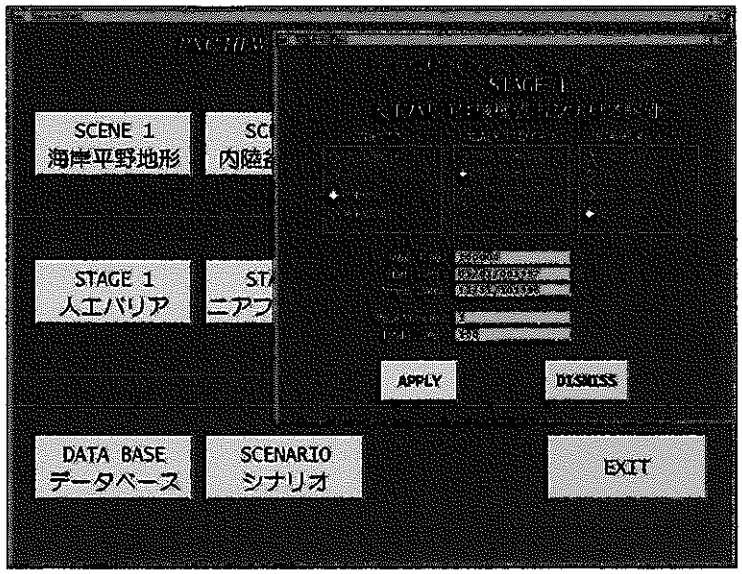

図３．４－４　ステージ１：人工バリア核種放出シナリオ選択画面

（２）３次元視覚化機能

図３．４－５に３次元視覚化機能を示す。図３．４－５ではシミュレーション中の画面を表示している。

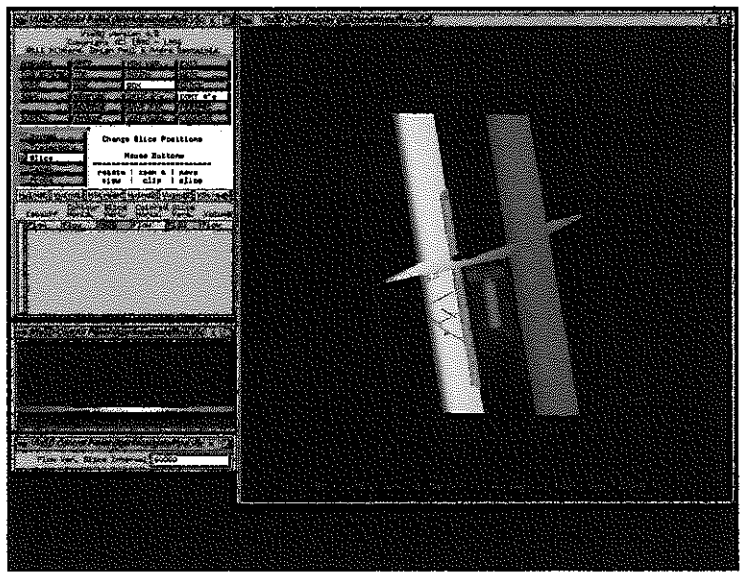

図３．４－５　３次元視覚化の例

（3）シミュレーション・パラメータ表示機能１：タイム・パラメータ

図３．４－６にシミュレーション・パラメータ表示機能１：タイム・パラメータを示す。図３．４－６ではシミュレーション中の画面を表示している。

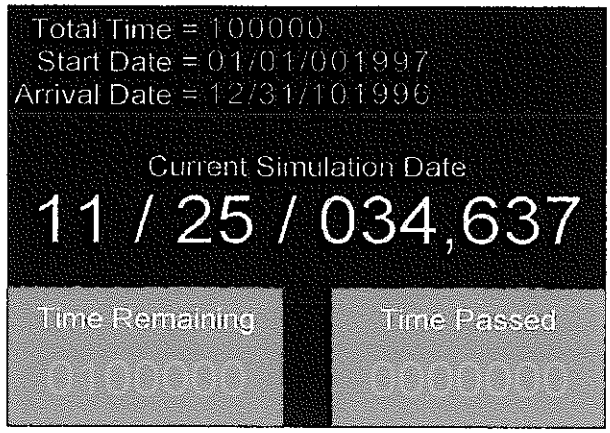

図３．４－６　タイム・パラメータ表示機能の例

（4）シミュレーション・パラメータ表示機能２：選択シナリオFEP

図３．４－７にシミュレーション・パラメータ表示機能２：選択シナリオFEPを示す。図３．４－７ではシミュレーション中の画面を表示している。

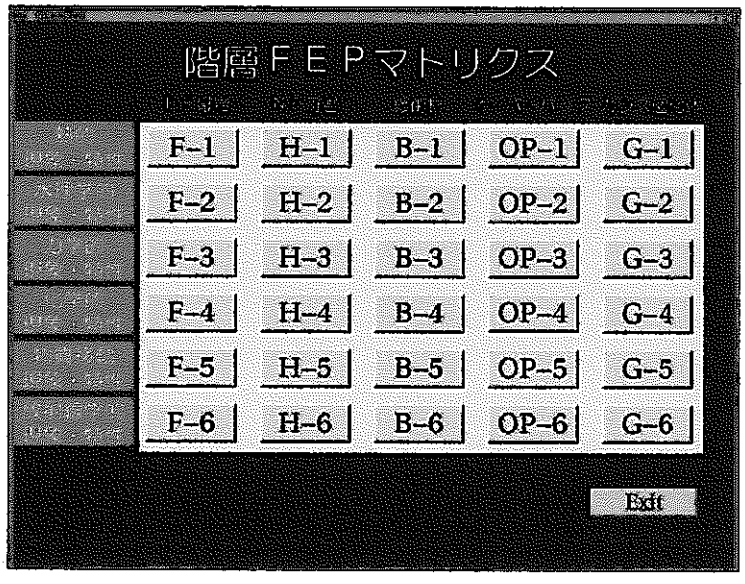

図３．４－７　選択シナリオFEP表示機能の例

（5）シミュレーション・パラメータ表示機能３：計算条件、等

図３．４－８にシミュレーション・パラメータ表示機能２：選択シナリオFEPを示す。図３．４－８ではシミュレーション中の画面を表示している。

図３．４－８　計算条件、等表示機能の例

（6）シミュレーション・パラメータ表示機能４：ステージ・パラメータ

図３．４－９にシミュレーション・パラメータ表示機能２：選択シナリオFEPを示す。図３．４－９ではシミュレーション中の画面を表示している。

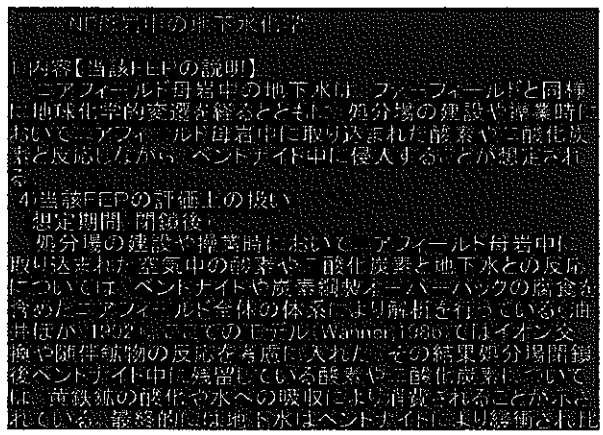

図３．４－９　ステージ・パラメータ表示機能の例

（7）データベース・アクセス

図３．４－１０にFEPデータベース・マトリクス画面例を示す。図３．４－１０は試作した分散型マルチメディアFEPデータベース・システムの例である。

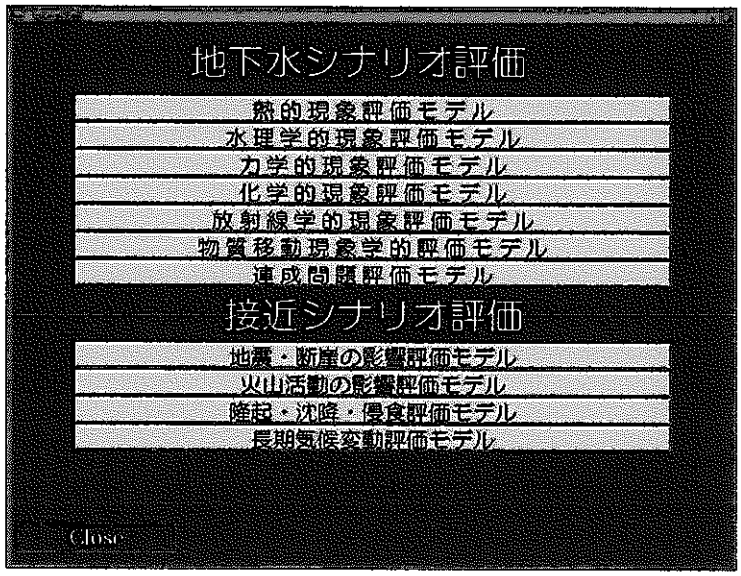

図３．４－１０　データベース・アクセス表示機能の例

３．４．２　試作対話システム制御機能概念

（１）主操作制御仕様：Main Console Panel

機能：地層処分性能評価ヴァーチャル・エンジニアリング・システムの起動

目的：ユーザー、ハードウェア等リソースを確認してシステムを起動する

仕様：試作システムでは、User ID, Password等を確認し、システムを初期化するログイン手続き用GUIを準備する。尚、User ID, Password については固定とし、それ以外であれば警告を出して終了。

表示：１）コンソール　システム・コンソールGUIを表示

２）ビュアー　クリアー後スタンバイ・メッセージ表示

３）ＰＣ　クリアー後スタンバイ・メッセージ表示

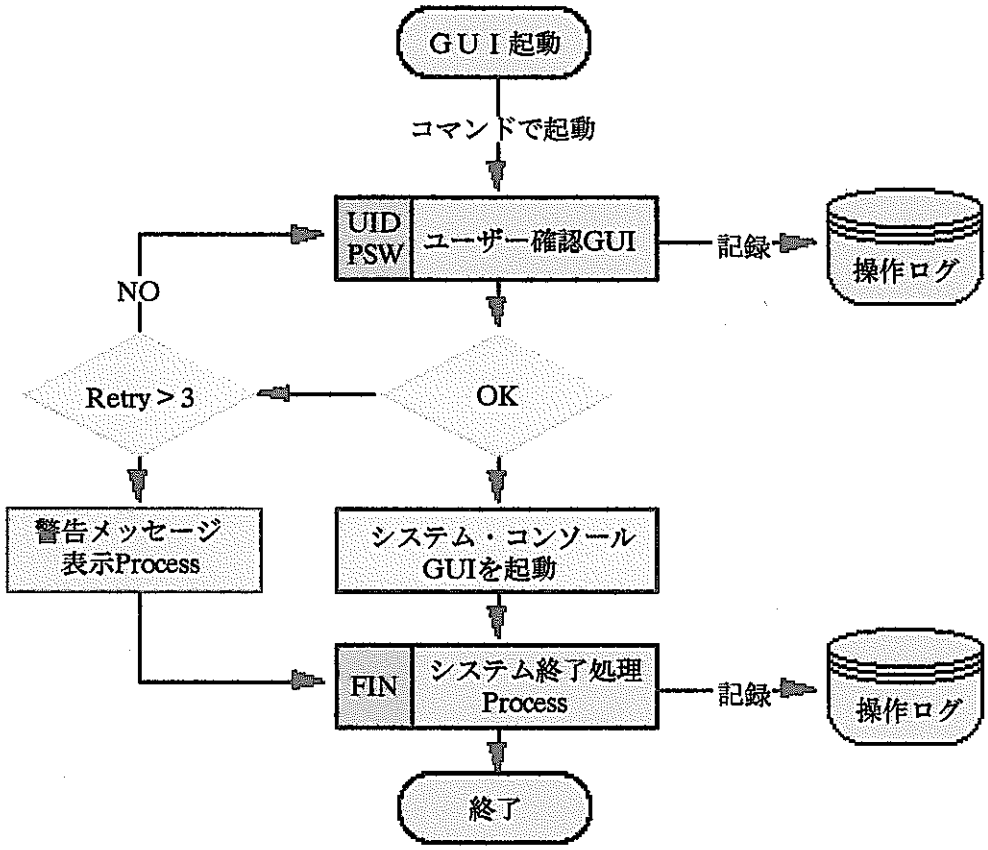

図３．４－１１　主操作制御仕様

（２）システム・コンソール制御基本ウイジェッド

機能：システム・コンソール・ボタン・ウイジェッド・コール

目的：シーン、ステージ、保守、終了処理機能を起動する

仕様：試作システムでは、天然地形を選択することにより、動作をリセットする。オブジェクト・ビュアーに選択シーンの天然地形を表示する。平成８年度に試作するシステムでは、天然地形シーンのオブジェクトはステージ４のオブジェクトと同等であるとする。ステージ１から４までステージ・ボタンは評価するステージに入るボタンとする。従って、ステージ・ボタンを選択すると選択されたステージのGUIが表示されるものとする。

表示：１）コンソール：ステージGUIを表示

２）ビュアー　：シーン、ステージで選択された３Ｄオブジェクトを表示

３）ＰＣ－Ａ　：ステータス情報表示（選択ウイジェッド等）

４）ＰＣ－Ｂ　：シミュレーション情報表示（初期状態）

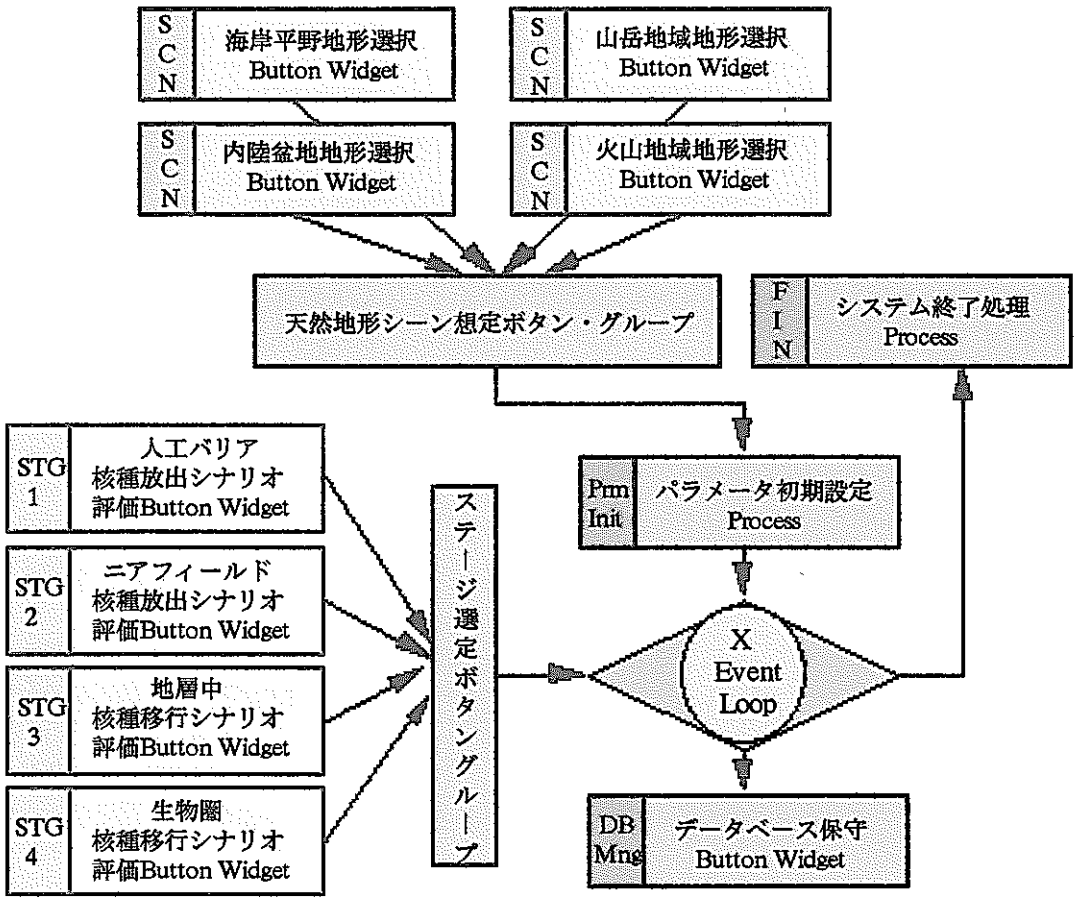

図３．４－１２　システム・コンソール制御基本ウイジェッド仕様

（３）シーン選択ボタン海岸平野：内部処理Button Widget

機能：シーン選択ボタン（海岸平野内部処理概要）

目的：想定される処分システムをとりまく天然地形を海岸平野に設定する。

仕様：試作システムでは、単に海岸平野地形に建設された処分システムの概観を３Ｄ表示するにとどめるが、シミュレーション情報ファイルに半島地形が選定されていることを示すフラグを立て、次のシーン選択アクションが行われるまでの間の性能評価プロセス等における参照に利用する。これは、将来的に天然地形の相違にかかわるシナリオの相違が考えられるため、フラグを立て、各ステージ、ターンにて、現在選択中の天然地形を確認する必要が生じることがシステム開発上予想されるためである。

表示：半島オブジェクトを大型モニタに表示。（改良Vis5D）

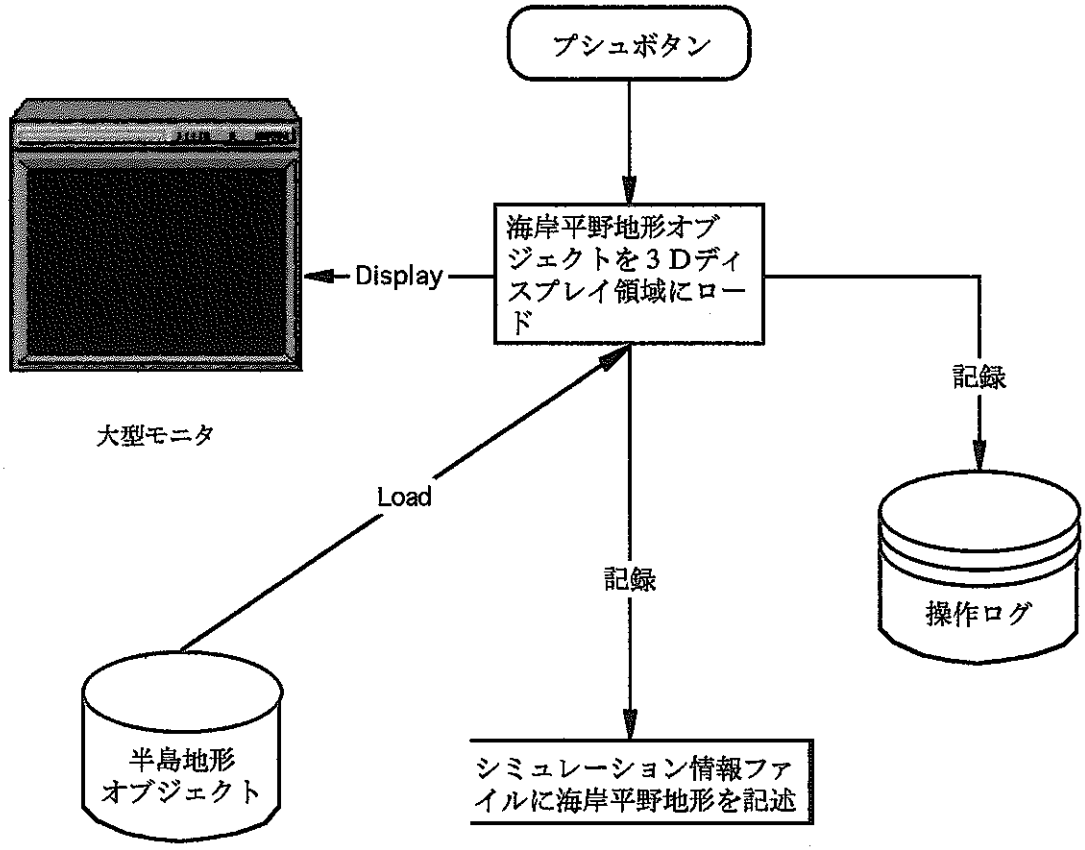

図３．４－１３　シーン選択ボタン海岸平野：内部処理

（４）シーン選択ボタン内陸盆地：内部処理Button Widget

機能：シーン選択ボタン（内陸盆地内部処理概要）

目的：想定される処分システムをとりまく天然地形を内陸盆地に設定する。

仕様：試作システムでは、単に内陸盆地地形に建設された処分システムの概観を３Ｄ表示するにとどめるが、シミュレーション情報ファイルに平野地形が選定されていることを示すフラグを立て、次のシーン選択アクションが行われるまでの間の性能評価プロセス等における参照に利用する。これは、将来的に天然地形の相違にかかわるシナリオの相違が考えられるため、フラグを立て、各ステージ、ターンにて現在選択中の天然地形を確認する必要が生じることがシステム開発上予想されるためである。

表示：平野オブジェクトをオブジェクト・ビューワー画面に表示。（改良Vis5D）

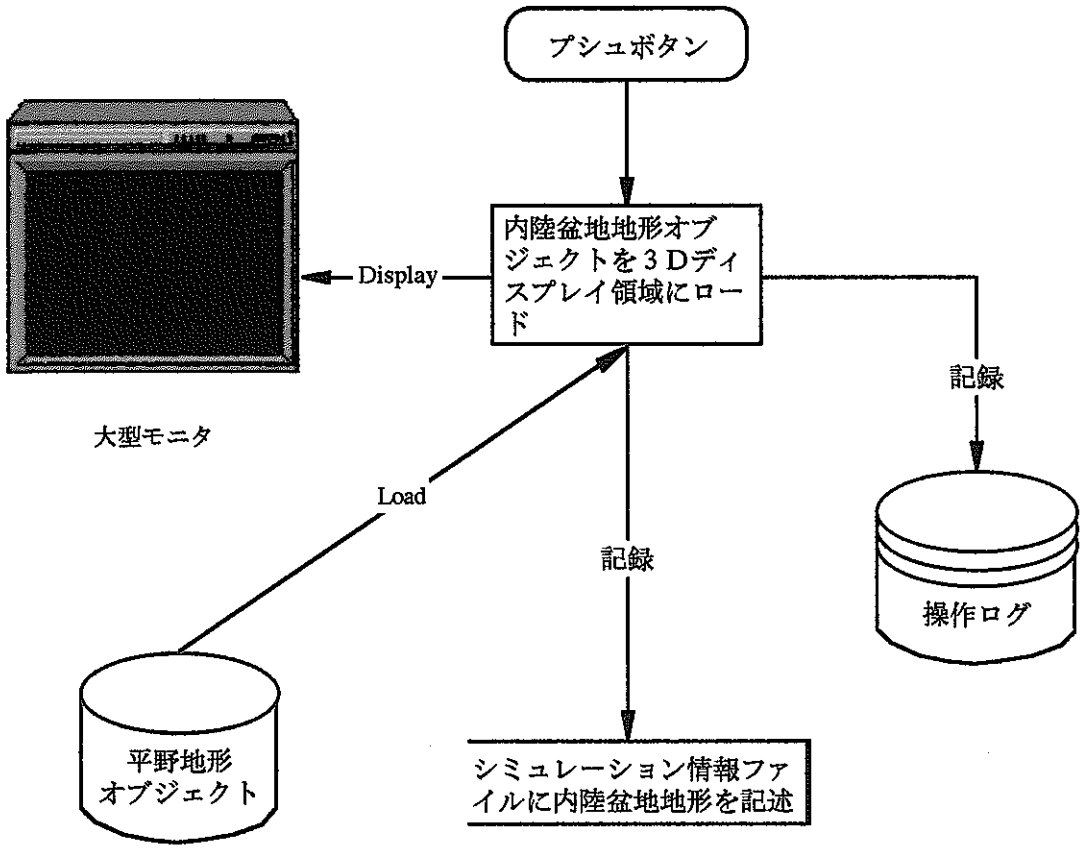

図３．４－１４　シーン選択ボタン内陸盆地：内部処理

（５）シーン選択ボタン山岳地域地形選択Button Widget

機能：シーン選択ボタン（山岳地域内部処理概要）

目的：想定される処分システムをとりまく天然地形を山岳地域に設定する。

仕様：試作システムでは、単に山岳地域地形に建設された処分システムの概観を３Ｄ表示するにとどめるが、シミュレーション情報ファイルに山岳地域地形が選定されていることを示すフラグを立て、次のシーン選択アクションが行われるまでの間の性能評価プロセス等における参照に利用する。これは、将来的に天然地形の相違にかかわるシナリオの相違が考えられるため、フラグを立て、各ステージ、ターンにて、現在選択中の天然地形を確認する必要が生じることがシステム開発上予想されるためである。

表示：山岳地域オブジェクトをオブジェクト・ビューワー画面に表示。（改良Vis5D）

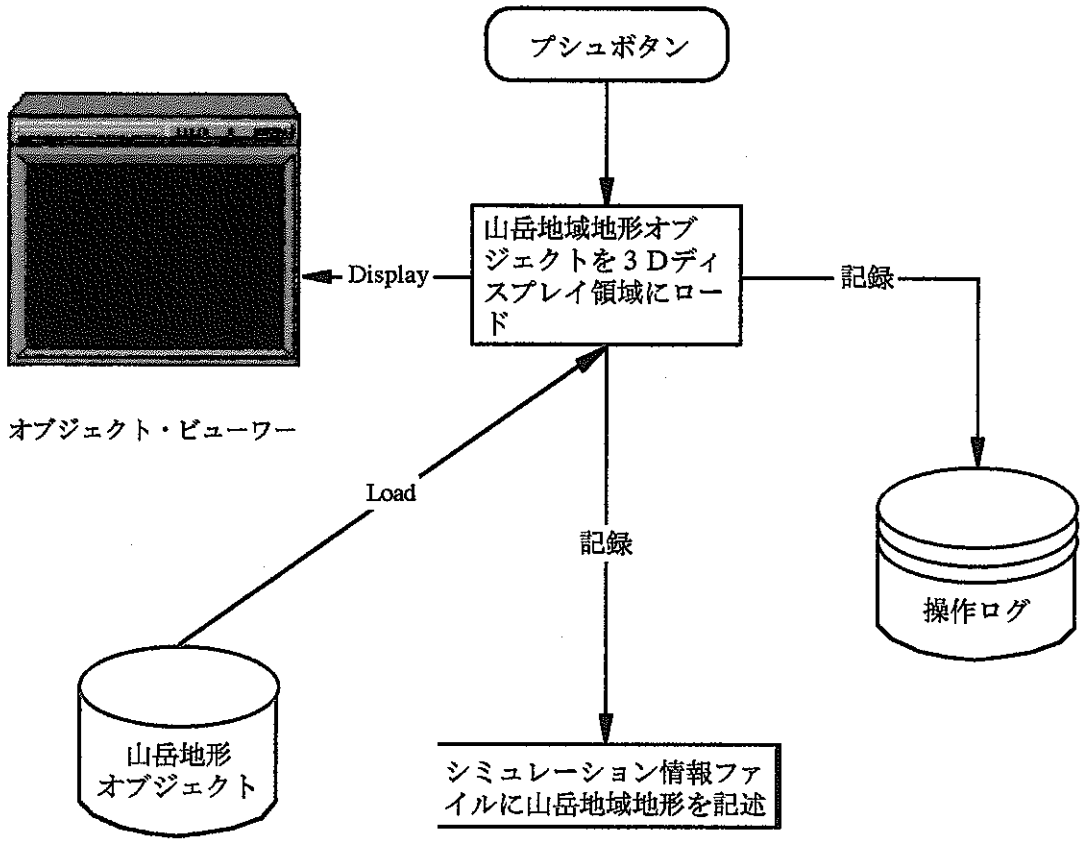

図３．４－１５　シーン選択ボタン山岳地域地形選択内部処理

（６）シーン選択ボタン火山地域：内部処理Button Widget

機能：シーン選択ボタン（火山地域内部処理概要）

目的：想定される処分システムをとりまく天然地形を火山地域に設定する。

仕様：試作システムでは、単に火山地域地形に建設された処分システムの概観を３Ｄ表示するにとどめるが、シミュレーション情報ファイルに火山地域地形が選定されていることを示すフラグを立て、次のシーン選択アクションが行われるまでの間の性能評価プロセス等における参照に利用する。これは、将来的に天然地形の相違にかかわるシナリオの相違が考えられるため、フラグを立て、各ステージ、ターンにて、現在選択中の天然地形を確認する必要が生じることがシステム開発上予想されるためである。

表示：山岳オブジェクトをオブジェクト・ビューワー画面に表示。（改良Vis5D）

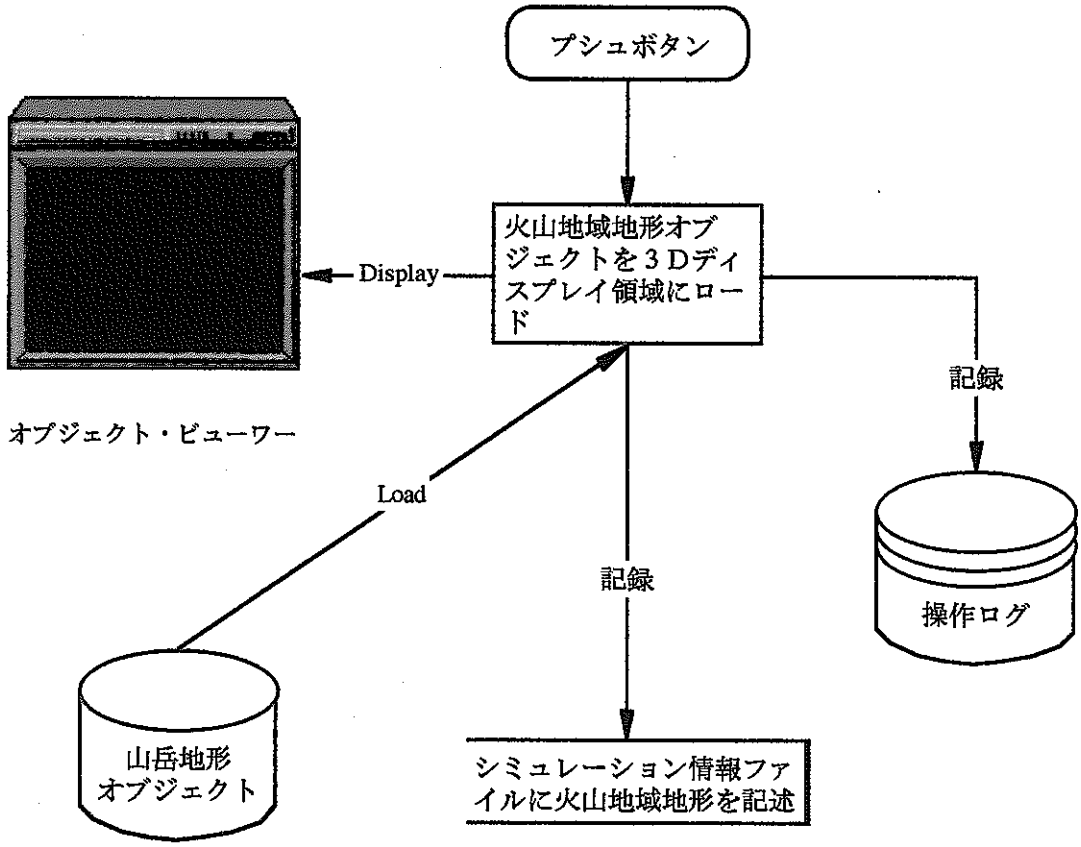

図３．４－１６　シーン選択ボタン火山地域：内部処理

（７）パラメータ初期設定内部処理

機能：システム・パラメータ初期設定

目的：シミュレーションに必要なシナリオ、ステージ、カレンダー等共通情報を初期設定する

仕様：試作システムでは、１ターンの間保持するシミュレーション情報について初期化する。これらの情報は、パラメータ・ディスプレイ・パネルに常時表示される。実際にはコンソール・システムの/usr/tmpにファイルが作成され、各パラメータ・ディスプレイ・システムにより、指定したタイム・インターバルで参照するものとする。

表示：Panel-A）シミュレーション用、カレンダーと時刻パラメータの表示。

Panel-B）現在、選択中のシナリオ関連の表示。

Panel-C）シミュレーション・パラメータ関連の表示。

Panel-D）ステージ、ターンの表示。

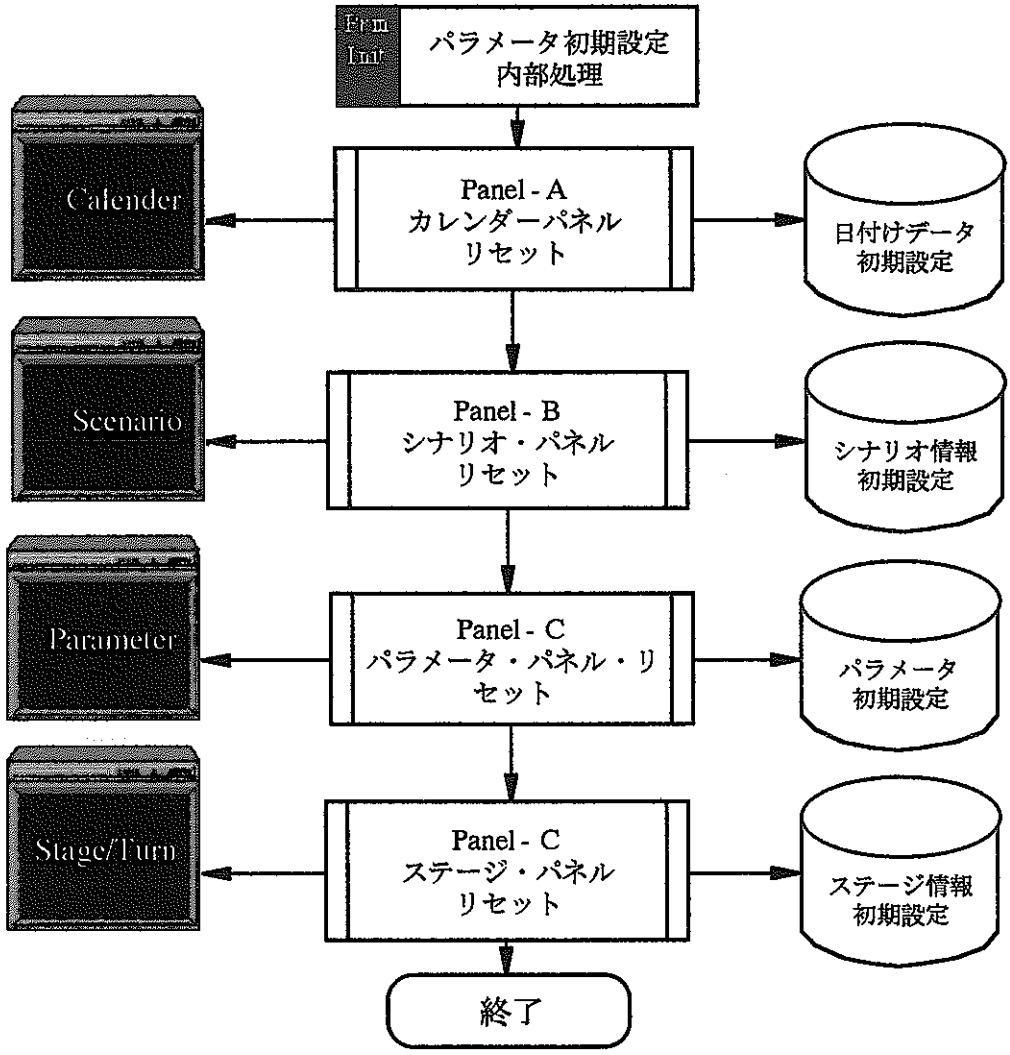

図３．４－１７　パラメータ初期設定内部処理

（８）ユーザー確認GUI

機能：ユーザー確認、システム初期設定

目的：ユーザー、ハードウェア等リソースを確認してシステムを起動する

仕様：試作システムでは、User ID, Passwordを固定とし、それ以外であれば警告を出して終了とする。

表示：１）コンソール　システム・コンソールGUIを表示

２）ビュアー　クリアー後スタンバイ・メッセージ表示

３）ＰＣ　クリアー後スタンバイ・メッセージ表示

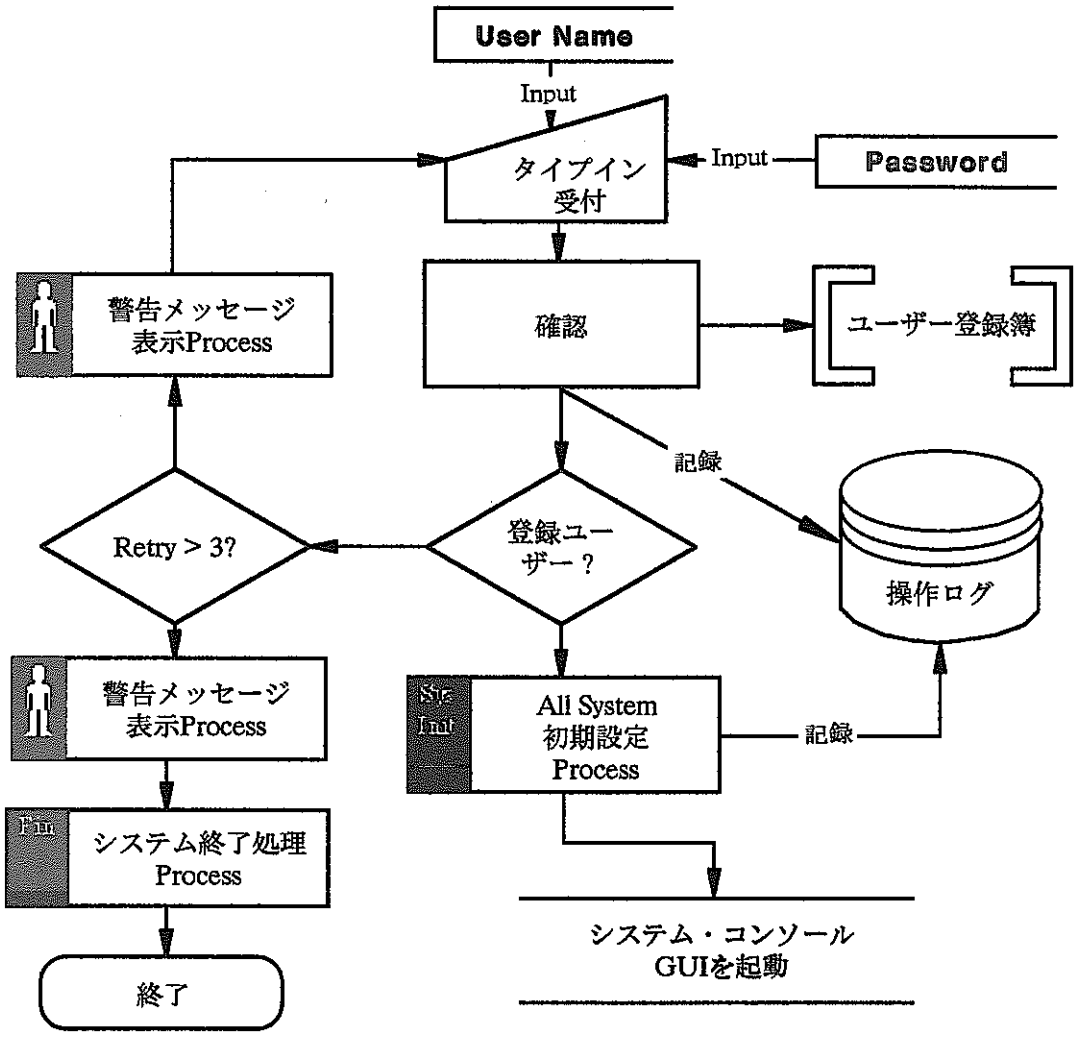

図３．４－１８　ユーザー確認GUI内部処理

（９）全システム初期設定

機能：地層処分性能評価ヴァーチャル・インターフェイス・エンジニアリング・システム初期設定

目的：ユーザー、ハードウェア等リソースを確認してシステム初期設定する

仕様：試作システムでは、各ディスプレイ画面等を初期設定する。準備完了メッセージを表示する。

表示：１）コンソール　システム・コンソールGUIを表示

２）ビュアー　クリアー後スタンバイ・メッセージ表示

３）ＰＣ　クリアー後スタンバイ・メッセージ表示

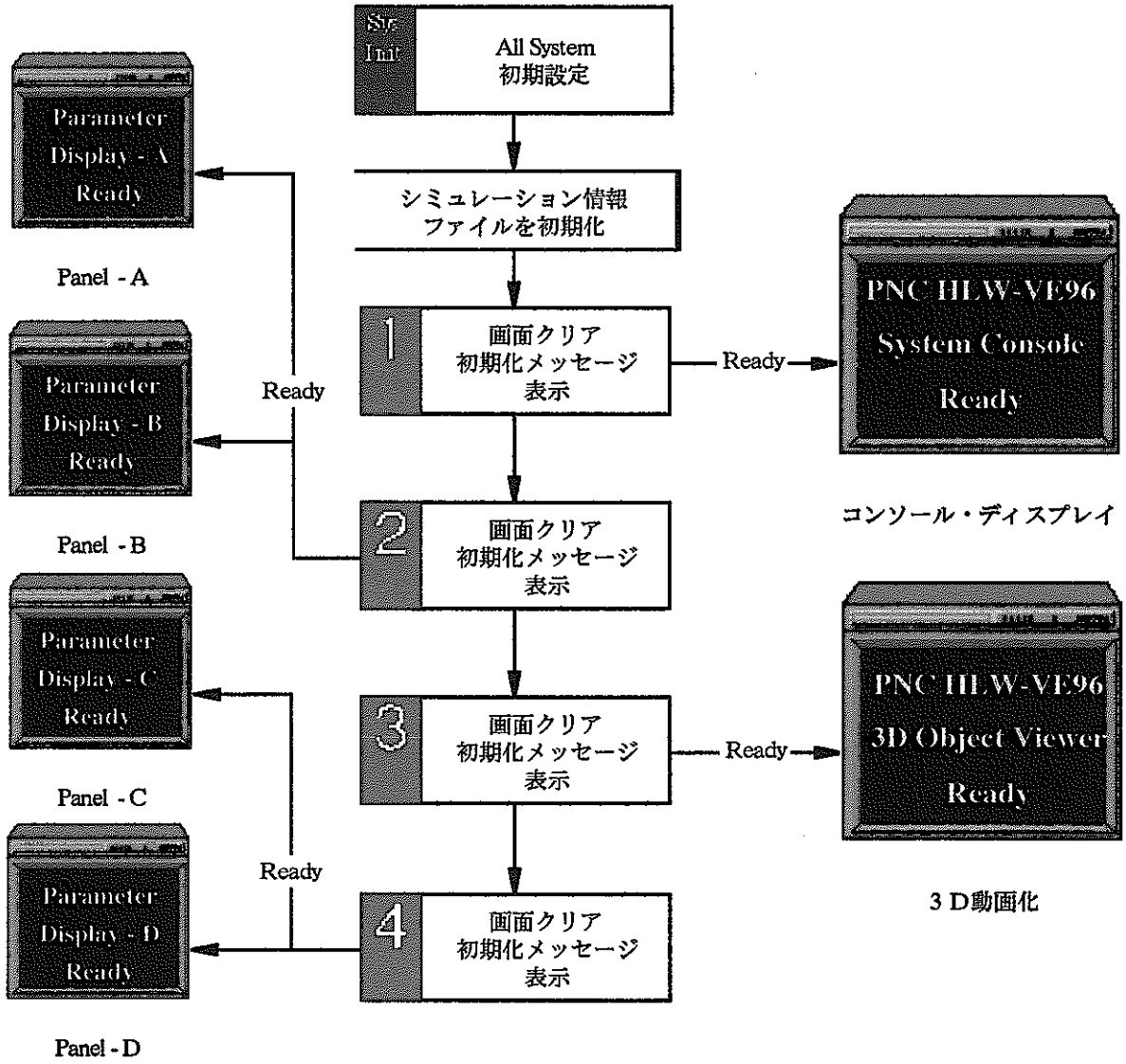

図３．４－１９　全システム初期設定内部処理

（10）Panel－A：カレンダー・パネル

機能：パラメータ表示装置　系統－A

目的：シミュレーション等の時刻、カレンダー情報の表示

仕様：試作システムでは、現在、実時刻、シミュレーション現在時刻等を表示し、シミュレーションの時間依存性の視覚的効果を高める。

備考：スライダー等のビジュアルな動図による表現を併用することが好ましい。

表示：１）シミュレーション終了時刻（取敢ず西暦１０万年、１０００世紀）

２）現シミュレーション時刻（実行時にはカウント・アップ）

３）実時間（実際の時計時刻）

４）表示仕様は下図を基本案として視覚的に見やすくスマートな表現を行う

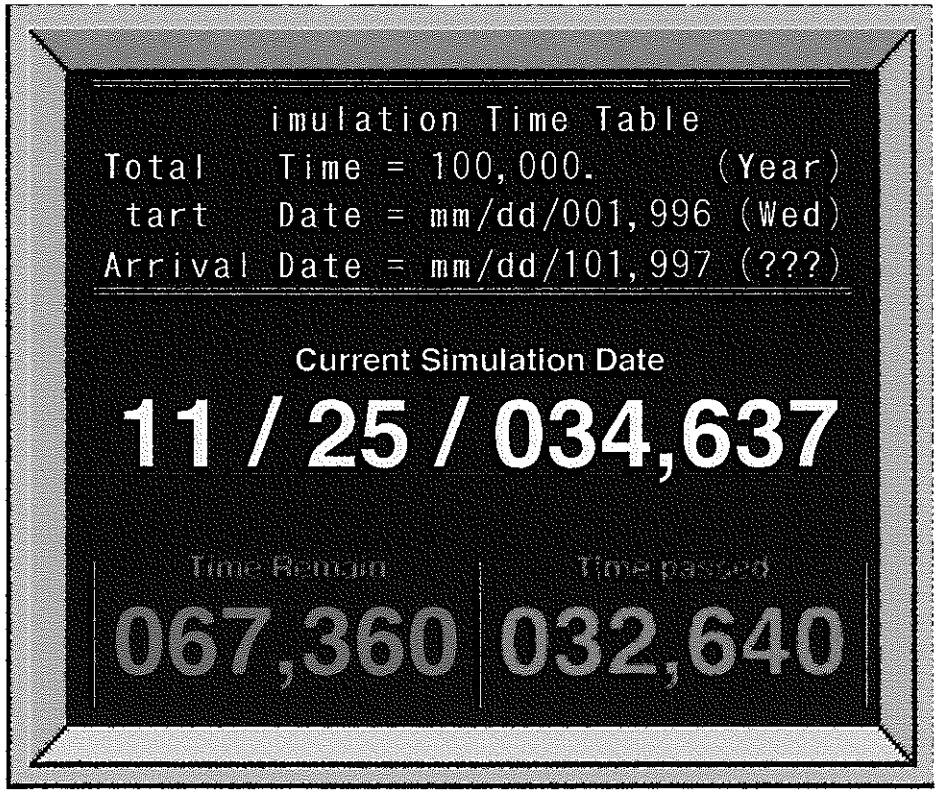

図３．４－２０　全システム初期設定内部処理

（１０）Panel－B：シナリオ・パネル

機能：パラメータ表示装置　系統ーB

目的：選択した評価シナリオに関する情報の表示。

仕様：試作システムでは、現在、評価中のモデルに関連するFEPリストを表示する。

表示：１）選択したシナリオをFEPリストから表示する

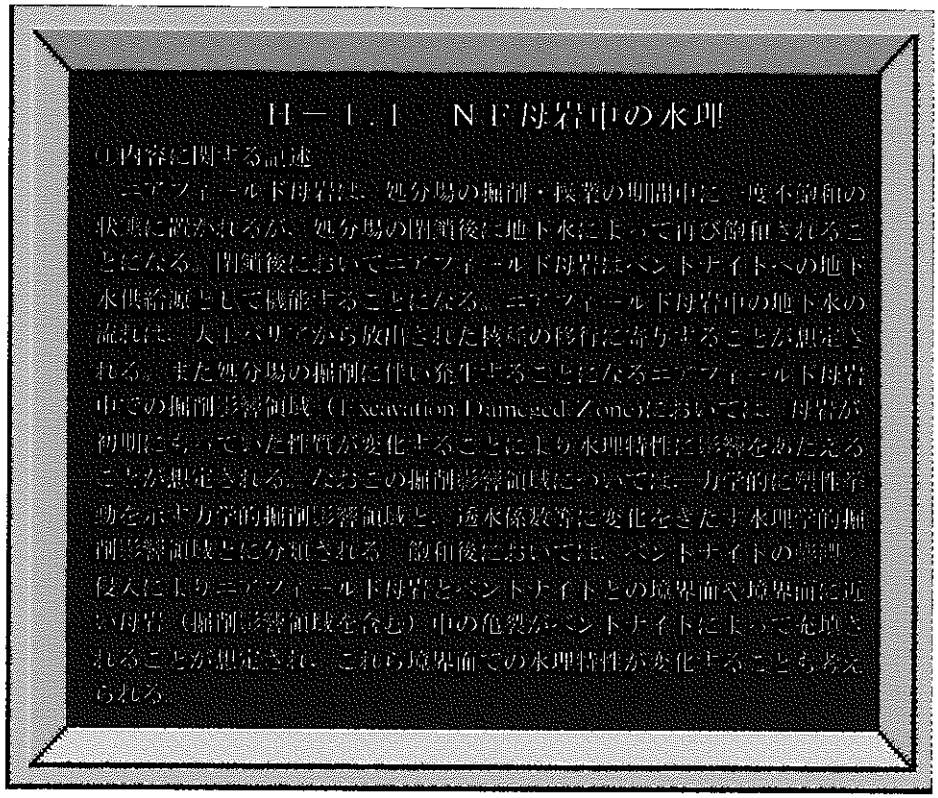

図３．４－２１　Panel－Ｂ：シナリオ・パネル

4　ユーザー・インターフェイスの高度化検討

ユーザーとの対話処理操作をより効率化すると供に、ユーザーによる意思決定を支援する為の参考となる情報を提供する機能について検討した。

地層処分ヴァーチャル・エンジニアリング・システムとユーザーとの対話は、評価シナリオの決定、シミュレーションの実行、実行結果の整理等、シミュレーションに直接関連する操作に始まり、シミュレーション結果の妥当性等を検証・確証するための関連情報をデータベースより検索し、解析結果と比較評価する、等のさまざまな性能評価計算に係わる操作が予想される。また、シミュレーション・コードやデータ等、ソフトウェアの管理に係わる操作も考えられる。対話システムを利用しやすいシステムとして構築するためには、その設計思想に使いやすさが吟味されていなければならない。

例えば、ある系統立った評価シミュレーションを実施する場合、シミュレーションは評価シナリオにそって進められるべきであり、個別の評価シナリオに関連するFEPリストやインフルエンス・ダイアグラムはシミュレーション操作の支援知識となるであろう。

4．1　地層処分性能評価シミュレーションとシステムの操作

使いやすいユーザーインターフェイスを設計するためには、操作全体の流れと、画面のレイアウト、メッセージの解かりやすさ等が吟味されなければならない。そのためには、システムの目標や機能について整理し、操作のゴールを明確にすることが重要である。

本項では、本システムのユーザーインターフェイスの設計開発や、高度化に係わる改良整備要件導出のために、評価作業全体について外観し、作業に内在する確かさや不確かさ等、本システムのユーザーの判断に委ねるべき事項を整理し、それらの意思決定を支援するためのユーザー・インターフェイスの基本的な技術等について検討する。

高レベル放射性廃棄物処分システムの安全性評価は、処分システム開発プロジェクトの各段階について行われる。地層処分ヴァーチャル・エンジニアリング・システムは、それらの各段階について、目的に適合した結果を提供できなければならない。このため、その各開発段階に応じて評価結果の使用目的が異なる。

当面の地層処分ヴァーチャル・エンジニアリング・システムの目的は、処分システムの開発における第2段階で実施される処分サイトの特定、及び処分システムの概念設計の妥当性を判断する情報を提供することにあると考えられる。具体的には、想定サイトでの処分の環境影響の概要、処分システムの設計オプションに関する多方面からの安全性評価比較の2点にあると考えられる。

4．1．1　安全性評価作業の基本的な進め方

安全性評価の基準としては、処分が公衆にもたらすリスクが、社会的に十分許容されるレベルかどうかで判断することが基本であると考えられる。また、安全性評価のためのシナリオ選定として効率的な評価を行うためには、FEPによる手法と、合わせて事象の発生確率と影響の程度に応じて評価方法を合理的に選定する必要があると思われる。従って、個別シナリオと確率論的シナリオを用いて、評価シナリオをより簡素化し、下記の分類に添ってシミュレーション等を行い、影響を評価することが安全性評価の基本的考え方と思われる。また、この考え方は、地層処分ヴァーチャル・エンジニアリング・システムの操作の基本的な考え方となるものである。それは、リスクの影響度と頻度から下

記のクラスに分類することにより、情報を構造化することができる。

Class 1：影響大頻度大　設計で対応する
Class 2：影響大頻度小　定性的な評価を行う
Class 3：影響小頻度大　定量的な解析評価を行う
Class 4：影響小頻度小
定性的な検討を行う。ここで大小の判断基準が問題になるが、このアプローチでは
＊「影響大」、「頻度大」の場合には両者は比較的良い精度で推定できること、
＊影響の大小より、事象の発生確率の大小の推定の方が技術的に困難であるという考えがベースにあると考えられる。

高レベル放射性廃棄物処分に係る安全性評価は、計画する多重バリアシステムによる放射性核種の長期間に亙る隔離能力を対象に行われなければならない。従って、システムの操作の基本は、この多重バリアの性能を数値的に評価し、今度の処分システム開発に参考となる情報を提供することを目的とした作業に適していることが望まれる。これらの問題を効果的に解決するには、次の様な考え方を基にすることが妥当であるとしている。

４．１．２　システム操作から見た評価期間の考え方

いづれの評価を行う場合にも、評価期間により評価手法も評価に必要な精度も異なる。そこで著書らは次の３期間に評価期間を分けるのが良いと考える。

（１）短期（約1000年まで。処分後の地下水浸潤等比較的過渡的な期間）

廃棄物中の全放射能量の時間変化を見ると、初期の1000年で約1／1000になることが知られている。それ以降は、精練に使ったウラン鉱石全体中に存在する量と大差ないと考えられる。放射能濃度でみるとウラン鉱石の約10倍程度になる。従って、前述の『影響大』のケースClass 1，2が該当する事象が起きる可能性のあるのは1000年までであり、この期間は『影響大』の事象が発生しないように設計することが望ましい。システムの開発に於ては、この間は複雑な地球化学的過程を中心とした現象に関心が集まることを重視して設計すべきである。

歴史的資料や考古学的建造物等の情報をデータベース化し、工学的手法参考になる情報を提供することにより設計作業を支援できる。また、事象の発生確率の大小をPSA等により、根拠をもって定性的に予測することが可能である。

しかし、現代の経験で確かなことが言えるのは100年程度であり、科学的予側の精度を上げるためには更に基礎研究が必要である。そのためには、この期間の現象解明に係わるより包括的な評価を支援する対話機能が必要である。

（２）長期（約1000－10,000年）

放射能濃度は1万年経過するとウラン鉱石と同じオーダになると考えられている。従って、長期の

評価に於ては影響は少ないと考えられるが、発生の確率が高いと思われる事象シナリオについて評価の関心が高まると思われるため、対話システムの機能は、より多くの気候変動や地質等のデータベースについて充実したものが求められると考えられる。

例えば、氷河期の周期は数万年オーダであり、約1万年程度であれば、自然条件、生物活動等が現在と同じと考えても不自然ではないと思われる。生物圏に漏出する放射能量による影響を現在の生活条件を根拠に定量的に被曝線量として評価しても不当ではないであろう。

また、第四期（約200万年）の地殻変動データを使うことにより、地日中における約1万年以内の事象は比較的定量的に評価できる。

（3）超長期（1万年以上）

1万年を越える超長期の評価においては、生物活動について人類の存続の疑問も含めて不確定要素が多くなり、被曝線量を評価する意義が疑わしくなるかも知れない。ウラン鉱床中の放射能濃度と処分システム中の平均放射能濃度は大差なくなると思われる。このように、長期の評価について当面は天然に存在する放射性鉱物の放射線影響との比較において、定性的な検討にとどめるのが適当と思われる。従って、対話システムは、より優れたナチュラル・アナログ・データベース等の機能を充実するべきであると考えられる。

4．1．3　システム操作から見た多重バリア方式の考え方

多重バリアシステムを採用した場合は、どの部分を各バリアの境界と考えるかが重要となる。例えば、米国NRCの基準では廃棄物パッケージと地層の2つのバリアに分けている。設計ベースで考えると、オーバーパックと緩衝材を2つのバリアと考え、更に地層バリアが加わり3重のバリアシステムと考えることができる。この場合、容器は密封機能、緩衝材は移行遅延機能、地層は隔離機能（接近できなく、移行距離が長い）と、それぞれ異なった機能で核種移行を阻止する働きをする。従って、この異なった機能を期待する評価シミュレーションにおいては、これらのバリア構成要素を明確に3次元オブジェクト等で視覚化して区別することにより、整理されたシステムを構築する必要があると考えられる。

また、長期の評価においては、高放射線被曝の事象の発生はほとんど考えられないので、極めて低い発生確率であることを定量的に評価することは必ずしも要求されないと思われる。単独で閉じ込め性の高いバリアを重ねる必要はない。しかし、我が国の地層は必ずしも安定ではないので、設計目標としては、地層の不安定性に起因する不確定要素を排除することを目的とした、多重バリアを採用した方が評価が容易となることが多重バリア方式の採用論拠である。この多重バリア方式の処分システムの機能を、周辺母岩と合わせて機能別に分類評価できる対話画面構成が重要である。

また、地層データベースについても詳細な調査結果から構成でき、確実な性能評価が可能な処分システム周辺の岩盤と距離による隔離機能を期待する広い範囲の地層の多重バリア・システム設計の支援機能を考えることができる。

４．１．４　システム操作から見た評価シナリオの考え方

当面の評価シナリオとしては、処分システムの建設時における掘削影響等からの地下水環境変動シナリオ、及びそれぞれが原因となった核種の放出シナリオを開発する必要がある。地層処分ヴァーチャル・エンジニアリング・システムの開発においては、シナリオは別途推進されている地層処分シナリオ・データベースの開発プロジェクトにより得られたものを応用する。

従って、ここでは、シナリオに添ってシミュレーションを進めるための対話機能設計上要求される基本的なシナリオに係わる事項をとりまとめる。

処分システムに及ぼす影響事象を、サイト特定、及び設計によりかなりを除くことができると考えられている。更に、除くことのできない事象のうち、放射能の漏洩に結び付くシナリオを摘出する方法として、それぞれの事象に定性的発生確率を与え、相対的発生確率から重要事象を求める方法がとられる場合が考えられる。

また、定性的な数値は、科学的根拠に基づいて与えることができるとは限らない。それらの数値の見積りに当たって、根拠となる科学的データが得られない場合は、まず専門家の見識に頼ることになる。また、同時に、公衆の感覚的判断も大切になる。そこにいわゆるPublic Criteriaの必要性が生じる。合理的なPublic Criteriaを形成するためには、まず、関連する科学的事実を公衆に伝達することが大切である。そして、長い時間をかけ、多くの人が理解できるよう整理する必要がある。以上のことから、シナリオに添ったシミュレーションを操作する機能には、確率論的評価面から重要事象が示されることや、将来のPublic Criteriaとの関係、社会システムへの影響といった包括的な機能が設計当初より含まれていることが望ましいと考えられる。

４．１．５　システム操作から見た評価モデルの考え方

処分システムの安全性評価のためのモデルは、放出核種の移行過程等を表現する複数のモデル（サブモデル）から構成されている（評価モデル・システム）。これらのモデルは、対話システムから見ればバックグラウンドで独立したタスクとして動作するオブジェクトとして扱うことが好ましいと考えられる。また、それらは、対話システムのウイジェッドや他のタスクからのメッセージで起動され、並列的に協調して動作する必要がある。

サブモデルは、基本的には核種の移行過程をできる限り正確に表現するものであるべきであるが、一方、評価モデル・システムは、総合的な解析を行える機能が必要なため、サブモデルは簡単なものが望ましい。

詳細モデルに於ては、我が国では火山による飛散、地殻変動による露出等の直接被曝シナリオも他国より慎重に評価する必要はあるが、基本的にはサイト特定でそれらの要因を削除できると考えられる。

例えば、地殻のゆっくりした上下動等、必ず起きる地殻変動として評価しなければならない事象もあるが、その場合は、境界条件の時間的変化による地下水の動きへの影響を考慮することになる。これらのことから、我が国においても、地下水による核種移行モデルが評価の基本となると考えられる。

従って、モデルの選定とシミュレーションの実行においては、サイト特定で削除できる要因等を明確に表示し、評価シナリオを簡素化する操作を視覚的に表現することが好ましいと考えられる。

（1）容器の耐久性評価

人工バリアは、固化体（キャニスタ入り）、容器、緩衝材からなる。容器の健全性は、核種の移行の開始時期を決めることになる。容器は完全封入機能を持ち、腐食等の破損による開口部が生じた時点でバリア機能を消失するとして評価する。小さな開口部であっても、一旦開口部ができるとバリア機能が他のバリアの機能に比べて極度に低下する。容器は、キャニスタとオーバパックの組み合わせを考え、代表的材料として、キャニスタはステンレス鋼を考え、オーバパックは炭素鋼を考える。容器の耐久性の評価は、腐食速度の評価と地震等の瞬間的な破壊につながる要因の有無の評価にわけて考えることができる。全面腐食、及び局部腐食の速度に影響する要因として、温度、酸性度、地下水の組成、放射線線量率、酸化還元電位等について考える。シミュレーションに於ては、これらの環境条件の時間変化を求め、その環境条件での腐食速度の積分値として、腐食の厚さを計算する手法等が用いられている。オーバーパックの腐食現象に係わる評価は、化学過程を中心としたシミュレーションになるものと思われ、シミュレーション結果の視覚化表現には新たな手法の開発が必要である。

また、瞬間的破壊としては、応力腐食割れと機械的破壊が考えられる。これらの破壊については、あらかじめ割れの生じる環境条件を実験的に明らかにし、その環境条件が処分環境にないことを確認する。機械的破壊については、破壊応力以上の外力が加わらないことを評価する。これらの要因が生じた時点で容器は破壊し、バリアとしての能力はなくなるものとする。これらの破壊についても、対話システムは優れた視覚化機能が要求されると考えられる。

（2）核種浸出評価

貯蔵時、輸送時の評価においては、時間に比例した浸出が生じるとして、浸出率により評価する。処分システム操業後のソースターム評価においては、処分環境の地下水中での平衡反応から決まる溶解度モデルと拡散、及び地下水の動きに構う物質移動により評価する手法が取られている。

溶解度モデルによる評価のパラメータとしては、処分環境の温度、地下水の組成・pH - Eh、共存固相（ガラス、容器、緩衝材等）の組成が挙げられる。放射線は水の放射線分解をもたらし、Eh - pH等を決める2次要因として重要である。これらの平衡反応による評価要因に加えて、より溶解度の低い結晶性化合物にゆっくりと変換し、実質上核種移行評価系から除かれる、いわゆる鉱物化も重要な評価シナリオである。このような地球化学シミュレーションについて対話システムは、熱力学データベースと連結したパラメータ・サーベイや感度解析機能を提供し、評価作業を支援する機能が求められる。また、化学反応を視覚化する手法の開発も大切であると考えられる。

地下水の流れについては、人工バリア内での地下水の動きは極めて遅いことから、主な物質移行機構は拡散になるので、ガラスの表面での核種の溶解度、及び緩衝材中の分散係数が物質移行評価の重要な評価パラメータである。ガラス固化体、及び人工構造物の特性は人工的に決まるため、この部分には不確定要素は少ないが、浸出評価における不確定要素は主に自然環境の不確定性による。従って、長期に亙る地下水流れを視覚化し、前述した化学過程を重ね合わせる等の表現技術の開発が対話システムに必要である。それにより、非常に緩慢な拡散中心とした流れに添って反応する過程の視覚化表現が期待できる。それは、核種の放出と輸送の過程をシミュレーションすることを目標としている地層処分ヴァーチャル・エンジニアリング・システムの対話機能開発の重要な課題と考えられる。また、その手法は、次項で触れる地層中での核種移行評価についても同様である。

（3）地層中での核種移行評価

地層中での核種移行評価も基本的には核種浸出評価と同じ構成である。しかし、地層中においては評価上最も重要な部分は地下水の動きである。特に、破砕帯等の母岩に比べ極端に水が通りやすい、いわゆる水みち（水理）の分布である。また、母岩の間隙水は一種の貯留タンクであり、母岩の中への分散速度も重要な評価要因である。従って、地形、地層構造、母岩及び水みちの透水性、母岩中のイオンの分散係数が主なパラメータであると考えられる。

地層中での化学反応モデルでは、可逆反応モデルと鉱物中に取り込まれ、実質上、溶液系から不可逆的に取り除かれる鉱物化モデルの組み合わせで考えることができる。原子炉によって著しく溶解度の異なるNp，U，Trにとって還元性物質の存在は評価上重要である。

即ち、溶解度、鉱物化の反応速度、還元物質の含有量とその還元特性が重要な評価要因である。これらの要因にまで立ち入った情報を自然環境で得るには、更に反応の種類、基礎データ（熱力学データ、反応速度等）に関する研究が必要であり、評価環境を想定して必要な吸着分配係数、溶解度を実測し、より汎用性のある基礎データとしてまとめていくことが必要である。

ソフト面では、PHREEQUE等で代表される多成分系での核種の安定な存在状態を求める計算コードの開発を進め、更に、反応速度、濃度依存性等を考慮した吸着移行評価コードを充実し、組み合わせていく必要がある。データベースについては、熱力学データやび吸着係数の収集が考えられる。

（4）生物圏中の核種移行評価

生物圏に到達した放出核種が人体に至るまでの過程は、原子炉、再処理施設等の地上施設から放出される核種の挙動と類似であり、そのモデル化及び関連パラメータについては、多くの研究が行われている。そのほとんどが適用可能と考えられる。ただし、TRU等の長半減期核種に関する情報、データはまだ十分ではないので、この分野に関する一層の研究が望まれる。従って、生物圏への評価に備えて、核種の崩壊等のデータベースや移行率、量等の作図等が機能に求められる。

4．1．6　システム操作から見たパラメータの不確定性について

地殻変動、地震等の接近シナリオと呼ばれる範疇に含まれる自然現象は、確率的事実であり、いつどこで発生するかを決定論的に予想することはできない。しかし、過去の発生の統計とプレートテクトニクス仮説から、ある程度推定することができる。その手法として、第四紀における地殻変動の統計的データを使い、長期評価に適合するデータを得る。これらの現象による大きな影響は、サイト特定で避けられるが、地震や雑続的隆起沈降の影響は考慮しなければならない。これらは、地下水による核種移行シナリオに影響する不確定性である。従って、対話システムには、将来これらの地殻変動や地震等の事象を与えられる機能が求められる。

地層の透水性、地下水の組成、岩石の組成等、地層の特性は、原理的には処分サイトが決定されれば測定によって求めることができるが、サイトを破損しないで行う測定精度には限界がある（掘削影響）。そこで、一般的に、類似地層での観測結果の統計処理により、それらの特性値を求めざるを得ない。具体的候補サイトが決まっていない現時点での評価では、地層の大分類の特性の統計的扱いをすることになると考えられる。従って、対話システムは、仮想のサイトにて評価シミュレーションを進めるための情報として、前述の地下水や岩石の組成等、幅広く情報をデータベース化して参照を可

能とすると供に、地形のオブジェクトや設計する処分システムを仮想環境として、コンピュータ中にそれらデータベースと関連させて構築できるものであることが望ましい。それらを構築する上での不確定な事項として下記が上げられる。これらの不確定さの原因については、当面のあいだ事実情報を詳細に記述するデータベースを構築して、優れた対話機能により随時評価することが望まれる。

（1）人工バリア

材料及び施工等の品質管理上の不確定性がある。地層処分システムでは、地層特性の不確定性がそれに比較して大きいので、その不確定性は無視できる。更に重要なことは、人工バリアの健全性の不確定性は、環境条件の不確定性に起因すると言うことである。特に経時変化が重要である。例えその経時変化が予測できても、長い時間の経時変化であれば、その条件での実証試験ができなく、モデルを使った予測に頼らざるをえないと思われる。

（2）生活環境

遠い将来の人類の居住状況を予測することは非常に困難である。そこで、現在の社会が変わらないものと仮定する。従って、サイト固有のデータを使うことは難しいであろう。陸地か海洋か、砂漠か農耕地か、井戸の利用か川の利用かと言う程度の区分で被曝経路シナリオを作り評価することになると思われる。

（3）測定精度

以上のように、パラメータそのものの特性として幅を持っている。また、測定技術が不充分である場合、技術はあってもサイトの損傷を避げるために使えない場合がある。後者の場合は類似サイトを詳細に調査することでその精度を高めることができる。

（4）長期の不確定性に対する保証

長期の事実を限られた期間で実証することはできない。それを補う一つの方法としてナチュラル・アナログ研究がある。評価事象の機構のモデルを作り、その事象のナチュラル・アナログを探し出し検討することにより、実証的データを得る。この方法の難しさは、それが置かれた環境条件の経時変化を把握することが困難である点である。

4．1．7　システム操作から見た総合評価の考え方

前述のように、処分システムの安全性評価の判断基準は、処分に起因するリスクが社会的に許容されるリスクレベル以下であるかどうかにおかれるものと考える。そして、この判断のために総合評価が考えられる。総合評価は2種類あると考えられる。

第一に、代表的な（適切な）処分シナリオについて適切な移行シナリオを選定し、決定論的評価を行う手法である。決定論的評価に於ては、シナリオが適切かどうか、評価結果の信頼性が十分かどうかについて議論がなされるであろう。それは長期にわたる諸条件の変化やモデル・パラメータの持つ不確かさを考えねばならないからである。そのためには、多方面に渡る情報を評価と供に参照しながら作業をすすめる必要がある。このような決定論的評価を検証するための情報を効率的に提供するこ

とは対話システムの重要な機能要件である。

もう一つの方法に確率論的評価がある。それにより、処分システムの安全性の規制的観点からの評価は、決定論的評価手法によって行われ、確率論的評価手法は、決定論的評価を補完するものとして位置付けられると考える。確率論的評価手法は、放出核種の移行過程の変化確率、及び移行過程を支配するパラメータの自然的、社会的条件の変化に伴う変動、または情報の不十分さによる不確かさによる最終のリスク評価結果の不確実さを定量的に解析するものである。

更に、評価結果の不確実さに大きく寄与するパラメータを明確に示す機能、いわゆる感度解析機能も持つものであると考えられる。また、確率論的手法は、シナリオを簡単にネットワークで画像表現し、操作を視覚ベースで行うシステム化の手法を提供するものとしても有効と思われる。

例えば、評価モデルとしては、廃棄物からの核種の浸出は一定割合で一定期間続き、緩衝材中の移行は拡散のみに支配され、地層中は一次元吸着平衡を保ちながら移行し、岩盤中の間隙水へ拡散し、飲料水として人間に到達して放射線被爆をもたらすという簡単なシナリオである。確率論的評価手法の開発では、手法そのものの開発とともに、入力データーベースの開発も重要な問題である。そして、入力データの確率分布をできるだけ正確に知ることが必要である。

4．2　システム操作から見た作業の不確かさについて

本システムを操作して性能評価シミュレーションを行うにあたり、最終的な目標となる生物圏からの放射性核種の隔離を多重バリアが機能として確保する過程の作業に下記の人的な不確かさが含まれると思われる。それらは、シナリオの選択と内容に係わる事項や、数学モデルの数式化と計算等、凡そ以下に示す領域に含まれる不確かさである。

シナリオの選択と内容
処分システムの将来における状態
評価モデルと計算（数学モデルの数式化と実際の計算）の数学モデルの数値解析上の不確かさや、プログラミングの不確かさ、精度計算上の限界
計算パラメータや熱力学データ等のデータ自僧に内在する不確かさ
データの不足、測定誤差、専門家の仮定、等

これらは、確率的な変動性による不確かさと知識不足からの不確かさに大きく分類されている。性能評価においては、容器の腐蝕や長期に渡る放射性核種の地層から生活圏への移行など、将来に亙る条件についての知識不足に依る不確かさが重要な研究課題となる。

また、不確かさを見積もる確率論的不確かさ評価は、主観的な確率を用いて行われていると考えられる。さらに、シミュレーション・プログラムの品質やデータの信頼性等、ソフトウェアに含まれる不確かさは、地層処分ヴァーチャル・エンジニアリング研究のような、情報処理に基盤をおいた性能評価にとって重要な課題となる。従って、ソフトウェアの不確かさは、前記の２つの不確かさ以外の第３不確かさと考えられる。

不確かさの評価は、サイトの選定と貯蔵所の設計研究などに用いられ、システム性能評価の結果に

おいて、予想の信頼性と精度、及び信頼度を示すことに特徴がある。確率論的不確かさ評価は2種類の情報を同時に与える。それは主観的な信頼性の有る期間と不確さのランキングである。前者は、サイトや貯蔵所の候補間の比較に要求されるもので、後者は、線量など評価値における不確かさに対して最も寄与する定量化された不確かさのランキングを示すものである。また、ソフトウェアの不確かさについては、別途複雑度解析等の定量的ソフトウェア品質評価技術の研究が進められている。

シナリオやモデル化における不確かさは、主に定性的な方法において扱われる。例えば、FEP手法が上げられる。それは組織的に良く文書化された過程が要求される。計算モデルに対する定性的な保証過程の展開的な適用が重要であり、性能評価の結果が規制の目的に使用される場合、特に問題となる。パラメータとデータにおける不確かさは次の方法を用いる際に定量的に扱われる。

- 表面応答解析
- モンテカルロ解析
- 微分解析
- 確率論的解析

不確かさ評価に至る前にスクーリング解析を行うことにより計算時間を最小化できる。不確さ評価結果に含まれる感度情報は、データの不確かさを減らす目的で行う研究のガイドとなる。この手法の採用により、次第に、より複雑な方法を用いる必要性が出てくるにつれ単純性が擁護されている。

例えば統計的サンプリング法がそれぞれ特徴を持って開発されている。

・単純なランダムサンプリング	：	単純で、健全な方法だが、多大な計算時間を要する。
・ラテン超多面体サンプリング	：	単純なランダムサンプリングよりも計算時間は少ないが使用する際に注意を要する。
・インポータンスサンプリング	：	更に計算時間を減少でき、解析をより関心ある領域に向えることができる。

不確かさの評価については、類似分野として、原子力プラント、再処理、廃棄物処理の評価における不確かさ評価で用いられた方法には共通点が見られる。すなわち以下のものがある。

- 入力データを含むパラメータにおける不確かさに関する方法
- 主観的な確率密度関数を用いることによりパラメータの不確かさを表わす方扶
- 単純なランダムサンプリングないしはラテン超多面体サンプリングを用いて確率密度指数からサンプルしたパラメータセットを出力するモデルの計算法
- メジアン、ミーン及び９５％の信頼値により表わされる不確かさを出力する方法

以上のように不確かさは、性能評価シミュレーションにとって重要な対象である。従って、対話システムは、この不確かさについて出来るだけ多くの情報を提供するように設計開発されなければならない。それは、情報の検索、知識ベース等による判断、補足的な数値計算を含むものである。

従って、それらの情報処理機能をもって、およそ以下に示す議論について支援することが求められ

る。それにより、性能評価シミュレーション全体の有する不確かさをスクリーニングする機能が必要である。

議論１：決定論的、確率論的、性能評価法の相対的な役割は何であるか？
決定論的な最良の評価と、パラメータの不確かさについての確率論的な処理法を結び付けて考察することが必要

議論２：処分システムの性能評価を実行する時、確率論的不確かさと決定論的不確かさを区別して扱う必要があるか？それとも合わせて扱うべきか？

議論３：定量化する必要のある全ての源に含まれる不確かさは、どの様な範囲に渡っているのか？
例えば、モデルとシナリオに依る不確かさを定量化する方法を探す必要がある。

議論４：不確かさ解析の結果をどの様にリスク評価や結果の信頼度提示に用いることができるか？

議論５：不確かさ解析に用いる主観的な確率密度関数の定義は、どの様になされるべきか？
例えば直接測定が不可能な時、専門家の判断を用いる場合に問題となる。

議論６：時間に依存するプロセスは、どの範囲に対して性能評価に取り入れるべきか？

議論７：異なるパラメータ間の相関をどの様に見積もることができるか？

議論８：リスクと線量率を評価する時、不確かさパラメータを扱う確率論的手法は規制側による決定を助成することに対してのみ適用ざれるのか？それは、処分に関する安全性のレベルを大衆に知らせるために用いるにはあまりにも複雑ではないか？

以上の議論から、地層処分ヴァーチャル・エンジニアリング・システムの開発においては、その対話システムの支援機能として、以下に示す要求にそった情報サービスを行える機能を開発する必要がある。

要求１：定量的な手法と非定量的手法を結び付けるべきである。

要求２：不確かさ解析の種類と型式は規制側の要求により決定されるべきである。

要求３：決定論的な計算と、確率論的不確かさ解析を結び付けることにより全体的なシステム性能評価を導く一般的なガイドラインとすべきである。

要求４：感度解析は、不確さ解析に対し密度な関係がある。

要求５：異なる不確さパラメータ間の相関を扱う方法の開発が必要である。

要求６：幾つかの二者択一的な統計的サンプリング法がメリットを持っており利用可能である。

要求７：主観的な確率密度関数の形において、パラメータの不確かさの表現がシステム的なアプローチに必要となり、特に主観的な専門家の意見が取り入れられる。

要求８：コードの質的な評価がモデルの不確さを減らすのに役立つ。

要求９：ベンチマークによる検証と確証試験が不確かさを減少させることの助けとなり、モデルのユーザーの信頼度を向上させる。

要求１０：システム的なアプローチは不確かさ解析を指導することに適用されるべきである。

４．３　対話機能から仮想環境へ（ヴァーチャル・エンジニアリングの将来）

コンピュータの重要性は、主にそれが、推論、創造、発見の道具として役立つところにある。対話機能は、その最も中心的な機能である。対話機能は、情報を正しく、解かりやすく、美しく、そして、見る者の創造力を駆り立てる魅力のあるものであることが設計上の重要な考慮点である。優れた対話機能の備えたコンピュータは、私たちの直観を超えた推論を行なうのに役立つ。本項では、そのような観点から地層処分ヴァーチャル・エンジニアリング・システムの将来を期待する上で参考となる、最近の情報科学の動向を外観し、今度の研究開発の参考となる話題をとりまとめる。

対話機能の中核となるグラフィック・ユーザー・インターフェイスは、実にさまざまな視点・視角からの視覚的表現を与えてくれる。対話機能は、音声合成、分子生物学、数学、そして美術まで広い分野で利用されている。それは、凡そ以下の話題について、利用者に期待のできるこれからの魅力的な機能を提供している。従って、グラフィック・ユーザー・インターフェイスは、１９９０年代の後半において、パーソナル・コンピュータから超高速コンピュータを利用できるワークステーションに至るまで実装されており、メインフレームの時代の文字ディスプレイは殆どの利用の場から姿を消している。

- 複雑なデータを表現する、新しいグラフィックスの方法が期待できる。
- 数学における美学の役割を紹介し、見かけ上単純なプロセスに棲む複雑さや美を認識するうえでコンピュータ・グラフィックスが役立つと期待できる。
- 一般に、コンピュータを使った創造的思考の促進することが期待できる。
- コンピュータが、シミュレーションや発見の道具として役立つことが期待できる。

４．３．１　複雑系とシミュレーション

（１）人工知能研究と複雑系（不完全性定理）

１９３１年にゲーテルは、「不完全性定理」を発表した。それは、算術を公理化しようとすると、それが真実であることが我々には“わかる”にもかかわらず、事実である事を形式的には決して証明できない事があることの証明である。以後、この定理は知識工学、人工知能研究等の分野で広く研究されてきた。

- 第１不完全性定理は「ある論理体系において、その論理体系によって作られる論理式のなかには、証明する事も反証することもできないものが存在する。」というものである。
- 第２不完全性定理は「ある論理体系が無矛盾であるとき、その無矛盾性をその体系の公理系だけでは証明できない」というものである。

この定理は、コンピュータの分野で人工知能の実現性と不可分である。この定理の延長線上に人工

知能は作成できないが、「機械にできること」と「人間にしかできないこと」を再考する為の一つの原点ということができる。人工知能の研究においては、人間知性を複製することは、全く機構が違うコンピュータというハードウェアでも、知識のルールさえ抽出できれば実現できると考えることを前提にしている。そのような論点から、過去の人工知能開発は、チェスゲームなど限定された範囲では成果をあげることができたが、人間の日常は、このような限定範囲にないことも研究の結果明らかになった。それは、ほとんどの人間は「暗黙知」という莫大な背景知識をもっていて、これが特定ルールに影響を及ぼしていることが明らかになったためである。人工知能賛同者は、心理学者、数学者を中心に展開したが、反対論は哲学者たちによって展開された。そうした反論は、現象学か反行動主義か、ゲーテルの定理のいずれかに基づいている。

例えば、自動車を運転するために、初心者は、どのスピードになったらギヤをどうするかなど、いろいろ考えないと走れない。一般のドライバーは、そのそれらの判断を素早く処置することができる。プロのドライバーとなると、もはや直感的理解によってのみ疾走している。また、中国語を全く知らない人に中国語の辞書を持たせ、ある部屋に閉じこめるとする。その部屋に外部から中国語のカードが差し入れられ、中の人は辞書を引き、その回答を外部にカードにして差し出すとしよう。このときに外部からは、部屋の中の人は中国語を理解しているように見えるが、実際にはそうではないはない。

スタンフォード大学の人工知能グループは、「エキスパートシステム」からの研究を進めていた。例えば、テレビの故障個所を特定するような１ステップづつの積み重ねの知識ルールを、それぞれのデーターベースにセットするといったものであった。これが有効と思われたのは、刺激→反応型の情報に対してであった。もし、システムが、あらかじめ何をなすべきかプログラムされていたら、それはプログラマーの知識であり、意識ではない。それを人工知能と呼ぶのは欺瞞であるという判断が反対論者の考え方である。いずれにしろ、論理記述方式等の従来研究されてきた知識ベースや人工知能のシステムは、データベースの拡張と考えられる。

（２）複雑系

複雑系とは、複雑なものを単純化せずに受け入れる、細かく分割しても単純にならないか、もしくは分割することにより、問題そのものの価値が失われるような系に対するアプローチである。それはまた、要素（エレメント）が集まってできた１つの系（システム）のうち、特に要素が集まったときに「新しい性質や、能力が生まれる系」のことと言うこともできる。さらにそれは、現代のオブジェクト指向等の最新の計算機科学を背景に、仮想世界におけるマルチ・エージェントが協調するシミュレーション技術により急速に進歩している理論である。従来、還元主義は完成物からそれを要素分解していくトップダウンの発想であるが、逆に要素より全体を把握できるか（未来を予測）を問うアップの発想も必要である。それらについては、現在では複雑系として、要素よりの予測をしても結果を推論することは不可能であるという結論を出している。したがって、結果から原因を予測することも推論にすぎないとしている。

数学的にはカタストロフィー、フラクタル、カオスなどに支えられ（ファジーやニューロもそこから派生したものである。）物理学、生物学、経済、社会の分野に広く応用しようとしている。複雑系のアプローチは、今後の課題である地球環境変動が、社会システムに与える影響等を評価する有効な戦略と考えられる。それは、従来の要素還元論的科学の方法とは別な方法を提供する。例えば、ラプ

ラスはニュートン力学の集大成として「天体力学」をまとめた。ここでは、宇宙のすべての物質の初期位置と初速度がわかっていれば、すべては予測できるということが述べられていた。ハイゼンベルグは「不確定性原理」を表し、それを否定した。それは、初期状態は決定できないということ。それ故「不確定性理論」でなく「原理」と言われている。

また、複雑系の代表としてとりあげた脳を持つ人間を要素とする、社会という系もまた複雑系と考えられる。社会は人間一人一人が持っている性質とは異なった性質や能力、機能を持っている。複雑系では、それを構成する要素に「要素の振る舞いのルール」を与えたときに、その要素から構成された系に「系の振る舞いのルール」が自発的に生じ、このルールが、要素にフィードバックされる。このフィードバックも、複雑系にとって非常に重要な振る舞いの一つである。これを人間と社会に置き換えて考えると次のようになる。人間は、はじめ「個人として振る舞うルール」を身につける。そして複数の人間が集まると社会が作られ、そこには「社会のルール」が出来上る。そして、その「社会のルール」は「個人として振る舞うルール」に影響を与える。つまり、社会のルールに適応できるように、自分の持っているルールがどんどん更新されていくのである。それは、社会に対する適合のための行動シナリオの変化として考えることができる。このように、複雑系では、系としての振る舞いのルールが自発的に生まれる。これを「創発」といい、複雑系では必ず見られる特徴である。

例えば、核燃料サイクル施設は、建設・運転プロジェクトとしてとらえると複雑系である。それは、要素としては個別の機器ごとに安全性設計というルールにそって開発されるが、プロジェクト全体の管理面のルールは、建設運転費用が妥当であり、地域住民の了解が得られることと言う「社会のルール」がある。そして、その「社会のルール」は安全性設計というルールに大きく影響するであろう。

複雑系は、どのようにして「創発」するのだろうか？例えば脳を例に考えてみても、細胞が複数個集まることによって、ひとつの細胞が持たない性質・能力が生まれるのであれば、その系を構成している要素そのものよりは、その（この場合は細胞と細胞の）関係に創発の仕掛けが隠されていることは容易に想像できる。要素と要素が、お互いをどのように関係付けしているのか、そしてどのような情報が交換されているのかが非常に興味のあるところである。

今から50年ほど前に、ウイナーは脳や通信・自動制御を取り扱うための理論として、サイバネティックスという考え方を提唱した。それは「すでに決定された変数と決定されていない変数のうち、目的の結果を得るために決定されていない変数を適切に制御する」という考え方のようであるが、そこでは、フィードバックと情報が重要な意味を持つ。また、ウイナーは微粒子の熱によるランダムな運動の研究から、そこに存在する必然的におこる現象と、偶然におこる現象のそれぞれの役割の検証が重要だと指摘している。ところがカオスによれば、見かけ上偶発的な現象のなかにもルールが存在する可能性を説明することが可能になりうる。複雑系である多くの「系」の中に、カオスを見つけることができる。システムアナリストがカオスという道具を手に入れたことは、サイバネティックスの時代から大きく変化した点であり、複雑系を理解する上で重要なきっかけになると考えられる。複雑系研究で知られている米国サンタフェ研究所では、以下に示すテーマで研究が進められている。

* 適応コンピュテーション
* 人工生命
* 生態の分析と総合への複雑適応系のアプローチ
* 複雑性、エントロピ、情報物理学
* コンピュテーション、ダイナミクス、推論
* 遺伝的データへのコンピュテーションアプローチ
* 複雑適応系としての経済
* 複雑適応系の基盤
* 複雑適応系としての人間社会
* 統合に関する中核的研究
* 理論的免疫研究プログラム
* 自己創造的、自立的エージェント

このうちのいくつかは、将来の生活に欠かせない要素技術になると思われる。研究テーマによっては、経済など、応用範囲が明示的に示されているものもあるが、そうでないものは、その応用範囲をいまの時点で限定するのは得策ではないと思われる。また、応用範囲を限定して研究しているプロジェクトの中からも、また違った分野への応用が期待される。いずれも、研究は「科学」の領域で行われているが、これらの要素技術が工学に応用されることが期待できる。

（3）カオス

行動の決定論的規則は、完全に予測可能な事象を生じると考えられてきた。カオスは、１９６１年ローレンツという気象学者が発見した数学モデルであった。彼が、天気予報を正確に予測しようとして、コンピュータで3回予測したところ、3回ともまったく違う答えが出たところから疑問は始まった。追求の結果、最初の予測は、少数点以下が6桁であるのに検算は小数点以下を3桁で計算していたことが原因であることが分かった。

従来、数学では小数点の違いは、予測の精度の違いぐらいにしか影響を及ぼさないと考えられていたが、最初のわずかな違いは全く違う結論を導き出す。これを初期値敏感性という。この現象を3つの連立方程式にまとめたのが、最初のカオスモデルである。

例えばD＝C/（A-B）を計算してみよう。A=100,B=99,C=100ならD=100となる。ではAの値を1%だけずらして101としてみる。するとD=50に半減してしまう。もしもAが99ではどうだろうか？これは良く知られたカオスである。その後１０年の後、ウィーナーによってカオスと命名された。

初期条件を与えれば未来予測が可能とするニュートン力学を大きく覆し、決定論の対局にこのカオスは位置するものとなった。数学的には決定論に従う。しかし、観測や数式によるあらゆる未来予測、構成予測は観測機械の精度や、小数点何桁まで数学モデルに組み込むかで近似値ではなく、全く違う結果が出されるのでは根源探しも同時に意味をなさなくなった。これは自然界における普遍的な現象という認識とされる。一般語では「メチャクチャ」にとらえられることがあるが、これは間違いである。科学においては、カオスとは基本的には決定論であり、ただ、初期値敏感性が全体に複雑に無限に影響を与えるという意味である。

カオスは科学の新しい領域を切り開きつつあるといえる。いままでの科学は、物事を単純に分解してきたが、この新しい科学の姿勢は、複雑にどこまでも複雑に関連する領域を広げていく。その有効な道具はコンピュータである。コンピュータは3つの側面で複雑系に貢献している。一つは、シュミレーションである。これによって、今まで計測不可能だったことが次々に解明されてきている。もう一つは、カオスの実体を生成している点である。インターネットを代表とするネットワークはカオスそのものである。

（4）カタストロフ

「原因がわずかずつ変化すると、結果も正比例的にわずかずつ変化する」だろうか？急激な変化によって、物事が不連続的に移りゆく様子を数学的にあらわすため、フランスの数学者トムが唱えた理論。カタストロフィーとは、大変動、破滅といった意味である。

自然界でも社会でも、日頃おだやかに推移していたものが、突然劇的に変化するという例は多い。たとえば、金属が何度も流体の圧力を受けたり、手で折りまげられたりすると、あるとき急に折れてしまう。地殻をつくっている岩板（プレート）は、ふだんは非常にゆっくり移動しているが、プレート同志で押し合う歪みが限度をこえたようなとき、はげしい動き（地震）を引き起こす。生物の体の中で少しずつ変化がおこると、あるとき幼虫は姿形を劇的にかえて蝶に変身する。こうした現象をあつかうには、滑らかな変化を解析してきた微積分の手法だけではなく、物事が不連続的に移り変わるパターンを調べるため、トポロジーをはじめとする現代数学のさまざまな手法が必要になる。

カタストロフィー理論は、1968年にトムが発表して以来、生物学や社会科学を含む多くの分野の研究者の心をとらえてきた。ダーシー・トンプソンは、１９１４年に「成長と形」のなかで、魚を方眼紙に描き、そのグリッドを変形させると近縁のいろいろな魚が描けることを示した。そしてこの手法を発展させるとカニの甲羅と人の頭蓋骨も同じでアーキテクトをもっていることも示した。この研究は、有機体の全体的な設計図はトポロジーを基礎としていることを強く示唆している。有機体の固有の幾何学的形状は後になってそれを埋めていくために、有機体は進化の過程での変化がたやすくなる。当面の大問題はどのようにして有機体の基本的トポロジーが特定されるかである。形態形成は、単に正しい種類の細胞が成長する胚の適切な場所に集まって生じるのではない。実際、発生の初期段階では、細胞の種類は通常決定されていないのである。

汎用型の細胞が、筋肉細胞になったり脳細胞になったりするのはなぜなのか。このことに最初の回答をだしたのがコンピュータ科学者アラン・チューリングであった。１９５２年にチューリングは、様々な化学成分の反応と拡散の過程に関する数学モデルを提唱している。そうした化学成分を「形態形成素」（モルフォゲン）と呼んでいる。さらにこのアイデアを洗練発展させていき、各細胞の最終的な運命はさまざまな形態形成素の濃度によって決定されるという現在の共通理解をもたらした。

発生生物学は、その背後にある数学的モデルを書き下すには非常に難しく、数学と哲学を間の領域にあるように思われがちだが、今ようやくこのプロセスを記述できる可能性が認められたのである。まだ洗練の余地があるが、それでもメカニズムが解明されないからといって、知識の探求を怠ってはならないのである。

物理学から生物学へのシフトは、物理学から半物理学（セミフィジック）へのシフトとしてとらえられる。生物学は物理学の一部であるとされた一部の還元論者から、生物学の一部が物理学であると

のパラダイムの転換が起こったのである。

（5）人工生命

１９９０年トム・レイ、ティエラは、簡単なルールによる電子生態系であったが、時間がたつにつれて驚くべき複雑な結果を生みだした。これは原子のスープからティエラの実験結果が導き出された答えは、進化の過程は特定の物質から独立したもので、生物のような炭素系生物のルールはプログラムというメモリのなかでもたやすく起こりうるということである。

この進化論的実験結果をゲーデルは肯定している。つまり、人間知性と等しい機械が作れるかもしれないとしている。しかし、ゲーデルはそのような機械が存在しても、我々はそれを理解できない。構築するものではなく育てるもの。もし人間と同じと認められる知識生命体を作り出したとしたら、ただちに人権の問題に至るだろう。還元不能性複雑なシステムは、より単純な部分へと分解することで理解できる。本当だろうか、従来の科学ではどんどん部分に分解して、これ以上分解できないところまでいくと、もうそれだけでその全てが解ったつもりになっていた。部分を知り尽くしても全体とは全く別のものである。

４．３．２　マルチエージェント・シミュレーションによる仮想世界の観察

（1）仮想世界の再現

８００年前に栄えた南西アメリカ・インディアンの文明であるアナサジは、考古学者たちにとって深い神秘に包まれていた。アナサジのように繁栄の絶頂にいた文明が、一夜にして消滅するとはどういうわけなのか？気候パターンとか作物収穫高とか、それ以外の環境条件について入手可能なデータは数多くあるものの、考吉学者たちはこの難問に対する答えをつかみあぐねている。その主な理由は、時代をさかのぼって再び“実験する”ことが不可能だからである。

つまり、無数の要因のうちの一つか二つを変えて、アナサジ族の移住行動にどのような影響を与えたかという問題を検証する実験はできない。この問題は、社会現象を研究する者すべてにとって基本的なジレンマである。歴史はたった一度だけ行われた実験である。それとも再び実験できるのか。

本当に次のような条件をコンピュータの中に作り出せると考えられるのだろうか。例えば、アメリカの南北戦争で南部連邦が勝っていたらどうなっていたか。また、１９７１年に原油の価格が４倍にならずに２倍にとどまっていたら何が起こっていたか。こうした状況をコンピュータで再現できるのか。ワシントンD．Cにあるブルッキングス研究所の研究員、ジョシュ・エプシュタインとロブ・アクステルは、この実験が可能だと考えている。

物理や工学の理論を作ったり、理解したりするとき、自然科学者は再現可能な実験を当然のことと考えている。再現可能な実験を社会科学の分野で行うために、エプシュタインとアクステルは、ゼロから社会秩序を‘育ててみる’ことにした。

常に変化する環境や、互いに相互作用するエージェント、さらには生存のための単純な規則にふさわしい環境、こうしたものを作り上げることによって実験を開始した。そうすると、社会全体の構造（取引、経済、文化）は、エージェントの相互作用から発展してくるに違いない。

社会問題についてエプシュタインは次のように述べている。「それを解決するのではありません。それを発展させるのです」と。エプシュタインとアクステルは、社会が発展していく実験場をコンピ

ュテラリウムと呼んでいる。

（2）ルール

エプシュタインの造った仮想環境で相互作用するエージェントは、シュガースケイプ（砂糖の風景）と呼ばれる居住環境の中で、それぞれ色の付いた点によって生き生きと表現されている。

この世界の至る所に、時間とともに濃度が変化する砂糖と呼ばれる食糧が置かれている。また各個体は、次の一組のユニークな特徴を備えている。性別、食糧を探知する視覚領野、代謝率などの変化しない特徴と、健康、婚姻関係、富といった変化する待徴である。

こうしたエージェントの振る舞いは、一組のかなり単純な規則で決定され、さらにその規則は、生存と繁殖のための常識的なルールで構成されている。一組のルールの代表例は次のようなものである。

- 砂糖がある最も近い場所を見つけよ。そしてそこに行って、代謝を維待するために必要な分だけ食べ、残りを蓄えておけ。
- エネルギーとその他の資源を十分に蓄積したならば、繁殖せよ。
- 異なる種類のエージェント（“部族”）に取り囲まれなければ、現在の文化のアイデンティティ（自分の待徴）を維持せよ。もし取り囲まれたら、そのエージェントと連合するように自分の特徴ないし好みを変えよ。

この程度の素朴なルールでも、実に奇妙で不思議なことが起こり始める。このシュガースケイプの世界において、砂糖は責色の点で表現されている。また、最初エージェントはランダムに分布している。

赤の点は、遠くから食糧を見つけことができる優れたエージェントであり、青の点は、ずっと近くからしか食糧を見つけられないエージェントである。もしほかのことを考慮しなければ、次の予想は合理的である。自然選択によれば、優れた視力を待っているエージェントがいつでも有利であろう。

実験（シミュレーション）の結果、それが事実であることがわかった。それは、集団の中で赤い点が優勢であることを示している。さらに、子孫に富（ここでは砂糖のこと）を相続できるようにして再び実験すると、その相続が明らかに彼らの生存に影響することがわかる。これは視力の劣るエージェントが、親から相続された砂糖を利用して、より数多く生存できている。

（3）経済学モデル

以上の単純な例は、コンピュテラリウムの活動を説明するには役に立つが、しかし社会構造について考えたり研究したりする方法について変革をもたらしてはいない。そのためには、システムにもっと多くの手を加える必要がある。エプシュタインとアクステルは、まさにこれを実行した。砂糖の濃度が周期的に変化するようにシステムに季節を付け加えると、エージェントたちは移住を開始したのである。

さらに第二の資源として香辛料を導入すると、次の新しいルールによって原始的な経済が出現した。そのルールとは、“自分が必要とする品物を持っている隣人が周囲にいないか探せ。もし見つけたら、

互いに満足する価格になるまでその隣人と交渉せよ。そして交渉がまとまったら、その価格で取引せよ”である。実験（シミュレーション）の結果、この原始的な交換経済でもある実験を行うのに十分な構造を備えている。その実験とは、経済学の中で最も大切にされた理論の一つ、いわゆる効率的な市場という仮説を検証するものであると考えられる。

（4）人間らしさ

古典経済学によると、ある経済において、すべての品物の価格は一定の均衡水準に落ち着くはずである。その経済の下では、どのエージェントも好みは変化しない。また彼らの寿命は十分に長いので、品物の需給についての全情報を処理することができる。言い換えると、その経済で価格に不均衡があれば、すぐに、さや取り売買（時間や場所の違いによって生じる価格差を利用して、利益を得るために行う取引）が行われるだろう。これは、効率市場仮説を変形したものである。その説によれば、入手可能な全情報は品物の価格にすぐさま織り込まれてしまう。

例えば、砂糖が不足しているなら、その事実は光のスピードで伝わり、砂糖の価格はすぐに跳ね上がるだろう。この仮説をテストするためにコンピュテラリウムが使える。テストを始めるに当たって、従来の経済学の要請から、以下の点をエージェントの属性として仮定する必要がある。

それは、個人の好みの不変性と情報を処理するための無限の寿命である。すると、古典経済理論から、すべての品物の価格が一つの決まった値に落ち着くと予言できる。次に、システムを元の状態にリセットし、エージェントに人間の性質を与えなければならない。人間の性質とは、すなわち有限の寿命、好みを変える能力、そして自分たちとは異なる好みを持つ子供を作れることである。このずっと現実的な仮定の下では、価格は多かれ少なかれランダムに揺れ動く。

より現実的な経済では、取引価格が唯一の水準に落ち着くことは決してなく、むしろランダムに動くように見えるが、かといって全くランダムというわけでもない。いつまでも価格はランダムにフラつき続けるように観察される。しかし、このフラつきは、ある一つの価格水準のまわりをただ上下しているだけである。この水準とは、効率的な市場の仮説が機能する理想的な仮定の下での均衡水準である。これは、効率的な市場という見方が、現実的な市場に単に制限を加えたものにすぎないということを強く示唆している。現実的な市場とは、古典経済理論の厳格な仮定を弱めたものである。たぶん、仮定が現実的になればなるほど、効率的な市場での均衡水準のまわりで価格は大きく変動するだろう。

エプシュタインは次のように指摘する。「もしエージェントがもう少し人間らしく振る舞えば、市場が教科書どおりに動くと仮定する根拠は全くありません。」このように、コンピュテラリウムを使って経済理論を検証できるのは効率的な市場の場合にかぎらない。

（5）富は公平に分配されるか

経済生活におけるもう一つの根本的な疑問は、富の分配についてである。ある一定の条件下では、社会全体における富の分配はだいたい公平になるのだろうか。それともエージェントは、いわゆるパレート分配（資源を最適に分配すること。イタリアの経済学者パレートの名にちなむ）を自己組織化するのだろうか。パレート分配では、ほんの少数のエージェントが全体の富のうちで大きなシェアを獲得する。

公平な分配、あるいはパレート分配のどちらが起こるかは、エージェントが使う個々の取引規則や、相続、移住、資源の再生などを定める規則によって左右される。そこで、これらのパラメータをいろいろと変化させて実験したところ驚くべき結果が現れた。最終的に富が公平に分配されるような条件を見つけ出すことが窮めて困難であることがわかったのである。

なぜ平等を目指す社会主義的な理想の実現がそれはど難しいのか。この実験結果は、それに理論上の理由があることを示唆している。要するにそうした理想は、基本的な人間の性質とあまりよくなじまないのである（少なくともその性質を経済の規則の中で具体化する限りにおいては）。

エプシュタインとアクステルが築いた（社会の）実験場からは、確かにたくさんのことがわかるだろう。文化集団や戦争や制度などの発生について、社会科学者の間ではさらに議論が問わされるかもしれない。すなわちコンピュテラリウムは、ボトム／アップ式に社会を研究する実験例を提供しているのである。このような実験により、局所的には単純だが、ダイナミックに発展する社会の動きを探究できる。

（6）試験的に開発されている仮想世界の評価

良いモデルと悪いモデルを区別するいくつかの特徴として次のようなものが考えられる。

- **忠実性**　：私たちの疑問に答えられるようにどれだけモデルが現実を表現しているかという程度。
- **単純性**　：変数の個数、サブシステム同士の相互作用の様子、特別に仮定された前提の個数について、そのモデルがどのくらい‘小さいの’という観点から判断したモデルの簡潔さ。
- **明瞭性**　：モデルと、そのモデルが与える予言と説明とを、理解するときの容易さ。
- **脱偏見性**　：モデルが、モデルの意図や目的とは何の関係もないモデル作成者の偏見から、どのくらい解放されているかという度合。
- **扱いやすさ**：モデルを用いて予言や説明をするために、必要なコンピュータ資源のレベル。

以上の概念を具体的に検討するために、現在開発されている仮想世界を評価してみる。

TRANSIMS：交通網シミュレーション・モデル

忠実性という点で、アルバカーキの道路交通網シミュレーション・システムTRANSIMSは、どの仮想世界と比較しても忠実に現実を反映するモデルとなっている。アルバカーキの道路交通網の詳細さ、システム中を移動する人々の統計、交通の流れの詳しい解明、これらがほぼ比類なき正確さで表現されている。そこで、TRANSIMSは忠実性について高得点となる。

あいにく、アルバカーキの道路交通網のTRANSIMSモデルは単純と言うには程遠い。少なくとも、そのモデルが何十万という変数（エージェント）を必要としているからである。しかし、ネットワーク内で移動する各個人の行動様式は、互いに影響を及ぼす局所的な相互作用も含めて、全く単純かつ容易である（たとえば“まっすぐに行け、左（右）へ曲がれ、止まれ”など）。そこで単純性の基準

に立てば、TRANSIMSは“中程度の単純さ”くらいに評価されよう。明瞭性の点では、たしかにTRANSIMSは高得点となる資格がある。モデルが画像を出力してくれるので、交通パターンとか混雑や流れの密度などの様子が画面上にハッキリと見て取れるからである。その上このモデルは、システム中を移動する各個人に関する非常に詳細な情報と、彼らが各瞬間に利用するさまざまな規則をも私たちに教えてくれる。したがって、このモデルの明瞭性は卓越していると言ってよい。ところで扱いやすさという話になると、TRANSIMSは、いささか優れているとは言えなくなる。

必要な計算を実行する際、このモデルに理論上の障害は存在しないが、移動する人々を追跡して表示するのに必要な計算量は真大なものである。実際このモデルを動かすには、超並列コンピュータの代表格であるコネクション・マシン5の計算力を必要とする。最後にTRANSIMSには、研究者の偏見が全く入っていないと思われる。

モデルにプログラムされているものといえば、ある環境の下で各個人が車を運転するとき、彼らがどのように反応するかを定める単純な規則だけである。ほかのことは、すべてこれらの規則が相互作用した結果、出現するのである。したがって、脱偏見性の点でもTRANSIMSは高得点となる。

ティエラ：生物進化シミュレーション・モデル

ティエラを開発したレイによれば、ティエラは現実の生物進化を模倣する厳密かつ詳細なモデルを目指したわけではない。むしろその目標は、進化とは何かを説明するのに貢献することである。そのためには、どのように遺伝子が進化するかについて、最も際だった特性（突然変異、遺伝交差、選択）を選ぶことが必要である。

ティエラでは、このような特性を模倣しようとしたが、現実の遺伝子についての詳細な特徴とか、時間経過に伴って現実の遺伝子が変化する様子などはすべて無視されている。したがって、現実の遺伝子に対するティエラの遺伝子の忠実性は、せいぜい中程度である。コンピュータ遺伝子を作り上げる際、レイが行った抽象化によって、非常にシンプルで明快なモデルができ上がった。

これは、モデルが進化のプロセスに対して鋭い洞察を与えた割には変数の個数が少ない、ということを意味している。もちろん、モデルには可変なパラメータが数多くあり、そのパラメータが変われば、遺伝子が連応しなければならない環境や遺伝構造も変化する。

結局、構造の単純性とか結果の明瞭性という点では、ティエラはかなり評価してよい。さらに、ティエラの扱いやすさについても同じことがいえるだろう。このモデルを動かすために、単純なＰＣから高性能のワークステーション（さらにはインターネット上の分散システム）にまで及ぶ、さまざまな計算機が用いられてきた。

ティエラによって明らかにされた進化の特徴は、ほとんどの人がアクセス可能な低機能の計算機でも発見できる。その上、実験に要する計算はそれほど多くないので、異味深い結果を手に入れるのに何日も何カ月も待つ必要はない。せいぜい数分（低機能の計算機の場合は数時間）くらい待てば十分である。ティエラでは研究者の偏見が一つある。それは、遺伝子集合が進化する際に、何が重要で何が重要でないかについて、レイが抱くプロとしての偏見である。しかし、レイが社会や経済や政治について何を考えているにせよ、その偏見がティエラの働きに影響したとは思えない。

そこでTRANSIMSと同様、ティエラも偏見を免れたモデルと見なしてよい。

シュガースケイプ：文化現象の進化モデル

エプシュタインとアクステルによる文化現象の進化を扱ったシュガースケイプというモデルは、ティエラよりもさらに比喩の度合が強くなる。私たちは、一袋の砂糖や一ビンの香辛料くらい、どこの家でも台所のどこかに置いてあると思うだろう。しかし、少し想像してみれば、こうした品物を台所の棚に並べるために私たちは日々働いているのである。そのようなさまざまな品物を比喩的に表現したものである。従って、厳密に言うと、このモデルの忠実性は非常に低い。この事実から次の疑問が出てくる。

シュガースケイプにおける経済ルールの発展や、“国際的な”取引などが、どの程度まで現実社会を反映しているのか。もしも反映しているならば、エージェント同士の非常に単純な相互作用から、現実の行動が出現しうることをこのモデルは示している。そこで、経済過程のモデル作りに興味のある人は、このシュガースケイプの結果から次のことがわかるだろう。

このモデルにおける有益な出発点とは、単純な規則を考えて、その相互作用の結果がどうなるかを観察することである。シュガースケイプのモデルでは、忠実性が低いかわりに単純性の度合が高まる。なぜならば、エージェントが砂糖と香辛料を手に入れようとして用いる規則も、彼らが及し合う相互作用も非常に限られているからである。エージェントの行動を把握し、なぜそのように彼らが振る舞うのかを理解するうえでこれは大きな強みとなる。

さらに、シュガースケイプで起こる出来事は非常に明瞭である。エージェントの行動は、この二次元世界で、いとも簡単に画像で表現されているからである。計算上の扱いやすさについても、シュガースケイプは高得点がつく。ちょうどティエラと同じ様に、このモデルは、PCやMacintoshのような低機能のパソコンで動くよう設計されている。したがって大方の人は、数分以内でシュガースケイプの実験を行うことができるのである。

（7）仮想世界を構築する方法

１９９４年にサンタフェ研究所の発表会で、メラニー・ミッチェルは、適応系の計算と学習についての業績を発表したとき、一枚のチェックリストを聴衆に披露した。それは、仮想研究用のシミュレーションを構築する際に従うべき代表的なステップを挙げたリストであった。

ミッチェルのリストには、ティエラやTRANSIMSのような仮想世界を組み立てるときに、何をしなければならないかがはっきり記されている。ミッチェルによれば、仮想世界の創造は次の手順で行われる。

・ある疑問に答えるとき、本質的と思われることだけを残し、現実世界の問題をできるだけ単純化すること。
・個々のエージェントや、それらの行動および相互作用のルールを、必要とされるランダムな要素も含めてシミュレートするプログラムを書くこと。
・乱数発生装置から生じるさまざまな値を使って、何度もプログラムを走らせてデータと統計資料を集めること。
・個々のエージェントが利用する単純なルールが、システム全体で観察される反応をどのようにして生じているのか、それを理解するように努めること。
・行動を起こす要因を確認するために、パラメータの値を変化させたときの影響をしっかり把握すること。
・もし可能ならば、さらにシミュレーションを単純化するか、必要と思われる新しい要素を付け加えること。

以上のステップは、実際の現象を扱う数理モデルを作るときに研究者が従うのと同じステップである。しかし大きな違いが一つだけある。それは、プログラムを書くというステップが含まれていることだ。個々のエージェントの行動や、エージェント同士の相互作用をプログラム化するのは困難な仕事である。同じように、エージェントが生息する環境をコード化する（コンピュータ内部に組み立てる）ことは、通常、とても扱いにくく時間がかかる。

そうしたモデルの仕組みや耐久性をテストする方法についても、窮めて重大なことがある。こうしたモデルに特有の巨大なパラメータ空間や非線形の相互作用のため、標準的な統計手法を使ってモデルの反応を明らかにすることは難しい。

最近、カーネギー・メロン大学のジョン・ミラーが遺伝的アルゴリズムを使って、パラメータ空間を探す効率的な手続きを提案した。その手続きは、モデルが暗に含むところを“解読する”方法を探し出してくれる。

（8）SWARMシステム

本項で紹介した例では、個々の研究者（トム・レイ、ジョシュ・エプシュタイン、ロブ・アクステル、クリス・バレット）とその協力者たちは、ソフトウェアを処理する面倒な仕事を自分自身でやらなければならなかった。だがその努力の結果、それぞれ異なる世界で彼らは実質的に同じ基本ソフトを発表した。

この二度手間の非効率に気付いたクリス・ラングトンは、人工生命に関する自らの研究を一時中断することにした。そして、シミュレーション・モデルが容易に作れるように、多目的に使えるシミュレーション・ツールの開発を行った。

ラングトンはそれをSWARMと名付けた。SWARMシミュレーション・システムは標準化されており、順応性と信頼性とを兼ね備えている。それは、一組のソフトウェア・ツールを研究者に提供することを目的としており、これを使って本書で紹介してきた複雑で連応性があるシステムの実験が行える。

特にSWARMは、エージェントが多数いるシステムを設計、分析、シミュレートするなど、いろい

ろな用途に利用される。SWARMシステムの主な目的は、大規模なコンピュータ・シミュレーションを作る際に、コンピュータを扱うという重荷から研究者を解放することである。この目的を達成するために、シミュレーションを動かす核心部分以外に、SWARMは広範囲にわたるツールを私たちに提供してくれる。

こうして研究者は、興味ある対象をモデル化するために、多目的なSWARMシステムを自分の思いどおりに利用することができるのである。SWARMを用いて創造できる世界はさまざまであり、TRANSIMSのような二次元平面の世界（その中ではエージェントが物理的に動き回っている）から、コミュニケーション網を表現する抽象的なグラフ（それを通じてエージェントは物や情報を交換する）にまでわたる。対象とする世界がどのようなものであれ、興味を持つ特定のシステムに研究者が集中できるようにSWARMは一般的な枠組みを提供してくれる。それにより、データの扱いやユーザーのインターフェイスなど、ソフトウェアおよびプログラム上の問題によって研究者が悩まされることはなくなった。

アリの世界をシミュレート

たとえば、SWARMシステムを利用して、アリの群落をシミュレートするとしよう。まずは、SWARMのライブラリーから、全体的な二次元領域を選択しなければならない。これは、群落のアリが動き回る領域を表現している。次に回じライブラリーから、一般的なタイプのエージェントを選ぶ。このエージェントは、その世界を動き回る方法、その世界の状態を感じ取る方法、さらに、ほかのエージェントたちとコミュニケーションする方法をすでに知っている。この全般的な世界とエージェントを選んだら、それにアリの世界に特有の属性やルールを与えて、この一般的なモジュールを特殊化する。

例えば、次のような特徴を含めようと考えるだろう。アリの巣の位置を定めたり、アリの出すフェロモンが分布している状態を決めたりするのである。また、時が経つにつれて、フェロモンが散らばるメカニズムも考慮したいだろう。こうしてアリというエージェントは、元の一般的なエージェントから二次元での移動コードを受け継ぐ。

次のステップでは、そのエージェントがフェロモンの濃度勾配に反応するように特別なコードをアリの世界に追加する。また、アリの移動を支配するルールを定めるコードも同様につけ加える。以上のように特殊化された空間とアリは、活動スケジュールとともにSWARMシステムの中に投げ込まれる。

こうしてアリの行動に関するオーダー・メイドのシミュレーションが得られる。本項では、実際の道路交通網をミクロな形でシミュレートしたTRANSIMSから、一般的な進化プロセスを研究するティエラまで、いくつかの仮想世界を調べてきた。

この２つの両極端（一方は可能な限り詳しく現実の構造をとらえようとし、他方はプロセスに関心を集中している）の間にコンピュテラリウムがあった。これは両方の特徴を兼ね備えている。

PNC ZJ1064 97-001 Vol.2

限定資料

分置01

地層処分ヴァーチャル・エンジニアリング・システムの開発（Ⅰ）

シミュレーション技術研究

（動力炉・核燃料開発事業団　研究委託内容報告書）

技　術　資　料		
開示区分	レポートNo.	受領日
Z	J1064 97-001 Vol.2	1998.4.14.

この資料は技術管理室保存資料です
閲覧には技術資料閲覧票が必要です

動力炉・核燃料開発事業団　技術協力部技術管理室

１９９７年３月

日商岩井株式会社

この資料は，動燃事業団の開発業務を進めるため，特に限られた関係者だけに開示するものです。ついては複製，転載，引用等を行わないよう，また第三者への開示又は内容漏洩がないよう管理して下さい。また今回の開示目的以外のことには使用しないよう特に注意して下さい。

本資料についての問合せは下記に願います。
〒107　東京都港区赤坂１－９－13
動力炉・核燃料開発事業団
技術協力部　技術管理室

限　　定　　資　　料
PNC ZJ1064 97-001 Vol.2
１９９７年３月

地層処分ヴァーチャル・エンジニアリング・システムの開発（１）

―シミュレーション技術研究―

浅原　進也１）、David REED２）
Trevor Sumerling ３）、Geoff FREEZE４）

要　旨

地層処分ヴァーチャル・エンジニアリング・システムの中核となるシミュレーション機能開発にかかわる予備的な検討を行った。検討の結果、シミュレーション機能は評価シナリオが基本となって、実行に必要な複数の計算モデルに対応した独立したオブジェクトとして動作する個別計算プログラムが複雑に組み合わされ、並列・協調して動作するアーキテクチュアが適当と考えられた。それにより、コンピュータ内部に構築された仮想世界において複雑な現象を数値的に実験し、新たな定式化手法として観察する可能性が期待できると考えられた。例えばそれは、コンピュータ内部に仮想的に構築された地層処分システムを対象に、シナリオに添って複雑にモデルを組み合わせて、それらを協調させた新しいシミュレーション技術開発課題と考えられる。

本研究では、そのような手法を、モデル協調計算手法と呼ぶことにした。モデル協調計算は、従来研究開発されてきた、トップダウン・アプローチ（現象全体を確率論的モデルで記述したもの等）とボトムアップ・アプローチ（現象全体を個別の要素に還元して個別の決定論的モデルで記述してゆく手法）とは別な、第３の手法として、次のような考え方とした。それは、コンピュータ中に組み立てられた仮想世界の中で、複数の数値計算モデルを協調させて並列的に実行することにより、従来のトップダウン・アプローチで押さえきれなかったより精密な現象過程や、ボトムアップ・アプローチで記述が難しかった、ある程度の自由度をもった現象のシミュレーションを計算で実現する方法であるとした。また、モデル協調計算技術の開発は、理論、実験といった従来の科学技術の方法とは別な、第３の科学技術の方法である計算科学技術の位置づけを明確にする科学思想とした。

このような協調計算機能については、複数のコンピュータを用途別に接続しそれぞれが協調して並列（平行）に動作し全体として一つの目的に沿ったシミュレーション動作を行うシステムの開発技術が必要となる。平成８年度は、それらの基礎的な機能を確認するためプロトタイプ・システムの試作を行った。試作の結果、計画する協調計算アーキテクチュアは技術的には開発が可能と考えられた。従って、次年度の課題として、個別のシミュレーション・モデルの連結機能研究と協調計算システム全体の基本設計が必要であると考えられた。

本報告書は日商岩井株式会社が動力炉・核燃料開発事業団の委託により実施した研究内容結果である。
契約番号：080D0330
事業団担当部課室および担当者：環境技術開発推進本部　処分研究グループ　梅木　博之

１）日商岩井株式会社
２）University of Reading, Enterpris, U.K
３）Safety Assessment Management Ltd. U.K
４）INTERA Inc. U.S.A

COMMERCIAL PROPRIETARY
PNC ZJ1064 97-001 Vol.2
MARCH 1997

Developmemnt of the Virtual Engineering System for High Level Radioactive Waste Geological Disposal Performance Assessment

Simulation System Development

Shinya ASAHARA 1), Geoff FREEZE 2),
Mark SAWYER 3), Trevor Sumerling 4)

Abstract

In the context of the Virtual Engineering System for Geological Isolation, preliminary research on the development of a computer-based simulation capability, which is one of the key technologies for the system, has been conducted.

The research work has demonstrated that such simulation architectures are appropriate for a system that treats individual numerical computational models as independent "objects" and, based on selected scenarios, executes in parallel collaborative computations among those "objects". This methodology can be defined as a third approach, which is different from either Top-down (probabilistic analysis) or Bottom-up (deterministic analysis) approaches. With this architecture, it is expected that numerical experiments of complicated phenomena can be simulated and observed in a computer virtual world.

In order to realize the methodology, it is necessary to develop a collaborative computation technology where functions are distributed in parallel to designated computers on a network, and by having the individual computers communicate with each other when a process request or data inquiry occurs. As a whole, the system, which may consist of multiple units of computers, performs simulations of natural phenomena. For the purpose of confirming the feasibility of the concept and the basic technology, a prototype has been developed. Further, work performed in FY 1996 indicates that such technology can be developed and that combining coupled model simulation studies and the preliminary design of a collaborative computation system will be needed for future research work.

Work perfoormed by Nissho Iwai corporation under contract with Power Reactor and Nuclear Fuel Development Corporation.
PNC Liaison : Radioactive Waste Management Project, Isolation System Research Program
Hiroyuki Umeki

1) NISSHO IWAI corporation
2) INTERA Inc. U.S.A.
3) University of Edinburgh, Edinburgh Parallel Computing Centre. U.K.
4) Safety Assessment Management Ltd. U.K

【 目 次 】

1 仮想世界としての地層処分ヴァーチャル・エンジニアリング …… 1
1.1 シミュレーションの目的（放射線防護） …… 9
1.1.1 米国における原子力施設に関する放射線防護基準 …… 9
1.1.2 ドイツにおける原子力施設に関する放射線防護基準 …… 12
1.1.3 スウェーデンにおける原子力施設に関する放射線防護基準 …… 13
1.1.4 サイト選定基準の考え方 …… 15
1.1.5 ヴァーチャル・エンジニアリングのシミュレーション …… 16
1.2 安全性評価とシナリオ …… 17
1.2.1 安全性評価に関わる時間 …… 17
1.2.2 長期の安全性を示す方法 …… 19
1.2.3 発生頻度の小さい事象の影響評価 …… 19
1.3 シミュレーションの領域 …… 25
1.3.1 シミュレーション期間 …… 25
1.4 地下水シナリオによる多重バリア方式の性能評価 …… 33
1.4.1 地質環境としての地下水シナリオ …… 34
1.4.2 広域地下水の流動 …… 36
1.4.3 局所的地下水の流動 …… 39
1.4.4 地下水の化学的性質 …… 40
1.4.5 鉱物組成、及び微視的空隙構造 …… 46
1.4.6 ニアフィールド問題 …… 49
1.4.7 適性問題へのアプローチ …… 55
1.5 数値実験オブジェクトとして考えた評価モデル …… 58
1.5.1 各種のモデル …… 58
1.5.2 モデル関連パラメータの不確定性について …… 62
1.5.3 システム操作面から見た総合評価について …… 64
1.6 ヴァーチャル・エンジニアリングの目指す科学技術 …… 68
2 オブジェクト指向のソフトウェア開発 …… 78
2.1 新たな研究開発の戦略 …… 81
2.1.1 オブジェクト指向表現 …… 82
2.1.2 方法論 …… 82
2.1.3 オブジェクト・シミュレーション …… 82
2.1.4 新しいパラダイムの必要性 …… 83
2.2 オブジェクト指向ソフトウェア開発 …… 85
2.2.1 クラス階層のソフトウェア構造 …… 87
2.2.2 クラス、オブジェクト、カプセル化 …… 89
2.2.3 サブクラス―継承とポリモーフィズム …… 90
2.2.4 プログラミング …… 90
2.3 C++言語によるモデル表現 …… 92
2.3.1 哲学的問題 …… 93
2.3.2 C++におけるプログラミングの特徴 …… 93

2.3.3 プログラミング原理 …… 95
2.4 Smalltalkによるモデル表現 …… 96
2.4.1 言語の特徴 …… 96
2.4.2 抽象化 …… 97
2.4.3 メッセージ交換による計算 …… 97
2.4.4 メッセージ式による制御構造（ブロック） …… 98
2.4.6 クラスとインスタンス …… 99
2.5 数理モデルの考え方 …… 101
2.5.1 プログラムの例 …… 103
2.5.2 オブジェクト指向運動モデル …… 107
3 並列計算技術の応用研究 …… 118
3.1 シミュレーション計算量の推定 …… 118
3.2 並列計算技術の現状 …… 123
3.3 並列シミュレーション・プログラム …… 127
3.4 並列計算プログラムの開発 …… 132
3.5 協調作業のための基本パラダイム …… 136
3.6 システム環境 …… 152
3.7 並列問題として地層処分性能評価 …… 153
3.7.1 試作並列シミュレーション機能 …… 155
4 シミュレーション問題への新たな観点 …… 161
4.1 複雑な現象と複雑なモデルとして捉える …… 165
4.2 ゲームとシミュレーション …… 180
5 動画化シミュレーション・プログラムの試作 …… 186
5.1 プログラム仕様 …… 186
5.2 固有の座標系 …… 194
5.3 座標系 …… 196
5.4 変数とデータ …… 198
5.5 ユーティリティ・プログラム …… 207
5.6 動画シミュレーション・プログラムの機能 …… 210
5.6.1 操作開始 …… 210
5.6.2 コントロールパネル …… 213
5.6.3 基本操作 …… 214
5.6.4 視点操作 …… 217
5.6.5 スライス …… 218
5.6.6 ボリュームレンダリング …… 220
5.6.7 速度の軌跡 …… 220
5.6.8 速度の変数 …… 222
5.6.9 テキストラベル …… 222
5.6.10 データ検証 …… 222
5.6.11 新しい変数の作成 …… 223
5.6.12 画像ファイルの保管と印刷 …… 225
5.6.13 テクスチャマッピング …… 226
5.6.14 Tclスクリプティング …… 226
5.6.15 キーボードの機能 …… 227
5.6.16 制限事項 …… 227

5.7 v5dimportのユーティリティ …… 227
6 まとめ …… 232

付録一A 動画化シミュレーション・プログラム 動作画面集

付録一B プロセス間通信

1　仮想世界としての地層処分ヴァーチャル・エンジニアリング

地層処分ヴァーチャル・エンジニアリング・システムの中核となるシミュレーション機能開発要件を整理するための予備的な検討を行い、仮想世界に於ける性能評価シミュレーションの概念をとりまとめた。

地層処分ヴァーチャル・エンジニアリング・システムのシミュレーション機能は、**評価シナリオが基本となって、実行に必要な複数の計算モデルに対応した独立したオブジェクトとして動作する個別計算プログラム（エージェントと呼ぶ）が、複雑に組み合わされ、並列・協調して動作するアーキテクチュア**を考えている。それは、仮想世界における環境実験として期待する技術開発戦略であると考えられる。それは、また、核種漏洩と移行評価に係わる広域的な問題を確率過程として扱う確率論的リスク解析と、個別の事象を有限要素法や有限差分法で扱う決定論的現象過程の解析という、２つの手法の中間に位置する計算による手法とも考えられる。

従って、別途実施している個別現象解明のためのコード開発や、それらを順序だって組み合わせてシミュレーションを実施する評価解析作業とは分離して考えた。

（1）新たな実験方法としての期待

地層処分ヴァーチャル・エンジニアリング・システム開発の期待は、コンピュータにより構築された仮想世界において、複雑な現象をシミュレーションにより再現して観察することにより、新たな定式化を期待することである。このような戦略においては、コンピュータ利用に係わる様々な最新の技術や情報科学・理論が、その世界を構築する基盤要素となっている。

また、コンピュータを利用した仮想世界に構築された地層処分システムを対象に、シナリオに添って複雑にモデルを組み合わせて、それらを協調させた新しいシミュレーションが期待できる。ここでは、そのような手法を**マルチ・エージェント・モデル協調計算手法**と呼ぶことにする。そのような場合、シミュレーションのソフトウェア開発手法は、従来の手法と大きく異なるであろうことも予想される。なぜならば、従来のコンピュータ・シミュレーションは、実験や理論といった活動を支援する道具としての位置づけで開発利用されてきたものであるが、ヴァーチャル・エンジニアリングにおいては、その実験や理論化（定式化）の場を、コンピュータ内部に構築された仮想世界に移そうという思想が根幹にあるためである。

数年前、ドイツのユーリッヒにある、原子核研究センターの所長ボルフ・ヘーフエーレ教授が、“仮定性：hypothticality（ハイポセティカリティ）”という言葉を造ったと言われている。それは、意思決定について次のように説明している。命がけの決断を下さねばならないとき、その決断をする以前に、状況を知るうえで必要な情報を手に入れたいのだが、その情報入手のために必要な実験や検査は実行できないような状況である。その当時、ヘーフェーレ教授の関心は、起こりうる危険に集中していた。当時、その危険性は、原子力の広範な利用に関係のある市民や環境保護主義者も関心を持って見つめていた。今では、この仮定性という言葉はより広い意味で使われている。今では、**複雑系**と呼ばれるシステムの振る舞いについて検証したいと思うあらゆることに対し、仮定性は同じように通用されているのである。

例えば、大気中に放出される二酸化炭素の量が、現在の２倍のレベルに達すると仮定しよう。すると、今から50年後の地球の平均気温にどのような影響が及ぶのだろうか。これは、地球温暖化の問題

を研究する際に気候学者が用いる標準的なシナリオである。この問題に対する答えは数多くある。全く影響がないというものから、気温が摂氏5度上昇するというもの（いわゆる温室効果）まで、さまざまな研究グループが、それぞれの推測を提案している。実際は誰にもわからないが、数多くの人々がそれに関心を抱いている。というのも、気温が大幅に上昇すれば、農業生産から南極の氷まであらゆることに劇的な影響を及ぼす可能性があるからだ。但し、全体の予測を今までの方法では正確に行うことはできない。何故ならば、二酸化炭素の量の増加に伴う気候の変動は、非常に広範な地球のダイナミクスに係わる現象が全体として増加後の現象に向かうのであり、個別現象の観測や予測では気候全体の質的な動きが捉えられないからである。

それは、あたかも車を構成するボルト1つの引っ張り強度から、そのボルトが利用されている車の性能を予測するようなものである。おそらく環境問題は、仮定性に基づく問題の代表的なものであろう。高レベル放射性廃棄物地層処分の性能評価は、環境中に建設する処分システムという人工物の環境に及ぼす問題と考えられる。このような問題は、純粋な自然現象ではなく、人の活動、即ち社会システムの活動が及ぼす自然環境への影響として捉えるべきである。そのような問題は、社会環境問題と考えるべきである。そのような複雑系の問題にアプローチする時、始めに考えなければならないことはモデルへのアプローチである。モデルへのアプローチは、研究をすすめて行く上で基本的な考え方となるのであるとともに、ソフトウェア開発に於ては、具体的なアルゴリズムやアーキテクチュアを左右する重要な課題である。

（2）モデル表現

例えば、飛行機の風胴試験をしている人とか、ファッション雑誌を読む人たちにとっては、これまでに述べたようなモデルという言葉の使い方は少々驚きかもしれない。彼らにとってのモデルとは、飛行機の胴体のように、現実に存在する物体のミニチュア版か、あるいは現実の人間を理想化した体型のことである。いずれの場合にもそれらのモデルは、数学の記号や、コンピュータ・プログラムとは何の関係もない。このようなモデルは、通常、**実験モデル**と呼ばれている。

それは、現実的対象が有する特徴を表現したり（たとえばミニ・カーや博多人形）、現実世界の不完全な容姿を理想的な姿（たとえばファッションモデル）で置き換えて、現実を物質によって表現しているからである。この種のモデルは、直接実験によって答えを出すタイプの問題に使える。

例えば、異なる原子力発電所の冷却系を流れる冷却水の振る舞いや、大気の条件下で飛行機の機体の周囲を空気はどう流れるかとか、理想的な体型を特別な服で覆ったらどう映るかとかの問題に利用できる。実験モデルは、日常生活でも科学でも重要なものだが本研究の対象ではない。

本研究の対象は、物質ではなく純粋な情報から構成されたモデルである。それは、現実を物質ではなく、記号で表現したモデルの領域には区別しなければならないさまざまなタイプがある。

例えば、数学にはモデル理論と呼ばれる研究分野があり、そこでは**論理モデル**というタイプが扱われる。図1－1は本研究の対象とする論理モデルの領域について示している。

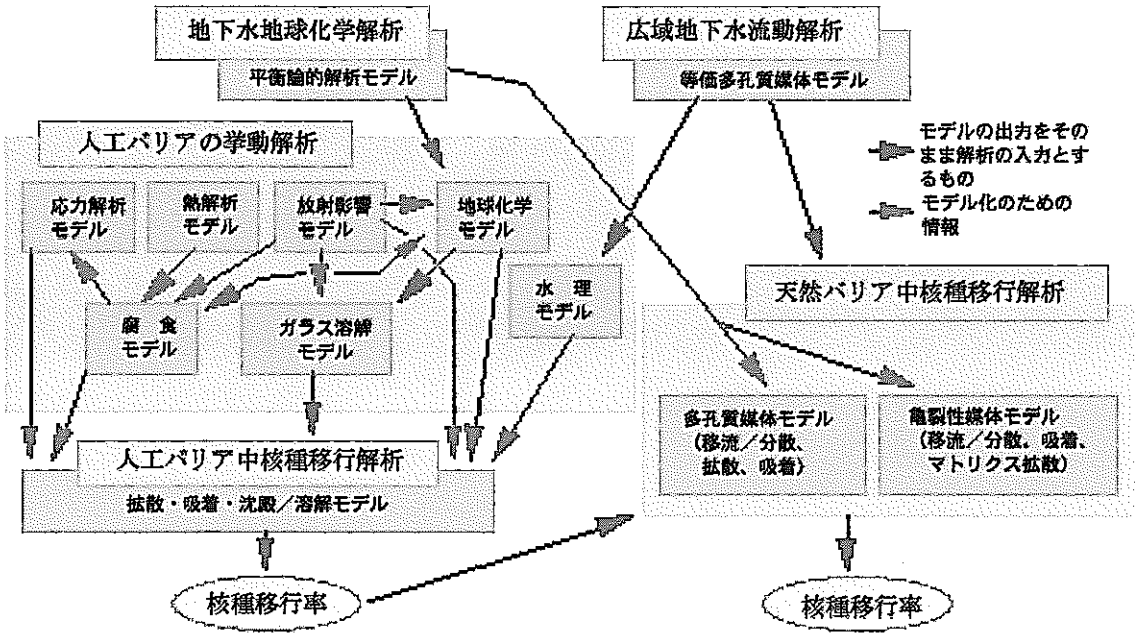

図１－１　地下水シナリオに対する解析モデルの体系

論理モデルは、まず公理と推論規則とを持っている。その公理から出発して推論規則を適用すると、その公理から新たに論理的に正しい記述を作り出すことができる。しかし、この段階で、公理と演繹規則から導かれる記述は、抽象的な記号の羅列、すなわち形式的な統語的体系（規則を使って機械的に作られた体系にすぎない。数学記号による表現はその例である。

（３）数理モデル

数理モデルの理論は、この抽象的な公理体系の性質を忠実に反映する操作規則を持った、具体的な数学体系を発見することである。言い換えれば、それは形式的な統語的体系に意味を付与することである。例えば、高校の幾何に出てくる点や直線の集合は、ユークリッド幾何学の形式的公理に対する具体的なモデルとなる。そのモデル理論は、点と直線という特別な体系を利用して、もとの形式的な公理系を説明し意味を与える。しかし、本研究の課題は、このモデル理論的な意味合いでのモデル理論ではない。それは、風洞やファッションモデルのような現実的な実験モデルと、厳密で抽象的な論理モデルとの中間にある、数理モデル（詳しくは数学的かつ計算科学的モデル）と呼ばれるモデルである。

数理モデルの典型例としては、ニュートンの運動方程式、$F = m\alpha$があり、これは質量mの粒子に力Fが働くと加速度がαとなるという関係を表している。このタイプのモデルは、数式を用いて明確に表現される。モデルをいくつかの微分方程式で示せば、その数値的に方程式を計算することにより、時間の経過につれてニュートン的粒子が位置と速度をどのように変化させるかが理解できる。すなわち、ニュートンの運動方程式に書かれた変数同士の関係をコンピュータ・プログラムにコード化すれ

ば、モデルができ上がるのである。いずれにしても、このモデルはF=mαといった抽象的な記号を用いて物理系を表現している。そのように、数理モデルは近似計算可能な幾つかの方程式により与えられるのであり、コンピュータを利用することにより、ある特定の条件についての振る舞いを計算結果で得ることができる。

（4）数理モデルと理論モデル

理論モデルは、観測された現象をうまく説明するために科学者が考案したもので、その現象の進行のプロセスやメカニズムを表現する仮想モデルである。このモデルは、一般によく知られた現象と対比させることによって提案され、その結果、モデルで表現された現象は理解しやすいものとなる。

例えば、原子の太陽系モデルがある。ボーアは原子核を太陽と考え、原子核のまわりを回る電子を惑星と見なしたのであった。数理モデルと同様、理論モデルもまた現実の状況を抽象的に表現しているわけだが目的が違う。なぜならば、理論モデルの目的は、予想ではなく説明のためにあるのである。このモデルの目的は、未来に起こる現象を予言することではなく、過去の観測を説明することである。しかし、理論モデルは、数式やコンピュータ・プログラムで表現することにより数理モデルに変わることもある。

例えば、惑星の形成はその例である。そこでは、引力を介して相互作用する無数の粒子が合体して惑星系を作るというイメージの理論モデルから出発している。この対象（相互作用する粒子群）をコンピュータ・プログラム（これ自身が形式的で数学的な構造を持つ）にコード化することにより、惑星形成のプロセスが数理モデル化できるのである。このモデルによって、さまざまな物理的条件の下（ガス雲の質量、その構成や回転率など）にある現実宇宙でどんな惑星系が観察されるのかが予言できる。このような手法は、広く物理・化学分野で利用されている。

（5）モデルの分類

以上述べたことをまとめると、モデルは少なくとも4つのタイプに分類できる。すなわち、実験モデル、論理モデル、数理モデル、そして理論モデルの4つである。本研究で主に関心を寄せるのは最後の2つであり、特にコンピュータを利用するモデルである。このようなモデルを利用して、交通渋滞とか惑星の誕生や生命の発生においてどのようなパターンが出現するのか、その数値的に実験することができる。その際、車の運転手、あるいは細胞といった個々のエージェントの行動から、それらのパターンが出現するのである。多くの種類の放射性核種をエージェントとして考えれば、それらの収着や沈殿、輸送をエージェントの行動と見立てるモデル化も考えられる。おそらく、それは化学の説明に用いられる教材モデルと多くの共通点をもっているであろう。

そのように、モデルのタイプにはさまざまなものがある。それらは、それぞれ独自の特徴と利点や欠点がある。

以上の考察から、従来の実験と理論という2つの研究領域に加わる第3の領域として、コンピュータによる仮想世界の数理モデルを基盤とした複雑系の領域が上げられる。それは、地層処分ヴァーチャル・エンジニアリング・システム開発の最も基本となる動機と期待である。それはまた、従来とられてきた科学技術の方法論である、要素還元論的戦略の限界に対する挑戦として期待することもでき

る。それには、以下のような従来の科学思想に関する反省的な考え方に基づく主張（新しい方法に対する期待）が上げられる。

主張１：自然は複雑であり、人類はそのごく表層部分を明らかにしたにすぎない。

主張２：単純な還元論的思考の限界は、気象予測モデル開発に端を発するカオス理論の中で明確にされた。現象理解や、定式化と現象予測の間にも、単純な還元論的思考を働かせることはできない。

主張３：複雑現象に対して、単一の力学的予測は不可能である。従って、個別事象に対する力学的予測はあきらめなければならない。従来の観点では、力学的予測をあきらめることは力学的法則が存在しないことを意味した。しかし、新しい観点では、力学的法則がたとえ存在したとしても、力学的予測はできないことが示される。同様のことが大気や地下水流れ等環境モデル全般を支配しているかも知れない。

主張４：観測者の視点を明示的に自然の中に導入することにより、純粋客観でも純粋主観でもない相対的な観点が入り込む。観測結果は、観測手段に相対的にしか意味をもたないので、単純に観測結果を「真理」と認めることは危険である。観測結果は、たしかに特定の観測手段にもとづく「事実」を与えるが、それは観測行為に対する相対的な真理である。このようにわれわれは、厳密には実験による客観性の獲得をあきらめざるをえない

主張５：純粋客観の視点は、論理によってのみ与えられ、観測によっては与えられない。すなわち純粋客観の世界は論理的な世界である。論理を信じ、数学的構造を客観性の基準にすえることができる。コンピュータでの実験やシミュレーションを、複雑現象の理解にとって非常に大切なものととらえるが、それは次のような理由による。

コンピュータ実験ではコンピュータは素過程として論理的に動いている。しかし、一方で、コンピュータ内で起こる現象を理解したり、定式化したりするとき必要になることは、それら論理の筋をすべて追うことではなく、コンピュータ内の現象を通して、養われた新しい直観にもとづくことである。このとき、まず信じることは、素過程での論理の正当性である。これを信じた後、今までの物理学的直観とは異なるかも知れない、新しい直観を養わなければならない。

たとえある事実が、物理的直観に反したとしても、このような意味で「数学的事実」であれば、それを真理とみなすことをためらってはならないであろう。このように、複雑現象に対する数学的定式化は、コンピュータの中で形式化するのが適当である。

主張６：コンピュータによって創られる「人工的」な世界と「自然」を、絶対的な基準で区別することは不可能である。コンピュータの中で世界を構築するという、工学的な手法は複雑現象の理解へと到達するおそらく唯一の方法を与えるであろう。複雑現象をコンピュータの中で形式化、モデル化することで、新しい型の「予測」が可能になることをみる。

それは、系の構造と歴史に対する可能な過程を提示すことで、単一現象の「先」を予測するのではなく（それ自体は、すでに示したように不可能である）、構造と歴史の集合を予測するのである。新たな方法を生み出す契機を与えたとみるべきだろう。カオス、複雑現象の理解をめざした研究から、今まったく新しい自然認識へ到達したと考える。それは人

工世界と自然との相対化であり、観測行為の新しい定式化をめざすものである。

地層処分ヴァーチャル・エンジニアリングで目指すものは、コンピュータによる仮想世界の構築とそこで生じる複雑な現象の観察という新しい定式化である。そこでは、コンピュータ・シミュレーションとコンピュータ実験は区別して扱う。前者は、従来よりとられてきた自然界の観測から定式化したモデルを近似的に数値計算により解く行為であり、後者は、コンピュータ内部の仮想世界で生じた過程を観察して定式化する行為である。地層処分ヴァーチャル・エンジニアリングは、この後者のアプローチを可能にする具体的なシステムを開発することである。

従って、現在別途に性能評価技術開発のために研究開発が進められている個別現象解析レベルの連成複合事象・現象シミュレーションや関連するプログラム統合化手法の研究は別の研究開発課題である。

（6）システムの技術開発方針

地層処分ヴァーチャル・エンジニアリング・システムを取り巻く技術は、具体的にはコンピュータ・ハードウェアとアーキテクチャ、及び言語とコンパイラ、モデル化とアルゴリズム開発を包含していると考えられる。それは、システムに必要な諸プロセスの実行にかなりの計算能力が必要になると考えられる。

とりわけ、化学的変化のモデル化、及び処分システム構成要素の構造分析、気候･地殻変動による変化の予測と併せて考える地下水流動のシミュレーションのようなアプリケーションでは、計算能力に対する要求がきびしい。それらのアプリケーションに加え、プロセッサスピードと入出力性能の両観点からも、高度な計算能力が地層処分ヴァーチャル・エンジニアリング・システム開発の必要条件になる。さらに、コンピュータによる精巧な可視化・仮想現実感に対するデータ処理要求が、これらの計算能力に対する要請に加えられる。

コンピュータ・ハードウェアについては、従来型計算アーキテクチャの性能は、プロセッサスピードによって決まってくる。これまでは、単一プロセッサによるアーキテクチャの性能に進歩がみられた。ハードウェア開発は、今日のマイクロ電子工学技術の物理的限界に到達しようとしている。

従って、より大きな計算能力を達成するために、並列計算が必要不可欠になると考えられる。プロセッサスピードの他にも、プロセッサとメインメモリおよびディスク間のデータ転送スピード等により、多くのアプリケーションに限度が設けられる。並列計算アーキテクチャを利用すると、計算能力が機械の規模に比例する可能性が生じる。この可能性は、とりわけ、データベース・アプリケーションに、また、大量なデータの可視化処理にもインパクトを与えるであろう。

ソフトウェア開発においては、これまでのソフトウェア構成方法は、近年のオブジェクト指向による開発の成果を基に始った発展しつつある新しい手法との間にはかなりの開きがあると考えられる。

ソフトウェア開発の経済性の面からは、既存のアプリケーションとデータベースをすべてオブジェクト指向システムとして再設計することは適当でないと考えられる。当面の間、システム開発は、旧来の方式と新たな方式の混合パラダイムを使って作業を続け、諸問題に対処していかなければならないであろう。例えば、既に定まったシナリオを評価する数理モデルとコードとしてできあがったものは、より高度な視覚化インターフェイスを接続して高度化し、アルゴリズムは再利用すべきである。

一方、始めから複雑な問題である、オーバッパックの腐食と地下水・ベントナイト相互作用などについては、複雑系モデル開発として考えるべきである。いずれの場合も、システム全体の設計はマルチ・エージェントなモデル協調計算としてすすめ、既存の数理モデルは、エージェントを特徴つけるメソッドとして組み込む方式等を研究開発すべきである。

地層処分ヴァーチャル・エンジニアリング・システム開発における、このような状況を避けて通れぬ現状は、アプリケーションやデータベースへの既存の蓄積を再利用させる、オブジェクト指向開発手法を採用する戦略をよく検討することを基本戦略として要求する。採用すべき戦略は、オブジェクト指向方法論と超高速計算技術のような関連技術における進歩の良さを無理に出したり、あるいは、排除したりするものであってはならない。代わりに、オブジェクト指向プログラミング言語や、ツールを使って精選したアプリケーションを少しずつ書き直していくより、オブジェクト指向アプリケーションと非オブジェクト指向アプリケーション、サービスとデータベースを統合する、オブジェクト指向ソフトウェアを開発することに主眼をおいてソフトウェア開発戦略を立案すべきである。

それには、以下のように新しい計算方法が適用可能な異なる３つのレベルを規定することできる。

基礎となる物理的モデルの詳細なシミュレーション
別個ではあるが関連するモデルの結合
フィーチャー（機能）の選択と最適化

計算機科学の近年の成果の中で、遺伝的アルゴリズムは、かなりの種類の難しい計算問題に対して伝統的な統計的手法より、効率が良いということがこのところ明らかにされてきた。このアプローチの主な利点は、評価空間のうち、関心を引き注目すべき分野に検索を集中し、染色体への適切なコード化により、主観的、客観的両情報を取り扱う観察された能力である。遺伝的アルゴリズムのメカニズムは、計算上単純で、並列化に非常に適している。

例えば、住民が仕切られている島をモデルに使うと、仕切られた各部分は、一部の人々同士の移動（染色体の移動）によって入れ替えられる情報に対応して徐々に進化する。染色体コード化には、地層処分ヴァーチャル・エンジニアリング・システムに適した選択シナリオの構成概念や、相伴うオペレーションとともにこの手法は開発の余地がおおいにある。

地層処分ヴァーチャル・エンジニアリング・システムの動作に関するさまざまな点で、新世代の高性能な並列計算を用いることの実現可能性も評価されている。少なくとも原理上は、ソフトウェア階層全体にわたって並列アーキテクチャの適用が必要であるということである。これらを踏まえて、システム開発ためのフレキシブルで説得力のある方法論の立案が必要である。それは、これまでの研究から得られた経験に基づいてはいるが、利用可能な最も進んだ意思決定ソフトウェアを組み入れたものであるだろう。

例えば、「インテリジェント・ソフトウェア」（ニューラルネットワークに連結した遺伝的アルゴリズム）を使用して操作の多くの部分の自動化や情報支援を図れば、対話システムの機能の最適化が容易になり、エラーを引き起こす潜在性が減少し、人的資源を最大限に活用することになろう。この方法論の内容は以下が考えられる。

- ・各段階における不確実性の明示表現
- ・分析の進行にしたがってシミュレーション感度を調べるため、インタラクティブな可視化製品の使用
- ・システム展開中のプログレッシブ（漸進的）、サイクリック（巡回的）、エピソディック（挿話的）変化を扱うため、動的シミュレーションの使用
- ・定常状態進化の考察から、主な分裂事象の知覚された重要性の認識への移行
- ・異なる世代のコードの結合使用を可能にするため、各技術領域における開発状態の違いを考え、ジェネリック・データベース・エンジンに連結した共通ソフトウェアインターフェースの使用

それらは、地層処分ヴァーチャル・エンジニアリング・システムの利用者が、数値モデルを選択するときに手引きし、数値計算実行に必要な入力データを検索し、対話型グラフィクスで計算を始め、シミュレーションの各時点の数値計算結果を評価すること等である。そのためには、一連の基準がコード選択過程で助けになるように規定されている。

システム開発においては、性能評価シミュレーションのために、システムの物理的、化学的及び生物学的特質を表す典型的なデータを用いてシナリオ全体をモデル化することが不可欠である。そのようなシミュレーションコードに必要なデータ選択と管理は、コードそのものの選択と同様に大切である。当面は、それらのコードや知識ベースシステムによって参照される、データ全てに対する直接の操作を実現することは難しいであろう。そのため、あらゆるデータを蓄積し、管理するためには、個々のデータベースから開発統合化する必要がある。そのようなデータベースエンジンは、特定のタスクについて、関連データすべての位置付け、照合、抽出ができるものでなければならない。

また、将来の構想として、地層処分プロジェクトの社会システムへのインパクトを評価するシミュレーションについても、研究課題としておくべきである。例えば、教育の第一歩、法律関係の調査を手助けするための、また、社会受容性に関する業務を支援する一般情報の提供には、各々、異なるタイプのデータが異なる複雑なレベルで要求される。そのような問題には、複雑系のアプローチは有効なものと考えられる。

（3）まとめ

地層処分ヴァーチャル・エンジニアリング・システムのシミュレーションは、評価シナリオの決定、シミュレーションの実行、実行結果の整理等、シミュレーションに直接関連する操作に始まり、シミュレーション結果の妥当性等を検証・確証するための関連情報をデータベースより検索し、解析結果と比較評価する等のさまざまな性能評価に利用されると予想される。

国の推進する高レベル放射性廃棄物処分システムの技術研究は、処分システム開発プロジェクトの各段階について行われると考えられる。例えば、現在実施されている基礎研究段階が終了すると、サイト特定段階のプロジェクトが開始されるであろう。地層処分ヴァーチャル・エンジニアリング・システムは、それら地層処分プロジェクトのトータルなライフサイクルについて、目的に適合した結果を提供できなければならない。このため、その各開発段階に応じて評価結果の使用目的が異なると考

えられる。

地層処分ヴァーチャル・エンジニアリング・システムの当面の目的は第２次とりまとめ後である。第２段階で実施される処分サイトの特定、及び処分システムの概念設計に係わる安全性評価をするための情報を、複雑なシナリオを組み合わせてシミュレーションし、提供することにあると考えられる。例えば、想定サイトでの処分環境影響の概要、処分システムの設計オプションに関する多方面からの安全性評価等であると考えられる。

従って、地層処分ヴァーチャル・エンジニアリング・システムによるシミュレーションは、地層処分した高レベル放射性廃棄物が超長期に亙り生物圏から隔離されること、言い換えれば人間生活圏の超長期に亙る放射線防護を目的とした、シミュレーションや数値実験を行うことを目的として開発するものである。

１．１　シミュレーションの目的（放射線防護）

地層処分ヴァーチャル・エンジニアリング・システムの開発にあたっては、放射性廃棄物の処理処分の課題を解決するための国の政策、安全基準、PA対策等の政治的なコンセンサスについて、常にそれを正しく反映したＣＧ等の表現技術開発が必要であり、かつ、固化体、容器、地層等の性質についての科学的な研究や、工学的な研究開発成果を総合的に情報として管理する必要がある。

そのような観点から、システムを設計開発するためには、常にそれらの情報を把握し、設計に反映することが大切である。ここでは、以上の観点から、高レベル放射性廃棄物地層処分の最も基本的な考え方である放射線防護基準に関する主要各国の考え方を整理し、シミュレーション機能開発の研究の参考とする。

１．１．１　米国における原子力施設に関する放射線防護基準

米国は、1982年の原子力政策法で、処分サイトの選定から閉鎖までのスケジュールを詳細に設定した。また、許認可機関である原子力規制委員会（NRC）は、地層処分基準（10CFR60）案を提案し、人工バリアによる閉じ込め期間、地下水が地表に達するまでの期間等それぞれのバリアについて性能要件を詳細に決めた基準を示している。

しかし、処分候補サイトがYucca Mountainに決められ、計画が具体化されるに従って、柔軟性に欠けるスケジュールや基準が災いして計画が難航してきた。そのため、再検討が行われ、柔軟性をもって計画を進めるよう、1992年に原子力法が改定された。その結果Yucca Mountainでの地層処分基準については、まず科学アカデミー（NAS）が環境保護庁（EPA）に勧告を提出し、それに基づいてEPAが安全基準を示し、それに沿ってNRCが技術基準を策定することになった。

米国機械学会の廃棄物環境修復国際会議（The international conference on nuclear waste management and environmental remediation）の1995年会議（Waste Management 95）では、処分の実施に責任を持つエネルギー省（DOE）が、NASの検討を助けるために、1994年4月にまとめた勧告の概要を「米国高レベル放射性廃棄物の許認可要件の国際的展望」と題して紹介している。

その中で以下の６つの基本条項を解説し、図１．１－１に示す国内の関連基準と図１．１－２に示す諸外国の防護基準との整合性について述べている。

省庁／組織	文　書	基準／指針
NCRP	NCRP（放射線防護審議会）Report No.91、電離放射線被ばく制限に関する勧告（1987）	p37-39：医療用を覗く人工放射線全線源による被ばく線量<100mrem/y(1mSv/y)
NRC	10CFR Part 20　放射線防護基準	§20.1301：操業による公衆への全実効被ばく線量<100mrem/y(1mSv/y)
NRC	基準案 10CFR Part 20 Subpart E（解体措置の放射線基準）8/22/94	§20.1401：サイトからの漏洩限度を決定グループの組織被ばく線量(TEDE)の平均値15mrem/y(0.15mSv/y)とする。
EPA	40CFR Part 190　原子力発電所の環境放射線防護基準	§190.10：主な操業による公衆の被ばく線量<25mrem/y(0.25mSv/y)（採鉱、輸送及び廃棄物処分場での操業を除く）
EPA	40CFR Part 191　使用済燃料、高レベル／TRU放射性廃棄物の処分の環境放射線防護基準	§191.13：核種ごとの集積漏出量 §191.15：公衆の平均預託実効被ばく線量<15mrem/y(0.15mSv/y)

図１．１－１　米国の原子力施設に関する放射線防護基準

国／機関	放射線基準	処分の時間枠
ICRP(Pub 60)	個人被ばく線量（100mrem/y(1mSv/y)（住居のRn及び医療による被ばくを除く）	規定なし
カナダ	個人リスク<10^{-6}/y	10,000y；長期時間枠での評価
フランス	個人被ばく線量<25mrem/y(0.25mSv/y)	10,000y；長期時間枠での評価
ドイツ	個人被ばく線量<30mrem/y(0.30mSv/y)	10,000y；長期時間枠での評価
北欧諸国	個人被ばく線量<10mrem/y(0.1mSv/y) 個人リスク<0.1mSv/y相当のリスク	合理的に予測可能な時間
スペイン	個人被ばく線量<10mrem/y(0.1mSv/y) 個人リスク<10^{-6}/y	規定なし
スイス	個人被ばく線量<10mrem/y(0.1mSv/y) 個人リスク<10^{-6}/y	規定なし
英　国	個人リスク<10^{-6}/y	10,000y；長期時間枠での評価

図１．１－２　各国の放射線防護基準指針

また、基準作成に対する要求は凡そ以下のとおりである。

要求１：基準は健康ベースであるべきである。

将来世代の集団への健康への影響として示すのが、公衆の健康や安全と比較しやすく判りやすい。

要求２：基準は、処分候補地域の住人の保護におくべきである。

環境境界に近づく人間、処分候補地域の住人、世界の人類、地下水、動植物等を対象にする方法があるが、処分候補地域の住人を対象にすることにより、これらを保護する条件を包含することになる。住人の保護という一つに基準を絞れば、許認可手続きで、その処分システムに従って、各バリアの基準を示すことになる。従来NRCが提案していたような個々のバリアの性能目標は、許認可基準に含まれるべきではない。

要求３：基準は10,000年以上の期間についての性能を定量的に示すことを要求すべきではない。

処分後、10,000年以後にも健康影響がある可能性があることはわかっているが、超長期の性能評価計算は不確実性が増し、サイトや設計を相対的に比較するだけの価値がなくなる。10,000年以上についての要件は定性的にすべきである。

要求４：放射性物質の漏洩の起因による、将来世代の人間侵入の可能性を減らすことについての要件は定性的にすべきである。

将来世代の人間侵入については、将来の技術や生活習慣を予測する必要があり、不確実性が大きすぎる。設計要件や制度的管理の定性的要件にすべきである。標識、土地利用制限、記録保管等詳しい要件を決めることは、処分場の閉鎖近くまで延期すべきである。それまでに得られる情報や技術をもとに適切な内容を決めることができる。

要求５：基準の定量性は確定的な数値で示し、適応性の証明では確率解析が使われるべきである。

10,000年以内での定量的評価の基準値そのものは、あくまでも一つの数値で示され、確率的分布を持ったものであってはならない。性能評価解析では確率計算を行い、確率分布で結果が出る。その分布を基準値と比較することによって、低確率の高影響事象を考慮に入れることができる。

要求６：基準は、個人被曝線量で表されるのが適切である。ただし、確かな見解が規制や指針に明確に示されることを条件とする。

健康ベースの基準を表すには、許容リスク、個人被曝線量、核種漏出量、構成システムの性能、各人工・天然バリアへの要件等がある。いずれも許認可手続きで適応性を示すように表現されるなら採用し得る。

被曝線量基準を採用するには以下の３つの見解が明確にされる必要がある。

> 個人被曝線量限度は従来10CFRPart20にある公衆の個人被曝線量限度と同じにする。
> 被曝線量限度はYucca Mountain地域に住む標準的住人に適応する。
> 将来の生態圏に関する仮定（将来世代の人間の挙動）を特定する。

以上のDOE勧告について同会議では、図１．１－２に示した海外の基準との整合性を次の３点から検討している。

基準の表現形式
一次放射線被曝線量限度
要件における時間枠

結論としては、DOEの勧告は現有の内外の基準指針の範囲にある。

１．１．２　ドイツにおける原子力施設に関する放射線防護基準

ドイツでは、岩塩鉱への処分を主に考えて計画を進めてきたが、地元の了承が得られず計画通りには進行していない。Waste Management 95では、岩塩鉱と合わせて花崗岩を候補岩種として対比しながらサイト選定基準、安全評価、及び地質調査について「高レベル廃棄物処分の安全基準の解析」と題して報告している。

米国とは対照的で、地層処分の設計はサイト特有の地層条件によるので、定量的基準としては、放射線防護目標のみが原子炉安全委員会（RSK）の勧告として示されている。

それは、「全ての合理的な漏洩シナリオによる個人への放射線被曝線量は、0．3mSv／年を超してはならない。」とされている。この値をもとに候補サイトの調査を行い、地質及び地殻構造についての基準を導くとの方針がとられている。

（1）サイト選定の主な基準

岩種は、岩塩と花崗岩を考え、深さは500m以上掘削し、1,200mを超さないようにする。次の要因によって除かれない全ての地域は、サイト選定の対象となり得ると考えられている。

生態学的要因（保護地域、工業地域と人口過密地帯、大容量貯水地域、水理要件の不適切区域）
地質学的要因（破砕帯、上下及び横ずれ運動地帯、地震活動地帯、マグマ地帯）
地質構造要因（不十分な広さ（＜10km2）、露天掘鉱山や地下鉱山、顕著な母岩変質）

（2）バリアの概念

地層処分の概念は、天然バリアと人工バリアの組み合わせでお互いの弱点を補いあう仕組みとしている。天然バリアを構成する２種類の天然バリア、即ち、岩塩と花崗岩では概念が異なる。岩塩は長期にわたってガスや液体を閉じ込める性質があり、シーリングが唯一の重要な人工バリアである。花崗岩は、水と廃棄物の接触を防ぐことは不可能であり、透水率が非常に小さい大規模な岩体により人工バリアとの組み合わせで安全目標が達成できる。すなわち、多重バリアの役割が一層重要である。花崗岩には以下の利点がある。

廃棄物を多くの破壊プロセスから十分遠く離れた場所に処分できる。
廃棄物に影響が及ぶ自然のプロセスが遅い。
人工バリアを保護する機能がある。
放射性核種の移行を遅らせる機能がある。

（3）安全性評価

岩塩はよくしまっている性質があるので、岩盤の初期の状態が明確であり、以後の評価も確定的な評価方法をとることができる。その意味で、岩塩では掘削の段階で岩盤を損ねないことが最も重要である。一方、花崗岩は、既に発生している割れ目の初期状態を正確に表すことが困難であり、将来の変化予測も難しい。したがって、確率論的な評価は欠かせない。

（4）地質探査

探査の目的は、処分サイトとして地質が適切であることを確認することであり、次の項目からなっている。

地層の様子を詳細に書きだし、それに基づき岩体の形状、及び地層環境についての地質学的モデルを開発する。
安全評価解析の関連データを評価する。
母岩のバリア機能に関連する地質学的シナリオを明確にする。

岩種によりバリア機能が違うので探査の重点目的も違ってくる。岩塩の場合は、処分場に適した岩体の大きさを調べることに主な目的がある。花崗岩の場合は地下水の主な通路となる割れ目の性質を調べることが主な目的となる。

1．1．3　スウェーデンにおける原子力施設に関する放射線防護基準

花崗岩処分を考えているWaste Management95では、「人間及び自然の保護」と題して、基本的事項についての考え方を紹介している。

（1）被曝線量限度

廃棄物処理施設、及び輸送システムは個人の被曝線量が50mSv／年、100mSv／5年、700mSv／（生涯）を超さないよう設計しなければならない。この値は、集団集積被曝線量2person・Sv／年－GWeの限度の内に含まれる。公衆については0.1mSv／年以下とする。集団集積被曝線量は5person・Sv／（操業期間・GWe）、500年について平均1mSv／年を超さないようにする。

（2）自然の保護

自然環境の保護とは、一般に人間の健康、生物種、生物資源の生産能力及び農耕地のことである。地層処分による影響評価の主点は生物種の保護にある。生物種を保護するには、個体ではなく集団であるが基準はこの保護基準で表される。そして、生体組織レベルで考えるべきである定量的基準としては、組織や成長過程で違うが、実験データによると1Gy／年から5Gy／年の範囲であるなら、多くの組織について観測できる影響は生じていない。

これらの実験研究は、通常、低レベル長期被曝の場合には留意していないことを考えると、100から1000の安全ファクターをとるべきである。1から50mGy／年を定量的基準とすべきである。処分場の実現性を評価する作業の場合は20mGy／年の値を使ってよいだろう。

（3）長期間の観点

処分場閉鎖後の放射性物質の漏洩については次の３つの期間に分けて検討される。

最初の1000年	：最も潜在的危険性を考えなければならない期間である。
1000年から次の氷河期	：（10,000年のオーダ）Cs137、Sr90の寿命より短寿命の核種は減衰し、長寿命の核種が隔離の対象になり、期間の経過に従ってバリア、社会、自然を考える際の不確定性が増す。
次の氷河期以後	：処分場を定量的に詳しく表すことは困難と思われる。10,000年までの時期が対象であり、次の氷河期以後の時期については定性的判断になる.

（4）人間侵入

将来社会は、その社会の意思での行動については責任を持たなければならない。将来世代の人間が意識的に行う人間侵入は防ぐことはできない。

以上の内容の報告の他に、「放射性廃棄物対策－国際的観点」と題して国際的動向も含めた紹介もしている。

数年前に、OECD／NEAのRWMC（放射性廃棄物管理委員会）は、IAEAのINWAC（国際放射性廃棄物諮問委員会）と協力して、「長期の安全性は評価できるか」と題し、国際的に集約した見解を文書（Collective Opinion）としてまとめているが、これは世代間倫理等、地層処分の基本的な問題についての新しいCollective Opinionの作成作業が進行していることを紹介している。

１．１．４　サイト選定基準の考え方

サイト選定基準は、放射性廃棄物の地層処分に適したサイトを見つけ、選んで行くときの指針を示すことにある。サイト選定にあたっては、選定段階を次の４段階に分けてそれぞれの段階で行うべきである。

- 概念設計段階
- 地域調査段階
- サイト特性調査段階
- サイト確認段階

また、サイト調査についての指針と調査して得なければならない科学的データとして、地質学的特性、自然変化の未来予測、地下水理、地球化学的特性、人間侵入に起因する事象、建設工事条件、廃棄物の輸送、土地使用及び社会的影響に関する指針とデータがあげられる。そして、それぞれについて必要性と詳細な項目が整理される。そのための情報処理システムの開発が調査を開始する以前に必要となる。

全体として科学技術的な面からの記述がほとんどを占めているのは当然のことであるが、サイト選定上しばしば問題になるのは、社会経済的な面、環境問題及び政治的な問題である。それらについては、まず適性サイトの見つけ方である。

もし、あらかじめ制限がなければ、大きな領域から特別のサイトを系統的に絞り込んで、それらのサイトが適切であることを確認できる。始めからサイトになる可能性が強いところがあることもある。これは、常識的に言われているような、広域調査で絞り込む方式のみでなく、最初に特定のサイトを決め、そこが適切であるかどうかを調べていく方式もある。

どのような方法で選ぶとしても、ベストのサイトを選ぶのではなく、安全上及び環境上の要求にあった場所であればよい。サイトは、最初から必ずしもベストである必要はない。そして、最終的にサイトを決めるには、社会経済的な面、環境問題及び政治的な問題についての考慮に基づく判断を含むであろう。従って、科学的判断のみで決まらないことを述べている。

サイトを選定するときのガイドライン、及び必要なデータについて述べている箇所では次のようなことが上げられる。放射性廃棄物の輸送について、処分場への廃棄物の輸送もまた処分場の場所についての公衆の合意を得るのに重要な要因である。場合によっては、新たなルートを造ったり、既存のものを改良する必要が生じる。サイト選定では、輸送が地元との話し合いで重要な項目である。また、環境保護については以下があげられる。

- 坑道掘削やその他の建設工事での環境破壊
- 地域住民に重要な価値を持っている区域への影響
- 水源の破壊
- 植物・動物への影響、特に危険にさらされている種について

土地使用についての適切なサイトの選定に当たっては、将来の開発や、その地域での計画との関連で、土地利用や所有権について考慮する必要があると指針を示した上で、土地の管轄権や所有権は経済性や公衆の合意を得ることに関連して多くの国で重要な原因である。

実施主体または国に土地の所有権があれば、サイト計画や労力を見積るのが簡単になる。さらに別の用途で土地を使用しているのを引上げさせることに伴う問題を少なくする。国有地あるいは実施主体の所有する土地が、公衆の合意を得る上で有利であることを述べている。

最後に社会的効果については、サイトはどのような場合でも、その地域に処分場を決めることによる利益が増進されるべきであり、負の社会的効果は最小になるようにすべきである。一般論として、高人口密度の地域から離れたサイトに特恵が与えられるべきである。しかし、社会基盤が変化しないようにし、労働力が得られる必要があると考え、地域社会での生活が破壊されないよう十分な配慮が必要である。

１．１．５　ヴァーチャル・エンジニアリングのシミュレーション

以上、放射線被曝防護という最も基本的な観点から、放射性廃棄物処分について各国の基準とサイト選定に係わる基本的な議論を概観した。その結果、高レベルだけでなく、放射性廃棄物処分全般について言えることは、議論の対象が、人間社会活動の結果として生じる汚染物質が、超長期に亙り充分人間生活圏から隔離されることを確認する為の技術・経済面からの手法の開発であると言える。それは、最も費用対効果の高い放射性廃棄物の処分方法を策定する為の議論とも言える。この様な議論は、極めて大規模で複雑な系の振る舞いの最適解を見いだすような問題であると考えられる。そのような問題は、個別に分解された問題について解明しても、それらを統合して全体を説明することは難しい場合が多い。なぜならば、問題のシステムとしての質が個別に分解されると失われる為である。そのような問題に対するアプローチとして、コンピュータ内部に仮想環境を構築し、個別に開発した現象過程のモデル等を試行錯誤的に議論とともに組み合わせ、シミュレーションを観察する問題解決戦略が考えられる。

地層処分ヴァーチャル・エンジニアリング・システムは、そのような観点から、従来シミュレーションや数値計算などと呼ばれてきた物理工学上の現象を、コンピュータにより数値的に計算模擬する技術をもとに、オブジェクト指向、並列計算法、認知科学等、革新するコンピュータ・サイエンスの成果を地層処分の安全性評価研究に応用し、処分の安全性に係わる技術論拠となる情報を多方面から総合的に評価することを目的として、専用に統合開発するコンピュータ・システムである。

それはまた、高レベル放射性廃棄物地層処分の安全性評価にかかわる、実験の難しい長期（数千から数十万年）に亘る現象を考慮した各種のシナリオを複雑に組み合わせて、システムの長期的挙動の評価を行ない、地層処分の安全性に係わる技術を判断し、処分システムの設計の最適化をするための指針等を得るとともに、多分野にまたがる専門家が、自己の領域と他の領域との関係を容易に理解するための基盤を提供することである。

例えば、地層処分システムの性能評価は、地震等の確率的に生じる事象による変化を除いて、地層中の環境が安定している場合、地層処分された廃棄体から放射性核種が放出され地下水により生物圏まで輸送されるシナリオを基本としている。そのような安定した地層中での環境に添って核種移行を考えた場合、それだけでも非常に複雑な環境評価モデル開発と評価の問題であることが解かる。

領域１：ガラス固化体（崩壊熱モデル、放射性核種放出モデル、等）
領域２：オーバーパック（腐食モデル、腐食物ベントナイト化合モデル、等）
領域３：ベントナイト（２層多孔質媒体地下水流れ、熱・オーバーパック腐食化合モデル、等）
領域４：人工バリア周辺（人工バリアと岩盤との熱応力モデル、コロイド、核種収着、沈殿、等）
領域５：地層中（広域的地下水汚染物質移行モデル、亀裂水理、等）

実現象は上記のモデルが複雑に組み合わさったものと考えられる。評価シナリオは、例えば温度場の変化が力学系に作用し、地下水流動を変更することにより人口バリアの冠水や亀裂を生じ、結果として核種が滲出して輸送されるシナリオが考えられる。このようなシナリオにそったシミュレーションを対話方式で試行錯誤的に行なうには、下記の操作機能がアプリケーション・システムに要求される。

操作機能１：シミュレーション・シナリオの一覧
　　シナリオは、充分に評価され妥当なものとしてデータベース管理されていること。
操作機能２：シミュレーション条件の決定
　　シナリオの選択、個別評価モデルの選択、計算条件の設定等
操作機能３：シミュレーションの実行と制御
　　シミュレーションの実行、実時間視覚化、計算結果の解析等

以上の様に、個別モデルが複雑に協調したシミュレーション機能は、高レベル放射性廃棄物地層処分のような、大規模で複雑な系の問題に対するアプローチとして新たな結果が期待できる。

１．２　安全性評価とシナリオ

安全性評価の基準としては、処分が公衆にもたらすリスクが社会的に十分許容されるレベルかどうかで判断することが基本であると考えられる。1990年11月にOECD／NEA主催で開催された「高レベル放射性廃棄物処分における放射線防護及び安全規準に関するワークショップ」、並びに1991年10月に開催されたIAEAのINWAC（国際廃棄物管理諮問委員会）サブグループに提出された資料を参考に安全評価の基本的な事項について示し、地層処分ヴァーチャル・エンジニアリング・システムの最も基本的な機能要件とする。また、代表的な安全評価報告書等で取り上げられている主要各国の安全評価シナリオを概観する。

１．２．１　安全性評価に関わる時間

安全性評価の前提として、人間の生活様式、生態圏の環境、及び地圏の環境の長期予測が必要となる。これらの条件の科学的に予測できる期間をどう考えるかが安全評価シナリオを構築してゆく上で

重要となる。高レベル放射性廃棄物中にはPu239、Np237等、10000年以上の長半減期の放射性核種が含まれており長期の問題は避けられない。

スウェーデン及びスイスの規制関係者がまとめた報告書（SKI - TR - 90 : 15）にある図１．２－１に示されている期間についての大略のイメージが一つの参考になる。「生態圏の環境変化を考慮すれば、その予測可能な期間は10,000年程度と推定される。地圏の環境変化を予測できる期間は1,000,000年オーダー程度。サイトを慎重に選べば、それ以上予測できるかも知れない。」としている。

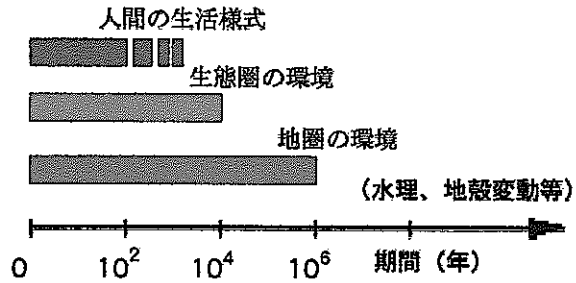

図１．２－１　処分システムの安全性評価に必要な前提条件が予測できると思われる期間

廃棄物中の放射性核種の量は時間が経過するに従って減衰し、大きな影響が生じる可能性も減るので、地圏での異常事象の発生確率の予測上高い精度は要求されなくなる。

英国国立放射線防護局では、期間の経過に伴って次のような尺度で規制することを提案している。その概要は以下の通りである。

0－100年	：制度的管理が残るので施設と同じ被曝線量限度を適用する。
100－10,000年	：生態圏は変わる可能性があるので、最大被曝を受ける個人ではなく、クリティカルグループを想定し、リスク限度を適用する。
10,000 - 1,000,000年	：地圏からの放射性核種の定常的漏出を制限することで人類の安全を保つとの考え方から、標準生態圏及び人間社会を想定し、リスク限度を適用する。
1,000,000年以上	：合理的に可能な限り環境への影響が少ないシステムであることを定性的に示す。

この提案のように、処分場を閉鎖した後の制度的管理の有効な期間は数百年程度であり、定量的な安全評価ができる期間は数万年程度が限界であろうとの点では、各国の規制関係者の見解がほぼ一致している。また、数百万年以上の定量的評価は、生物の歴史や地殻変動等による物質移動を考えると、ほとんど意味がないとの考えが一般的であるように思われる。

１．２．２　長期の安全性を示す方法

将来の人間の生活習慣や生態圏環境を正確に予測することはできない。従って、現在と同じとして評価することが一般に行われている。このことの意味は、将来の人間の被曝線量を評価するというより、廃棄物から地圏を通過して生物圏に漏出する放射性核種の量を現在の生物圏の環境条件での被曝線量に換算し、将来の放射線的環境が処分によって現在と変わらないことを示すことにあると理解した方が解りやすい。

北欧５ケ国の規制関係者がまとめた提案書では、10,000年程度以上の期間については、地表の侵食によって土壌や岩石中の自然のアルファ核種が生態圏に移行する量と廃棄物に起因する漏出量を比較することを提案している。

アルファ核種のみを取り上げているのは、自然の放射性核種と廃棄物中に含まれる放射性核種の種類は必ずしも同じではなく、直接比較できないので、放射線学的及び化学的持性が似かよった自然のアルファ核種と比較することにしたものである。

１．２．３　発生頻度の小さい事象の影響評価

慎重に処分サイトの選定を行っても、断層、火山等の急激な地殻変動を完全に否定することはできない。発生頻度が小さくても影響の大きい事象は評価する必要がある。しかし、それは一般的に起こり得る事象と同じ規準で規制する必要はない。被曝線量をガン発生確率でリスクに換算し、被曝の原因となる事象の発生確率を乗じてリスクで規制しようという考え方と、事象の起こり得る程度をランクづけして、それに応じた線量規準を設ける方法がある。先に述べたSKI－TR－90：15では次の提案をしている。

発生頻度	
1から10のマイナス3乗／年	：通常事象として、被曝線量規準で規制
10のマイナス3乗から10のマイナス7乗／年	：リスク規準で規制
10のマイナス7乗／年以下	：評価免除

この提案は次のように解釈することができる。数千年以内でも普通に起きるような事象は、一般的に起きる事象として被曝線量規準で規制する。それより低い発生頻度と思われる事象については、発生頻度の少なさに応じ、緩やかな線量規準にしようとの考え方である。線量の規準を1mSv／年とすると、数千年間では、ほとんど発生しないような事象に対しては1Sv／年の被曝線量でもよいことになる。数万年間経過すれば、ガラス固化体の放射能濃度は非常に低くなるので、リスク限度を超えるような事象はほとんど考えられない。地層の安定性についての予測も時間の経過に比例して詳しい予測でなくてよくなる。

リスクの影響度と頻度の考え方は、地層処分ヴァーチャル・エンジニアリング・システムのシミュレーション・モジュール設計に基本的な指針を与える。それは下記のクラスに分類することにより情報を構造化することができる。

Class 1：影響大頻度大　設計で対応する
Class 2：影響大頻度小　定性的な評価を行う
Class 3：影響小頻度大　定量的な解析評価を行う
Class 4：影響小頻度小
定性的な検討を行う。ここで大小の判断基準が問題になるが、このアプローチでは
＊「影響大」、「頻度大」の場合には両者は比較的良い精度で推定できること、
＊影響の大小より、事象の発生確率の大小の推定の方が技術的に困難であるという考えがベースにあると考えられる。

高レベル放射性廃棄物処分に係る安全性評価は、計画する多重バリアシステムによる放射性核種の長期間に亘る隔離能力を対象に行われなければならない。従って、システムの操作の基本は、この多重バリアの性能を数値的に評価し、今度の処分システム開発に参考となる情報を提供することを目的とした作業に適していることが望まれる。これらの問題を効果的に解決するには、次の様な考え方をもとにすることが妥当であるとしている。

また、前述のリスクに含まれないリスクとして、ボーリング調査等で廃棄物の一部を知らないで取り出してしまう危険性がしばしば問題となる。（SKI - TR - 90：15）では、「地下資源のない場所を処分場にすることで、ボーリング調査等によるリスクを少なくする。」。また、「制度的管理に関する全ての資料が失われた後のことを考えた場合でも、将来、ボーリング調査をするような人は、そのことにより、どのような影響があるかを判断する能力を持ち合わせているだろう。従って、現在の人間活動を前提にした評価は疑いなく安全側である。」との考え方をとっている。

（4）安全性評価シナリオ

計画するヴァーチャル・エンジニアリング・システムは、開発されたシナリオに沿ってさまざまなモデルを複雑に組み合わせ、協調させてシミュレーションを行い、経過を観察する機能を提供することができるよう開発する。そこで、シミュレーション・シナリオの情報としての扱いが問題となるため、シナリオについて概観する。

地層処分は、多重バリアの全体システムの働きによって廃棄物中の放射能が充分減衰するまでバリア中に閉じ込めておくことが前提となっている。廃棄物中の放射性核種が生物圏に漏出する経路は、地下水に溶けて移動するのが主と考えられ、火山の噴火、断層活動等による直接的な露出による被曝の可能性は極めて少ないと想定される。

前者は通常起こり得るシナリオとして被曝線量を定量的に評価し、後者の直接的な露出については異常シナリオと考え、発生確率が極めて低いことに着目した評価方法（例えばリスク評価）が導入される。

例えば、通常シナリオである地下水シナリオは次のような考え方（PNC TN1410 96-071）により示される。地下水シナリオに沿って構築されるモデル体系による地層処分システムの性能評価では、まず地下水流動と化学的特性に関するモデルにより地質環境の条件を与える。

次にこれを境界条件として、廃棄物から発生する熱の影響も含め、地下水の浸入に伴う人工バリア

内での物理的・化学的変化を扱うモデルにより、処分後の過渡状態における人工バリアの挙動を把握する。これらによって予測される環境の下に、廃棄物、オーバーパック、緩衝材等、人工バリア自体の地下水との相互作用に関するモデルにより、その性状の変化を把握し、その長期的健全性を評価する。さらに物質移動現象を取り扱うモデルにより、ガラス固化体からの放射性核種の溶出及び溶出した放射性核種が人工バリアや地層を経て生物圏に到達する過程を解析する。最後に生物圏モデルにより線量を算出する。

上記のモデル体系の中で、地質環境条件の設定に必要なモデルとして次の４つが挙げられる。

(a) 地下水流動モデル
(b) 地下水水質モデル
(c) 物質移動経路モデル
(d) 岩盤力学・熱特性モデル

多くの場合、次の事象系列が基本となっている。それらのモデルや最終的な評価対象である生物圏のモデルを組み合わせてシミュレーションを行うシナリオとして、例えば以下のようなシナリオが考えられる。

1：地下水が処分場に浸入し、緩衝材中を移行し廃棄物容器と接触ずる。
2：廃棄物容器が腐食する。
3：廃棄物容器の密封が破られる。
4：廃棄物から放射性核種が浸出する。
5：浸出した核種が緩衝材中に拡散する。
6：周辺の岩盤へ拡散・移流により移行する。
7：岩盤の断層、割れ目または水を通しやすい地層中を移流・拡散により移行する。５－７の過程で核種によっては沈殿・再溶解あるいは吸脱着を繰り返す。
8：表層に貫通する断層あるいは水を通しやすい層を通って表層の地下水流、川、湖等に漏出し、生態圏に広がる。
9：飲料水、食物等の摂取及び塵埃等の吸収により体内へ取り込む。

評価においては、この基本的なシナリオに影響を与える要因による変動を評価しなければならない。このようにして提案されたシナリオについて評価を行う場合、これらの考えられるシナリオに関連する個別の現象について、仮定とモデルを考え、具体的な評価シナリオとしてシミュレーションによる評価に持ち込むことができる。例えば、上記の条件から多重バリアシステムの性能について評価を行うためには、地下水の存在を前提とし、ガラス固化体から溶出した核種が地下水を介して人間の生活圏に至るという過程について解析することが重要である。このような過程は一種の想定であり、地下

水シナリオと呼ばれる。解析を行うにあたっては、地下水シナリオに関連する現象をモデル化することが必要となる。

地層処分システムにおいては、地下に広く存在する地下水がガラス固化体中の放射性核種を溶出させ、これを人間の生活圏に運ぶという過程への対策として多重バリアシステムが考えられている。

ガラス固化された高レベル放射性廃棄物は、オーバーパックに封入され地層中に設置される。この際、オーバーパックと地層との間には粘土質の緩衝材が充填される。これらの人工的に設けられる多層の安全防護系が人工バリアである。

一方、核種の吸着、遅延といった安全防護機能を本来的にそなえている地層は天然バリアとよばれ、人工バリアと天然バリアのもつ種々の機能を多重に組み合わせたものが多重バリアシステムである。期待される機能は以下のようにまとめられる。それらの理由から、下記の地下水シナリオを想定しシミュレーションを準備することができる。

1：透水性の低い緩衝材は、周囲からの地下水の浸透を制限し、その中での地下水の動きを極めて遅くする効果がある。これによって、ガラス固化体と地下水との接触を抑制する。

2：オーバーパックは、透水性の低い緩衝材によって地下水の移動が制限されるため、腐食が生じにくく、ガラス質に封じ込まれた放射性核種を容器内に長期間閉じ込めておくことができる。

3：オーバーパックが破損し、地下水が廃棄物と接触したとしても、緩衝材中に浸入する地下水の量は限られており、また長半減期の放射性核種の大部分は、もともと深部の地下水に溶解しにくいと考えられていることから、ガラス固化体から溶出する放射性核種の量を制限する。また、地下水に対して比較的溶けやすい一部の核種については固化材であるガラスの保持機能が溶出を制限する。

4：緩衝材中ではほとんど水の動きがないため、地下水中に溶け込んだ放射性核種の移動を拡散支配にする。また、緩衝材は放射性核種を吸着しやすい性質を持っているので、それによっても核種の移動の程度を制限する。このため放射性核種の移動には長時間を要し、この間にその放射能を減衰させる。

5：緩衝材中から地層に移動した放射性核種は、地層中の地下水の動きが緩慢であるうえ、岩石へ吸着しながら移動するため、その動きは地下水の動きに比べてさらに遅くなる。この移動期間中に放射性核種の放射能を減衰させる。また、地層の複雑な空隙構造が、地下水中放射性核種をいろいろな方向に分散させ、次第に希釈する。

図1．2－2に主要各国が検討している処分概念と、評価レポートで採用されている評価シナリオの概要を示す。評価結果は、通常シナリオに関する限り、線量限度を何桁も下回っている。また、影響が最大となるのは10,000年以後の遠い将来であるとの報告が多い。評価の例を図1．2－3に示す。

国名 評価報告書		米国 NUREG/CR1667 (1988)注)	カナダ 2nd Post-Closure Assessment Report (1985)	スウェーデン KBS-3 (1983)	スイス Project Gewähr 85	ベルギー SAFIR (1989)
処分条件	1.処分の対象	使用済燃料、ガラス固化体	使用済燃料	使用済燃料	使用済燃料、ガラス固化体	使用済燃料
	2.処分概念	ネバダの凝灰岩の地層（ユッカマウンテン）の不飽和層に処分	カナディアンシールド花崗岩地帯の地下500～1,000mの地層に処分。チタン製容器に封入し、坑道の床に掘削した処分坑道に定置し周囲に緩衝材（ベントナイト）を充填する。	結晶質岩（花崗岩）地帯の地下約500mに処分。鋼製キャニスターに居れ、坑道の床に掘削した処分孔に1本ずつ定置し周囲に緩衝材を充填する。	スイス北部の花崗岩層地下1,200mに処分。厚さ25cmの鋳鋼製容器に封入し、横坑道内に定置し、坑道内空間に緩衝材を充填する。	ベルギーの北東部モルの地下約230mの粘土層（ブームクレイ）に処分。 高レベル廃棄物を単独で処分する方法と中レベル廃棄物を一緒に処分する方法を検討中
評価シナリオ	1.通常シナリオ	地下水移行シナリオ	地下水移行シナリオ	地下水移行シナリオ	地下水移行シナリオ	地下水移行シナリオ
	2.変動シナリオ	・気象変化の伴う水理条件の変動 ・新しい断層の形成地震 ・不飽和層における地球化学的特性の変化流入水の変化 ・コロイドの形成 ・ボーリング調査 他	・基本シナリオのパラメータの変動 ・容器の早期破壊 ・緩衝材の性能劣化 ・グラウティングの性能劣化 ・氷河期来襲による気候変化 ・生活様式の変化	・スカンジナビアの隆起 ・初期キャニスタ破損 ・酸化雰囲気の変化 ・コロイドの形成	・基本シナリオのパラメータ変動 ・上部地層への井戸の掘削 ・気候変動 ・定性的評価：氷河、火山、隕石、地震、岩盤弱部変動、キャニスター移動、シーリング破損、その他	・気象変化の伴う水理条件の変動 ・新しい断層の形成、地震 ・不飽和層における地球化学的特性の変化流入水の変化 ・コロイドの形成 ・ボーリング調査 他

注）Sandia N.L.の研究レポートで評価報告書ではない。

図１．２－２　主要各国が検討している処分概念

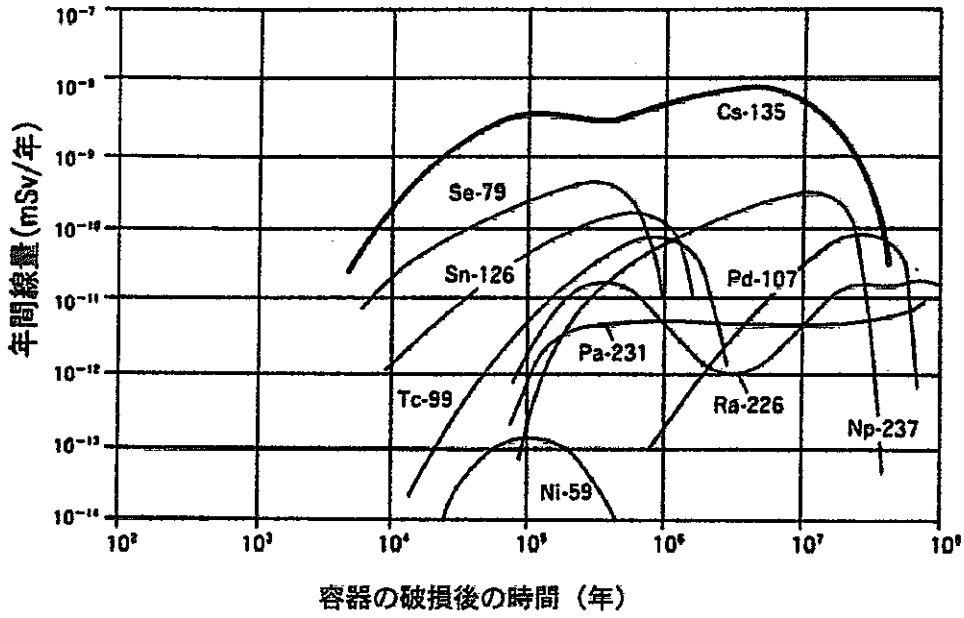

図１．２－３　スイスに於ける被曝線量評価例

地層処分ヴァーチャル・エンジニアリングにおいて評価シナリオは、別途開発しているシナリオ・データベースにより準備されたシナリオを利用することを前提としている。但し、シミュレーションにおいては、シナリオの詳細を適宜に参照することが必要である。シナリオの詳細はFEP（Feature, Event, Process）手法により記述されることが一般的と考えられる。従って、システムはこのFEP情報

を適切に管理し、与えられた評価シナリオについて補足的な情報を提供できなければならない。

FEPは、処分システムからの放射性核種の放出や輸送に直接的、及び間接的に影響を及ぼすと考えられる対象についての記述文書の集合である。それは、想定されるリスクとして各界の専門家等により示される文章とその関連からなる情報の組である。

FEPとみなされたものは、それぞれ原因及び発生の可能性、そして結果についてだけでなく、他のFEPとみなされたものとの最終的相互作用についても連係して分析されなければならない。このような相互作用には順序による影響も考慮する必要があり、場合によっては、新たに生じたFEPが既存のFEPに影響を与えることがある。処分システムの健全性の評価に関する複雑な情報を適切に取り扱うためには、性能評価の方法論に関する戦略について深く研究する必要がある。この方法論の重要な構成要素としてシナリオ開発がある。シナリオ開発の基本目標は、処分システムが超長期にわたり健全性を確保できるかどうかを評価することである。シナリオ開発の最も重要な戦略の一つは重要な問題を確認しやすくすることである。

シナリオをわかりやすく文書ファイル化することによって、関連シナリオを広範囲に再吟味し更新することができる。それを用いて多方面の専門家により見直しが行なわれることは、公正で合理的なシナリオを開発する上でも、放射性廃棄物の安全な処理という重要な問題について一般的な合意を得る上でも、おそらく最良な方法といえる。

FEPによるシナリオ開発経験からプロセスシステムと呼ばれる新しい考え方が開発された。プロセスシステムが開発された動機は、主にスクリーニング作業が予想以上に複雑で時間がかかりすぎる手法であると判断されたためである。

プロセスシステムは「処分システムやその周囲の環境におけるバリア特性、例えば放射性核種の地下水との化学反応等を説明するのに必要な、あるいは、一定の外的な条件から少なくともある程度決定論的に予測できるような、すべての現象（FEP）の組織立った集合体」と定義づけされる。

従って、それは全体シナリオといった広範な確率論を中心として扱うような領域と、特定地域の地下水の地球化学モデルといった、分解された過程である個別の領域との中間のサイズの複雑度のモデルを扱う問題に帰着する。また、個別の現象解明については、還元論に基づいて分解された現象過程と現実の実験により検証する決定論的手法がとられ、自由度を下げた方程式から数学モデルが開発される。決定論的手法の弱点はモデル単体の検証は可能であるが、それらを重ね合わせた場合に正しい解が得られるか正確な見通しを得ることが難しい事、また、理想化された実験環境から開発されたモデル式は、現象を普遍的に記述する力は一般にないことが上げられる。

プロセスシステムとは、一定の外的な条件によるシナリオが、FEPに影響を及ぼすような対象によって定義づけされる。現在の知験の範囲内で、より高いレベルに達するための手がかりとなる手法は、プロセスシステムを体系的に形成してコンピュータにより視覚化することである。

このようにシナリオに添ったシミュレーションを実現するためには、シナリオの基礎となる情報（FEPリストや調査データ、分析データ等）を総合的に管理し、シミュレーションを支援する情報を

提供する機能を開発することが要求される。以下に、FEPの例を示す。

――――――――――――――――――コロイドの形成――――――――――――――――――

（1）内容

腐食生成物は、ベントナイト空隙水化学に応じてコロイドを形成する。これらのコロイドは、核種を吸着したり、取り込んだりしてベントナイトの微細空隙を移行する。コロイドの移行は、それら微細空隙幅や表面電位の様な物理化学特性に依存する。

（2）評価（想定期間: オーバーパック破損後）

腐食生成物やそのコロイドは核種を吸着するであろう。コロイドの移行は、ベントナイトのフィルトレーション効果により重要ではないと考えられるが、ベントナイトの実効空隙径はいまだ明らかにされておらず、今後の研究が必要である。

（3）モデル化

（4）文献リスト

・McKinley,I.G.(1985): The Geochemistry of the Near-field, Nagra NTB 84-48.

（5）諸外国での取扱い

[Kristallin-I]

放射性のコロイドは、主としてガラスの溶解や沈澱過程で廃棄物近傍に形成されるであろう。ベントナイトの非常に重要な機能は、コロイドの廃棄物からの移行を妨げることで、それはベントナイトの微細空隙構造によって達成できる。ベントナイトの可塑性と膨潤性は、コロイドが移行できるような空隙や割れ目がその中に存在し得ないことを保証するものであることが期待されている。

1．3　シミュレーションの領域

いづれの評価を行う場合にも、評価期間により評価手法も評価に必要な精度も異なる。そこで著書らは次の3期間に評価期間を分けるのが良いと考える。

1．3．1　シミュレーション期間

（1）短期シミュレーション（約1000年まで。処分後の地下水浸潤等比較的過渡的な期間）

廃棄物中の全放射能量の時間変化を見ると、初期の1000年で約1／1000になることが知られている。それ以降は精練に使ったウラン鉱石全体中に存在する量と大差ないと考えられる。放射能濃度でみるとウラン鉱石の約10倍程度になる。従って、設計対応や定性的評価手法が該当する事象が起きる可能性のあるのは1000年までであり、この期間は『影響大』の事象が発生しないように設計することが望ましい。処分システムは、この間は複雑な地球化学的過程を中心とした現象に関心が集まることを重視して設計されるべきである。

また、ナチュラル・アナログ研究の一環として、歴史的資料や考古学的建造物等の情報をデータベース化し、工学的手法の参考になる情報を提供することにより設計作業を支援できる。しかし、現代の経験で確かなことが言えるのは100年程度であり、科学的予側の精度を上げるためには更に基礎研究が必要であると思われる。そのためには、この期間の現象解明に係わる、より包括的な評価を支援する対話機能が必要である。

（2）長期シミュレーション（約1000－10,000年）

放射能濃度は、1万年を経過するとウラン鉱石と同じオーダになると考えられている。従って、長期の評価に於ては影響は少ないと考えられるが、発生の確率が高いと思われる事象シナリオについて評価の関心が高まると思われるため、対話システムの機能は、より多くの気候変動や地質等のデータベースについて充実したものが求められると考えられる。

例えば、氷河期の周期は数万年オーダであり、約1万年程度であれば自然条件、生物活動等が現在と同じと考えても不自然ではないと思われる。生物圏に漏出する放射能量による影響を現在の生活条件を根拠に定量的に被曝線量として評価しても不当ではないであろう。

また、第四期（約200万年）の地殻変動データを使うことにより、地日中における約1万年以内の事象は比較的定量的に評価できる。

（3）超長期シミュレーション（1万年以上）

1万年を越える超長期の評価においては、生物活動について人類の存続の疑問も含めて不確定要素が多くなり、被曝線量を評価する意義が疑わしくなるかも知れない。ウラン鉱床中の放射能濃度と処分システム中の平均放射能濃度は大差なくなると思われる。超長期のシミュレーションにおいては、そのような理由から地質学的年代に亙るナチュラル・アナログや古気象学、古生物学等の情報も参考となるため、それらを適宜に管理し提供できる機能が必要となる。

ナチュラル・アナログ研究においては、地層処分された廃棄物から放射性核種が生物圏に漏出するのを防止、あるいは抑制するバリアシステムの機能時間と放射能毒性レベルとの関係が図1．3－1のように簡単に示されている。

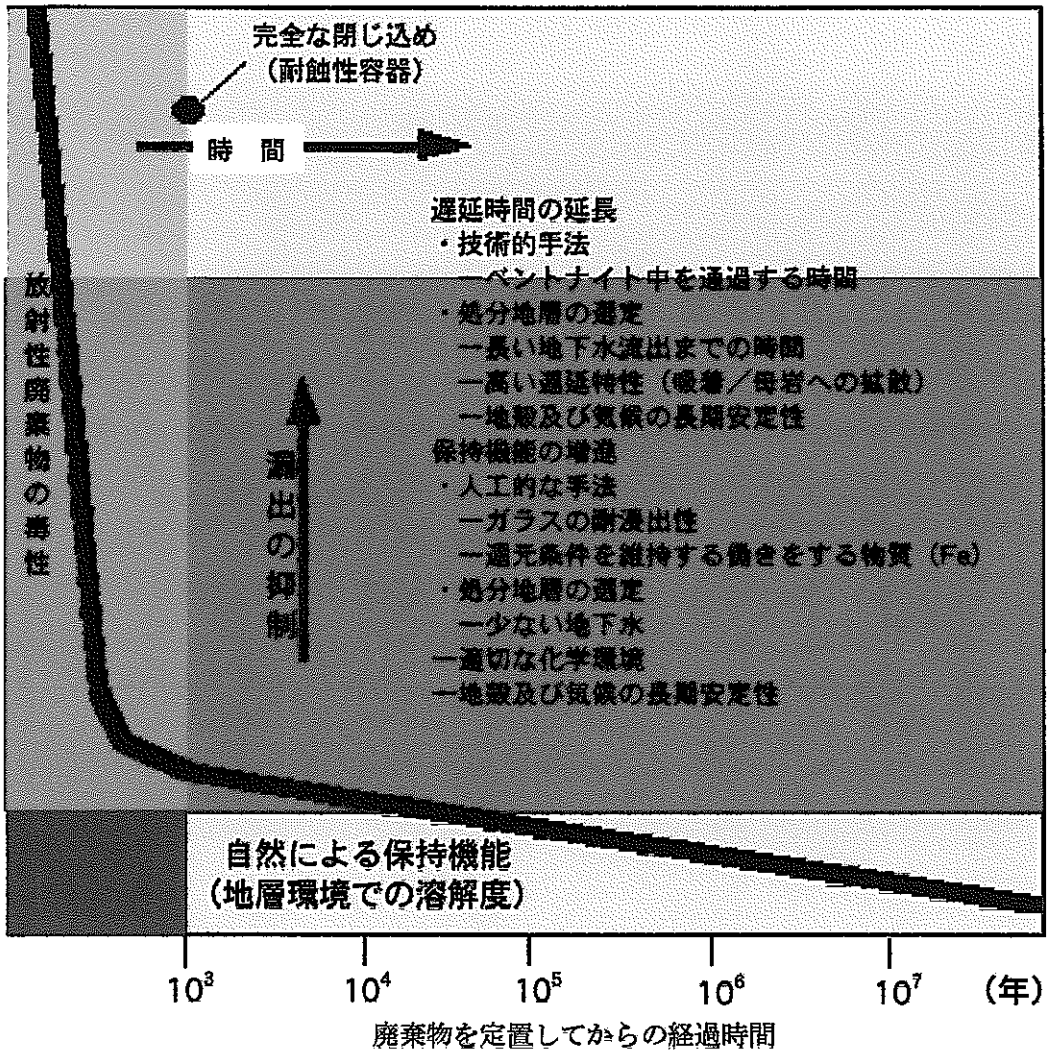

図１．３－１　バリアシステムの機能時間と放射能毒性レベルとの関係

また、放射性核種がニアフィールドからファーフィールドを経由して生物圏に至る過程を図１．３－２に示す。

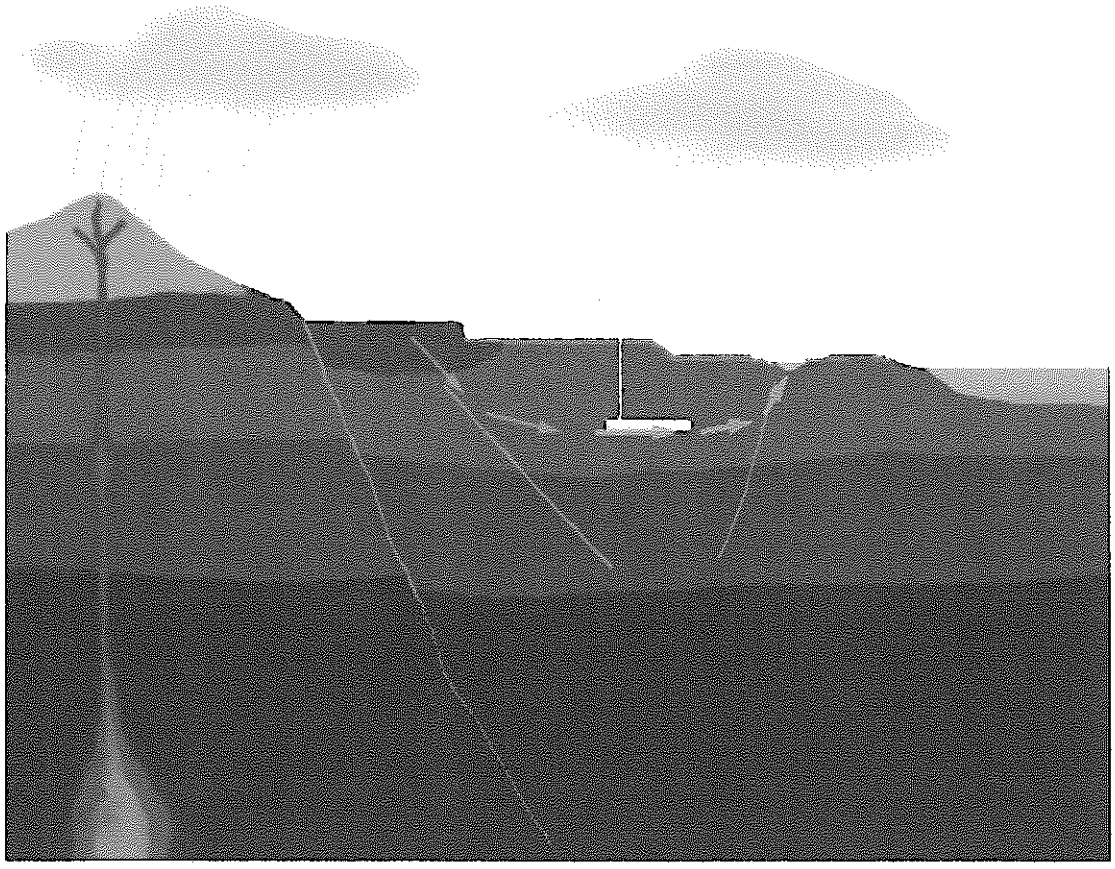

図１．３－２　放射性核種が生物圏に至る過程

容器の閉じ込め性能が確保されている間に、短半減期の放射性核種が減衰し放射性廃棄物の全体の放射能は何桁も下がる。孔が開いても残った金属容器の残骸は周囲の地下水を通元性に保ち、金属酸化物は放射性核種を吸着する。近年、この他に図１．３－３に示す処分場で発生するガスの影響が話題となっている。

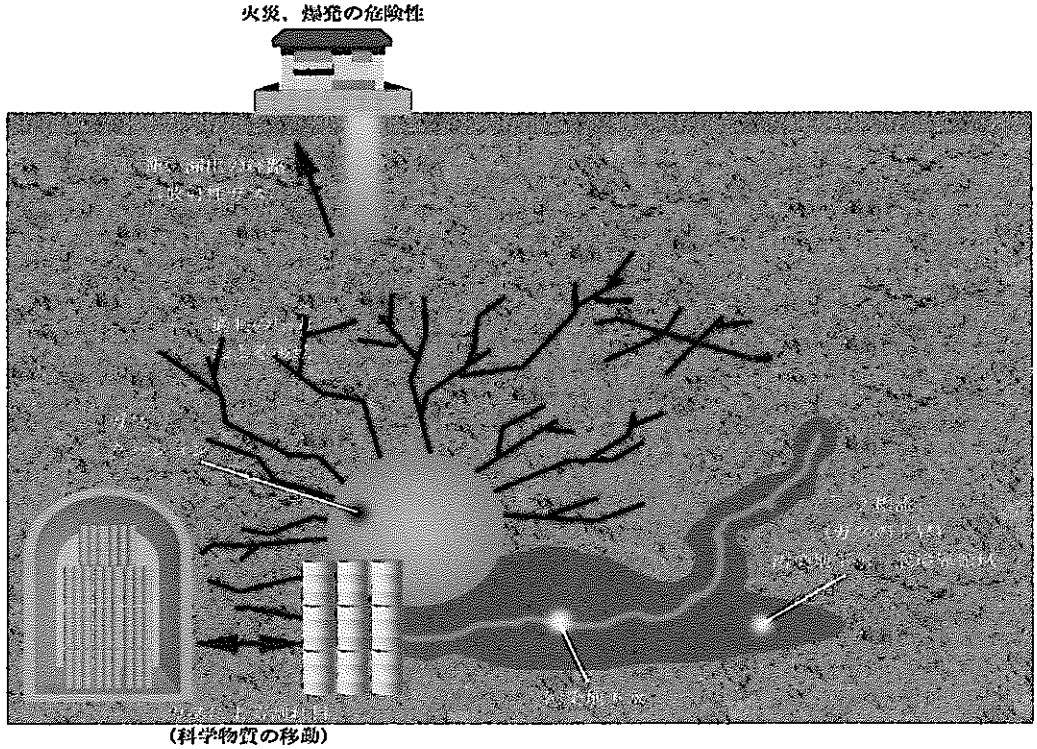

図１．３－３　ガスの影響

廃棄物が腐食して発生するガスは地下水の動きを変える。α放射線による水の放射線分解による局所的な酸化性環境下で廃棄物は溶解する。岩、ベントナイト、廃棄物及び容器の劣化生成物への吸着や酸化還元フロントでの沈殿で放射性核種の移行が遅れる。

１．３．２　ナチュラル・アナログ

ナチュラル・アナログ研究は、このような地層処分の概念と長期の核種移行過程を念頭に進める。ガスの影響や微生物の影響については、石油工学等での研究成果などを参考にするとよい。

また、廃棄物に含まれる放射性核種のなかには天然に存在しない元素がある。ナチュラル・アナログ研究では化学的性質の類似する元素の挙動で近似する。しかし、類似していた元素同志も原子価が変われば性質が著しく変わってしまう。どのような元素の挙動で近似するかは、ケースバイケースで考える必要がある。

例えば、Npについては、Np（V）はPa、Np（IV）はTh、酸化還元の変化を伴う現象の研究ではUで近似するのがよいと考えられている。TcはRe、AmはEu、Pu（IV）はTh、Pu（Vl）はUである程度近似される。

ナチュラル・アナログ研究は、研究対象によりいろいろな自然環境が使われる。最も多く使われているのはウラン鉱床である。使用済燃料の腐食溶解現象、酸化還元フロントや割れ目での放射性核種の挙動、コロイド形成等の核種移行助長現象、吸着現象や岩盤への拡散現象、半減期の違う核種を使った時間依存現象等多くの項目の研究が行われている。緩衝材中での核種移行現象やベントナイトの

変質など緩衝材の性質の研究には粘土が専ら使われている。このように、長期の評価について、当面は、天然に存在する放射性鉱物の放射線影響との比較において定性的な検討に留どめるのが適当と思われる。

ナチュラル・アナログ研究情報は、シミュレーションを試行錯誤的に進める上で重要な参考情報を与えることができる。従って、あたかもシミュレーションのヘルプ機能のような、凡そ以下の分野のナチュラル・アナログ情報について充実するべきである。

（1）ガラス

ガラス質の安定性については、失透して結晶化するため長く安定には存在しないといわれているが、アポロが月で採取した天然のガラスは3.7x10の9乗年前にできたものであり、北アメリカで採取した天然ガラスの半数以上は2x10の6乗年以上、4x10の7乗年前のものもあったとの調査結果がある。

そのほか、1.1x10の9乗年前のものも発見されており、条件によってはガラス質も非常に長い間安定に存在することがわかっている。

水による溶解速度と変質については、生成物天然ガラスと0℃の氷解水、海底火山の天然ガラスと約4℃の海水、火山灰中のガラス質と常温の雨水、玄武岩質ガラスと温泉水などいろいろな種類の天然ガラスを使い、温度もいろいろと異なる水と長い間接触した場合の溶解量や表面の変質の様子が調べられている。

海水による浸出についての調査の例では、溶解速度は3～20マイクロm／1000年程度であり、海底堆積物で埋っている場合は約0. マイクロm／1000年との値が報告されている。変質生成物は、状造を形成しており、ガラス質から可溶性成分が抜けた薄い層、非結晶性の二次生成物層、及びその上に粘土鉱物などの結晶性の鉱物の層の凡そ3層が存在する。この厚さは0－1000マイクロmである。これが代表的な変質層であり、廃棄物を固めているホウケイ酸ガラスの実験室試験で認められる変質層の構成とほとんど同じである。

今後のナチュラル・アナログ研究は、二次鉱物への廃棄物成分の取り込みについての研究に重点が移されるべきであるとされている。

考古学上のガラス製遺物は、紀元前BC500年頃のエジプトのガラス製品の保存状態が調べられている。人工のガラスでも水に接触している場合の変質層の様子はよく似ているが、湿気のある空気に接触していた人工のガラスでは、表面に天然のガラスでは認められない微細なひび割れがしばしば認められる。

（2）鉄および鋼

天然の鉄金属が見出だされることは、まれであるが、隕石、グリーンランドのDisko島及びドイツのBUhlの天然の鉄についてのナチュラル・アナログ研究の例がある.

考古学上の遺物としては、スコットランドのInchtuhil遺跡で約1,900年間、約5mの深さに埋まっていた約百万本の釘があげられる。釘の長さは6－40cmである。鉄の腐食は埋まっている場所が酸化性環境か還元性環境かで大きく異なる。

10の6乗年間以上残っているBUhlの天然の鉄は透水率の低い玄武岩に埋まっている。鉄から玄武岩の中への酸化還元フロントの移動は106年間で1＝2cmである。Johnsonらは、これまでの天然の鉄や考

古学上の鉄製造物の通元性環境での腐食の調査結果を整理して、0.1～10マイクロm/年という値を出している。

（3）ベントナイト、セメント、コンクリート

ベントナイトは、イライト化して膨潤性等の性質が変わることが心配されているが、100℃に保たれたとしてもイライト化するには10の7乗年以上かかることがわかっている。温度が上るとベントナイトが固まり割れが生じやすくなることもあるが、100度C以下では重要な変化は生じない。

コンクリート及びセメントについては、1,700年前のローマのコンクリートの調査等により、コンクリートの主要成分であるケイ酸カルシウム化合物は長期間安定に存在するとの結果が出ている。

古いコンクリートの間隙水の研究は、ニアフィールドの放射性核種の濃度評価にとって大切であり今後の課題である。

（4）溶解度と化学種

放射性廃棄物の処分は、地下水の動きができるだけ少ない場所が選ばれるのが原則であり、ニアフィールド、ファーフィールド供に平衡状態での核種濃度の評価が重要であり、地球化学モデルが多用されている。その検証のためにナチユラル・アナログ研究が行われている。

地球化学モデルは、一般に地層鉱物と地下水の特性から考えられる化学平衡式を摘出し、システム的に解く方式がとられる。この方式で求めた濃度とフィールド調査での値とはかなり違う場合が多い。そこで、高pH鉱泉のMaqarinサイトでの研究などでは、従来の方法に加えてフィールドに存在する微量鉱物まで詳しく調査し、見つかった微量鉱物のうち、どれが水中濃度を決めているかを探すという方法が取られている。地球化学モデルでの評価値は、一般に計算値より実測値の方が濃度が薄いとの結果になるが、Maqarinサイトの研究例では、CaUO4の値を使ったためウランについて逆に実測値の方が濃い結果になった。フィールドでは、CaUO4の純粋のものは発見されてなく非結晶性の物質が発見され、この物質が水中のウラン濃度を決めているであろうとされている。

一般に化学平衡が成り立っているとして計算されるが、自然界での結晶化現象など非常に遅い反応があり、その速度がわかっていない場合が多い。また、固溶体や共沈現象は地球化学モデルには組み込み難い宿命を持っている。これらの点が実測値の方が低くなる主な原因である。

（5）移行遅延現象

結晶岩の割れ目については、割れ目の内側を覆っている鉄の水酸化物、有機物及び粘土鉱物への吸着、及びそれらの物質との沈殿作用が主であることが多くのナチユラル・アナログ研究で明らかにされている。すなわち、安全性評価では吸着は可逆的であるとして評価されるが、不可逆の沈殿成分が多い。

また、鉄の水酸化物は特に重要である。吸着による遅延作用のみでなく、吸着したコロイド粒子となって放射性核種を逆に動きやすくする場合も観測されている。堆積層中での物質の拡散についてのナチユラル・アナログ研究で興味あるものとして、スコットランドのLoch Lomondでの研究がある。

淡水の湖の底に厚さ2m程度の薄い海洋堆積物がある。この堆積物は6900－5400年前のものであることがわかっている。この堆積層から淡水の堆積物への臭素の拡散分布が調査されている。その調査結

果では拡散係数は8xl0マイナス11乗m2/sであった。

堆積層での微量元素の移行調査で良く知られているものとしては、Oklоの天然原子炉に隣接する粘土層への核分裂生成物の移行状況の調査があげられる。生態圏と地圏との境界領域の移行調査については、原爆実験による放射性核種の地中分布の調査があげられる。それによると、100m深さの井戸でも9ケ月後にはRu106が検出されており、5年間の連続モニタリングで岩石の割れ目を約300m移行したことが確認されたとの報告もある。

これらの結果は生態圏での放射性物質の移行評価モデルを検証するにはよいデータである。

（6）岩盤中の拡散

ウランの崩壊系列核種を測定した例では、10の5乗年程度の間に酸化性の地下水が流れた場所の岩盤でウランが減少している所の割れ目からの距離は30－40mmであったとの報告がある。また、海水に浸潤されていた岩盤が上昇して淡水に浸潤された岩盤を使った調査がある。

割れ目に沿った間隙水の塩分濃度の分布を測定したところ、拡散係数は2x10のマイナス9乗m2/sであったとの報告がある。岩盤中の拡散は、酸化還元フロント移動との組み合わせの現象として研究されている例もある。岩盤の中に還元スポットが生ずる現象が見つかっているが、花崗岩での還元スポットでの調査では、還元性領域は約10の6乗年の間に半径約10mmしか広がらないとの調査結果が出ている。以上のような調査から、一被に放射性廃棄物の処分の評価に関係する岩盤の拡散領域は100－200mm程度を考えればよいとされている。

（7）地下水の放射線分解

Oklоの天然原子炉現象があった場所から、Moが80%、Tcが35%、Ruが25%失われている。そして、まわりの酸化還元フロントで止まっている。これらの元素は、酸化状態の高い化学形でなければ移動しないので、水の放射線分解で生じた酸素、または酸化性物質で一旦は酸化されたであろうと考えられている。そして、放射線分解で生じた水素が粘土層に拡散して形成した還元領域で還元され沈殿したと考えられている。

Cigar Lakeでの研究でも、ウラン鉱石（UO2）の表面に少量ではあるが、酸化状態の高いウラン酸化物が存在することが確かめられている。

（8）酸化通元フロント

結晶質岩については、PoCos de Caldasのウラン鉱床の岩盤の風化側の茶色の領域と青灰色との境界を使って詳しく研究されている。境界は非常にシャープである。ウランは還元側境界で濃度が増している。Mo、Ce等酸化通元に敏感な元素はウランと同じ挙動をとる。その他の酸化通元条件で化学形が変化しない微量元素も、鉄の水酸化物との共沈、固溶現象、あるいは吸着現象で多かれ少なかれ酸化通元フロントに濃縮している。そして、10の8乗年間以上移動しないで固定されている。

（9）コロイド

例えば、Po - gos de Caldasでは浅いところと深いところで調べ、浅いところでは、コロイドは主に粘土で0.8mg/l程度存在しており、深いところでは、主に非結晶性の鉄の水酸化物で0. 5mg/l程度存

在している。また、コロイド中のTh230/U234比から、ウランはコロイド中からは少なくなっていることがわかっている。高pHの鉱泉であるMaqarinでの調査でもコロイドの存在量は1mg/l程度と少なかった。

1．4　地下水シナリオによる多重バリア方式の性能評価

多重バリアシステムを採用した場合は、どの部分を各バリアの境界と考えるかが重要となる。例えば、米国NRCの基準では廃棄物パッケージと地層の2つのバリアに分けている。設計ベースで考えると、オーバーパックと緩衝材を2つのバリアと考え、更に地層バリアが加わり3重のバリアシステムと考えることができる。この場合、容器は密封機能、緩衝材は移行遅延機能、地層は隔離機能（接近できなく移行距離が長い）と、それぞれ異なった機能で核種移行を阻止する働きをする。従って、この異なった機能を期待する評価シミュレーションにおいては、これらのバリア構成要素を明確に3次元オブジェクト等で視覚化して区別することにより、整理されたシステムを構築する必要があると考えられる。

また、長期の評価においては、高放射線被曝の事象の発生はほとんど考えられないので、極めて低い発生確率であることを定量的に評価することは必ずしも要求されないと思われる。単独で閉じ込め性の高いバリアを重ねる必要はない。しかし、我が国の地層は必ずしも安定ではないので、設計目標としては、地層の不安定性に起因する不確定要素を排除することを目的とした多重バリアを採用した方が、評価が容易となることが多重バリア方式の採用論拠である。この多重バリア方式の処分システムの機能を周辺母岩と合わせて、機能別に分類評価できる対話画面構成が重要である。

また、地層データベースについても詳細な調査結果から構成でき、確実な性能評価が可能な処分システム周辺の岩盤と距離による隔離機能を期待する、広い範囲の地層の多重バリア・システム設計の支援機能を考えることができる。

処分システムに及ぼす影響事象を、サイト特定、及び設計により、かなりを除くことができると考えられている。更に、除くことのできない事象のうち、放射能の漏洩に結び付くシナリオを摘出する方法として、それぞれの事象に定性的発生確率を与え、相対的発生確率から重要事象を求める方法がとられる場合が考えられる。

また、定性的な数値は科学的根拠に基づいて与えることができるとは限らない。それらの数値の見積りに当たって根拠となる科学的データが得られない場合は、まず専門家の見識に頼ることになる。また同時に、公衆の感覚的判断も大切になる。従って、いわゆるパブリック・クライテリアの必要性が生じる。合理的なパブリック・クライテリアを形成するためには、まず関連する科学的事実を公衆に伝達することが大切である。そして、長い時間をかけ多くの人が理解できるよう整理する必要がある。

以上のことから、シナリオに添ったシミュレーションを操作する機能には、確率論的評価面から重要事象が示されることや、将来のパブリック・クライテリアとの関係、社会システムへの影響といった包括的な機能が設計当初より含まれていることが望ましいと考えられる。

１．４．１　地質環境と地下水シナリオ

当面の評価シナリオとしては、処分システムの建設時における掘削影響等からの地下水環境変動シナリオ、及びそれぞれが原因となった核種の放出シナリオを開発する必要がある。地層処分ヴァーチャル・エンジニアリング・システムの開発においては、シナリオは別途推進されている地層処分シナリオ・データベースの開発プロジェクトにより得られたものを応用する。従って、シナリオに添ってシミュレーションを進めるための機能設計上要求される事項について基本的事項をとりまとめる。

ここでは、現在提案されている（PNC TN 1410 92-081）地下水シナリオに添って、設計開発するシミュレーション・システムの基礎的な要求機能を検討する。

地下水シナリオの記述にあたっては、地下水の流動や化学的性質に影響を与える地層処分システムの場としての特徴や、そこで生ずると考えられる種々の事象、プロセスを検討する必要がある。そのため、地層処分システムを人工バリアが設置される環境である、地質環境と人工バリアの設置によって影響を受けるニアフィールドという２つの空間領域に大別し、場としての特徴を記述し、次いでそれぞれの領域において生ずると考えられる事象、プロセスを考慮して地下水シナリオを作成することとした。

地質環境は、地下水の流動状況と化学的性質、さらに核種移行等に関連する化学反応や物質移動を規定する場として捉えられる必要がある。地下水の流動状況については、広域流動系と局所的流動系とによって、また地下水の化学的性質については、地下水供給域と深部岩体における地球化学的特徴によって考えることができる。地球化学反応や物質移動に対しては、鉱物組成及び微視的空隙構造が重要である。地質環境は、シミュレーションを行う場合の仮想環境を構成する情報と考える。我が国の地質特性等から、ヴァーチャル・エンジニアリング・シミュレーションを行うための仮想地質環境を構築するには、海岸平野、内陸盆地、山岳地域、火山地域の４つに分けて構築することが適当であると考えられる。また、別の考え方として、地殻の力学的安定性について地質データ等を基に分類し、モデルとなる仮想地質を構築することも有効かと思われる。いずれにしても地層処分ヴァーチャル・エンジニアリングにおいては、仮想地質環境をデータ等から開発することが基礎的な要件であると思われる。

（１）仮想環境の構成

仮想の地質環境を構築するには次の方法が考えられる。第一の方法として、実際の調査データ（衛星やリモートセンシング・データ）から地表面に関する３次元の情報を取りだし、それをもとにポリゴナル・データ（ポリゴン）を作成する方法である。第二の方法は、特定した地域の情報が利用できない場合であり、この場合は、前述の調査データやフラクタル・シミュレーション等を利用して、理想的な仮想地形モデルを作成しポリゴン情報とする。地質情報は、実際には地表データから得られるポリゴンから作成する一枚の形状オブジェクトが基本となる。また、それは１．４．２に示す側面境界と接続し、一つの地質形状モデルを構成する。従って、前述の地表形状オブジェクトは、構築する仮想地層モデルの最上位のクラスに位置する情報と考えるべきである。また、側面情報は、それのサブクラスとなるべきである。

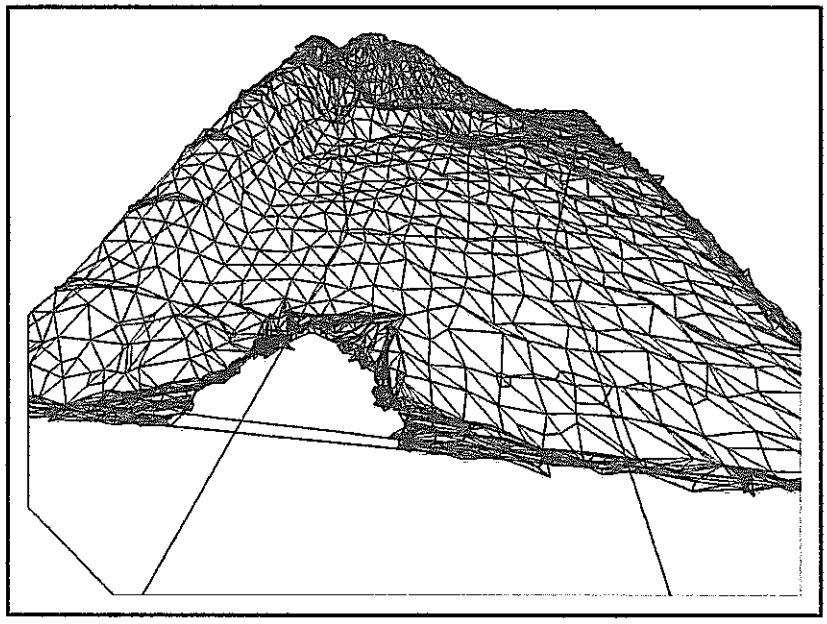

図１．４－１　地表形状オブジェクトの例

（２）エージェントの扱い

仮想の地層環境を構成するエージェントは環境要素すべてである。従って、それらのエージェントを詳細に調べることは現在は難しい。ここでは、将来において最低必要と考えられるエージェントについて示す。第一に地上処分施設が上げられる。

図１．４－２　廃棄体受け入れシーン

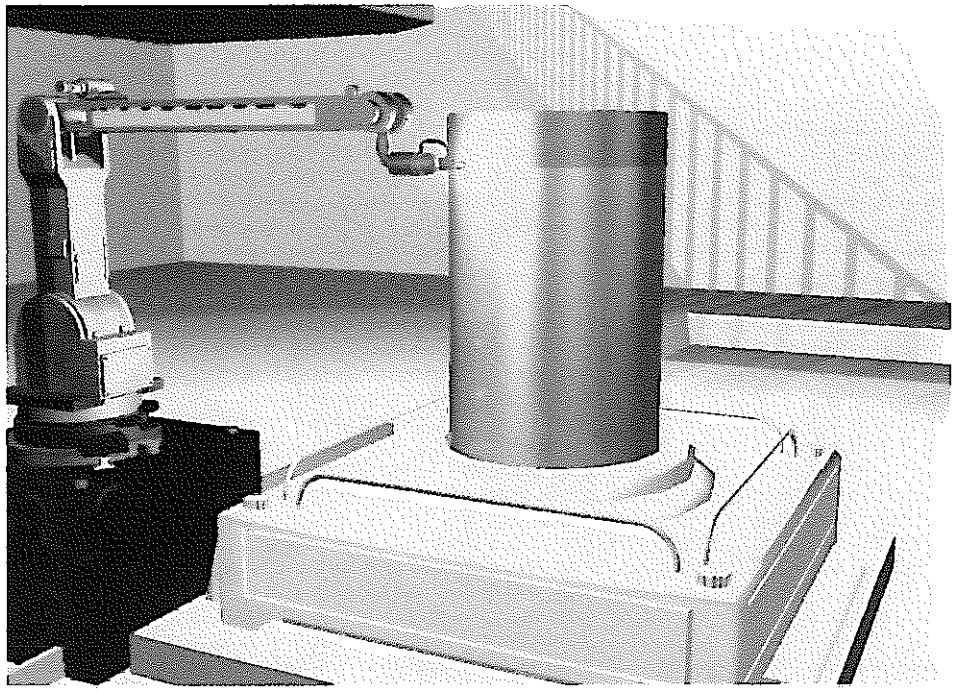

図１．４－３　オーバーパックの熔接シーン

おそらく、地上施設は処分システムと生活圏を接続する最も可能性の高い接点であろう。地上施設のエージェント要素は処分設備である。これらは、操業時には搬入された廃棄体を地下の処分ピットまで搬送し埋設する。そのモデルは、ちょうど交通のシミュレーションと類似しており、地震、電源喪失等で大きな影響を受けるシステムである。また、地下水や想定できる岩盤等、地層環境を構成するおおまかな環境要素を接続することにより、よりリアルなシミュレーションが可能となるであろう。

図１．４－２、図１．４－３は、地上施設のエージェントとして考えられる要素の試作データから動作シミュレーションの１シーンを取り出したものである。

１．４．２　広域地下水の流動

広域の地下水流動系は、地下水供給域から地下水排出域までの循環系全体の地下水流動状況を記述するものとして重要である。広域の地下水流動系をモデル化するためには、日本で観察される地形、地質に基づき、系の境界条件と系内での流動経路を規定する水理地質構造を設定することが必要である。そこで、系の上部境界条件は、生物圏の下層部にあたる土壌下面のポテンシャルによって規定し、下部境界としては、一般的に透水性が深部に行くに従い減少することから、系全体の流動状態に対して流束が相対的に無視しうるような深度を考えるものとする。

他方、側方境界は、地形、地質的要因によって、地下水系が分断される場合と他種の水系である海との境界として設定される場合とが考えられる。また、割れ目の発達した系では、断層が側方境界の一つとして考慮される必要がある。従って、鉛直断面に着目すると、側方境界のバリエーションとしては以下の組み合せが考えられる。仮想世界のとしての地形考えた場合、以下は側面の境界を与えるものである。

1：断層ー断層
2：断層ー海
3：断層ー山
4：山ー山
5：海ー海
6：山ー海

ここで、山ー山を側方境界とするパターンの中には峡谷と盆地とが含まれる。これらは、三次元的な流束の集散状況が異なることから評価対象としては区分して考える。結晶質岩、及び割れ目の発達した堆積岩においては、地下水流動系が断層により分断されていることが一つの特徴となるのに対して、新第三紀系などの割れ目の少ない堆積岩系は、平野あるいは堆積盆として形成されている。ここでは、透水性の異なる層（例えば、砂岩と泥岩等）が発達し、巨視的流動状況は、この層序に支配されることが特徴である。

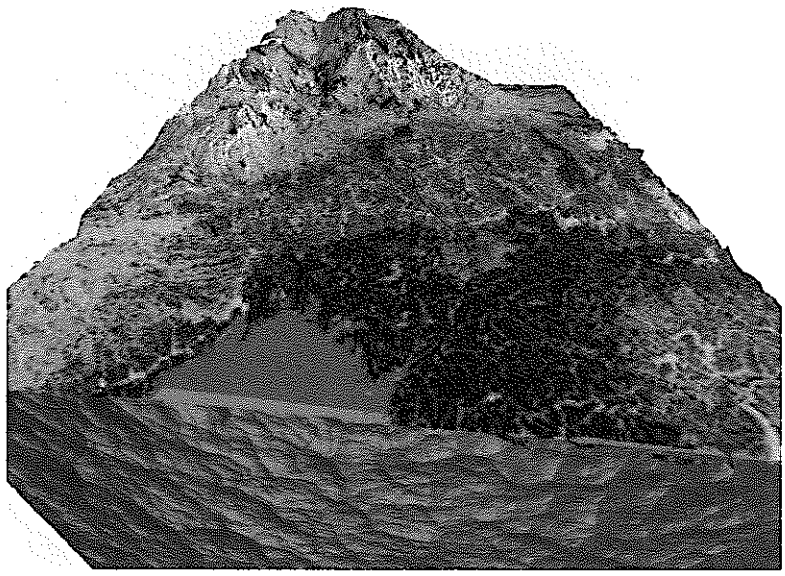

図１．４－４　側方境界を含む仮想地層モデル

また、難透水性層の下位に高透水性層が位置する様な場合、被圧帯水層が形成されることとなる。さらに、これら２種類の系が共存するパターンとして、割れ目の少ない堆積岩に覆われた結晶質岩等がある。この系では、上記の特徴に加えて、新第三紀堆積岩と結晶質岩との境界において、新第三紀堆積岩層での流動状況に対応したポテンシャル分布が生じ、これが下位の結晶質岩中の地下水流動に関する上部境界条件を規定することとなる。

（1）仮想環境の構成

広域地下水の仮想環境は、前述の仮想地層クラスのサブクラスとして接続するべきである。仮想環境構成要素は、岩盤と亀裂等広域的な浸透流の流路に影響する物理形状である。また、前述の砂岩と泥岩等、透水性の異なる層ごとにクラスを編成することも有効かと思われる。

クラス編成上問題となると考えられる対象に亀裂がある。亀裂は、それ自身では空隙であるため体積をもつが、従来の亀裂水理コードなどでは、面やジャンクションとして扱っている物が殆どである。また、亀裂は非常に小さなものまで多数分布しているため、モデルを考えた場合、ある程度の亀裂より小さなものは確率的に分布しているとして記述することが望ましいとも思われる。

（2）エージェントの扱い

地下水を与えられた環境で独立して動くエージェントとして考えた場合、次のアルゴリズムが考えられる。第一に従来の手法の発展系である。それは、従来とられてきた離散近似的手法（有限要素法や有限差分法）を基礎としたモデルの組み合わせである。例えば、広域亀裂水理を計算するコードでは、与えられた条件に基づく地下水の流れ場を時間とともに得ることができるため、例えば核種移行コードを組み合わせて利用することが考えられる。この方法は、シミュレーションを組み合わせる方法であるため、コード間のメッセージ交換をプログラムすることは可能であるが、組み合わされたコード間の数理モデルの整合性を完全に調整することは難しいと思われる。

一方、マルチ・エージェントによる協調計算を想定した広域地層の仮想環境を構成することを考えた場合、地下水流れは地下水の微小体積要素をそのままオブジェクトとしてモデル化した粒子群になるであろう。その方法によれば、個々のオブジェクトは水分子としての属性を与えることも期待できる。

問題となるのは、計算量が桁違いに異なることであるが、近い将来開発されるであろうテラフロップ級（1テラは1秒間に10の12乗回の浮動小数点演算を行う計算速度）のコンピュータを利用すれば、かなりの精度の計算が期待できる。図1．4－5は、20,000個のオブジェクト粒子を微小な流速で動かした例である。

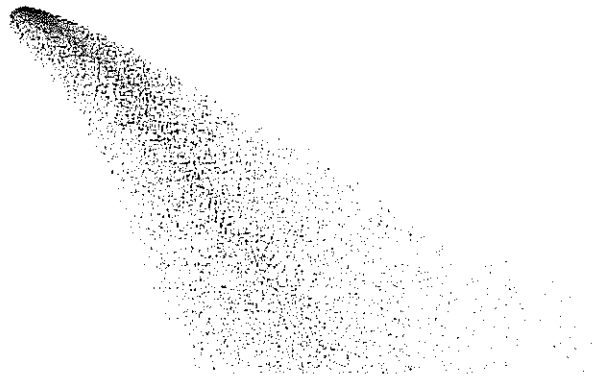

図1．4－5　簡単な微小粒子オブジェクト・シュレーションの例

１．４．３　局所的地下水の流動

結晶質岩や固結の進んだ堆積岩は一般に比較的小さい空隙率を示すが、岩体の生成時より存在している一次的空隙に加えて、その後の応力履歴等を反映した二次的な空隙、即ち割れ目、節理等を有することが特徴である。二次空隙は、岩体中の粒界等の不均質境界に応力が加わってクラックを生ずることによって形成される。形成された二次空隙は、地下水の卓越した流路となり、選択的に風化作用を受けることとなる。可塑性を有する堆積岩では、岩石形成時以降に加えられる応力に従って変形することにより、一般に大きい開口割れ目等は生じにくいと考えられる。

これらの岩石は、堆積時の粒子間空隙に対応した一次的空隙を有しており、空隙は堆積後の続成作用において、圧密による空隙の縮小と二次鉱物の沈殿過程を経る。従って、微視的構造は砕屑粒子間を二次鉱物群がとりまき、かつ、一次空隙が一部残されたものとなっている。このような堆積岩においては、一般に顕著な層序や異方性が認められる。特に、泥岩中に砂岩層が存在する場合には、それが相対的に高透水性の地下水流路となる。

（１）仮想環境の構成

局所的地下水を取り巻く環境は、花崗岩や玄武岩のような結晶岩質である。それは処分システム全体を取り巻く環境である。従って、このような領域について仮想環境を構成するには、結晶岩質の岩盤、処分システム全体が主な要素となるであろう。評価にあたっては小さな空隙による、水理、岩盤力学、温度変化等に関心が集まると思われる。

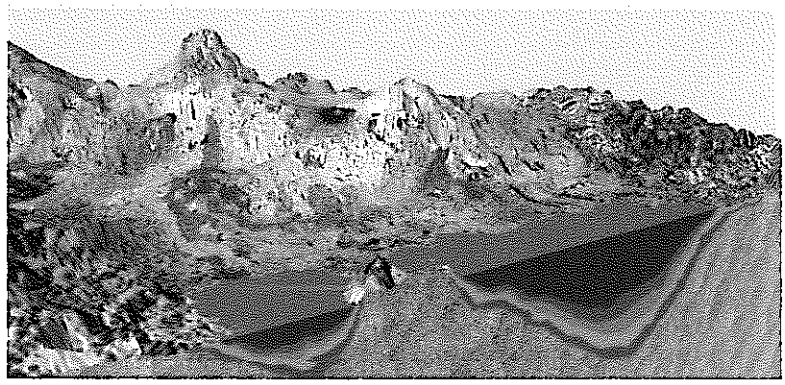

図１．４－６　岩盤と地層のモデル例

仮想岩盤のモデルは、ボーリング等の調査データや前述の生成データを利用して地層をつくることが良いと考えられる。図１．４－６は、試作したオブジェクトから視覚化したものである。

（２）エージェントの扱い

花崗岩や玄武岩のような固い岩盤中のほとんどの流れは、不連続な破砕帯（割れ目）を通して流れる。破砕体の流れのモデルは、従来より次のようなアプローチがなされている。

例えば、十分大きなスケールにおいて、一連の小さな割れ目は等価な多孔質媒体としてモデル化できる。しかし、処分場近傍あるいは断層のような大規模な破砕帯に関しては、特に破砕帯専用のモデ

ルを用いるべきである。破砕帯内の流れをモデル化するアプローチには、不連続破砕帯（ディスクリートフラクチャ）法と二重有孔（デュアルポロシティ）法がある。二重有孔法は、岩層モデルと破砕帯モデルを重ね合わせたモデルである。それに伴う移動関数は、岩層と破砕帯間の流体の移動の関係を表している。従来の不連続破砕帯モデルは、一般的に割れ目中の流れがほぼ平行板間の流れで近似できることを仮定する。また、岩層中の流れは多孔質媒体中の流れとして解析する。

破砕体をオブジェクトとしてモデル化することは概念としては可能と考えられる。図１．４－７に示すような調査データ等をもとに、破砕体をランダムな形状の体積をもったポリゴンの組み合わせで生じた隙とする手法として考えることができる。

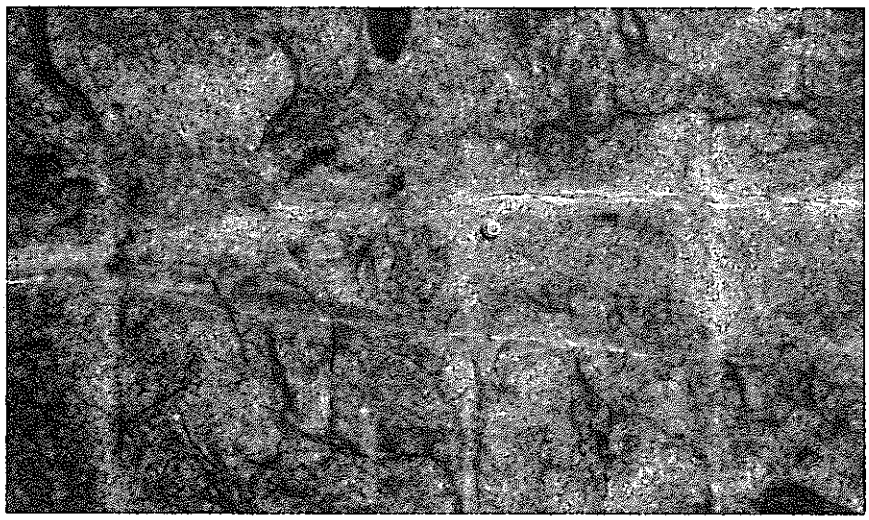

図１．４－７　亀裂の例

問題は、ポリゴンを生成するにあたり、必要なランダムな局面生成のアルゴリズムを決定することと考えられる。いずれにしろ、この方法はミクロからマクロまでの一環した破砕帯モデルの記述を可能にすることが期待できる。そのような手法で生成された３次元空間中に詰め込まれた不規則な形状のポリゴンが接する隙は、前述の微小粒子モデルで示した水粒子の流路を形成する。そのとき、水粒子と岩質との収着や沈殿等の現象はクラス階層として組み込むことが期待できる。そのような手法は、地下水の流れと化学的性質を個のモデルとして組み込むだけで全体の流れ実験を数値的に行うことが可能であろう。但し、計算速度は現在の超高速コンピュータとは桁違いのものが要求されるであろう。

１．４．４　地下水の化学的性質

（１）地下水供給域

地下水供給域は、地下水の水質形成に関する地球化学的プロセスの初期条件を考える上で重要な要素となる。この領域においては、広く土壌が分布しており、生物圏との接点として有機物を介した化学反応が活発に生じている。特に、有機物の腐敗は、地表水に含まれる溶存酸素を消費することで深部地下水が還元性を維持する一条件となる。また、花崗岩や石灰岩の風化によって生じた表層は植物の成育に必要な養分に乏しく、従って、植生もそれほど発達しないため、有機物を介した化学反応も

活発とならない傾向がある。

(i) 仮想環境の構成

地下水供給源の仮想環境の構築は、ボーリング・データ等より物理環境のクラス階層をポリゴンを形状モデルとして組み立てることが期待できる手法と考えられる。

地下水と一般に呼ばれている対象では、地下にあるすべてがすべて地下水ではない。土壌の粒子を取りまく水がその量を増してくると、土粒子の間で保持されにくくなって、さらに下方に移動するようになる。このようにして地下に集まった水は、水を通しにくい層の上に集まって、さらに横の方向に移動するようになる。このような状態にある地下の水を地下水と呼んでいる。地下水の発生は2層のモデルとして考えることができる。物理モデルは土粒子であり、これに付着する水粒子のモデルは、オブジェクト指向では前述の破砕帯と同様な手法で土粒子モデルを構成し、水粒子の飽和条件を適当に与えることにより、地下水を発生させることができると思われる。

地下水は、飽和的に地下の空隙を満たして存在している。これを飽和帯という。このような水体の上面より上の方は水で飽和されているのではなく、むしろ空気の方が多い。このような部分を通気帯と呼び、これと飽和帯との境界が地下水面である。地下水は図1．4－8のように分類される。

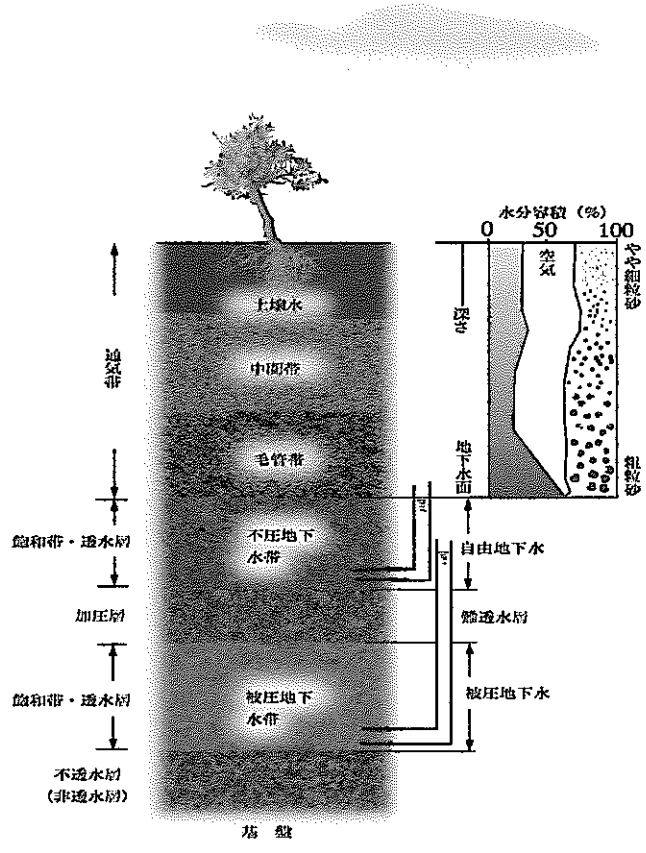

図1．4－8　地下水の分類

固結していない礫や、砂のように水をよく通す地下の地層を透水層という。シルトや粘土のように水を通しにくい地層を難透水層という。これに対して、固結した岩盤は非透水層であるが、難透水層と区別する必要のないときは、これらを不透水層とよんでいる。地下の状態は、透水層と難透水層とが重なり合って互層をなしていることが多い。このような帯水層中の地下水は、上下の難透水層に挟まれて被圧されている。これを被圧水という。被圧されていない地下水を自由水という。地下水が発生する限度は涵養と地質構造との相互関係によって決定され、これを水文地質構造と呼ぶ。地形・地質と地下水との一般的関係を表１．４－１に示す。

表１．４－１　地形・地質と地下水の関係

大地形	小地形	地質	賦存形式	地下水位
平地	沖積谷，海岸平野，三角州，扇状地，氾濫原，砂丘	沖積層（洪積層）	自由地下水 被圧地下水 一部水脈	浅い
台地	隆起海岸平野 隆起扇状地 段丘	洪積層（第三紀層）	自由地下水 被圧地下水	比較的深い
丘陵	丘陵（主として単斜・向斜の構造を有する）	第三紀層	被圧（一部自由）地下水，一部水脈	深浅様々である
火山斜面	火山斜面 溶岩台地	溶岩・軽石火山岩屑	裂か水脈，一部水層あり	湧泉以外に地下水乏しい
山地	山地・準平原	基盤台	裂か水脈 洞穴水路	湧泉のみ

（建設省河川砂防技術基準（案）より）

表１．４－１から解かるように、地下水のモデルを仮想環境として構築する場合は、第一に地形・地質のモデルを表１．４－１の分類を参考に作成するべきである。この作業は、ボーリング・データ等を参考に理想化したオブジェクトとしてデータベース化することが重要である。いずれにしろ、地下水シナリオを中心に評価する場合は、この地形・地質の条件が基本的な環境要件となるため、それらのデータベースはシステム開発の必須事項である。

(ii) エージェントの扱い

エージェントの扱いについては、地質構造上の開孔にこよって地下水の発生、運動や地下水面の形成が異なるので、地下水を次のように分類して考え、発生と流れのメカニズムを前述の地形・地質のデータ上に組み合わせる手法が必要である。

・**間隙水（pore water）**

堆積物、とくに地質時代の新しい沖積物の中の地下水で、開孔の大きさは毛細管の程度である。水はこの間隙の間を縫って流れる。

・**割目水（fissure water）**

レツコ水ともいう。固結した、あるいは結晶質の岩石の中の地下水で、開孔の大きさはもちろん超毛細管であるか、毛細管状の枝をもっている。水は間隙水のように流れたり、空洞水のように流れたりする。

・**空洞水（cavern water）**

洞穴水、またはカルスト水ともいう。開孔は大きな導管、洞穴などで石灰岩や溶岩などの中の空洞中の地下水がこれに属する。地下水は貯留されている量がその大部分で、地下水体に供給される量、またはこれから流出する量の何十倍、何百倍である。地下水体は、地下の岩石中で平衡を保つ貯水池を形成している。重力または、揚水によって引き出されるのは、ほんの一部分で、これをその地層の産出率という。残りの大部分は地層によって保留され、利用できず、これを保水率という。

地下水体は主として地質構造によってその範囲が限定され、その地下水体の形態が不圧帯水層、被圧帯水層、漏水性被圧帯水層などのいずれを形成するかは、地質構造の差によって判別できる。

このように地下水体を規定する地質構造を水文地質または水理地質構造（hydrogeological structure）と呼んでいる。水理地質を図１．４－９に示す。

つぎに帯水屑の水理地質構造であるが、地形図、地質図や現場調査からその発達状態を推定する。地形、地質から判断される一般的な帯水層の発達は次のように考えられる。

・**河谷**

現川の河床下、あるいはそれに沿って透水性の良い砂礫層からなる、旧河道堆積物が帯状に発達し、伏流水が流れていることがある。不圧地下水の場合が多いが、粘土質地層に被われていると被圧地下水の賦存することもある。

・**堆積盆地**

砂礫層、浮石層、粘土層が規則的に堆積し、地下盆地を形成している。盆地中央に掘られた井戸は自噴することが多い。

・**堆積盆地**

周辺部地下水の重要な滴養地域で、帯水層が河床下に直接露出する場合は、川の流量の減少が認められることもある。

以上のように地下水供給域と流れは、水分地質図によりとりまとめられた情報を参考にオブジェク

トやモデルを組み立てることが適当であると考えられる。但し、これらは一般に浅部地層領域のモデルであるので、深地層処分においては当面問題視されないであろう。従って、地下水の発生とその種類、それらの地球化学的性質がモデル化の課題と考えられる。

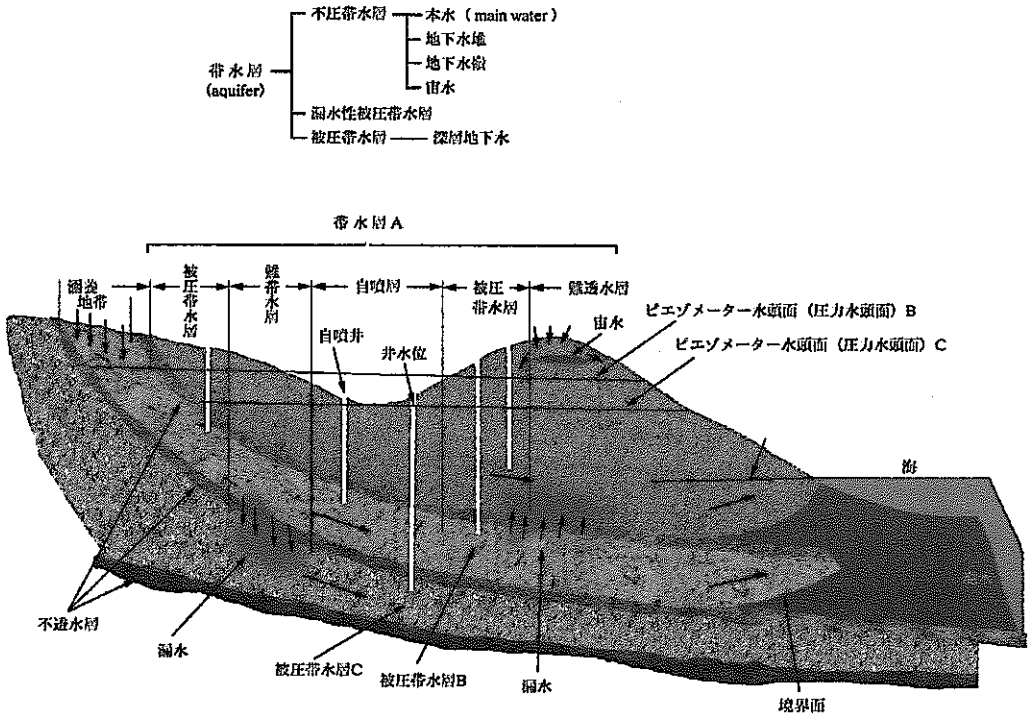

図１．４－９　水文地質図

（２）深部岩体地下水の化学的性質

深部岩体中の割れ目を有する系においては、二次空隙の近傍において、粘土鉱物、鉄鉱物及び炭酸塩鉱物に富む場合がある。また、反応生成物である二次鉱物、特に粘土鉱物が滞留するような条件下においては、二次空隙はこれらの鉱物によって充填され、割れ目のもつ水理学的な不連続性を次第に失う。このような微視的構造に対応して原則的には、二次空隙中には循環水が、一次空隙中には岩石の生成時点の水（固結の進んだ堆積岩の場合、堆積時の海水が陰イオン排除により閉じ込められた化石水等）が満たされていることとなる。

しかし、実際にはこれらのそれぞれ異なる水は、溶質の拡散により混合されているのが普通である。他方、割れ目の少ない堆積岩中では、粘土鉱物に富む層で、かつ、陰イオン排除が生ずる程度に地圧の高い深部で形成されたものにおいては、化石水が空隙を満たしているものと考えられる。

これらの２条件のうち、少なくとも一方を満足しない層においては、循環水が存在すると考えられる。この場合にも、これら２種類の水は溶質の拡散による混合作用を受けているものと推定される。

(i) 仮想環境の構成

深部岩体の仮想環境を造る基本点な手順は前述してきた方法と基本的に同じである。

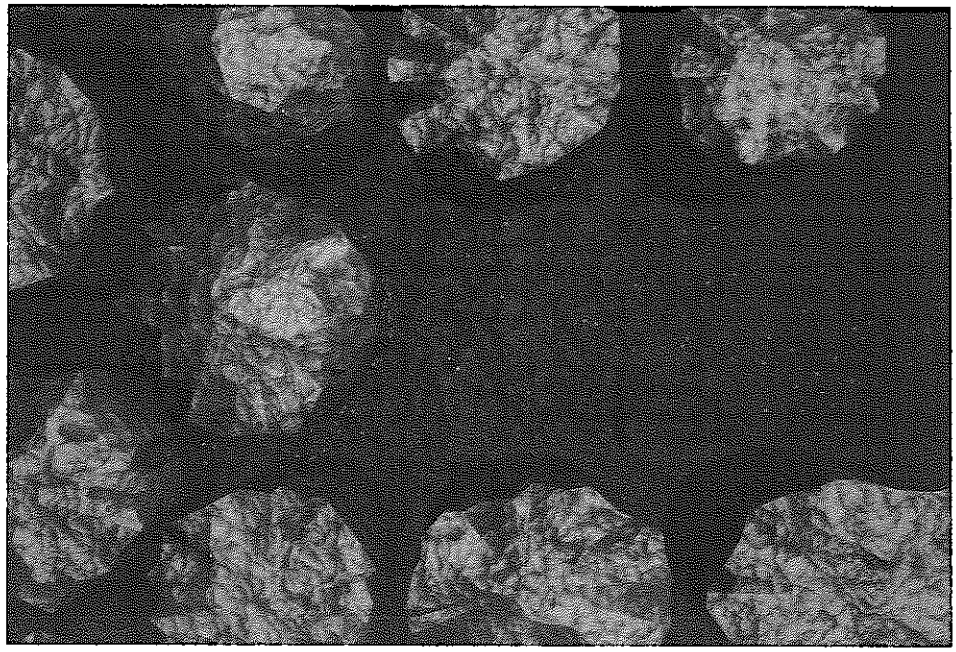

図１．４－１０　空隙と溶質移行

図１．４－１０は、試作したオブジェクト・シミュレーションから視覚化したものである。これは、乱数より生成したポリゴン球を用いた空隙の構成例を示している。球体ポリゴンの表面特記を乱数生成して微小岩片を近似し、それを適当に立体空間に詰め込むことにより、恰も砕いた岩や砂を箱に詰めたようなモデルを造りあげることができる。

但し、深部岩体地下水の環境の関心ごとは、核種の溶質移行にかかわる地下水の地球化学的性質にあるため、岩体自体は前述してきたモデルが利用できる。但し、空隙については、空隙率が影響するKdなどをモデルとして考えた場合に重要となる物理モデルである。従って、仮想環境を構築する上で注意すべき点は、空隙の再現であると考えられる。

(ii) エージェントの扱い

深部岩体地下水の化学的性質に係わる関心ごとは、地下水と地下水に含まれる不純物、岩体中の無機物、及び廃棄物（人工バリア）との化学的相互作用を評価することである。それは、この相互作用が及ぼす核種輸送の遅延効果等を期待することを含んでいる。従って、この問題は、核種輸送媒体としての広義の地下水の浸透流解析に含まれると考えられる。浸透流解析については、鉱物組成及び微視的空隙構造にて考察する。

深部岩体地下水の化学的性質について、それが個々の独立した水分子の振る舞い（タスク）として動作するオブジェクトを考えることができる。その時、水溶液中の分子は化学平衡、酸塩基礎反応、電離平衡等の反応や、沈殿・溶解、酸化還元反応等、化学的振る舞いをする。分子と分子の間には、

いろいろな引力があり、これが分子の熱運動とつり合って、気体、液体、固体など物質の巨視的な状態を作り、また溶解性、吸着性などの性質を成分分子に与えている。

従って、独立した水分子のオブジェクトを考えた場合、それが前述のような反応を起こすための性質としてオブジェクト指向モデルの考え方を与えることができる。例えば、水分子というクラスを考えた場合、それは質量を持った微小粒子（インスタンス）に物理的性質を与える変数（引力等の物理関数）を定義すると考える。このように物質自身をインスタンスとして考え、そこに働く力を関数とすると、一つの分子のオブジェクトをモデルとして考えることができる。

このオブジェクトは、相互にメッセージパッシングすることにより、互いの状態を変化させることができると考えられる。そのようなオブジェクトを充分な量だけ準備して、初期条件を与えて計算を行うモデルを考えることができる。

その時、個々のオブジェクトが相互にメッセージを交換し、物理関数として与えられる状態によって性質を決めることが出来ると思われる。勿論、原子レベルに落とすことも考えられるが、計算量等から暫くは現実的でないと考えられる。

- **ファンデルワールス力（関数：Van der waals power）**
  ファンデルワールス力は分子間に働く弱い引力であり、分子が近づいたとき現われる一種の静電気的な力である。
- **水素結合（関数：hydorgen bond）**
  ファンデルワールス力より強い、分子間引力である（2～8kcal／mol）。これはO、N、Fなどの強陰性元素の原子が水素原子を介して行う結合である。
- 配位結合（関数：Coordinate bond）
  配位結合は電荷の移動を伴って形成される共有結合であり、イオン結合と共有結合の中間の性質を持っている。
- **分子（イオン）の重さ、密度、大きさ（インスタンス）**
  分子（イオン）間に働く力のほかに、分子の重さや密度、大きさそのものも性質をあらわす。
- 溶解（関数：dissolution）、分配（関数：partition）、沈殿（関数：precipitation）
  液体の分子間引力は、完全に混じり合う場合と、一部しか溶け合わず2相に分かれる場合とが生じる。溶解、分配、沈殿はそのような性質を表す。
- **吸着（関数：adsorption）**
  気体または溶液中の分子が、間体表面に吸着される現象も以上と同様の親和力で説明される。

１．４．５　鉱物組成、及び微視的空隙構造

（１）結晶粒の等方的分布

砂岩、深成岩及び片麻岩では、基本的に結晶粒が等方的に分布し粒間に空隙が存在する。このような空隙中の物質移動の支配的メカニズムとしては、基本的には拡散を考えることができる。ただ砂岩については、水理学的条件によって、移流が物質移動の支配的メカニズムとなる場合もあると考えられる。

花崗岩類、及び同系の片麻岩を構成する主要な鉱物は斜長石（Naに富む）、アルカリ長石、石英及

び黒雲母であり、特に斜長石、アルカリ長石については、変質に伴う空隙が見られる例が多い。ハンレイ岩類及び同系の変成岩においては、これに代わって斜長石（Caに富む）と角閃石、輝石、カンラン石が主要鉱物となる。

また、砂岩については、石英を多く含むが、同時に、長石、雲母、粘土鉱物、炭酸塩鉱物、及び硫化鉄等の鉱物を含む。花崗岩類についても、変質層の近傍においてはこれと同様の鉱物組成が考えられる

(i) 仮想環境の構成

深部岩体地下水の化学的性質の場合と基本的に同じ考え方やクラスを利用できる。基本的な考え方は次のとおりである。鉱物組成の及ぼす影響は岩盤力学と中心とした物であるため、結晶岩質等の鉱物質のクラスを考える。例えば、それは立方体で密に詰めるものとする。また微視的空隙構造を再現するため、微視的な亀裂を持つクラスとしてポリゴンによる不均一な多面体を考える。次に、対象とする微視的空隙構造をもつ岩質を空間として考え、先ほどの２種類のクラスを適当な配分（乱数等）で詰める。そのようにして、微視的空隙構造を含む結晶岩質の仮想環境を造ることができると考えられる。

(ii) エージェントの扱い

考えられるエージェントは、微小鉱物組織と水分子のオブジェクトである。これらの協調は前述の水分子の性質等の物理量を関数とした連係のモデルを考えることができる。

（２）粘土粒子等の異方的分布

泥岩、頁岩、結晶片岩においては、シート状の粘土粒子等が方向性を持って重なり合い、そこに、石英や長石の粒子及び有機物片や黄鉄鉱が入り込むという配置を示す。曲ガラス質マトリクス火山岩は、生成時に比較的速やかに冷却されたことにより、非晶質のガラスをマトリクスとして有する。この中に、黒雲母、石英、斜長石（Naに富む）あるいはカンラン石、揮石、斜長石（Caに富む）が点在する。また、ガラス質部分の変質に伴い、ある領域がスメクタイト等の粘土鉱物によって占められる場合もある。

(i) 仮想環境の構成

深部岩体地下水の化学的性質の場合と基本的に同じ考え方やクラスを利用できる。基本的な考え方は結晶粒の等方的分布と同じである。

(ii) エージェントの扱い

考えられるエージェントは、微小鉱物組織と水分子のオブジェクトである。これらの協調は前述の水分子の性質等の物理量を関数とした連係のモデルを考えることができる。

（３）浸透流モデル

鉱物組成、及び微視的空隙構造の間を流れる地下水のモデル化手法は、一般に地質媒体の特性によ

り２つに大別した方法が取られている。第一に水の飽和状態になっている岩盤中の流れであり、第二に不飽和状態にある岩整中の流れである。これらの流れのモデル化にあたって、熱の影響を入れるか入れないかで更に分割して考えることができる。

(i) 飽和流れモデル

飽和流れモデルは、等価多孔質媒体に起き換え得る多孔質媒体や破砕媒体中の地下水流れ分布のモデル化である。それは、等温での流れ状態や、流れが圧力と熱により決定される一般的な非等温での流れ（熱一流れ連成モデル）を記述する。等温での飽和流れモデルは、処分システムの近傍では温度上昇域が流れ場に影響を及ぼすような現象に限って役に立つ。熱流れ連成モデル（非等温）は、熱により引き起こされた流れ（対流）と、伝導や移流による熱移動の相互作用を記述する。この種のモデルは、ニアフィールドの流れを解析するために利用する。

等温飽和流れ 非等温飽和流れ

(ii) 不飽和流れモデル

不飽和流れモデルは有孔性媒体のモデルである。不飽和流れ状態は、処分場建設施行段階と初期貯蔵段階で岩盤が排水する処分場近傍で生じる。このモデルには、基本的に以下の３種類の不飽和流れモデルが考えられる。

空気相を無視した不飽和等温琉れモデル 空気と水の二相不通和等温流れモデル 蒸発と水の非等温流れモデル

等温不飽和流れモデルは、ファーフィールド評価等に用いられる。ニアフィールドや処分場設計にそれらのコードが適用できるのは建設施行時の水流入量から透水性を逆算して求めるような場合に限られる。

(iii) 破砕媒体中の流れコード

花崗岩や玄武岩のような固い岩盤中の処分場では、ほとんどの地下水の流れは不連続な破砕帯（割れ目）を通して流れると考えられる。スケールを充分大きくとれば、一連の割れ目は等価な多孔質媒体としてモデル化できる。しかし処分場近傍や断層のような大規棋な破砕帯に関しては、特に破砕帯専用のモデルを用いるべきである。破砕帯内の流れをモデル化するアプローチには、不達続破砕帯（ディスクリートフラクチャ）法と二有右孔（デュアルポロシティ）法がある。二有右孔法は２つの媒体を重ね合わせたモデルである。一つは岩層であり、もう一つは破砕帯である。

移動関数は、岩層と破砕帯間の地下水の移動を記述している。不連続破砕帯モデルは、一般的に割れ目中の流れがほぼ平行板間の流れで近似できることを仮定して造られる。岩層中の流れは、多孔質媒体中の流れとして解析する。

(iv) 被砕体中の溶質移行モデル

ほとんどの破砕媒体中の溶質移行コードは飽和流れ状態を仮定している。これらのモデルは媒体が十分に飽和に達した後に、処分場近傍で廃棄物からの放射性核種移行を評価することを目指している。破砕媒体中の溶質移行を解析する際に特に重要点は岩層への吸着である。

１．４．６　ニアフィールド問題

ニアフィールドは、処分施設と人工バリアの影響が大きいという意味で、地下水流動の観点からは局所的流動系の特異な領域と考えられる。ニアフィールドは埋め戻された坑道群と人工バリア、及びこれらが建設されたことによる影響を受けた周辺母岩から成る。

（１）坑道群

廃棄物を定置する処分坑道は、複数のパネルに分割され、各パネルの外周は主要坑道と呼ばれる資材や人員運搬用の坑道で囲まれる。また、地上施設と地下施設との間はアクセス坑道で連絡される。建設、操業及び閉鎖は、パネル単位で行い、処分場全体として見れば、これらの工程が並行して進むことがあり得る。

(i) 仮想環境の構成

仮想の坑道群は設計情報等から３次元のデータとして作成する。この時、独立したエージェントとして扱うべきと考えられる要素（例えば、アクセス坑道、処分坑道等）は、個別のクラスとして定義する。また、それらの接続は力学系の関数を利用して行うべきである。それにより、仮想の坑道群は、地震や断層等、近くの力学的条件に応答することができるであろう。図１．４－１１は、約１００万ポリゴンで生成した縦置きの処分坑道全景の配置である。図が示すように仮想坑道は、処分坑道、アクセス坑道、緩衝材、オーバーパック、固化体の独立したオブジェクトを接続して構築している。従って、接続関数を利用することにより外部の力に応答して変形する手法が考えられる。

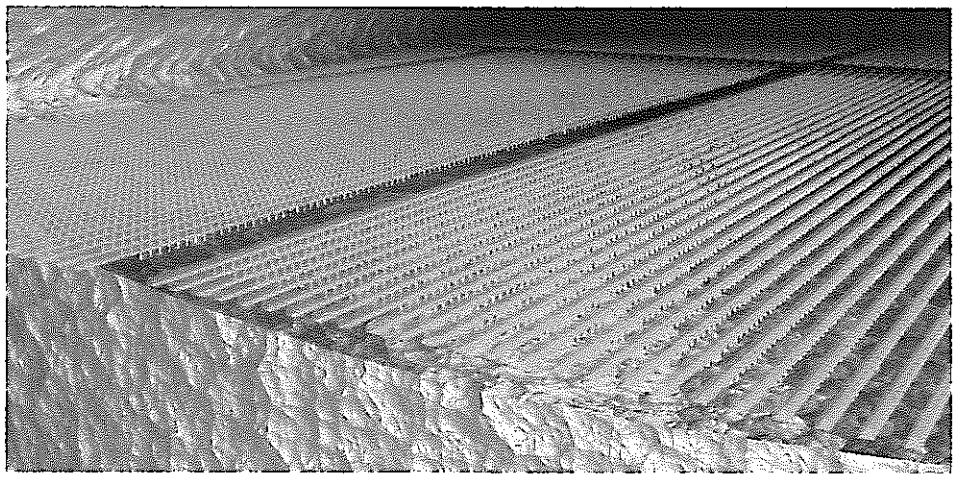

図１．４－１１　仮想竪置き処分坑道全景

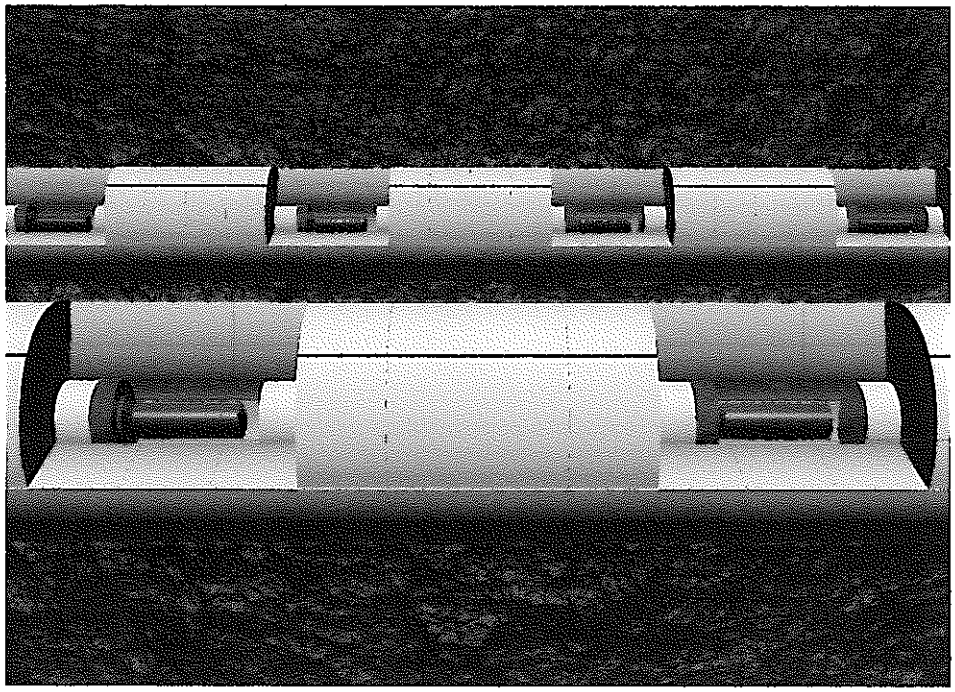

図１．４－１２　仮想横置き埋設方式坑道全景

図１．４－１２は試作した坑道オブジェクトから視覚化したものである。この程度のポリゴン数（100万）からなる仮想世界を用いて、視覚的に有る程度の満足感が得られる量の水分子エージェントモデルを用いたシミュレーションは、恐らく数十テラフロップスの計算速度（現在再高速の超並列計算機の約1000倍の計算速度）のコンピュータを利用することができるようになってから可能であろう。図１．４－１３に図１．４－１２に示す仮想坑道の内部を示す。

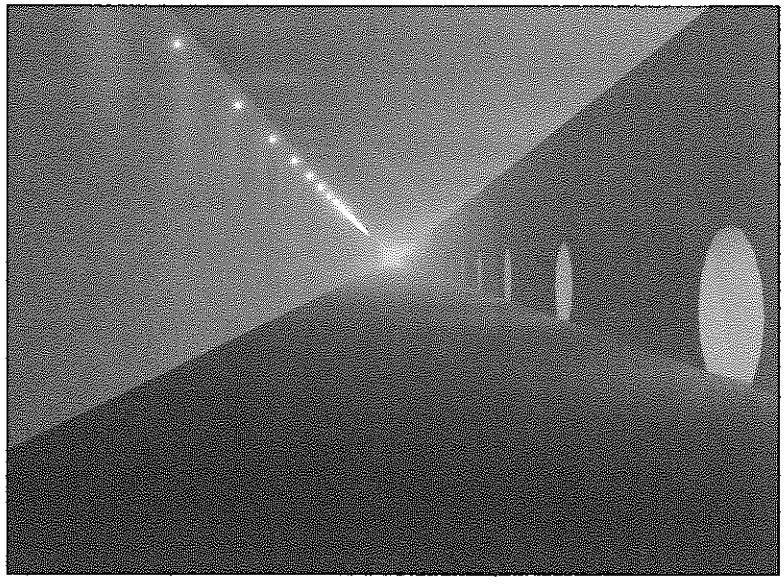

図１．４－１３　図１．４－１２の内部

また、並列計算システムを利用することにより、より説得力のあるシミュレーションが期待できる。オブジェクト指向による、シミュレーション・プログラム開発には新しい考え方が必要でありその影響はモデル開発にまで及ぶものと思われる。それは、シミュレーションの考え方を革新する思考方法と捉えることもできるであろう。

地層処分ヴァーチャル・エンジニアリング・システムの開発プロジェクトでは、複雑なモデルを試行錯誤的に組み合わせてシミュレーションを実行するシステム開発技術の基礎的な部分の試作と評価を行なった。その結果、以下の新しい概念とシステム開発技術の成果が得られ、これからの計算科学技術の新しい成果が期待できる。

従来研究開発されてきた、科学技術のトップダウン・アプローチ（現象全体を確率論的モデルで記述したもの等）とボトムアップ・アプローチ（現象全体を個別の要素に還元して個別の決定論的モデルで記述してゆく手法）とは別な、第３の手法としてモデル協調計算という手法を考え導入した。

モデル協調計算

コンピュータ中に組み立てられた仮想世界の中で、複数の数値計算モデルを協調させて並列的に実行することにより、従来のトップダウン・アプローチで押さえきれなかった、より精密な現象過程や、ボトムアップ・アプローチで記述が難しかった、ある程度の自由度をもった現象のシミュレーションを計算で実現する方法とした。この、モデル協調計算は、理論、実験といった従来の科学技術の方法とは別な、第３の科学技術の方法である計算科学技術の位置づけを明確にする科学思想とした。

モデル協調計算は技術的に可能である

モデル協調計算を実現するための基礎技術は、オブジェクト指向によるシミュレーション、協調計算結果を実時間で３次元可視化する技術、異機種の複数計算機に機能分散を行ない１システムとして統合する技術を確立するとこにより基本的に可能であると考えられる。

システム開発には個別の専門的知識より広範な知識が必要と思われた

システムの設計開発にあたり、現象評価に係わるシナリオ、個別現象過程、そして、ソフトウェア・エンジニアリングについて広範な知識が必要であると判断された。個別の専門領域に限定してきた従来の技術者教育の今後課題として考えられる。

（2）人工バリア及び周辺母岩

人工バリアは、ガラス固化体、炭素鋼オーバーパック、及びベントナイトの緩衝材によって構成され、処分坑道内に水平に設置（坑道横置方式）あるいは処分坑道に垂直に掘削された処分孔に設置（坑道竪置方式）される。周辺母岩は、処分坑道の掘削により、力学的、水理学的に影響を受け、ゆるみ領域を形成する。人工バリア周辺の地下水の流動については、既に述べた母岩の局所的流動系における異方性、不連続性の向きと処分坑道の向きの組み合わせによって種々のパターンが考えられる。

(i) 仮想環境の構成

人工バリア及び周辺岩盤の仮想環境は、前述の仮想坑道を構築するクラスのサブクラスとして次の順序のクラス階層を構成する、周辺母岩、緩衝材、オーバーパック、ガラス（キャニスタ付き）である。

なお、埋設方式に竪置きと横置きの２通りがあるので、それらは別個のオブジェクトとして分類して構成するべきである。オブジェクト間の接続（例えば、オーバーパックとベントナイト）は、人工バリア内の複雑な現象を記述する複数の関数によるものとして考えることができる。例えば、母岩の応力、オーバーパックの腐食等、場の状態や時間で変化する物理量を接続関数として考えることができる。図１．４－１４、図１．４－１５及び図１．４－１６は、竪置方式の試作したオブジェクトから視覚化したものである。

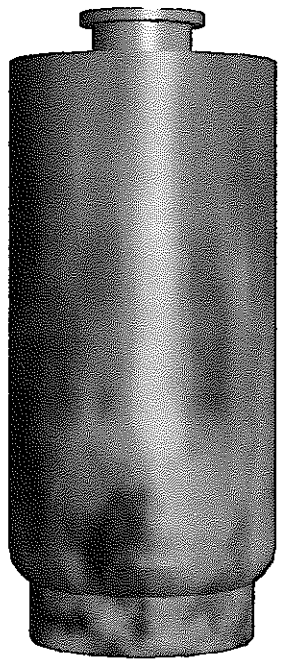

図１．４－１４　竪置方式　固化体キャニスタ全景（TVF）

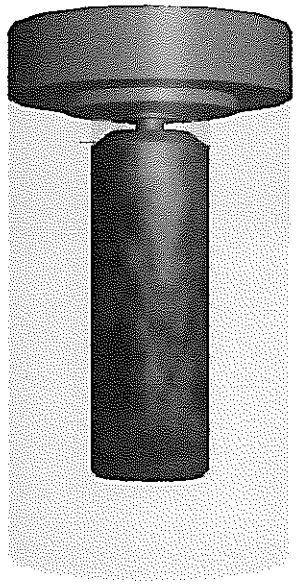

図１．４－１５　竪置方式　固化体キャニスタ＋オーバーパック

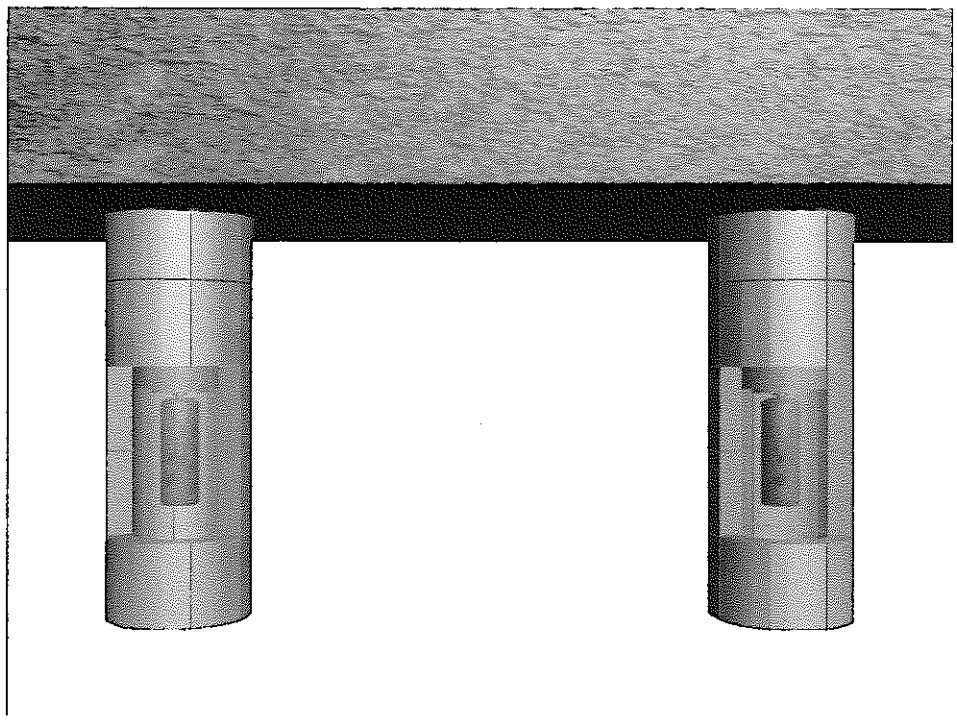

図１．４－１６　竪置方式　固化体キャニスタ＋オーバーパック＋ベントナイト

図１．４－１７、図１．４－１８、図１．４－１９は、竪置方式の試作したオブジェクトから視覚化したものである。

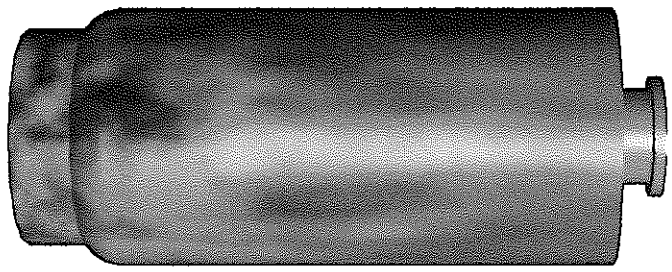

図１．４－１７　横置方式　固化体キャニスタ全景（TVF）

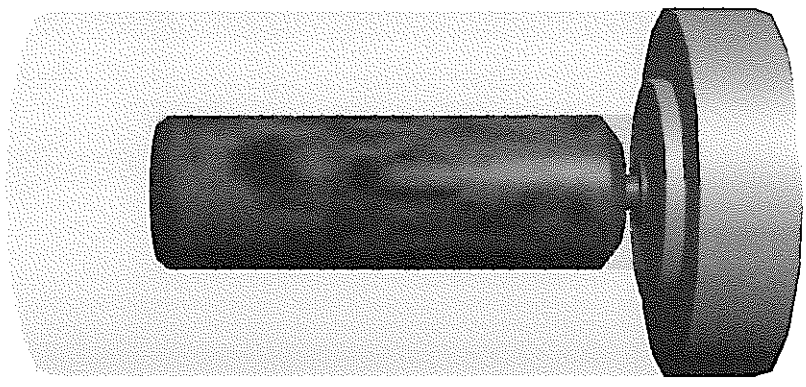

図１．４－１８　横置方式　固化体キャニスタ＋オーバーパック

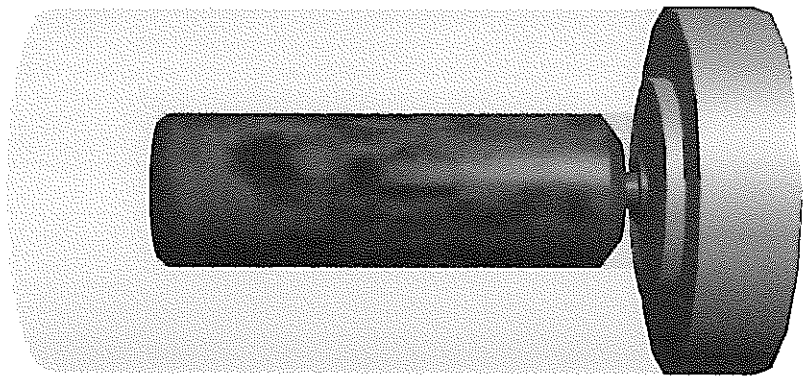

図１．４－１９　横置方式　固化体キャニスタ＋オーバーパック＋ベントナイト

１．４．７　連性問題へのアプローチ

性能評価では、地下水の流れとともに移行する核種の異なるプロセスが連成した現象にかかわる知見が必要である。このような連成モデルの研究が、熱伝達、地下水挙動、岩盤力学および地球化学を含む処分システムの長期的性能に影響する重要なプロセスに関する、理解を深めるため多面的にすすめられる必要がある。この研究の焦点は、まず異なる地質媒体ごとに、どの連成相互作用が重要であるかを系統的に明らかにする事である。連成モデルを開発するためには、その準備として以下の事項を事前に整理することが必要である。

- 連成プロセスの分類とリストアップ
- 現在の知識でとらえ得る限りの処分システムの長期性能に影響する各プロセスの重要度評価
- 各プロセスの不確定性のランク付け
- 不確定度の低減化を計れるキーとなる課題の解決に向けての研究法の検討

例えば、流体・熱システムや、処分システム周辺と同様の条件であると想定される天然岩石・水理システムに関する情報の整理が必要である。そのようにして、上記の研究課題が適正に位置付けられ、どのような条件の場合に連成相互作用が重要となるかをより良く理解できるようになる。処分システムの性能予想には、現実に発生し得る主要なプロセスに対する合理的な見通しが必要である。

主要な物理化学的プロセスは、熱（T）、水理（H）、力学（M）および、化学（C）である。これ

らの４つのうちから、考えられる連成の組み合せは11種（4C2＋4C3＋4C4）である。なお、連成のモデルには、各プロセスが１方向的に時系列で連成する形式と、相方向的に連成する形式がある。

これらの各連成プロセスを全て包括するような評価システムを決定論的に導くことは難しいであろう。いくつかは、解析できるようになり始めているものもあるが、全く手が付けられていないものもある。オブジェクト・シミュレーションは、そのような限界を越える一つのアプローチである。以下、各連成関係を記述するプロセスを示す。これらのプロセスの中の一部は、メンバー関数等としてオブジェクトの動作を決定する法則として記述できることが連成モデル開発として期待できる。

並列に動作する独立したタスクとしてのオブジェクトをエージェントとして定義し、それらを協調させてシミューションをすすめるマルチ・エージェント・モデル協調計算の手法で連成モデルを開発することを考えた場合、従来の複数本の支配方程式を連立させて、差分法や有限要素法で離散近似するシミュレーションに比べ、遥かに複雑のモデルを造ることが期待できる。何故ならば、個々のプロセスは実験等により定式化されて得られるため、それを独立した関数として受け付けるクラス階層は自然な連成記述を可能にするためである。

強連成　熱と化学のプロセス。

熱拡散
相変化（固体の平衡状態）
固体の分解
準安定固体

強連成　熱と流れのプロセス。

浮力流れを含む１相あるいは２相対流
相変化とInterference
熱的物性変化
熱的浸透
ガス拡散
毛管・吸着

強連成　熱と岩力のプロセス。

熱クラック
破壊変形
熱破砕
熱クリープ
熱膨張

強連成　化学と流れのプロセス。

分離、同位体変換
錯体、共沈

溶解
沈殿
吸着・脱離
酸化環元反応
加水分解
酸性反応
拡散
化学浸透と限界濃過

強連成　流れと岩力のプロセス。

水力学的破壊
間隙水圧変化
破砕帯の水力学的エロージョン
粒子の沈殿
剥離を引き起こすせん断効果
亀裂開口の変化
粒子の濾過

化学（弱）熱（強）流れ（強）化学のプロセス。

溶解、沈殿
時間依存の溶解沈澱
浸透による流体移動
気相での化学的移行
気体と固体の分離
粒子の移動（コロイド）
平衡状態
熱拡散
熱的浸透

熱（強）岩力（弱）化学（弱）熱のプロセス。

鉱物相の相変化
脱水
クリープ
脱水とスウェリング

熱（強）岩力（強）流れ（強）熱のプロセス。

水力学的破壊
地震の引き金。（応力再分配、間隙水圧）

岩盤の割れ目の開閉挙動（応力再分配、熱的接合、間隙水圧）
破砕
強度変化（応力腐食、加水分解）
水和作用

岩力（弱）化学（強）流れ（強）岩力のプロセス。

このプロセスは考慮しない。何故ならば、熱的な効果が無ければ、物質移動は、低温での沈殿、応力腐食、及びイオン交換性スウェリング（50℃程度以下）を除いて、形状変化につながるようなものではないからである。

熱、岩力、化学、流れの複雑な連成モデルのプロセス。

貫孔作用（選択性溶解とトンネル腐食）、非均一浸透
沈殿（破砕帯、母岩）
岩の水熱的変性
溶出と沈殿に伴うヒートパイプ効果
オーバーパックまわりの鉛直方向蒸発（擬縮サイクル、圧力溶解）

１．５　数値実験オブジェクトとして考えた評価モデル

処分システムの安全性評価のためのモデルは、放出核種の移行過程等を表現する複数のモデル（サブモデル）から構成されている（評価モデル・システム）。これらのモデルは、対話システムから見ればバックグラウンドで独立したタスクとして動作するオブジェクトとして扱うことが好ましいと考えられる。また、それらは、対話システムのウイジェッドや他のタスクからのメッセージで起動され、並列的に協調して動作する必要がある。サブモデルは、基本的には核種の移行過程をできる限り正確に表現するものであるべきであるが、一方、評価モデル・システムは総合的な解析を行える機能が必要なため、サブモデルは簡単なものが望ましい。

詳細モデルに於ては、我が国では火山による飛散、地殻変動による露出等の直接被曝シナリオも他国より慎重に評価する必要はあるが、基本的にはサイト特定でそれらの要因を削除できると考えられる。

例えば、地殻のゆっくりした上下動等、必ず起きる地殻変動として評価しなければならない事象もあるが、その場合は、境界条件の時間的変化による地下水の動きへの影響を考慮することになる。これらのことから、我が国においても、地下水による核種移行モデルが評価の基本となると考えられる。

従って、モデルの選定とシミュレーションの実行においては、サイト特定で削除できる要因等を明確に表示し、評価シナリオを簡素化する操作を視覚的に表現することが好ましいと考えられる。

１．５．１　各種のモデル

（１）容器の耐久性評価

人工バリアは固化体（キャニスタ入り）、容器、緩衝材からなる。容器の健全性は、核種の移行の開始時期を決めることになる。容器は完全封入機能を持ち、腐食等の破損による開口部が生じた時点

でバリア機能を消失するとして評価する。小さな開口部であっても、一旦開口部ができると、バリア機能が他のバリアの機能に比べて極度に低下する。容器は、キャニスタとオーバパックの組み合わせを考え、代表的材料として、キャニスタはステンレス鋼、オーバパックは炭素鋼が提案されている。（図１．４－１５、図１．４－１８参照）。

容器の耐久性の評価は、腐食速度の評価と地震等の瞬間的な破壊につながる要因の有無の評価にわけて考える。また、全面腐食及び局部腐食の速度に影響する要因として、温度、酸性度、地下水の組成、放射線線量率、酸化還元電位等について考える。シミュレーションに於ては、これらの環境条件の時間変化を求め、その環境条件での腐食速度の積分値として腐食厚さを計算する手法等が用いられている。オーバーパックの腐食現象に係わる評価は、化学過程を中心としたシミュレーションになるものと思われ、シミュレーション結果の視覚化表現には新たな手法の開発が必要である。

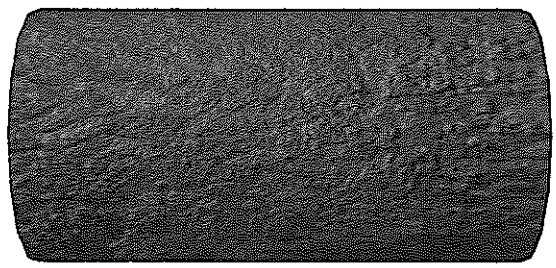

図１．５－１　オーバーパックの腐食概念

図１．５－１は、試作したオブジェクトから腐食を視覚化したものである。また、瞬間的破壊としては応力腐食割れと機械的破壊が考えられる。これらの破壊については、あらかじめ割れの生じる環境条件を実験的に明らかにし、その環境条件が処分環境にないことを確認する必要がある。

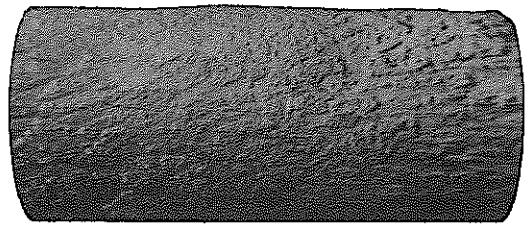

図１．５－２　オーバーパックの変形概念

機械的破壊については、破壊応力以上の外力が加わらないことを評価する。これらの要因が生じた時点で、容器は破壊しバリアとしての能力はなくなるものとする。これらの破壊についても、対話システムは優れた視覚化機能が要求されると考えられる。

（２）核種浸出評価

貯蔵時、輸送時の評価においては、時間に比例した浸出が生じるとして浸出率により評価する。処分システム操業後のソースターム評価においては、処分環境の地下水中での平衡反応から決まる溶解度モデルと拡散、及び地下水の動きに伴う物質移動により評価する手法が取られている。

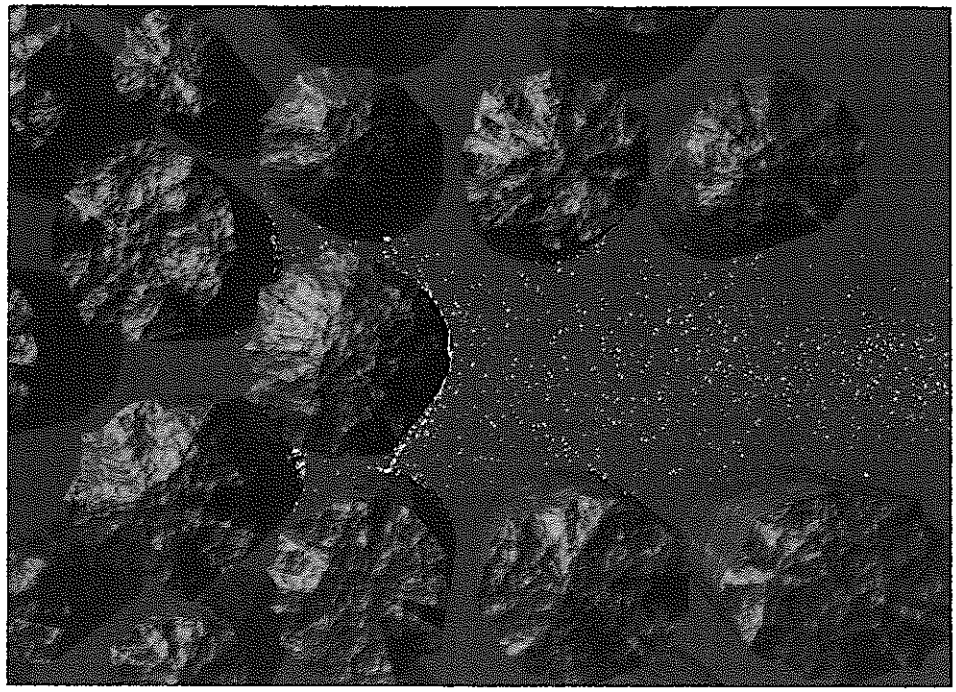

図１．５－３　物質移行の例

溶解度モデルによる評価のパラメータとしては、処分環境の温度、地下水の組成・pH - Eh、共存固相（ガラス、容器、緩衝材等）の組成が挙げられる。放射線は水の放射線分解をもたらし、Eh - pH等を決める２次要因として重要である。これらの平衡反応による評価要因に加えて、より溶解度の低い結晶性化合物にゆっくりと変換し、実質上核種移行評価系から除かれる、いわゆる鉱物化も重要な評価シナリオである。

このような地球化学シミュレーションについては、対話システムは熱力学データベースと連結したパラメータ・サーベイや感度解析機能を提供し、評価作業を支援する機能が求められる。また、化学反応を視覚化する手法の開発も大切であると考えられる。

地下水の流れについては、人工バリア内での地下水の動きは糧めて遅いことから、主な物質移行機構は拡散になるので、ガラスの表面での核種の溶解度及び緩衝材中の分散係数が物質移行評価の重要な評価パラメータである。ガラス固化体及び人工構造物の特性は人工的に決まるため、この部分には不確定要素は少ないが、浸出評価における不確定要素は主に自然環境の不確定性による。

従って、長期に亙る地下水流れを視覚化し、前述した化学過程を重ね合わせる等の表現技術の開発が対話システムに必要である。それにより、非常に緩慢な拡散中心とした流れに添って反応する過程の視覚化表現が期待できる。それは、核種の放出と輸送の過程をシミュレーションすることを目標としている、地層処分ヴァーチャル・エンジニアリング・システムの対話機能開発の重要な課題と考えられる。また、その手法は次項で触れる地層中での核種移行評価についても同様である。

（3）地層中での核種移行評価

地層中での核種移行評価も基本的には核種浸出評価と同じ構成である。しかし、地層中においては評価上最も重要な部分は地下水の動きである。特に、破砕帯等の母岩に比べ極端に水が通りやすい、いわゆる水みち（水理）の分布である。また、母岩の間隙水は一種の貯留タンクであり、母岩の中への分散速度も重要な評価要因である。従って、地形、地層構造、母岩及び水みちの透水性、母岩中のイオンの分散係数が主なパラメータであると考えられる。

地層中での化学反応モデルでは、可逆反応モデルと鉱物中に取り込まれ、実質上、溶液系から不可逆的に取り除かれる鉱物化モデルの組み合わせで考えることができる。原子価によって著しく溶解度の異なるNp、U、Trにとって還元性物質の存在は評価上重要である。

即ち、溶解度、鉱物化の反応速度、還元物質の含有量とその還元特性が重要な評価要因である。これらの要因にまで立ち入った情報を自然環境で得るには、更に反応の種類、基礎データ（熱力学データ、反応速度等）に関する研究が必要であり、評価環境を想定して必要な吸着分配係数、溶解度を実測し、より汎用性のある基礎データとしてまとめていくことが必要である。

ソフト面では、PHREEQE等で代表される多成分系での核種の安定な存在状態を求める計算コードの開発を進め、更に反応速度、濃度依存性等を考慮した吸着移行評価コードを充実し、組み合わせていく必要がある。データベースについては、熱力学データや吸着係数の収集が考えられる。

（4）生物圏中の核種移行評価

生物圏に到達した放出核種が人体に至るまでの過程は、原子炉、再処理施設等の地上施設から放出される核種の挙動と類似であり、そのモデル化及び関連パラメータについては多くの研究が行われている。そのほとんどが適用可能と考えられる。ただし、TRU等の長半減期核種に関する情報、データは、まだ十分ではないので、この分野に関する一層の研究が望まれる。従って、生物圏への評価に備えて核種の崩壊等のデータベースや移行率、量等の作図機能等が機能に求められる。

1．5．2　モデル関連パラメータの不確定性について

地殻変動、地震等の接近シナリオと呼ばれる範疇に含まれる自然現象は確率的事実であり、いつどこで発生するかを決定論的に予想することはできない。しかし、過去の発生事象の統計とプレートテクトニクス仮説から、ある程度推定することができる。

その手法として、第四紀における地殻変動の統計的データを使い、長期評価に適合するデータを得る。これらの現象による大きな影響はサイト特定で避けられるが、地震や雑続的隆起沈降の影響は考慮しなければならない。これらは、地下水による核種移行シナリオに影響する不確定性である。従って、地層処分ヴァーチャル・エンジニアリング・システム設計には、将来これらの地殻変動や地震等の事象を与えられる機能が求められる。

地層の透水性、地下水の組成、岩石の組成等、地層の特性情報は、原理的には処分サイトが決定されれば測定によって求めることができるが、サイトを破損しないで行う測定精度には限界がある（掘削影響）。そこで一般的に、類似地層での観測結果の統計処理により、それらの特性値を求めざるを得ない。具体的候補サイトが決まっていない現時点での評価では、地層の大分類の特性の統計的扱いをすることになると考えられる。

従って、地層処分ヴァーチャル・エンジニアリング・システムは、仮想のサイトにて評価シミュレーションを進めるための情報として、前述の地下水や岩石の組成等、幅広く情報をデータベース化して参照を可能とするとともに、地形のオブジェクトや設計する処分システムを仮想環境としてコンピュータ内に、それらデータベースと関連させて構築できるものであることが望ましい。

それらを構築する上での不確定な事項として下記が上げられる。これらの不確定さの原因については当面の間、事実情報を詳細に記述するデータベースを構築して、優れた対話機能により随時評価することが望まれる。

（1）人工バリア関連パラメータの不確定性

材料及び施工等の品質管理上の不確定性がある。地層処分システムでは、地層特性の不確定性がそれに比較して大きいので、その不確定性は無視できる。更に重要なことは、人工バリアの健全性の不確定性は環境条件の不確定性に起因すると言うことである。特に経時変化が重要である。例えその経時変化が予測できても、長い時間の経時変化であれば、その条件での実証試験ができなく、モデルを使った予測に頼らざるをえないと思われる。

（2）生活環境関連パラメータの不確定性

遠い将来の人類の居住状況を予測することは非常に困難である。そこで現在の社会が変わらないものと仮定する。従って、サイト固有のデータを使うことは難しいであろう。陸地か海洋か、砂漠か農耕地か、井戸の利用か川の利用かと言う程度の区分で被曝経路シナリオを作り、評価することになると思われる。

（3）測定精度関連パラメータの不確定性

以上のようにパラメータそのものの特性として幅を持っている。また、測定技術が不充分である場合、技術はあってもサイトの損傷を避けるために使えない場合がある。後者の場合は類似サイトを詳細に調査することでその精度を高めることができる。

（4）長期の不確定性に対する保証関連パラメータの不確定性

長期の事実を限られた期間で実証することはできない。それを補う一つの方法としてナチュラル・アナログ研究がある。評価事象の機構のモデルを作り、その事象のナチュラル・アナログを探し出し検討することにより実証的データを得る。この方法の難しさは、それが置かれた環境条件の経時変化を把握することが困難である点である。

（5）不確かさへのコンピュータによる演繹的推論

以上のような潜在的なモデルの不確かさに対する戦略として、知識工学分野の演繹推論等の応用がシステムのシミュレーション支援機能として期待できる。

演繹とは、確かとされる前提から出発して、厳密な推論を重ねて新しい知識を導くことである。前提となる知識は、公理や命題、定理であり、推論には、三段論法、数学的帰納法、帰謬法などが使われる。そこから導かれるのは、再び命題や定理となる。この数学で使われる演繹的な推論の形式を整

え体系化したのが数理（形式）論理学であり、それをさらに発展させたのが数学基礎論である。現代の科学は数学の表現法を借りている。

そこで、科学における記述も、前提命題、公理、証明法というように数学基礎論のような記述スタイルに整理できるのではないかという考えがある。このような公理論的な記述法は、物理学で部分的に試みられてきたが、あまり成果が上がらなかった。つまり、そのような体系立った厳密な論理展開が実験に基礎を置く現代科学の進歩に必須というわけではなかった。また、一見厳密に思える自然科学の論理体系も、形式論理学で記述が可能なほどの抽象性と厳密さとを持ち合わせていないことも明らかとなった。

よく知られているように、コンピュータ内部の演算はすべて２進数の代数演算で行われている。この演算はブール代数と呼ばれる形式である。ブール代数は、数理論理学でいう古典論理と同等である。古典論理というのは、われわれの日常出会う自然な論理であり、古くから使われていた論理である。ついでにいえば、よく知られた論理としては、この他に直観論理、量子論理、ファジイ（あいまい）論理などがある。

コンピュータのブール代数的な演算機能を利用すれば、古典論理に基づく推論を機械的に実行させることができる。もちろん、そこでは１と０の列からなる入力データと出力データをどういう内容に対応させて解釈するかという作業が残っている。この部分に効率的な対話機能を開発すると、コンピュータに演繹的な推論を実行させることができる。

演繹的推論システムの方法論は凡そ以下のとおりである。

- **人間、特にある領域における経験者の知識をコンピュータで機械的に操作できるような推論規則として表現する。**
- **この推論規則を、あたかもデータのように考えて他のプログラムから独立させ、知識ベースとしてコンピュータに格納しておく。**
- **知識ベースを構築、管理したり、それを用いて演繹的な推論を効果的に展開することに適したプログラミング言語、およびそれを用いたシステム環境。**

一般のソフトウェア・システムでは、人間の書き下す操作手順がプログラムの本体を構成する。ところがエキスパートシステムでは、経験知識の表現だけを他のプログラムからも、データからも独立させて定義して利用する。いずれにしろ、演繹推論等の知識工学はモデルの不確かを補う情報を提供する可能性が期待できる。

１．５．３　システム操作面から見た総合評価の考え方

前述のように、処分システムの安全性評価の判断基準は、処分に起因するリスクが社会的に許容されるリスクレベル以下であるかどうかにおかれるものと考える。そして、この判断のために総合評価が考えられる。総合評価は２種類あると考えられる。

第一に、代表的な（適切な）処分シナリオについて、適切な移行シナリオを選定し決定論的評価を行う手法である。決定論的評価に於ては、シナリオが適切かどうか、評価結果の信頼性が十分かどう

かについて議論がなされるであろう。それは、長期にわたる諸条件の変化やモデル・パラメータの持つ不確かさを考えねばならないからである。そのためには、多方面に渡る情報を評価とともに参照しながら作業をすすめる必要がある。このような決定論的評価を検証するための情報を効率的に提供することは対話システムの重要な機能要件である。

もう一つの方法に確率論的評価がある。それにより、処分システムの安全性の規制的観点からの評価は、決定論的評価手法によって行われ、確率論的評価手法は、決定論的評価を補完するものとして位置付けられると考える。確率論的評価手法は、放出核種の移行過程の変化確率及び移行過程を支配するパラメータの自然的、社会的条件の変化に伴う変動、または情報の不十分さによる不確かさによる最終のリスク評価結果の不確実さを定量的に解析するものである。

更に、評価結果の不確実さに大きく寄与するパラメータを明確に示す機能、いわゆる感度解析機能も持つものであると考えられる。また、確率論的手法は、シナリオを簡単なネットワークで画像表現し、操作を視覚ベースで行うシステム化の手法を提供するものとしても有効と思われる。

例えば、評価モデルとしては、廃棄物からの核種の浸出は一定割合で一定期間続き、緩衝材中の移行は拡散のみに支配され、地層中は一次元吸着平衡を保ちながら移行し、岩盤中の間隙水へ拡散し、飲料水として人間に到達して放射線被爆をもたらすという簡単なシナリオである。確率論的評価手法の開発では、手法そのものの開発とともに、入力データベースの開発も重要な問題である。そして、入力データの確率分布をできるだけ正確に知ることが必要である。

（1）シミュレーションの不確かさについて

本システムを操作して性能評価シミュレーションを行うにあたり、最終的な目標となる生物圏からの放射性核種の隔離を、多重バリアが機能として確保する過程の作業に下記の人的な不確かさが含まれると思われる。それらはシナリオの選択と内容に係わる事項や、数学モデルの数式化と計算等、凡そ以下に示す領域に含まれる不確かさである。

シナリオの選択と内容
処分システムの将来における状態
評価モデルと計算（数学モデルの数式化と実際の計算）の数学モデルの数値解析上の不確かさや、プログラミングの不確かさ、精度計算上の限界
計算パラメータや熱力学データ等のデータ自身に内在する不確かさ
データの不足、測定誤差、専門家の仮定、等

これらは、確率的な変動性による不確かさと知識不足からの不確かさに大きく分類されている。性能評価においては、容器の腐食や長期に渡る放射性核種の地層から生活圏への移行など、将来に亙る条件についての知識不足に依る不確かさが重要な研究課題となる。

また、不確かさを見積る確率論的不確さ評価は主観的な確率を用いて行われていると考えられる。さらに、シミュレーション・プログラムの品質やデータの信頼性等、ソフトウェアに含まれる不確か

さは、地層処分ヴァーチャル・エンジニアリング研究のような情報処理に基盤をおいた性能評価にとって重要な課題となる。従って、ソフトウェアの不確かさは前記の2つの不確かさ以外の第3不確かさと考えられる。

不確かさの評価は、サイトの選定と貯蔵所の設計研究などに用いられ、システム性能評価の結果において、予想の信頼性と精度、及び信頼度を示すことに特徴がある。確率論的不確かさ評価は2種類の情報を同時に与える。それは主観的な信頼性のある期間と不確さのランキングである。前者はサイトや貯蔵所の候補間の比較に要求されるもので、後者は、線量など評価値における不確かさに対して、最も寄与する定量化された不確かさのランキングを示すものである。また、ソフトウェアの不確かさについては、別途複雑度解析等の定量的ソフトウェア品質評価技術の研究が進められている。

シナリオやモデル化における不確かさは、主に定性的な方法において扱われる。例えば、FEP手法が上げられる。それは組織的に良く文書化された過程が要求される。計算モデルに対する定性的な保証過程の展開的な適用が重要であり、性能評価の結果が規制の目的に使用される場合、特に問題となる。パラメータとデータにおける不確かさは次の方法を用いる際に定量的に扱われる。

- 表面応答解析
- モンテカルロ解析
- 微分解析
- 確率論的解析

不確かさ評価に至る前にスクーリング解析を行うことにより、計算時間を最小化できる。不確さ評価結果に含まれる感度情報は、データの不確かさを減らす目的で行う研究のガイドとなる。この手法の採用により、次第に、より複雑な方法を用いる必要性が出てくるにつれ単純性が擁護されている。

例えば統計的サンプリング法がそれぞれ特徴を持って開発されている。

- 単純なランダムサンプリング　：　単純で健全な方法だが、多大な計算時間を要する。
- ラテン超多面体サンプリング　：　単純なランダムサンプリングよりも計算時間は少ないが使用する際に注意を要する。
- インポータンスサンプリング　：　更に計算時間を減少でき、解析をより関心ある領域に向けることができる。

不確かさの評価については、類似分野として原子力プラント、再処理、廃棄物処理の評価における不確かさ評価で用いられた方法に共通点が見られる。すなわち以下のものがある。

・入力データを含むパラメータにおける不確かさに関する方法
・主観的な確率密度関数を用いることによりパラメータの不確かさを表わす方法
・単純なランダムサンプリングないしはラテン超多面体サンプリングを用いて確率密度関数からサンプルしたパラメータセットを出力するモデルの計算法
・メジアン、ミーン及び９５％の信頼値により表わされる不確かさを出力する方法

以上のように不確かさは、性能評価シミュレーションにとって重要な対象である。従って、対話システムは、この不確かさについて出来るだけ多くの情報を提供するように設計開発されなければならない。それは、情報の検索、知識ベース等による判断、補足的な数値計算を含むものである。

（2）ヴァーチャル・エンジニアリング・シミュレーションの議論

シミュレーションは、およそ以下に示す議論について支援することが求められる。それにより、性能評価シミュレーション全体の不確かさをスクリーニングする必要がある。

議論１：決定論的、確率論的、性能評価法の相対的な役割は何であるか？
決定論的な最良の評価とパラメータの不確かさについての確率論的な処理法を結び付けて考察することが必要

議論２：処分システムの性能評価を実行する時、確率論的不確かさと決定論的不確かさを区別して扱う必要があるか？それとも合わせて扱うべきか？

議論３：定量化する必要のある全ての源に含まれる不確かさはどの様な範囲に渡っているのか？
例えばモデルとシナリオに依る不確かさを定量化する方法を探す必要がある。

議論４：不確かさ解析の結果を、どのようにリスク評価や結果の信頼度提示に用いることができるか？

議論５：不確かさ解析に用いる主観的な確率密度関数の定義はどのようになされるべきか？
例えば直接測定が不可能な時、専門家の判断を用いる場合に問題となる。

議論６：時間に依存するプロセスは、どの範囲に対して性能評価に取り入れるべきか？

議論７：異なるパラメータ間の相関をどのように見積ることができるか？

議論８：リスクと線量率を評価する時、不確かさパラメータを扱う確率論的手法は規制側による決定を助成することに対してのみ適用ざれるのか？それは、処分に関する安全性のレベルを大衆に知らせるために用いるには、あまりにも複雑ではないか？

以上の議論から、地層処分ヴァーチャル・エンジニアリング・システムの開発においては、その対話システムの支援機能として、以下に示す要求にそった情報サービスを行える機能を開発する必要がある。

要求１：定量的な手法と非定量的手法を結び付けるべきである。

要求２：不確かさ解析の種類と型式は規制側の要求により決定されるべきである。

要求３：決定論的な計算と、確率論的不確かさ解析を結び付けることにより全体的なシステム性能評価を導く一般的なガイドラインとすべきである。

要求４：感度解析は、不確さ解析に対し密度な関係がある。

要求５：異なる不確さパラメータ間の相関を扱う方法の開発が必要である。

要求６：幾つかの二者択一的な統計的サンプリング法がメリットを持っており利用可能である。

要求７：主観的な確率密度関数の形において、パラメータの不確かさの表現がシステム的なアプローチに必要となり、特に主観的な専門家の意見が取り入れられる。

要求８：コードの質的な評価がモデルの不確さを減らすのに役立つ。

要求９：ベンチマークによる検証と確証試験が、不確かさを減少させることの助けとなり、モデルに対するユーザーの信頼度を向上させる。

要求１０：システム的なアプローチは不確かさ解析を指導することに適用されるべきである。

１．６　ヴァーチャル・エンジニアリングの目指す科学技術

地層処分ヴァーチャル・エンジニアリングは、従来からの実験に基礎をおいた理論を構築する科学の手法ではアプローチの難しい（方法自身に限界がある）問題に対する、第３のアプローチとして今後研究開発する科学技術の領域である。そのアプローチの考え方の中心は、複雑系やカオスと呼ばれる数学の分野である。また、その分野はコンピュータによる仮想世界のシミュレーション技術と密接に関係した世界である。それは、コンピュータの中に構築された世界で生じる現象を観察することにより、新たな定式化を期待しているためである。

ここでは、同分野の先端的な研究所として知られている、サンタ・フェ・リサーチのジョン・キャスティ氏の文章を引用し、今後の地層処分ヴァーチャル・エンジニアリング・システム開発の参考にする。

（１）何が科学の限界をもたらすか？

自然や社会についての疑問を解決するに当たって、人類の能力や知験には限界がないと思っている人たちにとっては、２０世紀の科学を振り返ることは、きわめて憂うつな経験となるだろう。今世紀に入って、科学における最高の業績の多くは、何ができず、何がわからないかを私たちに教えてくれた。このような限界を定めた最も有名な定理は、たぶん**ゲーテルの不完全性定理**だろう。この定理によれば、どんな演繹推論体系も、その体系内の言語を使って表現される算術に関するすべての疑問には答えられない。要するに、十分に強力で無予盾な論理体系は、どれも不完全なのである。

ゲーテルの定理から数年後に、アラン・チューリングは、コンピュータのプログラムについてゲーテルの不完全性定理と等価な定理を証問した。それによれば、入力データを処理しているときに、プログラムが定まるかどうかを判定する体系的な方法は存在しない。最近になってグレゴリー・チェイティンが、情報理論の見地から、ゲーテルによる証明可能性の概念を調べた。その結果、どんな論理体系においてもその真偽が定まらない単純な算術命題の例を見つけた。このチェイティンの結果によると、そのような数学の問題は私たちにとって複雑すぎるだけなのだという。

ゲーテル、チューリングおよびチェイティンの定理は、**数学の世界を理解する人間の能力に限界があることを示している**が、同様な限界が、有名なハイゼンベルクの不確定性原理にも当てはまる。一見したところ、この原理は物理量の測定において私たちの能力に限界があることを物語っているようにも思える。しかし、もう少し注意深く調べると、ハイゼンベルクの不確定性とは、実は量子論を構成する数学上の形式化がもたらした限界であって、それは現実世界の構造自体に固有の限界なのかもしれないし、そうではないのかもしれない。

同じような見方が、ほかの理論で見られる限界にも当てはまる。たとえば、社会選択理論におけるアローのパラドックスや分子生物学におけるセントラル・ドグマ（DNA情報がRNAに読み取られ、タンパク質が合成されること）である。両者ともに、現実世界での認識や行動に関する限界というよりも、むしろモデルがもたらした限界である。そこで、次のようなことが知りたくならないだろうか。自然現象や社会現象に関する問題で、科学の無力さゆえに、その答えを永遠に見つけられないものが存在するのだろうか？

（2）科学知識とは何か？

この問題に直面して、最も頭を悩ませる作業は、科学知識とは何を意味するのかをはっきりさせることだろう。哲学者たちは何年もかけて、何が知識を構成しているかという問題に取り組んできた。しかし、この問の終点はまだ私たちの視野に入ってきていない。そこで多少は議論の余地はあると思うが、この哲学上の難問を一刀両断にするために、**科学知識は一組の規則または一本のプログラムの形を取る**という立場を採用しよう。

すなわち、問題を一組の規則に放り込むと、論理的な演繹が行われて答えが出力されると考えるのである。もちろん、どんな規則でも良いというわけではない。規則が科学的であるためには、単純性、明瞭性、脱偏見性、扱いやすさなど、いくつかのテストをパスしなければならない。いったんそうしたフィルターを通過すれば、あと必要な属件といえば、科学理論に通常備わっているアルゴリズムの概念くらいである。

科学理論は問題に答えるために使われる。私たちが使用する規則は、たぶん３つの異なる世界（現実の世界、数学の世界、計算の世界）のどこかに存在するだろう。以下で見るように、何か特定の問題に答えるために一組の視則を適用する場合、この３つの世界の区別に留意することが重要である。

科学知識がコンピュータ・プログラムと同様に扱えると考えると、計算の困難という問題が生じる。たとえば、有名な巡回セールスマン問題という難問があるが、その計算量は問題のサイズとともに指数関数的に増大すると信じられている。180程ある世界各国の首都を最小コストで巡る旅を調べるために力づくで計算を行うとしよう。すると、可能なルートは180!（＝180x179x178xn.....x2x1）通りあるが、その各々を計算するには最高速のコンピュータを使っても、宇宙年齢をはるかに超える時間を要するだろう（だがそうした計算は原理的には可能である）。

このような問題は、私たちの興味をかきたてはするが、ここでは主要な関心事ではない。私たちが注目するのは、**答えを出すためのプログラムが全く存在しない問題**である。すなわち、原理的に答えられない問題に私たちは興味を抱いているのだ。それでは、自然や人間の世界において、科学に潜む論理的な障壁を知るうえで役立ついくつかの問題に目を向けてみよう。

（3）3つの問題

科学知識の限界をより明確にするために、物理学と生物学と経済学の各分野から、よく知られた3つの問題を調べることにする。それらは、大陽系の安定性に関する問題と、タンパク質の畳み込み問題、市場の効率問題である。

太陽系の安定性

この問題では、ニュートンの法則に従って動くN個の質点が対象となる。そして、次のようなことが起こる質点の初期条件（位置と速度）が存在するかどうかを探るのである。

> **将来の有限時間内に、2つ以上の物体が衝突する。**
> **将来の有限時間内に、1つ以上の物体があらかじめ定められた以上のエネルギーを得て、最後には系の外に飛び出してしまう（脱出速度に達する）。**

N＝10の場合、これは“大陽系は安定しているか？”という問題を数学的に定式化したものとなる。ノースウェスタン大学のジェフ・サイアは、こうしたシステムの安定性に関する一般的な問題に1つの解答を与えた。そのとき彼は五体のシステムを構成したが、そのうちの一つの物体が、有限時間内に非常に大きなエネルギーを獲得することを見出した。だがこれは、現実の世界の問題を数学的に解釈して解いたものにすぎない。すなわち、五体のシステムが実際にどうなるか誰にもわからないのである。

タンパク質の畳み込み

あらゆる生命を作り上げているタンパク質は、数多くのアミノ酸が連なって形成されている。そこでは、アミノ酸がネックレスのビーズのように長くつながっているが、どのビーズがどこに位置するかという命令は細胞のDNAに含まれている。しかし、ビーズが正しい順序に並ぶと、そのタンパク質は、生体中で特定の機能を果たす独特な三次元構造に折り畳まれる。それは、ちょうど104個のアミノ酸からなる非常に短いタンパク質だった。タンパク質の折り畳みで最大の難題は次のようなものだった。

一列に並んだ特定のアミノ酸は、自分自身をどんな三次元の配列へ折り畳むのだろうか？タンパク質が折り込まれるときに、最低の工ネルギー状態になると一般には考えられている。そして、実際に何千種類ものアミノ酸から組み立てられたタンパク質が、ほんの一秒かそこらで最終的な構造へと折り畳まれて行くのが観察される。

だが、このプロセスをコンピュータでシミュレートしようとすると、ほんの100個ほどのアミノ酸からなる非常に短いタンパク質が、最終的に折り畳まれる形を見出すのにさえ、スーパーコンピュータを使っても10の127乗年かかると推定されている。

事実、アビエズリ・フランケルが1993年に示したように、タンパク質の折り畳み問題を数学的に形式化したものはNP完全な問題である。このことが意味するのは、巡回セールスマン問題と同じように、折畳み問題も計算が困難だということである。それでは、自然はいったいどうやっているのか？これは、タンパク質の折り畳みに関する計算面での疑問である。次に、人間世界の出来事から解答できない問題の候補を一つ取り上げる。

市場の効率

アカデミックな金融理論が依存している大黒柱の一つに、金融市場が効率的だという考え方がある。簡単に述べると、この考えは、株や商品の価格に影響するどんな情報もすぐに市場で処理され、証券などの現在価格に反映されるということを意味する。インフレの影響を無視すれば、市場の効率によって価格は全くランダムな形で動くというのである。すると、過去の株価チャートのような、街で手に入る情報に基づくどんな売買手法も役に立たないということになる。

すなわち、長期間にわたり平均して市場よりパフォーマンスがよい売買手法は存在しえない。理論にしても同じことである。現実の市場は、アカデミックな金融理論にあまり従っていないように思える。そして、雑誌などは次のような例外で埋め尽くされている。たとえば小企業の利点や株価収益率の効果がある。

前者によると、小企業の株の方が大企業の株よりもパフォーマンスがよい。また後者によれば、株価収益率の低い株の方が高い株よりもパフォーマンスがよくなる。この２つの例外は、市場の効率という考え方にかなりの疑問を投げかけている。

（4）現実の世界と数学の世界

私たちは、自然界と人間社会について３つの大きな問題を検討した。太陽系の安定性、タンパク質の折り畳み、そして市場の効率である。それぞれに対する答えは次のようになろう。**おそらく大陽系は安定しておらず、タンパク質の折り畳みを計算するのは困難で、金融市場は全く効率的ではなさそうだ。**だが、このような答えに共通するのは、現実の世界の問題を数式で表現している点である。

つまり、現実の世界の問題自体に対する答えではないのである。たとえば、N体問題に関するサイアの解答は、現実の世界の重力に従って動く惑星については何も語っていない。フランケルの結論も同様だ。タンパク質の折り畳みの計算が困難だということは、どのようにしてタンパク質が数秒で正確な配列を計算してのけるのかという問題には触れていない。また、ウォール街で働く賢いトレーダーたちは、何十年もの間、効率的な市場という仮説を無視してきた。

以上の例から次のことがわかる。もし、現実の世界で科学的に解答できない問題を探したいならば、自然界および人間社会の現象と、そうした世界の数理モデルとを注意深く区別しなければならない。従来の科学研究では、通常、こうした世界をいっしょくたにしても特に害はなかった。しかし、ここで、もし貴重なものと無用なものとを一緒に捨てるという愚を避けたいのなら、現実の世界と数学の世界を区別するのは非常に重要である。

どの瞬間においても、次の点だけは明確にしておかなければならない。現実の世界を数式で表現して、その特性について数学上の主張をしているのか、それとも、現実の世界で暗示される因果関係について現実の世界での主張を行っているのか。この２つは明らかに異なる。したがって、この点をもう少し調べることは有益だろう。

（5）３つの世界を一つに

私たちが関心を抱いているのは、現実の世界における解答不能な問題である。ある問題が論理的に解答不能であることを証明する手段は、数学の世界においてのみ手に入る。そのうえ、科学知識とコンピュータ・プログラムは同等であると考えたので、数学の世界も解答不能の決定に関与してくる。

こうした世界を順に調べてみよう。

現実の世界は、時間とか位置といった直接観測できる量と、それから導かれるエネルギーなどの量からなる。こうした量を使って、私たちは惑星の位置やタンパク質の配列などを観測する。このような量は、一般に離散集合を構成し、その測定値は有限集合から選ばれる。その上、こうした測定は一般に不確実である。

一方、数学の世界では、現実の世界で観測されるものを記号で表現する。そこでは、空間も時間も連続体であると仮定される。さらに、位置や速度などを表す数学記号は、たいてい、整数とか実数や複素数などの値を取る。すなわち、どの集合にも無数の要素があることになる。さらに、数学には確率論があって、それは偶然に影響されるランダムな事象を扱う。

最後に、計算の世界がある。これは、片足を物理的な装置という現実の世界に置き、もう一方の足を抽象的な数学の世界に置くという奇妙な位置を占めている。計算を一組の規則（アルゴリズム）の実行と考えるならば、計算は、記号と記号の関係を扱うことになり、純粋に数学的な対象となる。だが、計算をコンピュータのメモリー・スイッチをオン・オフに変えるプロセスと見なすならば、それは物理的に観察できる世界にしっかり根付いていることになる。

科学知識の限界について有効な理論を作れる可能性が少しでもあるならば、これらの関係（現実の世界、数学の世界、計算の世界）を最優先に留意しなければならない。この３つの世界が何か特定の問題（たとえば大陽系の安定性）を表現する限り、なんとかしてその３つの世界を一致させる必要がある。そこで、このような一致が実際にどのように行われるのか２、３の研究法を考えてみよう。

（６）１つあるいは２つ？

考察の対象をもう一度確認しておこう。科学的な手段での解決が不可能な現実の世界の問題に私たちは関心を抱いている。そのような問題の存在（または非存在）を立証するためには、２つの選択肢がある。すなわち、すべての議論を自然界にのみに制限するか、自然界と数学および計算の世界をかみ合わせるかである。

第一の方法を選び、もっぱら自然界の範囲内にとどまることにしよう。すなわち“大陽系は安定しているか？”といった問題を数学的な表現に変えることを禁じるのである。また、答えを出すために、論理的な証明メカニズムを使うことも同じく禁止してしまう。ここで難しいのは、科学的に答えを出せない（あるいは出せる）ことの証明が私たちの目標となっている点である。しかし、証明の概念そのものは数学の世界にしか存在しない。

したがって、私たちは現実の世界において、証明に代わる概念を見つける問題に直面することになる。自然界で証明に代わる第一の候補は、因果関係の概念である。これを採用してみよう。すると次のような主張が導かれる。もし、因果関係の連鎖から得られた最終結果が問題の答えとなるならば、その問題は科学的に解答可能となる。たとえば、大陽系の安定性の問題は、惑星の位置、速度、質量で始まる因果関係の連鎖によって解決されて、これらの因果関係の連鎖により惑星の軌道が求められてゆくのである。こうして次々と因果関係をたどることにより、現在観察している精度では惑星の衝突や離脱は有りえないことが示されるかもしれない。

しかしながら、複雑な状況下で確実な因果関係の連鎖を作り上げることは、たぶん大変な仕事になるだろう。特に人間が関与している状況下では、なおさらである。そこで、次に自然界と数学の世界

を交わらせる方法を考えてみる。

（7）ゲーテルを越えられるか？

現実の世界における特定の問題を解決するために、数学の証明という手法を使いたい。この場合、まず最初に問題を何らかの数学的な表現に書き換える（コード化する）必要がある。たとえば、微分方程式とか有向グラフなどである。それから、数学の世界に特有な方法やテクニックを駆使して、問題を数学的に解決するのである。そして、最終的に得られた解答（もしあるなら）を現実の世界の言葉に解読すればよい。その全体的なプロセスの概要を図１．６－１に示した。このコード化と解読のプロセスは、私たちが関心を持つ現実の世界の現象と、それを数学の世界で表現したものとの間でモデリングの関係をなしている。ここで大切なのは、次の主張に説得力を与えることである。すなわち、“問題を数学的に書き換えても、それが現実の世界で起きていることを現実に表現している”ということだ。これは、モデルの妥当性という昔からの問題を言い換えたものにすぎない。

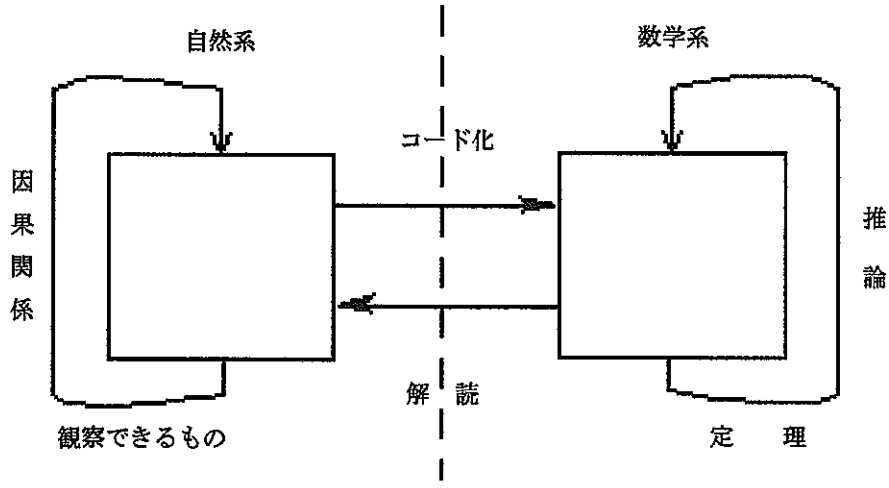

図１．６－１　コード化と解読の関係

すなわち、自然についての数学モデルと、自然自身とが互いに何らかの関係を持つことを私たちはどうやって知るのか？これはメタ科学的な問題で、その解決にはモデル理論を展開する必要がある。しかし別の方法もある。決定不能に関する数学上の結果は、すべて無限集合（整数、実数、複素数）に依存している。

しかし少し前にも述べたように、観察される現実の世界では、測定結果は有限集合からなる値を取る。このような有限集合に対して、論理体系におけるゲーテル型の決定不能な結果は存在しない。同様にして、演繹的ではない方法｛たとえば、帰納法や仮説形成（何らかの現象に出くわしたとき、その現象を説明する理論（仮説）を考え出すときの発想法）｝は、ゲーテル流の決定不能を超えた世界に私たちを導く。もし演繹以外の方法を使う体系や有限集合に数学の形式化を限定すれば、あらゆる数学の問題は決定可能となり、その問題に答えることができる。

したがって、そうした数学の問題を解読した現実の世界の問題も同じく解答可能と予想される。解答可能性を研究する一つの方法は、こうした“非ゲーテル”の形式化を使って展開される。これに関連することだが、次のようなことも考えられるかもしれない。人間の頭脳は、チューリング型コンピ

ュータ（ただし非常に洗練された形で）であることにより、その想像能力は制約を受けるのか？もちろん、これは“強力なAI”に関する中心問題である。

すなわち、機械は私たちと全く同じように思考できるのか、最近、ストックホルムにある未来研究所の支援により、私と心理学者のマーガレット・ボーデン、数学者のドナルド・サーリ、経済学者のエイク・アンダーソンらが、この問題に取り組んだ。この研究によれば、自然科学や数学と同様、創造的を芸術においても、人間の創造能力はチューリング型コンピュータの厳密な制約をほとんど受けていないようである。もちろん、チューリング以外にも他の計算モデルがあり、人間の知性は何らかのタイプの超チューリング・マシンかもしれない。

たとえば、DNAコンピュータがそれである（現在のところ、これは研究中である）。もしも人間の知性が、その認識活動において規則の支配を超越できるならば、人間の知験は、決定不能性というゲーテル流の限界には影響されないことになるだろう。

（8）ジョン・キャスティ氏の主張

この非ゲーテル流の特徴を踏まえたうえで、科学知識の限界について私個人の考え方で締めくくらせてほしい。すなわち、数学の世界における決定不能性と同様なことを、現実の世界で示すには何が必要とされているのだろうか？

私が思うに、科学の問題で解答不能に陥るのは、自然が矛盾しているか、または不完全であるかのどちらかの場合であろう。ところで、こうした数学上の概念（無矛看および不完全）は、物理現象の世界では何に相当しているのか？無予盾とは、自然界には正真正銘のパラドックスが存在しないということである。たしかに、量子力学は奇妙な世界観を提供したが、私の見解では、粒子が左と右へ同時に動いたり、水が上と下へ同時に流れたりすることはない。

そのようなパラドックスと思えるものに、いつも私たちは遭遇してきた。たとえばクェーサーの赤方偏移や、初期宇宙の膨張率が小さくなることなどである。だが、そのような場合には、いつもそれに引き続いて現れた研究や理論が疑問を解決してきた。

そこで、自然界の無予盾性は公理であると私は確信している。自然の完全性とは、何か物理的な状態が理由もなく起こらないことを意味している。要するに、いかなる結果に対しても必ず原因があるのだ。もう一度繰り返すが、この主張に対する完璧な反例を私は思いつかない。そこで、自然は無矛盾であるだけでなく、完全でもあるということを私は作業仮説と見なしたい。

（9）まとめ

地層処分ヴァーチャル・エンジニアリング・システムの開発では、ソフトウェア複雑化の要因を把握し管理することは重要である。一般にアプリケーション・システムは明確な開発目標を持っている。例えば、動的な原子炉冷却系の熱・流動特性のシミュレーション・システム、再処理工程の確率論的安全性評価等である。

地層処分ヴァーチャル・エンジニアリング・システムは、従来シミュレーションや数値計算などと呼ばれてきた物理工学上の現象を、コンピュータにより数値的に計算模擬する技術をもとに、オブジェクト指向、並列計算法、認知科学等、革新するコンピュータ・サイエンスの成果を地層処分の安全性評価研究に応用し、処分の安全性に係わる技術論拠となる情報を多方面から総合的に評価すること

を目的として、専用に統合開発するコンピュータ・システムである。

平成８年度に、地層処分ヴァーチャル・エンジニアリング・システムの開発課題について基盤となる計算科学技術関連技術を研究・整理し、地層処分ヴァーチャル・エンジニアリング・システム開発基盤を確立する研究を実施した。研究にあたり、試作評価の必要なハードウェア・ソフトウェア開発技術については個別に試作を行い、それらを統合化して、プロトタイプ地層処分ヴァーチャル・エンジニアリング・システムとして実証評価を行った。

地層処分ヴァーチャル・エンジニアリング・システム開発プロジェクトの目標は、高レベル放射性廃棄物地層処分の安全性評価にかかわる、実験の難しい長期（数千から数十万年）に亘る現象を考慮した各種のシナリオを複雑に組み合わせて、システムの長期的挙動の評価を行ない、地層処分の安全性に係わる技術を判断し、処分システムの設計の最適化をするための指針等を得るとともに、多分野にまたがる専門家が、自己の領域と他の領域との関係を容易に理解するための基盤を提供することである。

また、今後の計算科学技術開発の開発課題である、複数の計算を組み合わせて試行錯誤的にシミュレーションするシステム（**モデル協調計算**と呼ぶこととした。これは複雑系モデルである）の基盤技術を実証することである。

地層処分システムの性能評価は、地震等の確率的に生じる事象による変化を除いて、地層中の環境が安定している場合、地層処分された廃棄体から放射性核種が放出され、地下水により生物圏まで輸送されるシナリオを基本としている。そのような安定した地層中での環境に添って核種移行を考えた場合、それだけでも非常に複雑な環境評価モデル開発と評価の問題であることが解かる。

領域１：ガラス固化体（崩壊熱モデル、放射性核種放出モデル、等）
領域２：オーバーパック（腐食モデル、腐食物ベントナイト化合モデル、等）
領域３：ベントナイト（２層多孔質媒体地下水流れ、熱・オーバーパック腐食化合モデル、等）
領域４：人工バリア周辺（人工バリアと岩盤との熱応力モデル、コロイド、核種収着、沈殿、等）
領域５：地層中（広域的地下水汚染物質移行モデル、亀裂水理、等）

実現象は、上記のモデルが複雑に組み合わさったものと考えられる。評価シナリオは、例えば温度場の変化が力学系に作用し、地下水流動を変更することにより人口バリアの冠水や亀裂を生じ、結果として核種が滲出して輸送されるシナリオが考えられる。このような、シナリオにそったシミュレーションを対話方式で試行錯誤的に行なうには、下記の操作機能がアプリケーション・システムに要求される。

操作機能１：シミュレーション・シナリオの一覧
　　シナリオは、充分に評価され妥当なものとしてデータベース管理されていること。
操作機能２：シミュレーション条件の決定
　　シナリオの選択、個別評価モデルの選択、計算条件の設定等
操作機能３：シミュレーションの実行と制御

シミュレーションの実行、実時間視覚化、計算結果の解析等

対話型実時間可視化シミュレーション機能は、今後の性能評価シミュレーションの技術研究に於て重要な課題である“個別の計算モデルを複雑に組み合わせ協調して計算させる”技術の研究環境である。

例えば、MIT X/11に代表される大規模で複雑なシステムプログラムの多くは、オブジェクト指向を開発戦略に取り入れている。それは、オブジェクト指向がソフトウェアの大規模化からくる複雑度の増大に伴う信頼性低下や、複数の計算モデルを組み合わせて実行すると言った要求に対する現在考えられる具体的な手法であるためと考えられる。従って、大規模で複雑な統合化シミュレーション・システムを開発するためには、オブジェクト指向の考え方を物理工学シミュレーションで取られてきた要素還元論と合わせて整理することにより、計算科学の領域である大規模複雑系シミュレーションの技術開発戦略として期待できる。

オブジェクト指向は、プログラム開発に関する新たな方法論である。この方法論によれば、メモリー上に展開される従来のプロセス、タスクといった従来の実行状態の手続きはオブジェクトと言う、より発展した概念に統合して考える。このオブジェクトの考え方に添って、物理・工学上の問題等をプログラミングすることにより、より複雑な問題を効率良くプログラムすることが期待できる。

また、並列計算システムを利用することにより、より説得力のあるシミュレーションが期待できる。オブジェクト指向による、シミュレーション・プログラム開発には新しい考え方が必要であり、その影響はモデル開発にまで及ぶものと思われる。それは、シミュレーションの考え方を革新する思考方法と捉えることもできるであろう。

地層処分ヴァーチャル・エンジニアリング・システムの開発プロジェクトでは、複雑なモデルを試行錯誤的に組み合わせてシミュレーションを実行するシステム開発技術の基礎的な部分の試作と評価を行なった。その結果、以下の新しい概念とシステム開発技術の成果が得られ、これからの計算科学技術の新しい成果が期待できる。

従来研究開発されてきた科学技術のトップダウン・アプローチ（現象全体を確率論的モデルで記述したもの等）とボトムアップ・アプローチ（現象全体を個別の要素に還元して個別の決定論的モデルで記述してゆく手法）とは別な、第３の手法としてモデル協調計算という手法を考え導入した。

モデル協調計算

コンピュータ内に組み立てられた仮想世界の中で、複数の数値計算モデルを協調させて並列的に実行することにより、従来のトップダウン・アプローチで押さえきれなかった、より精密な現象過程や、ボトムアップ・アプローチで記述が難しかった、ある程度の自由度をもった現象のシミュレーションを計算で実現する方法とした。この、モデル協調計算は、理論、実

験といった従来の科学技術の方法とは別な、第３の科学技術の方法である
計算科学技術の位置づけを明確にする科学思想とした。

モデル協調計算は技術的に可能である

モデル協調計算を実現するための基礎技術は、オブジェクト指向によるシミュレーション、協調計算結果を実時間で３次元可視化する技術、異機種の複数計算機に機能分散を行ない、１システムとして統合する技術を確立するとこにより基本的に可能であると考えられる。

システム開発には個別の専門的知識より広範な知識が必要と思われた

システムの設計開発にあたり、現象評価に係わるシナリオ、個別現象過程、そして、ソフトウェア・エンジニアリングについて広範な知識が必要であると判断された。個別の専門領域に限定してきた従来の技術者教育の今後の課題として考えられる。

本章の参考文献

- PNCTN1410 91-009、地層処分研究開発の現状（平成元年度）、動力炉・核燃料開発事業団
- PNCTN1410 92-081、高レベル放射性廃棄物地層処分研究開発の技術報告書、平成３年度、平成４年９月、動力炉・核燃料開発事業団
- PNCTN1340 93-001、動燃技報、高レベル放射性廃棄物の地層処分研究開発特集、１９９３年３月、動力炉・核燃料開発事業団
- PNCTN1410 94-094、地層処分研究開発の現状（平成５年度）、１９９４年１１月、動力炉・核燃料開発事業団
- PNCTN1410 96-071、地層処分研究開発の現状（平成８年度）、１９９６年１２月、動力炉・核燃料開発事業団
- JAERI - research 96-011、高レベル放射性廃棄物地層処分の安全性研究、１９９６年３月、日本原子力研究所
- デービス、P. 他、『数学的経験』、柴垣和三雄・清水邦夫他訳、１９８７年、森北出版
- アイゲン、M. 他、『自然と遊戯（ゲーム）』、寺本英・伊勢典夫他訳、１９８１、年東京化学同人
- グリック、『カオス』、止田院亮監修、大貫昌子訳、１９９１、年新潮文庫
- ピーターソン、I. 、『現代数学ワンダーランド』、奥田晃訳、１９９０年、新曜社
- パイトゲン、H、リヒター、P. 、『フラクタルの美』、宇敷重広訳、１９９１年、シュプリンガー・フェアラーク東京
- ポーストル、D. 、『宇宙の網目』、はやしはじめ訳、１９９３年、白揚社
- ラッカー、R. 『無限と心』、好田順治訳、１９８６年、現代数学社
- スチュアート、I. 、『カオス的世界像』、須田不二夫・三村和男訳、１９９２年、白揚社
- パーシグ、R. 、『禅とオートバイ修理技術』、五十嵐美克訳、１９９０年、メルクマール
- ポーデン、G. ハリス、K. 、『ことばの科学入門』、廣瀬肇訳、１９８４年、メディカルリサーチセンタ
- オッペンハイム、A. シェイファー、R. 、『ディジタル信号処理（上・下）』、伊達玄訳、１９７８年、コロナ社
- ガモフ、G. 、『1、2、3一無限大』、（G・ガモフコレクション、『宇宙＝1、2、3一無限大』所収）、崎川範行訳、１９９２年、白揚社
- ニューマン、W. 他、『対話型コンピュータグラフィックス（I、U）』、大須賀節雄監訳、１９８４年、マグロウヒル出版
- ウィルソン、I. 、『トリノの聖骸布』、木原武一訳、１９８５年、文芸春秋
- プリッグス、J. ピート、D. 、『鏡の伝説』、高安秀樹他訳、１９９１年、ダイヤモンド社
- フェダー、J. 、『フラクタル』、松下貢・早川美徳他訳、１９９１年、啓学出版
- マンデルブロ、B. 、『フラクタル幾何学』、広中平祐監訳、１９８５年、日経サイエンス社
- パイトゲン、H、ザウペ、D. 、『フラクタル・イメージ』、山口昌哉監訳、１９９０年、シュプリンガーフェアラーク東京
- エイブラハム、H. ショー、D. 、『ダイナミクス力学系（工・下）』、東保光彦訳、１９９０年、現代数学社
- ベルジェ、P. 他、『カオスの中の秩序』、相澤洋二訳、１９９２年、産業図書
- ホフスタッター、D. 、『メタマジック・ゲーム』、竹内郁雄他訳、１９９０年、白揚社
- シュベンク、T. 、『カオスの自然学』、赤井敏夫訳、１９８６年、工作舎
- ガードナー、M. 、『数学ゲーム（I、I）』、高木茂男訳、１９７４年、講談社ブルーバックス
- ホフスタッター、D. 、『ゲーデル、エッシャー、バッハ』、野崎昭弘他訳、１９８５年、白揚社、河日津一郎、『Morphogenesis』、１９８５年、JCC出版局
- S. J. Brocoum, et. al., "International perspecive on U.S. high-level waste licensing requirements"

- V. Braeuer, et. al., "Analysis of criteria for high level waste disposal"
- M. Jensen et. al., "Protection of man and nature - Some criteria acceptance of nuclear waste repositories"
- S. Norrby "Radioactive waste management • An international view"
- NEA, IAEA, CEC "Disposal of radioactive waste: Can long-term safety be evaluated PA international collective opinion" (1991)
- NEA, Safety series No.111-G-4.1 Siting of Geological disposal facilities (1994)
- Andersson, J. and Eng, T. (1989) The joint SKI/SKB Scenario development project. In Proc. NEA/IAEA/CEC Symposium, Paris 1989.
- Andersson, J. et. al., (1989) The joint SKI/SKB Scenario development project. SKI reports TR 89:14.Billington, D.E. et. al., (1989) Radiological assessment of deep geological disposal: work for UK Nirex Ltd. In Proc. NEA/IAEA/CEC Symposium, Paris 1989.
- Bingham, F.W. and Barr, G.E. (1979) Scenarios for long-term release of radionuclides from a nuclear waste repository in Los Medanos region of New Mexico. Sandia National Laboratories report SAND78-1730.
- Bonano, E.J., et. al., (1989) Demonstration of a Performance Assessment methodology in High-level waste disposal in basalt formations. NUREG/CR.-47 SAND86-2325.
- Bonano, E.J., et. al., (1990) The use of expert judgments in performance assessment of High-level waste repositories. In Safety Assessment of Radioactive Waste Repositories. Proceedings of the Paris Symposium, OECD, Paris, 1990.
- Bonano, E.J., and Davis, P. (1989) Performance assessment methodologies for the analysis of high-level nuclear waste repositories. Rad Waste Man and Nucl Fuel Cell Cycle, Harwood Acad. Publ. 13 (1-4) 229-239
- Boulton, G.S. Time-dependent modelling of environmental change: The prediction of Quaternary glaciations. In Safety assessment of radioactive waste repositories. Proceedings, OECD, Paris, 1990.
- CEC: Performance assessment of geological isolation systems for radioactive waste - Summary. CEC Nuclear and Science report series PAGIS, 11775EN (1998).
- Cranwell. R.M., Campbell.; J.E. Helton., J.C., Inman., R.L., Longsine., D.E., Ortiz., N.R. Runkle., G.E. and Shortencarrier., M.J. (1987) "Risk methodology for Geological disposal of Radioactive Waste:" Final Report, SAND81-2537, NUREG/CR.-2452, Sandia National Laboratories, USA
- Cranwell, R. M. et. al., (1987) Risk methodology for geologic disposal of radioactive waste: final report. NUREG/CR.-2452, SAND81-2573.
- Cranwell, R. M. et. al., (1982, revised 1990) Risk methodology for geologic disposal of radioactive waste: scenario selection procedure. NUREG/CR.-1667, SAND80-1429.
- D'Alessandro, M. and Bonne, A., Radioactive waste disposal into a plastic clay formation. CEC report EUR7111 (Also by Harwood Academic Publishers, 1981)
- (1986) First report on the acquisition of data for use in Probabilistic risk assessment of underground disposal of radioactive waste. UK DoE report DoE/RW/86.071.
- Dames and Moore Intl, (1977) Evaluation of environmental change and its effects on the radiological performance of a shallow engineered disposal facility at Elstow, Bedfordshire. UK DoE report: TR-D&M-9; DoE/RW/87.124. UK Department of the Environment, London, Wiley, London.
- Dames and Moore (1988) Earthquake effects on groundwater systems: an Introductory Review. UK DoE report DoE/RW/89.052.
- Dames and Moore (1988) Background studies: climatic and geomorphological aspects of the evolution of shallow-land buried sites for radioactive waste disposal. UK DoE report DoE/RW/88.055.
- Escalier des Orres, P., Devillers, C., Cernes, A. and Izabel,C. Determination des scenarios a prendre encompte dans l□appreciation de la sureted□un site pour le stockage des dechets radioactifs en formation geologique profonde. In Proc. NEA/IAEA/CEC Symposium, Paris 1989.
- Environmental Protection Agency (1985) Environmental standards for the Management and disposal of spent nuclear fuel, high level and transuranic radioactive wastes, Final Rule. 40CFR Part 191, Federal Register 50 (182) Washington, D.C.
- Foley, M.G. et. al., (1982) Geologic simulation for a hypothetical site in the Columbia Plateau: Results. Pacific Northwest Laboratory report PNL-3542-2. Check versus other ref which may be earlier version.
- Eisenberg, N. (1992) Performance assessment for High-level waste repositories in risk assessment: A survey of

characteristics, applications and methods used by Federal Agencies for Engineered systems. Nuclear Regulatory Commission, Washington D.C. 20555-0001.

- Grimwood, P. and Thegerstrom, C. (1990) Assessment of the risk associated human intrusion at radioactive waste disposal sites. Some observations from NEA workshop. In Safety assessment of radioactive waste repositories. Proceedings of the Paris Symposium. OECD, Paris 1990.
- Guzowski, R. V., et. al., (1989) Potential scenarios for use in performance assessment high-level radioactive waste repositories in unsaturated tuff. NUREG/CR.-477 SAND86-7170.
- Hunter, R.L. (1983) Preliminary scenarios for the release of radioactive waste from a hypothetical repository in basalt of the Columbia Plateau. NUREG/CR-3353
- Hunter, R.L. (1983) Preliminary scenarios for the release of radioactive waste from a hypothetical repository in basalt of the Columbia Plateau. Sandia report SAND83-1352
- Hunter,R.L., Barr,G.E. and Bingham, F.W. (1983) Scenarios for consequence assessments of radioactive waste repositories at Yucca Mountain, Nevada test site. Sandia National Laboratories report SAND82-1277.
- Imbrie, J. et. al., (1984) The orbital theory of Pleistocene climate. In Milankovitch and climate, ed Berger, A.L., Reidel, Dortrecht, 269-305.
- INTERA Environmental Consultants Inc., (1983) GSM: Geologic simulation model for a hypothetical site in the Columbia Plateau: Large computer version of nuclear waste isolation report ONWI-447
- INTERA Environmental Consultants Inc., (1983) FFSM Far-field state model, Office of Nuclear waste Isolation report ONWI-436.
- International Atomic Energy Agency (1981) Safety assessment for the underground disposal of radioactive wastes. Safety series No, 56, IAEA, Vienna.
- Kane, P. (1992) VANDAL version 1.3 Technical overview. UK DoE report. DoE/RR/92.095. UK Department of the Environment, London.
- Laurens, J-M., Thompson, B.G.J. and Sumerling, T.J. (1989) The development and application of an integrated radiological risk assessment procedure using Time-dependent probabilistic risk analysis. In Proc NEA/IAEA/CEC Symposium, Paris.
- Logan, S.E. and Berbeno, M.C. (1977) "Geologic modelling in risk assessment radioactive waste management" in Risk analysis and Geologic modelling: relation to the disposal of radioactive wastes into geological formations. Proceedings of an OECD/NEA and CEC workshop in Ispra 23-27 May, 1977. 77-1158.
- NAGRA: Projekt Gewahr 1985 - Feasibility study; NAGRA Projekt Gewahr report series NGB 85-01 - NBG 85-09, NAGRA, Baden, Switzerland, 19485.
- National Research Council: A study of the isolation system for geologic disposal of radioactive wastes; National Academy Press, Washington, D.C. (1983).
- NEA (1989) Risks associated with human intrusion at radioactive waste disposal sites. Proceedings of an NEA workshop. OECD/NEA, Paris.
- NEA (1994) Characterisation of Long-term Geological Changes for Disposal sites. Proceedings of NEA workshop, Paris, France 19-21 September 1994
- NEA SCENARIO CATALOGUE (1988) A compilation of questionnaire responses on scenarios for performance assessments of radioactive waste disposal. OECD/NEA, Paris.
- NEA [PAAG/DOC (88)2] Systematic approaches to Scenario development. A preliminary report of the NEA working group on the Identification and selection of scenarios for Performance assessment of nuclear waste disposal.
- (1990) Safety assessments of radioactive waste management. Proceedings of the Paris Symposium, 1989. OECD/NEA, Paris 1989.
- (1991) Disposal of radioactive waste: Review of safety assessment methodologies. A report of the Performance Assessment Advisory Group of the Radioactive Waste Management Committee. OECD/NEA, Paris, 1991 Peaudecerf, P. and Blanc, P.L. (1990) French language publication
- Read, D., Krol, A & Thompson, B.G.J. & Thompson (1987) A structured programme of research into the Chemical aspects of radioactive waste disposal. In Scientific basis for waste management, Vol. 10. Mat Res Soc, Boston USA.
- Ringrose, P.S. et. al., Probabilistic simulation of the long-term evolution of radioactive waste disposal sites. In Proc. NEA/IAEA/CEC Symposium, Paris 1989.

- Roberds, W.J., Plum, R.J. and Visca, P.J. (1984) Proposed methodology for completion of scenario analysis for the basalt waste isolation project, Rockwell Hanford Operations Report RHO-BW-CR-147P.
- Sagar, B. and Janetzke, R.W. (1991) Total performance assessment computer code: description of executive module. Center for Nuclear Waste Regulatory Analyses, San Antonio, Texas. CNWRA 91-009.
- SKBF/KBS: Final storage of spent nuclear fuel - KBS-3, Stockholm (1989).
- Stephens, M.E. and Goodwin, B.W. (1989) Scenario analysis for post-closure assessment of the Canadian concept for nuclear waste disposal. In Proc. NEA/IAEA/CEC Symposium, Paris 1989.
- Sumerling, T.J. (1991) Dry run 3: A trial of methodology for the probabilistic risk assessment of underground disposal of radioactive waste that accounts for long-term environmental change: Overview. UK DoE Report TR-DR3-10.
- Sumerling., T.J. and Read., D. (1993) Approaches to long term performance assessment of deep underground disposal of radioactive wastes: A European perspective. Mat. Res. Soc. Symp. Proc. [294] pp 951-960
- Thompson, B.G.J. (1989) The time dimension in risk analysis: examples from recent work in the United Kingdom. In Risk Analysis in Nuclear Waste Management, ed. A. Saltelli et. al., Kluwer, Dordrecht.
- Thompson, B.G.J. (1987) The development for procedures for the assessment of the underground disposal of radioactive wastes: research funded by the Department of the Environment, 1982-1987. Rad Waste Man and the Nucl Fuel Cell Cycle, Vol. 9 pp 215-256.
- Thompson, B.G.J. [PAAG/DOC (88)11] A method of overcoming the limitations of conventional scenario-based risk assessments by using Monte Carlo simulation of possible future environmental changes. Technical report TR-DoE-12.
- Thompson., B.G.J. & Sagar., B. (1993) "The development and application of integrated procedures for post-closure performance assessment, based on Monte Carlo simulation: the probabilistic systems assessment (PSA) approach". Reliability Engineering and System Safety [42] pp 125 - 160
- Thorne, M.C. (1993) The use of expert opinion in formulating conceptual models of underground disposal systems and the treatment of associated bias. Reliability Engineering and System Safety.
- Tierney, M.S. (1991) Combining scenarios in a calculation of the overall probability distribution of cumulative releases of radioactivity from the Waste Isolation Pilot Plant, South Eastern New Mexico. SAND 90-0838. Sandia National Laboratories, Alberquerque, NM.
- Tyrer, M. Bennett, D.G. Read, D & Yunus, I (1995) "Near Field and Chemical Transport Modelling . Rep. UK DoE/HMIP TR-Z2-9
- USEPA (1985) "Environmental standards for the Management and disposal of spent nuclear fuel, high level waste and transuranic radioactive waste"; Final Rule. Federal Register [50] Washington D.C.
- Wilmot, R.D. (1993) The treatment of climate-driven environmental change and associated uncertainty in post-closure assessment. Reliability Engineering & System Safety. Vol. 42, pp181-200.
- Wittingham, R.B. (1989) Human intrusion into nuclear waste repositories: a systematic approach to the definition of scenarios using human reliability modelling. In Proc NEA Symposium, Paris 1989.
- Woo, G. (1989) Is the risk of human intrusion exaggerated. In Proc NEA Symposium Risk associated with human intrusion at Radioactive waste disposal sites, Paris, 1989.
- Wusche, D.M. et. al., (1995): Second interim assessment of the Canadian concept for nuclear waste disposal, AECL-8373-4.
- Ash, J., Sexton, H. (1985) A surface with one local minimum. Math. Magazine. 58: 147-149.
- Computer Graphics World (1989) Simple Beauty. Nov. 12(11): 78-81. (Describes author's work.)
- Dawkins, R. (1986) The Blind Watchmaker. W.W. Norton: New York. (Dawkins also uses the term "biomorph" to describe his interesting computer-generated biological shapes.)
- Dewdney, A. K. (1989) Computer recreations. Catch of the day: biomorphs on Truchet tiles, served with popcorn and snails. Scien. Amer. July 261(1): 110-113.
- Fadman, C. (1958) Fantasia Mathematica. Simon and Schuster: New York (A book of stories and diversions all drawn from the universe of mathematics).
- Flanigan, F. (1983) Complex Variables. Dover: New York. (A good introduction to complex variables.)
- Gardner, M. (1986) Knotted Doughnuts and Other Mathematica/ Entertainments. Freeman: New York.
- Hardy, G. A Mathematician's Apology. Cambridge Univ. Press: New York.
- Hobson, E. (1965) Squaring the Circle. Chelsea Publishing Co.: New York.

- Lawrence, J. (1972) A Catalog of Special Plane Curves. Dover: New York.
- Peterson, I. (1987) Pictures worth a thousand numbers. Washington Post, July 19th (Sunday). section C3, first page.
- Peterson, I. (1987) Portraits of Equations. Science News 132(12): 184-186 (and cover picture).
- Rivlin, R. (1986) Computer graphics: the arts. Omni Magazine. 8: 30.
- Rivlin, R. (1986) The Algorithmic Image. Microsoft Press: Redmond, WA. (also see excerpt in Computer Graphics World, August 1986).
- Rucker, R. (1988) Mind Tools.' The Five Levels of Mathematical Reality. Houghton Mifnin Co.: Boston.
- Sorenson, P. (1989) Pickover's strange attractor. Special Effects Business. December/ January 1 (3): 6-7; 53.
- Voelcker, J. (1988) Picturing randomness. IEEE Spectrum. August 25(8): 13.
- Anderson, B. (1988) Graphics high end: war of words. Computer Graphics Today. July Issue. pg. 1. (This article is not about fractals, but rather it describes the new class of graphics supercomputers which can be used to create fractals with startling speed.)
- The Beauty of Fractals (1987 photo essay) Computers in Physics. Nov./Dec. pp 26-3 1 .
- Aqvist, J., Tapia, O. (1987) Surface fractality as a guide for studying protein-protein interactions. J. Molec. Graph. 5(1): 30-34.
- Barnsley, M. (1988) Fractals Everywhere. Academic Press: New York. 15
- Briggs, J., Peat, D. (1989) The Turbulent Mirror. Harper and Row: New York. (Good for beginners.)
- Boyd, D. (1973) The residual set dimensions of the Apollonian packing. Mathematika 20: 170-174. (Very technical reading).
- Brooks, R., Matelski, J. P. (1981) The dynamics of 2-generator subgroups of PSL(2,C). In Riemann Surfaces and Related Topics.' Proceedings of the 1978 Stony Brook Conference.
- Kyra, I. and Maskit, B. (eds.) Princeton University Press: New Jersey. (Note: this 1978 paper contains computer graphics and mathematical descriptions of both Julia and Mandelbrot sets).
- Devaney, R., Krych, M. (1984) Dynamics of exp(z). Ergod. Th. & Dynam. Sys. 4: 35-52.
- Dewdney, A. K. (1985) Computer Recreations. Scien. Amer. 253: 16-24.
- Denning, P. (1985) Computing in the frontiers of science and engineering, cover page description. Commun. ACM. Vol. 28.
- Douady, A., Hubbard, J. (1982) Iteration des polynomes quadratiques complexes. Comptes Rendus (Paris) 2941 : I 23-1 26.
- Foley, J., and van Dam, A. (1984) Fundamentals of Interactive Computer Graphics. Addison-Wesley: Massachusetts.
- Family, F. (1988) Introduction to droplet growth processes: simulation theory and experiments. In Random Fluctuations and Pattern Growth.' Experiments and Models. Stanley, H., Ostrowsky, N. (eds.) Kluwer: Boston.
- Family, F., Meakin, P. (1988) Droplet growth. Phys Rev. Lett. 61 :428.
- Family, F., Landau, D., eds. (1984) Kinetics of Aggregation and Gelation. North-Holland: Amsterdam.
- Feder, J. (1988) Fractals. Plenum: New York.
- Fatou, P. (1919/1920) Sur les equations fonctionelles. Bull. Soc. Math. Fr. 47: 161-271 .
- Gardner, M. (1968) Packing of circles and spheres. Scien. Amer. 218: 130-125.
- Hamilton, W. R. (1969) Elements of Quaternions, Vol. I and II, reprinted by Chelsea ·Publishing Co.: New York.
- Hirsch, M. (1989) Chaos, Rigor, and Hype. Mathematical Intelligencer. 11(3):6-9. (Pages 8 and 9 include James Gleick's response to the article.)
- Hubbard, J. (1986) Order in chaos. Engineering: Cornell Quarterly. 20(3): 20-26 (Winter Issue).
- Julia, G. (1918) Memoire sur l'iteration des fonctions rationnelles, J. Math. Pure Appl. 4: 47-245.
- Kadanoff, L. (1986) Fractals: where's the physics? Physics Today February, 6-7.
- Kausch-Blecken, H. Schmeling, V., Tschoegl, N. (1970) Osculatory packing of finite areas with circles. Nature. March 225: 1119-1121.
- Kaye, B. (1989) A Randomwalk Through Fractal Dimensions. VCH Publishers: New York.
- La Brecque, M (1985) Fractal Symmetry. Mosaic 16: 10-23.
- Lakhtakia, A., Vasundara, V., Messier, R., Varadan, V. (1987) On the symme-tries of the Julia sets for the process z - zp + c. J. Phys. A: Math. Gen. 20: 3533-3535.
- Lakhtakia, A., Vasundara, V., Messier, R., Varadan, V. (1987) The generalized Koch Curve. J. Phys. A: Math. Gen.

20: 3537-3541.

- Lakhtakia, A., Vasundara, V., Messier, R., Varadan, V. (1986) Self-similarity versus self-affinity: the Sierpihski gasket revisited. J. Phys. A: Math. Gen. 19: L985-L989.
- Mandelbrot, B, (1983) The Fractal Geometry of Nature, Freeman, San Francisco.
- Mandelbrot, B. (1983) On the quadratic mapping z - z2 - μ for complex μ and z: The fractal structure of its M set, and scaling. Physica I 7D: 224-239.
- Musgrave, K. (1989) The synthesis and rendering of eroded fractal terrains. Computer Graphics (ACM-SIGGRAPH). July 23(3): 41-50.
- Norton, A. (1982) Generation and display of geometric fractals in 3-D, Computer Graphics (ACM-SIGGRAPH) 16: 61-67.
- Peitgen, H., Richter, P. (1984) Die Unendliche Reise. Geo (German Edition) 6: I OO-I 24.
- Peitgen, H., Richter, P. (1986) The Beauty ofFractals. Springer: Berlin.
- Saupe, D. (editors) (1988) The Science ofFractal Images. Springer: Berlin.
- Peterson, I. (1984) Ants in the labyrinth and other fractal excursions, Science News. 125: 42-43.
- Richardson, L. (1960) The problem of contiguity. In Statistics of Deadly Quarrels. (Wright, Q.,, Lienau, C., eds.) Boxwood Press: Pittsburgh.
- Robinson, A. (1985) Fractal fingers in viscous fluids. Science 228: 1077-1080.
- Roux, S., Guyon, E., Sornette, D. (1988) Hull percolation. J. Phys. Math. Gen. 2 1 : L47 5-L482.
- Schroeder, P. (1986) Plotting the Mandelbrot Set. Byte December 207-21 1 .
- Sorenson, P. (1984) Fractals. Byte. Sept. 9: I 57-172 (a fascinating introduction to the subject).
- Thomsen, D. (1982) A place in the sun for fractals. Science News. 121 : 28-32.
- Ushiki, S. (I 988) Phoenix. IEEE Trans. Circuits and Syst. July 35(7): 788-789.
- Voss, R. (1985) Random fractal forgeries. In Fundamental Algorithms in Computer Graphics. R. Earnshaw, ed. Springer-Verlag: Berlin.
- Voss, R. (1988) Fractals in nature: from characterization to simulation. In The Science of Fractal Images, Peitgen, H., Saupe, D. (editors).
- West, S. (1984) The new realism. Science 84, 5: 31-39.
- West, B., Goldberger, A. (1987) Physiology in fractal dimensions. Amer. Scien. 75: 354-365. (This article describes the fractal characterization of the lung's bronchial tree, the Weierstrass function, the fractal geometry of the heart, and "fractal time") .
- Witten, T., Sander, L. (1981) Phys. Rev. Lett. 47: 1400.
- Aono, M., Kunii, L. (1984) Botanical Tree Image Generation. IEEE Computer Graphics and Appl. 4: 10-34.
- Bloomenthal, J. (1985) Modeling the mighty maple. Computer Graphics (ACM SIGGRAPH). San Francisco. 19(3) 305-31 1.
- Braun, S. (1986) Botany with a twist. Science 86. 7: 63-64.
- Cohen, L. (1964) Random walk with transition probabilities that depend on the direction of motion. Math. Magazine 37(4): 248-250. (This work was used for designing plants).
- Dixon, R. (1983) The mathematics and computer graphics of spirals in plants. Leonardo 16: 86-90.
- Jena, R. (I 984) Mathematical Approach to Pattern and Form in Plant Growth. John Wiley and Sons: New York.
- Kappraff, J. (1986) The geometry of coastlines. Comp. and Maths. with Appls. 12B: 655-67 1 .
- Kappraff, J. (1986) A course in the mathematics of design. Comp. and Maths. with Appls. 12B: 91 3-948.
- Kawaguchi, Y. 1982. A morphological study of the form of nature. SIGGRAPH July '82 Proc.: Boston. 16(3): 223.
- Kolata, G. (1984) Esoteric math has practical result. Science 225: 494-495.
- Prusinkiewicz, P., Lindenmayer, A, Hanan, J. (1988) Developmental models of herbaceous plants for computer imagery purposes. Computer Graphics (ACM-SIGGRAPH) • August 22(4): 141-1 50.
- Reeves. W., Blau, R. (1985) Approximate and probabilistic algorithms for shading and rendering structured particle systems. Computer Graphics (ACM SIGGRAPH) 19(3): 313-322.
- Rivlin, R. (1986) The Algorithmic Image. Microsoft Press, WA.
- Thompson, D. (I 961) On Growth and Form. Cambridge: Cambridge, UK.
- Viennot, X., Eyrolles, G. Janey, N., Arques, D. (1989) Combinatorial analysis of ramified patterns and computer imagery of trees. Computer Graphics (ACM-SIGGRAPH) July 23(3): 31-40.

- Yessios, C. (1979) Computer drafting of stones, wood, plant and ground materials.
- Computer Graphics (ACM-SIGGRAPH) August '79 Proc.: Chicago 13(2): 190.
- Brown, D. (1987) Competition of cellular automata rules. Complex Systems. I: 169- 180.
- Conway, J., Berlekamp, E., Guy, R. (1982) Winning Waysfor Your Mathematical Plays. Academic Press: New York.
- Levy, S. (1985) The portable universe: getting to the heart of the matter with cellular automata. The Whole Earth Review Magazine. Winter issue, 42-48.
- Maeder, D. (I 987) The free energy concept in cellular automaton models of solid-solid phase transitions. Complex Systems l: 131-144.
- Peterson, I. (1987) Forest fires, barnacles, and trickling oil. Science News. 132: 220-22 1 .
- Poundstone, W. (1985) The Recursive Universe. William Morrow and Company: New York.
- Schrandt, R., S. Ulam (1 970) On recursively defined geometrical objects and patterns of growth. In Essays on Cellular Automata, A. Burks, ed. Univ. of Illinois Press, Chicago.
- Wolfram, S. (1983) Statistical mechanics of cellular automata. Rev. Modern Physics. 55, 601 -644.
- Abraham, R., Shaw, C. (1985) Dynamics - The Geometry of Behavior, Part 3.-Global Behavior. Aerial Press: California. (Actually, the entire book collection of Aerial Press, including the Visual Math Series, is an educational wonderland).
- Abraham, R., Shaw, C. (1983) Dynamics - The Geometry of Behavior, Part 2.-Chaotic Behavior. Aerial Press: California.
- Aronson, D., Chory, M., Hall, G. and McGehee, R. (1980) A discrete dynamical system with subtly wild behavior. In New Approaches to Nonlinear Problems in Dynamics, P.J. Holmes, ed. Soc. for Industry and Appl. Math.
- Berge, P., Pomeau, Y., Vidal, C. (1984) Order Within Chaos Wiley: New York.
- Campbell, D., Crutchfield, J., Farmer, D., Jen, E. (1985) Experimental mathematics: the role of computation in nonlinear science. Commun. ACM. 28: 374-389.
- Crutchfield, J., Farmer, J., Packard, N. (1986) Chaos. Scien. Amer. 255: 46-57;
- Devaney, R. (1986) Chaotic bursts in nonlinear dynamical systems. Science 235: 342-345.
- Dewdney, A. K. (1987) Probing the strange attractions of chaos. Scien. Amer. July Issue, 108-1 1 1 .
- Collet, P., J.P. Eckmann. (1980) Iterated Maps on the Interval as Dynamical Systems. Birkhauser: Boston.
- Feigenbaum, M. (1979) The universal metric properties of nonlinear transformations. J. Statistical Physics. 21 : 669-706.
- Feigenbaum, M. (1981) Universal behavior in nonlinear systems. Los Alamos Science. I: 4-27.
- Finney R., D. Ostberg (1976) Elementary Differential Equations with Linear Algebra. Mass: Addison-Wesley.
- Fisher, A. (1985) Chaos: The Ultimate Asymmetry. Mosaic 16: 24-30.
- Fischer, P., Smith, W. (1985) Chaos, Fractals, and Dynamics. Marcel Dekker, Inc.: New York.
- Glass, L., Mackey, M. (1988) From Clocks to Chaos.' The Rhythms of Life Princeton Univ. Press: New Jersey.
- Hassell, M. (1974) Insect Populations. J. Anim. Ecol. 44: 283-296.
- Hofstadter, D. (1981) Strange Attractors. Scien. Amer. 245: 16-29.
- Hofstadter, D. (I 986) Metamagical Themas.' Question for the Essence of Mind and Pattern. Bantam: New York.
- Kudrewicz, J., Grudniewicz, J. and Swidzinske, B. (1986) Chaotic oscillation as a consequence of the phase slipping phenomenon in a discrete phase-locked loop. Proc. of the IEEE Int. Symp. on Circuits and Syst. l: 74-78.
- Levi, B. (1981) Period-doubling route to chaos shows universality. Physics Today. March, pp. 17-19.
- Lorenz, E. (1963) Deterministic nonperiodic flow. J. Atmos. Sci. 20: 130.
- May, R. (1976) Simple mathematical models with very complicated dynamics. Nature. 261 : 459-467.
- Moon, F. (1987) Chaotic Vibrations. John Wiley and Sons, New York.
- Nussbau'm, R., Peitgen, O. (1984) Special and spurious solutions of ~(t) = - af(x(t - I)). Memoirs of the American Mathematical Society. 51 : l- 129.
- Peterson, I. (I 986) Toying with a touch of chaos. Science News. 129: 277-278.
- Schwenk, T. (I 976) Sensitive Chaos. Schocken Books: New York.
- Shaw, A. (1984) The Dripping Faucet as a Model Chaotic System. Aerial Press: California.
- Sinanoglu, O. (1975) Theory of chemical reaction networks. J. Am. Chem, Soc. 97:2309-2320. (The Sinanoglu references are included here because they are helpful for "Pattern Formation and Chaos in Networks" on page 142.)

- Sinanoglu, O. (1981) l- and 2-topology of reaction networks. J. Math. Phys. 22: I504- I5 I 2.
- Stoker, J. (1953) Mathematical methods in nonlinear vibration theory, In Proc. Symp. Nonlinear Circuit Analysis, Vol. 2. Interscience Publishers: New York. 28-55.
- Taubes, G. (1984) The mathematics of chaos. Discover Magazine. September Issue, 30-45.
- Bidwell, J. (1973) Pascal's triangle revisited. Mathematics Teacher. 66: 448-452.
- Crandall, R. (1978) On the "3x+1" problem. Math. of Computation 32: I 28 1 - I292.
- Crypton, D. (1985), Prime numbers and the national security. Science Digest. Oct. 86-88.
- Dodge, C. (1969) Numbers and Mathematics. Prindle, Weber, and Schmidt: Boston (easy reading).
- Gardner, M. (1977) Pascal's triangle. In Mathematical Carnival. Vintage Books: New York.
- Garner, L. (1981) On the Collatz 3n+1 problem. Proc. Amer. Math. Soc. 82: I 9-22.
- Gordon, J., Goldman, A., Maps, J. (1986) Superconducting-normal phase boundary of a fractal network in a magnetic field. Phys. Rev. Let. 56: 2280-2283.
- Hayes, B. (1984) Computer recreations: on the ups and downs of hailstone numbers. Scien. Amer. 250: I0-16.
- Holter, N., Lakhtakia, A., Varadan, V., Vasundara, V. Messier, R. (1986). On a new class of pianar fractals: the Pascal-Sierpihski gaskets. J. Phys. A: Math. Gen. 19: 1753-1759.
- Jansson, L. (1973) Spaces, functions, polygons, and Pascal's triangle. Mathematics Teacher, 66: 71-77.
- Lakhtakia, A., Vasundara, V., Messier, R., Varadan, V. (1988) Fractal sequences derived from the self-similar extensions of the Sierpihski gasket. J. Phys. A: Math. Gen. 21 : 1925-1928.
- Legarias, J. (1985) The 3x+1 problem and its generalizations. Amer. Math. Monthly, January 3-23. Schroder, M. (1984) Number Theory in Science and Communication. Springer: New York (This book is recommended highly. An interesting book, by a fascinating author).
- Schroder, M. (1 982) A simple function. Math. Intelligencer. 4: I58-161 .
- Stein, M., Ulam, S., Wells, M. (1964) A visual display of the distribution of primes. Math. Monthly 71: 516-520.
- System Product Interpreter User's Guide, Release 4 (1984) IBM manual (SC24-5238-2).
- Trigg, C. (1982-83) Palindromic octagonal numerals. J. Recr. Math. 15(1): 4 1 -46.
- Usiskin, Z. (1973) Perfect square patterns in the Pascal triangle. Math. Magazine Sept.-Oct. 203-208.
- Wagon, S. (1985) The Colatz problem. Math. Intelligencer 7: 72-76.
- Wiggin, B. (1988) Wonderous numbers - conjecture about the 3n+1 family. J. Recr. Math. 20(I): 52-56.
- Zhiqing, L. (I985) Pascal's pyramid. Math. Spectrum I 7(1): 1-3.
- Benzinger, H., Burns, S., Palmore, J. (1987) Chaotic complex dynamics and Newton's method. Physics Letters A. I 19: 441-445.
- Grove, W. (1966) Brief Numerical Methods. Prentice-Hall: New Jersey (Has an excellent section on root-finding methods).
- Hamming, R. (1973) Numerica/ Methods for Scientists and Engineers. Dover Publications, New York.
- May, R. (1976) Simple mathematical models with very complicated dynamics. Nature. 261 : 459-467.
- Peterson, l. (I987) Zeroing in on chaos. Science News I3 1 :1 37-1 39;
- Press, W. Flannery, B. Teukolsky, S., Vetterling, W. (1986) Numerical Recipes. Cambridge University Press: New York.
- Tritton, D. (1986) Chaos in the swing of a pendulum. New Scient. July Issue: 37-40.
- Wimp J. (1984) Computation with Recurrence Relations. Pitman Publishing: Boston.

2　オブジェクト指向のソフトウェア開発

本章では、地層処分ヴァーチャル・エンジニアリング・システムを開発するための、基本的なソフトウェア開発方法論となるオブジェクト指向についての方法論を明らかにするための検討を行う。それにより、前章で示したマルチ・エージェントによる協調計算の発展した考え方として、モデル協調計算手法の開発可能性に関する基本的な背景を整理する。

地層処分ヴァーチャル・エンジニアリングは、計画する処分システムの性能を評価するために提案された評価シナリオにそって、さまざまなモデルを複雑に組み合わせてシミュレーションを行い、結果を評価し、処分システムの健全性を超長期に亙り確認する機能を提供することが主な開発目標である。それは、漏出した放射性核種が、生物圏に至らないよう隔離防護するための工業製品である人工バリアと、天然の地層によるバリア効果をもって、２重に安全性を確保しようとする設計思想について確証を得るための手段として、コンピュータ（計算）を利用する科学技術の手法である。そのため、計画する処分システム（人工バリアと天然バリアから構成される放射性核種の漏出・輸送防護機能を提供する系。従って、処分システムは処分場とは区別して考えるべきである）について、超長期的な時間を含む地層環境中での振る舞いを予測評価することが性能評価の基本的な課題となる。性能評価においては、計画する処分システムに関する数10万年までの期間を含む、超長期的な地層環境変動と多重バリアの振る舞いに関わる仮説を立て、それについて試験やシミュレーションを用いて検証を行い、総合的な予測の確証を得る手法が一般に取られてきている。

そのような性能評価作業では、各種の現象解明に関わる数理モデルやシミュレーション・コードの開発がなされている。開発された数理モデルやシミュレーション・コードは、一般に試験や観測で得られたデータから定式化されたものである。それは、設計論拠の決定や安全性評価等の、ある意味で汎用な意味合いをもつモデルであると考えられる。一方、研究作業においては、設計（安全解析を含む）と評価は基本的に異なると考えるべきである。

設計作業においては、例えば処分場建設工事における水抜きや、発破の地下水の水理への影響、吹き付けコンクリートが地下水に与える化学的特性、ロックボルトが岩盤に与える力学的特性等は、許容できる範囲であるかといった、個別の設計課題に関わる安全性評価や論拠造りに、数理モデルやシミュレーション・コードが利用される。このような設計作業における仮説の検証や論拠造りの特徴は、評価（解析）しなければならない問題が比較的簡素化されていること、もしくは、簡素化できることがあげられる。なぜならば、前述のように議論の対象となる工学的問題は、始めからある程度限定された問題自身が明確な問題を対象として扱うことができるためである。

一方、性能評価は設計作業ではなく、環境アセスメントの領域であると考えられる。作業の目的は、人間が開発した処分システム（自然に対する開発行為と考える）が自然環境を変更し、最終的には生物の生存を脅かす可能性があるのかを評価することである。問題設定においては、処分システム全体としての地層環境中における振る舞いが問題視され、個別の現象過程からは予測しきれないシステム全体の振る舞いについてのアプローチが重要になる。従って、一般に性能評価は、環境アセスメントと同等のアプローチとして、その初期の段階では、枠組み（フレーム）とシナリオを開発することから始める。これはまた、環境開発として生じた地層中の処分システム全体が、環境に与える影響を超長期的に評価し、将来的に地球環境問題に発展しないことを確認する作業とも考えられる。

環境問題の基本は、人類の自然に対する働きかけ（開発行為）が、その量・質および時間等のすべ

ての面において、自然のもつ許容量を超えてしまったことにより発生するものと考えることにある。すなわち、自然のシステムが保有している環境維持許容量以上に、人類システムからの自然システムへの要求量が過大であるため、その未処理分が蓄積され、顕在化して行くことにより生じる問題と考えられる。

それは、本来産業や地域開発の真の目的が、健康と安全の維持や確保等、人類生活の質を改善することであるはずのものが、実際には、その目的とはまったく逆の結果をもたらす場合といえる。このような地球環境問題を発起する可能性のある開発行為には、自然体系の構造と機能・多様性を考慮しつつ、かつ人間活動をも含めた相互依存システムの維持管理を十分に検討・評価する活動が必要である。例えば、国連環境計画等による**「新世界環境保全戦略」**においては、人類の行動規範として次の3つのことが行われなければならないと示されている。

- **生命を支えるシステムを保全すること。**
  地球を生存に適する状態に保つ生態学的プロセスが機能することであり、気候が安定し、大気と水は浄化され、水の循環は調節され、必須元素は再循環され、土壌が造られ再生され、生態系は自ら回復・再生する機能を保全すること。
- **生物学的多様性を保護する。**
  すべての植物種・動物種・その他の生物種を保護するだけではなく、種内の遺伝系統や食物連鎖等個々の生態系そのものの多様性を現在より少ない状態としないこと。
- **再生可能な資源の利用は持続可能な方法で行なう。**
  再生可能な資源には、土壌・野生生物種および家畜（栽培）種・森林・放牧地・耕作地・漁業を支える海洋・淡水生態系が含まれる。資源の利用は、資源の回復・再生能力の範囲内にとどめ、持続可能な利用を将来に亙り確保すること。

こうした地球環境問題は、地球が長い年月をかけて作り上げてきた全体のバランスが壊れ、その結果として国境を越えた地球的なスケールで、その影響が現象化したものであり、それらのほとんどが不可逆的である。そのため、この問題の解決にはグローバルな国際協力を要し、さらに科学技術的対応以外に、経済社会活動や人間のライフスタイルの在り方といった根源的課題も含んでいる。

以上のように、性能評価を環境アセスメントの観点から捉えると、アプローチについて以下のように整理することができる。

- **人類システムと自然システムの相互作用問題であるとして扱うこと。**
- **地層中に建設された処分場と多重バリア、そして自然体系という段階的プロセスに対する、総合的な枠組みが必要である。**
- **設定された枠組みに添った、多方面からの評価シナリオの開発が必要である。**
- **開発された評価シナリオを検証するための手段が必要である。**
  手段として、野外調査、室内試験、コンピュータ利用が上げられる。

上記のアプローチにおいて、人類システムと自然システムの相互作用問題であるとして扱うということは、地層処分性能評価問題が、本質的には社会・経済的なリスクを含んだ人類の活動であることを示している。それは、妥当な予算で環境・社会に影響を与えず、放射性核種が消滅するまで安全に

隔離処分することを要求する。

地層中に建設された、処分場から漏出した核種移行に関わる段階的な地層環境プロセス評価の枠組みの作成は、現在のところ、地質学、地球物理・化学等の知見や、野外調査等により得られたデータから、人的活動として組み立てるものと考えられるが、現代の地球化学や進化生物学では、歴史が無作為な発展として理解されてきている（繰り返さない）ので、地質情報等から過去のデータを取りだし、未来に向けて安易に外挿することで説得力は得られないと考えられる。地球環境は、例えばカンブリア期の生物の爆発的発生と不運な死滅、恐竜等、偶然生じて二度と繰り返さないと考えるほうがむしろ自然なモデル感と思われる。そのような仮定のもとに、どのように未来プロセスの枠組みを決めれば良いのかは、今後の重要な研究課題となるであろう。本件は、本研究の領域を外れるため対象として取り上げない。

評価シナリオの開発は、地層環境アセスメントの枠組が定まった後、それに従って、処分場より漏出する放射性核種の地層中移行に関わる事象について、系統的に説明する仮説を準備する作業である。この作業の複雑さは、予想される評価シナリオが、設計段階に考慮しなければならない事象も含む核種移行に関わる想定事象について、フィードバックを含んだ連結事象のループとなり、そのままでは計算可能性が失われることが考えられること（決定不能問題）、また、FEP等から構築された場合、シナリオそのものに心理的・主観的側面が含まれると思われ、それを除去することは当面困難と予想されることが上げられる。これらの問題に関する考えられる戦略として、第一に、自己再帰的なフィードバック・ループを持つネットワークの問題については、段階的プロセスを世代と考え、遺伝子アルゴリズム等を利用して最適化を計り、計算可能な問題に帰着させることが考えられる。また、FEP等に含まれる主観や心理作用も、問題は信念や世界観を持つ文脈依存のエージェント・モデル等で用いられる可能様相公理系等、数論や論理学に出発点を置いた系統だった評価を文脈に対して行い、判断する手法等が考えられる。それらは、今後の重要な研究課題となるであろう。本件は、本研究の領域を外れるため対象として取り上げない。

シナリオを検証する手段には、野外調査、室内試験、コンピュータの利用等があげられる。これらのうち、野外調査や屋内試験は、前述の理由から、未来に対する系統だった予測評価に利用するには、充分な情報を提供することは難しいと考えられる。従って、コンピュータによる試行錯誤的な数値実験が期待できる総合的な評価手段である。地層処分ヴァーチャル・エンジニアリング・システムはそのような期待に基づいて開発するものである。

以上のように性能評価作業をとらえてくると、それは将来起こるであろう地層中を中心とした環境の変化と、放射性核種の移行に関わる説明の理論モデルを組み立てる活動と考えることができる。また、作業にあたっては、設計検討のように個別の現象について解明を行っていく方法をとることは難しく、処分システム全体の挙動として捉えて行くべきである。そのような問題の領域では、数理モデルは、始めからある程度の自由度をもった連成形式となるであろう。それら連成モデルを試験等から導くことは非常に難しいであろう。なぜならば、一般に、試験はモデルを簡素化し、最も支配的な法則を見い出すことに目標が設定されるためである。また、そのような試験条件から定式化された数理モデルは、一般に単純であり、試験にて示されるような理想化された環境化において観測される現象を、ある特定の条件の範囲でよく記述（近似）するものであるが、現象そのものに対する普遍的な記述を始めから除外されていると考えるべきである。

従って、目的とする連成現象を記述する複雑な数理モデル開発することは、解析的（理論）にも、試験的（実験）にも難しいと考えられる。そのような要求においては、試験は、それ自身の有する還元性から、より複雑なモデル開発に利用することは矛盾している。また、複雑な連成条件を示す非線形連立偏微分方程式を組み立てることは可能と思われるが、恐らくそれは解析解を持たないであろうし、離散近似する場合も多くの困難を有するであろう。さらに、例えある程度の条件の基に離散近似を行って、計算可能性（アルゴリズム）を見い出しても、それを具体的な超並列コンピュータやマシン・パラレルのメッセージ交換アルゴリズム等にプログラムをするための、多くの知識、技術が必要となるであろう。そのように考えると、これ迄に示してきた性能評価のアルゴリズム開発戦略は、従来のアプローチでは非現実的である考えられる。そこで、それらの従来型アプローチとは根本から異なった戦略が、このような中規模の複雑度をもつ問題に必要となる。そのアプローチの最も基本的な考え方は、オブジェクト指向によるモデル開発とシミュレーション手法である。以上のことから、本章ではシステム開発の基本要件として、オブジェクト指向によるモデルの考え方を導き、あらたな手法研究の準備とする。

２．１　新たな研究開発の戦略

近年、オブジェクト指向プログラミング言語の研究が進み、いくつかの応用分野が考えられるようになった。たとえば、システムのプロトタイピングや知識ベース・システムなどは、今後の有望な応用分野として期待されている。これらの分野に共通な特徴は、考えている問題や、その解決プロセスの表現が、オブジェクトとその間のメッセージパッシング技術によって自然に表現できる、ということである。

本章では、オブジェクト指向ソフトウェア開発戦略を、物理化学等のシミュレーションに導入する場合について、基礎的な技術をとりまとめ、地層処分ヴァーチャル・エンジニアリング・シミュレーションのプログラム開発手法のとして位置付ける。オブジェクト指向を物理化学等のシミュレーションに導入することにより、以下のような事項を期待できる。

- オブジェクト指向での表現が適していると思われるニュートン力学系は、物理学の最も基本的な分野であり、また、評価に於ては掘削や地震影響等比較的、非専門家も関心を示しやすい議論の対象となる。
- 力学系内の個々の対象（坑道、廃棄体、亀裂、等）をオブジェクトで、また、対象間の力の及ぼし合いをメッセージパッシングによって表現することが、比較的自然に行なえる。
- 熟達者が問題を解くときによく行なうような、いくつかの対象をまとめて１つの仮想的な対象とみなすという思考方法が、その仮想的な対象を新しいオブジェクトとして作り出すことによって、自然な形でシミュレートできる。この考え方は、モデル開発の領域では連成問題として議論され、計算機科学の領域からは、並列に協調して独立したタスクとして動作するオブジェクト（即ちエージェント）のプログラミング・アルゴリズムとして扱われる。

２．１．１　オブジェクト指向表現

例えば、オブジェクト指向型表現が、力学の問題の構造を自然に表現できるならば、学習者が自分で問題を作り、システムにその問題を解かせることによって問題を理解する（もしくは観察する）ことが比較的自然に行なえるであろう。そのような機能は、地層処分ヴァーチャル・エンジニアリングのシミュレーション機能開発にとって重要な設計開発思想である。なぜなら、学習者の考える問題表現とシステム内部でのオブジェクト指向的表現が、ある程度似たものとみなせるからである。そこで、将来の地層処分ヴァーチャル・エンジニアリング・システムは、問題解決システムに問題作成のためのインタフェイスを統合した、問題理解のためのシステムとなることが予想できる。

２．１．２　方法論

オブジェクト指向は、一つの方法論である。それはソフトウェア開発における一つの方法論であると考えることができる。従来型のFORTRAN言語に代表される手続き型プログラミングでは、計算はその順序が決められていた。オブジェクト指向では、シミュレーション対象モデルを変量と式の組み合わせで記述し、それが外界からのメッセージを受け取ると、内部にある式に従って変量を変化させることにより、モデルの状態を変化させるプログラミング手法を考えることができる。そのようなモデルとプログラミングの手法は、次の点が従来のメインフレームに代表される手続き型プログラミングと相違する特徴と考えられる。

従来、一つのプログラムとしてまとめ上げられたプログラムは、オブジェクト指向の考え方では、個々に独立してメモリー上にロードされ動作すると考えた。それらがメッセージを交換しながらシミュレーションを実行していく処理は、プログラミング上自然に並列的になるので、並列処理や分散処理には適していると考えられる。また、このようなシミュレーションの内部処理技術は、シナリオに沿って複雑な連成問題を計算していく、地層処分ヴァーチャル・エンジニアリング・システムの設計思想に沿ったものである。

２．１．３　オブジェクト・シミュレーション

オブジェクト・シミュレーションは、計算機科学を基盤として生じてきた新たなシミュレーションの考え方である。ソフトウェア開発にあたっては、従来型のシミュレーション・プログラムの開発手法とは異なった技術、手法が基盤となるため計算科学技術の基盤研究開発課題として重要である。

オブジェクト・シミュレーションにおいては、実際に複数のプロセッサやマシン環境で実行する場合の実行プロセスの同期制御等が重要課題となると考えられる。それは、メモリー上にロードされた個々の手続きは、メッセージを交換しあって状態を変える。メッセージは、テキスト形式のファイルとして受渡しされることを原則とする。状態とは、メッセージに応じて変化するメモリーの内容であると考えられる。一つのモデル（処理単位）は、状態量を格納する領域と、それを決定する操作からなるとする。ここに言う“操作”とは、従来の手続き型プログラミングにいう命令の集合を決定する手法として扱い、状態量を格納するメモリーと“組”にして考えるものとした。

開発する地層処分ヴァーチャル・エンジニアリング・システムのオブジェクト・シミュレーション機能においては、シミュレーション条件を組み立てる為に、物理現象を個々の単純な個別現象シミュレーション・モデルの組に分解し、単純化した個別モデルについて、その振る舞いを決定する計算条

件を付加し、それらを関連付けて実行することが望ましいと考えられる。そのためには、オブジェクト思考モデル分析に関するより詳細な研究が必要である。

以上のように、物理モデルを示す支配方程式からプログラミングの為に離散方程式を導き、それをもとに一連の計算をプログラム言語に書き下ろすと言う、従来用いられてきた手続き型プログラミングの方法と異なったモデルと解法のアプローチとして、オブジェクト指向をとらえることができる。また、このオブジェクト指向によるソフトウェア開発手法は、これからの計算科学技術の開発課題である“**マルチエージェント協調計算手法を応用した科学技術シミュレーション**”研究の基礎的な技術を提供する。

オブジェクト指向によるシミュレーションと、従来型の有限要素法や差分法によるシミュレーションは基本的に異なった思想に基づくものである。誤解のないよう下記の通りに区別する。

・シミュレーション（決定論）

シミュレーションは、実験等で観測された物理化学現象等を記述する支配方程式を導き、それを基に計算コードとしてプログラムしたものを利用する。これは、観測された事実を計算式（数理モデル）に置き換えたものを利用して数値的に再現することを目指している。数理モデルから出発する。

・数値実験（オブジェクト指向論）

オブジェクト指向のシミュレーションは、保存則等の場の方程式から出発するものではない。それは、分解された対象、個々について外部からの情報の交換と、それにより状態の変化を記述する操作からなる。従って、シミュレーションはより実験的である。オブジェクト・シミュレーションを従来のシミュレーションと分離して数値実験と呼称すると良いかもしれない。いずれにしろ、オブジェクト・シミュレーションでは、シミュレーションを行うことにより結果を観察し、定式化する為の参考とする戦略をとる。オブジェクト・シミュレーションが、一般のハードウエア試験に比べ有利な点は、シミュレーションを構築する系が素過程であり、最終的には０と１のディジタル信号に完全に分解できることである。一般の実験は、そのように分割可能ではなく複雑さを残す。

このように、オブジェクト指向はより理想的な仮想試験環境を提供すると考えられる。

２．１．４　新しいパラダイムの必要性

不幸なことに、良く知られているナビエ・ストークス方程式は、解析的には（つまり微分積分学の公式を使ってその解を数式として求めても）めったに解かれ得ず、コンピュータの力を借りた数値的な解析方法（数値計算）によらなければならない。ここに、次のような伝統的な方法論ではとても解決できないように思われる問題が生じる。

コンピュータで二次元問題（それは空間の二次元でも、空間の一次元と時間の一次元でもよい）を扱うことは比較的普及している。計算量もＰＣやワークステーションで殆どの問題は充分扱えるであろう。しかし、三次元空間での定常流や平面内の時間依存性のある流れのような流れの解のひとつを得るためには、大規模な計算（超並列計算機等のスーパーコンピュータ上で数時間あるいは数日間かかる）が必要となるであろう。さらに、三次元空間内の時間依存性のある流れのような四次元流を解

くのは、膨大な計算量（数週間、あるいは数ヵ月かかるかも知れない）となるであろう。このように、問題の次元とともに計算所用時間が急速に増大する理由は、数値解析と科学的なデータ表現の本質に根ざしている。

例えば、一次元空間に沿って、すなわち線上に分布したデータの計算のみに限って考察する。その直線を点で区切って有限の格子に分け、各々にひとつの数値を割り当てる。仮に10個の格子点があるとしよう。このような一次元問題は10個の数値を必要とする。二次元問題なら二次元格子を必要とするので、10x10すなわち100個の数値を必要とする。三次元問題は10x10x10で1000個の数値を必要とし、四次元問題では一万個となる。もし、1km立方の地層中を流れる地下水を1mメッシュで表現し、計算しようと思えば、空間的に1000x1000x1000、そして時間的には1年毎に1000年の計算をしようとしたら1000個の格子を必要とするだろう。この時、３つのパラメータ（x方向に地下水速さ、y方向に地下水速さ、z方向に地下水速さ）の各々が、10通りの値をとって変化するとしたら10の12乗すなわち一兆個のデータ点を計算する必要がある。その場合、一次元問題を解くのにスーパーコンピュータで10秒かかるとすると二次元問題を解くには1000秒かかり、三次元問題を解くには16分と40秒、そして四次元問題を解くには約28時間かかるであろう。

以上のようにして、３つの重要なパラメータに地下水の流れがどのように依存するかを調べるのに要求される1000個の四次元問題を、この方法で解くには約1年かかるであろう。その結果、膨大な量の数値が計算結果として得られるだろう。そしてそれは、「なぜ」については、いかなる理解も与えてくれないであろう。このように計算することのみに主眼を置いた方法論を、地層処分ヴァーチャル・エンジニアリング・システムの開発に持ち込むことは、費用対効果、現実性を考えると危険であり、評価という観点からも妥当でないと考えられる。

なぜならば、問題が、工業製品の設計を確認することであるならば、それが機能することがわかればよいので、「なぜ」に気を配る必要はない。同じことが、前述した処分場の設計にあてはまる。処分場設計では、機能することを確認するために安全性評価をおこなうが、環境アセスメントとしての性能評価では「なぜ」に答えることが要求される。このように機能確認の為の計算は、科学的洞察を得る上では役立たないであろう。それに答える何か新しいパラダイムが必要である。

そのためには、地下水の現実的な物理とはまったく無関係な、非常に一般的な疑問を投げかけることから始めることが大切であると思われる。これは、流体力学等における伝統的な研究気質に対立するものと思われるが、数学的、科学的には意義があるアプローチと考えられる。

はじめに問題の一般的な本質を理解し、それから攻略計画を立てるために、それを理解することに努力すべきである。それには、他分野からの「技術移転」も重要である。物理についての仮定が特殊なものでなければないほど、その結果はより広範囲に応用可能なものになるだろう。オブジェクト指向は、そのような観点からのパラダイムである。同様に系の対称性は何か。その対称性をもつ系において可能なパターンのカタログは何か。これら可能なパターンが実際に起こるかどうかを決定するためには何を計算しなければならないか。そしてその安定性はどうか。これら一般的な疑問を整理してから計算をすることにより、計算量を減らしシミュレーションをより現実的な作業に組み込むことができると思われる。そのように、シミュレーションに置いては、唯、超高速コンピュータを利用して大量の計算結果を吐き出す数理モデルを開発することより、むしろ、問題の本質的な部分を整理してモデル化にあたるべきである。

例えば、前述のモデル対称性の例では、流れのパターンについて、より多くの知識を整理してデータベース、知識ベース等の情報ベース化することにより、計算そのものを簡素化するだけでなく、この系以外の同じ対称性をもつ他の任意の系について、数学的な特徴の幅広い理解を得るであろう。例えば、多孔質媒体中の浸透流れ解析については、微小な領域から広域的領域まで共通した対称性流れの考え方を見い出して応用できるかも知れない。そのようにして、可能な限りモデルに独立な概念を使用し、最後にモデルの詳細な物理の研究を行うべきである。例えば、地下水流れの対称性の議論は、可能なパターンの一般的な範囲を選択することを示すであろう。そして物理は、どのパターンが実際に起こるかを教えてくれる。

本項で主張していることは、**“知りたいものによっては別の有効な戦略が存在する”**ということであり、かつ異なった戦略は、異なったタイプの問題に対して有効であるということである。オブジェクト指向がそのような戦略としてシステム開発に期待できるとともに、モデル化においても、対称性の整理が、より現実的な数理モデルを開発する一つの戦略であることも合わせて示した。このような科学に対するパラダイムは、地層処分性能評価シミュレーション・システムのような、統合化された複雑系として振る舞うアルゴリズムを開発する場合には特に重要であると考えられる。

２．２　オブジェクト指向ソフトウェア開発

前章までに、オブジェクト指向プログラミング（OOP:Object-Oriented Programming）とは、ソフトウェア・システムの設計、及び実現のための比較的新しい方法論であることをみてきた。その主要目標は、ソフトウェアの拡張可能性と再利用可能性を改善してプログラミングの効率を向上し、ソフトウェア開発の費用対効果比を向上させることである。また、オブジェクト指向プログラミングにより、ソフトウェア開発の設計面と開発面の結び付きを一層深めることができると考えられる。さらに、物理化学等の問題について新しいモデル化手法の開発の可能性があると考えられる。

ソフトウェアの設計および実現の方法は、実に様々であるが、このオブジェクト指向開発法は成功の可能性、将来性の面でかなり期待をしてよい開発法であると思われる。個々の応用分野において、オブジェクト指向プログラミングの採用により成功したとの報告はあるが、地層処分関連のソフトウェア設計開発分野では、オブジェクト指向方法論による成功例はまだ報告されてないようである。

（１）型

オブジェクト指向プログラミングは、抽象データ型とクラス、型の階層構造（サブクラス）、継承、ポリモーフィズムなどいくつかの新しい概念を軸に構成されている。抽象データ型（abstract data type）とは、型（type）とその型に付随する演算（operation）を取り囲む一つのモデルである。この考え方が最もオブジェクト指向らしさを表すとともに、最も議論される対称と考えられる。それは、演算を定義することにより、その型の動作が特性づけられるとする考え方である。

（２）クラス

多くのオブジェクト指向言語では、クラス定義（class definition）により、基本となる抽象データ型のすべての演算のインターフェイスを定義し、そのデータ型の動作を記述する。また、クラス定義でデータ型の実現の詳細、つまりデータ構造も定義する。通常、実現の詳細は、クラスの中でしかアク

セスできない。そのようなデータ構造はプライベート（private）と呼ばれる。一方、データ構造のすべて、または、ある部分がクラスの外からもアクセスできる場合、そのようなデータ構造をパブリック（public）と呼ぶ。

クラスは情報のモデル化の抽象的な方法論であると考えられる。性能評価シミュレーションをオブジェクト指向で組み立てることを考えた場合、評価シナリオはこの抽象データ型により開発されたクラス階層で記述される必要がある。このことは、シナリオ開発において、その導出過程にクラス概念を適用する研究の必要性を生じるものである。また、この評価シナリオのクラス導出は、性能評価をオブジェクト指向で捕える研究の最も基礎的な部分である。

（3）演算（メソッド）

型で定義された演算も普通パブリックとプライベートに分類される。パブリックな演算はクラスの外からもアクセスできる。一方、プライベートな演算はクラスの中でしかアクセスできない。オブジェクト指向言語は、クラスで定義された演算をメソッド（method）という。メソッドとは、オブジェクト指向でない言語のプロシジャや関数にあたる。仮にクラスのパブリックな演算が、汎用的で応用範囲が広ければ、そのクラスは再利用可能なソフトウェア部品の基本となり得るであろう。

（4）オブジェクト

オブジェクト（object）とは、その型が特定のクラスであると宣言された変数（ここに言う変数とはクラス定義されたデータ構造とメソッド）である。オブジェクトは、クラス定義で定義された（プライベートおよびパブリック）データ領域の、すべてのコピーを保有することにより、状態をカプセル化する。オブジェクトは、クラス定義されたメソッドを呼び出すことにより動作させる。この時オブジェクトへメッセージ（message）を送るという。メッセージは、非オブジェクト指向言語のプロシジャ呼出しや関数呼出しと同様にパラメータを添えることができる。

（4）インスタンス

各クラスの変数は、そのクラスのインスタンスを表現する。従って、同じクラスで宣言されたオブジェクトでも、その値の集合はそれぞれ異なる。オブジェクト指向言語が拡張性が高いといわれるのは、プログラマが新しい型を定義し、その性質、動作をクラス定義で与えることができるためである。これら新しいクラスは、プログラミング言語自身で定義されている型と、ほとんど同様に扱うことができる。

（6）階層構造

また、オブジェクト指向の考え方は、サブクラスによる型の階層構造の定義を可能にする。サブクラス（subclass）のオブジェクトは、親クラスの性質（またはその部分）を受け継ぐが、親クラスには無い特性も有することができる点である。サブクラス構造を用いて、ソフトウェア開発のコストを低下させることができる。また、サブクラスを用いて段階的開発もできる。既存のソフトウェア要素を修正したり（その要素のソース・コードは入手可能と仮定する）書き直したりせずに、新しいサブクラスを基本的なクラスの集合から作ればよいからである。これら新しいサブクラスのオブジェクトが

ソフトウェア・アーキテクチャの基本構造を定めることとなる。

これらの特徴が、オブジェクト指向をソフトウェア再利用に向いた手法、即ち、ソフトウェアの部品化と規格化に向いた手法として考えられる理由である。この考え方は、今後のソフトウェア開発にとって強力な手法と考えられる反面、従来の手続き型プログラミングとは考え方に大きな相違があるため、開発蓄積してきたソフトウェアの知験とプログラムを断続無く、オブジェクト指向の世界に成長させる手法の開発が今後の研究課題である。

2．2．1　クラス階層のソフトウェア構造

オブジェクト指向ソフトウェアの構造は、そのソフトウェアの動作を特徴づけるクラスの集合から構成される。各クラスのオブジェクトは、クラスのメソッドを呼び出すことにより実行を行う。これは、オブジェクトへメッセージを送る操作で実現する。従って、メッセージがオブジェクトに対して行われる操作を表現することになる。オブジェクト指向プログラミングは、操作する手続きよりも操作されるデータに重点をおいている。

データはソフトウェア分割の基本となる考え方である。これは、クラスの概念が拡張されたデータ型であることが背景となっている為である。実際、オブジェクト指向ソフトウェア設計では、ソフトウェアを構造化データの型、つまり基本クラスとサブクラスに分解し、それら基本クラスとサブクラスそれぞれの動作を定義することがソフト開発の中心となる。

また、オブジェクト（つまりクラス（サブクラス））変数は、対象としている問題の物理的要素、または論理的条件を特徴付ける要素（例えば物理変数）として扱う。従って、オブジェクト指向で物理工学上の問題を扱う場合、対称となるモデルを記述するクラス階層モデルを設計することが研究課題の中心となるであろう。

オブジェクト指向ソフトウェアの概念を定義し、そのシステム設計を行う段階においては、クラス集合（サブクラスを含む）、及びその定義とオブジェクトのみを定義することが一般的である。各クラスの動作はメソッド・インターフェイスで決定する。しかし、メソッドの実現上の詳細は、システム設計には含めない。通常のオブジェクト指向言語では、クラスで定義されるメソッドのインターフェイスと、その実現の詳細は独立に定義できる。このことにより、システム設計とその実現を分離できることになる。これは、オブジェクト指向方法論に於ける重要な概念であるが、前述の通り性能評価などに於ては、従来蓄積してきたFORTRANコード等とのインターフェィス設計技術としてメソッドの考え方を応用するには今後の研究が必要である。

このように、概念（メソッド・インターフェイスを伴うクラス定義）を、その実プログラムの生成（クラスのデータ構造とアルゴリズムを実現するコードの生成）と切り離すことは、オブジェクト指向プログラミングの場合、また再利用可能性や保守コストの削減のために非常に重要である。

オブジェクト指向プログラミングの概念は、メソッド・インターフェイスの形で定義され、クラス内に閉じ込められるため、再利用可能性（reusability）に優れている。ユーザは、メソッド・インターフェイスで定義された、クラス・オブジェクトの動作さえ理解していれば十分であり、そのオブジェクトの実現の詳細について知る必要はない。メソッドの実現は、ユーザから見えない“ブラック・ボックス”に閉じ込められる。

さらに、データ構造やアルゴリズム（すなわちクラス実現のコード）の実現に変更があっても、そ

のクラス内のコードの部分だけ変更すればよいので、保守性（maintainability）も優れている。その場合、クラス・インターフェイスに変更は生じないので、クラスの外は何ら影響を受けない。このインターフェイスは、クラスの外からクラス・オブジェクトを“利用する”ときの利用の形式を定めることになる。

（1）ソフトウェア開発管理面からのオブジェクト指向ソフトウェア開発の狙い

基本クラスを用いて再利用を推進したり、サブクラスを用いる段階的開発法を活用して、ソフトウェア開発に要する時間とコストを削減することが可能と考えられる。従って、地層処分ヴァーチャル・エンジニアリング・システムのような、試行錯誤的技術開発を伴う大規模システム開発手法として有効と考えられる。何故ならば、ソフトウェアの部品化・標準化による再利用の推進にともなうシステム開発効率の向上が期待できるためである。

（2）性能評価シミュレーション・コード開発面からの課題

性能評価シミュレーション・コードの開発面から考えた場合、オブジェクト指向の方法論は多くの研究開発課題を含むものと考えられる。第一に、クラスという抽象化された情報のモデルに、従来のFORTRANに代表する数理モデル（手続きモデル）の開発手法が、どの程度適用可能であるかを評価しなければならない。

従来の手続きモデルにおいては、式は変数と定数から表現されるものとして与えられ、プログラミングにおいては、変数は名称付きの参照可能なメモリーを示し、定数はコンパイラがプログラム内部の適当な番地に配置した数値データとして扱われた。従って、式は変数と定数を格納するメモリーに対する演算操作命令に展開され、操作結果は左辺の変数領域に格納されるという計算制御手法であった。これは、演算とデータ（変数、定数）が分離して考えられた手法である。

手続き的操作は、このようにプログラム内部に宣言されたデータ領域と、それを参照して計算を進めていく順序だった式の記述表現による算法である。これに対しオブジェクト指向の方法論では、操作とデータを組み合わせたクラスという概念に添って、プログラミング・アルゴリズムを考えなければならない。

例えば、式のそのものをクラスとして考えることができる。関数F（ｘ，ｙ，ｚ，ｔ）は、ある物理量を与えるクラスとして考えることができる。ここに、ｘ、ｙ、ｚ、ｔはインスタンス変数である。即ち外部からメッセージとして与えられる。この関数F()の内部処理は、プライベートなメソッドとして考えた場合。F()はメッセージｘ，ｙ，ｚ，ｔにより一意に定まる物理量として扱うことができる。

このように、オブジェクト指向について、性能評価シミュレーション・コードを開発面から考えた場合、ソフトウェアの基盤から設計開発を行わなければならないことが言える。また、既存FORTRANコードとのインターフェィスを独自の技術開発課題となると予想される。

オブジェクト指向ソフトウェア開発に於ては、その初期における設計段階で全体構造を示すことができるため、開発プロジェクトの信頼性（reliability）も優れている。その方法によれば、システムを構成する主要部分がまず最初に形作られ、それから個々の部分が組み立てられていく。その主要部分は、抽象的な性質で定義される。

従って、全体の統合テストを下位レベルの詳細仕様を定める前に行うことができる。そのような手

法は、ソフトウェア開発手法の一つであるラピッド・プロトタイピングの一手法として好例である。システムを、主要なクラスとそのオブジェクトに分解してから、システムの動作を特徴づける重要なメソッドをコードとして組みこんでいくことにより、システムの各“部分”がどのように全体として動作するか判断することができる。従って、試作システムの開発にあたってこの方法を適用した。

2．2．2　クラス、オブジェクト、カプセル化

（1）クラス記述

クラス記述とは、そのクラスのインスタンスであるオブジェクトの動きの特性を示す性質と特色をすべて定義することである。

クラス定義のプライベート部では、構造体データ構造を定義する。また、そのクラスの中でしかアクセスできないメソッドのインターフェイスも定義する。それらのメソッドは、パブリックなメソッドの実現を支援するためにしばしば用いられる。

クラスのプライベート部は、クラスの内部でしか操作できないが、他のクラスのオブジェクトを含む場合もある。クラスのパブリック部では、さまざまな応用分野で、そのクラスを再利用できるようにメソッドのインターフェイスを定義する。そのメソッドは、そのクラスの外からクラスのオブジェクトへメッセージを送ることによって呼び出すことができる。

（2）カプセル化

カプセル化（encapasulation）とは、個々のソフトウェア・オブジェクトを定義する手法の一つであり、データをできるだけ局所に閉じ込めることである。カプセル化とは、次のことで特徴づけられる。

- **オブジェクトの内部ソフトウェアすべての有効範囲を定める明白な境界。**
- **オブジェクトが他のオブジェクトとどのように相互作用するかを記述するインター フェイス。**
- **ソフトウェア・オブジェクトの機能の詳細を示す保護された内部実現。その実現の詳細は、そのオブジェクトを定義するクラスの外からはアクセスできない。**

カプセル化の概念はクラス記述と関連がある。しかし、同時に問題解法のための様々な構成要素が、どのようにグループ化されているかということをより詳細に示すことに有効である。カプセル化の単位はオブジェクトであり、オブジェクトの性質はクラスの中で記述されている。そして、同じクラスに含まれるオブジェクトは共通の性質をもっている。したがって、クラスのカプセル化というよりも、オブジェクトのカプセル化という考え方が正確であると思われる。このようにカプセル化を定義すれば、クラスの各インスタンスは、独立したカプセルあるいは問題解法の要素を構成するよう記述できる。

２．２．３　サブクラス－継承とポリモーフィズム

前節でみたように、オブジェクト指向で問題解決のためには、オブジェクトを用いてデータをカプセル化し、オブジェクトをクラスのインスタンスと定義する。クラス階層構造には、サブクラスと呼ばれる他のクラスの下に位置するクラスがあり、特にＣ＋＋言語では導出クラスという。クラス階層構造の中では、サブクラスは一つのクラスの下でグループにまとめられていて、サブクラスはそのクラスの特殊な場合と考えられる。

一般に、クラス階層構造では下位のものほど特殊となり、上位に進むほど汎用となる。大部分のオブジェクト指向言語では、例えばクラスＰがサブクラスＳの親である場合、サブクラスＳのオブジェクトｓは、親クラスＰのオブジェクトｐを使えるところならばどこでも使える。このため、メッセージの共通集合（つまり演算子）を、クラスＰとクラスＳ両方のオブジェクトへ送ることができる。また、同じメッセージを親クラスとサブクラスの双方のオブジェクトへ送ることができることをポリモーフィズム（polymorphism）という。ポリモーフィズムの各オブジェクトは、所属するサブクラスに合った方法で共通のメッセージ形式に対応する。

例えば、メソッドprintはオブジェクトの状態に対応するデータ・フィールドを出力すると定義されているとしよう。ポリモーフィズムでは、同じメッセージprintをクラスとサブクラスのすべてのオブジェクトへ送れば、オブジェクトがprintにそれぞれ（おそらくサブクラスごとに異なるであろうが）対応する。例えば、異なったサブクラスのオブジェクトでは異なったデータ領域を出力するかもしれない。

異なったオブジェクトに対する同様の演算に対し、同一メッセージで対応できる点は、人間の問題解決思考に近い。実際に、整数、浮動点小数、文字、文字列、データ・レコードの印刷方法がいちいち異なるという方が不自然なのである。

以上のように、ポリモーフィズムは、オブジェクト指向の問題解法パラダイムの重要な特徴である。オブジェクト指向言語にもいろいろあり、サブクラスのオブジェクトが親クラスの特性を継承、追加、削除（オーバライド）する方法にも種々の方式が考えられている。たとえば、サブクラスが複数の親クラスをもつという多重継承を支援するオブジェクト指向言語もある。

２．２．４　プログラミング

オブジェクト指向プログラミングでの課題は、ソフトウェアをどのようにクラス分解し、既存のソフトウェア・システムにどのように機能を追加し、いかに適切なタイプとサブタイプのクラス階層構造を構成するかということである。

（１）ソフトウェアをクラスに分割する

手続き型プログラミングに経験の深い人は、オブジェクト指向プログラミングでプロセスをクラスとしてモデル化することは不自然と思うかもしれない。たとえば、学生の名前や成績を記憶し、修正、削除、学生の名前と成績のレポートを印刷するための出席簿データベース・プログラムを書く場合、普通は、まずデータ型rollbookを定義し、その基本データ型rollbookに付随する演算として挿入、削除、修正、印刷などのメソッドを定義する。

しかし、挿入、削除、修正などの操作は、rollbookという型よりむしろ、editというプロセスに付随

すると考える方が妥当と思われる。したがって、プロセスeditをクラスとして宣言すればよい。そうすれば、このクラスeditは、データベース情報をオンラインで対話型形式で編集するような応用に再利用できる。クラスrollbookは、名前や成績の挿入、削除、修正を行うクラスeditのオブジェクトを含むようにする。

（2）既存のソフトウェア・システムに機能を追加する

オブジェクト指向プログラムでは、関数やプロシジャはそれぞれクラスに付随している。そのため、既存システムに新しい機能を追加する場合、設計者またはプログラマは、既存クラスに新たなプロシジャを追加するか、新しいクラスを作成するか、サブクラスを作成するか選択しなければならない。

新しい機能をオブジェクト指向システムのどこに追加すべきかを決める場合、再利用可能性が重要な判断規準である。追加した機能を既存クラスのオブジェクトでも再利用できるようにするならば、関数、またはプロシジャに基づく新しいクラスを作成すればよい。この新しい関数を実現するために、既存の型の内部の詳細にアクセスしなければならないならば、既存クラスに追加メソッドとして新しい関数を付け加えるべきである。新しい関数が与えられたクラスの既存の関数（メソッド）の修正の形ならば、サブクラスをつくり、このサブクラスに関数を追加して親クラスの同じメソッドを無視すればよい。

（3）タイプ（型）とサブタイプの階層構造

クラスとサブクラスの作成方法の一つは、トップダウン（top-down）式にデータ型を分解（decomposition）する方法で、この場合、プログラマはシステムの主要データ要素をクラスと考えてモデル化する。最上位のデータ要素（クラス）はより特殊なサブクラスに分割され、このプロセスはクラスとオブジェクトの階層構造が構築されるまで続く。

例えば、システムの大きなクラスとしてautomobileを考えてみよう。クラスautomobileはサブクラスengine、transmission、brakes、drive_train、exhaust_system、suspensionなどに分割される。

次にサブクラスengineは、ignition、fuel_injection、starterなどの要素に分割され、他のサブクラスも同様に分割される。

さらに、サブクラスignitionは、spark_plugsとsolenoidに分割される。

このとき、クラスとサブクラスはおのおの適切な演算（メソッド）の集合を定義しなければならない。一般に、トップダウン式のデータ型の分解では、階層の頂点のクラスは、その階層の下位のクラスすべてに共通な特性をメソッドの形で有する。

クラスとサブクラスのもう一つの作成方法は、ボトムアップ式分解（bottom_updecom-position）で、この場合、サブクラスを拡張して親クラスを一つつくるということを繰り返してより大きなクラスを構成していく。

ボトムアップ式分解の例として、例えば、GUIの開発においては、親クラスにscreen_windowをもつものと考えよう。クラスscreen_windowは、スクリーン・ウィンドウ全体の最小限の特徴を有する。一連のサブクラスは、基本となるクラスscreen_windowにさらに機能を加えたもので、前景と背景の色、ウィンドウの境界線の形状などを定義するサブクラスのオブジェクトで構成される。

２．３　Ｃ＋＋言語によるモデル表現

C＋＋は、Cプログラミング言語をもとに開発したもので、非常に少ない例外を除けば、Cをその部分集合として保持している。その基本言語であるC＋＋の部分集合Cは、その型、演算および文と対象となるコンピュータとの間に密接な関係があり、それが数、文字およびアドレスを直接扱えるように設計している。

空き領域の操作newおよびdeleteを除いて、個々のC＋＋の式と文は、典型的には隠されている実行時のサブルーチンによって支援する必要がない。C＋＋は、Cと同じ関数呼び出し、および返りの手順を使っている。このような比較的効率のよい機構さえ高すぎるという場合には、C＋＋関数はインラインに代入でき、実行時のオーバヘッドなしに関数という記法上の便利さを享受することができる。

Cの元来の目的の一つは、最も要求がきびしいシステムプログラミングの仕事において、アセンブリコードを置き換えることであった。CとC＋＋の違いは、主として型と構造をどれだけ強調するかの程度である。Cは表現力があり自由さがある。C＋＋は、もっと表現力がある。しかし、そのような表現力を達成するために、プログラマはオブジェクトの型についてもっと注意を払わなければならない。

オブジェクトの型を知っていると、プログラマが非常に細かく操作を指定しなければならない場合でも、コンパイラが式を正しく扱うことができる。また、オブジェクトの型を知っていると、コンパイラはテストのときまで居残ってしまいそうなエラーの検出も可能となる。型システムを使って、関数の引数をチェックすること、偶然による障害からデータを保護すること、新しい型を提供すること、新しい演算子を提供すること等を行っても、それ自身実行時間のオーバヘッドを増加することは、Cが設計されて以来プログラムの規模が増大するようになってきた事実を反映している。

1000行以下の小規模のプログラムは、よい形式の規則をすべて破っているような場合でもなんとか無理やり動かすことが可能である。それより大規模なプログラムは、それほど簡単ではない。10,000行のプログラムの構造が悪いと、古いエラーが除去されるのと同じ速さで新しいエラーが生じてしまう。C＋＋は、大規模なプログラムを合理的な構造にすることが可能となるように設計されているので、単一のプログラマが25,000行のコードを扱うことも不自然ではないだろう。

多くの大規模なプログラムが存在するが、一般に動いているものは、多くのほとんど独立した部品からできあがっており、それらの部品は、それぞれ前に述べた限界よりもずっと下のものであることがわかる。当然、プログラムを保守していく難しさは応用の複雑さによるのであり、プログラムのテキスト行数に単純に依存するのではない。

一般にプログラムは、すべてのコードの切片がよく構造化され、機械に独立で読みやすくなっているわけではない。C＋＋は、安全性や理解のしやすさなどを無視して、ハードウエアの機能を直接操作できるように考えている。これはまた、そのようなコードを優雅で安全なインターフェイスの背後に隠してしまう機能をもっている。

２．３．１　哲学的問題

プログラミング言語は、２つの関連した目的に対して役立っている。即ち、プログラマが実行すべき動作を指定する道具を提供すること、およびプログラマが何が出来るのかを考えるときに使う概念を提供することである。

第一の目的の為には、理想的には“機械に近い”言語が必要になる。その結果、機械のすべての重要な様相を、プログラマにとって合理的で明らかであるような方法で、簡単かつ効率よく処理するべきである。C言語は、主としてこの点を考慮して設計した。第二の目的は、理想的には“解決しようとする問題に密接している”言語を必要とし、その結果、解の概念を直接かつ簡潔に記述することができることが要求される。

C＋＋を生み出した機能は、主としてこの点を考慮したものである。再プログラムしたりする言語と、問題や解を想像するための言語との間には関係がある。それ故、プログラマのエラーを除去することを目的として言語の機能を制限することは危険である。

また、自然言語の場合と同じように、少なくともバイリンガルであることは非常に有利である。言語は、プログラマに概念上のツールの集合を提供している。即ち、これらが仕事に不適切であれば、それらは見捨てるだけである。たとえば、ポインタの概念を制限しても、それはプログラマにベクトルと整数演算を使って、構造体やポインタを実現するように強制するだけである。よい設計とエラーの消滅とは、単に言語の機能だけでは保証できない。型システムは、単純でない仕事にとって特に有用であるに違いないクラスの概念は、強力な概念ツールである。

２．３．２　C＋＋におけるプログラミングの特徴

理想的には、プログラム設計の仕事は２つの段階に分けて行う。即ち、最初に問題を明確に理解することから始める。次に、解に関係する鍵（キー）となる概念を規定する。最後に、その解をプログラムとして記述する。問題や解の概念の詳細は、しばしばそれをプログラムとして記述することにより明確に理解される。それがプログラミング言語の選択の問題である。

たいていの応用では、プログラムの中で基本型の一つとして、あるいは付随した静的データなしの関数の一つとして簡単に表現できないような概念がある。そのような概念を与えたとき、プログラムの中でそれを表現するクラスを宣言してみる。

クラスは型である。即ち、そのクラスのオブジェクトがどのように振る舞うのか、たとえばそれをどのように生成するのか、どのように操作できるのか、そしてどのように消去するのかを指定する。クラスは、また、オブジェクトをどのように表現するのかも指定する。

しかし、プログラム設計の初期の段階では、それは主要な問題ではない。また、そうしてはならない。よいプログラムを作成する鍵は、それぞれのクラスが単一の概念を明確に表すように設計することである。それは、しばしばプログラマが次のような疑問に焦点を絞っていなければならないということであろう。即ち、このクラスのオブジェクトをいかに生成するか。このクラスのオブジェクトはコピーできるのか、消去できるのか。そのようなオブジェクトにどんな操作が可能なのか。このような疑問によい解答が得られないのならば、まず第一にその概念は、“明確な”ものではなかったであろう。

次に、“すぐに問題をコード化する”ことを始めるかわりに、もうすこし問題を考え、解の考察を

してみることがプログラム開発姿勢として必要である。扱うのがもっとも容易である概念は、いろいろな種類の数、集合、幾何的図形など伝統的な数学の形式に沿ったものである。実際、そのような概念を表すクラスの標準ライブラリがシステム環境にあるべきである。プログラムの中でクラス間の関係を組織化すること、即ち、解の中で使っている異なった概念の間の正確な関係を決めることは、個々のクラスを配置することより難しいことが多いと思われる。従って、設計に於ては、すべてのクラス（概念）が、すべて他のどれかに依存するような混乱した結果になることは避けるべきである。

例えば、AとBと２つのクラスを考えてみよう。“AがBからの関数を呼び出す”、“AはBのものを生成する”、“Aは、Bのメンバを含んでいる”などの関係は、めったに大きな問題にはならない。また、“AがBからのデータを使っている”という関係は、ほとんど除去できる（単純に公開したデータのメンバは使わない）ことがほとんどである。

問題となるのは、“AはBのひとつであり、そして．．．”というように自然に表現される関係が大部分である。複雑さを制御する最も強力な知的ツールのひとつは、階層的な順序づけをすることである。即ち、最も一般的な概念を根に設定した木構造をつくって、関連した概念を組織することである。C＋＋では、導出クラスがそのような構造を表している。

プログラムは、しばしば木構造の集合として組織化できる。即ち、プログラマは基本クラスをいくつか指定し、それらがそれぞれ導出クラスの集合をもっている。仮想関数は、しばしば概念の最も一般的な組（基本クラス）に対する演算の集合を定義するために利用できる。必要ならば、これらの演算の解釈は、特定のクラス（導出クラス）のために再定義できる。ただし、この組織化は当然限界をもっている。

特に、概念の集合は、ときには有向非循環グラフとして組織化した方がよく、そこでは、一つの概念は一つ以上の他の概念に直接依存できる。たとえば、“AはBのひとつであり、またCのひとつであり、そして．．．”という場合である。C＋＋では、これを直接支援することはしないが、このような関係も表現できる。

場合により、有向非循環グラフでもプログラムの概念を組織化するのには不十分にみえることがある。ある概念は、本質的に相互依存に思える。相互依存のクラスが小さいもので理解が容易にできるならば、循環的な依存関係は問題にする必要はない。フレンドクラスの考え方は、C＋＋における相互依存型のクラスの集合を表すために使うことができる。

プログラムの概念を、木でもなく有向循環グラフでもない一般的なグラフにのみ組織化でき、また相互依存性を局所化できないとするならば、どんなプログラミング言語も有用とは言えないであろう。基本概念の間になにか簡単に述べることができる関係を見つけ出さなければ、そのプログラムは管理不可能になってしまうであろう。多くのプログラミングは、基本型、データ構造、簡単な関数、標準ライブラリからのいくつかのクラスだけを使って単純で明確に実施できる。新しい型を定義するための機構全体は、本当に必要が生じる場合以外は使うべきではない。

２．３．３　プログラミング原理

プログラムを開発する行為は、ある問題に対する解のアイデアの具体的な表現をつくっていることと考えられる。そのとき、プログラムの構造が、これらのアイデアを直接反映するようにすることが大切である。それには、以下の手法が基本となる。

- 対称概念が異なったアイデアと考えることができるならばそれをクラスにせよ。
- 対称概念が異なったものと考えることができるならば、それをあるクラスのオブジェクトとせよ。
- ２つのクラスがなにか重要なことを共通にもっているのならば、それを基本クラスとせよ。

行列や複素数のような数学的な実体、あるいは結合リストのような低レベルの型を実現するのではないクラスの定義をする場合には以下の通りにするべきである。

a) 全域的データを使わないようにする。
b) 全域的なメンバでない関数を使わないようにする。
c) 公開されているデータのメンバを使わないようにする。
d) a) 、b) およびc) を避けるため以外に、フレンドを使わないようにする。
e) ほかのオブジェクトのデータのメンバに直援アクセスしないようにする。
f) クラスに“型フィールド”を置かず、仮想関数を使うようにする。
g) 意味のある最過化以外にはインライン関数を使わないようにする。

Ｃ＋＋は、Ｃを基に開発された言語である。Ｃ＋＋は、Ｃの機能や柔軟性をそのまま備えており、ハードウェアとソフトウェアのインターフェイスや、低レベルのシステム・プログラミングの処理も可能である。また、Ｃの効率の良さ、経済性、表現力も備えている。さらに重要なことは、Ｃ＋＋がオブジェクト指向プログラミング機能や、高級な問題抽象化を行えることである。Ｃ＋＋は混声言語である。問題を純粋にオブジェクト指向的に解きたい場合は、オブジェクト指向パラダイムを活用すればよい。一方、Ｃ＋＋は、Ｃにいくつかの概念を付加した手続き型言語でもある。Ｃ＋＋のソフトウェアは、新しいオブジェクト指向パラダイムにも、手続き型プログラミング・パラダイムにも対応できる。

Ｃ言語の場合、プログラマは基本システムを直接制御できるので、厳密な意味でのボトムアップ方式で、つまり、まずシステムの下位構造、それから上位構造という順序で進めていく。Ｃ＋＋ではこの過程が部分的に逆になる。つまり、トップダウン式とボトムアップ式が共存する。抽象オブジェクト（abstract object）の必要性が認識され、他の抽象オブジェクトとの相互作用が定義されたとき、その抽象オブジェクト内にカプセル化されたデータ構造と、アルゴリズムの初期設計を始めるのが望ましい。

ラピッド・プロトタイピングとは、あるクラスのシステム内での他のクラスとの関連を調べられるように、そのクラスの迅速な開発を行うことである。本研究の為の試作プログラムの内、GUIはこの手法を用いて開発した。この場合、各クラスを洗練かつ完成した形に実現するのは、システムの全構造（クラスの集合とそれらの相互関係）が定まったあとにすべきである。

オブジェクト指向言語では、拡張可能性も重要である。オブジェクト指向ソフトウェアの設計者は、問題の構成要素およびその演算を、解を実現する言語へできるだけ自然に移そうとする。大部分の高

級プログラミング言語では、内蔵された言語の構成概念や性質に限度があるためこれは難しい。

たとえば、FORTRAN言語によるプログラミングでは、プログラマがレコード構造の集合を表現しなければならない場合、スカラー・データの多重配列を用いる。その多重配列は、共通インデクスで連結されている。そのため、たとえば各レコードを表現するためにデータが５つあれば、レコードの配列を表現するために定義された、５つの配列の同じインデクス位置にアクセスすることにより、それらのデータが得られる。

それよりも、抽象オブジェクトdata_baseを定義して、そのオブジェクト内の情報を挿入、削除、アクセス、表示するメッセージを受け取るようにするべきである。

この場合、data_baseオブジェクトの処理は、人間が行うような自然な形でなされる。たとえば、data_baseオブジェクトに新しいレコードを挿入する場合には、単に以下のようにすればよい。

```
data_bese. insert (data)
```

data_baseとは、適切に宣言されたオブジェクトであり、関数insertとは、オブジェクトdata_baseを支援するためにクラス内で定義されたメソッドであり、パラメータdataとはオブジェクトに追加されるデータである。

このように、地層処分ヴァーチャル・エンジニアリング・システムの開発に於ては、各部の機能をオブジェク指向プログラミングにより開発する。

例えば、グラフィック・ユーザー・インターフェイスの開発は、OSF/MotifWidgetを利用してＣ＋＋言語のオブジェクト指向プログラミング機能を用いて開発している。また、ネットワーク・レベルで並列に動作するパラメータ・ディスプレイはWindows95のリソースを利用してＣ＋＋言語のオブジェクト指向プログラミング機能を用いて開発している。それらの詳細は別紙を参照のこと。

２．４　Smalltalk言語によるモデル表現

論理型プログラミングとか関数型プログラミングというのは、ソフトウェアに関する議論というより、計算メカニズムの基本に関する議論であると思われる。それに対して、オブジェクト指向は、一見、計算メカニズムを云々しているように見えるが、従来のソフトウェア構成法の概念と根本的に異なる新しいソフトウェア構成法の概念を与えている。

オブジェクト指向プログラミングという概念が、比較的世の中に広く知られだしたのは、ゼロックス社のパロアルト中央研究所（PARC）で開発された言語Smalltalk-80からである。Smalltalkは、1970年代初頭、Alan Kayが、理想のパーソナルコンピュータとして頭に描いたDynabook（携帯型、高精度ディスプレイ付き、視聴覚入出力装置付き、ネットワーク内ワークステーション）のための基礎言語である。

２．４．１　言語の特徴

さて、Smalhalkはオブジェクト指向プログラミングという、プログラミング・パラダイムを徹底的に追求した言語であると言われているが、ここで“言語”という言葉の意味が、従来のプログラミング言語の“言語”の意味から大きく飛躍していることにも注意する必要がある。

われわれがプログラミング言語を示す場合は、計算機にプログラマの意志を伝達するための、いわば文法構造化された文字列を指す。しかし、Smalltalkは人間が計算機と交信するための可能な限り広い情報伝達媒体として言語としてとらえられている。Alan Kayが理想として描いた、音や絵を縦横に駆使する交信にはまだ程遠いが、“言語”のマニュアルの中に、高精細ビットマップ・ディスプレィ上での“操作”が混入して記述されているのはなかなか興味深い。すなわち、Smalltalkは文法書が単独で存在するような言語ではなく、画面を通じて計算機と人間が対話する場（あるいは環境）そのものとして開発されている。

２．４．２　抽象化

プログラミングの歴史における最大の発見の１つはサブルーチンであると言われている。サブルーチンの概念と、それを実現するハードウェア機構の発明により、同じことを何度も書かなくて済むようになった。それぞれのサブルーチンを名前で呼ぶようになったとき、プログラマは、ある一連の動作を総称する抽象化を行なったといえる。

あるレジスタから別のレジスタの値を引くといった、低レベルの概念だけから平方根を求めるという高度な手順を構成しえたわけである。更に、平方根のサブルーチンを使って、二次方程式を解くサブルーチンを作る。このように、サブルーチンの上にサブルーチンを重ねることによって、次第に高い抽象化が得られる。

今日的な言葉を使っていえば、ふつうのプログラミング言語は、ごく単純な基本演算から出発して何段階にも抽象化を重ねる手段を与えている。抽象化の各段階は、手続きとか関数とか呼ばれ、それらのもう一段高い抽象化はパッケージと呼ばれる。

ところで、もともと基礎にある基本演算自身が抽象的概念である。すなわち、動作は目に見えたり、手で触れたりするようなものではない。これをさらに抽象化していくという過程は、知的作業になる。

人間の言葉における抽象化は、むしろ“もの”の抽象化のほうが多い。同様に、ものあるいは物理的イメージを基礎とした抽象化の方法を、プログラミングの世界でも実現することを試みたプログラミングの方法論が、オブジェクト指向プログラミングであると考えられる。

２．４．３　メッセージ交換による計算

従来の言語の基本演算をオブジェクト指向を主体として考えるとどうなるであろうか。たとえば、3+4は、ふつう＋という演算が主体で、与えられた２つのデータ（ここでの用語に従えば、“もの（オブジェクト）”）は従である。与えられたデータをどう処理するか演算が決める。これが、従来の考え方である。このような考え方では、＋は操作（プログラム）であり、3や4はデータとして操作とは分離される。

それに対し、オブジェクト指向では、3（オブジェクト）に+4が送り込まれたと考える。たとえば次のように書く、3+4*5結果は35である。掛け算が足し算に優先するという規則はSmalhalkでは使わない。電卓と同じように、書いた順に計算を進めていく。4*5を先に行いたいのであれば、3+（4*5）と書く。

Smalltalkでは、+4や*5を総称してメッセージといい、メッセージを受け取って実際の仕事をするものを受け手（レシーバ）という。また、操作は＋のほかにもたくさんの種類があるのでセレクタ

(selector) と呼び、操作の付属物は、ふつうの関数の用語にならって引数 (argument) という。

Smalltalkでは、ほとんどすべての計算をオブジェクトがメッセージを受け取って返事をすることと考える。このような表現をメッセージ式と呼ぶ。例えば以下の通である。

```
単項メッセージ式の例
    aPoint x：点aPointのx座標を聞く
    nameOfPerson size：文字列nameOfPersonの長さを聞く
キーワードメッセージ式の例
    ages at:'Alan Kay'：agesと言う表から、Alan Kayの年齢を探索する
    Point x: 100 y: 100：x座標が100、y座標が100に点を作図する
```

Smalltalkの式は、メッセージ式だけでなく、代入式変数名←メッセージ式がある。これも、“変数名”という“オブジェクト”が←をセレクタとする、メッセージを受け取っている形と考えられないことはないが、Smalhalkではそう考えない。実際、xという変数があったとしてX←X+4と書いた最初のx (変数名) と２番目のx (数値) で、xの意味が異なるからである。Smalltalkの変数は、最初の英文字が大文字か小文字かに応じて大域的か局所的かになる。たとえば、x、aPoint、originDeltaPointなどは局所変数、EnterpriseSpaceShip、Pointなどは大域変数である。

四則演算などは、二項演算の形をしている。従って、上記の式を二項メッセージ式と呼ぶ。二項メッセージ式のセレクタには特殊文字が使われる。Smalltalkには、このほか単項メッセージ式とキーワードメッセージ式がある。単項メッセージ式は引数のないメッセージ伝達を表わし、キーワードメッセージ式は引数のあるメッセージ伝達を表わす。また、単項、二項とも、重なれば左側が優先する

２．４．４　メッセージ式による制御構造（ブロック）

オブジェクト指向では、基本的な演算はすべて受け手がメッセージを受け取り、それに対して返事をするという形で表現できる。つまり、オブジェクト主体で考えることができる。

また、論理演算においても真理値そのものを“オブジェクト”と考えると、真と偽はそれぞれtrue、falseと表わされる。

実はtrueもfalseも擬変数というタイプの変数で、それの値が本当の“真”と本当の“偽”である。擬変数は変数であるが、代入文では値を変えられない。擬変数の例として、ほかにnil、self、superがある。次に、プログラム自身も“オブジェクト”と考える。すぐに実行しないプログラムを、“オブジェクト”として取り扱うために角カッコを使ってくくることができる。これをブロック (Block) という。例えば、以下のとおりである。

```
[cycleCount <- cycleCount + 1]
[index <- index + 1.  arrayX at: index put: 0]
```

２番目の例のピリオドは、文を区切っている。ブロックを実行するためには、以下のように書く。

```
[cycleCount <- cycleCount + 1] value
```

つまり、valueという単項メッセージを送ればよい。trueはifTrue：aBlockというメッセージを受け取ると、aBlockにvalueを送る。falseはifFalse：aBlockを受け取ると、aBlockにvalueを送る。両者を重ねたifTrue：ifFalse：というセレクタもある。これはif - then - elseに相当する。

このようにして、通常のプログラミング言語がもっているような基本機能を、ほとんどオブジェクトがメッセージを受け取って返事を返すという統一的な原理で構成することが、オブジェクト指向の根本的な考え方である。

２．４．５　モジュール化

これまでは、Smalltalk-80におけるオブジェクト指向プログラミングのうち、メッセージ伝達による計算機構を主として文法的側面から調べた。その結果、3+4のような算術式が、従来とはまったく異なった見方で扱われることがわかった。

プログラム設計では、全体をいくつかのモジュールに細分化し、そのモジュールをまた細分化し、といったソフトウェア構成法が一般的であるが、その際問題になるのがモジュール間のインタフェイスである。

従来のプログラム言語では、サブルーチンや手続きがモジュールの単位であった。前述したように、これは基本演算の上に積み上げられた抽象モジュールである。この抽象化は、扱うデータ構造と無関係に行なわれた。従って、例えば在庫の出荷待ち行列を表わすのに、配列を使ったか、ポインタでつながったリストを使ったかが、最後まで見えてしまう。

プログラムの字面上では、抽象化が進むにつれ、そういう構造は見えにくくなるが、見えていることに変わりはない。いずれにせよ、このような場合、データ構造が抽象化から取り残されているのは明らかである。

そのようなプログラムを改良する場合は、結局、底辺まで調べざるを得ない状況に陥ることが一般的である。従って、どのレベルまで深く調べればよいのか、あらかじめ見当がつかない演算の抽象化とともに、それに付随したデータ構造をも抽象化のカプセルの中に取り込もうとしたのが、いわゆる抽象データ型プログラミングである。

抽象データ型プログラミングでは、手続きとデータがパックになるので、より高い抽象化の段階でデータ構造を直接（配列かリストか？）で触ることが不可能になる。抽象データ型プログラミングは、従来のプログラミングとオブジェクト指向プログラミングへの中間点であると考えられる。すなわち、オブジェクト指向では、データ構造の抽象化が主となり、演算の抽象化が従になる。

２．４．６　クラスとインスタンス

Smalltalk-80が用意しているデータ構造の抽象化の一例を図２．４－１に示した。図の中で大文字で始まる名前は、それぞれ抽象的なデータ構造の概念を表わす（値を変えない）大域変数である。これらの抽象概念はクラス（class）と呼ばれる。図２．４－１に示したクラスは、抽象度の高いものほど上に位置した木構造をなす。クラスCollectionは、とにかく、ものの集まり（集合という意味ではない）で、ここでは一番高い抽象概念である。ものの集まりの中に順序があるかどうかで具体化が少し進む。

順序があるものが、SequenceableCollectionである。順序のないものは、要素が（検索用）キーで参照できるかどうかでまた二技に分かれる。こうして、下に行くに従って、“もの”としての具体的イメ

ージが明らかになり、一番下にはArray（配列）とかString（文字列）が現われてくる。確かにStringは順序のついた文字の集まりの一種ではある。

あるクラスのすぐ下にあるクラス、つまり具体化の進んだクラスを上のクラスのサブクラス（下位クラス：subclass）と呼び、逆に上のクラスを下のクラスのスーパークラス（上位クラス：superclass）と呼ぶ。われわれが概念を抽象化するのと同様に、スーパークラスは、いくつかの（サブ）クラスに共通な性質を抜きとって構成した一段高い抽象概念になっている。

抽象概念といっても、言語の中の１つの概念であるので、クラスはある種の“記述”のかたまりであると考えるべきである。

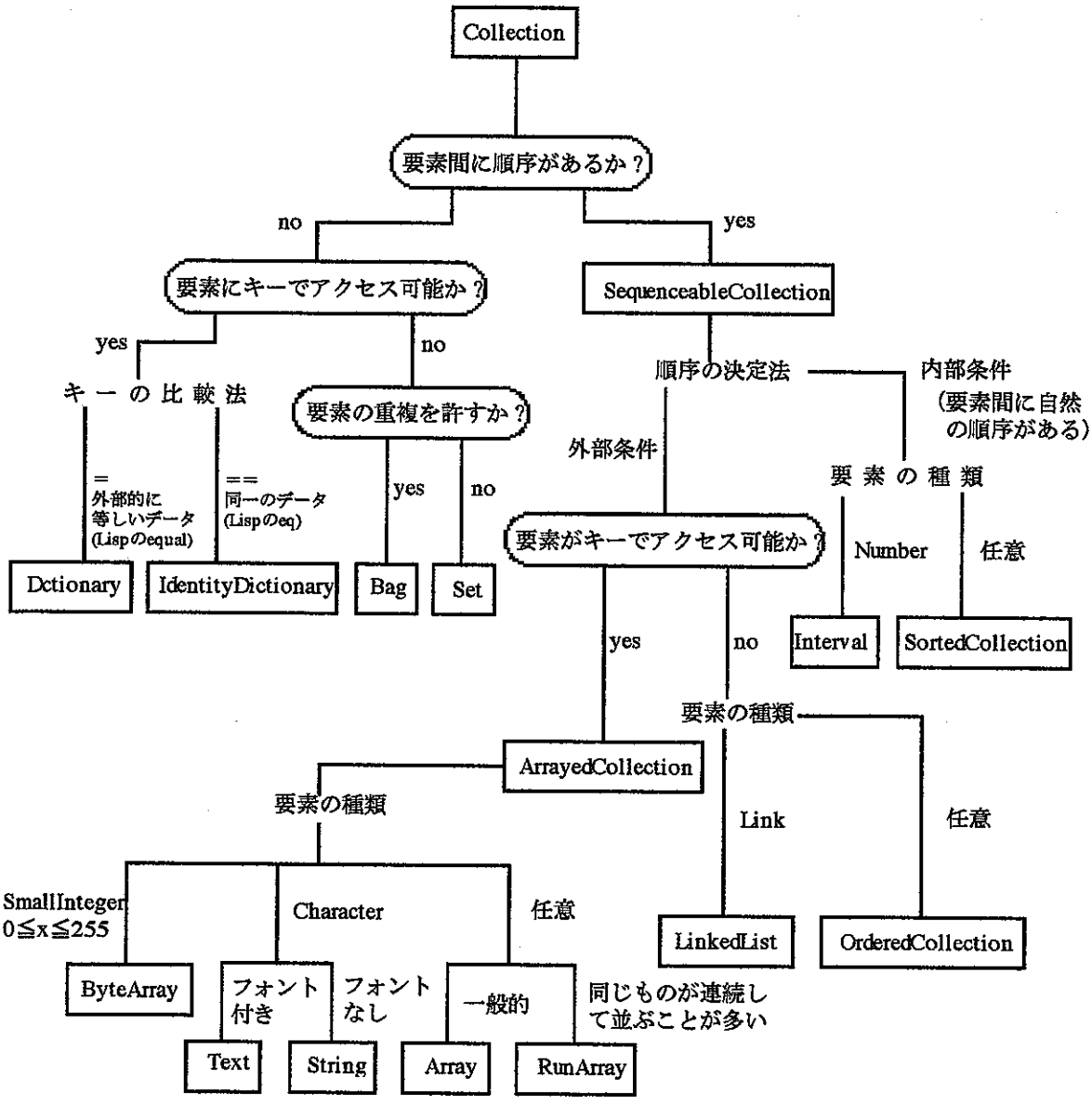

図２．４－１　サブクラス構造の例

クラスは定義どおり、共通な性質をもったものの集まりを表わす。Smalhalkでは、ものの性質を、メッセージに対する返答法で表わす。つまり、メッセージという、いわば外部からの干渉（観測）に

対して、どう反応するかが、ものの性質であるという考え方である。また、どういうメッセージに対して反応するかについて定めた概念をインタフェイスと呼ぶ。

次に、反応すると決めたメッセージに対して、具体的にどういう反応をするか。これがメソッド（method）である。従って、インタフェイスにあるメッセージそれぞれに対してメソッドが存在する。メソッドは一般の手続きとほぼ同様と考えて良い。以上からわかるように、クラスは具体的な個々のものに対する型紙（あるいは設計図）のようなものであると考えるべきである。クラスに属する（あるいはクラスという設計図から生み出された）具体的な個々のものをインスタンス（instance）と呼ぶ。たとえば、Stringというクラスのインスタンスは、以下の生成された文字列である。

```
'Waste Management', 'People', 'Groundwater Flow', '.....'
```

どれも文字列が当然もつべき性質は共通にもっている。例えば、書式順序を尋ねるメッセージ“<=aString”は、どのインスタンスも受け取って反応することができる。ここで、aStringは、クラスStringの１つのインスタンスを値としてもつ変数である。aStringという変数名を使ったからそうなるというわけではない。Smalltalkでは、定数として書かれた以外のものにはデータ型がない。データ型、つまりどのクラスのインスタンスであるかは、常に実行時にチェックされる。

しかし、クラスという記述だけでは記述しきれないインスタンス固有の性質がある。多くのクラスは、インスタンスごとに差をつけるために変数を用意している。これをインスタンス変数（instance variable）と呼ぶ。クラスは、インスタンスに何がしかのインスタンス変数が付随しているということを指定するが、インスタンス変数に具体的にどんな値が入っているかは知らない。知っているのはインスタンスである。

インスタンス変数には名前付きインスタンス変数と、添字型インスタンス変数の２種類がある。名前付きインスタンス変数は、ふつうの言語と同じで、名前で参照できる変数である。添字型インスタンス変数は、名前はないが、何番目という指定でアクセスできる。１つのインスタンスに、名前付き変数と添字型変数が同居していてもかまわない。添字型は無名である。添字型変数とは、１次元の配列のことである。以上のことから、クラスStringのインスタンスは、クラスCharacterのインスタンスを値としてもつ添字型変数をもっているらしいことが伺える。

オブジェクト指向では、クラスまたはそのインスタンスを外側から見ると、インタフェイスしか見えない。あとは全部ブラックボックスである。インスタンス変数やメソッドは隠れている。こういう、いわば内部機構を外から見えなくすることを情報隠蔽と呼ぶ。情報隠蔽はモジュール化の基本的手段である。

２．５　数理モデルの考え方

前項からオブジェクト・シミュレーションを実現するためには、オブジェクト指向ソフトウェア開発のためのモデル開発と、クラス設計が最も重要であると考えられた。オブジェクト指向を用いて、物理工学的モデルをシミュレートする場合、従来の解析的手法（支配方程式、基礎方程式の導出）、数値解析（離散方程式の導出）プログラミングとは、かなり異なった手法（思考論理）が要求される。

特に重視しなければならない点は、オブジェクト指向によるモデル組立は、数理モデルの組立にま

で影響しかねないと考えられることである。従来のシミュレーション・モデルは、自然界の現象を実験室等の理想化された環境で再現、観測し、観測結果から現象の振る舞いの方程式（以下、数理モデルと記す）を導きだす手法を一般的としていた。このようにして導かれた数理モデルは、数理モデル全体が対象とする現象を記述するものであり、基本的には分割可能ではない。つまり、数理モデルを分割することにより、数理モデルとしての意味を失う。

例えば、ある連立方程式が力学系モデルとして与えられた場合、モデル式を分割しても、個別に意味のあるアルゴリズムを引き出すことはできない。従来の実験と数値解析的手法は、解析的な数理モデル開発を目標としたため、得られた数理モデルは完成したモデルとして考えるべきである。このように、数理モデルを実験等から得られたデータをもとに解析的に導いて検証した場合、得られた数理モデルは、それを導いた現象過程の範囲で数値計算による初期値、境界値問題に解を与えることができる。それは、従来一般に取られてきた数理モデルの開発手法である。そのような手法の限界は、得られたモデルの記述する範囲が、理想化された系での観測（計測）を基準にした、いわば閉じた理想系でのモデルのため、自然界のように外部からの事象影響等を考慮した記述は一般になされないと言うことである。言い換えれば、得られた数理モデルが利用できる範囲は、当該モデル単体で計算して得られる数値の範囲であり、例え、複数の数理モデルが解析的に得られたとしても、それらは、個々に独立しており、相互の連成もしくは協調は記述されていないということである。

そのような数理モデルの開発は、設計要素の個々の確認には有効であって、システム全体の振る舞いを評価するために組み合わせて利用することは基本的にできないと考えられる。なぜならば、個々の数理モデルは、それ自身に連成条件が記述されていないためである。

オブジェクト指向を、将来の地層処分性能評価モデルの組立戦略とすることは、数理モデル開発に分割と再編成の可能性が期待できることが最大の魅力である。また、総合的な多方面からのシナリオに沿った性能評価シミュレーションと言う考え方は、恐らく分割と再編成、そしてメッセージ交換を伴う、並列的なオブジェクト指向の数理モデルを構築するめどが立たなければ、理想だけにとどまるであろう。幸い、本年度の研究では、図１．４－２０の例に示すように、破砕帯空壁を通過する核種に見立てた微小粒子のシミュレーションを試行することができた。このモデルは、非常に単純な微小粒子と、１本の運動力学を記述するメンバー関数からなっているが、動画化されたシミュレーションの見かけはリアルである。恐らく、化学特性を示す各関数を与えることにより、破砕帯空壁中を浸透する、地下水の地球化学的性質の数値実験を行うことができるであろう。このような、素過程の上に組み立てられる数値実験を試行錯誤的に行うことにより、分配係数の利用に関わる議論等多くの性能評価に関わる問題研究を、より不確かさの無い条件の基で行うことができるであろう。

オブジェクト指向におけるポリモーフィズムの概念に従えば、モデル化を抽象的な段階から開始できる。それはまた、試行錯誤的モデル開発作業効率を向上させると考えられる。そのような観点から、本章では、オブジェクト試行によるモデル開発の方法論を、簡単な力学系モデルを例に明らかにする。

それにより、考えている問題や、その解決プロセスの表現が、オブジェクトとその間のメッセージパッシングによって素直に表現できることを明らかにする。オブジェクト指向型表現をNewton力学の

問題解決システムに導入した場合、以下のような利点が考えられる。

- Newton力学は、物理学の最も基本的な分野であり、今後の研究の基礎となり得る。
- 力学系内の個々の対象（たとえば、斜面、滑車、ブロックなど）をオブジェクトで、また、対象間の力の及ぼし合いを、メッセージパッシングによって表現することが比較的自然に行なえる。
- 熟達者が問題を解くときによく行なうような、いくつかの対象をまとめて１つの仮想的な対象とみなすという思考方法が、その仮想的な対象を新しいオブジェクトとして作り出すことによって、自然な形でシミュレートできる。

さらにここでは、このようなシステムの応用として、Newton力学を理解するための研究システムについて合わせて検討する。オブジェクト指向型表現が、力学の問題の構造を自然に表現できるならば、研究者が自分で問題を作り、システムにその問題を解かせることによって、問題を「理解する」ことが比較的自然に行なえるようなシステムを開発できると思われる。

なぜならば、研究者の考える問題表現とシステム内部でのオブジェクト指向的表現が、ある程度似たものとみなせるからである。そこで、問題解決システムに、問題作成のためのインターフェイスを接続した、問題理解のためのシステムが考えられる。このようなシステムの可能性は、地層処分ヴァーチャル・エンジニアリング・システムの可能性を更に高めるものである。さらに、場合によっては、システムが問題を出して、研究者が考察・判断するいう形式も考えられる。それについては、ゲーミング・シミュレーションの話題のため次章にて考察する。

２．５．１　プログラムの例

典型的なNewton力学の問題として、図２．５－１に示すような問題を考える。図２．５－１に示すのように、滑車にブロックがさがっている。ブロックの重さは、それぞれ、b1が10kg、b2が15kg、b3が30kgとする。簡単にするために滑車の重さはゼロとする。このとき、ブロックb1の加速度を求めることを問題とする。

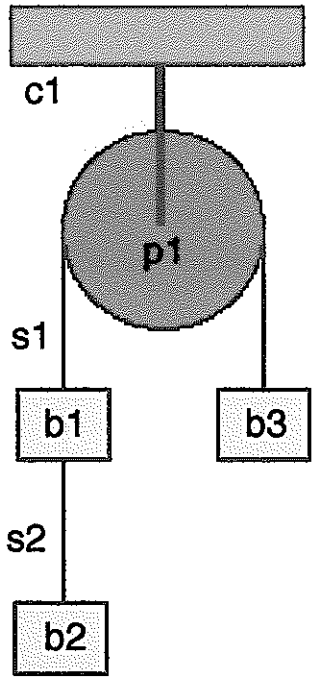

図２．５－１ 滑車のモデル

問題解決にあたり、滑車やブロックのような物理的対象を、それぞれオブジェクトと考えると都合がよい。たとえば、図２．５－２にブロックb1と滑車p1のオブジェクト表現が示してある。これらは、いわゆるインスタンスである。また、s1、s2はブロックb1に接した糸（string）、c1は天井（ceiling）であり、それぞれ別のオブジェクトとして表現する。さらに、contact、up、downなどは、物理的対象の空間的位置関係を表わしている。

```
(b1 (isa (block))
    (value (contact (s1) (s2)) (up (s1))
           (down (s2)) (mass (10.0))))
(p1 (isa (pulley))
    (value (contact (c1) (s1))
           (contact*axis (c1))
           (contact*disc (s1))
           (up (c1))
           (down (s1))))
```

図２．５－２　ブロックb1と滑車p1のオブジェクト表現

一方、滑車p1は一般的な“滑車”（pulley）の特殊ケースであり、一般の“滑車”は、クラスとして図２．５－３のように表わす。

```
(pulley (isa (class))
        (ako (object))
        (methods (key-ceil nil)
                 (key (obj))
                 (key-wall (obi))
                 (gravity nil)))
```

図２．５－３　滑車のクラス表現

このオブジェクトの中には、メソッドがLispの関数の形で書かれており、天井や壁など、他のオブジェクトとの位置関係などを計算するようになっている。

```
(monitor (isa (class)) (methods (start nil)))
(planner (isa (class))
         (methods (make nil)
                  (normal-plan (at p-at))
                  (one-force-plan nil)
                  (solve-tension-plan nil)
                  (solve-push-force-plan nil)))
(solver (isa (class))
        (methods (solve-relative-acceleration (rel))
                 (solve-relative-tension (rel))
                 (ready nil)
                 (make-kei nil)
                 (make-tension (s-name))
                 (make-push-force (o-lis))
                 (solve-goal nil)
                 (solve-one-variable nil)))
```

図２．５－４　主要なコントロール用オブジェクト

たとえば、図２．５－３で、key - ceil、key - wallは、それぞれpulleyに対してceiling、wallがどのような位置関係にあるかを計算する関数、gravityは重力の方向を与える関数、keyは引数objに束縛されたオブジェクトとの位置関係を計算する関数である。図２．５－３のように、クラスの中にメソッドを書けば、それらがそのクラスに対して適用されるメソッドとして、Lisp関数の形で自動的に生成される。ほかにも、力や張力、重力のような物理的概念を初め、“右”とか“左”、“上”、“下”のような位置関係の概念など、問題を解くのに必要な概念をオブジェクトの形で表現する。

```
monitor start
>>>>> please input goal object(s) <<<<<                              (a)
b1                                                                  (a)
>>>>> please input goal attribute <<<<<                             (a)
acceleration                                                        (a)
>>> all*objects --> (c1 p1 s1 b1 s2 b2 b3)                         (b)
>>> attention - - > (b1 b2 b3)                                      (c)
>>> partial*attention- - > ((b2 s2 b1 s1 b3))                       (d)
>>>>> Make : gravity-1 of b1                                       (e)
>>>>> Make : gravity-2 of b2                                       (e)
>>>>> Make : gravity-3 of b3                                       (e)
***** CURRENT GOAL ----> make-one-kei
>>>>> (b2 b1) are regarded as kei-1 by gravity-kei strategy.       (f)
>>> Change : gravity-2 of b2 ---> force-1 of kei-1                 (g)
>>> Change : gravity-1 of b1 ---> force-2 of kei-1                 (g)
>>> Composit : force-1 + force-2 - - > force-3 of kei-1            (g)
>>> attention - - > (kei-1 b3)                                     (h)
>>> partial*attention - - > ((kei-1 s1 b3))                        (i)
>>>>> (kei-1 b3) are regarded as kei-2 by string-kei strategy.    (j)
>>> Change : force-3 of kei-1 ----> force-4 of kei-2               (k)
>>> Change : gravity-3 of b3 ----> force-5 of kei-2                (k)
>>> Composit : force-5 - force-4 --> force-6 of kei-2              (k)
>>> attention ----> (kei-2)
>>> partial*attention ----> ((kei-2))
***** CURRENT GOAL ----> solve-acceleration
>>> Motion*equation of kei-2 ...                                   (l)
 direction ----> irnaginary*right                                  (l)
((((15. o + 10. O) + 30. O) * variable-1)                          (l)
=((30. O * G)-((15. O * G) + (10. O * G))))                        (l)
----> 0. 8909090909090909 (m/ss) ... acceleration of kei-2
***** CURRENT GOAL ----> solve-goal
>>>>>>>>>> ANSWER <<<<<<<<<<
***** b1's acceleration ----> o. 8909090909090909 (m/ss)
```

図２．５－５　動作例

一方、問題解決のコントロールは、やはりオブジェクトと、その間のメッセージパッシングによって行なわれる。図２．５－４に、コントロールのための主要なオブジェクトが示されている。図２．５－４の中のmonitorはトップレベルで働く。つまり、monitor startコマンドによってmonitorにstartというメッセージを送ると、motinorの中のstartというメソッドが起動し、オブジェクトgoal、planner、solverを順に呼び出して、問題解決をコントロールする。

図２．５－５に、問題解決システムからの出力例が載せてある。 (a) はユーザからの入力で求めるべき未知変数がブロックb1の加速度であることを意味している。 (b) は問題に含まれる物理的対象のリストである。システムは、これらの対象を見つけたうえで、 (c) のようにまずブロックに注目し、さらに (d) のように、糸でつながったブロックを捜し出す。そのうえで、 (e) のような形で各ブロックの重力が付け加えられた、それぞれのブロックに対応するインスタンスが生成される。

次に、システムは、問題を解くにあたって１つとみなせるような対象があれば、それらをまとめて１つにする。図２．５－５では、 (f) でこれが行なわれており、ブロックb1とb2をまとめたkei-1と呼ばれるオブジェクトが作られている。kei-1にかかる力には、 (g) にあるようにforce1、force2、force3という名前がつけられる。こうして、考えるべき対象は、 (h) のようにkei-1とブロックb3となり、さらに (i) のように、これらの対象のどれが１つの対象とみなせるかどうかが調べられる。kei-1とb3

は糸でつながっているので、（j）のように、kei-2という名前の新しいオブジェクトが作られ、１つの仮想的な対象とみなされる。そして（k）のように、kei-2にかかる力に新しい名前がつけられる。ここまでくれば、（仮想的な）物理的対象はkei-2 １つだけになったわけで、kei-2についてのNewton方程式を立てて解くのは容易である。（1）はシステムが立てた式で、答は0.89m／ss、ただし、加速度の方向は下方である。（1）にimaginary＊rightと示されているように、加速度の方向は、システムにとっては“右”になっている。これは、システムが滑車p1にかかった糸の（仮想的な）方向を“左右”とみなしているからである。

２．５．２　オブジェクト指向運動モデル

前項では、オブジェクト指向による簡単な物理系のシミュレーション・モデルをプログラム言語レベルで概観した。ここでは、簡単な鋼体の運動モデルについて視覚的にオブジェクト・シミュレーションについて検討する。

はじめに、図２．５－６から図２．５－１１に示すような代表的な形状の物体を考える。これは、我々の空間を占有する物体のニュートン力学での基本的物理形状と考えることができる。これらの形状は、我々の物理空間を占有する物体の形状を細分化した時に得られる、基本的な特徴を理想化していると考える。本項ではそれらを物理オブジェクトと呼ぶことにする。

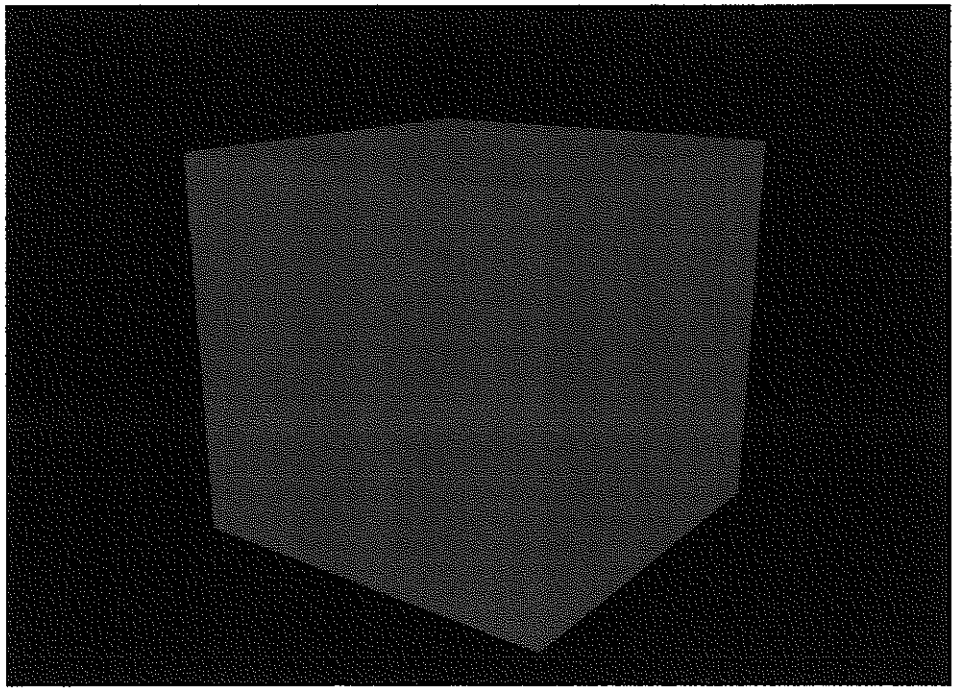

図２．５－６　立方体オブジェクト

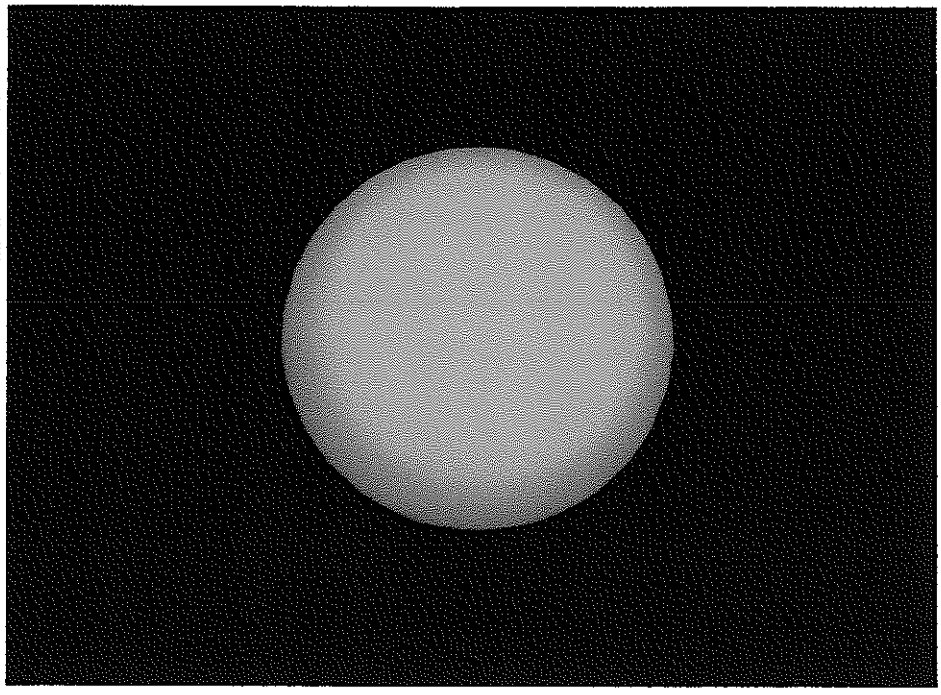

図２．５－７　球オブジェクト

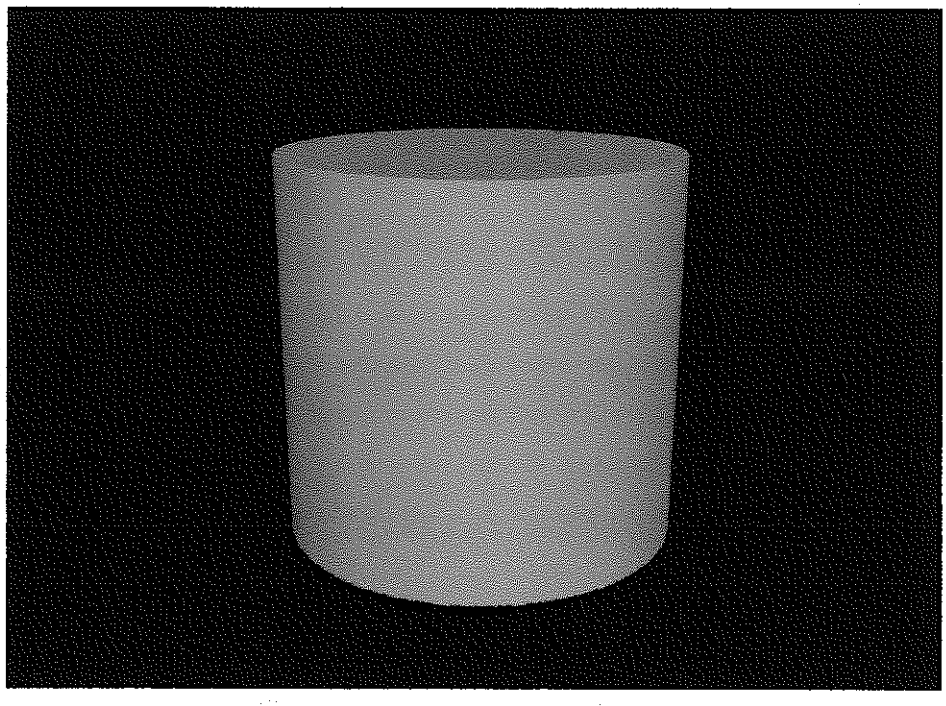

図２．５－８　円柱オブジェクト

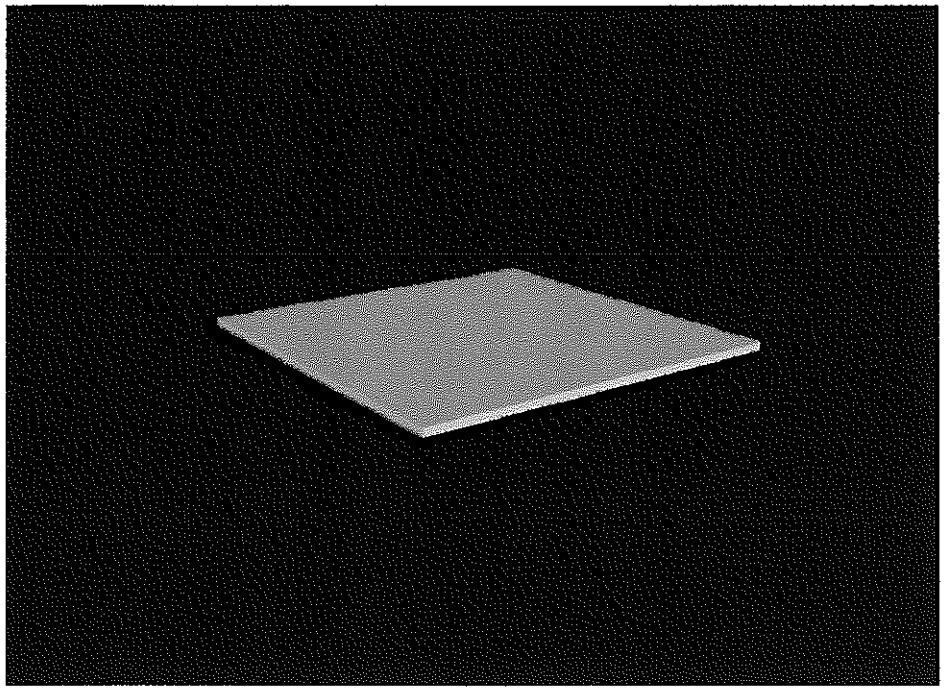

図２．５－９　平板オブジェクト

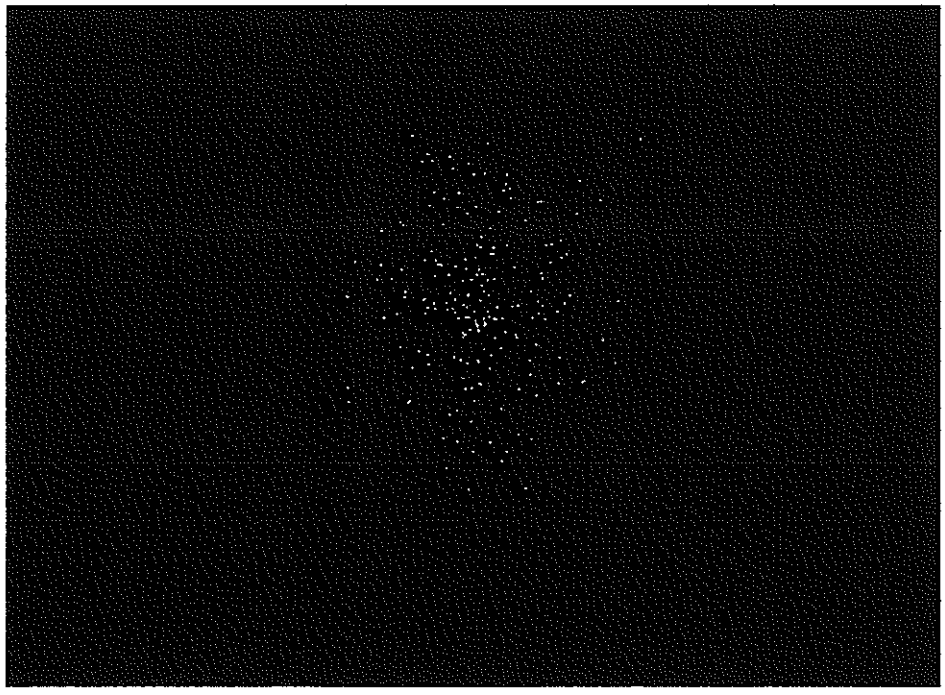

図２．５－１０　質点オブジェクト

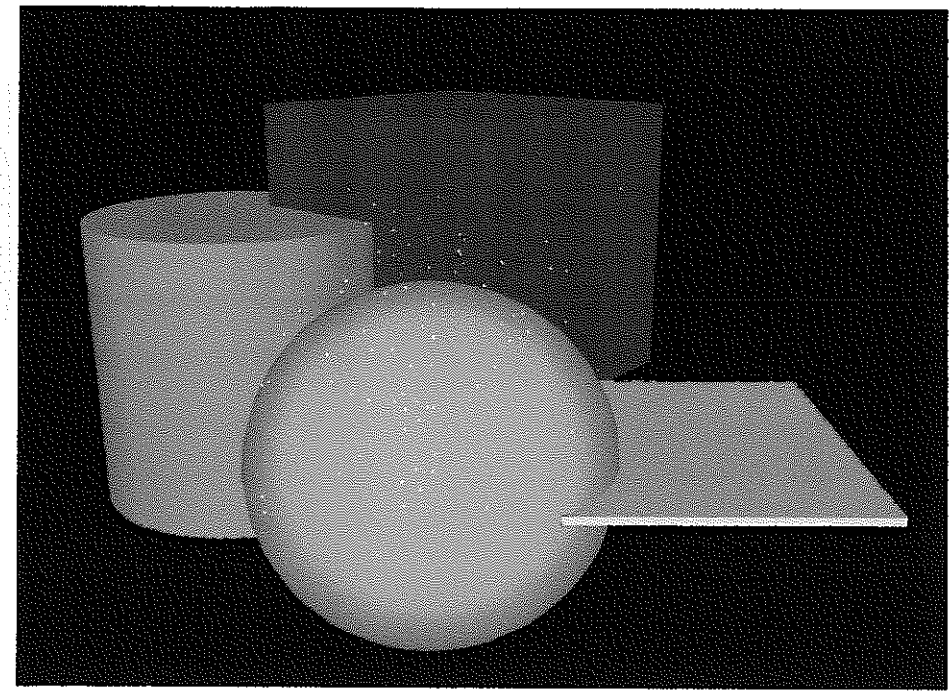

図２．５－１１　複合オブジェクト

ここでは、オブジェクト指向によるシミュレーション・モデルについて理解を容易にするために、図２．５－６から図２．５－１１に示す物理形状オブジェクトを鋼体として考える。シミュレーション・モデルは、計算可能な最も単純な物理形状を想定する。これは、オブジェクト指向によるモデル組立の重要な考え方である。

オブジェクト指向のモデル開発は、できるだけ単純化された最小の物理単位を考えると解かりやすい。例えば、水は水分子を理想化した質点として考え、それを物理に関わる単純な式（例えば運動）とで表し、シミュレーション・モデルは、それが観測に充分な量（例えば１０万個、１００万個）の存在とする。また、破砕体のモデルは、小さな球が詰まった水槽を仮定し、水の流れはそれらが造る空隙にそって前述の多数の質量粒子が個別に移動すると考えることができる。これらのオブジェクトは、それ単体で質量を持ち、外力による運動を行うとして考える。すると、これらのオブジェクトは、それ自身の特徴を表すデータ（例えば、立方体なら８つの頂点を表すベクトル）と、外力により釣り合うための関数（F＝maと摩擦を表す式等）から記述できる。このよう考えると、対象とするオブジェクトは、逸れ自身の特徴を表す情報と、外部からのメッセージにより振る舞いを決定する関数の組として考えることができる。Ｃ＋＋プログラムでその考え方をモデル化するならば、前者はインスタンス変数であり、後者はメンバー関数である。それを組み合わせた情報構造がクラスであり、クラスにメッセージを送り、具体的なインスタンスがプロセスとしてメモリー上で決定した状態がオブジェクトである。

シミュレーションに於ては、オブジェクトの動作を決定する“運動の方程式”を考える必要がある。運動の方程式は、オブジェクトの座標、速度、加速度、そして力を記述する。ここでは、簡単化の為

対象を鋼体（対象に外力が働いても、その大きさ、形が変化しない理想物体）として考える。オブジェクト指向モデルでは、この段階で１つの鋼体の記述を終了する。従って、オブジェクトは、外部からメッセージとして力を受けると、その反作用として移動する。また、移動先に他のオブジェクトがあれば、それに衝突しメッセージとして、力や加速度を与えることができる。これに比べ、有限要素法等では、釣り合いや変位を基準に、場としてモデルを考えているので、対象は系を自由に動き回ることができない点が大きな違いとして上げられる。そのような理由からも、有限要素法等で自由に動き回る対象を精密にモデル化することには限界があると思われる。

その様にして、例えば平面上に置かれたn個の鋼体の衝突運動のシミュレーションを、運動と平面の摩擦だけ考慮することで、n体の連成問題をシミュレーションすることができる。これは、場の方程式による初期値問題として示されるn体問題と、問題自身は同じであるが、シミュレーション手法は全く異なっている。その考え方は、一つの対象についてのみ物理的に考察し、それらをn体まとめてコンピュータ中で衝突実験する考え方である。そのために重要となるソフトウェアモデルの考え方が、クラスとインスタンスを備え、関数として外部からメッセージを受け取ることのできるオブジェクト指向である。

（１）仮想世界に於けるオブジェクトの座標

始めに３次元空間におけるオブジェクトの位置と原点について考える。

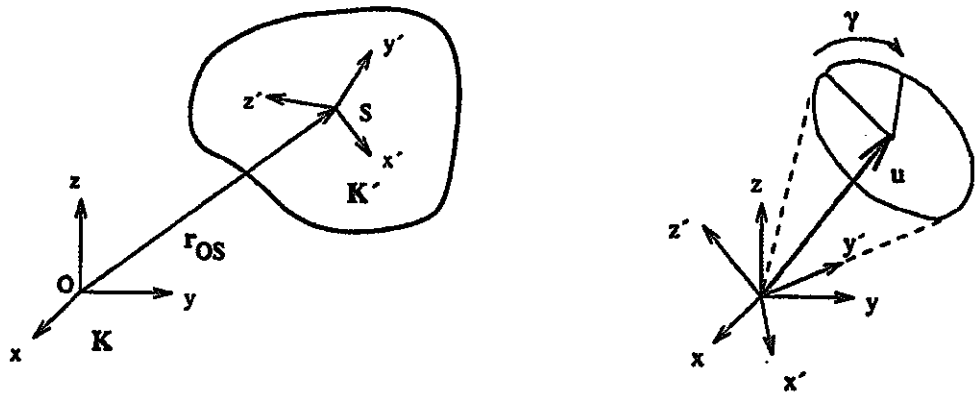

図２．５－１２　鋼体の移動

図２．５－１２に示すように、鋼体が運動した時の始めと終りの２通りの系について考えると、固定座標系Kからのベクトルr**OS**は移動後の位置を示している。ベクトルの先端を新たに移動した系K'の中心（新たな原点）として考えると、移動前の系Kと移動後の系K'の変換を四元数のオイラーパラメータで表現することができる。

$$q = (q_0, q_1, q_2, q_3) \qquad \textit{Euler Parameters}$$
$$q_o = \cos\left(\frac{\gamma}{2}\right)$$
$$(q_1, q_2, q_3)^T = u \sin\left(\frac{\gamma}{2}\right)$$

式２．５－１

次に、ベクトルの回転にｔついて式２．５－２の様に整理し、図２．５－３に示すK'からKへの変換行列を得る。

$$q \times v = \left(q_0, \vec{q}\right) \times \left(v_0, v\right)$$
$$= \left(q_0\, v_0 - \vec{q}\ \vec{v}, q_0 \vec{v} + v_0 \vec{q} + \vec{q}\vec{v}\right)$$

式２．５－２

$$A = \begin{bmatrix} 2\left(q_0^2 + q_1^2\right) - 1, 2\left(q_1 q_2 + q_0 q_3\right), 2\left(q_1 q_3 - q_0 q_2\right) \\ 2\left(q_1 q_2 - q_0 q_3\right), 2\left(q_0^2 + q_2^2\right) - 1, 2\left(q_2 q_3 + q_0 q_1\right) \\ 2\left(q_1 q_3 + q_0 q_2\right), 2\left(q_2 q_3 - q_0 q_1\right), 2\left(q_0^2 + q_3^2\right) - 1 \end{bmatrix}$$

式２．５－３

得られた変換マトリクスから二つの系を結び付けるベクトルの回転が得られる。次に図２．５－１３に示すように、鋼体の移動先Pを考えるとベクトルrOPは式２．５－４の通となる。

$$\vec{r}_{OP} = \vec{r}_{OS} + \vec{r}_{SP} = \vec{r}_{OS} + A^{-1} \cdot r_{SP}$$

式２．５－４

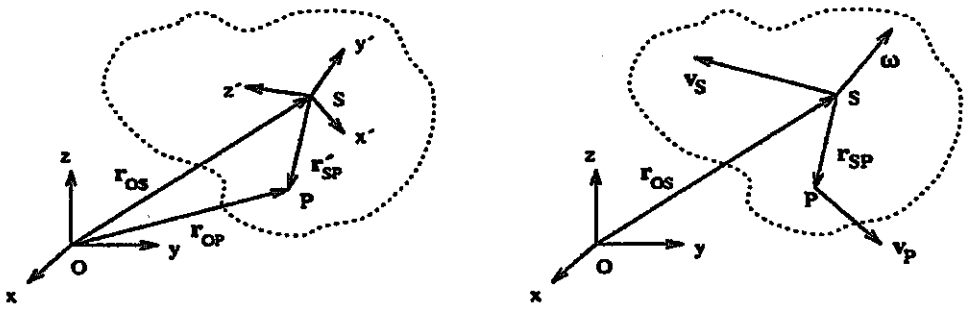

図２．５－１３　速度と座標

$$\vec{v}_P = \vec{v}_S + \vec{\omega} \times \vec{r}_{SP} \qquad \omega\text{:角速度}$$

式２．５－５

また、点Ｐの速度Vpは、式２．５－５で示すことができる。このように、鋼体の運動を変換マトリクスで表現することにより、それらの衝突を伴う移動のシミュレーションが、オブジェクト指向プログラミングで可能となる。その場合、一つの鋼体についてのモデルの型紙として、クラスの概念を考える。クラスは、鋼体の特徴を表す変量（インスタンス変数：例えば球なら半径Z）と３次元空間中での運動の座標を決める変換マトリクス・モデルから成るとする。プログラムは、このクラスを定義することにより行う。このプログラムの実行は、クラスに外力やインスタンス変数を決定するメッセージを送ることにより行う。この時、プログラムはクラスから唯一のオブジェクトとして、メモリー上で活動する。従って、クラスにメッセージを送ることにより、複数のオブジェクトをメモリー上に造りだすことができる。このようにして生成されたオブジェクトは、相互にメッセージを交換し、それに応じて状態を変化させることができる。それにより、多数の鋼体の複雑な衝突を含む運動のシミュレーションが可能となる。

（２）隙を通過する粒子

次に前述の考え方を拡張して、隙を通過する粒子を次のように考えることができる。この問題で利用するオブジェクトは２種類のみである。それらは、隙を造るために多数必要なボール・オブジェクト。物理モデルは、このオブジェクトが詰まった水槽を考える。

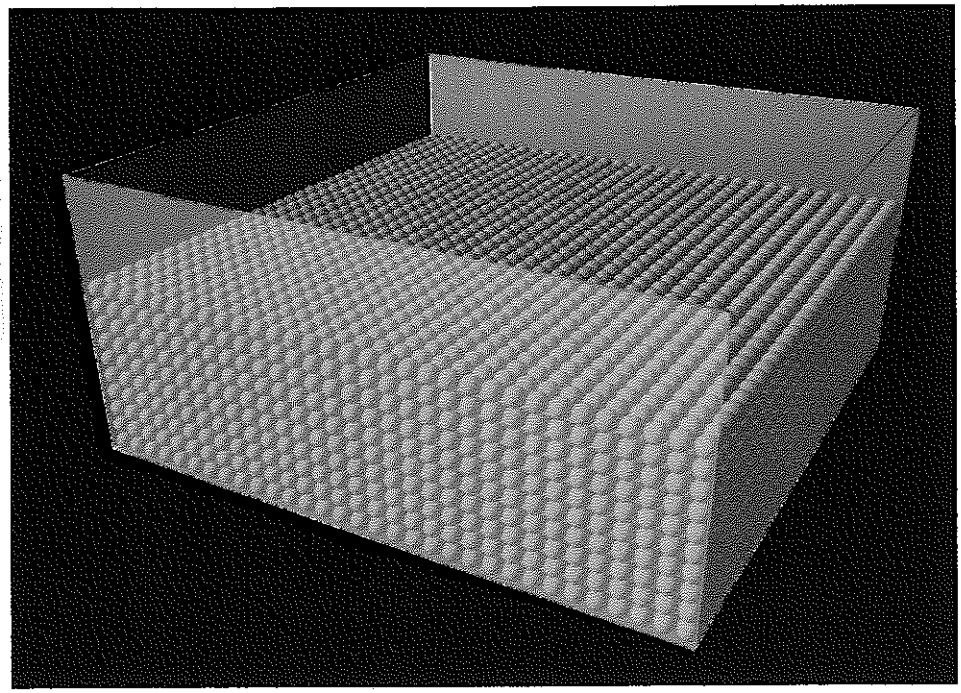

図２．５－１４　ボールの詰まった水槽

次に、粒子のモデルを考える。粒子のモデルは、外部から与えられる速度ベクトルに従って移動すると考える。シミュレーションする系は、図２．５－１４に示す水槽の一部を考える。次に多数の粒子を速度境界として与えることとする。この時、粒子の運動を制限する条件として、ボールとの摩擦、重力加速度、ボールとの衝突を考慮すると、流入した粒子は、あたかもコロイド等の微小粒子が浸透水流れにそって微小隙を流れる様子を再現した様に運動する。図２．５－１５から図２．５－１９にシミュレーション例を示す。

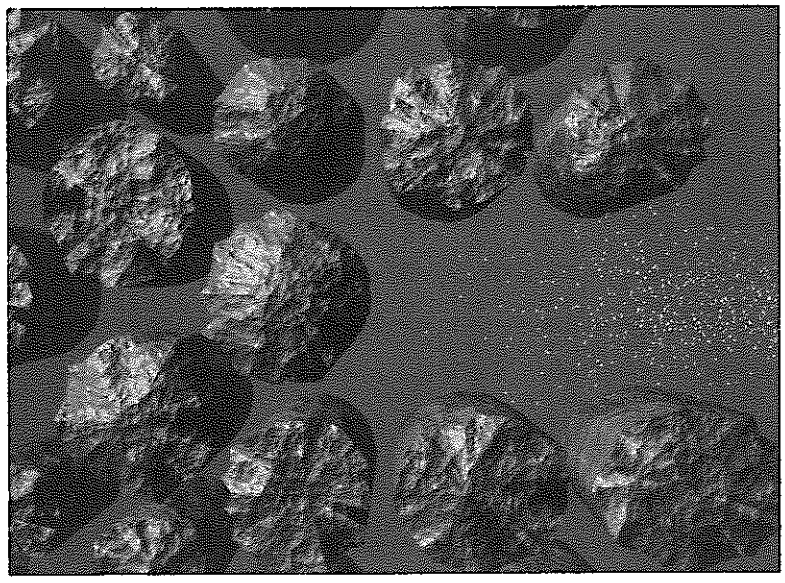

図２．５－１５　粒子の隙移動シミュレーション　スナップ１

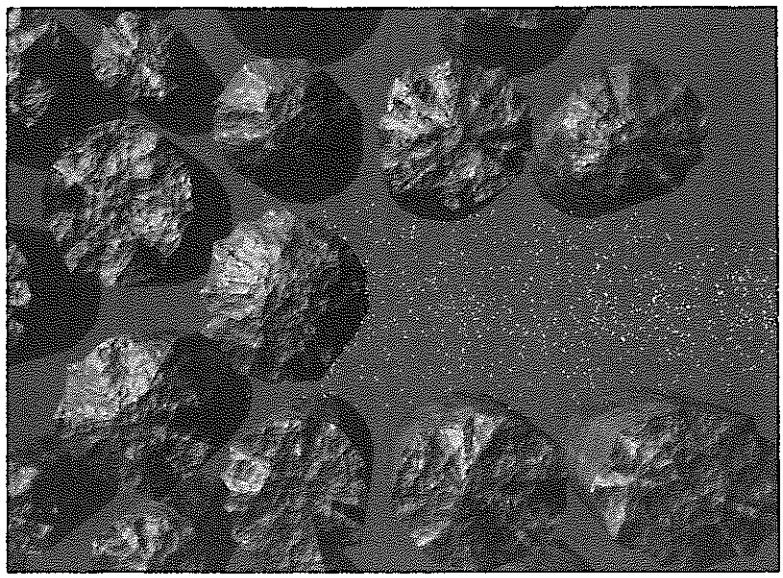

図２．５－１６　粒子の隙移動シミュレーション　スナップ２

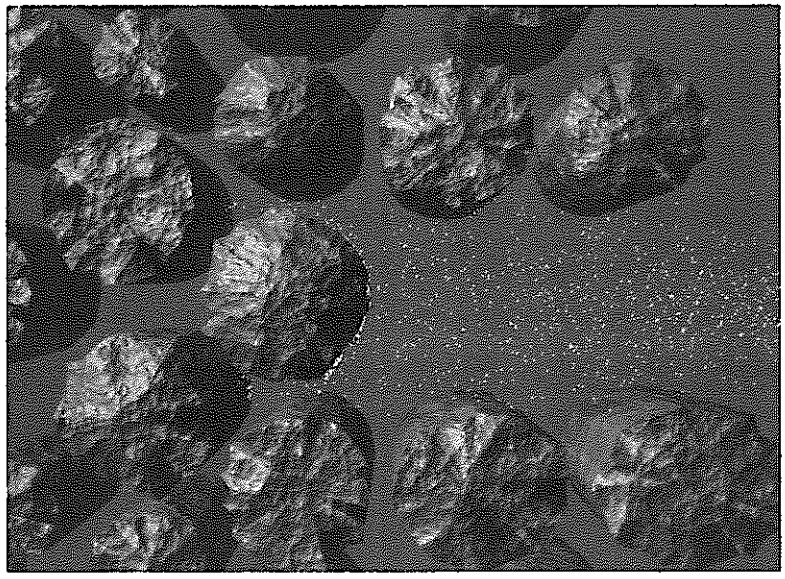

図２．５－１７　粒子の隙移動シミュレーション　スナップ３

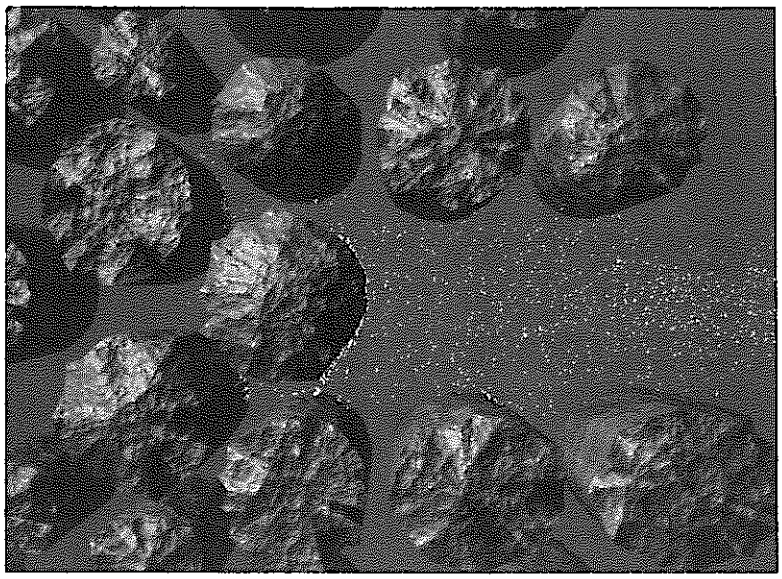

図２．５－１８　粒子の隙移動シミュレーション　スナップ４

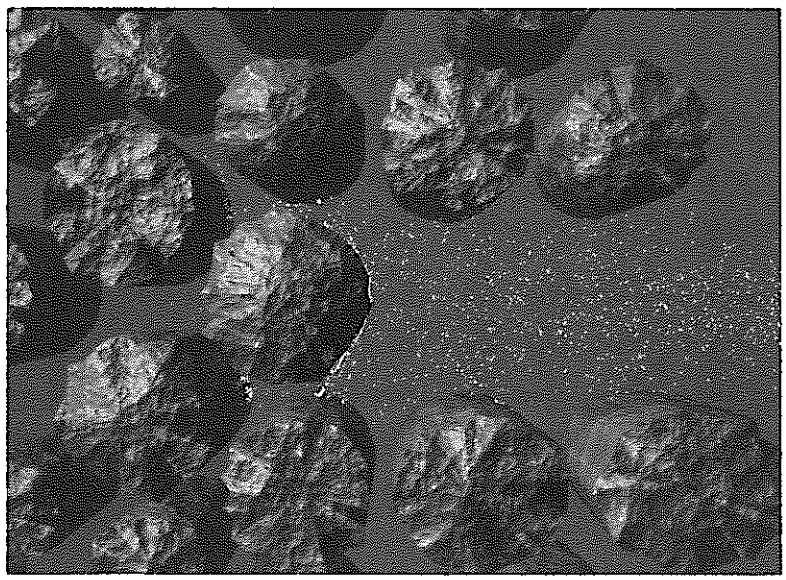

図２．５－１９　粒子の隙移動シミュレーション　スナップ５

以上のように、オブジェクト・シミュレーションの考え方は、従来のシミュレーションの考え方とは異なったものである。

それは、シミュレーション対象を極端に単純化するとともに、クラスとインスタンス、メッセージ交換という概念をプログラミングに採用し、移動や衝突を伴う対象をシミュレーションすることを可能にする。そのための最も基本的な手法は以下のとおりである。

1：対象を構成する最も単純な最小要素を物理モデルとして考える。例えば、構造物はそれを構成する要素の組として考える。流体はそれを構成する分子等を代表的な粒子として考える。従って、流体は非常に単純な対象に分解できる。

2：対象とする物理モデルの運動について数理モデルを考える。対象が粒子であれば、移動や、摩擦、衝突等の扱いは単純なモデルとなる。このようにして、導いた数理モデルは外部から与えられたメッセージ（外力等）に応答して物理モデルの位置等決定する関数と考える。

3：対象となる物理モデルは、それを特徴付ける変量（インスタンス）と状態を決定する関数からなるプログラム・アルゴリズム（クラス）として記述する。

4：クラスに対して固有の現在座標等のメッセージを送り込むことにより、実行プロセス（オブジェクト）を必要な数だけ産み出す（C＋＋ではインスタンス化と言う）

5：各オブジェクトは、互いにメッセージを交換し、時間とともに摩擦、衝突等の状態を確認し状態を変える。その状態を視覚化して観察することにより、リアルなシミュレーション結果が得られる。

ただし、その様にして得られたシミュレーションは、場の状態や釣り合い等を記述する方程式から出発していない。従って、例えば、流れの問題を扱うことを考えた場合、従来のナビエ・ストークス方程式等の場の方程式から出発する有限要素法や有限差分法とは本質的に異なった考え方である。

従来の手法が、どちらかと言うと観測された事実を数値的に再現するための定式化を目指していたことに比べ、オブジェクト指向は数値的に実験を行う手法ととらえることがより適切と考えられる。

3　並列計算技術の応用研究

地層処分ヴァーチャル・エンジニアリング・システムのシミュレーション機能について、一部並列計算技術を応用することにより計算速度の向上等を図り、より大規模かつ詳細なシミュレーションを可能とすることを目的とした研究を行った。

現在、並列計算技術については、各種のハードウェア・アーキテクチュアやコンパイラ等が開発されているが、メーカー固有の技術に左右される課題も少なくない。例えば、データ・パラレルに代表される超並列計算技術については、ハードウェアメーカー各社で独自の技術開発が行われており、仕様の統一がなされていない事が現状である。また、ネットワークにて接続された複数の異機種コンピュータを利用して並列に計算処理をすすめる、ネットワーク・パラレル技術の開発も進められている。この方式によると、並列計算環境を特別なハードウェアに限定しないで構築することが可能となる。ネットワーク並列計算技術は、1990年代前半の初期の段階では各種の手法が提案されていたが、メーカー、大学、研究所が協力して仕様の標準化が開始されたところである。

3．1　シミュレーション計算量の推定

並列コンピュータについて理論的、及び知的な関心が寄せられていることは別として、この分野の研究を推し進める推進力となる以下の２つの要素が考えられる。

その１：高性能コンピュータへのとどまることを知らない需要の増大
その２：期待される高性能コンピュータの開発技術を制限してしまう物理的な障害

地層処分ヴァーチャル・エンジニアリング・システムで必要とされる、多くの処理操作には高い計算能力を必要とする。特に、地下水流れのシュミレーション、化学的反応のモデル化、処理場周辺の岩盤力学モデル、気候や地質学上の予測などのモデル開発には、高い計算能力が必要とされると考えられる。

また、それらのアプリケーションの実行を支えるものとして、データベース・システムが考えられ、それは、入出力性能についても高い能力を必要とするであろう。

更に、シミュレーションの視覚化や、仮想世界の構築の必要性からも、より高い性能のコンピュータが求められるであろう。以上のように、地層処分ヴァーチャル・エンジニアリング・システムを実現するための、コンピュータ・ハードウェアに求められることは、第一に非常に高速な計算と入出力処理能力であると考えられる。

ハードウェア面から見れば、従来のコンピュータ機器の性能は、処理速度がどれだけのものであるかによって判断されてきた。

従来、単一のプロセッサに基づくアーキテクチュアが、その性能を飛躍的に高めてきたが、ハードウェアの開発も現在の技術では物理的な限界にきていると言われている。例えば、最近のVLSIチップのライン幅は約0.5ミクロメーターで、それは写真食刻（エッチング）技術で使用される波長（数千オングストローム）に近い数値である。高性能化のためには、信号線を密接に接続することによって、相互誘導効果が生じ、回路の操作速度を早めるために現在広く使われている小型化の技術がもはや妥当なものではなくなってきている。

小型化の技術が妥当なものではなくなるのがいつになるのかについては、いろいろな予測があるが、多くの技術者は、それは西暦2010年前後であろうと予測している。従って、地層処分ヴァーチャル・エンジニアリング・システム・プロジェクトでは、現在のハードウェア・アーキテクチュアを基盤とした研究開発を継続していくことができると考えられる。

並列処理では、複数のプロセッサの計算能力を組み合わせることによって、全体の処理速度が高まることを狙ってシステム開発を行う。この処理能力によって、シミュレーションがより迅速に行われるだけでなく、より複雑な解析が可能になり、より高度なシミュレーションを行うことも出来るようになることが期待されている。

また、プロセッサの単体速度に加えて、プロセッサ、主記憶装置、及びディスク間をデータが転送される速度に、より多くの処理が制約を受ける。それについても、並列入出力アーキテクチュアを使用することによって、制約を受けずに、機器の大きさに応じて性能を段々高めていくことも可能になると考えられる。特にデータベース・システムや視覚化処理の性能向上では、効果が期待されている。

（1）計算量

地層処分ヴァーチャル・エンジニアリング・システムのシミュレーションに要求される演算速度を見積ることは、モデルの正確な構造に関する知見なくしては難しい考えられるが、その概要を把握することは大切である。ここでは、シミュレーション領域を3次元の格子として考え、計算量に関わる凡その情報を把握する。最も単純な計算として、3次元立方格子でモデル化された計算領域を差分法に代表する隣接格子計算で考える。各格子における汚染レベルを取り囲む格子の汚染レベルをもとに決めるというものである。

隣接格子計算においては、あるひとつの格子で定まる物理量は、それに最も近い周囲の格子における物理量によって決まると考える。対角上の格子も数えると、3次元空間ではひとつの格子に26の「隣接要素」が存在する。従って、各格子について、計算は27回（各隣接格子についての計算が26回、それに前回の反復のあとの当該格子の計算が1回）行う必要がある。シミュレーションが、反復のたびに10個の変数を更新し、それぞれの更新に10回の浮動小数点演算が必要であると仮定すると、下記の量の計算が行われることになる。

27*10（変数／反復・格子）*10（FLOP／変数）＝2700（FLOP／反復・格子）

また、滑らかな視覚化シミュレーションのためには、毎秒24フレームの割合で100万個の格子すべてについてシミュレーションを実行する必要がある。その場合、以下の計算量が単位時間（秒）あたりで必要である。

2700（FLOP／反復リフシュ）*24（反復／秒）*1*1,000,000.＝65GFLOPS

この65GFLOPSという計算速度は、1秒間に10の9乗回の浮動小数点演算を行う速度（ギガ）を単位としたものである。この65ギガフロップスは、現在各国で利用されている汎用の超並列コンピュータの最高速度に近いものである。この数字は、10個の未知数について（100, 100, 100）の格子点を持つ

3次元シミュレーションを実時間で動画再現するために必要な計算速度を表している。

（2）メモリー容量

メモリー容量は、それぞれの変数が4バイト（単精度）のメモリを占有すると仮定すると、シミュレーションの1回の反復に必要なメモリー容量は下記のとおりとなる。

1*1,000,000.*10（変数／格子）*4（バイト／変数）＝40Mバイト

上記は変数の数が10と、比較的少なく見積っている。従って、実際のプログラム開発に於ては、その2倍から3倍見込むことが適当と思われる。従って、このようなプログラムは約100MB程度のメモリーを要求すると考えれば良いと思われる。

また、各サイクルの計算が、たとえその場でできるとしても（連続して行われる個々の反復の結果は同じメモリ位置に保存される）、中間的な結果の記憶のための空間は、やはり割り当てておく必要がある。それに、20%のオーバヘッドを見込んだデータ空間の規模は100～150Mバイトと見積られる。

さらに、計算がその場でできない場合は、データ領域は反復2回分を記憶するのに十分な容量を備えていなければならない。この最悪のケースでは、データ空間は最高200～300Mバイトの容量を備えている必要がある。それでも、物理モデルが2km立方の場合、1格子の大きさは20m立方である。廃棄体は緩衝材直径で約3mなので精度の面で充分ではない。

（3）データ転送速度

この項では、シミュレーション機能と視覚化機能の間のデータ転送速度の概算値を導き出す。シミュレーション域全体を見る場合、半透明立方体視覚化モデルでは、反復のたびにシミュレーション・データ空間の内容すべてが視覚化機能に送られる必要がある。シミュレーション・データ空間の大きさは、144～480Mバイトと見積られる。毎秒24回の反復数でデータ空間全体を転送するためには、毎秒3.5～11.5Gバイトのデータ転送速度が必要となる。シミュレーション空間の一部だけを表示する場合は、データ転送速度の見積り値もそれに応じて下がることになる。

（4）視覚化機能

例えば、半透明立方体視覚化モデル（ボリュームレンダリング）では、ディスプレイ用に選ばれた変数をデータ空間から抽出し、データを圧縮したうえで、ディスプレイ準備機能に、そのデータを転送する操作を視覚化機能によって行う。計算量の見積り用として、ここでは極めて単純な形式のデータ圧縮を想定する。すなわち、データ圧縮は、シミュレーションの最終回の反復において数値が変化した変数だけを転送することによって行うと考える。

毎秒24回の反複数で実行されるシミュレーションで、3つの変数を同時に表示する場合には、視覚化機能はデータ領域から最大で以下の変数を抽出する必要がある。

3（変数／格子・反復）*1*1,000,000.*24（反復／秒）＝72,000,000.0（変数／秒）

抽出後、それぞれの変数の数値は、その前の回の数値と比較されディスプレイ準備機能に出力される。データの比較と出力に3回の浮動小数点演算を見込んだ場合、視覚化機能の所要演算速度は以下のように見積ることができる。

72,000,000.0（変数／秒）*3（FLOP／変数）＝22MFLOPS

視覚化機能は、抽出データを圧縮した後、ディスプレイ準備機能へと転送しなければならない。この時、データ圧縮が90%の効果を持つとしたら、3つの変数のディスプレイに必要となるデータ転送速度は以下のようになる。

1,000,000.0* 4（バイト／変数）*3（変数／格子・反復）*0.1 =1.2（格子・反復）

毎秒24回の反復数で実行されるシミュレーションの場合、視覚化機能とディスプレイ準備機能の間の通信チャネルの所要帯域幅は次のようになる。

1.200,000.0（バイト／反復）*24（反復／秒）=29（メッシュ／秒）

次に、ディスプレイ準備機能は、シミュレーション区域の3次元半透明画像の図形表現を行い、それによって得られた画像データを圧縮し、ディスプレイ機能へと転送する。

図形表現のためには、視線上にある全ての格子の整数加重合計を、画像のそれぞれのピクセルについて計算する必要がある。この計算は、赤、緑、青の3原色のそれぞれについて繰り返されなければならない。この計算には、各格子毎に5回の整数演算が必要であるとすると、シミュレーションの反復1回毎に必要とされる整数演算の回数は次のようになる。

800*1000*100ピクセル*3（変数／反復）*5（10P／変数・ピクセル）
＝1.2*10の9乗（10P／反復）

毎秒24回の反復数で実行されるシミュレーションの場合、最悪のケースを想定したディスプレイ準備機能の所要演算速度は次のように見積ることができる。

1.2*10の9乗（10P／反復）*24（反復／秒）＝28,800. MIOPS

この数は、シミュレーションの1回の反復の間に、画像中のすべてのピクセルを計算し直すのに必要な演算の数を示すものである。但し、画像全体の計算のやり直しが必要となるのは画像を回転させる場合だけである。殆どのシミュレーションのシナリオでは、表示されたピクセルのごく一部を計算し直すだけですむ。そこで、画像更新速度が画像の回転中に遅くなるのを認めることにすれば、通常のシミュレーションのシナリオにおける所要演算速度の見積りを行うことができる。

例えば、1回の反復の間に表示されているピクセルのうちの5%が変わるものとすると、シミュレ

ーションの１回の反復毎に必要とされる整数演算の回数は次のようになる。

40*1000*100ピクセル*3（変数／反復）*5（10P／変数・ピクセル）＝60*10の6乗（10P／反復）

そこで、毎秒24回の反復数で実行されるシミュレーションで、ディスプレイ準備機能に要求される演算速度は次のように見積ることができる。

60*1,000,000.0（10P／反復）*24（反復／秒）＝1,440 MIOPS

図形表現の完了後、ディスプレイ準備機能はディスプレイ機能にデータを転送しなければならない。表現された図形データは、表示ピクセルひとつが24ビットの整数データ要素ひとつによって構成されている。

この24ビットの整数データは３つの8ビット・バイトに分かれ、それぞれのバイトが各ピクセルにおける、赤、緑、青のそれぞれの色の農淡を表現している。画像が800×1000ピクセルを占めると仮定すると以下のデータの転送が必要となる。

800*1000ピクセル*3（バイト／ピクセル・反復）＝2.4*1,000,000.（バイト／反復）

毎秒24回の反復数で実行されるシミュレーションの場合、最大負荷のケース（データ圧縮が一切行われない）を想定したデータ転送速度は次のようになる。

2.4*1,000,000.（バイト／反復）*24（反復／秒）＝58（Mバイト／秒）

データ圧縮アルゴリズムが90%の効果を持つ場合には、所要データ転送速度は毎秒5.8Mバイトに下がる。半透明立方体視覚化モデルには1280×1024ピクセルのラスタ式ディスプレイが必要である。

（４）基本データ処理量

以上のように、３次元にて100,100,100格子の分割された計算モデルを差分法等でシミュレーションする場合、ハードウェアに対する凡そのデータ処理速度の要求は以下のとおりである。これらの数値は、地層処分性能評価シミュレーション・システム・ハードウェア要求性能のある程度の目安である。

計算速度	：70 GFLOPS以上
メモリー容量	：200MB以上
入出力速度	：28MB/sec
画像処理速度	：60MByte/sec

３．２　並列計算技術の現状

大規模シミュレーション技術開発として良く知られたグランドチャレンジ（1992年 合衆国高性能計算通信プログラム）は、その目標を達成するのに必要なアプリケーションの内容、処理速度、記憶容量、そして入出力を次のように明示している。

近い将来、テラフロップの処理速度、１テラバイトの主記憶容量、及び１テラバイトの入出力性能のコンピュータが、流体力学、海洋の循環、気候のモデル化等の問題解決に利用されるであろう。

例えば、72時間の天気予報では、およそ10Gフロップの処理速度が必要とされている。現行のスーパーコンピュータの処理速度は、グランドチャレンジのアプリケーションに必要な処理速度の100倍から10,000倍も遅いものとなっている。

コンピュータ・ハードウェアは常に革新されている。市場は主要メーカーによる接収、合併など最近いくつかの大変革を経験し、業務内容も大きく様変わりしている。ここ数年間、従来の伝統的な“メインフレーム”設備から、より小さい“事業レベル”のサーバーシステムに移り変わってきている。この傾向は、すぐに入手可能なCPU技術と迅速なネットワーク機能を有する生活必需品として、小型化した多機能分散処理システムへの変遷ともいえる。

従って、コンピュータ・ハードウェアの技術革新について次のことがあげられる。これらは多年度に亙る、ハードウェアとしての地層処分ヴァーチャル・エンジニアリング・システムの構築プロジェクトを推進するうえで重要な考慮事項である。

考慮事項１：モアの法則をマイクロプロセッサ技術に継続して適用可能であること。30年前にインテルの共同創設者である、ゴードンモアによって次のように述べられている。

＊ RISCとCISCのチップトランジスターの記録密度は18カ月ごとに倍増し続ける。

＊モアの法則は沈静化の兆しがない。

これに対する、否定的な見解は以下のとおりである。

＊RISCとCISCのユニットの容量が大きすぎるので、特注品の超高性能プロセッサは、価格及び性能の競争において敗北してしまった。これは、ベクトル系の“メインフレーム”機器の衰退を示している。

考慮事項２：超並列コンピュータ（MPP）が大規模に市販されるようにはなっていない。MPPシステムはプログラム化することが困難で、より進んだソフトウェア技術に遅れをとっていることは、超高性能プロセッサ全体の市場では、すでに失敗していることを意味している。

高性能コンピュータの実用的なアプリケーションを特色づける２つの重要な特性がある。性能を問題にする際、これから述べる２つの特性を必ず考慮にいれるべきである。

リアルタイム特性	：多くのアプリケーションにおいて、計算結果が有効である限られた時間内に計算がおこなわれなければならない。
精度に依存した特性	：より高い精度を得るためには、できるだけたくさんの問題を処理しなければならない。

精度に依存した特性は、リアルタイム特性と組み合わされて言及されることが多い。例えば、同じ計算量の問題があるとするならば、より高い精度を得るためには、より短い時間で処理されなければならないし、たくさんの問題があるとするならば、それぞれ同じ位の時間、もしくはより短い時間で処理されなければならない。このことは、単位時間内での計算速度向上パラダイムとして知られており、その主な目的は、必ずしも時間を節約することではなく、**一定の実行時間内に、より高い精度で問題を処理する**ことである。このために計算能力を高める必要性が生じ、コンピュータ自体の性能を高める必要性も生じる。

多くのアプリケーションは、外部（すなわち環境）の物理的プロセスをシミュレーションすることに基づいている。シミュレーションの質がいいものかどうかは、モデルが外部（環境）を物理的にいかに詳細にわたって記述しているかにかかっている。これらの物理的プロセスは、３次元空間と時間で（つまり４次元的に）おこなわれている。

例えば、気象予報、電磁場の分析、インタラクティブ（対話型）システム、及び海洋のモデル化などに適用されている。また、多くのアプリケーションにおいて、ボルツマン輸送方程式を使って物理的なプロセスが表わされている。ボルツマン輸送方程式によって、粘性流体、半導体における電子の流れなどについての物理的なプロセスを表わすことができるようになった。

物理的プロセスは、一般に４次元の別々のポイント（グリッドポイント）で、偏微分方程式から求められた一次代数方程式を使って表わされる。マトリクスの大きさは、1,000から10,000位のシステムにもなり得る。上記の方法を用いてシミュレーションの精度を高めるためには、より細かいグリッドを選び、より広い空間でシミュレーションを行わなければならない。計算量の見積りは前節で示した。

（１）並列ハードウェア

並列コンピュータは、凡そ20年間出回ってきているが、学界や研究センターのような場所でのみ使われていることが現状と思われる。現在使われている並列コンピュータは、主に次の２つのタイプに分けられる。

分配型メモリーシステム：超並列プロセッサ(MPP)やワークステーションクラスターを含む。
共用メモリーシステム：対称型マルチプロセッサ(SMP)や並列ベクトルプロセッサ(PVP)を含む。

MPPシステムは、何百個ものあるいは何千個もの安価なRISC、もしくはCISCプロセッサからなり、迅速な特注品のインターコネクトと接続されている。MPPの代表的なシステムは、Cray T3D/T3Eシリーズ、日立SR2201、インテルパラゴンシリーズや富士通VPPシリーズなどである。

富士通VPPシリーズは若干タイプが異なり、超大規模プロセッサというよりは、むしろベクトル系のMPPのシステムであるといえる。多くの場合、MPPシステムは、迅速なネットワークを有した、大規模な同機種のワークステーションクラスターとみなされる。

それとは反対に、産業界や大学研究機関などで頻繁に使われるようになっているワークステーションやPCのネットワークは、対応の遅いタイプのMPPと考えることができる。実際、典型的なデスクトップシステムのコンピュータ機器の方が、典型的なMPPノードよりも高い処理能力を持っていることも事実である。

迅速なイーサーネット、FDDI、ISDN、ATMスイッチングなどのネットワーク技術のめざましい

発展に伴い、ワークステーションクラスターにおいて、その高い適用性そして費用対効果からも様々な環境に適合する、並列プラットフォームが開発されれていくと思われる。

PVPコンピュータは、多重ベクトルプロセッサに単一の共用メモリー域を有した初期の並列システムである。PVPコンピュータは、いくつかの強力な汎用システムを供給しているが、特注プロセッサ価格が高いことから、ますます高価になっている。例えば、Cray T90シリーズやNEC SX-4が上げられる。

SMPは、ある意味ではPVPをさらに開発してできたシステムで、一つの共用メモリースペースを有してはいるが、ベクトルチップのかわりにRISC、もしくはCISCプロセッサを使用している。このように安価なプロセッサを使用しているために、SMPは価格の面でも性能の面でもPVPより優れ、RISC技術が非常に進んでいるため、プロセッサの基本性能までも優れたものにしている。

現在使われているSMPのシステムには、サン・エンタープライズサーバー、デジタル・アルファサーバー、HPコンベックス・エクセンプラーやSXクラスシステム、SGI／クレイ・オリジン2000などがある。HPコンベックス・エクセンプラー、SXクラスシステムおよびSGI／クレイ・オリジン2000の主な特徴は、それらが不均一メモリーアーキテクチュア（NUMA）を有していることで、メモリーは物理的に分配されるが、ユーザーからは一つの共有メモリーアドレススペースと見える。上記の２つのシステムは、高い計算能力の他に視覚化性能という点でも優れている。

並列HPCの主な４つのカテゴリーの中で、SMPシステムだけがコンピュータ市場において広く受け入れられている。PVPシステムが非常に高価であること、MPPとSMP技術が、１つのアドレス・スペースと標準的な操作性を有したNUMA SMPクラスターに融合していく限り、この傾向はおそらく続くであろう。デスクトップシステムのクラスターは、エントリーレベルで、もしくは必要に応じ使用する超高速計算システムとして、それなりの重要性を保ち続けるであろうが、ネットワーク及びインターコネクト技術がますます進んできたことによって、PVPとかSMPなどとはっきり区別しにくくなり、区別することがたいして意味のないこととなるかもしれない。しかし、並列という概念が今後も超高速計算システムにおいて重要な要素であり続けることは確かである。

NUMAベースのSMPに対して圧倒的な支持が寄せられていることは確かであるが、システムの基本性能の点から言えば、現在のところ世界で最も有力なシステムはMPPである。1996年末のトップ50のスーパーコンピュータのリスト（http://www.netlib.org/benchmark/top500/top500.list.html）によると、上位10機の汎用コンピュータは、日立、インテル、クレイ、富士通で生産されたMPPシステムである。それぞれのコンピュータは、すべてのシステムが大学研究機関や政府機関で使われている。

（２）並列ソフトウェア

MPPが広範囲で使用されていない原因として、並列ソフトウェアツールが完備されていないことによる。仮に超高速計算システムの包括範囲が変動するとしたならば、広範囲の技術が必要になり、その結果、アプリケーションの軽量化が確実に必要になるが、今のところ軽量化はまだまだ充分とはいえない。欧州や米国では国家的なレベルで、多大な資金を投入し超高速計算システムの研究を行い、その結果、ここ２、３年の間でMPI（Massage Passing Interface）のような軽量な通信技術やPVMのような標準化により、安定したアプリケーションを確実に開発、生産できるようになってきている。

現行の並列ソフトウェアツールは層を形成していると考えることもできる。最も重要で最も下に位

置する層がプロセッサ間のコミュニケーションである。標準化のために多大な研究が行われるのがこの層で、今では明確なAPI（Application Programming Interface）がある。メッセージ交換インターフェイス（MPI）と並列仮想機器（PVM）の主に２つのメッセージ交換の標準があり、そのうちMPIは標準化団体組織によって正式に認められている。さらに、共用記憶機器は、IEEE Posix（Posix：ポータブルOS仕様）スレッド標準に基づいて、ますますスレッドプログラムライブラリを提供している。これらの３つの標準インターフェイスによって、安定した軽量なプラットフォームを開発することができる。

アプリケーションプログラムからプロセッサ間の詳細なコミュニケーションを分離させようという試みの中で、次のレベルの並列言語がうまれた。このレベルにおいて、唯一実在する標準が、高性能FORTRAN即ちデータをFORTRAN90に並列拡張したものである。現在、様々な分野で、いろいろな研究がおこなわれ、特に並列C＋＋についての確固とした標準書の草案をつくろうとするエスピリ基金によるヨーロッパ・プロジェクトの研究活動は注目に値するが、今のところ大した成果はあげられていない。

並列計算用コンピュータ言語は２つの種類に分けられる。一番目の種類の言語は、特性が明確にされているために、プログラマーはアプリケーションの中の並列性を容易に確認することができる。二番目の種類の言語においては、並列コンパイラに内在する並列性を並列コンパイラーが検知できる能力、及び並列コンパイラーが並列コードを自動的に生成できる能力にプログラム言語が大きく影響を受ける。

自動的に並列性を検知することができることによって、プログラマーは並列性を意識してアプリケーションを記述する必要はなくなったが、そのかわりコンピュータサイエンスという見地から、理論上のいくつかの難題に直面する。

並列コンパイラは1970年代半ば頃から存在してきたが、現在、能率的な汎用並列コンパイラは存在していない。コードの最適化におけるスケジューリングに関するいくつかの難しい問題と関連しているため、手続き型のコードと同じくらい能率的な汎用コードを開発できるか疑問視されている。

従って、超高速計算用コンパイラは、販売業者によって市販される様々な共用メモリープログラムツールとしばしば組み合わされ、超高速計算用FORTRANがコンパイラを使用する時、並列データ構造を記述せよという指示に従うことになる。これらのツールは、スレッドライブラリー、もしくはメッセージ交換インターフェイスのトップで使用される。これらを確実に、能率的に利用するためには、かなりの特殊な技術が必要である。

ソフトウェアの最後の層は、デバッグ装置やプロファイラ、そして性能モニター等の並列プログラムツールを保護している。

現在のところ、これらのプログラムツールは数少なく、販売業者間で特別使われる程度である。軽量なプログラミングツールも出回りはじめてはいるが、単一プロセッサシステムで利用可能なツールの数と比較すると、並列ソフトウェアにおける遅れははっきりとしているが、これからの研究開発に応じて発展する可能性はある。

３．３　並列シミュレーション・プログラム開発の事例

（１）既存のコードの修正

科学及び工学シミュレーションの分野で、コンピュータに関する様々な問題に取り組むためにソフトウェアの開発と改良に膨大な時間と費用がかけられてきた。それを批判する意見は、既存のコードを古臭い、誰からも愛されない“古いコード”として片づけてしまい、現在の高性能コンピュータには適さないと考えるが、FORTRAN言語及びＣ言語の何百あるいは何千ものラインを再構成（再利用）できるという実用性を見逃している。

時間的なこと、既存のソフトウェアやデータとの互換性、保守のしやすさ、使い方が容易で費用対効果も高いなどのたくさんの理由から、走り書きのコードを書き換えるよりは、むしろ既存のコードを並列プラットフォームにポートする方法がしばしば適用される。

ポート化は、アプリケーションの特質及びアルゴリズム構成に大きく左右される。シミュレーションコードの中には、この既存コードを直接、並列プラットフォームにポートする方法では処理しきれないものもある。例えば、広範囲に分配されたデータ間の包括的な相関関係を有するようなアルゴリズムの場合には、走り書きのコードを再構成した方が容易なこともある。しかし、多くのアプリケーションにおいて、既存コードを直接並列プラットフォームにポートする方法で適切に処理できる。詳細に検討設計し、分配処理の必要性をよく理解することにより、非常に効率性の高い並列処理が可能になると考えられる。

並列化において重要な問題は、書き換えコードに関する研究者の知識が曖昧であるということである。特定のアプリケーションにコードを使用する科学者たちは、そのコードを自分たちで開発していくであろうし、そのアプリケーションについての専門知識を有しているグループも、それを開発していくであろうが、コンピュータサイエンスの分野では必ずしもそうではなく、いくつかの科学的、及び技術的問題が引き起こされる可能性がある。

例えば、並列コードの分野には詳しくないコンピュータ科学者によって書き換えられた時、書き換えられた、もしくは並列化されたコードが妥当なものであるかについて非常に疑わしいこともあり得る。並列化に適されるコードがそのように書き込まれるため、一般に並列化テクノロジーをこのような問題に適用するのは非常に困難であると考えられる。

（２）データ・パラレル・プログラム事例（パラミックス・プロジェクト）

パラミックス・プロジェクト独自の研究とは、既存のソフトウェアをデータ並列超高速計算プラットフォームへ転送するための標準化を図ることである。パラミックスは、英国政府運輸局のリンクプログラムの一環として設立され、その目的は、交通コンサルタントSIAS社（英国エジンバラ）によって開発された微視的交通シミュレーター（Micsim）を変換して、エジンバラ並列コンピュータセンターの大量データ並列接続機CM-200で処理操作が行えるようにすることであった。

パラミックスは、ある流体モデルを使用し、交通量の増減のシステムをシミュレーションするよりは、むしろ交通渋滞の様子、あるいはドライバーの反応などをより正確に把握するために微視的なシミュレーターによって、１台１台の車両をモデル化するためのものである（いわゆる、マルチエージェント協調計算である）。

これは、コンピュータ技術の上では、非常に高レベルな手法である。少なくとも、リアルタイムで

交通システムを、スコットランド全体の幹線道路網の規模にモデル化することができるようになるためには、超高速計算テクノロジーの適用が必要となる。パラミックスで使われている交通網は、リンクで接続されたノード（ジャンクションにある）のシステムから成り立つ。それぞれ一方向のリンク（すなわち各車線）は多くの車両の列に細分され、これらの列が並列処理されるデータの基礎となる。これらの車両の列は、すべて同じ大きさで、それぞれの列が１つのプロセッサに割り当てられる。モデル化が繰り返されるたびに（例えば、ジャンクションで車両が列を変更する時）プロセッサ間で情報の伝達が行われ、列における車両の状態が更新される。

このように、同じ大きさの列で設定した車両を細分化することによって、16、384ビットのシリアルプロセッサを有するCM-200の並列データトポロジーに適切に配置することが出来る。

当初、Micsimのコードは、シンキングマシンの並列データC言語を使用しCM-200にポートされていた。Micsimが、50ジャンクションの交通網で200台の車両をシミュレーションできたところでは、パラミックスの並列データならば、3,000ジャンクションにもわたる交通網で250,000台の車両をシミュレーション処理することができる。これによって、シミュレーションスピードが最初のソフトウェアを使用した時と比較して実に1,000倍以上も高速になったのである。

大量のデータを並列処理するシミュレーションに加えて、パラミックスの主たる特性は、リアルタイムで視覚化ができるということである。グラフィック・ワークステーションが直接シミュレーターと接続されていて、精巧な視覚化が可能となり、交通網のスナップ写真をその場で見ることもできる。それぞれの車両の追跡も可能で、交通渋滞のパターンを直接モニタリングすることや、ユーザーのフィードバックをシミュレーターに戻すことも可能である。高性能並列コンピュータと視覚化が結びついたことは、超高速計算が有効に活用されていることを示すものである。

パラミックスの最初のプロジェクトが素晴らしい成功を収めたことから、さらに研究開発が進められており、メッセージ送信に基づいたバージョンや、優れた渋滞管理機能を有したものなどの開発が進んでいる。プロジェクトそのものの地道な努力から、1994年には、スコットランド戦略情報技術大賞を、1995年にはコンピュータワールド／スミソニアン大賞を受賞するに至った。さらに、パラミックスは、特別に創設されたパラミックス交通シミュレーション社の庇護を受けて、英国だけでなく世界中でそのシステムを商品化することに成功している。

（３）タスクファーム（AEAのMascot）

それぞれ独立した計算に分類できるようなアプリケーションにおいて、タスクファームの手法は必要に応じて並列化するための優れた方法になることがある。タスクファームは、マスタープロセスがワークロードをそれぞれのタスクに分割して、それらのタスクを１つひとつのスレーブ・プロセスに割り当てる働きをしている。

Mascotとは、AEAのデコミッショニングとラドウェストによって書かれた、モンテカルロ法による安全性評価プログラムである。その主な特徴は、いくつかのケースが与えられた時に、結果的にそれぞれ安全性がどのように評価されるのか、いろいろと算出できることである。Mascotは、最初クレイY/MPハードウェアを使って処理操作が行えるように設計されていたが、ベクトル化ができないという特性が本来備わっていたために計算効率が低い。計算効率化を高める方法として、RISCワークステーションと同一機種のクラスターで処理操作できるように設計し、並列バージョンで処理操作を行え

るようにすることであった。

エジンバラ並列コンピュータセンターによる、この並列バージョンのpMascotは、マスタープロセッサに入出力及び制御機能を行うようにさせ、任意の数のスレーブプロセッサに計算させるものである。タスクファームに基づいた並列化作業のフレームワークを迅速に始動させるために、エジンバラ並列コンピュータセンターによって設計されたタスクファームライブラリ（PUL-TF）が使われている。コードには、さらに異常公差機能が導入され、それにより、中断時間内に完了されなかったタスクが別のスレーブプロセッサで処理されるようになった。

このタスクファームの手法をpMascotに適用した結果、並列コードがたった2つのSGIワークステーションでY/MPコードよりも約4倍も高速になったことから、タスクファームの手法のpMascotへの適用が成功であったことは明白である。4台のワークステーションでpMascotを処理操作させた際、処理時間を削減（Y/MPでは70分かかっていたのが50分になったという例が100もある。）することができた。さらに、タスクファームモデルが本来スケーラブルであるため、pMascotは高いレベルの並列効率で処理操作された。

（4）非構造化データの分解方式 （BAeのFlite 3D）

コンピュータによる流体力学のシミュレーションは、最新の航空宇宙業界の設計プロセスにおいて今や欠くことのできない重要な要素となっている。使用されるアルゴリズムの忠実度、及びモデル化される問題の規模は、ここ20年の間に劇的に向上してきた。このことによって、高い計算能力への需要がますます高まり、スケーラブルな並列構成が必要になってきている。

英国航空宇宙局（BAe）によって開発されたFlite 3Dソフトウェアは、立方格子に基づいた非構造化計算格子を使用して、複雑な航空機ジオメトリにおける流動をシミュレーションするものである。非構造化格子は、複合した流動の解法が算出できるかもしれないという点で、コンピュータによる流体力学にとっては素晴らしいアプローチといえる。

収束を促進するために、目の粗い格子で解法を算出する多重グリッドのような手法は、計算効率を高めるために使われる。より大規模の格子や細部にわたるジオメトリにおいては、大規模並列システム並みの能力とメモリーを有する計算能力が必要となる。

BAeとの共同研究でエジンバラ並列コンピュータセンターによって開発された並列Flite3Dソフトウェアは、領域分割方式を非構造化格子コンピュータに適用し、スケーラブルな並列コードをつくりあげている。Flite3Dへの並列化の適用には、次の2つの重要な領域がある。

> **領域1：包括的な非構造化格子を重複していないサブドメインへ最初に分配すること。**
> **領域2：分配された格子サブドメインを管理し、包括的解法において一貫性を保持すること。**

最初の段階は、多くのコンピュータ・アルゴリズムやツールを有するグラフ分割をしてみることである。並列Flite3Dにおいては、このグラフ分割は、既存のシリアルプロセッサのプログラムに組み込まれていて、分配された格子を説明するファイルをつくる。次の段階は、分配された静的な非構造化格子コンピュータをサポートするために設計された、並列計算ライブラリやMPIベースのPUL-SMを使って、流れ計算部を並列化することである。PUL-SMは、初期化の機能やファイルから格子を分配する機能、サブドメインの境界における最新の流動変数を保持する機能、そして流動のチェックポイ

ントや最終的な結果を詳しく書き込む機能を有している。

Flite3Dにおける完全に独立した格子の多重グリッド機能は、各レベルの多重グリッドを管理するため、ライブラリ・ユーティリティを多重使用することにより並列バージョンに保たれる。サブドメインの境界で必要とされる、プロセッサ間のコミュニケーションが最小限になるように、様々なグレードの格子間で変数が補間された。

このようにライブラリが使われることによって、Flite3Dのソースコードへの変換が最小限に保たれ、確実に軽量化された。Flite3Dの最終的な並列バージョンは、ネットワーク化されたワークステーションとクレイT3D MPPシステムの両方で、T3D特定最適化のコンパイル時間のフラッグを使って処理操作された。

試験においては、この領域分割バージョンは、飛躍的にスケーラブルになり、T3Dプロセッサの場合には64分の速度を示し、128のプロセッサでは80分の速度アップを達成した。クレイY/MPで処理操作されるシリアルコードと比較しても、この領域分割バージョンは大成功を収め、Y/MPより20分程高速にするためには、並列コードは8個のT3Dプロセッサを必要とする。メモリーモデルを見ると、256のプロセッサにおいて並列Flite3Dが4,000万テトラヒドラ以上の包括的格子を管理できていたことがわかる。

（5）地下水の流動のモデル化(AEAのNammuとNapsac)

地下水の流動をコンピュータを使ってシミュレーションするのは、地層処分ヴァーチャル・エンジニアリング・システムの重要なアプリケーションの1つである。有限差分や有限要素の手法に基づいて、このシミュレーションをするために、いくつかのコンピュータパッケージプログラムが開発されている。

数値計算をするためには、多大な計算能力を必要とするいくつかの方程式が使われるようになる。地下水の流動のパッケージは、今のところ限られたユーザーにしか使われておらず、ユーザーからの需要が限られているために複雑な問題をシミュレーションすることはできない。

並列処理問題として、地下水の流動をシミュレーションモデルを取り上げたものが、AEAテクノロジーによってつくられたNapsac及びNammuプログラムである。これらのコードは、地下水の流動、亀裂及び気孔性媒体における核種の移動などをそれぞれモデル化するために使われる。AEAテクノロジーとの共同プロジェクトによって、エジンバラ並列コンピュータセンターは、より大規模でより複雑なモデル化を可能にするため、これらのパッケージプログラムの並列バージョンを開発した。

どちらのコードも非常に複雑で何万ラインものFORTRANコードを有している。それらのコードは同一のコンピュータを共有しており、そのため、エジンバラ並列コンピュータセンターは、このコンピュータを並列化を中心にした研究に利用している。

並列化は領域分割方式に基づいて設計が行われた（すなわち、どのプロセッサにもモデルセクションが割り当てられている。）。各サブドメインにおける並列計算、単一のプロセッサを使った順次計算、そして最終的な並列計算の3つの段階でコンピュータによる全体的な処理操作が行われるようにソフトウェアが再構築された。このようにして、直接的な計算方法が保持される一方で、並列化によって処理速度が早くなっている。メッセージ交換（MPI）用の標準ライブラリを使ってプロセス間のコミュニケーションも行われた。この作業を完成するのに一人の人間の年齢ぐらいの年月が費やされ

た。

その結果、適度の数のプロセッサを使うと能率的に処理が行われることがわかり、データを能率的に分解することの必要性が強調された。このことはまた、例えば直接的な計算方法よりは、むしろ反復型の計算方法を使うなど、たくさんのプロセッサを使って適切に問題処理できるような解法アルゴリズムについての研究をさらに深めていくことが必要であることを示していた。

（6）並列多重ブロック（BaeのRansmb）

Ransmbのソフトウェアは、官用航空機を設計するために、Bae地域航空機AVRO部門によって開発され使用されている。Flite3Dソフトウェアと同じように、Ransmbもまた、複雑で全体的な航空機ジオメトリの空気の流れを計算するために設計されたコンピュータによる流体力学シミュレーションコードである。

このようなアプリケーションの重要な利点は、風洞を使わないで、空気力学、航空力学上の設計性能を直接予測することが可能になるかもしれないということである。しかしながら、必要なターンアラウンドタイム（総処理時間）内に必要なレベルの精度で処理操作するためには、非常に高い性能の並列コンピュータが必要とされている。

Ransmbは、そのコンピュータモデルに多重ブロック手法を適用している。航空機のジオメトリは、いくつかの別々のブロックに分けられており、それぞれのブロックは、カーテシアンメッシュで別々にグリッド化されている。そして、ブロック単位に計算が行われ、着実に収束解法に移行していくのである。それぞれの最新データは、ブロックが処理される範囲の及び隣接するブロックにあるセルの“halo”からのデータを必要とする。ソフトウェアは、収束を促進するための多重グリッド機能や広範囲にわたるモデリング手法などが特徴であり、そのモデリング手法には層流や乱流などのモデリングも含まれている。

このモデルには元来並列機能があり、各プロセッサは与えられた一式のブロックを使って処理される。並列設計の重要な要素は、プロセッサ一式のロードバランスを保ってブロックをグループ分けすることである。ブロックがいったん分割されると、ブロック単位に処理が行われる。ブロックを更新するためには、隣接するブロックからのデータが必要で、そのためにプロセッサ間のコミュニケーションが必要となる。すべてのプロセッサにおける、すべてのブロックが更新されると、必要なデータがプロセッサ間で転送され、次の更新が始まる。

Ransmbの並列バージョンは、AVROの協力を得てEPCCによって開発されたもので、クレイT3D MPPシステムで使われることを目的にしていた。性能が優先事項の1つであったために、コミュニケーションルーチンはT3D固有の共有メモリーライブラリを参考に設計されていた。受入テストの結果、充分なロードバランスが保持されていれば、このアプローチによって性能がスケーラブルになることがわかった。

このようにして、Ransmbがスケーラブルな並列解法をつくりだしたことによって、エンジニアたちは、クレイT3Dシステムなみの処理能力と記憶容量を享受することができるようになる。このことは、現行の問題の総処理時間が短縮したという意味では、直接的な生産性が向上したことを示し、以前よりも複雑なジオメトリをより正確に算出する能力が高まったことも示している。

３．４　並列計算プログラムの開発

問題を速く解くための１つの方法は、その問題を細分化し、すべての部分を同時に解くようにアルゴリズムを組み立てることである。細かく分ければ分けるほど、問題の解決は速くなるが、分散させる効果が無くなるまでに細かく分け過ぎては問題そのものの意味を失う。このような、簡単かつ明快な観察が、すべての並列プログラミングの基本となっている。また、これは、２章にて論じたオブジェクト指向シミュレーション・モデルの考え方にうまく適合する。

「並列プログラム（parallel program）」は、大きな、または難しい問題を解くために、細分化手法を使ったプログラムである。ソフトウェアを組み立てるということは、組織的または技術的な挑戦である。並列プログラムは、多くのプロセッサを同時に使って実行することを目的としている。それぞれのプロセッサは、問題の一部分を分担して、全体が同時に実行を進める方式である。１つの問題を解くのに複数のプロセッサを使って、1台のプロセッサの場合の複数倍の性能を実現することが並列計算の狙いである。現実には、このような理想的な線形な性能向上をいつでも実現できるわけではないが、並列処理は非常に多くの種類の現実の重要なプログラムを高速に実行するための、確実な方法と考えられる。

並列プログラムを実行する複数のプロセッサ（ハードウェア）は、すべてが１個の箱の中に実装されていてもよい（この場合には、マルチプロセッサまたは並列処理計算機となる）し、またネットワークで結ばれた独立した自律的な計算機の集合であってもよい。マルチプロセッサのプロセッサ台数は、数台から数千台にまで亘るであろう。また、これらのプロセッサを接続するには多くの方式がある。ネットワークはLANでもWANでもよく、ネットワークの中の計算機はどのようなものでもよい。

並列処理の特徴に、「非同期並列処理（asynchronous parallelism）」がある。非同期並列処理とは、（おおまかにいうならば）それぞれのプロセスは好きな処理を実行できるような並列処理のことである。これに対して、「同期並列処理（synchronous parallelism）」はすべての処理が同期を取って進められ、基本的には同時には同じ処理しか実行できないという特別な方式である。

ソフトウェアの面では、一般に、ソフトウェアを複数プロセスの集合として組み立てる理由は、いろいろあるといえる。並列処理による高性能の達成というのは、その１つにしか過ぎないであろう。また、性能向上以外の狙いを持ったマルチプロセス・プログラムがある。それは、協調ソフトウェアと呼ばれ、それはコンピュータサイエンスの分野以外にも見られるものである。

世界は全体的にみれば、非同期集合を集めて作られた非同期集合である。社会システム、経済システム、機械システム、生物システムは、すべてこのように考えることができる。人間の集合も、プロセスの集合が使っているのと同じソフトウェア構造によって説明できる。協調ソフトウェアの芸術と科学を扱う１つの方法は、協調システム一般の解析というより、広い課題への小宇宙的な紹介とすることである。これは、新しいモデル化の戦略であるとも言える。

協調ソフトウェアは、多くの独立した活動から、いかに１つの仕事が合成されるかという研究のための場でもある。

並列プログラミングにおいては、計算言語と協調言語の両言語が必要である。計算言語で記述された個別の活動の集合を、統一されたプログラムに組み立てるための鍵（キー）となるのが協調言語である。従って、計算やローカルなデータオブジェクトの処理には、計算言語（Fortran、C、lispなど）を使う。並行実行、ならびにそれらの間での通信の実現には、協調言語を使う。

また、標準的な計算言語の機能と協調言語の機能を1つの言語に含めることもできる。計算言語の例はC言語である。協調言語の例はLindaである。

（1）並列性

プログラムの実行を高速にするための実証された手法が並列処理である。どの時代においても、最も強力な計算機は並列処理計算機であると思われる。何故ならば、最善を尽くして、その時点での最高の技術を使って、最も高速のプロセッサを設計したとすれば、それを2台使えばより高速のシステムを構築できるのは明らかであるためである。

他方、中位レベル、または最下位レベルのシステムが上げられる。おおまかにいえば、時分割処理計算機やファイルサーバ、またはワークステーションやパーソナルコンピュータ（この2つについては急速に差異がなくなりつつある）が、それぞれ対応する。

製造メーカの立場でいえば、素子、通信バス、電源、端末、オペレーティング・システムなどのハードウェア、およびソフトウェアの基本要素を揃えたならば、1台のプロセッサのシステムのみにとどまっているのは意味をなさない。同じ匡体にプロセッサボードを挿入するというのが、システム性能向上のための最も価格・性能比のよい方法であることが多い。

初期の並列処理計算機は、一般に中位から高レベルの市場を狙っていた。BBNバタフライ、シークェントバランス／シンメトリ、InteliPSC、アライアントFX／8、アコアマルチマックス、nCUBEなどは皆この範疇に属する計算機である。それに比べ、最近の計算機になると、さらに高性能でより低価格になっている。例えば、シリコングラフィックス社、ヒューレットパッカード／アポロおよびゼニス社の新しいマルチプロセッサシステムや、インモス社のトランスピュータを使ったシステム（メイコ、コジェント、コーラス）などがこの範疇に属する。

一方、計算機ネットワークは、並列処理のより大切なベースといえよう。1つの箱の中に納められているプロセッサでは、LANや広域ネットワークで接続されたプロセッサの間よりも、より高速の通信を実現できるのが普通である。ネットワーク並列計算では、難しい問題を持ってきて、これを部分問題に分割し、それぞれの部分問題を複数個のプロセッサのどれかに割り当てる。これらのプロセッサが1個の箱に納められているのか、はたまた国中に分散しているのかの違いには左右されない。

もちろん、並列処理計算機の上では効率よく実行できるが、ネットワークではうまくいかないといったプログラムも当然存在する。また、多くのプログラムは環境によらず効率よく実行できるであろう。

たとえば、最新の超大型スーパーコンピュータ（これ自身並列処理計算機であるかもしれない）でも解けないような問題があったとしよう。そのときには、次の方法としてスーパーコンピュータのネットワークに進むことになる。考えられる資源は、超高速スーパーコンピュータそのものではなく、超高速スーパーコンピュータ全部（またはネットワークに接続されているだけのスーパーコンピュータ）の集合である。

あたかも1台の並列処理計算機のように、この集合を単一の問題を解くのに使うことを考えることができる。近年各所の事務室または研究室には、ネットワークでつながれた沢山のワークステーションやパソコンがあり、並列処理のための絶好の（多分、最も並列処理に適した）環境になっているであろう。

それらのローカルエリア・ネットワークの全ノードの計算能力を加えると、それはとてつもないものとなることがある。ノードの能力が大きくなればなるほど、そしてノードの数が大きくなればなるほど、全体の能力も大きくなる。能力を発揮させるには、多くのノードを使って並列処理を行わなければならない。もし、多くのノードがやることがなく空き（アイドル状態）になる時間が一日の中にあれば、その時にそれらのノードの能力を集めることによって、一時的に並列処理計算機を構成できる。

しかし、多くのノードが使われているときでも、経験によれば（平均的には）ほとんどはそれほど高負荷で使われているわけではない。計算能力を十分必要とするような問題を持っている利用者でも、常に計算機を能力限度まで使っているわけではない。

例えば、ある大きさの組織で、100台のワークステーションまたはパソコンを接続したネットワークがあるとしよう。また勤務時間中には、これらの計算機は平均的に半分だけ使われているとする。これは、組織が勤務時間中に（平均的に）50台のプロセッサからなる並列処理計算機を持っていることと同等になる。

このことは、それぞれの利用者が机上に並列処理計算機を置いたことと同等である。机上にどのような計算機を置くにしろ、その計算能力を拡充するための控えとして、ネットワークを活用することができる。ネットワーク全体で使われていない計算能力を集めて、現在必要な仕事に回すことができる。並列処理プログラムは、逐次処理プログラムが持っていないこのような特別の拡張性を持つことになる。より多くのプロセッサを持っていれば、より多くを使うことができる。

このようなネットワークの計算能力のプールを利用して並列計算を行う方法をネットワーク並列（又は、マシン・パラレル等）と呼ぶ。

今あるプログラムをより速く実行させるためには、新しい計算能力が必要である。さらに、新しい種類のソフトウェアをサポートするにも、そのような能力が必要とされる。より計算能力の大きな計算機があれば、より大規模な問題をより詳細に、かつより正確に解くことができる。

また、より大きく複雑なシステムのシミュレーションをより良く行うこともできる。さらに、より細かく複雑な科学計算の可視化も可能になる。科学者や技術者は、計算分子動力学や天気予報、気象解析、飛翔体設計および他の多くの問題の進展のために、これらの技術を活用するであろう。

また、並列処理計算機は、複雑な画像処理ならびに高品質の音声や手書き文字認識にも使える。科学・技術分野以外の最も大切な並列処理の用途は、データベースであろう。複雑な探索条件に従って大規模なデータを高速に検索するためにも、並列処理は役に立つであろう。さらに、並列処理は実時間アニメーションないし仮想現実感の応用も広げよう。航空電子工学や、流れ込む入力データを処理する高レベルモニタプログラムなどがその例である。

（2）協調

これからのソフトウェア開発の現場では、「すべてのプログラマにとって、並列処理は基本的なものとなる」であろう。さらに、「協調がコンピュータサイエンスの中心的な話題になる」であろう。並列化技法について学ぶということは、協調プログラミングの一種について学ぶということであると考えられる（並列技法の多くは、他の種類の協調応用にも文字通り適用可能である）。

協調プログラミングについて学ぶということは、すべての計算の基本を学ぶということにもなる。

個々の「エージェント」を接合してプログラムを作って行くというプロセスを指して協調と呼ぶこととする。

ここでいうエージェントとは、例えばオブジェクト、プロセスやタスク、またはスレッド（thread）など、それ自身で独立した実行主体のことである（それ自身で独立した実行とは、他と並行かつ非同期的な実行のことである）。

個々のエージェントを協調させるということは、接合してできあがった集合をプログラムとみなすことができるということである。協調させられたエージェントは同じ問題を処理している。協調に当たっては、これらの独立な活動体がお互いに必要に応じて通信したり同期をとったりできるようになっていなければならない。協調言語は、このような要求を満たすように設計されている。

協調は概念的にも重要である。協調は、ソフトウェアの空間の基礎となる２つの直交座標の１つとして扱えるからである。有用な計算機システムを構築するには、計算と協調が不可欠である。すべての計算は、最終的には外部世界とつながらなければならない。つながれるということには、協調するということが含まれている。計算機とその利用者は、その計算が何か意味のあることを行うことを目的としている限り、お互いに協調関係になければならない。

この簡単な観察の帰結は次のとおりである。オペレーティングシステムは、見方によっては、特別の協調言語といえる。また、利用者／計算複合体は協調そのものである。さらに、ソフトウェアプロセスと人間の「プロセス」の両方が協調するためには、これを実現するための新しいソフトウェアが必要であるということになる。ここで「新しい」といったが、これは必ずしもその例すら私たちが分かっていないということを意味はしているわけではない。ただ、その概念がまだ具体的に一般化されてはいないということである。

このソフトウェアを、「**チューリングウェア**（Turingware）」と命名する。人間とプロセスが、普通の協調プログラミング手法によって、単一構造の中に織り込まれる。この複合体の中では、プロセスも人も、お互いに自分の相手がプロセスなのか人間なのかを意識しない点に、新しい情報モデルが考えられる。

UNIXのコマンドインタプリタのようなシステムは、場合によっては端末のスクリーン（ということは究極的には人間ということになる）と普通のファイルを区別なく扱うことができる。その意味で、これは「チューリングウェア」ともいうべき性格を持っているともいえる。しかし、究極のチューリングウェアにおいては、人間は単にデータ供給者の片方というだけではない。人間はネットワークのノードとして、積極的に計算に関与する。

言い方を替えるなら、システムのモジュールの１つは人間そのものである。目標は３つある。まず最初の目標は、速くて柔軟で高機能な情報の交換の枠組みを作るために、並列プログラミングの手法を採用することである。なお、この枠組みのことを、「協調枠組み」と呼ぶことにする。例えば、統合化された地層処分ヴァーチャル・エンジニアリング・システムはチューリングウエアであるべきである。

並列プログラムと分散システムは、どちらも協調プログラムである。並列プログラムは高速実行のために、並行性を利用する。分散データベースや電子メールおよびファイルサーバのような分散システムは、計算機が物理的に分散配置されているので、これを活用するために並行プロセスを使っている。50台のワークステーションから構成されている分散メーラシステムでは、それぞれのワークステ

ーションの上で１つずつのプロセス、総計50のプロセスが走行している。しかし、この目的はなにも１台の計算機上でのメーラ（Mailer）の50倍の性能を実現することではない。協調が目的である。

協調プログラムを構成する構成要素は、それぞれ１つのプログラム言語で書いてもよいし、またいくつかの異なったプログラム言語で書いてもよい。言葉を替えるならば、協調プログラムは多様な言語のプログラムでもある。協調プログラムの構成要素は、距離的にもまた時間的にも離れていてもよい。２つのプロセスが同時に２つの異なった計算機で実行され、一方が他方に情報を送る場合には、その情報は空間をまたがって送られることになる。もし、この２つのプロセスが同じ計算機で実行される場合でも（例えば一方が火曜日に実行されて、他方が木曜日に実行される）、空間的に分散した場合と論理的に同じ協調問題となる。「火曜日プロセス」は「木曜日プロセス」に情報を送るが、これは空間をまたがった情報伝達ではなく、時間をまたがった情報伝達である。結果は、同じように協調プログラムとなる。「時間をまたがった協調」言語は、普通はファイルシステムまたはデータベースの形をとることが多い。このため、時間をまたがったプロセス集合と空間をまたがったプロセス集合とは違うものという認識をしがちであるが、基本的な概念は全く同じである。並列処理という範疇で考えられた多くの協調手法は、そのままファイルシステムやデータベースシステムに適用できる。

これからは、利用者は小さいプログラムを組み合わせて大きいプログラムを作るという場面により頻繁に遭遇するであろう。このような手法は、これまでの技術の集積を補うものである。コンピュータネットワークは、もはや特別のものではない。ネットワークを統合システムとして扱うことには、むしろ望ましい。

今後は、どのプログラミング言語をとっても、それだけでよいということにはならない。少なくともある程度の期間は、多数の言語が存在し続けるであろう。多数の言語が混在したプログラムを扱えることは、明らかに望ましいことである。特殊目的のハードウェアも現われ出した。ここにグラフィックマシンとデータベースサーバと汎用計算ノードがあれば、これら３つの計算機をすべて使うような応用を考えるのは、きわめて自然なことである。そして、プロセッサの数は増え続けるであろう。プロセッサが安価でたくさんあるというような環境では、協調プログラミングは必須である。応用プログラムを実行するよりも、たくさんのプロセッサがあるような時代になれば、１台のプロセッサしか使えないようなプログラムは完全に非社会的となるであろう。そのような面から、今後の統合化システムの開発において協調計算技術の開発は重要な課題である。

３．５　協調作業のための基本パラダイム

並列アルゴリズムを使ってどのようにプログラムを作るのだろうか。その過程において、次の３つの基本的な事項に特に注目すべきである。それらは、結果並列法、手順並列法、および専門家並列法である。これらの言葉の意味は後で定義する。これらの３つを、並列性の基本パラダイムとする。これらは、並列性について考える上での、また並列プログラムを設計する上での、さらに並列性を解析する上での、３つの異なったアプローチとなる。

これらの基本パラダイムに対応して、３つの並列プログラミング手法（概念を実用的なプログラムに変換する実際的な手法）がある。それらは、メッセージ交換、分散データ構造、およびライブデータ構造である。それぞれのプログラミング手法は、並列プログラムにおけるプロセスの役割とデータの分散を異なった視点から見ている。基本パラダイムとプログラミング手法は、必ずしもこれだけに

限られるものではない。しかし、ここでの目的は、協調枠組みに適用できる概念的なパラダイムとプログラミング手法を理解し、それらの間の関係を調べることである。

（1）パラダイム

並列プログラムは、それぞれのパラダイムに対応して、自然なプログラミング手法がある。それぞれの手法を、お互いに明確に関連づけることができる（言葉を変えていうならば、「手法xを使ったプログラムを手法yを使ったプログラムに、次のような明確と定義された方法で変換することができる」）。したがって、並列プログラムを書くには、問題を最も自然に表わすパラダイムを選び、そのパラダイムに最も合った手法でプログラムを書く。もし、できあがったプログラムが十分に効率的でなかった場合には、自然な手法の代わりに更に効率のよい手法を考えプログラム変換をする。

並列性を、プログラムの結果、プログラムの動作の手順、およびプログラムを構成する専門家の集合という3つの観点から捉えることができる。例えば、家を建てたいとする。このときに、大勢の人が働く並列処理を採用するのは当然の成り行きである。しかし、並列処理が実際に活用される場面はいくつかに分かれる。まず、並列処理を最終の生産物（結果）によって考えることができる。結果は多くの個別の構成要素に分けられる。前面の壁、後ろの壁、両脇の壁、内部の壁、土台、屋根などがある。結果をこれらのように構成要素に分類すれば、それぞれの構成要素の製造を並行して進めることができる。そして構成要素の製造が終わり次第、それらを組み立てていく。土台の製造に作業者を一人、前面の外部壁の製造に作業者を一人、さらに両脇の壁の製造に作業者を一人といったように作業者を割り当てる。すべての作業者は、作業を同時に開始する。別々の作業者が、土台の据付や外部壁の組み立ておよび屋根の棟上げなどに従事する。他人の作業の終了を待ち合わせることが必要になるまで、それぞれの作業者は並列に作業を進める。まとめると、それぞれの構成要素を作る作業に対応して一人ずつ作業者が割り当てられる。そして、他の作業の終了を待つことが必要になる時点（これがどこかは、仕事の内容によって自然に決まる）まで並列に作業を進める。これを**結果並列法**という。

他の考え方として、並列性を仕事をする作業者の観点から眺めることができる。家を建てるには、それぞれの技術を持った作業者が必要である。例えば、測量士、土を掘る人、基礎を作る人、大工、屋根ぶきなどである。家を建てるのに必要な、それぞれの技術に対応する専門家を集めて、建築グループを構成する。建築は全員で一緒に始めるけれども、最初は、ほとんどの作業者は手持ち無沙汰で待つことになる。しかし、建築が進むにつれて、多くの作業（したがって作業者）は一緒にできるようになる。大工と基礎工は一緒に並行して働くことができるようになる。また、屋根ぶきは、鉛管工が管路を設置し電気工事屋が配線をしているかたわらで屋根をふくことができる。大工は木工作業をするが、その間に多くの仕事が並行して進められる。このような進め方は、特に流れ作業的（パイプライン）な仕事に適している。流れ作業では、1つの製品が次から次へとラインを進んでいくことによって完成されていく。もし、数軒の家を建てるのであれば、大工が1軒目の仕事をしている間に、基礎工は2軒目、そして測量士は3軒目の仕事を、それぞれすることができる。このような作業法は、1軒の家を建てる場合のように製品が1個しかない場合でも並列性を引き出す方法として適用できる。作業者は、それぞれ特定の仕事に割り当てられた専門家である。仕事の性質からの制限に抵触しない限り、お互いに並行して作業する。このような並列性を**専門家並列法**という。

最後に、家を建てるときにやらなければならない仕事の手順という観点から並列性を眺めてみよう。仕事のリストを書き出して作業者を集める。作業者はそれぞれ何か仕事を選んで、それを始める。仕事が終わったら、作業者は次の仕事を選択する。作業者にはそれぞれ特別な区別はない。作業者は、初めから、予めどれか特別の仕事に割り当てられているのではない。必要になったら、どのような仕事でもする。仕事の手順に従って仕事が進む。作業者は必要に応じて、何回でもこの手順を参照することになろう。手順に現われる仕事としては、お互いに順序関係のない仕事の集合もある。この場合には、その集合の中から好きな仕事を選ぶようにと作業者に指示することになる。また、仕事の間に順序関係がある場合もある。その場合には、まずこれを行い、次にこれに取りかかれといった指示をすることになる。まず、土台がいる。土台の構築は、手順の中では仕事の集合として書いてある。次に骨組みがいる。次は屋根である。そして、壁やしっくいへと進む。「何でも屋」のチームを作る。メンバは誰でも何でもできる。まず、みんなで集まって土台を作る。次に同じメンバで壁を作る。さらに屋根へと進む。次に何人かは鉛管の配管をする。その間に残りの人間は電気工事や他の仕事をする。誰が何をするかは、全く任意に決められる。まとめると、手順書から仕事を選んでは作業者をそれに割り当てて、その仕事に従事させるということを全部の仕事が終了するまで繰り返す。作業者は、仕事上の制限にでくわすまでお互いに並列に作業を進める。このような方法を**手順並列法**という。

このようなパラダイムは、抽象的な対象について整理することを要求するが、統合化システム開発における基本的なアーキテクチュア設計思想となるため、システム設計の初期の段階で充分に検討されなければならい課題と考えられる。これらの並列のパラダイムの区別は、時には曖昧になることもあると思われる。

また、これらを混ぜ合わせた方法も考えられる。例えば、専門家並列法に手順並列法を併用することを考えてみよう。専門家のチーム（例：大工）が、大工仕事を手順並列法に従って行うというのが、その例である。いずれにせよ、上記の３つのパラダイムは、問題を扱う上で明確に区別できる３つの考え方である。それは、例えば地層処分性能評価のような複合問題を対象とするシステムを開発するときの問題解決の基本戦略である。

以上のように、結果並列法は最終製品の形態に注目した考え方である。専門家並列法は作業者の組合せに注目した考え方である。そして、手順並列法はするべき仕事の表に注目した考え方である。

これまで、家の建築を題材に話を進めてきた。次の（そして最終的な）課題はソフトウェアである。しかし、これらのパラダイムは、人間のありとあらゆる活動にまんべんなく適用できると考えられる。これらのパラダイムのソフトウェアに関する適用性は下記の様に考えることができる。

- **最終結果となるデータ構造体を中心に並列処理を考えることができる。**
  結果のすべての要素について同時に計算することによって、並列処理を達成できる。
- **複数のタスクの処理手順を基に並列性を考えることができる。**
  複数のタスクの実行に、多くの作業者を割り当てる。
- **ある種の論理的なネットワークを構成している専門家の集合に基づいて、並列性を考えることもできる。この論理的なネットワークのノードの中で同時に実行できるものがあれば、それが並列性になると考えられる。**

それでは、どのパラダイム（どの並列性）を使うかということは、どうしたらわかるであろうか。

再び、家の建築について考えてみよう。結果としては、3つのすべてのパラダイムが使われる。あらかじめ工場で部品（壁、屋根、階段など）を作っておいてこれを現場で組み立てる場合には、これらの部品は、全てバラバラにそして（理論上は）同時に工場で作られる。

これは、結果並列法そのものである。他方、「納屋作り」においては、作業者のグループは必要な仕事のリストに出てくる項目を次から次とこなしていくことになる。これは、手順並列法の一形態である。しかし、最も普通の方法は、手順並列法を採用したある種の専門家並列法である。それぞれの作業者（またはチーム）は専門を持った専門家である。そして、それぞれの専門家が同時に仕事を進めていくことによって、並列性が生まれる。そして、同じ専門家チームの多数の作業者が、手順に従って作業することによって2番目の手順並列法が生まれる。

ソフトウェアの場合にも、問題によっていずれかのパラダイムが他よりも自然な選択となる。選択は問題に依存している。場合によっては、選択は自然である。また場合によっては、全部のパラダイムが等しく自然である。このように複数の選択の余地があることが、混乱と当惑の基になることが多い。しかし、これはむしろ並列性に富んでいてプログラマには並列性の選択の幅が広い場合の多いことの証ととらえるべきである。多くの場合、並列プログラムを設計する最も容易な方法は、結果のデータ構造体を基にすること（結果並列法）であると思われる。

- 自分のプログラムは、結果として多数の要素を持つデータ構造体を出力するのか（またはこのような構造体を基に結果を考えることができるか）を考えてみる。
- そのデータ構造体の各要素が、お互いにどのような関連があるのか、また入力とどのような関連があるのかを正確に指定できるかを考えてみる。もし、これが可能であれば、結果並列法に基づいてプログラムを書くことは（適当なプログラム手法を知っているものとして）簡単である。おおまかには、そのようなプログラムは、次のようになるであろう。

（データ構造体を、＊＊のように構成する。この構造体の全要素について同時に計算する。すべての値が求められたら終わりとする。）

結果の構造体の要素は、お互いに依存しないこともある。そのような場合には、すべての計算は同時に始まって並列に進行する。または、ある要素の計算は他の要素の値が決まらないと始められないかもしれない。そのような場合には、全要素の計算は同時に始まるが、要素によっては計算は遅れることになる。

そのような要素については、それが依存している要素の計算が終わるまで、計算を開始できないことになる。例えば、n要素の2つのベクトルAとBがあるとする。そしてこの和を求めるとする。このプログラムは次のようである。

- n要素のベクトルを作る。
- Sのi番目の要素の値は、AとBのそれぞれi番目の要素の値を加えれば求められる。

Sの要素はお互いに完全に独立である。加算は他の加算に依存しない。結果として、計算は並列に実行できる。結果並列法は、構造体と依存関係があらかじめ明確な結果の列を求めることが目的である問題に対しては、非常によい出発点である。しかし、すべてのプログラムがこのような条件を満たすわけではない。例えば、結果の形式や様式が入力に依存するようなプログラムを考える。テキスト

の体裁を整えたりコードを翻訳したりすることを並列に行うプログラムがその例である。これらのプログラムの出力は、その大きさが、前もってわからないようなバイト列であるとか表の集合である。また、その他の例としては、１つのオブジェクトが繰り返して演算を施されるような処理（例えば線形代数のLU分解では与えられた行列が何回も演算される）がある。さらに、数値演算結果を目的とするのではなく、処理の効果を得ることを目的とするようなプログラム実時間制御プログラムやオペレーティングシステムもその例である。

手順並列法は、多くの異なった問題に容易に適用できる非常に便利な方法である。この形式の並列法の最も柔軟な実現が、マスターワーカー・パラダイムである。マスターワーカー・プログラムでは、マスタープロセスが計算を始めて、同一のワーカープロセスの集合を生成する。それぞれのワーカープロセスは、計算のどの部分でも実行できる。ワーカーはするべき仕事を探して、その仕事を実行するといったことを繰り返す。仕事が残っていなければ、プログラム（またはこのステップ）は終わりである。プログラムは（少なくとも１個はあれば）ワーカーの数によらず、同じように実行を進める。同じプログラムが引き続いて、1個、10個また1000個のワーカーで実行されるかもしれない。もし仕事が即座に分散できるならば、負荷の平滑化も自然に行われることになる。あるワーカーが時間のかかる仕事をしている間に、他のワーカーが多くの簡単な仕事をすることもある。

例えば、社員の人事データベースを考えてみよう。このデータベースから、被扶養者当たりの給料が最も低い社員を探すとする。レコード0が与えられたときに、前記の給料・被扶養者比を計算する関数をr（Q）としよう。手順は以下のとおりである。

データベースのすべてのレコードに関数rを適用して、rが最小のレコードの識別子を求める。

このプログラムを次のようにマスターワーカー・プログラムにすることができる。マスターは袋にデータベースのレコード・オブジェクトを全部入れる。それぞれのオブジェクトが、一人ひとりの社員レコードに対応する。各ワーカーは、順にレコードを袋から取り出して、rを計算して、結果をマスターに返す。マスターは、それまでに求められた最小値を覚えておき、すべての仕事が終わったときに結果を報告する。

専門家並列法は、論理ネットワークを基本にして考えることのできるようなプログラムに適用できる。このネットワークのノードは、比較的自律的な計算に対応しており、ノード間の通信はあらかじめ決められたパスを通る。対象とするプログラムやシステムの振舞いをネットワークで表わすと、その振舞いがよく理解できるような場合に向いているわけである。ネットワークは、物理的なモデルに対応していることもあるし、またアルゴリズムの論理的な構造に対応していることもある（パイプラインまたはシストリック計算がその例である）。

物理的なシステムがあって、これをモデル化するときには、ネットワークを使う方法は特に自然な方法である。例えば、回路シミュレーションを考えてみよう。それぞれの回路素子を独立したプロセスで実現して、これを基に並列プログラムを作ることができる。その他にも、それぞれ独立した責任単位に自然に分解できる問題がある。単位間には明快な通信路がある。さらに、このような構成に基づいた「協調専門家」型のヒューリスティックなモニタについても考えることができる。

例として、全国規模のトラック輸送会社が、道路状況や天気および交通事情などを基に、２地点間の輸送時間を見積るということを考えてみよう。特別の並列プログラムを次のように構築する。

アメリカ大陸の地図を論理ネットワーク化する。各州はこのネットワークのノードとなる。ワイオミングノードは、同州の交通事情や通過時間などの情報を常に最新のものに保持する責任をもつ。ニューハンプシャーからアーカンサスまでの輸送時間を求めるためには、まず経路を決めて、トラックに対応するデータオブジェクトにこの情報を渡す。トラックは、ニューハンプシャーに行って、そこを通過するのに必要な時間を見積もってもらう。トラックは経路に従って、通過する州を順々に訪問する。最終的には、トラックはアーカンサスに到着して、そこまでの必要時間を出力する。多数のトラックが同時に、このネットワーク上を動いていることもあることが重要な特徴である。

パラダイムの調査のまとめとして、最後に2つの特別の場合について述べる。これらの特別の場合とは、「データ並列」と「推論並列（OR並列と呼ばれることもある）」である。データ並列は制限付き手順並列法であり、データ構造体のすべての要素について同時に演算を施していく。もし、それぞれの手順がデータ構造体への演算であるならば、結果としての手順並列プログラムはデータ並列処理の例になっているであろう。

経験的には、データ並列処理は、普通は同期型の並列処理計算機で採用されている。この方式ではデータ構造体の全要素について、同時に、そして同期して演算が施されることになる。すなわち、各時点において各ワーカーは与えられた構造体の一部に対して、同じ作業の同じステップを行う。

推論並列は、論理プログラミングで使われることが多い方法であるが、ヒューリスティックサーチの並列アルゴリズムとしても大切である。この方法では、並列に行われる仕事の中のいくつかは、最終的には不要となるかもしれないという前提の基に並列処理がなされる。プログラムの中に、「ｘをまず試して、もしｘが失敗したらｙを試す（そしてさらにその他の選択のリストが続く）」といったような文が出てきたら、ｘとｙそしてその他の選択をすべて同時に試すことによって並列処理を行う。

もしｘが失敗しても、ｙはすでに実行を開始している。これは、われわれの分類法では手順並列法の特別の形に対応する。多くのワーカーがリストにある作業に同時に投入される。ただし、これらの作業のうち1つの作業の結果だけが最終生産物に使われる。

本項では、並列処理のパラダイムについて検討してきた。その結果、並列処理技術は単に計算速度の高速化を目的とする算法ではなく、むしろ、協調計算システムのアーキテクチュア設計に係わる基本的な技術手法と考えることが妥当であると思われる。また、そのような立場から並列処理を捉えた場合、並列処理は協調計算の基本パラダイムとして以下の3つのアプローチが考えられる。

- 結果並列法　：対象を構成要素の並列性から組み立てる
  例えば、現象全体を個別の現象結果を並列させて説明する
- 専門家並列法：対象を処理の並列性から組み立てる
  例えば、現象全体を個別の現象過程を並列に連成させて説明する
- 手順並列法　：対象と手順（シナリオ）の並列性から組み立てる
  例えば、現象全体をそれを生じるシナリオを並列に組み合わせて説明する

以上のように整理すると、協調計算に係わる並列処理パラダイムは、地層処分性能評価そのものに対するパラダイムであり、重要な考え方を示唆していると考えられる。それはまた、性能評価手法という枠組みを組み立てるための、具体的な方法論を情報処理モデルによって組み立てる可能性を示している。従って、本件は、オブジェクト指向におけるパラダイムとともに、今後の地層処分性能評価

シミュレーション・システム開発における重要な研究開発課題である。

（3）プログラミング・パラダイム

メッセージ交換では、多くの並列プロセスを生成して、それぞれのデータ構造体をどれかのプロセスに内包させる。プロセスはメッセージの交換によって通信する。メッセージ交換法では、プロセス間で共用されるデータオブジェクトはない。

それぞれのプロセスは、自分のローカルデータ・オブジェクトにアクセスすることしかできない。通信のためには、プロセスはローカルスペースから他へデータオブジェクトを送る必要がある。このためには、プログラマは明示的にプログラムの中にデータ送信とデータ受信命令を組み込まなければならない。

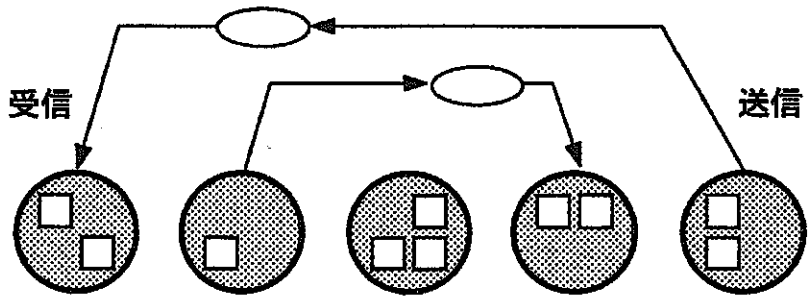

図３．５－１　メッセージ交換の考え方

図３．５－１は、メッセージ交換でプロセスの数と、それらの間の関係がプログラムの構造を決定する様子を示している。並行プロセスの集合はメッセージを交換して通信する。それぞれのデータ・オブジェクトは、いずれかのプロセスの中に包含される（プロセスは円で、データオブジェクトは四角で、メッセージは楕円でそれぞれ示されている）。

もう一つの考え方としては、プロセスを独立した単位としてみなすことをやめて、結果として得られるデータ構造体の形でプログラムを作成する方法がある。この場合、このデータ構造体の各要素が、それぞれ独立したプロセスに相当すると考える。

計算が終了したところで、これらはデータオブジェクトとなる。これらのプロセス相当のデータは、メッセージを送って通信するのではなく、それぞれデータ構造体の要素として「参照」し合う。もしプロセスPがQのためのデータを持っているならば、PはQにメッセージを送るのではなく、Pは処理を終了して結果を出す。Qはそれを直接読み出す。このような仕組みをライブデータ構造体プログラムという。この考え方は半逐次的なデータフローをモデル化することができるので、ネットワークモデルによる平行処理などを考える場合に都合が良いと思われる。

このような情報処理モデルでは、データ構造とその要素の数、そしてそれらの間の関係がプログラムの構造を決定する。それぞれの並行プロセスは、いずれかのデータ・オブジェクトの中に包含される。それらは、その要素の通信についてのみ責任を持っているため、通信のための明示的なメッセー

ジの送受命令は使わない。他のプロセスが生成した値が必要になれば、そのプロセスが内包されているデータ・オブジェクトを参照する。

メッセージ交換もライブデータ構造体も、それぞれすべてのデータ・オブジェクトが並列プロセスに分散されているという点においては類似している。メッセージパッシングでは、プロセスはプログラマによって明示的に生成されて、通信も明示的に行われ、データを繰り返して他のプロセスに送ることができる。他方、ライブデータ構造体では、データ構造体を作っていく過程において、プロセスが暗に生成される。またデータ構造体の要素を参照することによって、通信も暗に行われる。それぞれのプロセスは、プログラムの他の部分から参照される１個のデータを作るだけである。

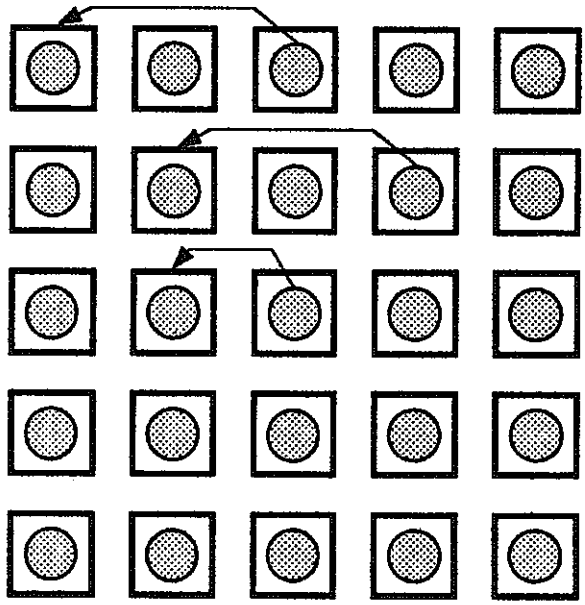

図３．５－２　ライブデータ構造体の考え方

データ構造体が、すべてプロセスに内包されるという考え方（メッセージ交換）から、プロセスがデータ構造体に融合するという考え方（ライブデータ構造体）の間には、データ・オブジェクトとプロセスの集合間の区別をする中間的な方法がある。

データ共用のために、多くのプロセスがデータオブジェクトや構造体に直接アクセスをする。共用データオブジェクトがあるので、共用オブジェクトにデータを残すことによって、プロセスは通信して協調することができる。これを分散データ構造体プログラムという

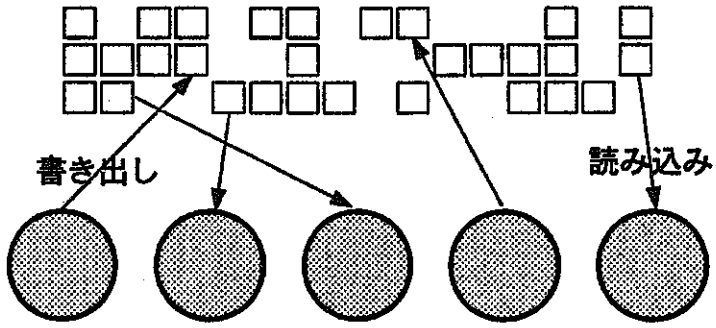

図３．５－３　分散データ構造体の考え方

分散データ構造体の考え方は、並行プロセスとデータ・オブジェクトがプログラム構造の独立した部分となると捉えることである。プロセスは共用データオブジェクトを読み書きすることによって、通信すると考える。図３．５－３にその考え方を示す。**結果並列法**は、ライブデータ構造体プログラムで自然に表現できる。例として、ベクトルの加算を考えてみる。この応用の中核はライブデータ構造体である。ライブデータ構造体はSというn要素のベクトルである。Sの各要素の中に、それぞれの位置についてA[i]＋B[i]を計算するプロセスがある。プロセスが終了すると消滅して、結果のみが後に残る。

専門家並列法はメッセージ交換が利用しやすい。プログラムを組むには、それぞれのネットワークノードに対応してプロセスを設けて、メッセージでエッジ間の通信を実現する。例えば、先の旅行時間の計算の問題に戻ってみよう。まず、論理ネットワークのノードをプロセスとする。そしてトラックをメッセージにする。ニューハンプシャーからネットワークにトラックを走らせるには、ニューハンプシャーに「新しいトラック」というメッセージを送る。このメッセージには、トラックの経路に関する情報が含まれている。ニューハンプシャーでは予測旅行時間を計算して、新しいメッセージを次のノードへ送る。このメッセージには、経路とこれまでの経過時間が含まれている。多くのトラックがあれば、多くのメッセージが同時にプロセスに蓄積することも有り得る。

プロセスがこれらのメッセージを捌くためには、メッセージのキュー（待ち行列）またはバッファ機構が必要になる。多くのメッセージ交換システムには、何らかのバッファ機構が埋め込まれている。そのようなネットワークのモデルがあるとしても、分散データ構造体の採用なしにはメッセージ交換だけでは不十分なことが多い。ネットワークの各ノードが共通（グローバル）の状態変数にアクセスしなければならないとすると、（分散データ構造体がなければ）これらの共通変数はどこかのノードのローカル変数として扱うしかない。すべてのアクセスは、この管理プロセスを介することになる。このような方法は概念的にも不適当で、またボトルネックの原因ともなる。

手順並列法は分散データ構造体に自然にマッピングできる。手順並列法では、多くのワーカーが実効的に１つの仕事にとりかかる。一般に、ワーカーは誰でもどの仕事でもできる。あるワーカーの仕事の結果を他のワーカーが使うというのが普通である。しかし、ワーカーは他のワーカーが何をしているかを知らない（また気にもしない）のが普通である。このような状況では、結果を分散データ構

造体に残しておくようにするのが便利である。それを使いたいワーカーは、誰でも、そこから取り出せる。こうすれば、特定の相手にメッセージを送ることを心配しなくともよいことになる。

さらに、マスターワーカープログラムの振舞いを考えてみよう。マスターワーカープログラムは、手順並列法の最も柔軟な実現結果である。ワーカーの集合があり、これらのワーカーにタスクを割り振りたい。どこにタスクをしまっておけばよいだろうか。もし手順にあるタスクが並列でそれらの間に何ら順序関係がないとすれば、マスタープロセスは、タスクディスクリプタの記述を分散バッグ構造体にしまっておくことができる。ワーカーはバッグを覗き込んで、タスクを取り出す。場合によっては、（多くが並列に実行できるとしても）タスクをある順序に並べておかなければならないこともあろう。この場合、タスクはキューまたはストリームに格納しておくことになる。

例えとして、マスターワーカーモデルに基づいた並列データベース検索について述べた。マスタープロセスが社員のレコードをしまうバッグは、分散データ構造体として実現するのが自然である。この構造体は、ワーカープロセスとマスターが直接アクセスできる構造体である。

（2）n体問題へのアプローチ

前述してきた並列化パラダイムについての例として、簡単なn体問題シミュレータを考えてみる。シミュレーションの繰り返しごとに、各物体と残りの物体間の作用力を計算し、これに基づいてそれぞれの物体の位置を更新する。この問題を家の建築の問題と同じように扱ってみる。結果並列法、手順並列法ならびに専門家並列法の3つの方法がある。

結果並列法では、問題を次のように言い換えることができる。n体を与えられて、シミュレーションをq回繰り返したいとすると、問題は、j回目の繰り返しの後のi番目の物体の位置が、要素M[i,j]であるような行列を計算することになる。行列の0列が、それぞれの物体の初期位置であり、最後の列が最終位置である。ライブデータ構造体を使うための第1ステップの設計はこれで終了する。

第2ステップは、それぞれの要素の値を他の要素の値を使って更新する処理を定義することである。繰り返しjにおける物体iの位置を計算する関数を、position(i,j)と書くことにしよう。当然position(i,j)は1回前の繰り返しにおけるそれぞれの物体の位置に依存する。

それは、行列のj-1列の値に依存する。適当なプログラミング言語を使って、この処理を書けばよい。M[i,j]の値を、position(i,j)で計算するプログラムを書く。関数positionを呼び出すごとに、暗にプロセスが起動される。そして、すべてのプロセスの起動と実行が並列に行われる。もちろん、1番目の列の計算が終わらなければ2番目の列の計算は始められない。

関数positionが、まだ計算の終わっていないM[x,y]の値を必要とする場合には、その計算が終わるのを待たなければならない。0番目の列の値は初期値として与えられる。これを使って1番目の列の値は並列に計算できる。そして2番目以降の列の計算も同様に進む（図3．5－4参照）。

もし物体の間に働く力が対称であれば、上のプログラムは必要以上の余計な計算をしていることになる。物体AとBの間の力は、物体BとAの間の力に等しいからである。この問題は簡単に解決できるが、ここでの目的は最も簡単な方法を示すことにある。

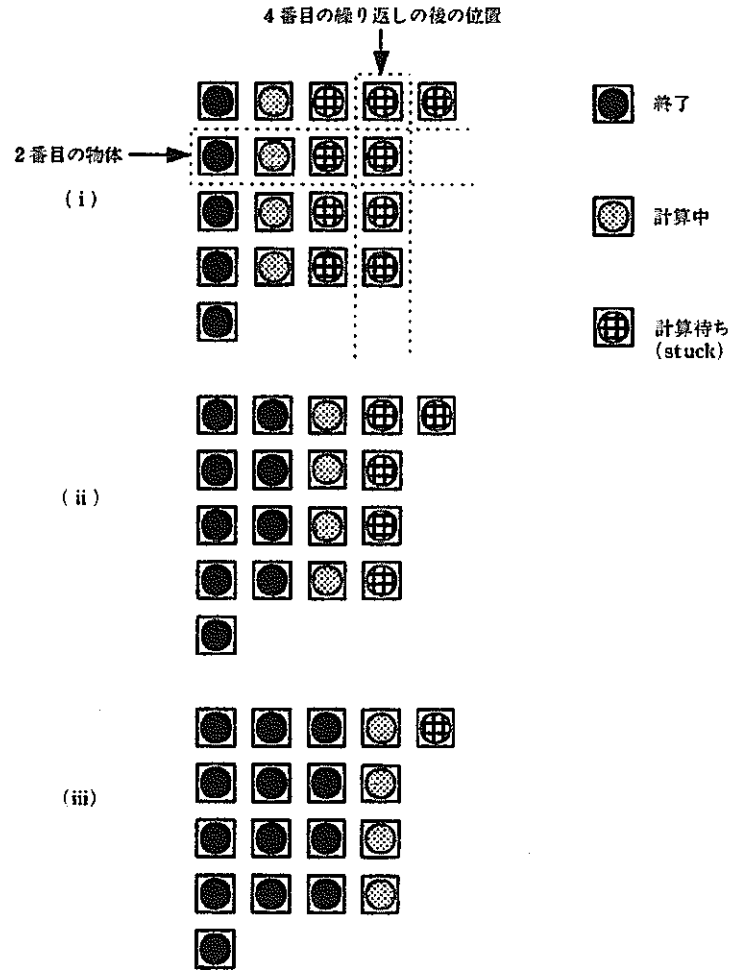

図３．５－４ ライブデータ構造体によるn体問題解決へのアプローチ

始めにnxqの行列を作り、それぞれの要素の中にプロセスを作る。要素M[i,j]の中のプロセスは、それぞれの物体の最後に計算した位置が入っている前の列の値を使って、j番目の繰り返しの後のi番目の物体の位置を計算する。列jのプロセスは列j-1のプロセスが終了するまで実行を開始できない。列j-1のプロセスが終すると、列jのプロセスは並列に計算できる。以上の手順で行列全体の計算が終了するまで、各列の計算を並列に続けることができる。

また、手順並列法によって問題を解決することもできる。仕事の手順は、**「集合に含まれているすべての物体について、次の位置を計算するという処理を繰り返し適用すること」**である。プログラムを書くには、まずマスタープロセスを作って、次にそのマスタープロセスにn個の初期タスク・ディスクリプタ（各物体に対応している）を生成させる。

最初の繰り返しでは、ワーカープロセスの集合の中のワーカーはすべて平等であり、タスクディスクリプタが無くなるまで、次々にワーカーはタスクディスクリプタを取る（これによって、すべての物体は新しい位置に移ったことになる）。この同じことを繰り返す。ワーカーが１個しかない場合には、繰り返しごとの計算時間はnの二乗に比例する。２個になると、これがnの二乗/2になる。

最後の繰り返しの結果の物体の位置情報は、分散テーブル構造体にしまう。これをそれぞれのワーカーが直接参照する。

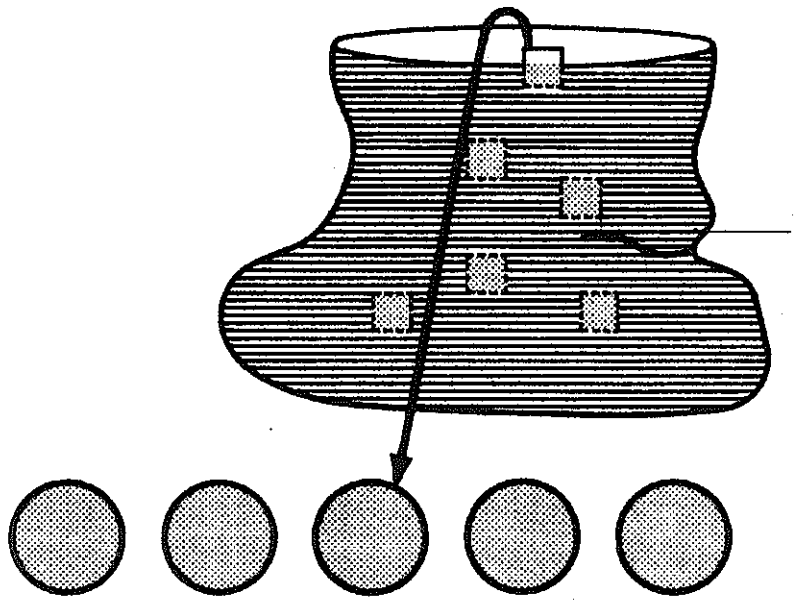

図３．５－５　分散データ構造体によるによるn体問題解決へのアプローチ

それぞれの繰り返しの後で、ワーカーは分散バッグからタスクを取り出して、対応する物体の新しい位置を計算する。各物体の最後の位置に関する情報は分散テーブルに入っている。それぞれの計算の後で、ワーカーは表（テーブル）を更新する方法もある。最後の位置に関する情報は、後で必要になるかもしれないので消さないでおく。

また、新しく計算されたデータをマスタープロセスに送る方法もある。この場合には、マスタープロセスが各繰り返しの後で、テーブルを一度に書き換える。

最後に専門家並列法では、まずプロセスの列を生成する。それぞれのプロセスは、それぞれの物体に対応している。すなわち、プロセスは、シミュレーションの間、それぞれ特定の物体の位置の計算に責任を持つ。各繰り返しを始めるに当たって、プロセスはそれぞれメッセージを使って、自分が担当している物体の位置を通知する。

プロセスはすべて同じ振舞いをする。繰り返しの初めに他のプロセスにデータを送るとともに、他のプロセスからデータを受け取る。メッセージの中にはプロセスが分担している物体の位置を計算するのに必要なデータが含まれている。このような処理を繰り返す。

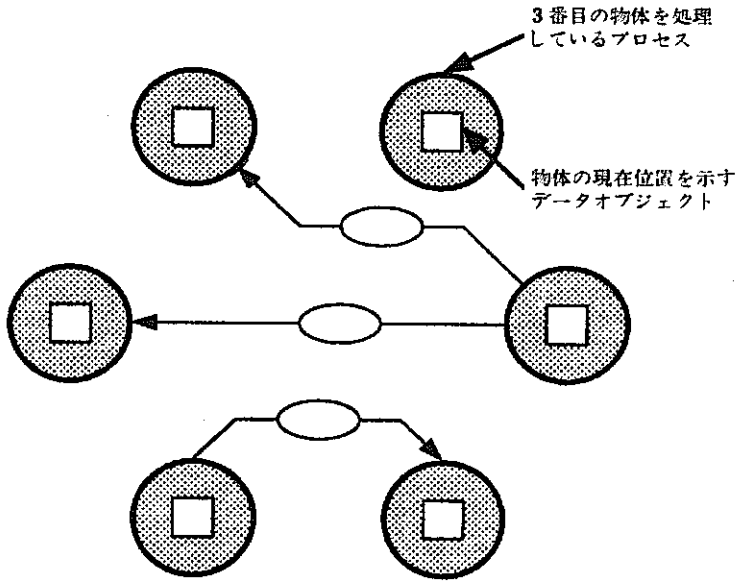

図３．５－６　メッセージ交換によるによるn体問題解決へのアプローチ

ライブデータ構造体プログラムはnq個のプロセスを生成（qは繰り返しの数で、nは物体の数）し、分散データ構造体プログラムは、好きなだけのワーカーを生成するが、このメッセージ交換プログラムはn個のプロセス（物体1個当たりプロセス1個）を生成する。

他の２つの方法では、プロセスが各物体の最後の位置に関する情報を必要とするときには、グローバルデータ構造体を参照した。また、ライブデータ構造体版では、このグローバルデータ構造体はライブデータ構造体で、その中にプロセスが包含されていた。メッセージ交換版では、プロセスは自分の外部のデータオブジェクトにはアクセスしない。プロセスはメッセージを送り合って、情報を交換する。

（３）パラダイムと効率

これまでに述べてきたプログラミング手法は、問題に則したパラダイムを選んで、そのパラダイムに適したプログラミング法でプログラムを書き、もし必要ならば、そのプログラムをより効率のよい他の方法を使ったプログラムに修正するというものであった。

もし、自然な方法が効率のよいプログラムにつながるのであれば、もちろんプログラムの修正は必要ない。そうでなければ、手法と効率の間の関係を理解することが必要となるためである。

並列協調処理では、取り扱うエージェントの数が増えるにつれて、計算量が爆発的に増大することが考えられる。従って、一度作り上げたアルゴリズムをより効率的なものにすることは重要な課題である。ここでは、この関係についての一般論について述べた後で、この効率向上のための修正について考える。

これまで述べてきたパラダイムの主な関係を図３．５－７に示す。

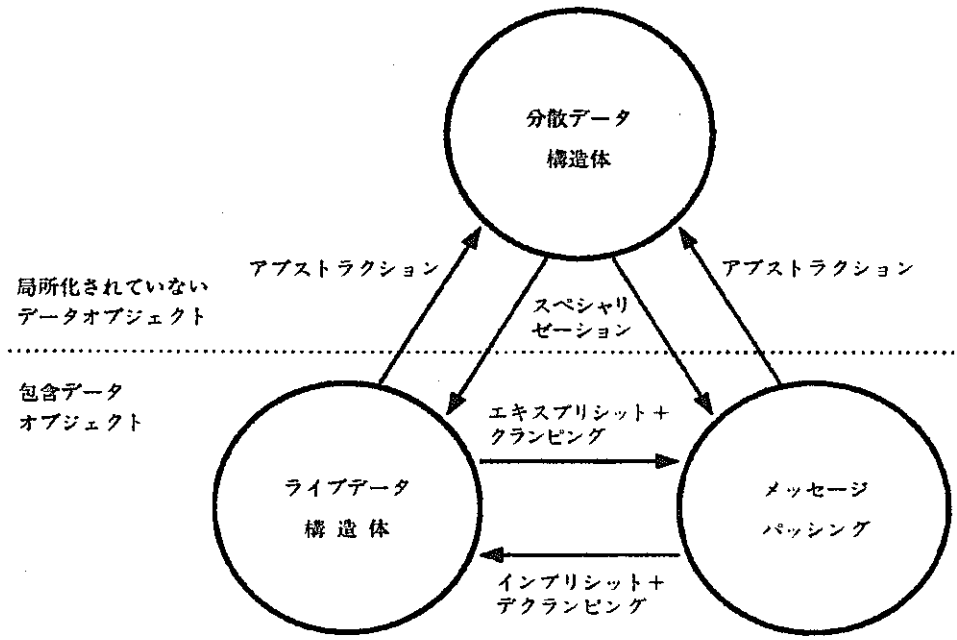

図３．５－７ 並列パラダイムの関係

ライブデータ構造体とメッセージ交換は、いずれも束縛データオブジェクト（captive data object）に基づいている。すべてのデータオブジェクトは、いずれかのプロセスに結び付けられている。分散データ構造体手法は、非集中データオブジェクト（delocalized data object）に基づいている。

オブジェクトは、いずれのプロセスにも結び付いていないで自由である。ライブデータ構造体またはメッセージ交換プログラムを分散データ構造体に変換するには、アブストラクション（abstraction）を使えばよい。アブストラクションは、データオブジェクトをそれが結び付いているプロセスから切り離して、分散データ構造体に入れる。こうすることによって、プロセスは、もはや単一のオブジェクトやオブジェクトの集合に注意を払う必要はなくなる。

分散データ構造体からライブデータ構造体やメッセージ交換プログラムに変換するには、スペシャリゼーション（specialization）を使う。スペシャリゼーションは、それぞれのオブジェクトをどれかのプロセスに結び付ける。

以上の考察から、ライブデータ構造体とメッセージ交換の間には緊密な関係があることがわかる。ライブデータ構造体をメッセージ交換に変換するには、通信を明示的に行う必要が出てくる。このためにクランピング（clumping）を使うことがある。ライブデータ構造体を使ったプログラムにおけるプロセスは、他のプロセスへ情報の明示的な通信を行う必要はない。単に終了して、結果としての値を生成するだけである。

メッセージ交換プログラムでは、データを送るプロセスは明示的に「メッセージを送る」命令を実行しなければならない。ライブデータ構造体プロセスが他のプロセスからのデータを使うときには、そのデータ構造体を参照する。これに対してメッセージ交換プロセスでは、「メッセージを受け取る」命令を明示的に実行しなければならない。

メッセージ交換が、ライブデータ構造体の単なる言い換えにしか過ぎないとしたら、なぜ後者から前者への変換にこだわるのであろうか。メッセージ交換手法は、クランピングを通じてより大きな自由度を提供する。ライブデータ構造体プログラムのプロセスは、１個の結果を計算して消滅する。それ以上に、別の結果を計算して提供することはできない。従って、メッセージ交換では、プロセスは好きなだけ結果を計算することができる。そして、それらを好きなときにメッセージとして送ることができる。

プログラムが実行されている間、結果を作り続けることができる。ここにクランピングの意味がある。また、１個のメッセージ交換プロセスに、全部のライブデータ構造体プロセスの仕事をさせることもできる。

（4）アブストラクションとスペシャリゼーション

以上のように、いくつかの変換について述べたが、それらの利点は何であろうか。図３．５－７では単純なネットワークの上で、いろいろなパスを選択することができる。それらのすべての詳細について今後の研究が必要である。

例えば、結果並列法が最も自然に適用できそうな問題があったとしよう。これに対して適当なライブデータ構造体プログラムを書いたが性能は悪かったとする。そこで何らかの変換が必要になったとする。まず、なぜこの例を特に取り上げたのであろうか。問題が適当であれば、ライブデータ構造体プログラムは設計も簡単で表現も適切にできよう。また並列性も大きくなるであろう。もちろん並列性の程度は、結果の構造体の大きさと要素間の依存度による。しかし、ライブデータ構造体手法によるプログラムは往々にして、プロセスの数も多くなり、それぞれのプロセスは極く少しの処理しか行わないといった細粒度の並列性（fine grain parallelism）になることが多い。したがって、最新の並列処理計算機の上では性能が悪くなりがちである。

例えば、結果としてデータ構造体の大きさが10,000要素もの配列になったとすれば、この方法では10,000ものプロセスが生まれることになる。もちろん、このようなプログラムを効率よく実行できないという理論的な理由はない。しかし、大部分の最近の並列処理計算機では、多数のプロセスを生成して協調させるにはオーバヘッドが大きすぎる。これは特に分散メモリ計算機では顕著であるが、軽量プロセスが実現できるとされている、共用メモリ計算機においてすら無視できないのが実情である。

せっかく並列処理をしても、多くのプロセスが細粒度の計算をする結果、並列処理の効果はなくなってしまう。もしライブデータ構造体プログラムを効率よく実行できるのならば、それはそれでよいが、アブストラクションを使い、同じアルゴリズムを分散データ構造体プログラムに変換することによって、もっと効率のよいプログラムに容易に変換することができる。ライブデータ構造体を受動的な構造体で置き換えて、プロセスを概念レベルで１段上のものとする。それぞれのプロセスを単一の要素に対応させるのではなく、多くの要素の面倒をみるようにする。

例えば100個のプロセスを生成して、それぞれのプロセスに100個の要素を計算させる。その結果、元のプログラムよりも粒度の大きいプログラムとなる。どれだけのプロセスを生成するかについては、プログラマが適当な数を選ぶことができる。このようにして、多数のプロセスを作ることによるオーバヘッドを避けることができる。

そのようにして作ったプログラムでもまだ十分に効率がよくないかもしれない。各プロセスから単

一のデータ構造体に読み書きしたとする。その時このデータ構造体は、どこかの共用メモリに格納しておかなければならない。共用メモリへのアクセスは、私用メモリへのアクセスよりも時間がかかる。最もこれは普通は問題なこともあるが、分散データ構造体が最適であることもいえる。

前述のn体問題について、ライブデータ構造体プログラムをアブストラクションやスペシャリゼーションの出発点とすることもできる。ライブデータ構造体プログラムはnxq個のプロセスを生成した。それぞれのプロセスは関数positionを１回だけ起動して終了する。

分散データ構造体プログラムをアブストラクションで生成してみる。Mを分散データ構造体とする。これは全プロセスがアクセスできる受動的なものである。0番目の列には各物体の初期位置を入れ、残りの列は空いている。k個のプロセスを生成する。それぞれのプロセスは、行列の一部を計算する。j番目の列を計算するには、プロセスはj-1番目の列に書いてある前の位置を参照する。

プロセスの数はプログラマが制御できる。もしそれが適当ならば、2～3個のプロセスのみを使うのでもよい。数台のプロセッサしかない場合である。このようにすればプロセス管理のオーバヘッドを小さくできる。元のプログラムから、この新しいプログラムを作ることは簡単である。新しいプログラムは元のプログラムに比べると、多少詳しさと理解しやすさには欠けるが、最後にスペシャリゼーションによって、最もオーバヘッドが小さいメッセージ交換プログラムを作ることができる。

それぞれのプロセスは、Mの一部を自分のローカル変数空間に格納する。Mは、もはや単一の構造体としては存在しない。プロセスは、最後に計算された位置情報に直接アクセスすることはできないので、これらの情報は、メッセージとして授受されなければならない。それぞれの繰り返しの後で、プロセスはメッセージを交換する。このメッセージには、プロセスが計算した最後の繰り返しの結果の位置情報を入れる。この結果、プロセス管理のオーバヘッドも小さくて、分散データ構造体を参照するオーバヘッドもなくなる。この最後のプログラムは、これまでのものよりもより複雑である。

プロセスは各繰り返しごとにメッセージを授受して、ローカル変数を更新しなければならない。また、このプログラムでは重要な概念上の変更がある。先の２つのプログラムでは、通信はデータ構造体の参照という位置付けであった。これは他のプログラムでも同じような概念である。これに対して、最後のプログラムでは、通信はメッセージ交換という形態をとる。これは標準的なプログラミング手法に比べると、概念的に全く新しいものである。

アブストラクションとスペシャリゼーションをいつ使うかという選択は、プログラミングの現実的なものである。それは応用分野、プログラムスタイルおよび並列処理計算機などに依存する。例えば、CLinda言語を使って最新の並列処理計算機の上でプログラムすれば、メッセージ交換に帰結するスペシャリゼーションを利用することは、ほとんど必要ない。ほとんどの問題は、効率のよい分散データ構造体プログラムにできる。この場合には、効率のよいプログラムを得るためには、分散データ構造体のためのアブストラクションが必須であるということである。

ライブデータ構造体を選ぶのが自然な場合、プロセスの数は多くてオーバヘッドも大きくなる可能性がある。それを避けるために、アブストラクションによって分散データ構造体プログラムに変換する。メッセージ交換ネットワーク型のプログラムが自然であることもあるが、プロセスの数も多くプロセス間通信も頻繁になる可能性がある。この場合、アブストラクションによってメッセージパッシングから分散データ構造体に変換できる。

例えば、10,000個の素子から成る回路をシミュレーションするとしよう。この場合、素子対応にプ

ロセスを設けて、素子間の信号伝達をメッセージの交換でシミュレーションすることが自然に見える。しかし、多分この方法はオーバヘッドも大きく性能も悪いであろう。そのような場合、アブストラクションによって、それぞれのプロセスが回路全体の状態を表わすデータ構造体の一部を分担するようにすれば、プロセス数を少なくできる。

3．6　システム環境

並列言語やシステムについての参考文献の多くは、並列または分散プログラミングを実現する統合並列プログラミング言語、統合分散プログラミング言語、またはオペレーティングシステムを扱っている。これに対して、Lindaは協調言語である。この違いは重要である。統合並列言語は、協調のための機能が組み込まれた完全に閉じた新しい言語である。協調をサポートするのはオペレーティングシステムである。

オペレーティングシステムは、入出力や数学ライブラリなどと同じように利用者プログラムから呼び出せるユーティリティルーチン（例：メッセージ交換）の集合を提供する。これらのルーチンはコンパイラの助けは使わない。

協調言語は協調モデルのみをサポートする言語である。新しいコンパイラ（例：C-Linda）は計算言語と協調言語を組み合わせた環境を実現する。現時点では、協調言語の製品はほとんどないがDongarraとSorensonのScheduleはひとつの例である。Scheduleは粗粒度のデータフローに関するBabbの研究に関連している。また、Strandは論理プログラムの観点からのアプローチである。今後、もっと多くの例がこの分野に出現するであろう。

メッセージ交換は基本モデルの中では最も広く普及している。いろいろな言語の中で形を変えて使われている。最もよく知られているのはHoareのCSP（Communicating Sequential Process）で、多くの人に影響を与えた。この結果生まれたのがOccamである。CSPもOccamも非常に緊密なメッセージ交換の概念に基づいている。メッセージの授受は同期動作である。メッセージを送ったプロセスは、相手プロセスがそれを受け取るまで動作を中断する。CSPもOccamも静的言語であり、プログラムの実行中にプロセスを動的に生成することはできない。このためCSPおよびOccamは、ここで論じているようなメッセージ交換プログラムをサポートする言語としては十分な表現力は持っていない。

モニタとリモートプロシージャコール言語も、次に述べる条件付きでメッセージ交換の範疇に分類できる。これらのシステムでは、通信はプロセスコールとしてモデル化されている。プロセスが、プロセスと通信するためには、どこか他のプロセスで定義されたプロシージャまたは受動的なグローバルモジュールのプロシージャを呼び出す。このような擬似プロシージャコールは、特別な形のメッセージ交換とみなすことができる。その方法では、プロシージャへの引き数がメッセージとなって送られ、結果も当然メッセージとして送られる。

それらの条件とは、場合によっては、このようなシステムは擬似分散データ構造体でも使われることがあるということである。共用データオブジェクトがモジュールに内包され、リモートプロシージャコールでアクセスされる。同じことはメッセージ交換システムでも可能であるが、プロシージャ形式のインタフェイスのほうがより便利である。

なぜ擬似分散データ構造体なのだろうか。分散データ構造体は多くの並列プロセスから同時にアクセスすることができる。もちろん場合によっては、データの正当性を保つために逐次アクセスを強制

しなければならないこともある。しかし、一般には多くの読み出し命令が同時に進行できる。さらに同じデータ構造体でも、異なった独立した部分に対する書き込み命令は同時に進行できる。配列の異なった要素への書き込みがその例である。

この範疇の言語は、プロセス間でグローバルなデータオブジェクトをサポートするが、一度には1個のプロセスからのアクセスしか許さないのが普通である。さらに単純な分散データオブジェクトもサポートしない。グローバルオブジェクトは、アクセスプロシージャの集合とともにパッケージ化する必要がある。モニタはHoareが最初に提案した。そして広く並行プログラミング言語の基礎として使われている。Concurrent Pascal、Mesa、Modulaなどがその例である。並列言語と異なり、並行言語ではプロセス間でのアドレス空間の共用はないことを前提としている。

ごく最近になって、モニタは並列オブジェクト指向プログラミング言語として、並列プログラミングで再び使われるようになってきた。リモートプロシージャコールはAdaの基礎となっている。Birrell とNelsonのRPCカーネルはシステムレベルでの効率のよい実現の例である。

メッセージ交換の他の変形はストリームである。送信側が（実効的に）メッセージストリームの最後にメッセージを付加する。受信側はストリームの先頭を調べる。このような通信法はKahnが最初に提案した。これは最近の並行論理プログラミング言語（ConcurrentProlog、Parlogなど）および通信機能が拡張された関数言語の基礎をなしている。

さらに多数の並列言語（例：Poker）やオペレーティングシステム（例：Vkerne、Mach、Amoebaなど）で通信法を実現するために、メッセージ交換はいろいろな形式で使われている。

それに比べると、分散データ構造体の使われ方は稀である。用語はLindaの中で初めて現われた。分散データ構造体は、並列doループを持つように拡張された特別のFORTRANの基礎としても使われることが多い。

例えば、JordanのForceがその例である。このようなシステムでは、並列性は主に並列ループー繰り返しが逐次的ではなく、並列に実行されるとして現われる。異なるループの繰り返し間での通信には、標準的なFORTRANの構造体を適用した分散構造体を使う。分散データ構造体は、何らかの形で、DallyのCST、BalとTanenbaumのOrca、KaleのChare Kerne、BrownのCODEなどで使われている。

またMultilispでもサポートされている。ライブデータ構造体は、いわゆるノンストリクトデータ構造体（完全に定義される前でもアクセスできる）をサポートしている言語で主に使われている。

Id Nouveau、MultilispおよびSymmetric Lispなどがその例である。

同じ考え方は、並列プログラミングのための多くの関数型言語（Sisal、Crystalなどが、その例である）の概念的な基礎をなしている。これらの言語を使ったプログラムは、名前の列にバインドされる値を定義する式の列からなる。式は同じ集合の他の式から返される値を使ってもよい。まだ他の式によって計算されていない値を使う式の計算は、それが終わるまで計算を始められないという制限を除けば、すべての式を同時に解くことができる。ライブデータ構造体を使った同じプログラムでは、それぞれの式を使ってライブデータ構造体の各要素の値を計算する。

３．７　並列問題として地層処分性能評価

前節までの検討で、並列処理技術は単に高速計算を目指した技術ではなく、ソフトウェアの協調作業により、対象となるモデルを組立て、問題解決にあたるという新たな方法論に期待できることが、

最も重要な関心ごとであることが解かった。また、並列処理モデル組立のパラダイムは、２章にて検討したオブジェクト指向を含む新しい情報処理モデルの概念により示されるものであるため、現在は研究中のパラダイムであると考えるべきである。しかしながら、前節までに概観してきた、手順並列、結果並列、専門家並列等の情報処理の考え方は、地層処分ヴァーチャル・エンジニアリング・システムのアーキテクチュア設計の根幹となる技術に触れているものと考えられる。

何故ならば、それらのパラダイムは非常に多くの新しい概念を含むものであるが、オブジェクト指向のパラダイムと同様に、多数プロセスとシナリオが混在している、地層処分性能評価手法開発に係わる情報処理面からの一つの可能性として期待できるためである。

ジェネリック・アセスメントとしての地層処分性能評価は、その枠組みを組立て、各プロセスを数理モデルとしてプログラムし、それらを協調させて実行し、結果を得るということを目的とした活動である。それをコンピュータ内部に組み立てられた仮想処分システムについて実行できるようにするには、システムのアーキテクチュアは、統合化手法として並列処理を取り入れることが必須となる。

（１）結果並列法：Result Parallelism

結果並列法の考え方は、モデルを構成要素の並列性から組み立てることである。例えば、現象全体を個別の現象結果を並列させて説明する場合が考えられる。この方法では、並列処理は最終結果となるデータ構造体について並列性を考える方法である。

この方法を性能評価シミュレーションのアーキテクチュアとして採用することを考えた場合、つぎのことが指摘できる。第一に、計算負荷の高い問題に対する速度向上による恩恵が期待できる。例えば、大規模３次元地下水流れ計算等では、問題を未知数の行列式等、計算結果の並列性の高い形で扱うことができるためである。従って、単体プログラムの計算速度の向上を計ることができるであろう。

連成問題においても、従来開発されてきた連成数理モデルから導かれるプロセスを並列化するという意味で高速計算が期待できる。いずれにしろ、結果並列法は、並列計算を考える場合の最も基本的な考え方であるが、これからのモデル化手法として期待できる協調計算技術開発を目指すというより、計算速度の向上を目指すべきである。

（２）専門家並列法：Specialist Parallelism

専門家並列法は、対象を処理の並列性から組み立てる。例えば、現象全体を個別の現象過程を並列に連成させて説明する。これは、問題がある種の論理的なネットワークで接続される専門家の集合として並列性を考える方法である。性能評価シミュレーションについて専門家並列法を考えると、それは、プロセス・システムの協調を計算を実現する手法として考えられる。

プロセス・システムは性能評価の枠組みを決める考え方である。それは、注目する事象を個別のプロセスに分解して、連成させて考える方法である。この方法を用いると、性能評価シミュレーション全体のアーキテクチュア概念をプロセス協調計算として扱うことができる。このとき、個々のプロセス・システムは専門分野として、地下水流れ、地球化学、岩盤力学、温度分布等、個別のプロセスが相互に協調してシミュレーションを進めるとして考えることができる。それらのプロセスはメッセージ交換を行って、平行して計算を進めるであろう。そのようなプロセスをコーディングするにあたり重要なことは、メッセージ交換のインターフェイスである。例えば、温度場が状態と変化させたなら、

そのフィードバックを岩盤力学プロセスは受けなければならない。そのためには、インターフェイスの統一や、メソッド、関数の仕様を統一する必要があると考えられる。

（3）手順並列法　：Agenda Parallelism

手順並列法は、対象と手順（シナリオ）の並列性から組み立てる。例えば、現象全体をそれに生じるシナリオを並列に組み合わせて説明する。これは、複数のタスク（エージェント等）の処理手順を基に並列性を考える方法である。この方法は、対象をネットワーク問題として考えることから始める。従って、地層処分性能評価では、FEP（Feature, Event, Process）から作成できるインフルエンス・ダイアグラムに見られるような処理を対象とするとき有効な並列化手法であると思われる。

実際には、この手順並列法に前述の専門家並列法を混在させて、インフルエンス・ダイアグラムを基準としたプロセス・システムの状態遷移シミュレーション・アーキテクチュアを考案することが理想的と思われる。

3．7．1　試作並列シミュレーション機能

地層処分ヴァーチャル・エンジニアリング・システムのアーキテクチュアを設計するための予備的な評価として、メッセージ交換による専門家並列機能を試作した。メッセージ交換仕様は、異機種ワークステーションとＰＣを混在させたネットワーク環境で、TCP/IPソケット、及びMPIを利用した。図3．7－1にその概念を示す。

図3．7－1に示すように、試作システムは2台のシリコングラフィックス社製ワークステーションと1台のサンマイクロ社製ワークステーション、及び2台のＰＣから構成した。これらのコンピュータは次のとおりに機能を（専門家として）分担している。第一に2台のシリコン・グラフィクス・ワークステーションであるが、一台は専用の3次元視覚化プロセッサとし、他方をシステム全体を操作するコンソール・プロセッサとした。また、サン・ワークステーションは専用のデータベース管理プロセッサとした。残る2台のＰＣはシミュレーションの進行状況を表示する、専用のパラメータ表示プロセッサと選択シナリオを表示するシナリオ表示プロセッサとした。

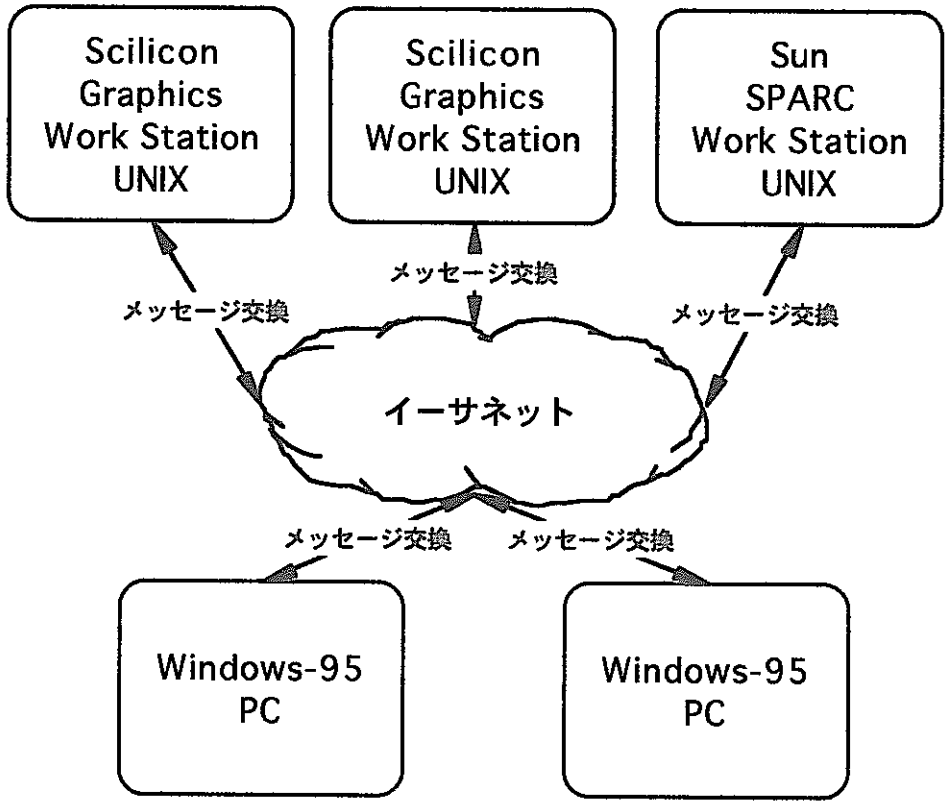

図３．７－１　試作システムのメッセージ交換による協調処理概念

尚、視覚化に利用したシミュレーションは、計算量等の条件から試験的に３次元の疑似的なシミュレーション・データを造りだすプログラムを開発して用いた。それにより、従来より開発されてきたFOTRAN言語によるシミュレーション・コードとのインターフェイスも合わせて検討した。

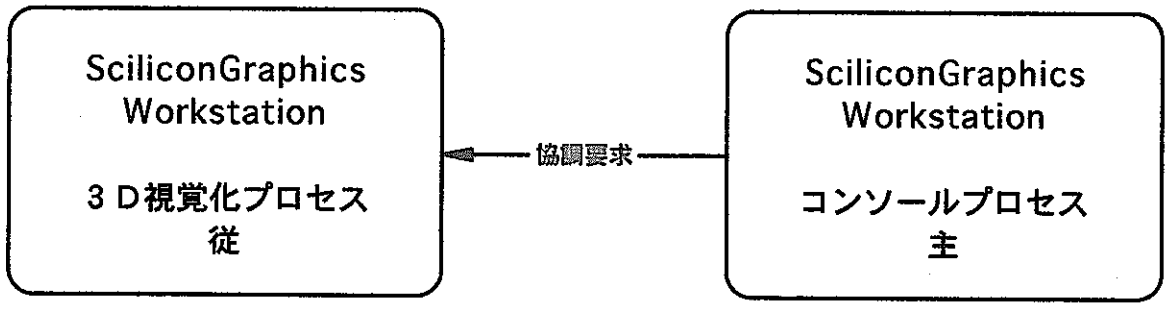

図３．７－２　３Ｄ視覚化プロセスの協調要求概念

図３．７－２に視覚化プロセスとの協調概念を示す。図から解かるように、視覚化プロセスは専用のワークステーションにて稼働する。この視覚化プロセスは別のコンソールプロセス（システム全体を操作する命令を管理するプログラム）から起動される。協調の開始は、シミュレーションに係わる各種のパラメータ等を、コンソール・システムと対話しながら決定し、シミュレーションの準備が整った時点で、コンソールプロセスがネットワークを通じて視覚化プロセスを起動し、並行動作を行う

よう初期設定する。

同様に、コンソールプロセスは、シミュレーション・パラメータ表示用ＰＣにも同様に初期設定メッセージを送信する。

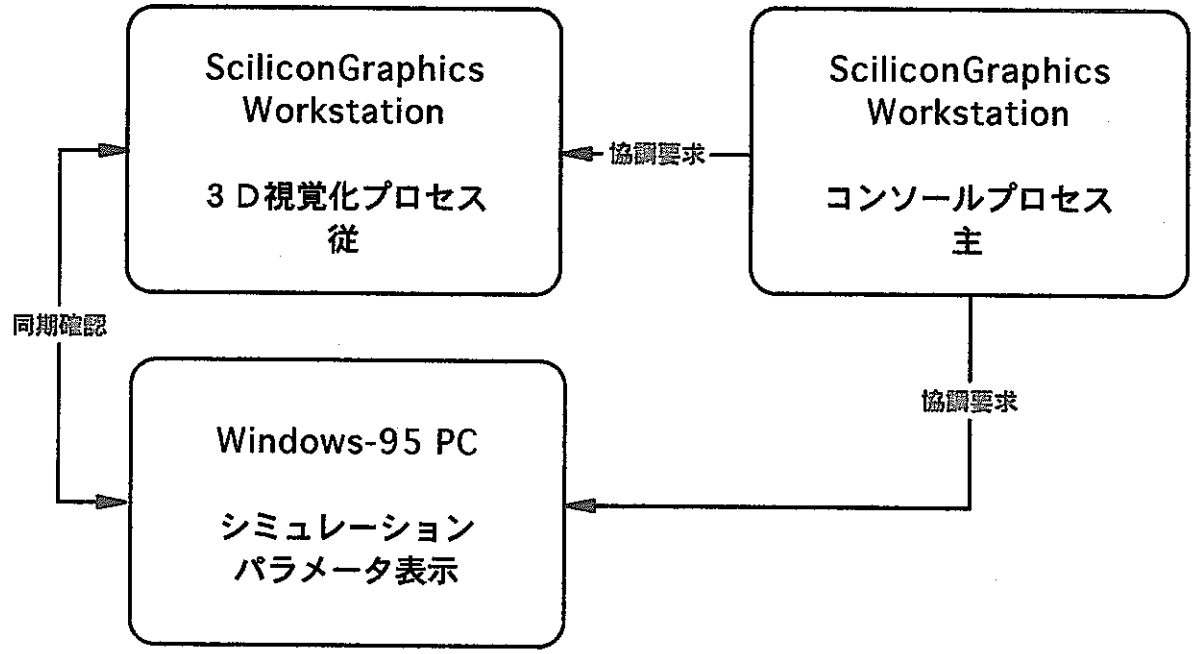

図３．７－３　ＰＣプロセス協調要求概念

図３．７－３にパラメータ表示プロセスへの協調要求概念を示す。パラメータ表示プロセスは、シミュレーションに係わる共通パラメータ（例えば、時刻、サイクル等）を表示するプロセスである。将来は、このプロセスが複数のモデルを協調させて計算を実行する主計時（マスターロック）として動作するよう研究する予定である。図３．７－３に示すように、コンソール・プロセスから要求された協調指令に沿って起動されたパラメータ表示プロセスは、３Ｄ視覚化プロセスと同期確認を行う。それにより、視覚化プロセスは、パラメータ表示プロセスの発信する時刻信号にそって動画再生を行う。今回の試作で利用したコンピュータは、計算能力が実時間動画化計算を実行するには充分でないため、計算結果を読みだして動画化することにした。その概念を図３．７－４に示す。

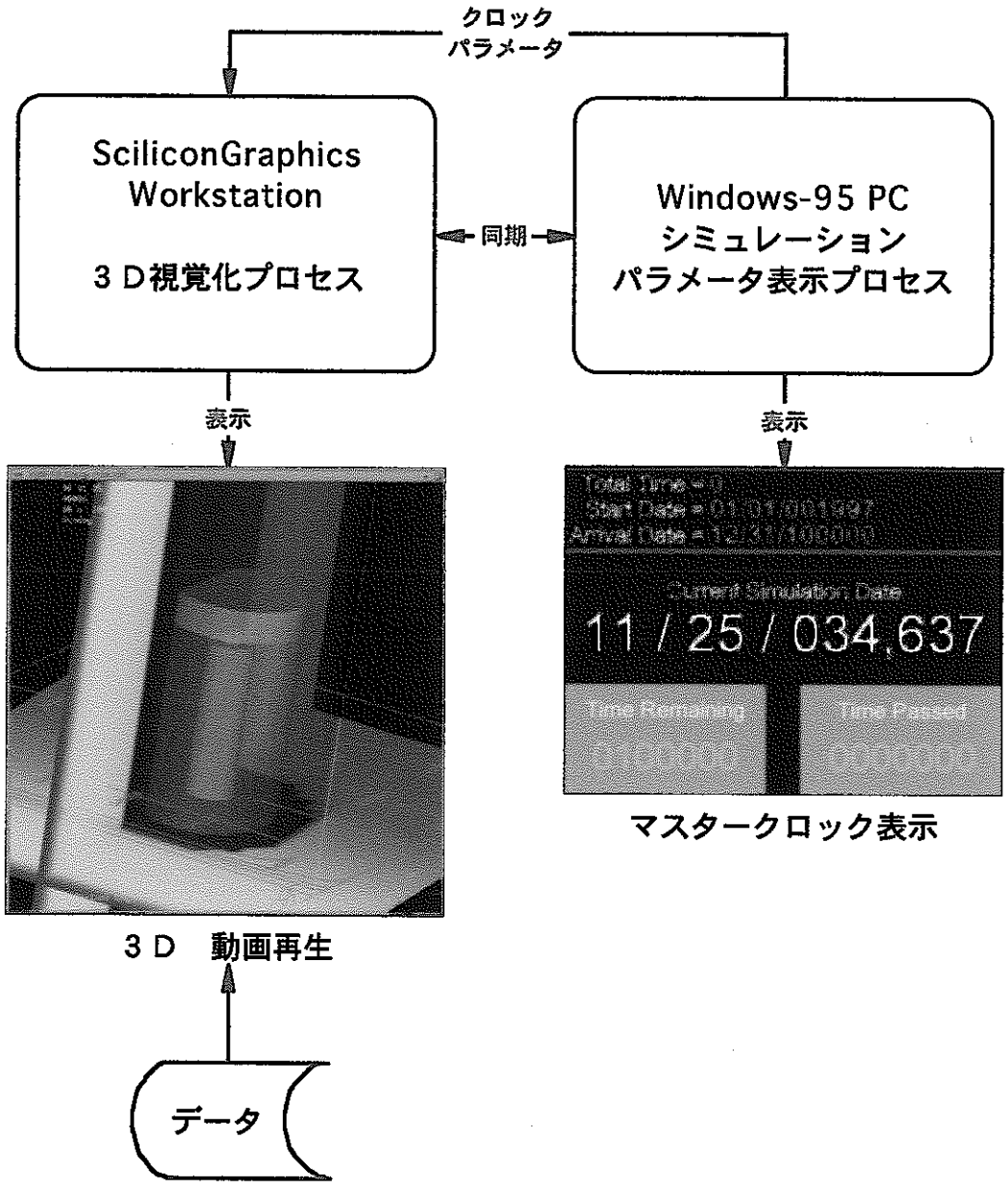

図３．７－４　クロック・パラメータによる協調処理

図３．７－４に示すように、試作したプログラムは、ＰＣにより実行される主計時プロセスと３次元動画再現プログラムの協調により動作するものである。この機能の動作確認により、複数の異なったコンピュータに並列的に配置されたプログラムを、一つの主計時プロセスによって協調動作させる技術を確認した。これにより、地層処分ヴァーチャル・エンジニアリング・システムの基本的なシミュレーション・アーキテクチュアを、マスタークロック・プロセスにより、並列的に協調して動作する構成として設計することが可能であることが解かった。それは、図３．７－５に示すような並列モデル協調計算アーキテクチュアのプログラム可能性を示すものである。但し、そのようなアーキテクチュアの開発は、従来のFORTRANプログラムに比べ飛躍的に複雑なプログラミングを要求すると考

えられる。

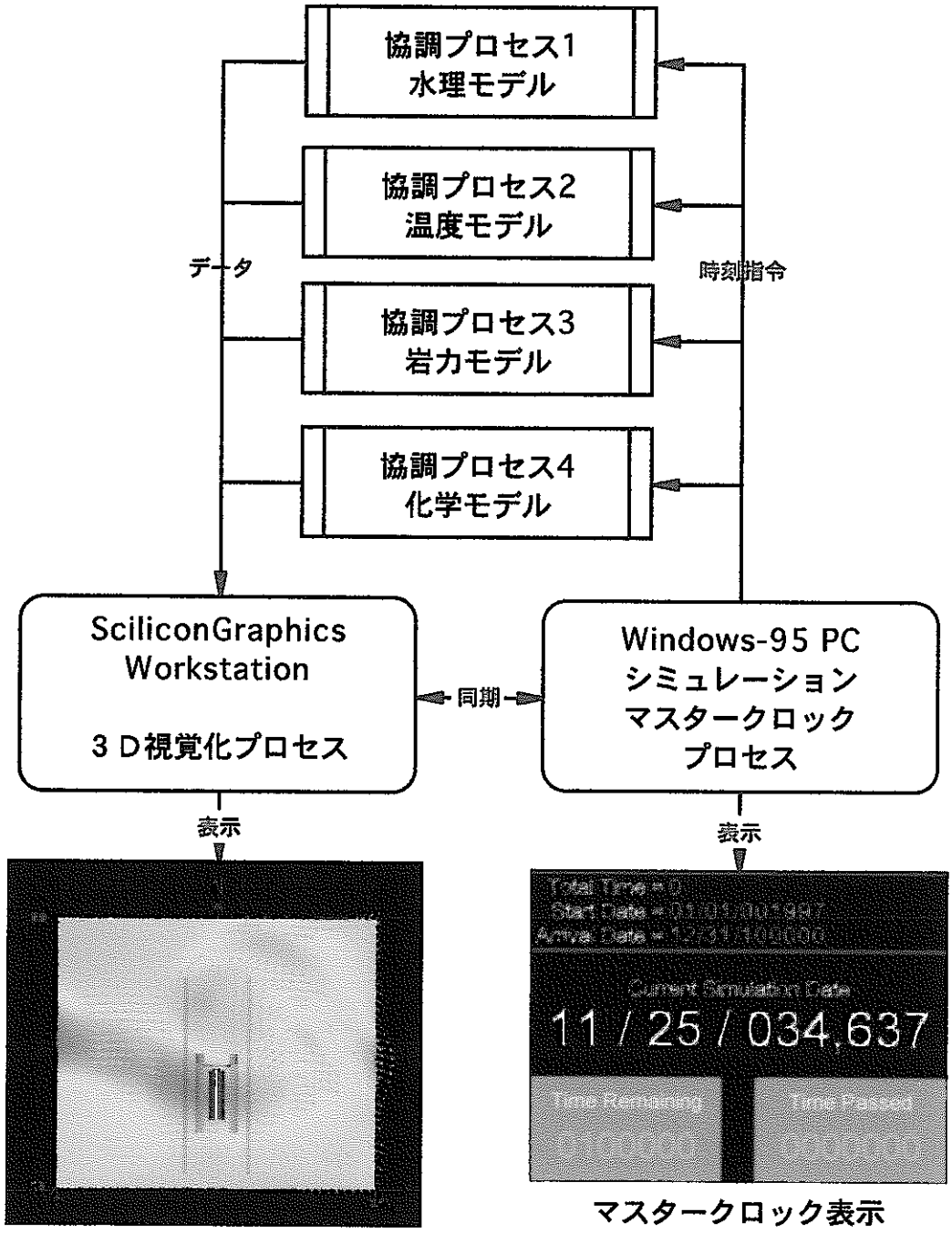

図３．７－５　並列モデル協調計算処理の概念

以上の検討から、図３．７－５に示すような並列したモデルの協調計算処理を考えることが出来る。このようなアーキテクチュアによる連成問題へのアプローチも、新しいシミュレーション手法として考えることができる。それはおそらく、複雑に関連するモデルを組み合わせて、視覚化シミュレーシ

ョンを行う機能を実現する一つの可能性として期待することができる。そのためには、以下の課題についてより研究する必要がある。

プログラミング技術の課題

協調計算のプログラミングは並列処理を基礎とするであろう。その時、並列処理は単なる計算速度の向上を狙ったものではなく、モデルとアルゴリズムの並列化により、直列計算（従来型）では扱えない（表現できない）問題を取り扱うことを主眼とするべきである。例えば、手順並列法、結果並列法、専門家並列法といった並列問題のモデル化について、その特徴を良く把握し、協調計算プログラムのアルゴリズム開発に応用する必要がある。また、並列協調計算を実現する情報処理モデルの基本的な考え方として、オブジェクト指向のモデル化を導入することも大切である。更に、将来の人的リスク等の問題を考えた場合、独立した判断によって動作するオブジェクトとしてのエージェントと言う概念を把握し、モデルに組み込む手法を検討することも大切であると考えられる。そのようにして、目標とするシミュレーション機能は、マルチ・エージェント／マルチ・オブジェクトな並列協調計算をマルチコンピュータで実現するものとしてアーキテクチュアを設計するべきである。

数理モデルの課題

並列協調計算を利用して、数理モデルを組み合わせた計算を実行することが考えられる。その時、個々のモデル空間座標は３次元であることが望ましいが、モデルとしては最も還元されたものを扱うことが良いと考えられれる。何故ならば、自由度の高い連成モデルは分割できないため、協調計算には向かないと考えられるためである（それらは、むしろ単体で利用すべきであろう）。但し、協調計算においては、個々のコードが、計算サイクルや時刻は他のコンピュータ（試作システムではＰＣ）のマスタークロックから受けること、連成させると物理量が繰り返し計算となるもの（例えば、圧力Ｐは、熱を連成させると密度の変化と勾配の釣り合いを解く繰り返し問題になるであろう）には、特別なメッセージ交換技術の開発が必要と考えられる。

新しいパラダイムの受容

いずれにしても、本研究で取り上げているシミュレーション手法（マシンパラレル・モデル協調計算）は、全く新しいシミュレーションの考え方である。この考え方は、シミュレーションと捉えるより、仮想世界における数値実験と考えるべきである。そのような、新しい科学に対する考え方が受け入れられることが重要な課題である。

4　シミュレーション問題への新たな観点

高レベル放射性廃棄物地層処分プロジェクト全体からみた性能評価の位置づけは、設計建設された処分システムが計画通りに性能を発揮するかを予測する手法を開発し、それをもって確認することである。従って、性能評価と言った場合、あたかも工場から出てきた組み立てられたばかりの試作車を試験場に持ち込んで、総合的な安全性、経済性、運動性能等を評価する場合と同様な立場から多重バリア方式の処分システム性能を評価しなければならない。そのような観点からの問題の捉え方は、個別の設計論拠の確認に係わる現象解明やモデル開発という観点とは異なった、いわゆるジェネリック・アセスメントの手法開発を要求する。

例えば、前述の試作車の場合、エンジンのシリンダから予測できそうな限界は、そのエンジンの凡その出力であり、車両全体の構造に係わる情報がなければ、運動性能を予測することは不可能である。同様に、地層処分システムの性能も、個別の現象（例えば、地下水の地球化学的特性）単体から全体を予測評価する枠組みを組み立てることはできないと考えられる。そのように、地層処分の技術開発に係わる重要な観点は、処分システムの設計建設に係わる個別要素評価手法は、参考になる技術を提供すると考えられるが、そのままでは、性能評価の一現象解明手段としてしか利用できないため、処分システムの設計に係ると技術と、性能評価に係る技術を明確に分離して開発していくべきである。

そのような観点から、性能評価においては、その枠組み（アセスメント・フレーム）を決めるための手法としてシナリオ開発があげられる。シナリオの開発は、計画する処分システムが計画通りに性能を発揮するか確認するために必要となる、支配的な事象とその関連を論理付けることである。そのための科学技術的方法がシナリオ開発法であり、確率論的手法（PSA : Probabilistic Safety Assessment）、文章（FEP : Feature, Event, Process）、現象の関連（PS : Process System）等、幾つかの手法が提案されている。

また、シナリオの開発においても、前述の設計と評価における問題の混合の様に、施工シナリオと評価シナリオは分離して考えなければならない。もちろん両者の関連はあるが、性能評価の枠組みを決定するにあたり、最も単純な枠組みから出発し、評価シナリオと施工シナリオ、操業シナリオ等を混乱させないことが大切である。例えば、建設施工時のシナリオとして、発破による掘削影響、ボア・ホール等により水理が変化することが知られているが、これは建設施工シナリオとして安全性設計課題として扱うべきである。また、操業時に生じるリスク（落盤による坑道の封鎖等）も分離して考えるべきである。更に、性能評価の枠組みに加える必要はあると考えられる、断層、地震等、確率的かつ動的なシナリオ（変動シナリオ）も分離し、当初は静的な地下水による核種漏洩と移行シナリオを取り上げることが基本的アプローチである。

（1）性能評価の難しさ

以上のように、性能評価シナリオを単純化しても、対象とする処分システムの性能を評価する枠組みは非常に複雑なものとなるであろう。最低限のパラメータを見積っても、それは地下水の水理、地球化学的性質、および廃棄体の発する熱の影響から生じる地下水二相流れや、岩盤応力変化、破砕体への影響による浸透流の変化に伴う化学的遅延効果の変化等、非常に複雑で再帰的な（フィードバックのある）現象を扱わなければならない。

さらに、評価においては、注目する現象が、これらの個別現象の連成現象であることが当初からの

課題である。従って、地層処分性能評価のアプローチの難しさは、始めから連成した問題として対象を考察し、モデル化しなければならないことである。このことは、地層処分性能評価と言う問題が、個別現象解析と異なった、本質的な難しさと、現象理解の方法論の相違を内在していることを示すものである。

以上の観点から、本項では、そのような複雑な系の理解を、従来の要素還元論に基づく数理解析的手法とは別な考え方として、コンピュータ内部で複雑な仮想世界を組立て、数値実験する考え方を予備的に検討し、今後の地層処分ヴァーチャル・エンジニアリング・システム開発における、性能評価シミュレーションの考え方の参考とする。

（2）地層処分ヴァーチャル・エンジニアリングにおける性能評価シミュレーションの考え方

従来のシミュレーション・コードの開発手法は、一般に複雑な物理現象等を簡単な過程に分割し、自由度を下げ支配方程式を簡略化することにより、計算可能な数値計算モデルを組立てプログラミングする方法がとられてきた。それらのコードは、研究開発現場に於ける個々の事象評価用ツール等として開発され、安全性研究等の成果を上げてきた。

従って、従来のシミュレーション・コードの開発は、“一般的な工学的手法として普及している要素還元論的方法に方法の基礎を置いており、分解された個別過程の専門的な現象の領域について、精密に記述されたモデルに理論の基礎を置いている”と考えられる。

また、このような決定論的方法とは別に、確率論的安全解析等の全体シナリオが評価の戦略となるような領域については、統計的手法を応用した研究が進められてきた。それは、核燃料サイクル施設の故障率等から事故発生確率や猶予時間を推定する、いわゆる事象の起因に関する評価手法であり、発生事象の規模評価となる決定論的手法としての、個別現象評価の必要性の論拠を導くための手法として位置付けられていると考えられる。

以上の観点から、従来のシミュレーション・コード（数理モデル）開発手法は、個別の物理化学過程を説明する知識としての成果であると考えられる。しかし、性能評価問題のような複雑で大規模な統合化シミュレーションを具体化するためには、物理現象の解明に係わる情報処理技術とは別な新たなソフトウェア工学分野の研究開発課題が生じると考えられる。

なぜならば、計画する性能評価シミュレーションの手段としての計算科学技術の具体的な姿は、コンピュータ、ネットワーク、グラフィックス、データベース等、様々な情報処理技術の領域で実現されるものであり、物理現象等の解明の成否は、直接ソフトウェア工学の影響を受けると考えられるためである。

また、地層処分ヴァーチャル・エンジニアリングは、実験の場を単純にハードウェアからソフトウェアに移行するという考えではなく、新しいソフトサイエンスの領域の学問を築く活動と捉えるべきである。それは、従来の還元論的な物理科学の方法論と、高価で柔軟性に欠ける実験装置という脇役にとって代わる２１世紀の科学の戦略と考えるべきである。そのような観点から、地層処分ヴァーチャル・エンジニアリング技術の領域は、従来のシミュレーションと呼ばれる、個別の数値計算ソフトウェア開発技術の領域と明確に区別して考えるべきである。当然、従来通りの手法により、個別現象解明のための技術の開発は、今後も推進するべきであり、それらは地層処分ヴァーチャル・エンジニアリングの推進に多くの基礎基盤的貢献が期待できるものである。

一般に、研究活動の目的は知識の生産である。実験は、研究という知的活動を進めるために行なう外界からのデータ収集である。研究者は、自己の知験や直感とデータを照らし合せて新しい知識を生成する。そのようにして、理論が組み立てられていく過程は、要素還元と逆の統合の過程である。この統合の過程は、還元の過程に比較し複雑であり、未知の領域を含む科学の方法として考えられる。そのような観点から、地層処分ヴァーチャル・エンジニアリング技術は、旧来の方法論に立脚した視点からの改善案ではなく、新しい科学技術の戦略として捉えるべきである。

例えば、地層処分ヴァーチャル・エンジニアリングは、実験の困難な対象をソフトウェアに置換するといった単なるシミュレーション要求ではなく、より複雑な現象を総合的に解明するための知識の統合化アプローチを考えるべきである。

（3）新しい考え方（実験装置としてのコンピュータ）

地層処分ヴァーチャル・エンジニアリングにおける性能評価シミュレーションの基本戦略は、物理現象という科学の領域について、それを個別の、より簡単な過程に分解して得られた数理モデルを数値的に解くために開発された、解析コード等と呼ばれるプログラムを統合化し、協調して計算させ、処分システムをとりまく現象を長期にわたり総合的に評価することである。

この際、具体的な研究の範囲は、大規模な統合化シミュレーション・システムを開発する際に必要となる、様々なソフトウェアの統合（Software Integration）に関する開発と管理の手法である。

例えば、良く知られた非線形有限要素法は、シミュレーション手法として成功してきた手法である。それは、対象となる現象過程を工学モデルとして数値的に評価し、設計対象の安全性等を判断する研究などに利用されている。それは、一つの工学的手段として、研究開発現場で受け入れられた数値計算ツールと捉えることができる。

このような場合、解析コードに対する研究開発現場の関心は、研究開発の核心に触れた対象（すなわち現象そのもの）である。例えば、それは評価すべき工学モデルと、それを評価するための解析コード自身の基礎方程式、各種の計算条件を構成する方程式、そして、解の工学レベルでの検証・確証である。また、解析コードの品質（検証・確証）は、研究室の試験設備や、大規模な実験設備等から得られた試験結果等との比較により判断されることが通例であった。対象とする解析コードが、与えられた条件のもとで、試験結果を模擬すると判断される数値を出力した事により確認される。そのような判断は、解析コードのツールとしての妥当性を外部から判断する方法と考えられる。

以上のように、研究の対象が、個別の専門的な現象領域である場合は、そのような判断基準によって、解析コードのツールとしての妥当性を評価することで、充分に工学レベルの実用に供したと考えられる。さらに、それらのコードは、一般的にブラック・ボックスとして利用されてきており、プログラミングに関する細々とした内容は、本質的な議論の対象にはされなかったと考えられる。

例えば、開発したコードの出力する数値が、試験結果をどれほど正しく模擬するかについては、多くの関心が寄せられてきたと思われるが、プログラマが領域変数を用いて反復計算をするアルゴリズムを導く時、移植性を考慮したコーディングを採用しているのか、それともベクトル計算機のような特別のアーキテクチュアの機能を引き出そうとしたコーディングを採用しているのか、または単に動作することのみに関心をもっていたのか、等の計算論の核心をついた部分ついてはあまり多くの関心は払われなかったと思われる。

（4）ソフトウェアのスケールファクターに伴う問題

以上の様に、複雑な現象過程をコンピュータによる仮想世界で再現する試みを期待すると、技術の領域に情報科学が重要な位置を占めてくる。それは、コンピュータがプログラム言語という方法を唯一の動作手段としていること、また、コンピュータは工業生産品であり、メーカー固有のアーキテクチュアがその動作と不可分であること、さらに、どれほどコンピュータが高速になっても、数学的には近似計算しかできないこと、プログラムは大規模複雑化するほどバグ（誤動作）も増えると言った問題を引き出すであろう。

これは、単一目的のソフトウェア開発から、大規模な統合化ソフトウェアの開発に移行する際にソフトウェアのスケールファクターの影響として生じてきた領域の問題であると考えられる。それはまた、過去のソフトウェア開発に内在していた問題が、大規模ソフトウェアの統合化を目指す現在のプロジェクトの課題として表面化してきたものである。これらを整理すると、地層処分ヴァーチャル・エンジニアリング・システムの性能評価シミュレーション・アーキテチュアを開発するために考慮しなければならない、ソフトウェア科学・工学の領域は大別して以下の通りとなる。

分野１：ソフトウェア統合化のための理論分野の研究領域

大規模な統合化シミュレーション・システムを開発するための方法論は、個別の専門領域に対象を分割し、自由度を減少させることにより、方程式等を簡素化してプログラミングを容易にし、それら個別のプログラムを統合して、多方面からシミュレーションし、最終的に支配的なモデルを導きだすと言う、決定論的シナリオによる方法論の妥当性を研究する領域である。それはまた、確立論的方法との比較、連結、そして複雑系モデルとして、トップダウンとボトムアップの中間に複数の計算モデルが、コンピュータの中の仮想世界で協調して計算を進めるという考え方を位置づけ、研究する領域でもある。

ソフトウェアの構造的手法の研究
ソフトウェアの複雑度の影響領域の研究
数値計算の信頼性に関する研究
論理処理、記号処理のシミュレーションへの応用
モデル協調計算手法に関する研究
カオス、複雑系、進化計算学の研究　　など

分野２：ソフトウェア統合化のための工学分野の研究領域

大規模な統合化シミュレーション・システムを開発するための工学的手法として、下記領域が研究対象として上げられる。

ソフトウェアのライフサイクル管理手法
ソフトウェアの品質管理手法
マルチ・コンピュータ・シングル・システム・アーキテクチュアの設計開発法
システムの機能を高度化する知識情報処理や認知科学の応用研究

4．1　複雑な現象と複雑なモデルとして捉える

地層処分システムの性能評価のような問題は、社会モデル、経済モデル、一般環境モデル等、各種存在している。これらの問題は、一般に大規模システム（Large Scale System）や複雑系と呼ばれている。反対に、単純モデルに還元された物理問題等を単純系と呼ぶ。

例えば、複雑系は日常生活の至る所に顔を出し、都市の交通網とか株式市場における価格変動とも関係がある。しかもそこに見られる複雑なプロセスは、単純系とは根本的に異なるものである。ふつう単純系は、相互作用の弱い少数の要素からできているか、もしくは逆に、容器内の気体や銀河系のように非常に多くの要素からできており、後者の場合、その振る舞いを研究するには統計的平均化という手法が使われる。

これに対して一般の複雑系は、車両のドライバーやディーラーといったエージェント（系を構成する行為者）からなるが、その要素の数は中規模である。ここで注意しなければならないことは、一般のエージェントとして扱われる対象は、自律行為者であることから、本研究が目指している現象モデルとは多少異なることである。従って、特にそのような物理化学現象（物理化学的な振る舞い）を自律的に再現するオブジェクトを**現象エージェント**と呼んで区別する。地層処分性能評価シミュレーションの狙いは、この現象エージェントの協調計算を開発することである。

そこで、複雑系におけるエージェントは、グローバル（全体的）な情報ではなく、ローカル（局地的）な情報に基づいて決断を下し、自らの行動ルールを更新していくことになる。こうして、複雑系、すなわちローカルな情報に基づいて行動する適応能力のある中規模のエージェントの集合は、今日まで科学が築き上げてきた単純系とはあまりにも異なっている。つまり複雑系の研究は、自然界や人間社会を科学的に探求する上で新しい考え方を与えると思われる。

科学で最も敬意が払われる原理の一つに、いわゆる“科学的”手法がある。この方法で欠くことができないのは、“再現可能な”実験という考え方である。仮説を確かめるためには、実験を行い、実験を行う研究室がなければ、科学の理論は生まれてこない。実験室には様々なものが考えられる。

例えば、化学実験室には、試験管、蒸留器等が完備してあり、場合によっては、原子核を調べるために粒子加速器が置かれることもある。こうした実験室は、ごく普通に見かけるものだが、いずれも単純系の物質構造を研究するために使われてきた。

一方、複雑系の情報理論的研究のための実験室として、研究者たちはコンピュータの内部に現実世界の複雑系を模擬した仮想世界を創造することを考える（ヴァーチャル・ラボラトリ）。それによって、現実に近い形で再現可能な実験が期待できる。

そのようにして、複雑系を探求する科学者たちは、ガリレオの時代に生きた物理学者と全く同じ立場にいる。そのようにして、どんな状況に対しても、コンピュータ内部に様々な世界を創り出せる能力を人間は持っていると考えることが第一歩である。コンピュータを使えば、現実の複雑系に対して無数のシミュレーションを納得するまで遂行でき、対象となる問題領域を単純なサブシステムに分割して考える必要などない。

（1）カオス、影響、感度、モデル化戦略

地層処分性能評価では、特に事象と事象の影響、感度が問題となる。このような問題は、対象を連成した現象エージェントとしてとらえ、ダイナミックに評価する課題を生じる。特に、ソフトウェア

工学（気象モデルの離散方程式のプログラミング）から発見されたカオスは、その代表的な例である。同様に、非線形系は必ずしも構造安定ではない。つまり、モデルを少し変えたことにより、結果が大きく変わりうることが数値計算の分野で知られている。

このことは、処分システムの性能評価でカオスを考えるべきかは別として、カオス系でモデルのパラメータの値をわずかに変えた場合や、モデルの表現のしかた（変数の選び方）をわずかに変更した場合に、その変化に対する敏感な依存性として成り立つということである。

ここで大切なことは、そのような疑問（処分システムを取り巻く環境がカオスのように変化する可能性があるのか？という疑問）が出されても、複雑系のモデルを基盤とした、マルチ現象エージェントによる協調計算法は対応できるということである。

さらに、モデルの示す振舞いが、観測に対して敏感に変化し不安定であることもある（このような性質を記述不安定性と呼ぶ）。記述不安定性は、自然現象のモデル化の在り方に変革を迫るであろう。それは、**「最も信頼しうるミクロの方程式から出発し、それを適度に粗視化したマクロな方程式を導き、最良と思われる近似を使ってモデルを導出する」**という従来の立場は、もしモデルが定量的に記述不安走であれば、その立脚点を失ってしまうであろう。

つまり、“真のモデル”に近いほど、定量的によいという信念の根本にあるモデル間の滑らかな“距離”というものが存在しないかもしれないからである。その場合でも、モデルは定性的には正しい結果を与えることがあるであろう。しかし定性的レベルの議論においては、もっと有力なモデル化が有りうる。それが構成論的に仮想世界をつくりあげる手法である。

例えば**結合写像格子法**（CML: Coupled Map Lattice）では、要素カオス、拡散的相互作用、移流などのいくつかの基本的な手続きによって構成されたモデルを扱う。それらは、従来の物理学と違って、第一原理から導かれるものではない。しかし、一方で、このモデルは複雑なダイナミクスの新しい現象論的なクラスを記述する力を持っていると言える。それはまた、様々な系の複雑現象を定性的にシミュレートしてきた成功例である。

結合写像格子法は特定の系のモデルではない。複雑現象に共通する、定性的な普遍性を理解するために構成されたものである。それだけに、その記述力は特定領域をはるかに越えて、複雑系のかなり一般的な構成モデルに成りつつあるのである。さらに複雑系においては、記述不安定性のため、モデルと現象との間に、はっきりとした定量的な“距離”があるわけではない。したがって、現象の詳細な知識に基づいた記述的なモデルは、現象を必ずしも定量的によく近似するとは限らない。仮に、偶然そういったよい近似としての記述的なモデルが得られたとしても、逆にモデルと現象の間の厳密な対応を知るすべはないのである。

結合写像格子法のような構成的モデルに対して、実験パラメータを、これこれに設定したら、このような測定値が、これこれになるといった意味の定量的な予測ができないという批判が浴びせられてきた。しかし、逆に構成的な研究方法を通して、むしろ現象学的な方法がいかに正当なものであるか、また現象学によって記述された普遍的なクラスがいかに大きなものか、さらには複雑現象の本質とは何であるかを知ることができるようになるであろう。

このような方法を通してのみ、あるクラスの複雑現象が、自然の様々なレベルで共通に存在するのかを知ることができ、また、どんなクラスの系がどのような現象を導くのかを予測することができると考えられる。

こういった仮想世界の構築によるアプローチは、特に地層処分性能評価のような、歴史性という要素の入った現象の理解には欠かせないのではないかと考えられる。歴史的に、一つのパスを通って起こってきたような現象は、再現性を重んじる従来の科学の方法論では扱えないのではないのかと言う議論がなされてきた。例えば進化である。

これに対し、人工的な世界をつくることは、有り得たかもしれないパスを眺観していくことを可能にするので、進化や脳のような発展、過去の気象等、変化する系の問題への新しい切口を開きうるのではないだろうかと考えられる。以上のように、複雑系の理解においては、次のような特徴があるように思われる。

現象により密着してたてられたモデルが、必ずしも普遍的記述力をもつとは限らない。
逆に現実からやや距離をおいたモデルが、大きな記述力をもちうるであろう。
そこで、仮想世界を計算機の中に構成し、その仮想世界を研究することで、逆により
よく現実の系を理解できると考えられる。この特性は、仮想世界と現実との関係を、
いかにとらえるかが重要な課題となってくる。

計算機科学は、ヴァーチャル・リアリティを「現実」のものとした。一方、カオスを計算機で計算することは原理的に不可能である。カオスを特徴づけているのは、非加算無限個の非周期解と加算無限個の周期解である。一方、計算機は有限オートマトンであるので、どんな無限性も計算できない。計算結果は結局は周期解になってしまう（ポアンカレの回帰定理：周期は非常に長いこともある）。さらに多くのカオス系では、個々の軌道の近似計算ができないことが知られている。

微分方程式や差分方程式のような偶然性をいっさい含まない決定論的な方程式は、初期条件を与えれば、以後の解軌道を完全に決定するはずである。ニュートン力学でよく知られているように、運動方程式と初期条件は論理的に独立である。しかし、方程式がカオスを生成する場合、軌道の初期条件鋭敏性により、軌道に関する将来の予測は不可能になる。この状態において、運動方程式と初期条件は観測を通じて相互依存することになる。つまり、カオスは両者の間には、もはや観測の仕方によって、きわめて敏感にその状態を変えるので絶対的な因果関係は認められない。

以上の議論は、複雑な物理現象をコンピュータ・シミュレーションに置き換える時の本質的な問題を提示する。それは、現象と数理モデル、そしてプログラムという、異なった世界に等価な論理を持ち込もうと努力する活動に対する基本的な前提条件である。

（2）ソフトウェア工学面からみた問題

前述してきたように、一般に研究活動におけるシミュレーション・プログラム開発の狙いは、仮説として組立たモデルについて、実験等の外界から得られた情報に基づき、評価や検証を行なうための手段として、コンピュータを利用しモデルの数値による近似解を得ることである。これは、経験から得られた知識を文字によって記録すること、更にそれを発展させて数字と数学記号により、より精密な量と変化の割り合い等を記述する活動の次に現われた知識の記述戦略と考えられる。

シミュレーション・プログラムの開発においては、複雑な物理化学現象を簡単な過程に分解し、それら分解した過程について、個別数学モデル（方程式）を準備し、それを数値的に近似計算するプログラムを開発するという手順がとられてきた。コンピュータの基本動作は2進法による論理演算であ

るため、実際にはモデル式を数値的に近似する離散式を導き、その離散式からコンピュータが実際に動作可能な２進の命令と呼ばれる一連の規則をディジタル信号に変換するための操作が必要になる。

そのための手段としてプログラミング言語があり、計算機械としてのコンピュータがある。従って、開発されてきた個別の解析コードは、開発段階では統合化を想定して計画されたものではないと言える。そのことは、開発された様々な計算コードを組み合わせることにより、現象全体を模擬することが技術的に可能か？そのためには、従来の計算コードは再利用可能か？どのように組み合わせれば良いのか？また、たとえ組み合わせることができても、それが妥当な選択であるのか？コードの組み合わせから生じるソフトウェア全体の複雑度の増加はどのような問題を生じさせるか？等、避けて通れない情報科学の分野の課題が潜在している考えられる。

また、開発した解析コードの出力する数値が、工学試験結果をどれほど模擬するかという疑問についての比較検証や、検証された解析コードの出力する数値と、新たに開発、もしくは改良整備したコードの出力する数値を比較するクロス・チェックにしても、判断の基準は工学試験結果であるが、対象としているコードの本質は、形式的なプログラミング言語により記述された規則であり、その本質は、言語とアルゴリズムという抽象的な哲学的原理という実験とは別な領域に出発点をおいている。

それは、ソフト・サイエンスの領域に本来属するものであると考えられる。その様な観点からは、与えられたコードが、試験結果を上手に模擬する数値を吐き出すと言われても、それがソフトウェアとして高い品質を保持しているものなのか？また、それが計算する範囲のモデルの普遍性はどうなのか？将来的に他のモデルを計算するコードと組み合わせて利用できる可能性があるのか？という質問に対する明確な回答は見当たらない。

従って、従来の解析コード等のソフトウェア開発は、個別の専門領域の要求から発生し、開発されたコードの妥当性は、それ自身の動作結果として生じるアウトプット（一般に数値）を工学試験結果等と比較検証すること、いわば、外部からの評価で、その信頼性、利用可能性等の妥当性が判断されてきたと考え、内部からのソフトウェア自身としての評価については、その手法等、未知の領域が残されている。それはまた、数理モデルとコンピュータ・プログラムは異なるものであることを認識する重要な観点である。特にプログラミングに於ては、条件分岐という概念が大きくアルゴリズムを支配する。以下にその極端な例を考える。

あられ数アルゴリズムは、正の数の集合に対する関数h：N -> Nによって以下の様に定義する。

$$h^k(n) = h\left(h^{k-1}(n)\right) \qquad where \quad k \in N$$

nが奇数の時　h（n）＝n／2
nが偶数の時　h（n）＝3 n＋1

式４．１－１

nを任意の初期値としての正整数とすれば、あられ数列は以下のフィードバック・ループで計算す

ることができる。図４．１－１に簡単なFORTRANプログラムを示す。このプログラムは、主プログラムと関数サブプログラムの２つからなっており、関数サブプログラムはn=55とした時のk=100に対するあられ数列を計算する。これは、典型的な手続き型プログラムの例である。あられ数は、初期条件に不規則に反応するような数列として整数論の領域で知られている。

```
      Program Hail_Number
C
      N     = 55
      K     = 100
      Do I = 1,K
      N     = NH(N)
      Write(10,1000) I,N
 1000 Format(2I10)
      Enddo
      Stop
      End
C
      Function NH(N)
      If(Mod(N,2).EQ.0)  NH = N/2
      If(Mod(N,2).NE.0)  NH = 3*N + 1
      Return
      End
```

図４．１－１　手続き型プログラム例

それは、最後に１に収束すると予想されているが証明はされていない。あられ数は、問題自身は極めて簡単だが、解決は極めて難しい問題である。図４．１－１に示す様なプログラムは、FORTRAN等の分野では、領域変数のアドレス計算などでしばしば見かけるものである。また、関数から解かるように、明らかに２つの計算式が交互に利用されて１つの整数列をつくりだしている。これは、プログラム面からみれば f(n)=n/2、と f'(n)=(3n+1) という個別の計算式を機械的に連係していることになる。そのような意味では、より複雑な個別プログラムを連係という複雑な計算を検討する単純な例として考えることができる。

図４．１－１から、従来型の静的な単一リンク方式によるプログラム実行形式は、ソースプログラムが実行形式ファイルとして生成され、メインプログラムと関数副プログラムからなり、実行は、シェル等のシステム・サービス機能により、実行形式ファイルがメモリーにロードされ、メインプログラムの開始番地（Entry Address）に制御が移されることにより開始される。

プログラミング作業は、対象となる問題領域を定められたプログラミング言語という約束に従って、２進の論理信号に変換する操作であり、対象とした世界との関係は論理記述のみである。この考え方は、コンピュータの誕生時から現代の超並列コンピュータまで継承されている。

プログラムは実行を開始すると、制御を関数NHのEntryに移し関数Nが実行される。この時、関数サブプログラムNHは、主プログラムからのみ参照可能であり、実行が終了すると、制御は主プログラムに帰されることである。このことは、実行形式ファイルは、主プログラムと関数サブプログラムNHからなっており、それらは一体化して生成さていることを示している。従って、関数サブプログラム

NHは、図４．１－１に示す例題プログラムの場合、主プログラムの式N＝NH(N)からのみ制御を受る。このような一つのプログラム内で全てアドレス解決が行なわれる場合、それを単一な実行形式（静的なリンケージ）と考える。

このように、手続き型プログラムのサブプログラムは、プログラミングのための思考の構造化の単位として利用されていると考えられる。構造化は、分割されたサブプログラムを呼び出す命令がプログラム内部にあることを前提に計画されたものである。従って、関数サブプログラムNHは、呼び出し側である主プログラムの利用目的により準備されたものと考えられるので、別のプログラム単位から呼び出されることは予定していない。プログラマは、関数サブプログラムNHに与える引き数の範囲と、それが返す値の範囲をある程度把握してプログラムを構造化している。従って、手続き型プログラムは、統合化を考えた場合、下記の構造上のリスクを生じる。

手続き型プログラミングにおいて、１プログラム単位で構造化されたサブプログラムが返す結果の範囲は、同プログラム内の呼び出し側にて定まる条件で定まる。従って、サブプログラムは、一般にそれ自身が返却する結果に責任を取らない。そのため、別プログラム単位から呼び出す処理を考えた場合、呼び出し側と呼び出されるサブプログラム両者が結果に責任をとることが出来なくなるため、新たな規則を開発適用する必要がある。

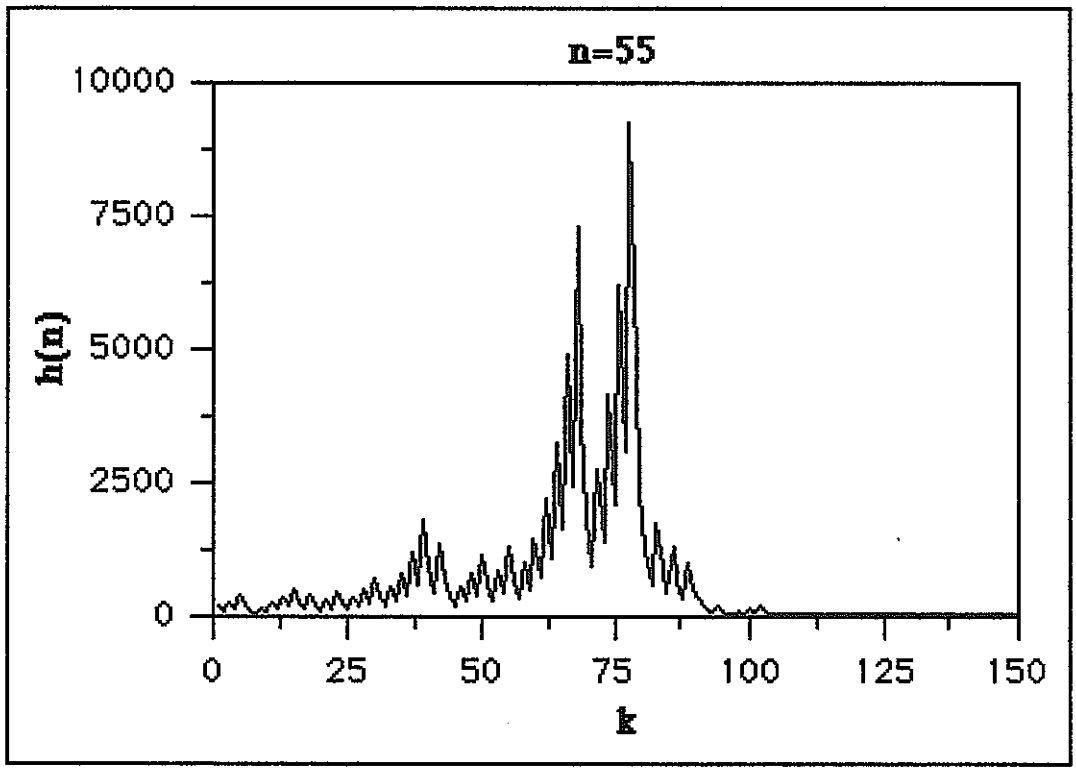

図４．１－２　ｎ＝５５、ｋ＝１５０についてのあられ数

図４．１－２は、図４．１－１に示すプログラムのアウトプットを作図したものである。ｎ＝５５のあられ数についてｋ＝１５０まで計算している。このあられ数は、初期値を少し変えただけで大きく変化するが、最終的には４、２、１と収束していくと仮定されている。図４．１－３に、初期値を５６とした場合の同プログラムのアウトプットを作図したものを示す。図４．１－２と図４．１－３

は少し（１だけ）の初期値の変化が全く異なった結果を生じる例の一つである。

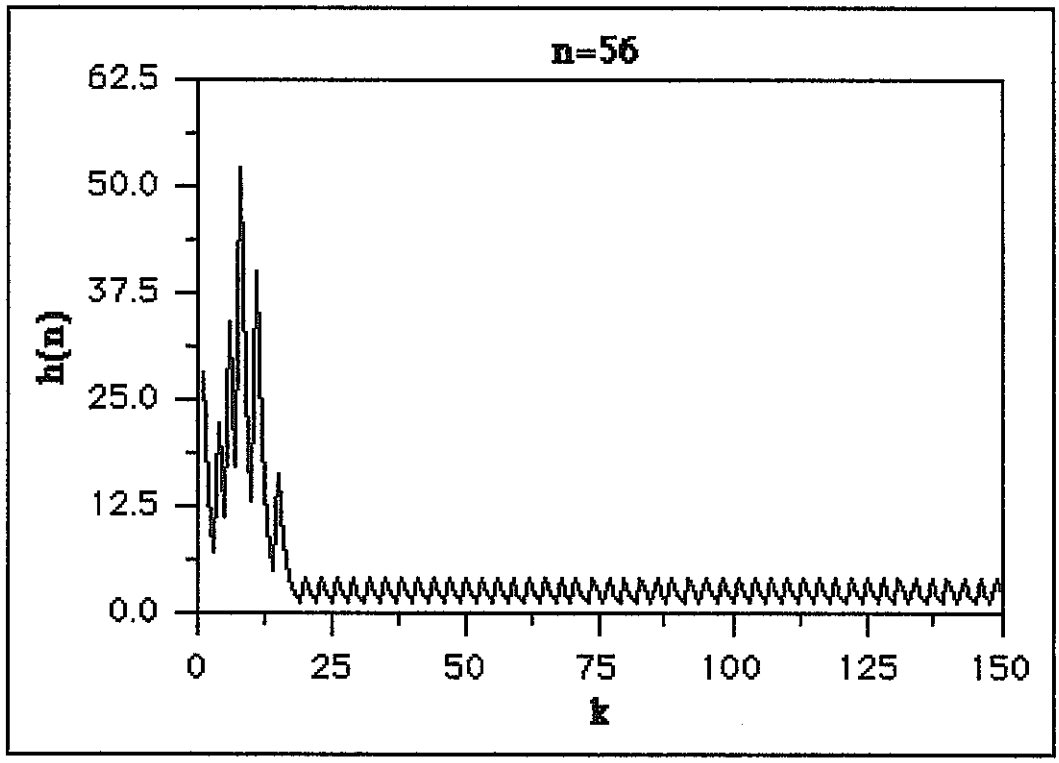

図４．１－３　ｎ＝５６、ｋ＝１５０についてのあられ数

図４．１－３と図４．１－４から繰り返し演算中のIF文等により分岐した条件演算は、計算を複雑にする。それは、見通しのいい計算を行なうことを阻害する要因でもある。あられ数のプログラムは、その簡単で極端な例を示しているが、1000行を越えるような計算ループの中に、数十の分岐を含むような場合、そのアルゴリズムはあられ数のような複雑な解の動きをする可能性が考えられる。

繰り返し文、条件分岐文の多用は計算アルゴリズムの複雑度を上げ、場合によっては計算結果を左右する原因となる可能性があると考えられる。それは、プログラムの規模と複雑度に比例して増大するリスクと考え管理する手法を開発し適用する必要がある。

ソフトウェアの統合化においては、FORTRAN言語等により開発されてきた従来の解析コードについて、ループや分岐文による複雑化に起因する計算の信頼性、安定性について工学試験データとの検証とは別に検証する手法を開発する必要がある。それにより、対象とする計算アルゴリズム部分の動作を具体的に把握し、同アルゴリズムの利用可能範囲（例えば、初期値の範囲）を定量的に把握しておくことが、それらを統合化の基盤とするために必要となる課題である。

次に、プログラムの実行について考える。説明を解かり易くするために、メモリーに展開された実行形式を、“プロセス”と呼ぶことにする。プロセスは、コンピュータのプロセッサが直接参照するメモリーにコピーされた実行形式と関連情報の状態と考える。従来のメインフレームでは、プログラムの実行形式（ロードモジュール等と呼称した）を必要なサブプログラムを主プログラムと一まとめにして番地付けを行ない生成した。従って、プログラムはサブプログラム全体の大きさに比例して肥

大していくが、シミュレーションコード等の繰り返し計算を多用するプログラムでは、プログラムの実行が局所化する傾向にあった。

例えば、差分法や有限要素法等の離散近似法を利用したシミュレーション・コードでは、未知数の行列計算や、n+1 時間ステップの場の状態量の釣り合いを計算するための反復計算等を実行するサブプログラムに計算処理が集中することが知られている。そのような経験から、ソフトウェアの統合化を計画した場合、静的な積み重ねによる実行形式プログラムの生成は、費用、性能面から現実的な計画ではない。むしろ、個別の実行の対象となるアルゴリズムを単体で準備し、並列的に実行しながら相互に情報の通信を行なう手法（ダイナミック・リンク）が現実的であると考えられる。

ダイナミック・リンクは、サブプログラム等の一連のプログラム実行に必要な個別のプログラム単位を、必要な時にメモリーにロードしてプログラムを実行する方法である。ダイナミック・リンクは、複雑化するソフトウェアの管理面での効率化、プログラム再利用、開発面での効率化等、大規模シミュレーション・ソフトウェア開発面で多くの利点が期待できるが、動的なプログラム制御技術面からの複雑度に起因するシステム全体の動作安定性確保のための手法、プロセス間メッセージ交換手法の開発、例外処理等といった新たな技術課題を生じる。

単一実行形式によるプログラムの実行を前提に開発されたコードを、ダイナミック・リンクによって、動的な並列処理により統合化する戦略には、要素となるプログラム自身がダイナミック・リンクを設計思想に有していない。そのため、動的な計算アルゴリズムの複雑化による計算の無秩序な進行、プロセス間の情報交換手段の欠如、プロセス間の同期性の欠如等のシステム開発リスクの発生が予測されるため、それを管理する手法の開発と適用が必要である。

予想される困難な問題として、ダイナミックな並列プロセス間の協調による数値計算の無秩序な進行を、どのように防止、監視、制御するかと言う課題が上げられる。コンピュータ内部で展開される動的な複雑計算過程に置ける計算秩序の問題は、それだけで充分なこれからの計算科学技術の基盤研究課題となると考えられる。

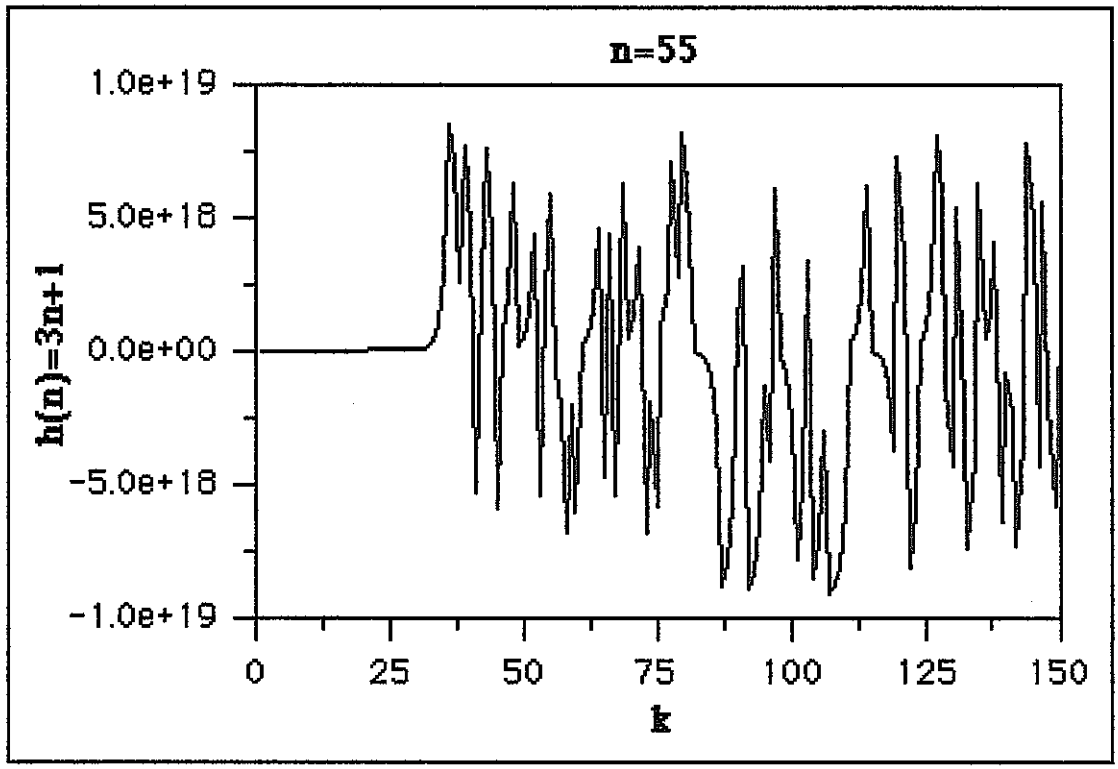

図４．１－４　無秩序な計算過程の発生

図は、h(n)＝3n+1としてn＝55ÅÃ＝150について計算した結果である。この計算は、kの値が36以降は整数がオーバーフロしてしまい、そのまま符号ビットにシフトする。そのため、数列はノイズのような動きをする。これは、予測できない計算過程の例として考えることができる。

35	2826782298093483445
36	8480346894280450336
37	6994296609131799393
38	2536145753685846564
39	7608437261057539693
40	4378567709463067464
41	-5311040945320349223
42	2513621237748503948
43	7540863713245511845
44	4175847066026983920
45	-5919202875628599855

図４．１－５　{h(n)＝3n+1,n=55} に於けるk=35から45までの値

（３）複雑な数値計算における創発という概念

私達が複雑な系に求めていることの一つに、系に予め人為的に埋め込まれていない性質が現れてくることがあげられる。このような自発的な生成に対して、最近、創発という言葉が使われ始めた。

創発に関する明確な定義はまだない。しかし、次の様に考えることはできるだろう。現れた現象を

説明するためには、そのプロセスを追っていくしかない。つまり、問題設定だけから、より短い説明を行うような情報の圧縮ができないということは、その必要条件である。創発という概念を最初に現在的な意味で用いたのはポランニーである。現在、この概念が広く使われる理由は、トップダウン対ボトムアップという、人工知能や神経回路網などでの論争を超えたいということと思われる

そのような複雑で大規模なシステム（地層処分システムの環境中挙動もそのような対象である）の評価を行う場合、計画したシナリオから、自発的な事象の生成等が考えられる。そのような場合、創発という概念は、不確定性や感度とともに重要な概念となるであろう。

これらの分野の議論では、「上」（トップ）のレベルは少数の自由度で記述ざれ、「下」（ボトム）のレベルは、非常に多い数の自由度が関係しているという前提が置かれているように思われる。ボトムアップでは、「上」の性質は「下」のレベルから、一種の「秩序パラメーター」として生成され、それが巨視的な振舞いを決めるとしている。逆にトップダウンでは、「上」のレベルからの少数の指令が「下」の振舞いを決めるとしているわけである。

創発に多く期待されているのは、この両者の間の流れを双方向にするというものであると思われる。創発という言葉には、双方向の流れを通して、前もってプログラムに「入れておかなかった」ことが、どちらのレベルに還元されることなく、自発的に生成されるのではないかという期待が込められているように思われる。

それでは、この自発的生成とはいかなる意味であろうか。この文脈で、カオスを含め、あらゆるもののシミュレーションに見られるように、コンピュータにプログラムを組んで走らせた結果に本当に創発が見えるのであろうか。

また、プログラムを組んだ段階では予期できなかったことが、シミュレーションの結果として現れることがしばしばある。そこで、これを創発と呼びたくなるのではあるが、これは我々の認識が、生成に対してプログラムのレベルでは働かなかったことを意味しているだけである。

さらに、有限状態の世界の中で、有限ステップの中で現れることで、プログラムに予め「入れておかなかった」ものが生成されることは論理的に有りえない。すると、人工知能不可能論にも似た創発不可能論が出来上がってしまう。では、創発を見たいとしている現在の人工現実（Artificail Reality）の研究は虚しいのであろうか？上の議論は、状態もステップもすべて有限としたことに基づいている。そこで一つの可能性は、無限のテープや状態を持ったマシンの導入である。無限のテープを利用できる可能性を持っていて、時間も可能性として十分長くとれるのであれば、プログラムの実行は無限に続く可能性があり、それはチューリングの停止問題に見られるように決定不能である。

その結果、プログラムに書き込んだこと以上のことは出ないとは、必ずしもいえないと思われる。そこに創発の一つの可能性がみられるのである。さらに、テープが連続無限の状態を持っていたらどうなるのであろうか。これは、実数計算を真正面からとりあげる研究によって考えられるであろう。

ムーアが提唱しているように、帰納論理を実時間での積分で定義し、このオペレーターがうまく働くことを仮定すれば、チューリングマシンの能力を越えることができる。

もしも、アナログコンピュータに不可避的に入り込む雑音を最小限にできれば、有限な実数の状態に非可算無限個の状態を圧縮して対応でき、積分が有限の時間で行えることから、このようなマシンの能力は、万能チューリングマシンと量的にも質的にも異なったものとなることが期待されている。実数計算論と従来のデジタルな計算理論との関係を明らかにすることにより、後者に基づいた形式的

体系では基盤づけが困難な、創発への道が開かれる可能性があると考えられる。

この他、よく議論される無限極限の一つの例として、個数を無限にした極限がある。統計力学における熱力学極限がその例である。この場合、個数を無限にする極限を先にとってしまうことで、多くの性質を導き出すのである。たとえば、熱平衡系の相転移の問題では、この極限により特異性が現れることが知られている。確かに、それによって、ゆらぎの増大など興味深い性質が現れるが、こういった性質は少数自由度での記述が可能であり、それ故、目指そうとしている「単純な記述には落ちない問題」というクラスには属していない。

ここで注目したいのは、カオスのもつ根源的なコントロール不可能性である。はじめに述べたように、カオスでは、力学系を駆動するためのプログラムが用意されているというところから出発したにもかかわらず、結果は、そのプログラムではコントロールできずに、「確率」を導入せざるを得なくなったわけである。

さらに、カオスは「情報」を生成することができるのである。そこで再度強調すれば、問題にしている系にカオスが含まれていれば、プログラムで書いておいた記述以上に複雑な振舞いが原理的に現れる可能性がある。ただし、これは力学系としての創発への必要条件にすぎないであろう。

創発を捉えるには、さらに内部からの観測主体をふまえた考察が必要になってくる。これは今後の重要な課題である。カオスが内在しているような要素間の連鎖や、カオスを用いた計算理論、特に実数（ないし実数時間）計算理論は創発を考える上でも重要となっていくだろう。一方、我々は、カオスをかなり満足のいく程度にコンピュータで計算することができる。これは、状態数が有限といえども、大きければ無限の“影”を見れるということを示唆している。そこで、何らかの意味で、無限の彼方に創発を起こしうるような系があれば、我々のもつ計算機でも“創発の影”を十分見てとれる可能性があることになると思われる。

以上のように、大規模で複雑なモデルをコンピュータで扱うことを考えた場合、創発という概念が無視できなくなる。それは、１セットの支配方程式から出発した旧来の計算コードと呼ばれる、単純な計算手続きから構成される計算と、前章でみたような複雑な並列計算の間では、計算量も当然であるが、モデル自律度と複雑度はかけ離れている（より実数計算論に近い？）と考えられる。また、カオスは大気のモデルから観測されたように、有限状態の計算論の中でもそれを見る可能性があると考えられる。従って、地層処分ヴァーチャル・エンジニアリングに於ても、開発を進めることにより“創発の影”を将来見る可能性を期待できると考えられる。

（4）複雑モデルの関係性のダイナミクス

ここで述べる多くの要素の間の関係性のダイナミクスの研究は、トップダウンやボトムアップなどとは異なる新しいアプローチによって基礎づけられる。例えば、人工知能研究に典型的に見られるトップダウン・アプローチでは、上のレベルでの指令によって、下のレベルの振舞いは決められている。

ボトムアップ・アプローチでは、そういった上からの指令なしで、下の要素間の相互作用によって、上のレベルが生成されることに重点を置いている。特にこのアプローチでは、トップダウン・アプローチにみられる上のレベルの硬さを解消し、上のレベルを自己組織しようとすることに特徴がある。

しかし、この場合、上のレベルが自己組織化されると、一般には、要素の関係は固定されてしまう。すなわち、上のレベルが一度できあがれば系全体の状態は固定されてしまう。この意味において、ボ

トムアップ・アプローチは、多くの要素からの秩序パラメータの生成という、伝統的な統計力学の枠組みを脱却していないと思われる。

この二つのアプローチのそれぞれの欠点を解消するために、次の段階として、上と下のレベルの間にフィードバックを入れるという考え方がでてくる。たとえば、簡単な“蟻”のモデルを例にとってみよう。これは、多数の“アリ”と、それらが出す化学物質フェロモンの濃度の場からなるモデルである。ここでは、アリの運動は、フェロモンの勾配で大きく影響を受け、その濃度が高い方向に向かう傾向があるとする。また、アリは、餌を持っているとフェロモンを出すとされている。したがって、あるアリが餌を見つけ出すと、そのアリが出したフェロモンにより、他のアリが集まり、そこで多数のアリがフェロモンを出すという正のフィードバックによって、餌に至るフェロモンの道が形成されると考える。

そのようにしてできたフェロモンの道が“上のレベル”であり、いったんそうなると、アリはその道に従って運動するようになる。この段階では、上のレベルによる下のレベル（個々のアリの行動）の支配が行われている。ただし、餌がなくなると、この道はまもなく消えていってしまうので、例えば、近い方から順に餌への道を作っては消していくことが可能になる。

しかし、このようなフィードバックを入れたアプローチも、複雑系のアプローチとしては多くの点で不十分であると思われる。まず、アリの運動は、餌の配置といった外から与えられた環境によって支配されているという点である。例えば、もし餌がそれぞれの位置に配給され続けていれば、いったん一番近いところに道ができればその道は固定されるであろう。

その意味で下のレベルの要素の自発性はなく、設定された環境によって決まっているといえる。このアプローチは、最も近い餌を探せという最適化問題の枠からはずれるものではない。さらに上のレベルは、たかだか1−2本の道であって、常に少数の自由度で記述されている。従って、このアプローチでは、もっと複雑な環境の問題を扱うことは不可能である。さらにいくつかの問題点がある。要素の間の関係は、基本的に固定されているという点、要素は一様な性質を常に保ち続けているという点などである。我々がここで扱おうとするアプローチの特徴は以下のようである。これは、そのまま複雑系の特徴を表している。

- 上のレベルは必ずしも少数自由度で規定されていない。
  上のレベルが存在するか否かといった知識は必ずしも始めから与えられていない。
- 下のレベルの各要素は自発的な運動を起こす。
- 外からの境界条件を仮に与えたとしても、各要素の運動は予測／制御されてはいない。
- 各要素の関係自体が変動しうる。

このように見てくると、上と下のレベルなど存在しないとした方がすっきりするように思われるかもしれない。このことは、どんな小さな差も巨視的なレベルへとつながっていくという、カオスのもつミクロとマクロの分離不可能性の当然な帰結であると思われる。

明白にどこが上のレベルであるという境界が、定まった箇所に入れられる問題が分離不能性の特質である。ここで、あえて上下のレベルという言葉を用いたのは、そのように捉えた方が分かりやすい瞬間や状態が数多くあり、その場合には、そういったレベルの生成、崩壊、競合、共存、階層性を見ていく方が理解しやすいことがあるからである。さらには、あえてレベルを区別する境界を導入して

矛盾を導くことで、レベルの不在性が示せる場合もあるであろう。ここまでの議論は、かなり抽象的なことから、それぞれが何を意味するか明らかではないかも知れないが、以上のような考え方が、モデルのダイナミクスに自然と現れてくると思われる。

特に、地層処分性能評価シナリオの開発においては、この関係性のダイナミクスは重要な論点であると思われる。それは、ダイナミックな関係は、トップダウンやボトムアップといった固定したモデル化のアプローチでは示しきれないということである。

複雑系が何らかの意味での非還元性を意味する以上、それを捉えていく新しい方法論が必要とされるであろう。そのための可能な方法としては以下のような例があるように思われる。

- 相空間を解剖しつつ、その構造自身の変化を静的、動的両面から見ていく方法
- 多彩な現象の中から、そこに内在する抽象的構造と関係性を再構築していき、それを通して一般的な概念に至る方法
- いくつかの基本的な概念要素を組み合わせて人工システムを構成していく方法
- ミクロでもマクロでもない中間のレベルから上下のレベルをにらみつつモデルをつくる方法
- 関係や過程などの動的状態を扱える集合論体系に立脚し、複雑系をとらえる新しい言語体系を構築する方法
- 従来の直感に反する事象を数学的に基礎づけ、計算機の中で起こる数学的経験を豊富にすることで新しい直感を養いながら理論化する方法。

また、全体の示す性質が、構成要素の性質に分割しては理解できないという複雑系の特徴は、還元論型方法論の限界を示している例であると考えられる。

（5）構成的理解

地層処分性能評価問題などでは、従来の記述的な方法だけでは理解されない巨大で複雑対象を相手にしている。そのような、複雑な現象の構成的理解のための問題意識の触発はカオスの研究によってなされた。従って、カオスは今後の研究の重要な課題となるであろう。特に、創発とカオスの関係は、現在はマルチエージェントによるモデル協調計算等の考え方でプログラムして、可能性を確認することができるため、現実的な工学面での応用も遠い将来ではないように思われる。

たとえば、乱流のオンセットに対してカオスは普遍的な見方を与えた。それは、乱流の満たすべき方程式を整然と解いていくより、場合により直接的には流体力学とは結び付かない簡単なカオスのモデルの研究が乱流への洞察を与えることがあったと解釈できる。カオス的な観点をさらに発展させることによって、記述的というより、構成的な理解の重要性が強調されるように思われる。

前節に述べたように、結合写像格子（CML）は、ある意味では時空間での複雑な現象への構成的なモデルの一つの例を与える。例えば、頻繁に議論される大気大循環モデル（GCM：Global Circulation Model）もこの例となっていると期待される。複雑系は、こういった「世界を構成していく」方法により、複雑現象の共通な構造を探っていくことを目指しているが、そのような「普遍性」に基づくアプローチは、従来の統計力学と変わらないのではないかという批判も考えられる。

また、多様な現象に対して用いられてきたものに博物学的記述がある。物理学者が行うこういった「博物学的」方向には、やはりどこかで普遍性を求めている点があると思われる。そのとき普遍的な

構造を見い出す努力と、それがいかに個別的に現れるかを見据えていくことが必要になってくる。ここで、カオス研究にすでに普遍性と個別性の二分法の克服がみえていたことを思い出すことは有意義であろう。カオスの生成機構には、ホモクリニシティーのように極めて普遍的な構造があるのではあるが、カオスの現れ方、アトラクターのトポロジカルな構造などには個別性があり、その両面を見ていくことがカオス研究を豊かにしてきたと考えられる。

（6）複雑系のとらえかた

複雑系を理解するためには、従来のように単眼的な見方ではなく、複限的な見方がしばしば必要になる。たとえば、時空カオスのパターンダイナミクスを理解するためには、実空間と相空間での記述を合わせ持たなければならない。力学系と計算論による見方は、カオスやセルオートマトンの複雑さに関して多くの情報をもたらした。ゲームのダイナミクスにおいては、力学系の理論に戦略という異なったレベルのアルゴリズムを絡めて初めて記述が成立する。脳神系の理解のためには、力学系理論、熱統計力学、確率論、集合論、神経回路網理論、解剖学、生理学、分子生物学、計算機科学、論理学などが必要である。

低次元系における数学的解剖学とは、相空間での幾何学的構造を抽出することであった。高次元系においても、類似の方法が必要になる。静的な複雑系における数学的解剖学は（エネルギーや適応度の）景観（ランドスケープ）の構造を抽出することだった。しかし、動的な複雑系においては、相空間における構造の抽出は更にむずかしい。それは解剖学というものがそもそも静的なものだからである。それについては、最近の多自由度のハミルトン系のダイナミクスの研究において、相空間の構造を解剖しつつ、その運動を理解する試みが発展している。

我々が考えている複雑系のモデルは、人工現実（もしくは仮想現実）を持つシステムである。このようなモデルをより知的にするためには、その中に何らかの内部観測が必要である。そのためには、計算機内での内部観測機能の構築の研究が重要である。そのことから、計算機内にニューロンが生成されるための内部的な機構を構築する努力がなされている。系の内部的な見方の重要性は、形式的体系の外からの記述を、再び形式的体系の中に含むように理論を構築することを、ゲーデルによって最初に指摘された。ここに体系の内と外の問題が形式化され、形式的体系の複雑さが明らかにされた。

しかし、内部的な見方が不可避であるというのは、系の記述不安定性と“観測”とが関連して現れてくるからである。外部からの見方は、客観性を正当化するが、カオスや生物の問題では、外部観測と内部からの記述とは必ずしも一致しない場合がある。これは必ずしも外からの観測だけでは理解しえないところである。そこでは、形式的体系における内部観測者の観点が、物理学、化学、生物学、にまで一般化されている。複雑系においては、すでに述べたように、外からの観測だけでは系のダイナミックな様相を十分に知ることはできない。複雑系においては、観測問題が本質的であり、系を記述する記述者（観測者）を込みにして系を記述する必要に迫られる。

（7）仮想世界の理論構築

以上の検討から、地層処分性能評価のような、複雑でダイナミックな系を構成するモデルについては、次のように基本的な考え方を整理することが出来る。

a. 自然は複雑であり、人類はそのごく表層部分を明らかにしたにすぎない。
b. 単純な還元論的思考の限界はカオス理論の中で明確にされた。
c. 複雑現象に対して、単一の力学的予測は破綻する。
d. 観測者の視点を明示的に自然の中に導入することにより、純粋客観でも純粋主観でもない相対的な観点が入り込む座標の存在が示された。
e. 純粋客観の視点は論理によってのみ与えられ観測によっては与えられない。すなわち純粋客観の世界は論理的な世界である。
f. コンピュータによって創られる「人工的」な世界と「自然」を絶対的な基準で区別することは不可能である。
g. コンピュータの中で世界を構築するという工学的な手法は複雑現象の理解へと到達するおそらく唯一の方法を与える。

自然科学と工学は、もはや一体として考えるべきであり、その区別は存在したとしても相対的なものである。そして、ヴァーチャル・エンジニアリングが、そのような観点で研究開発されることにより、新しい科学哲学や技術を生み出すことが期待できる。従って、新しい科学技術の観点としてのヴァーチャル・エンジニアリングの基本思想として、次のようにまとめることができる。

従来の観点では、力学的予測を諦めることは力学的法則が存在しないことを意味した。しかし、新しい観点では、力学的法則がたとえ存在したとしても、力学的予測はできないことが示される。従って、b. が示すように、現象理解や定式化と現象予測の間にも、単純な還元論的思考を働かせることはできないのである。次にd. により、観測結果は観測手段に相対的にしか意味をもたないので、単純に観測結果を「真理」と認めることは危険である。観測結果は、確かに特定の観測手段にもとづく「事実」を与えるが、それは観測行為に相対的な真理である。

このように、実験による客観性の獲得を諦めざるをえないが、しかし一方e. により、論理を信じ数学的構造を客観性の基準に据えることができる。そのためには、コンピュータでの実験やシミュレーションを複雑現象の理解にとって非常に大切なものと捉えるが、それは次のような理由による。

コンピュータ実験では、コンピュータは素過程として論理的に動いている。しかし一方で、コンピュータ内で起こる現象を理解したり、定式化したりするとき必要になることは、それら論理の筋をすべて追うことではなく、コンピュータ内の現象を通して養われた新しい直観に基づくことである。このとき、まず信じることは、素過程での論理の正当性である。これを信じた後、今までの物理学的直観とは異なるかもしれない新しい直観を養わなければならない。
たとえ、ある事実が我々の物理的直観に反したとしても、このような意味で「数学的事実」であれば、それを真理とみなすことをためらってはならないであろう。数学は、それ独自の自律運動を遂げているのであるが、それとは別に、複雑現象に対する数学的定式化はコンピュータの中で形式化するのが適当である。

また、f. とg. により、複雑現象をコンピュータの中で形式化、モデル化することで、我々は新しい型の「予測」が可能になることをみる。それは、系の構造と歴史に対する可能な過程を提示することで単一の現象の「先」を予測するのではなく（それ自体は、すでに示したように不可能である）、構造と歴史の集合を予測するのである。a. でみた認識は、科学の非力を意味するのではなく、自然の超

越性を意味しており、そのことにより、逆に我々の自然認識がf. とg. のような新たな方法を生み出す契機を与えたとみるべきである。そのようにして、複雑現象の理解をめざした研究から新しい自然認識へ到達することが期待できる。

それはまた、人工世界と自然との相対化であり、観測行為の新しい定式化をめざすものである。そのようにして組み立てられた仮想世界で新しい科学技術を開発することができると思われる。それが、ヴァーチャル・エンジニアリングの基本的な思想である。

4．2　ゲームとシミュレーション

コンピュータ・シミュレーションの分野には、物理化学問題のシミュレーションとは別の、ゲームの観点からのシミュレーション（ゲーミング・シミュレーションと呼ばれる）がある。本項では、このゲーミング・シミュレーションについて、その特徴を概観し、シミュレーション・システム開発の参考とする。ここで、シミュレーションについて、次のように考える。

シミュレーションとは、現実のあるいは提案されたシステム、プロセス及び環境が持つ中心的な特徴あるいは要素についての操作的モデルである。この考え方は、以下の側面を強調する。

- シミュレーションとは、ある形式のモデルである。
- そのモデルとは、静的なものでなく動的なモデルである。
- 対象システムの選択された要素のみが含まれている。
- 対象システムといっても、いくつかの異なる種類がある。

科学技術系のシミュレーションは、従来から研究開発の現場で広く普及して来たため、考え方が普及しているが、それ以外のモデルのシミュレーションについては今後の課題と考えられる。

例えば、地層処分性能評価シミュレーションは、漏出した核種が生活圏に到達しないことを、もしくは、例え到達しても長時間に亙る地層からの移行の結果、核種の放射線強度が天然の放射線源より下回る程度に崩壊していることを確認することに目的がある。

実際には、地層処分プロジェクトの観点からは、核種の移行に起因する非科学技術系の問題も考えられる。例えば、処分システムの建設と操業に係わる、安全性、経済性、社会心理等の“処分プロジェクトの及ぼす社会システムへのインパクト”も重要評価事項であると考えられる。そのような問題は、物理科学に基礎をおいた数理モデルによる記述という観点より、むしろゲーム性の強いモデルとなると思われるが、シミュレーションにおいては、システムの中心的特徴が同定され、モデルとして表現されることで、現実世界のシステムと同じような動きが現われる。

さらに、ゲーミング・シミュレーションの特徴有る応用として教育が上げられるであろう。例えば、地層処分性能評価のような動機（核種が生活圏まで漏出してこないか？）を確認するための複雑な科学は、一般人や学生には動機を理解できてもプロセスを理解することは難しい。そのような場合、ゲーミング・シミュレーションは、より解かりやすく説明することが期待できる。

（1）考え方の違い

ゲームという言葉は、全体的にせよ部分的にせよ、プレイヤーの意思決定に基づいて動作するシミュレーションに用いられる。ゲーム的側面を持つシミュレーション、すなわちゲーミング・シミュレーションにおいては、環境と参加者の活動とがゲームとしての特徴を持っている。

プレイヤーには、演ずべき役割、達成すべき目標、行うべき活動、何が成しうるかの制約、そして各プレイヤーの行動や、システム内での他の要素（偶然を含む）の作用の結果としての（正と負の）利得が与えられる。ゲーミング・シミュレーションでは、ゲームにおける役割、目標、活動、制約、結果、そしてそれらの結び付きが、現実世界から抽出され形作られる。先に用いた言葉でいえば、ゲーミング・シミュレーションは、現実世界のシステムのこれらの要素をシミュレートしているのである。例えば、立地問題などの土地利用ゲームを考えると、ある地域と、その周辺での土地利用の決定に影響を与える要因の中で、比較的基本的なもののいくつかを、ゲーミング・シミュレーションの中で抽象化して表現している。このときプレイヤーは、それらの要因により設けられている制約のもとでプレイする。

プレイヤーは、どのように資源を用いるかを意思決定することで、施設を建設し、運用し、維持する。ゲームの始めにプレイヤーには、税制、許される土地利用の範囲、雇用主の給与支払額と雇用者の所得、そして、企業と労働者のための交通・輸送費が情報として与えられるであろう。プレイヤーは、定期的に資産を再評価しなければならず、このとき減価した建物を修繕するかどうかの意思決定を行う必要がある。もしプレイヤーが、施設や地域活性にある程度以上の費用をかけていないと、プロジェクト推進が難しくなる可能性が高くなるであろう。

このような問題には、ゲーミング・シミュレーションの手法が物理科学のモデルと代わって重要となる。

（2）ゲームとシミュレーション

シミュレーションとゲームの間の区別は明確でない。この２つの言葉はしばしば同義語として用いられる。実際、風洞や宇宙船の例、あるいは医者や医療スタッフを訓練するための救急救命シミュレーションのような、ゲームでないシミュレーションは数多くある。同様に、いくつかのゲーム（特に多くのカードゲームやボードゲーム）は、シミュレーションではない。その構成要素が、現実世界のシステムやプロセスを表現するものとして設計されていないからである。

ゲーミング・シミュレーションは、ゲームとシミュレーションの双方の特徴を持つ、教育・訓練用の教材として開発されるソフトウェアの総称である。ゲームや略式のドラマのようなものとして考えるのではなく、参加者が外側でなく内側に入り込んだ、一種の事例研究のように考えるのがよい。

（3）ロール・プレイングとゲーミング・シミュレーションの相違

ロール・プレイングは、いくつかの重要な意味でゲーミング・シミュレーションとは異なる技法である。ロール・プレイングは、ゲーミング・シミュレーションの要素ではあるが、後者は他の構成要素も含んでいる。大部分のロール・プレイングにおいて、参加者は、ある役割を割り付けられるとともに、自分の置かれた状況についての一般的な説明を受ける。しかし、そこから先の行動は自由である。一方、ゲーミング・シミュレーションにおいては、役割は、相互に作用しているシステムにおい

て定義される。すなわち、他の役割と相互作用するような役割に力点が置かれる。

モデルは、動的な相互作用の基礎を形作る。さらに、ゲーミング・シミュレーションにおいては、特定の人々のパーソナリティや立場という点から、「憶測しなおすこと」はほとんどない。参加者は、シナリオや役割規定だけでなく、彼らの行動を方向付ける目標や目標達成のために利用できる資源、彼らが取ってよい、あるいは取ってはいけない行動を規定するルール、そしてプレイの順序について指示される。参加者の行動やルール違反の結果がどうなるか、また、参加者の役割や行動に組み入れられていない他のシステム要素が、どのように反応しそうかについても知らされる。参加者の行動が、望ましい最終目的を達成する上で成功か失敗か、目標が見える形でモデルが構築される。

このようにゲーミング・シミュレーションは、構造化や形式化の程度や、個々の役割演技よりも相互作用プロセスを強調する点において、ロール・プレイングと異なっている。その上、教育におけるロール・プレイングでは、多くの場合、数人の学生が参加する一方、クラスの残りの者が傍観者となってしまう。あるいは、複数のロールプレイを並行に行うために、クラスが小グループに分割されてしまうであろう。しかし、ゲーミング・シミュレーションをクラスの中で使う場合は、すべての学生が参加者となるので受け身の傍観者はいない。

（4）ゲーミング・シミュレーションと教育

ゲーミング・シミュレーションは、これまで数多くの教育・訓練目的のために、設計され、利用され、成功を収めている。その主なものは以下のとおりである。

- 動機づけと興味づけ
  主題とする事柄に関する動機づけと興味づけ。
  特定の話題をさらに研究するための動機づけと興味づけ
- 教育、情報の提供
  あるいは他の形式ですでに与えられた情報の強化事実全体像またはシステム的な理解。特定の役割のシステム全体に対する関係選択肢、政策、課題についてのより広い認識特定の政策や事象により起こりうる結果
- 技能開発
  批判的な思考と分析。意思決定、コミュニケーション技能企画書の作成、放送や新聞の記事原稿の執筆、予算書の作成といった特定の技能求職活動やチーム内労働者の管理、危機事態への対処といった将来発生するかも知れない特定業務への準備
- 態度変容競争や協調といった社会的価値他の役割にある人々に対する同感
- 自己による評価、他者による評価

個人の知識、技能、思い込み、態度。リーダーシップ能力に対する自覚参加者の知識、業務、思い込み、態度。リーダーシップ能力に対する教師（またはトレイナーや雇用主）の評価以上に挙げた目的の項目は、個々の教師やトレイナーの狙いの多くを含んでいるが、ゲーミング・シミュレーションだけが、これらの教育上の狙いを達成できる唯一の方法ではない。

実際に、講義、事例研究、ロール・プレイング、視聴覚教材などを通じて、教師やトレイナーは、これらの狙いを追求し、たいていは達成してきた。ゲーミング・シミュレーションは、ある種の目的

を達成するためのよりよい方法といえる。ゲーミング・シミュレーションは、目的により他の技法を代替することもあるが、他の技法と連係して用いると効果をより発揮するもので、一般的にいえば、教師やトレイナーがすでに持つ技法の「道具箱」に応用すべきものである。

（5）教育とゲーム

従来、教育や訓練の標準的な形は、講義だけか、あるいは講義と討論の組み合わせによるものであった。ここでは、このようなアプローチの限界をいくつか簡潔に示し、ゲーミング・シミュレーションによって、どのようにそれらを克服できるか示す。

基本的な問題点は、伝統的な講義形式においては、学習者が情報の単なる受け手であるということにある。教室内の学習者は、自分の考えを発表することはほとんどないので、彼らの関心は、頻繁に他の目的やより大きな関心のある事柄に移りやすい。このため、教師は複雑なトピックを述べる際には、適切な箇所より始めなければならず、終わりまでその道筋に沿っていかなければならない。このような直線的なアプローチでは、学習者にとって全体の本質を把握することが難しい。

このアプローチでは、学習者は、一度に一断片しか見ることができないことが問題である。直線的アプローチでは、決して全体の理解に結びつかない。このことは、事象と行動の“同時性”を理解させる場合に顕著である。

同様に、伝統的な教室での“議論”は、教師による講義のあとに、コメント、質疑、解答という形で行われる。そのような議論は、多くの場合、最も重要なコメントを持ったから人ではなく、グループの中で最もお喋りな人の意見から始まる。同じように、与えられた話題を考察するのに費やされる時間は、話の内容の相対的な重要性というよりは、話し手のこだわりの程度（そして、それを打ち切ることに対する教師の躊躇の程度）で決定される。さらに、議論が講義の中での最初の論点より始まり、話された順序で順番に進んでいくことはまれである。

したがって、多くの場合、質疑応答は論理的順序とはまったく関係なく行われる。３番目に、講義あるいは言語モデルを利用するときの問題点は、“システム”としての特性を明確に表現しづらいことにある。

与えられた要素や行動の間の多重の関連や、その帰結を言葉で表わしていくのは難しい。１つの社会システムを単にそれに飛び込んでいくだけでは、それがあまりに複雑すぎて、理解不可能なことが多いのであるが、一方で、言葉による記述では、過度の単純化を避けられない。

ゲーミング・シミュレーションによる教育はどのように違うのか。これには、いくつかの答えがある。第１に、ゲーミング・シミュレーションでは、講義とは反対に、教室のひとり一人は“能動的な学習者”である。彼らは意思決定をし、結果を受け入れ、立場を明確にすることで、システムが機能するようにしなければならない。プレイ後の事後分析において、何がなぜ生じたのか、それがどのように現実世界に対応しているのかということに焦点を当て、さらに、ゲーミング・シミュレーションの限界について議論する。

第１に、学習者は様々な“視野”から活動する機会が与えられるべきである。学習者は、情報の単なる受容者であってはならず、時として、発信者、判定者、交換者となるべきである。

第２に、どのような活動にも、それ自体の目的と動機があるべきで、活動が（成績のような）ある結末に至るための単なる手段であってはならない。即ち、効果的な学習環境の中では、活動は自己目

的的である。

第3に、学習者は権威への依存から解放されるべきであり、自分自身の力で考えることを許容されるべきである。このようにして、学習者は学習プロセスにおいて、より“生産的”になれる。

第4に、学習環境は学習者の活動に対して反応するようになっているべきである。環境は、単に学習者にフィードバックを与えるだけではなく、学習者が“思慮深く”自分の進歩を評価できるよう支援すべきである。

教育現場では講義を行い、その後で議論をするという形式でなく、むしろゲーミング・シミュレーションにおいてこそ、学習者は提供された論題の全体像を経験する。というのは、ゲーミング・シミュレーションでは、システムの構成要素は1つ1つ別々にではなく同時に与えられるからである。さらに、プレイの後で行う議論や分析は、無遠慮な発言や断定的な主張といったものでなく、通常、役割によって構造化される。

また、ゲーミング・シミュレーションは、特に“システム”としての特性を表現するのに有効である。システムというものは、言語モデルを通じてよりも、むしろ系統図や絵のような、図式モデル（graphic model）、あるいは物理的表現を使った方が明確に記述できると長い間認識されてさた。

たとえば、人体の血液循環システムは、言葉で表現するよりも系統図で書いた方が、はるかに分かりやすいことを考えてみるとよい。シミュレーション、すなわち作動しているシステムのモデルは、図式モデルよりも、教育用具として、さらに大きな可能性がある。

ゲーミング・シミュレーションの教育学的成果を評価する研究は数多い。ゲーミング・シミュレーションによって、教育上の問題のすべてが解決されるわけではないが、非常に多くの社会システムや社会問題について、教育や学習のための刺激的な機会が提供されることは明らかである。

ゲーミング・シミュレーションは、教室の中だけにとどまらず活用されている。というのは、多くの公共政策の論議では、意思決定者にとっては全体的な見通しとシステム的な理解が必要となるが、これらはゲーミング・シミュレーションを行うことで得られるからである。

（6）ゲーミング・シミュレーションとコンピュータ・シミュレーション

抽象化の程度としては、ゲーミング・シミュレーションは、ロール・プレイングとコンピュータ・シミュレーションの間に位置する。コンピュータ・シミュレーションでは、構造やプロセスについての数学的モデルを詳しく展開するためにコンピュータを利用する。システム内の様々な関係を表現するために、数学的記号や方程式が使われ、時間の経過に対する変化を調べるためにモデル方程式に示す通りの演算を繰り返し行う。

例えば、特定の問題解決にモンテカルロ・シミュレーションとゲームのどちらかを選ぶ場合には、2つの方法論の長所と短所を考慮しなくてはならない。シミュレーションは、ランダムなシナリオ、極端なシナリオ、統計的出力分布といったものをすばやく容易に作り出す能力がある。

ところが、ゲームは、ただ一度あるいは統計的に有意でない数回の反復しかできない。シミュレーションは、経営者が対処すべき問題の範囲を理解するのに役立つ計算結果を与える。反対に、ゲームは、「実用的」な感じを与えるとともに、シミュレーション設計者が仮定した人々の反応ではなく、実際に人々がどのように反応するかを知る機会を与えてくれる。

分析技法およびシミュレーション技法が、構造の比較的単純な競争状況および紛争状況をモデリン

グするのに適しているのに対して、ゲームでは、現実の状況に特有の多元的な複雑性を導入し、理解することが可能になる。コストを考えると、ゲームは、設備やスタッフへの投資を必要とするだけでなく、膨大な事柄を考察しなくてはならなくなる可能性がある。シミュレーションは、統計的に有意なデータを得るために、膨大な大型コンピュータ運用費用が必要となる可能性が高い。一般的にゲームには次のことがいえる。

> **それは、人間の反応や系の相互作用が予想できなかったり、人間が研究対象のプロセスの一部となっているような探索型研究に、専門家の考えを引き出すために、状況の本質、特に競争的状況の本質を理解させるための訓練に適している。**

シミュレーションは、プロセスの最初か最後に人間の意思決定入力があるような状況や、問題が比較的単純な状況に適している。

5　動画化シミュレーション・プログラムの試作

本章では、試作した動画化シミュレーション・プログラムについて仕様を中心に記述する。動画化シミュレーション・プログラムは、ウィンコンシン大学マジソン校の宇宙科学技術センターで試験的に開発された、Vis5Dと呼ばれるソフトウェア・システムを地層処分性能評価シミュレーション・コードで利用できるように改訂して試作したものである。改訂して試作したソフトウェアを「動画化シミュレーション・プログラム」と呼ぶこととする。

動画化シミュレーション・プログラムは、5次元の長方形空間を持つデータにより動作する。データは“格子”の各地点における実数であり、3つの空間次元と1つの時間次元、および複数の物理的な変数を列挙するための次元からなる。動画化シミュレーション・プログラム・システムには、動画化シミュレーション・プログラム、視覚化プログラム、5次元データを管理・分析するための幾つかのプログラム、データのファイル書式を変換するコード、ソースコードのサンプル、及び動画表現の仮想世界を構築する処分システム・オブジェクト等が含まれる。

5．1　プログラム仕様

以下に、動画化シミュレーション・プログラムの実行に必要なハードウェアとソフトウェアについて、及びシステム環境仕様について示す。

（1）システムの必要条件

動画化シミュレーション・プログラムは、以下のシステム環境で動作する。メモリー容量は最低32MB以上が望ましく、グラフィクス機能には最低8ビットのカラーディスプレイが必要である。

・シリコングラフィックス社ワークステーション
　IRIXのVer.4.0.1もしくは、それ以上のもの。
・IBMワークステーション　RS6000
　モデル320H以上、AIXのVer.3もしくは、それ以上のもの。
・ヒューレットパッカード社ワークステーション　7000または9000シリーズ
　XP-UX A.09.01もしくは、それ以上のもの。
・DECワークステーション　アルファ
　OSF/1 V.1.3もしくは、それ以上のもの。
・ライナックスと互換性があるIBMのPC
　Pentium 75MHzもしくは、それ以上のもの。または、ライナックス1.0もしくは、それ以上のもの。

（2）動画化シミュレーション・プログラムの構成

動画化シミュレーション・プログラムは、“Vis5d-4.2”ディレクトリに格納されている次のファイル及びサブディレクトリから構成されている。

README	READMEテキストファイル
README.ps	READMEポストスクリプト・ファイル
NOTICE	GNUパブリック・ライセンス・ファイル
PORTTING	Vis5Dポーティングノートファイル
LAMPS.v5d	サンプルデータファイル（LAMPS）

SCHL.v5d	サンプルデータファイル（ボブ・シュルシンガーの雷雨モデル）
OUTLSUPW	世界大陸地図ラインファイル
OUTLUSAL	米国地図（州未分割）ファイル
OUTLUSAM	米国地図（州分割）ファイル
EARTH.TOPO	地球地形ファイル
api.htm	Vis5dのAPI（Application Program Interface）に対するハイパーテキスト
script.html	Vis5dのスクリプト言語に対するハイパーテキスト
*.tcl	例題スクリプト
Vis5d	Vis5d視覚化ログラム
v5dappend	v5dファイル追加用
v5dinfo	v5dファイル・インフォーメーション・ユーティリティ
v5dstats	v5dファイル・統計用・ユーティリティ
v5dedit	v5dファイル・見出し編集用・ユーティリティ
v5dsimport	v5dファイル・変換用・ユーティリティ
gr3d_to_v5d	McIDAS GR3Dファイルのv5dフォーマット変換用・ユーティリティ
comp_to_v5d	comp5dファイルのv5dフォーマット変換用・ユーティリティ
listfonts	SGIのフォントIRIS GL用配列・ユーティリティ
src/	Vis5dのソースコード格納サブディレクトリ
util/	Vis5dのユーティリティ格納サブディレクトリ
lui5/	LUIユーザーインターフェイス・ライブラリ格納サブディレクトリ
Mesa/	Mesa3Dライブラリ格納サブディレクトリ
import/	v5dsimportソースプログラム格納サブディレクトリ
userfuncs/	ユーザー作成解析機能格納サブディレクトリ
contrib/	コントリビューションファイル格納サブディレクトリ
convert/	サンプルデータ変換用ソースコード格納サブディレクトリ

動画化シミュレーション・プログラムの動作は最低32MBメモリー必要とする。システム環境のメモリーが32MB以上あるならば、動作のための設定値を変更することが望ましい。設定値の変更はサブディレクトリ/src内のvis5d.hインクルードファイルのMBSパラメータ値を変更する。また、地系図およびマッピングファイルの省略値を変更する場合は、サブディレクトリ/src内のvis5d.hファイルのTOPOFILEまたはMAPFILEパラメータを変更する。なお、vis5d.hファイルの変更後は、プログラムを再コンパイルする必要がある。

動画化シミュレーション・プログラムをネットワーク上で複数のユーザーで利用するならば、Vis5dプログラムと各データファイルを一般的に使用される/usr/localといったサブディレクトリ内に移動したほうが利用上都合がよい。その場合、vis5d.hファイル内容に各ファイルが何処に位置（保存）しているか変更しなければならない。

（3）動画化シミュレーション・プログラムへのデータ入力

動画化シミュレーション・プログラムは、5次元の長方形空間を持つデータにより動作する。5次元については、まず3次元として（行（横）、列（縦）、高さ（または緯度、軽度））の立体空間、4次元は時間であり、5次元目は、気温、気圧、水の容量等、複数の物理的な変数を列挙したものである。

動画化シミュレーション・プログラムでデータファイルを表示するためには、必要なパラメータ（行、列、高さ、タイムステップ等）を設定する。例えば、地理的な位置と方位（マッピングの投影図）、変数の名称、各タイムステップや現在の日時等である。

動画化シミュレーション・プログラムは、v5d形式とcomp5d形式のファイルに対応し、それぞれは圧縮された3次元書式データであることから処理速度が早く効率的である。

動画化シミュレーション・プログラムに外部の解析コードを組み込む（接続）ためには、v5d形式

ファイルに変換するプログラムの作成が必要である。v5d形式ファイル変換は、v5dライブラリの機能を用いて行うことができる。次に示す４つのサンプルプログラムは、動画化シミュレーション・プログラムへのデータ入力例を示したものである。これらは、convert/サブディレクトリに格納されている。

foo_to_v5d.f	投影する長方形を想定し線による垂直座標を設定（FORTRAN）
foo2_to_v5d.f	投影する垂直座標に合致させるために各変数に垂直レベルの異なった数値を与える（FORTRAN）
foo_to_v5d.c	投影する長方形を想定し線による垂直座標を設定（C）
foo2_to_v5d.f	投影する垂直座標に合致させるために各変数に垂直レベルの異なった数値を与える（C）

変換プログラムを作成するにあたっては、v5dCreate（またはv5dCreateSimple）、v5dWrite、v5dCloseの機能を使用する。v5dCreateとv5dCreateSimpleはv5d形式ファイルを作成する。v5dCreateは作成のための基本パラメータを設定するのみであり、v5dCreateSimpleはv5dCreateより複雑なパラメータを与えることができる。作成機能は、CおよびFORTRANバージョンがある。

（4）FORTRAN言語による呼出し可能な機能

以下にFORTARN言語とのインターフェイス仕様を示す。

a) v5dCreateSimple

```
integer function v5dcreatesimple( name, numtimes, numvars, nr, nc, nl,
                          varname, timestamp, datestamp, northlat, latinc,
                          westlon, loninc, bottomhgt, hgtinc )
character* (*) name
integer numtimes
integer nr
integer nc
integer nl
character*10 varname(MAXVARS)
integer timestamp(*)
integer datestamp(*)
real northlat
real latinc
real westlon
real loninc
real bottomhgt
real hgtinc
```

b) v5dCreate

```
integer function v5dcreate( name, numtimes, numvars, nr, nc, nl, varname,
                            timestamp, datestamp, compress, projection,
                            proj_args, vertical, vert_args )
character* (*) name
integer numtimes, numvars
integer nr
integer nc
integer nl(*)
character*10 varname(MAXVARS)
integer timestamp(*)
integer datestamp(*)
integer compress
integer projection
```

```
real proj_args(*)
integer vertical
real vert_args(*)
```

（5）C言語による呼出し可能な機能
以下にC言語とのインターフェイス仕様を示す。

a) v5dCreateSimple

```
int v5dCreateSimple( name, numtimes, numvars, nr, nc, nl, varname,
                     timestamp, datestamp, northlat, latinc, westlon,
                     loninc, bottomhgt, hgtinc )
char *name;
int numtimes;
int numvars;
int nr, nc, nl;
char varname[MAXVARS][10];
int timestamp[], datestamp[];
float northlat, latinc;
float westlon, loninc;
float bottomhgt, hgtinc;
```

b) v5dCreate

```
int v5dCreate( name, numtimes, numvars, nr, nc, nl, varname, timestamp,
                     datestamp, compress, projection, proj_args,
                     vertical, vert_args )
char *name;
int numtimes, numvars;
int nr, nc, nl[];
char varname[MAXVARS][10];
int timestamp[], datestamp[];
int compress;
int projection;
float proj_args[];
int vertical;
float vert_args[];
```

v5dCreateとv5dCreateSimpleの使用による変数：

name	v5dファイル作成の名称
numtimes	タイムステップの数量（最低1回）
numvars	変数の数量（最低1回）
nr	全ての立体格子における行数（最低2回）
nc	全ての立体格子における列数（最低2回）
varname	変数の名前の配列： varname(1)=最初の変数名 vername(2)=二番目の変数名 … vername(numvars)=最後の変数名
timestamp	HHMMSS書式のタイムステップに対する時間ラベルの配列： timestamp(1)=最初のタイムステップの時間 timestamp(2)=二番目のタイムステップの時間 … timestamp(numtime)=最後のタイムステップの時間
datestamp	YYDDD書式のタイムステップに対する日付ラベルの配列： datestamp(1)=最初のタイムステップの日付 datestamp(2)=二番目のタイムステップの日付 … datestamp(numtimes)=最後のタイムステップの日付

関数v5dCreateSimpleのみによって使用される変数：

nl	全ての立体格子におけるレベル数（最低1回）
northlat	段階においてのボックスの北端の緯度

atinc	段階においての横列間の増加分（正）
westlon	段階においてのボックスの西端の経度（正の西経度）
loninc	段階においての縦列間の増加分（正）
bottomhgt	kmにおけるボックスの底部の境界線
hgtinc	kmにおけるレベル間の増加分（正）

関数v5dCreateのみによって使用される変数

nl　立体格子における変数毎のレベル数：
- nl(1)=最初の変数に対するレベル数
- nl(2)=二番目の変数に対するレベル数
- ...=...
- nl(numtimes)=最後の変数に対するレベル数

compress　圧縮モード（各格子点毎1、2または、4バイト）

projection　射影の型の表示：
- 0=線形、長方形、総称ユニット
- 1=線形、長方形、円筒からの等距離
- 2=ランベルト正角図法
- 3=立体（投射）
- 4=循環された

proj_args　投射変数

projection=0ならば
- proj_args(1)=立体ボックスの北境界線
- proj_args(2)=立体ボックスの西境界線
- proj_args(3)=行間の増分
- proj_args(4)=列間の増分

projection=1として、それ以外は
- proj_args(1)=立体ボックスの北緯束縛
- proj_args(2)=立体ボックスの西経束縛
- proj_args(3)=行間の増分
- proj_args(4)=列間の増分

projection=2として、それ以外は
- proj_args(1)=標準緯度1
- proj_args(2)=標準緯度2
- proj_args(3)=北／南極の行
- proj_args(4)=北／南極の列
- proj_args(5)=列に対する水平並列
- proj_args(6)=km列間の増分

projection=3として、それ以外は
- proj_args(1)=中心の緯度（度）
- proj_args(2)=中心の経度（度）
- proj_args(3)=投射の中心の行
- proj_args(4)=投射の中心の列
- proj_args(5)=中心における列間の行送り

projection=4として、それ以外は
- proj_args(1)=回転球体上の北境界線
- proj_args(2)=回転球体上の西境界線
- proj_args(3)=行間の増分
- proj_args(4)=列間の増分
- proj_args(5)=(0,0)に対応する地球緯度
- proj_args(6)=(0,0)に対応する地球経度
- proj_args(7)=回転角度

endif

vertical　縦の座標系の型の表示：
- 0=総称ユニットにおいて均等に配置されたレベル
- 1=kmにおいて均等に配置されたレベル
- 2=kmにおいて不均等に配置されたレベル

vert_args　縦の座標系の変数：

vertical=0ならば
- vert_args(1)=底辺レベルの高さ
- vert_args(2)=レベル間に設置

vertical=1として、それ以外は
- vert_args(1)=kmにおける底辺レベルの高さ
- vert_args(2)=kmにおけるレベル間に設置

```
vertical=2として、それ以外は
  vert_args(1)=格子レベル1（底）の高さ（km）
  vert_args(2)=格子レベル2の高さ（km）
  vert_args(n)=nがnl配列において最大値にあるときの、格子レベルN（頂点）
               の高さ（km）
vertical=3として、それ以外は
  vert_args(1)=格子レベル1（底）の圧力（mb）
  vert_args(2)=格子レベル2の圧力（mb）
  vert_args(n)=nがnl配列において最大値にあるときの、格子レベルN（頂点）
               の圧力（mb）
endif
```

（6）関数v5dWrite機能

v5dWrite機能は、データの単一立体（3D）格子をv5dファイルに書き込むようになっている。格子はタイムステップと物理的可変数によって識別される。v5dWriteの概要は以下の通りである。

FORTRAN言語呼び出し可能な機能：

```
integer function v5dwrite( time, var, data )
integer time
integer var
real data(*)
```

C言語呼び出し可能な機能：

```
int v5dWrite( time, var, data )
int time;
int var;
float data[];
```

変数処理機能：

time	範囲内でのタイムステップ数 [1..numtimes]
var	範囲内での可変数 [1..numvars]
data	格子値の立体配列；値の数値=nr*nc*nl(var)で、データ [row+nr* (col+nc*lev)] として整列しており、行は北から南へ、列は西から東へ、レベルは底から頂点にそれぞれ上昇する。

（7）関数v5dClose機能

v5dCloseは、最後の格子が書き込まれた後にv5dファイルを閉じる機能である。変数は必要ない。v5dCloseの概要は以下の通りである。

FORTRAN言語呼び出し可能な機能：

```
integer function v5dclose
```

C言語呼び出し可能な機能：

```
int v5dClose()
```

各作成機能は、正常終了は1を返し、エラーが発生すると0を返す。データ変換プログラムのどの例を見ても、v5dCreate/v5dCreateSimpleへの変数に対応した可変数がある。これらの可変数を初期化するかはユーザーの判断次第である。

例えば、データファイル内のnumtimes（タイムステップの数）にnumvars（可変数の数）を割り当てるとする。これらの可変数を全て初期化した後、v5dCreate（または、v5dCreateSimple）コールは、v5dファイルを作成する。もし、可変数の初期化のいづれかを失敗したならば、それに該当するエラーメッセージが出される。

次に、変換プログラムはループ状態をつくる。そのループの内に、適切なタイムステップと物理的

な可変数に対するデータを読み込むコードを挿入しなくてはならない。指定された配列の中にデータが読み込まれるとv5dWriteコールはデータ圧縮を行い、v5dファイルへ書き込む。全データが書き込まれた後に、v5dCloseは呼び出される。変換プログラムのコンパイルを終了したならば、データファイルでテストし、v5dファイル上のv5dinfoとv5dstatsユーティリティプログラムの動作を確認する。

以下に典型的な値の例を示す。もしもどれかがfoo_to_v5d.fプログラムを使用していた場合、各可変数に割り当てられる可能性がある。

＝割当＝	＝コメント＝
numties=5	5　タイムステップ
numvars=4	5　物理的可変数
nr=30	各立体格子内の30行
nc=40	各立体格子内の40列
nl=20	各立体格子内の20レベル
varname(1)="U"	U（x方向速度成分）
varname(2)="V"	V（y方向速度成分）
varname(3)="T"	W（z方向速度成分）
varname(4)="P"	圧力
varname(4)="T1"	温度
timestamp(1)=140000	2:00:00 pm
timestamp(2)=141500	2:15:00 pm
timestamp(3)=143000	2:30:00 pm
timestamp(4)=144500	2:45:00 pm
timestamp(5)=150000	3:00:00 pm
datestamp(1)=97036	1997年の36番目の日（2月5日）
datestamp(2)=97036	〃
datestamp(3)=97036	〃
datestamp(4)=97036	〃
datestamp(5)=97036	〃
northlat=60.0	ボックスの北境界線は緯度30度にある
latinc=1.0	各30行間で緯度1度ある
westlon=100.0	立体ボックスの西境界線は経度100度にある
loninc=0.5	各40列間で経度0.5度ある
bottomhgt=-500.0	ボックスの底は-0.5kmにある（海抜レベル）
hgtinc=1.0	各20格子レベル（頂点は19.0km）の間1km

行、列、レベル、タイムステップ、変数の各数値は、データポイントの総数である。この例としては、30*40*20*5*4=480,000である。実数のデータファイルは、20レベルからなる100列によって100行となり、50タイムステップと合計100,000,000データポイントに対する10の変数を持つ。

foo_to_v5dプログラム（v5dCreateSimpleを使用する）とfoo2_to_v5dプログラム（v5dCreateを使用する）における相違は、後にどのような射影、縦の座標系、各物理的変数に対する異なった格子レベルの数値を指定し、データ圧縮制御を行うかによる。射影を表示するのに、projection値を0,1,2,3のどれかに設定し、その後proj_args配列において投射従属パラメータを指定する。縦の座標系の指定も同様である。

各変数に対する異なった格子レベルの数値を設定することも必要に応じ有用である。例えば、殆どの変数が30の格子レベルを持つが、もし、その一部はより低い格子レベルで、例として、たった１つだったとしよう。動画化シミュレーション・プログラムは無視するか、または疑似のデータ値を与えて特別のレベルを満たしてしまうこととなる。v5dCreateの機能を用いて、格子レベルがnl配列パラメータを持った、各々が独立した物理的変数が幾つあるかということを指定することができる。例えば、もし格子がC列とR行で、データ配列内の値の数値が、変数VがC*R* nl (V)に等しくなるようにv5dW

riteに指定した場合、省略時設定によって、各変数に対する最下部の格子レベルが立体ボックスの底に表示される。各格子は多くのレベルがあるにもかかわらず上方向に拡張する。しかしながら、時々、特定の変数の最下部の格子レベルは上昇するよう位置付けられる。これは地下水流れデータの例である。

格子数は（30x40x20）であり、P（圧力）とT1（温度）という名前の２つの変数を持つことになる。これはどのようにlowlevの配列が、この状況に対する解になるかを示す例である。

vernum	varname(varnum)	nl(varnum)	lowlev(varnum)
1	S	8	0
2	T	8	0
3	P	10	8
4	T1	10	8

lowlevの配列はv5dCreate機能では指定されない。それは、lowlevがv5d Create機能が確立された後に開発されたためである。それに対して新v5d SetLowLev機能は、LowLev配列とともに呼び出される。この分離した機能は、その呼び出し順序を変えることなくv5dCreateの機能性を拡張するために付け加えられたものである。

（８）関数v5dSetLowLev機能

v5dSetLowLevは、v5dCreateとv5dCloseの間のどのポイントでも呼び出すことができる。

v5dSetLowLev機能の概要は以下の通りである。

FORTRAN言語呼び出し可能な機能

```
integer function v5dsetlowlev(lowlev)
integer lowlev(*)
```

C言語呼び出し可能な機能

```
int v5dsetLowLev(lowlev)
int lowlev[];
```

可変数の描写

```
lowlev    各変数に対して格子レベルにおいて縦のオフセットを特定する。
           lowlev(1)=最初の可変数に対してオフセットする
           lowlev(2)=2番目の可変数に対してオフセットする
                 ...  =  ...
           lowlev(numvars)=最後の可変数に対してオフセットする
```

v5dCreateとv5dcreateの機能で、どのように格子データを圧縮するかを制御する。省略時設定は、格子の値に対して１バイトの整数になるよう線形に設定される。これは、ほとんどのデータに機能し、それはタイムステップ、変数、整数レベルの各結合に対して、その基準の要因が独立して選択されるためである。

更に、格子点毎に１バイトに圧縮することにより、データをメモリ内に常駐させ、動画化シミュレーション・プログラムの高度の対話式利用を可能にする。v5dCreateとv5dcreateの機能の圧縮の引数は、格子点値が１バイトか２バイトの整数に設定するか、または４バイトの浮動小数点値（圧縮されない）として残すかを選択することができる。まず１バイトの整数に圧縮し、その後１バイトで精度に問題があれば、２または４バイトを使用する。

動画化シミュレーション・プログラムは、データファイル内の各変数に対して物理的な装置を特定させる。

v5dSetUnits()機能は２つの引数を獲得する。それらは変数の数値と文字列単位である。もしもファイル内の最初の変数がPで単位がミリバールならば、以下により設定することができる。

```
C:v5dSetUnits(1,"millibars")
Fortran:call v5dsetunits(1,"millibars")
```

v5dCreate、v5dWriteおよびv5dCloseを使用するプログラムをコンパイルするためには、src/v5d.oとsrc/binio.oファイルを関連づけなくてはならない。最終的に、データが地下水モデルによって生成されるとすれば、v5d Create、v5dWrite、v5dCloseの機能を使用し、v5dファイルディレクトリを生成するモデルを修正する。

５．２　固有の座標系

動画化シミュレーション・プログラムは、新しいマッピングと縦の座標系の考え方を提供する。マッピングの項を使用する場合、地球の緯度／経度への立体格子内のデータの行と列間の関係を照会する。縦の座標系の項は、大気中の高度（または海中、地層中の深度）への立体格子内のデータの縦のレベルを表現する。マッピング機能は以下の通りである。

（１）包括的な直線

これは線形の、規則正しく配置された暗黙の装置を持たない座標系である。このシステムは、データが地球科学に関連しない時は有用である。南北の座標は上方向に、東西の座標は左へそれぞれ拡張する。投射は４つのパラメータによって確定する。

NorthBound	立体ボックスの北境界線
WestBound	立体ボックスの西境界線
RowInc	格子列間の増分（行送り）
ColInc	格子行間の増分（行送り）

例：立体格子は80行と60列、NorthBound＝100.0 m, WestBound＝50.0 m, RowInc＝0.5 m, ColInc＝0.5 mを持っている。その時、

南境界線は60.5メーターの地点にある。例えば、southbound=NorthBound－ (RowInc*(rows-1))
東境界線は79.5メーターの地点にある。例えば、eastbound=NorthBound＋ (ColInc*(columns-1))

（２）直線を成す緯度／経度（円柱から等距離）

緯度は北へ（図形表示の中で上方向に）、経度は西へ（図形表示の中で左方向に：正の西緯度）それぞれ拡張する。投射は４つのパラメータによって確定される。

NorthBound	[90S、90N] の領域の緯度での立体ボックスの北境界線
WestBound	[-180E、180W] の領域の経度での立体ボックスの西境界線
RowInc	0より高い緯度での格子行間の増分（行送り）
ColInc	0より高い緯度での格子列間の増分（行送り）

例：立体格子が30行と60列、北束縛=70.0、西束縛=140.0、行の増分=1.0、列の増分=0.5を持っていたならば、

南境界線は緯度41度の地点になる。例えば、（Northbound - RowInc*(rows-1)）
東境界線は経度110.5度の地点になる。例えば、（westbound=NorthBound - ColInc*(columns-1)）

（3）ランベルト図法

以下の6つのパラメータにより確定された円錐の投射。

Lat1, Lat2	[90S、90N] の領域の、1番目と2番目の標準緯度。緯度1と緯度2は虚数の円錐が地球の球体を交差するところで確定する。緯度1と緯度2は同じ符号を持っており、共に正か負に統一されてなくてはならない。また更に緯度1は緯度2と同等かそれ以上でなくてはならない。
PoleRow, PoleCol	これらのパラメータは立体格子座標系に関連する北か南の極の地点を示す。緯度1と緯度2が正ならば、北の極は圧縮され、さらに南の極も圧縮される。
CentLon	中央の緯度：このパラメータは、地球の緯度が立体格子列になるよう並列される役割を表す。
ColInc	kmの中央の緯度と標準経度における格子列間での増分（行送り）

例1：立体格子が35行と40列を持つ、ウィスコンシンを中心とした米国のランベルト正角図法。

```
Lat1=70.0
Lat2=20.0
PoleRow=-35.0
PoleCol=20.0
Central Longitude=90.0
ColInc=100.0
```

例2：立体格子が35行と40列を持つとした場合のオーストラリア上のランベルト正角図法。

```
Lat1=-20.0
Lat2=-70.0
PoleRow=60.0
PoleCol=20.0
Central Longitude=-130.0
ColInc=200.0
```

備考：極がランベルト正角図法において明白な時は、普通は範囲を定めない（例えば、Longitude is> 180AND<-180）V字型の領域（極においての先端と共に）があるということを留意すべきである。この領域では、マッピングラインも地形図も正確でない。

（4）立体画法の方位

立体方位画法は5つのパラメータで範囲が定まる。

CentLat, CentLon	投射の中心の緯度と経度。虚数の円錐の先端はこの座標を超える。
CentRow, CentCol	投射の中心の行と列。格子の縦横の表示は投射の中央に位置している。これらの値は立体ボックスの外側になるだろう。
ColInc	kmの格子列間での増分（行送り）。このパラメータは投射の基準を管理して

いる。

例：立体格子が40の行と列を持ち、北極を中心とした立体方位画法を求める。

```
CentLat=90.0
CentLon=0.0
CentRow=20.0
CentCol=20.0
ColInc=200.0
```

（5）回転した直線構成の緯度／経度

長方形の緯度／経度座標系で、地球の緯度／経度に関連した球体上にあるとする。南北座標は、球体上で上方向に、東西座標は左方向にそれぞれ拡張する。その投射は7つのパラメータで確定する。

NorthBound	[-90S、90N] の領域の緯度での立体ボックスの北境界線
WestBound	[-180E、180W] の領域の経度での立体ボックスの西境界線
RowInc	0より高い緯度での格子行間の増分（行送り）
ColInc	0より高い緯度での格子列間の増分（行送り）
CentLat,CentLon	回転した球体上の緯度／経度=(0,0)へ結合する地球の緯度と経度。
Rotation	約(0,0)点である回転した球体の時計回りの角度

例：小さい地域上では地球はほぼ平坦であり、この縮小モデルに対して四角に近い格子を作成するため、この機能を利用することができる。例えば、縦横41の四角に近い格子を発生させ、以下をもってウィスコンシンを更に小領域分割することができる。

```
NorthBound=2.0
WestBound=2.0
RowInc=0.1
ColInc=0.1
CentLat=43.0
CentLon=90.0
Rotation=0.0
```

5．3　座標系

動画化シミュレーション・プログラムは、以下の座標系の考え方を提供する。

（1）均等に配置された構成単位

これは線形の縦の座標系で、立体格子内のレベルを均等に配置する。設定された単位は何も含まれない。その座標系は2つのパラメータで確定される。

BottomBound	底部の境界線、立体ボックスの境界線
LevInc	格子間の増分（行送り）

例：立体格子が20レベルで、0.0メーターになる底部の境界線と、レベル間での0.1メーターを仮定する。

```
BottomBound=0.0
LevInc=0.1
```

（2）均等に配置された距離

線形の縦の座標系であり、格子レベルは均等に配置されている。座標系は2つのパラメータで確定される。

BottomBound	kmの立体ボックスの底部の境界線
LevInc	0以上のkmの格子間の増分（行送り）

例：立体格子が20レベルで、格子レベル間で0.5キロメーターを仮定する。

```
BottomBound=0.0
LevInc=0.5
```

（3）不均等に配置された距離

線形の縦の座標系で、格子レベルが不均等は配置する。座標系はNの高度パラメータの配列によって確定される。Nは立体格子のレベルの数値である。格子レベルの数値が各変数毎に異なると、Nは格子レベルの最大値になる。

Height(1)	kmの1番目（底）の格子レベルの高さ
Height(2)	kmの2番目の格子レベルの高さ
...	...
Height(N)	kmのN番目（頂点）の格子レベルの高さ

高さの値はNと共に上昇しなくてはならない。

例：立体格子が10レベルで、格子レベルが頂点よりも底に近く、より接近して配置すると仮定する。

```
Height(1)=0.0
Height(2)=0.1
Height(3)=0.2
Height(4)=0.3
Height(5)=0.4
Height(6)=0.6
Height(7)=0.8
Height(8)=1.0
Height(9)=1.3
Height(10)=1.6
```

対数の記数法で縦軸の表示を可能にする。これは対数のコマンドラインの選択で、プログラムの開始時に行う。この場合、縦軸は高さに連動した対数だが、圧力に連動した線形ではない。高さ（H）と気圧（P）の関係は以下の通りである。

```
P=1012.5*e^(H/-7.2)      (^denotes exponentiation)
H=-7.2*Ln(P/1012.5)      (Ln denotes natural log)
```

1012.5と-7.2の定数は、対数の選択時に置き換え可能な省略値である。

（4）不均等な気圧配置

これは線形の縦の座標系で、格子レベルが不均等に配置する。座標系はNの高度パラメータの配列によって確定される。Nは立体格子のレベルの数値である。格子レベルの数値が各変数毎に異なると、Nは格子レベルの最大値になる。

```
Pressure(1)     kmの1番目（底）の格子レベルの高さ
Pressure(2)     kmの2番目の格子レベルの高さ
   ...          ...
Pressure(3)     kmのN番目（頂点）の格子レベルの高さ
```

高さの値はNとともに下降しなくてはならない。再配置計算の目的で、動画化シミュレーション・プログラムは高さ（H）と気圧（P）の間の関連を仮定する。

```
P=1012.5*e^(H/-7.2)      (^denotes exponentiation)
H=-7.2*Ln(P/1012.5)      (Ln denotes natural log)
```

v5dファイルの形式だけが、新しいマッピングと縦の座標系の情報を繋ぎ合わせる機能を持っている。v5dファイルがプログラムに読み込まれた時、この情報は地形図、マッピングラインおよび速度の軌道の計算を準備する。

プログラムは他に2つの投射表示も支援する。それらは球状と円柱状である。長方形の立体ボックスを描く代わりに、これらの投射は実際に立体ボックスを球状か、または円柱の形態に曲げる。これらの投射は、球状、または円柱状の値をもって投射選択を指定することによって使用される。

球状の選択は、立体の球体上のデータを表示するため使用される。円柱状の選択は、平坦で円形の地形図上のデータを表示するため使用される。それはLAMPSデータファイルをサンプルとして使用し検証するとよい。

5．4　変数とデータ

情報の分析と視覚化は、動画化シミュレーション・プログラムの重要部分である。例えば、データファイルの内容が、U、V、Wという名前の変数ならば、情報がベクトルの3つの構成要素になると仮定し、等高線と速度ベクトルを表示するよう使用される。正のU値は東へ、負のU値は西へ動く。正のV値は北へ、負のV値は南へ動く。正のW値は上へ、負のW値は下へ動く。U、V、Wに対する単位は、一般的な射影か縦の座標系が使用される時以外は、秒毎のメートルになると仮定する。その場合、単位は毎秒Xになる。Xはnorthbound、westbound、rowincおよびcolincのパラメータに使用される単位である。

もし、速度ベクトルの構成要素に対するU、V、Wを使用したくなければ、命令ライン上で他の速度の変数名を設定するか、または動作中にそれらを入力することもできる。

厳密に言えば、U、V、Wが速度の動きを表す必要はない。それらは、地下水のようなどんな流れの場をも表すよう使用することができる。しかしながら、視覚化の目的のためのある定数によって、U、V、Wを基準化するべきである。

動画化シミュレーション・プログラムは、どんな格子データ値でも、不確定か“紛失”を扱うことができる。例えば、観察に基づくデータファイルが不完全で、間違った値を含むことが往々にしてあ

るが、データ変換プログラムの中で、格子値がその値を1.0e30以上に割り当てることによって失われるということを示すことができる。失われたデータは、同一表面内の輪郭スライスとして、色彩スライス内の黒い領域として現われる。また、データプローブは失われたデータを“Missing”として表示することができる。

（1） McIDAS－3D格子データファイル

McIDAS GR3Dファイルは、データの立体格子書式の一つである。立体格子は、単一時間に複数の物理的変数の値を列挙するため、短い順序に構成されている。物理的変数の短い順序は、より長い順序に繰り返され、多くのタイムステップを通しての段階を経る。

これらのファイルは、GR3Dnnnnの型の名前を持ち、そのnnnnは0001と9999の間の4桁のデジタル数字である。

立体格子ファイルは、各立体格子へのディレクトリ・エントリを有し、それは格子のサイズと地形の位置、データ、格子配列のデータの物理的変数の時間と名称を表示している。5次元データファイルは、全て同じサイズと地形の位置において、立体格子ファイル内の立体格子の分類から成る。格子の分類は、同類の物理的変数で、終始前方に進む。例えば、気候モデルからの格子の順序は次のようになる。

格子番号	日付	時間	物理的変数の名称
1	97035	000000	U
2	97035	000000	V
3	97035	000000	W
4	97035	000000	T
5	97035	000000	P
6	97035	010000	U
7	97035	010000	V
8	97035	010000	W
9	97035	010000	T
10	97035	010000	P
11	97035	020000	U
12	97035	020000	V
13	97035	020000	W
14	97035	020000	T
15	97035	020000	P

このデータファイルは、3つのタイムステップと5つの物理的変数から成る。物理的変数は、U、V、Wの速度ベクトルと、気温のT、気圧のPの成分である。日付は1997年2月4日で、タイムステップは深夜の1amと2amである。前述した通り、日付はYYDDD、時間はHHMMSSの書式である。

（2）McIDAS 3D格子ファイルの作成

以下のプログラムの例は、立体格子ファイルを構築し、5次元データファイルに対するデータで立体格子を満たす。このプログラムは、例としてsample.Fとし、そのメイクファイルはsample.mである。立体格子ファイルへのデータ読み込みで最も簡単な方法は、サンプルのプログラムを変更することである。呼び出されるサブルーチンは、全てsrcサブディレクトリ内のlibmain.aライブラリ内にある。付録-Aに変換プログラムのサンプルリストを示す。以下、サンプルプログラムを参考に仕様を解説する。

```
C THE MAIN PROGRAM OF YOUR CONVERSION PROGRAM MUST
C BE NAMED SUBROUTINE MAIN0
C
      SUBROUTINE MAIN0
C
C THE NEXT TWO COMMENTS ARE PRINTED BY THE 'help sample' COMMAND
C ? SAMPLE program to convert data to 3D grid files
C ? sample gridf#
C
C DIMENSIONS OF 3D GRID
C NOTE NLATS AND NLONS MUST BOTH BE LESS THAN OR EQUAL TO 150
C NLATS, NLONS AND NHGTS MUST ALL BE AT LEAST 2
      PARAMETER (NLATS=31,NLONS=51,NHGTS=16)
C
C NUMBER OF PHYSICAL VARIABLES AND NUMBER OF TIME STEPS
C NOTE EITHER OR BOTH MAY BE EQUAL TO 1.  THAT IS, Vis5D DOES
C NOT FORCE YOU TO HAVE MULTIPLE VARIABLES OR TIME DYNAMICS.
      PARAMETER (NVARS=5,NTIMES=100)
C
C ARRAY FOR 3D GRID DATA
      REAL*4 G(NLATS, NLONS, NHGTS)
C ARRAYS FOR GRID FILE ID AND GRID DIRECTORY
      INTEGER ID(8), IDIR(64)
C ARRAY FOR VARIABLE NAMES
      CHARACTER*4 CNAME(5)
C
C LATITUDE, LONGITUDE AND HEIGHT BOUNDS FOR SPATIAL GRID
      DATA XLATS/20.0/,XLATN/50.0/
      DATA XLONE/70.0/,XLONW/120.0/
      DATA XHGTB/0.0/,XHGTT/15.0/
C
C STARTING DATE IN YYDDD AND TIME IN HHMMSS
      DATA JDAY/88035/,JTIME/020000/
C TIME STEP IN HHMMSS
      DATA JSTEP/000100/
C
C NAMES OF THE FIVE PHYSICAL VARIABLES
      DATA CNAME/'U   ', 'V   ', 'W   ', 'T   ', 'P   '/
C INITIALIZE GRID DIRECTORY TO ZEROS
      DATA IDIR/64*0/
C
C READ GRID FILE NUMBER FROM COMMAND LINE.  IPP WILL
C CONVERT THE PARAMETER # 1 TO AN INTEGER, WITH A DEFAULT
C VALUE OF 0.
      IGRIDF=IPP(1,0)
C IF ILLEGAL GRID FILE NUMBER, PRINT ERROR MESSAGE AND RETURN
      IF(IGRIDF .LT. 1 .OR. IGRIDF .GT. 9999) THEN
        CALL EDEST('BAD GRID FILE NUMBER ',IGRIDF)
        CALL EDEST('MUST BE BETWEEN 1 AND 9999 ',0)
        RETURN
      ENDIF
C
C CALCULATE GRID INTERVALS
      XLATIN=(XLATN-XLATS)/(NLATS-1)
      XLONIN=(XLONW-XLONE)/(NLONS-1)
      XHGTIN=(XHGTT-XHGTB)/(NHGTS-1)
C
C DATE AND TIME FOR FIRST TIME STEP
C IDAYS CONVERTS YYDDD FORMAT TO DAYS SINCE JAN. 1, 1900
      IDAY=IDAYS(JDAY)
C ISECS CONVERTS HHMMSS FORMAT TO SECONDS SINCE MIDNIGHT
      ISEC=ISECS(JTIME)
C
```

```
 64 C INITIALIZE GRID IDENTIFIER TEXT TO BLANKS
 65 C NOTE LIT CONVERTS A CHARACTER*4 TO AN INTEGER*4
 66       DO 10 I=1,8
 67 10    ID(I)=LIT('    ')
 68 C
 69 C SET UP DIRECTORY ENTRY
 70 C
 71 C DIMENSIONS OF GRID
 72       IDIR(1)=NLATS*NLONS*NHGTS
 73       IDIR(2)=NLATS
 74       IDIR(3)=NLONS
 75       IDIR(4)=NHGTS
 76 C
 77 C LATITUDES AND LONGITUDES IN DEGREES * 10000
 78       IDIR(22)=4
 79       IDIR(23)=NINT(XLATN*10000.)
 80       IDIR(24)=NINT(XLONW*10000.)
 81       IDIR(25)=NINT(XLATIN*10000.0)
 82       IDIR(26)=NINT(XLONIN*10000.0)
 83 C
 84 C HEIGHTS IN METERS
 85       IDIR(31)=1
 86       IDIR(32)=NINT(XHGTT*1000.)
 87       IDIR(33)=NINT(XHGTIN*1000.)
 88 C
 89 C CREATE THE GRID FILE
 90       CALL IGMK3D(IGRIDF, ID, NLATS*NLONS*NHGTS)
 91 C
 92 C LOOP FOR TIME STEPS
 93       DO 200 IT=1,NTIMES
 94 C
 95 C SET DATE AND TIME IN DIRECTORY ENTRY
 96 C IYYDDD CONVERTS DAYS SINCE JAN. 1, 1900 TO OUR YYDDD FORMAT
 97       IDIR(6)=IYYDDD(IDAY)
 98 C IHMS CONVERTS SECONDS SINCE MIDNIGHT TO OUR HHMMSS FORMAT
 99       IDIR(7)=IHMS(ISEC)
100 C
101 C LOOP FOR PHYSICAL VARIABLES
102       DO 190 IV=1,NVARS
103 C
104 C SET VARIABLE NAME IN DIRECTORY ENTRY
105       IDIR(9)=LIT(CNAME(IV))
106 C
107 C ************************************************************
108 C READ YOUR DATA FOR TIME STEP NUMBER IT AND VARIABLE NUMBER IV
109 C INTO THE ARRAY G HERE.
110 C NOTE THAT G(1,1,1) IS THE NORTH WEST BOTTOM CORNER AND
111 C G(NLATS,NLONS,NHGTS) IS THE SOUTH EAST TOP CORNER.
112 C MARK A GRID POINT AS 'MISSING DATA' BY SETTING IT = 1.0E35
113 C ************************************************************
114 C
115 C CALCULATE 3D GRID NUMBER
116       IGRID=IV+NVARS*(IT-1)
117 C WRITE DATA IN G AND DIRECTORY IN IDIR TO 3D GRID
118 C NOTE WE PASS THE NEGATIVE OF THE GRID NUMBER (I.E. -IGRID)
119       CALL IGPT3D(IGRIDF,-IGRID,G,NLATS,NLONS,NHGTS,IDIR,IGNO)
120 C
121 C END OF PHYSICAL VARIABLE LOOP
122 190   CONTINUE
123 C
124 C INCREMENT DATE AND TIME, CONVERT JSTEP FROM HHMMSS TO SECONDS
125       ISEC=ISEC+ISECS(JSTEP)
126 C IF SECONDS CARRY PAST ONE DAY, ADJUST SECONDS AND DAYS
```

```
127       IDAY=IDAY+ISEC/(24*3600)
128       ISEC=MOD(ISEC,24*3600)
129 C
130 C END OF TIME STEP LOOP
131 200   CONTINUE
132 C
133       RETURN
134       END
```

IGMK3D、IGPT3Dのルーチンは、立体格子の構造に対するインターフェイスである。ライン90におけるIGMK3Dへの呼び出しは、立体格子ファイルを作成する。そのパラメータは以下のようになる。

```
1 INTEGER*4    立体格子ファイル番号を表す
2 INTEGER*4    32バイトのファイル名文字列（8配列）
3 INTEGER*4    格子点の最大数
```

立体格子ファイルの作成後、IGPT3Dがライン119上で呼び出され、立体格子データをファイルへ出力する、各タイムステップと物理的変数の結合のためである。そのパラメータは以下のようになる。

```
1 INTEGER*4    書き込みのための立体格子ファイル番号
2 INTEGER*4    書き込みのために立体格子の数を引く。
        これは次の空の格子へ書き込みを表示する、0または、正の値である。
3 REAL*4       配列-書き込みのための格子点の配列
4 INTEGER*4    緯度の番号の格子配列の1次元
5 INTEGER*4    経度の番号の格子配列の2次元
6 INTEGER*4    高度の番号の格子配列の3次元
7 INTEGER*4    立体格子に対するディレクトリ（16配列）
8 INTEGER*4    IGPT3Dによっリターンされる、実際に書き込まれた立体格子の番号
```

動画化シミュレーション・プログラムは、立体格子ファイルの範囲を越えたデータを配置することができる。この場合、反復する変数とタイムステップの格子の順序は、格子ファイルの境界線を越えて連続してとられる。

単一立体格子ファイルは、100,000,000個の格子点（400メガバイト）が最大容量である。データファイルがこの格子点数より大きいならば、新しい立体格子ファイルを作成するためサンプルプログラムを変更すべきである（IG RIDFを増加、IGMK3Dを呼び出すことにより）。それはN番目のタイムステップで行われ、Nタイムステップは、立体格子ファイルをはめ込む。

動画化シミュレーション・プログラムは、格子ディレクトリ内の情報に基づいた正方格子のデータを処理し、そのディレクトリは、サンプルプログラム内のIDIR配列に格納されている。ライン40でIDIRを全て0に初期化する。立体格子の大きさは、IDIR（ライン72から75）の1から4の項目に設定されている。なお、この文書に記載したデータファイルの大きさに制約があることに留意すべきである。

立体格子の日時は、ライン97と99のようにIDIRの6と7の項目内に設定されている。それらは、前述のYYDDDとHHMMSSの書式内に表されている。これらの書式と、日時の計算を簡単にする書式との間の交換のために、libmain.a内に4つの機能がある。IDAYS機能は、ライン60内で1900年1月1日からYYDDD書式を日に変換する。ISECS機能はHHMMSS書式を、ライン125、127、128で秒単位に変換する物理的変数名は4アスキー文字で、ライン105でIDIRの項目9へ詰め込む。libmain.a内のLIT機能は、CHARACTER*4（文字）をINTEGER*4（整数）へ変換する。

格子の空間地点は1000度の位置の緯度と経度と、メートルの高度（標高）の項内に記載されている。格子要素G（1,1,1）は格子の底部の北西の角に、またG（NLATS, NLONS, NHGTS）は頂点の南東の角に位置している。格子の緯度と経度は、ライン78から82でIDIRの21から25の項目に記載されている。格子の高さは、ライン85から87で31から33の項目に記載されている。

NINT機能は、実数（REAL）を最近似値の整数（INTEGER）に変換するためにFORTRANに組み込まれている。緯度、経度と高さの間隔は、単純に継続的な格子点の間の距離である。緯度は北半球では正で、経度は西半球で正になり、当然高さは海抜以上で正になる。

サンプルプログラムの変更における実数作業は、データをライン107から113におけるG配列に取り込むようにコーディングを行う。あるデータに対しては、緯度、経度、高度の座標を模造する。しかし、もしもデータが地理学的でその基準が大きい場合、データの位置を的確に描写することになり、他の幾つかの射影からの緯度、経度、高度においてデータが規則正しく配置されるために再度サンプルを示す必要がある。更にLAT、LON、HGTになるよう命じるパラメータのためにデータ配列を転置したり、G（1,1,1）が底部の北西の角であることを明確にするパラメータにおいてのデータ配列を反転させる必要性がある。模造された座標において、表示の中の正確な‘対称性’を得るためには、データ配列の転置か反転が必要になる。動画化シミュレーション・プログラムは、1.0E30より高い値の配列によって表示された格子点は削除する。

緯度、経度、高度の座標を模造するならば、プログラムの地形図とマッピング表示は意味がなくなると考えられる。データファイルのために軌跡を計算すれば、的確な座標を使用するか、相対的な時間、距離、速度の基準が無矛盾であるために、厳重な注意を払うかのどちらかである。そうでなければ等高線軌跡経路は現実に即さないことになる。

libmain.a内のIPP機能は、ライン45でINTEGER*4のコマンドパラメータの値を戻す。libmain.a内にCPPとDPPという類似の機能があり、コマンドパラメータに対してCHARACTER*12（上記の場合のように変換された）とREAL*8の値を戻す。それらは全て類似の機能のパラメータを持つ。

```
1 INTEGER*4            コマンドパラメータの番号の順序
2(IPP)INTEGER*4        コマンドパラメータの省略値
     または、
2(CPP)CHARACTER*12     コマンドパラメータの省略値
     または、
2(DPP)REAL*8           コマンドパラメータの省略値
```

キーワードに基づいたコマンドパラメータを抽出する手段もある。これはlibmain.a内のIKWP、CKWP、DKWPの機能をもってなされる。それらは'-keyword'形式のキーワードの後に位置しているコマンドパラメータを得る。IKWPはINTEGER*4、CKWPはCHARACTER*12（上記の場合のように変換された）、DKWPはREAL*8を戻す。それらは全て類似の機能のパラメータを持つ。

```
1 CHARACTER*12         コマンドライン内のキーワード列
2 INTEGER*4            キーワード後のコマンドパラメータの番号の順序
3(IKWP) INTEGER*4      コマンドパラメータの省略値
     または、
3(CKWP) CHARACTER*12   コマンドパラメータの省略値
     または、
3(DKWP)REAL*8          コマンドkパラメータの省略値
```

libmain.a内のNKWP機能はパラメータの順序の番号を戻す。その機能のパラメータは下記のとおりである。

1 CHARACTER*12　　文字コマンドライン内のキーワード列。

殆どの機械上のREAL*4書式は、REAL*8書式の部分集合ではなく、それらの中で3番目の機能パラメータと同様、REAL*8としてのDPPとDKWPを明確に宣言する（コマンドパラメータの省略値として）。

FORTRANではなく、C言語でコマンド変換プログラムの作成をする場合は、'sample.c.'のファイルを見なくてはならない。それはstdio内でC言語の構造とルーチンを使用した格子ファイルの読み込みと書き込みをいかに容易にするかの例を含む。

（3）McIDASユーティリティ

データファイルが立体格子ファイルにあるときは、以下のコマンドを使用して、コマンドに関するディレクトリ情報を並べることができる。

```
igg3d  list  I  J  -gr3df  N
```

Nは立体格子ファイルの番号で、IとJはリストにする格子番号の値域を与える。以下のコマンドを使用して、データ値の迅速なアイデアを得ることができる。

```
igg3d  info  I  J  -gr3df  N
```

最小と最大の値、平均、標準偏差、紛失データに対して、立体格子ファイル番号NのIからJの番号の格子に対して印をつける格子点の番号等を並べる。

データファイルには次元の制限があり、プログラムを使用し視覚化することができる。現行では、30の物理的変数と400のタイムステップの最大に制限される。

```
NLATS<2
NLONS<2
NHGTS<2
```

動画化シミュレーション・プログラムの動作性能（処理速度）は以下の式に反比例するであろう。

```
NLATS*NLONS*NHGTS*NTIMES*NVARS
```

64メガバイト・メモリーのシステムは、凡そ25,000,000が限界である。動画化シミュレーション・プログラムはgg3dとigg3dのプログラムを提供し、それはこれらの問題解決ため使用される。gg3dプログラムは立体格子を以下のコマンドを使用し、新しい配列の次元と緯度、経度、高度における新しい範囲を再度サンプルし直す。

```
gg3d  samp  N  I  M  J
gg3d   ave  N  I  M  J
```

NとIは立体格子ファイルと格子の数であり、MとJはその行き先の番号である。'samp'のバージョンは、線形に格子の間に点値を補間することにより格子点値の行き先を計算する。'ave'のバージョンは、多重の格子点値を平均化することによって格子点の行き先を計算しする。gg3dはどのようなキーワードにもよらず、直線複写操作を行う。以下のキーワードでgg3dコマンドを呼び出す。

```
-size NLATS  NLONS  NHGTS
```

寸法と異なる格子のために、格子の行き先に対して格子次元を設置する。

```
-lat   XLATS   XLATN
-lon   XLONE   XLONW
-hgt   XHGTB   XHGTT
```

格子の範囲から異なるように、格子の行き先に対しての緯度、経度、高度に範囲を設定する。-lat、-lon、-hgtのキーワードは実数の引数を得る。

igg3dプログラムは立体格子の複写と削除、その時点でのそれらの補間に対してオプションを提供する。立体格子の順序は以下のコマンドを使用してコピーされる。

```
igg3d  get  N  I  J  M  K
```

Nが立体格子ファイルの発生源の番号にあるときは、IとJは格子発生源の番号の値域で、Mは格子ファイル番号の行き先、Kは格子番号の行き先の開始点である。単一格子は、以下のコマンドを使用して立体格子ファイルの中でコピーされる。

```
igg3d  copy  I  J  -gr3df N
```

Nが立体格子ファイル番号にあるときは、Iは格子発生源の番号で、Jは格子の行き先の番号になる。格子の値域は以下のコマンドで消去される。

```
igg3d  del  I  J  -gr3df  N
```

Nが立体格子ファイル番号にあるときは、IとJの間の格子番号は消去されることになる。igg3dのコマンドは、時間補間に対して2つの異なったオプションを以下のとおり提供する。

```
igg3d  ave  K  I  J  D  T  -gr3df N
```

格子番号Kが立体格子ファイル番号Nにおいて、全てのIとJの格子番号の間での補間によって作成する。格子番号Kは日付D（YYDDD書式で）、時間T（HHMMSS書式で）を割り当てる。格子IとJの相

対的な重みは、線形の時間補間を仮定するこの日付と時間から計算される。格子Kが日付と時間においてIとJの間になければ、igg3dはエラーメッセージを印刷する。igg3dのコマンドは、より複雑な時間補間をも提供する。

```
igg3d  int  I  D  T  -setdel  S  M  -lag  U  V  -gr3df  N
```

日付D（YYDDD書式）、時間T（HHMMSS書式）を割り当てる立体格子ファイル番号Nの次の空のスロットに格子を投入する。この格子は、ファイル番号N内の格子番号I、I+S、I+2S、...I+(M-1)Sにおける全ての格子の順序から補間される。この格子の順序は日付と時間においては昇順になる。igg3dは日付Dと時間Tを括弧でくくる分類から連続する2つの格子の間で、順序と線形の補間を検索する。さらにその補間は座標系の中で行われ、定数速度（U、V）で作動し、UとVが秒毎のメートルにあり、Vは南から北への動きに対して正で、Uは西から東への動きに対して正になる。順序から括弧でくくる2つの格子は、緯度と経度において日付Dと時間Tの位置へけた送りされ、その結果は、空間でけた送りされた2つの格子の間で補間される。更に、順序においての格子が変数名'U'か'V'を持つディレクトリエントリにおいて識別され、速度（U、V）に対応する構成要素が格子の値から減算される。

igg3dの'int'のオプションは複雑に見えるが、タイムステップの新しい順序への5次元データファイルを再補間するためにスクリプトを書き込む、及び再補間時にぶれを除くようにするためには、Mをタイムステップの番号と、Sを5次元データファイルの物理的変数の番号と等しく設置する。Iのパラメータは、補間される変数の最初のタイムステップにおける格子番号と等しく設定する。このigg3dのオプションは、データファイルの格子ファイルの終点にある新しい格子を投入するが、それを別の格子に移動させるために'igg3d get'を使用することができる。以下のコマンドを使用することができる。

```
igu3d  make  N  M
```

立体格子ファイル番号Nを作成するために各々M地点以上の立体格子にさせる。立体格子ファイルの名称は以下の形式を持つ。

```
GR3Dnnnn
```

nnnnは格子ファイル番号の4つの10進法の数字で、4つの数字を成す必要性があれば、先行の0をもって埋め込まれる。

５．５　ユーティリティ・プログラム

動画化シミュレーション・プログラムには、いくつかのユーティリティ・プログラムが備わっている。それらは以下のとおりである。

v5dinfo

使用方法：v5dinfo file

処理機能：V5dinfoは、立体格子のサイズ、タイムステップの番号、変数の名称、等として与えられたv5dファイルに関する情報を印刷する。このプログラムはcomp5dファイル上でも作動する。

V5dstats

使用方法：V5dstats file

処理機能：V5dstatsは、v5dファイル内の格子データに関する簡単な統計の情報を印刷する。

V5dedit

使用方法：V5dedit file.v5d

処理機能：V5deditは、写増投射、縦の座標系、名前付きファイル内の変数名の見出し情報を変更することができる対話式、メニュー式のプログラムである。このプログラムはcomp5dファイルとは同時に動作しない。

V5dappend

使用方法：V5dappend［-var］［...］file.v5d［...］target.v5d

処理機能：V5dappendは、1つの大きなファイルを作るために多くのv5dファイルを追加することができる。天候モデル等の各タイムステップ毎に分割したv5dファイルを生成するならば、有効であろう。

- 出力ファイルから省略する変数の任意リスト。例として、変数UとTHETAを省略したければ、UとTHETAの引数を使用する。
- 目標のファイルに追加するv5dファイルのリスト。
- 作成（出口でなければ）、または追加（目標ファイルが既に存在していれば）する目標のv5dファイルの名称。

格子（行、列、高さ）の次元は、それらを一緒に追加する各ファイル内で同一でなくてはならない。射影と縦の座標系情報は、最初に入力したファイルから取り出され、残りのファイルは省略される。

gr3d_to_v5d

使用方法：gr3d_to_v5d N M file.v5d C

処理機能：gr3d_to_v5dは、(a)McIDAS GR3Dファイル(s)をv5dファイルに変換する。Nは、最初の

格子ファイルの名称を表す番号で、Mは変換する格子ファイルの番号、ファイルv5dは作成するファイルの名称、Cは、圧縮に（省略値は1）使用する各格子点が何バイトになるか表示すると1、2、または、4になるということを各々表している。

例：N=20でM=4ならば、ファイルGR3D0020、GR3D0021、GR3D0022、GR3 D0023は、v5dの名前付きのファイルに変換して読み込まれる。

igg3d

使用方法：igg3d...

処理機能：igg3dはMcIDAS GR3Dファイル上での操作の多様性を実行している。

igu3d

使用方法：igu3d...

処理機能：igu3dはMcIDAS GR3Dファイル上での操作の多様性を実行するのに有益である。

gg3d

使用方法：gg3d...

処理機能：gg3dはMcIDAS GR3Dファイルの再サンプル取りには有益である。

listfonts

使用方法：listfonts

処理機能：SGIシステム上でのみ使用されるlistfontsは、動画化シミュレーション・プログラム3Dウィンドウで使用可能なIRIS GLフォントを列挙する。フォントの列挙の後に、フォントの任意選択をすることにより、vis5d内でそれを使用することができる。SGI以外のシステム、または、OpenGLを使用するシステムに対しては、フォント選択にはxls fonts、または、xfontselプログラムを使用する。

comp5d

使用方法：comp5d N M filename

処理機能：comp5dは、動画化シミュレーション・プログラムを使用し、McIDAS GR3Dファイルをcomp5dの書式に変換する。Nは最初の立体格子ファイルの番号で、Mはデータファイル内の格子ファイルの番号である。Mパラメータはデータファイルを、複数の格子ファイルにまたがり、データファイル内で立体格子の総数で混乱させないようにする。

ファイル名は圧縮された格子ファイルの名称である。希望するどんな名称でも選択することはできるが、comp5dは上記の場合の全ての文字に変換することを留意しておかなくてはならない。

データファイルが速度ベクトルの構成要素を持つならば、速度のキーワードを速度の構成要素の部分集合選択するか、または、圧縮されたファイルに対して'SPD'という名前付きの水平の速度を計算するために速度のキーワードを使用することができる。緯度、経

度、速度ベクトルの縦の構成要素は個々に'U'、'V'、'W'と名付けられなくてはならない。速度キーワードを使用すれば、速度に関連した変数（例：U、V、W、&SPD）のみが速度の後に並べられ、圧縮されたファイル内に吸収される。例えば、U、V、Wの構成要素を持つ立体格子ファイルから圧縮されたファイル内でSPDとWを吸収させるには、以下のコマンドを使用する。

```
comp5d   N   M   F   -wind   SPD   W
```

help

使用方法：help utilityname

処理機能：helpコマンドは、igg3d、gg3d、comp5dユーティリティと名付けられたパラメータ書式を表示する。例：help igg3d

maketopo.c

utilディレクトリ内のこのプログラムは、新地形図ファイル（*TOPO）を生成するためのテンプレートプログラムである。命令等はファイルの最初にある情報参照。maketopoをコンパイルするには、maketopo.m.と名前が付いたメイクファイルを参照。

makemap.c

utilディレクトリ内のこのプログラムは、新McIDASのマッピングアウトライン（OUTL*）ファイルを生成するためのテンプレートプログラムである。命令等はファイルの最初にある情報参照。makemapをコンパイルするにはmaketopo.m.と名前が付いたメイクファイルを参照。多くのラインセグメントのロットを持つマッピングを作成するならば、それは紛失した特別異常なラインセグメントで表示される。src/ globals.h内でMAXMAPVERTとMAXMAPSEGを変更させることによって、これを固定し、動画化シミュレーション・プログラムを再度メイクする。

newmap.c

utilディレクトリ内のこのプログラムとmapfunc.fは、新しいマッピングアウトラインファイルを作成する。マッピングアウトラインファイルの頂点を変形するのに使用される。

より詳細な情報は、newmap.cとnewmap.mのファイルを参照。

５．６　動画化シミュレーション・プログラムの機能

動画化シミュレーション・プログラム使用法について記述する。

５．６．１　操作開始

動画化シミュレーション・プログラムは、以下のコマンドで実行を開始する。

（１）動画化シミュレーション・プログラム file.v5d [options]

[options] は以下のとおりである。

-alpha

スクリーン制御方式を透過伝送機構でなくアルファ・ブレンドとする。

-area N

[SGI only] は読み込むためのMcIDAS領域ファイルの順序の一番目を指定し、それから立体ボックスの内部を表示する。

-box x y z

縦横比か立体ボックスの比例を指定させる。省略値は、2　2　1である。

-barbs

速度ベクトルの地点で速度のbarbsを使用する。

-date

時計の'yyddd'の地点での'dd month yy'を使用する。

-font xfontname

-font glfontname height

立体ウィンドウ内の時計をテキストラベルに対して使用するフォントを設定する。また、'vis5d'とだけタイプし、オプションを調べることによってシステム上で使用されるフォントのオプションの形を確定することができる。最初の形はXウィンドウシステムのフォントの名称である。システム上のXのフォントのリストを見るためxlsfontsのコマンドを使用する。2番目の形はIRIS GLのフォントの名称とサイズ（72=1インチ）である。システム上のGLのフォントのリストを見るため動画化シミュレーション・プログラムで格納されているリストフォントのコマンドを使用する。

例1：vis5d LAMPS.v5d-font fg-30
例2：vis5d LAMPS.v5d-font Helvetica30

-full

境界線のない、全画面サイズのウィンドウとして立体ウィンドウを開く。

-funcpath pathname

ユーザーのFORTRAN機能を検索するためのディレクトリを設定する。

例　vis5d LAMPS.v5d-geometry 640x480-10+10

-hirestopo

高解像度の地形図を表示する。これは高速図形ハードウェアを持ったシステム上でのみ推奨される。

-legend position size

カラーキャプションの位置とサイズを設定する。位置の値は、各々1（底部、省略値）、2（頂点）、3（左）、4（右）である。サイズはキャプションバーの高さで、10から1000の間（省略値=128）にある。

-log [a] [b]

線形の代わりに、対数軸の高さを表示する。オプションの変数であるaとbは、高度／気圧の方程式内の基準で、パラメータ因子でもある。省略値は、各々1012.5と-7.2である。

-map file

OUTLSUPWの省略値と違うマッピングファイルを使用する。

例：vis5d LAMPS.v5d-map OUTLUSAL

-mbs n

32メガバイトのメモリサイズを仮定したシステムを置き換える。

-path pathname

電流の代わりにマッピングと地形のファイルに対する異なった経路を使用する。

例：vis5d LAMPS.v5d -path/usr3/data

-projection p

射影の表示を設置し、省略値はその自然の投射（データファイルから獲得される）の中でデータを表示させる。pは次のどちらかである。

```
cylindrical   円柱の地球上のデータを表示
spherical     球状の地球上のデータを表示
```

最初の3文字が重要である。

例：vis5d LALMPS.v5d -projection spherical

-quickstart

動画化シミュレーション・プログラムの開始時には、たとえファイル全部がメモリ内で適合したとしても、どの格子もロードしてはならない。格子は必要に応じて読み込まれる。このオプションはNFSを通してファイルを読み込む時に役立つ。

-rate ms

省略値のアニメーションの率を変える。msは枠の間においてミリ秒の最小限の遅れである。省略値は100msである。

-scriipt script.tcl

自動的に実行するvis5d/Tclスクリプトを設定する。

-sequence filename

地形図を超えたテクスチャマップの印象の順序を持つファイルを設定する。なお、全てのシステム上で使用可能ではない。

-texture rgbfile

SGI .rgbのファイルを、地形図上のテクスチャマップを設定する。なお、全てのシステム上で使用可能ではない。

-topo file

EARTH.TOPOの省略値でなく他の地形図ファイルを使用する。

-trajvars uvar vvar [wvar]

どの変数が軌跡の追跡に使用されるかを設定する。省略値はU、V、Wである。

例：vis5d LAMPS.v5d-trajvars U2 V2 W2

-vertical v

縦の座標系、データファイルから得られる省略値を設定する。vは以下の内の1つである。

包括的	-汎用体内で線形に、均等に行送りされたレベル
均等	-km内で線形に、均等に行送りされたレベル
不均等	-km内で線形に、不均等に行送りされたレベル

vの最初の3文字だけが重要かつ必要とされる。追加の変数に対しては指示がある。

例：vis5d LAMPS.v5d-vertical nonequal

-wdpy xdisplay

異なったXディスプレイの上にウィジェットを置く。スライドとビデオの作成の組合せにおいて役立つ。

例：vis5d LAMPS.v5d-full -wdpy pluto:0

-wide w

画素（省略値は1.0）内のラインセグメントの幅を設定する。ビデオ作成に役立つ。

例：vis5d LAMPS.v5d-wide 3.0

-wind2 uvar vvar [wvar]

Hwind2、Vwind2、Strm2ベクトルスライスを描写する時に使用する、U、Vと（任意選択で）Wの速度構成要素の変数の補助的なセット名を指定する。同時に視覚化したいと思う速度ベクトルの構成要素の2つのセットに役立つ。

例：vis5d MYDATA -wind2 U2 V2 W2

変数なしで動画化シミュレーション・プログラムを開始させた場合、全てのコマンドライン選択とキーボード機能のリストを得ることになる。他方動画化シミュレーション・プログラムはデータファイルの読み込みによって開始する。

5．6．2　コントロールパネル

動画化シミュレーション・プログラムを起動すると、2つのウィンドウが表示される。立体ウィンドウはスクリーンの右側に、コントロールパネルは左側に位置している。その左上は、現在時刻を表示するアナログとディジタルの組合せの時計である。コントロールパネルには幾つかのボタンが集まっている。

[ANIMATE]	[STEP]	NEW VAR	EXIT
[TEXTURE]	TOP	SOUTH	WEST
[TOPO]	[MAP]	BOX	CLOCK
SAVE	RESTORE	GRID #'s	CONT #'s
[ANIM-REC]	REVERSE	[SAVE PIC]	[PERSPEC]
SCRIPT	INTERP	UVW VARS	

これらのボタンは動画化シミュレーション・プログラムの主要機能の制御に使用される。ボタンの幾つかは、大括弧 [] に囲まれており、動画化シミュレーション・プログラムが開始すると空白になって表示される。これは現在のデータファイルに対してボタンが当てはまらない時に起こる。それはボタンがコマンドライン選択に矛盾するか、または、その機能がハードウェアに対して使用不可能かのどちらかである。ラジオボタンの次の集合は、マウスを用いて立体ウィンドウを制御するものである。

正常　：正常なマウスモードは、立体ウィンドウ内の図形を循環、急上昇、パンすることに使用される。
軌跡　：速度の軌跡の作成と描写のために使用される。
スライス：水平な縦のスライスの再位置付けのために使用される。
ラベル　：立体ウィンドウ内でテキストラベルを作成、編集するために使用される。

証明　：立体格子を通して立体カーソルを作動させることにより、個別の格子の価値を検証するため使用される。

これらのモードは相互に排他的であり、その時点で唯一つが選択される。これらのボタンの右側は、マウスボタンキャプションである。選択されたモードに対するボタンの表示を次に示す。

Hwind1　　Vwind1　　HStrm　　Hwind2　　Vwind2　　VStrm

速度ベクトルのスライス（HwindかVwind）は、速度の方向内の地点である小さな矢印を描くことによって速度の値を描写する。各回線区分の長さはその振幅を表示する。回線区分の末尾は、全て立体ボックスを通して水平、または縦の面で固定されている。水平の速度の流線型のスライス（HS trm）は、水平面に流線型を描くことによって速度の値を描写する。縦の速度の流線型のスライス（VStrm）は縦面に流線型を描くことによって速度の値を描写する。スライス面の位置は、"Slice"モードになっているマウスで変更することができる。

コントロールパネルウィンドウの底の部分は、ボタンの平面マトリックスを包含する。各行はデータファイル内の物理的変数に対応し、各列は図的表現の型の１つに対応する。正確な行と列を選択することによって、立体の同一面、水平の輪郭のスライス、縦の輪郭のスライス、水平の色彩スライス、縦の色彩スライス、或いはボリュームレンダリングというような、どんな変数をも見ることができる。このボタンのマトリックスは、ウィンドウ内で適合するより多くのボタンの行があるならば、画面移動することができる。ボタンマトリックスの画面移動をするには、マウスを使ってスクロールバーをドラッグしても、或いはキーボード上の上下矢印のキーを押してもできる。

どの図形の表示も、左マウスボタンをクリックすることによって制御される。各々の図形の型は、スイッチを入れると現われる小さなポップアップ制御ウィンドウを持つ。制御ウィンドウは各図形の型によって異なる。表示を切り換えず、図形の制御ウィンドウを持ち出すには、中央のマウスボタンを使用する。図形が表示されている時は、その表示中の異なった変数を識別しやすくするボタンと同じ色になる。図形の色を変えるには、右マウスボタンをクリックすると４つのスライダとともに小さなウィンドウが現われる。赤、緑、青の基準を変えることにより、どんな色をも作ることができる。

コントロールパネル・ウィンドウに他のウィンドウが覆い被さった場合、マウスポインタが立体ウィンドウにある間に、'F1"キーを押して前面に持っていくことができる。これは特に'-full'オプションの使用時に役立つ。

５．６．３　基本操作

コントロールパネルの中でボタンの一番上の集合は、動画化シミュレーション・プログラムの主要機能を操作する。

ANIMATE

このトグルボタンは、動画を出すか、または閉じる。前面の動画には、左か中央のマウスボタンを使用し裏面の動画には右のボタンを使用する。１つのタイムステップでデータファイルが見える時は現われない。動画を遅く、または早くする時は、マウスのカーソルが立体ビューウィンドウにある間

に、SとFのキーを叩く。

STEP

このボタンはどのマウスボタンを押すかによって3通りの使用法が可能である。

左ボタン　　：1つのタイムステップより以前の段階
中央のボタン　：最初のタイムステップへ行く
右ボタン　　：タイムステップの後方へデータファイルが1つのタイムステップと共に見える時は、このボタンは現われない。

NEW VAR

物理的変数を複写するか、または、外部の解析機能を呼び出すために使用される。

EXIT

プログラムから出ること。EXITの確認を問うためウィンドウが現われる。

TEXTURE

テクスチャマップがロードされていたら、その表示を出したり消したりして切り換える。

TOP

どのマウスボタンを押すかによる。

左または中央：立体ウィンドウを省略値の最初の視点へリセットする。
右　　：立体ウィンドウを最後の視点へ設定する。

SOUTH

どのマウスボタンを押すかによる。

左または中央：立体ウィンドウを南の視点へ設定する。
右　　：立体ウィンドウを北の視点へ設定する。

WEST

どのマウスボタンを押すかによる。

左または中央：立体ウィンドウを西の視点へ設定する。
右　　：立体ウィンドウを東の視点へ設定する。

TOPO

地形図の表示を切り換える。地形図ファイルが見つからない時はこのボタンは現われない。地形図の色を編集するには、右マウスボタンでTOPOをクリックする。

MAP

マッピングラインの表示を切り換える。マッピングファイルが見つからない時はこのボタンは現われない。マッピングラインの色を編集するには、右マウスボタンでMAPをクリックする。

BOX

立体ボックスの表示を切り換える。

CLOCK

時計の表示を切り換える。

SAVE

最新の図形と色を保管する。同一表面、スライス、速度の軌跡、色の各変数を準備後、それらを保管し復元するために用いる。ファイル書式は、Tclスクリプトである。

RESTORE

SAVEボタンで保管された情報を復元する。

GRID #s

緯度、経度、キロメートル内のデータファイルの境は、通常ボックスの端に沿って表示される。格子座標内の番号を表示するには、このボタンを使用する。

CONT #s

輪郭線スライス上に描かれた番号は、このボタンでスイッチオン・オフを切り換えることができる。

[ANIM-REC]

このボタンは動画のように作動するが、立体レンダリングが遅いとシステム上の動画を早くする。各タイムステップがレンダリングされたら、画像がメモリ内に保管される。動画のループが画像を反復する時は、メモリから動画が早くなる立体ウィンドウへ複写される。

REVERSE

立体ボックスと時計は、黒い背景では通常白く描かれる。このオプションはそれを反転させ、白い背景に黒いボックスと時計を描く。

SAVE PIC

立体ウィンドウ内画像をファイルへ保管する時に使用する。どのようなシステムを使用しているかによって、多くの異なったピクチャファイル書式を支援する。SGIシステム上で、IRIX CDROMから導入された'tops'、'frombin'、'togif'等のプログラムが確実に得られなければならない。SGIシステム上でOpenGLを使用する時は、'fromxwd'プログラムも必要である。

PERSPEC

透視と直交の見える投射の間で切り換える。

SCRIPT

動画化シミュレーション・プログラムTclスクリプトの実行に使用される。このボタンをクリックすると、Tclスクリプトの実行のため選択した所にファイルリクエストが現われる。

INTERP

シェルウィンドウ内でTclコマンドを入力すると動画化シミュレーション・プログラムの対話式の解説を始める。INTERPが稼働中は、動画化シミュレーション・プログラムは中断される。

UVW VARS

軌跡と速度のスライスを計算するのに使用する変数の名称を指定することができるウィンドウを開く。

LEGENDS

立体ウィンドウ内のカラーバーキャプションの表示を切り変える。

５．６．４　視点操作

'Normal'（通常）のマウスモードでは、マウスは立体ウィンドウ内のデータを見るために使用される。左のマウスボタンを押し、カーソルが立体ウィンドウ内にある間にマウスを動かすと、立体画像が回転する。なお、ボックス回転の自由の３つの段階の２つを制御できる。左マウスボタンを解放し押し直すことによって、ボックス上の"grip"（握り）を変えることができる。一連のマウスの動きの中で、左ボタンを解放し押し直すといった実行に伴い、その動きを通してボックスの制御を覚えていくことになる。

中央のボタンは、マウスをどう動かすかによって全く異なる２つのことを制御する。中央のボタンを押し下げて遠ざけるように滑らせるとボックスは拡大する。マウスを手前に向かって滑らせると、ボックスを縮小させることになる。中央のボタンを押し下げてマウスを右に滑らせると、目に見えない面を（例えばクリッピングされた面）ボックスの内容の視点の切り取りを生じさせるボックスの中へと移動させる。マウスを左に滑らせると、クリッピングされた面を手前へ持って行き、最後は全くボックスの外へ出してしまう。

右マウスボタンを押すとウィンドウ内のボックスが変換される。これはボックスの中央にない物を掘り下げて検討（拡大？）したい時に使用する。ボックス回転に対する回転の中央は、ボックスの中央というよりも、むしろ画面の中央に位置している。

同一の表面（立体の輪郭を持つ表面）は、同一値を特定することにより境になる立体の巻を表す。同一表面は指定された同一高度、内部の容量は同一値より多い（または、少ない）値を含む。外部の容量は同一値より少ない（または、多い）値を含む。

コントロールパネルのボタンマトリックス内のボタンの最初の列は、同一表面の制御を行う。左マウスボタンでこれらのボタンの１つをクリックすることにより、現われるスライダとOKボタンをともにポップアップウィンドウを出す。スライダ上の同一値を選択し、全てのタイムステップに対しての同一表面を発生させるためOKボタンをクリックする。

ANIMATE（動画）を切り換えると、同一レベルである軌跡の表面をタイムダイナミクスに見ることができる。表面が動画と共に非同期的に発生することから、時計針の回転のような全てのタイムステップに対する表面を見ることは出来ない。新しい表面は連続的な時計の周期上に現われる。

中央のマウスボタンで同一表面ボタンをクリックすると、表面をオンまたは、オフに切り換えずに

ポップアップウィンドウを呼び出す。

（1） 同一表面の色

同一表面は一色で全体的か、または他の物理的変数の値による色のどちらかで描かれる。

同一表面の色を変えるには、右マウスボタンで適切な同一表面ボタンをクリックする。ウィンドウは、変数名の列（最初のボタンは"monocolor"と表示されている）、赤、緑、青、透明の表示がしてある4つのスライダと共に現われる。

省略値によっては単色が使用される。同一表面の色を変えるには、赤、緑、青のスライダを動かすだけである。

"monocolor"でなく他のボタンをクリックした場合、動画化シミュレーション・プログラムが他の物理的変数による同一表面を描くことに気づく。赤、緑、青のスライダは、カラーテーブルの編集プログラムで置き換えられる。マウスを使って新しい曲線を描くか、または、キーボードの上下左右のカーソルキーによって、カラーテーブル（マッピングデータ値を色へ）を変えることができる。

例えば、LAMPS.v5dのデータファイルを見るとしよう。速度の同一表面の速度を40メートル／秒に設定する。その同一表面は青である。右マウスボタンを使ってSPD同一表面ボタン上にクリックしてするとカラーウィンドウが現われる。ウィンドウ内のTボタンをクリックすると、気温にともなって同一表面が色づく。マウスボタンか、またはカーソルキーを押すことにより、カラーテーブル内の赤、緑、青の曲線を"drawing"（描く）ことによって、気温の値からマッピングを色に変更することができる。

5．6．5 スライス

スライスは、立体ボックス内でデータの平面交差する場所を見せることができる。これらのスライスは水平、または縦に配列されていて、軌跡の線、色づけされたスライス、速度ベクトル、速度の流線の何れかを描くことになる。

コントロールパネル上のボタンの最後の集合はボタン、スライスを制御する5番目の列に渡る秒のマトリックスである。水平の軌跡スライス、縦の軌跡スライス、水平の色付けされたスライス、縦の色付けされたスライス等、各々に対するボタンの列がある。データファイルがU、V、Wの変数を含んでいたら、速度ベクトルのスライスボタンの行が出てくることになる。水平の速度ベクトルと縦の速度ベクトルの各々に対する２つのボタンがある。

スライス上で起動／回転するには、左マウスボタンで適切なボタンをクリックする。スライスに対する初期の位置はボックスの中央である。緯度、経度、高度の項の中の正確なスライスは、各スライスの一角の近くの細かい数字の位によって与えられる。図形座標の代わりに格子座標としての番号を印刷するには、コントロールパネル上のボタンである"GRID#s"を切り換える。

スライスの位置は、マウスを使って対話しながら（双方向に）変換可能である。それには、最初にSLICEラジオボタンを選択することによってSULICEモードにすべきである。スライスを移動し、マウスで簡単にスライスの角を指すには、右マウスボタンを押し、新しい位置にドラッグする。縦のスライスは更に頂点か底部の角の中央を“つかむ（グラブ）”か、それをドラッグすることによっておこる垂直の動作へと移動する。スライスは動画モードにある時に動かされるが、新しいスライスが非

同期的に計算されるので、跳ね上がりが発生する。

（1）輪郭線スライス

水平、または、縦の輪郭線スライスが表示されると（ボタン列2と3）、小さな制御ウィンドウが現われる。このポップアップウィンドウ内に輪郭線間で使用するための時間間隔を入れることができる。その間隔を変更する新しい番号をタイプインするだけである。

負の間隔を指定すると正の値が実線で描かれる一方、負の値を持つ全ての輪郭線が破線で描かれる。時間間隔の値の後で、(a, b)の値を選択的に指定することができる。その値は、描かれているaとbの間の輪郭値のみを生じさせる。例えば、以下の数値を入力する。

```
-10 (-30, 20)
```

これは、破線で描かれている負の線を持つ10の時間間隔において、-30と20の間の値に対する輪郭線において生じる。コントロールパネル上の"CONT#s"ボタンは、スライス内部の輪郭番号の表示を切り換える。

（2） 色付けされたスライス

水平または、縦の色付けされたスライス（ボタン列4と5）が表示されると、カラーテーブルウィンドウが現われる。このポップアップウィンドウ内で、データ値からのマッピングを色に変換することができる。コントロールパネルボタンのLEGENDSを選択すると、立体ビューウィンドウ内でカラーテーブルも表示される。

ウィンドウはデータ値の範囲全体に渡って、赤、緑、青の図形を表す。赤、緑、または青の機能を変更するには、左、中央、または右のマウスボタンを押し、新しい機能を得るためにマウスでドラッグする。省略値によって、横の行のデータ値が青に、高データ値が赤へとマッピングされる。

マウスの代わりにキーボードのカーソル（矢印）を使って、その形状と省略値の機能の曲線を変更することができる。曲線を左か右に動かすには、左／右キーを、曲線の形状を変えるには、上／下のキーを押す。

データ値の機能としてのスライスの透過伝送機構の変更を行う時には、マウスまたは上／下キーを使用する間、シフトキーを押したままにしておく。

カラーテーブルウィンドウに対する、その他のキーボード制御の番号は次のとおりである。

r	赤、緑、青の値をリセットする
R	透過伝送機構の値をリセットする
c	色を画面の外のクリップボードへコピーする
p	色を画面の外のクリップボードから貼り付ける
s	色の値をファイルへ保管し、シェルウィンドウ内にファイル名を入力
l	ファイルから色の値をロードし、シェルウィンドウ内にファイル名を入力

（3）速度ベクトルスライス

速度ベクトルスライスは、HWIND-1、VWIND-1、HWIND-2、VWIND-2とラベル表示されたコントロールパネルの中央の近くのボタンで表示されている。これらの図形に対するポップアップウィン

ドウは、速度ベクトルの密度と縮尺を制御する２つの分野を持つ。縮尺パラメータは、描かれているベクトルの長さを掛け算するため使用される。全てのベクトルの長さを倍にする場合は2.0を入力する。長さを半分にする場合は0.5を入力する。密度パラメータは、どの位の速度ベクトルが表示されているかを制御する。この値は0と1の間である。ベクトルの数を1/2にするには0.5を、1/4ならば0.25を入力する。その両方のパラメータに対する省略値の値は1.0である。

（４）速度流線スライス

速度流線スライスは、ラインセグメントを連結する速度の経路を示す。ポップアップ制御ウィンドウは、流線（縮尺パラメータは使用されていないことに留意）の密度を制御する。密度パラメータは、どの位の数の流線が表示されているかを制御する。この値は0.5と2.0の間である。流線の数を1/2にするには0.5を、2倍ならば2.0を入力する。省略値の密度は1.0である。

（５）スライスの色

スライスの制御ボタンの色は、スライスそれ自体の色と一致する。スライスの色を変えるには、右マウスボタンでスライスのボタンをクリックする。赤、緑、青のスライダを持ったウィンドウから色を変更するためにスライダを動かす。

５．６．６　ボリュームレンダリング

ボリュームレンダリングは半透明の色のもやのような立体視界を表示するための技術である。とはいえ、ある物理的変数のボリュームレンダリングは見られず、他の物が正確な色のマッピングで効果的に表示されている。

コントロールパネル上のボタンの縦の6列はボリュームボタンである。一回に１つしか表示されない。ボリュームレンダリングが始動すると、カラーテーブルとともにポップアップウィンドウが現われる。このカラーテーブルは、上述の色付けされたスライスで描かれたのと全く同じ方法で使用されている。マウスかキーボードを使用し、データ値を色付き、透明にマッピングする機能を変えることができる。再度、シフトキーを押さえている間、マウスで曲線を描くか、上／下キーを押すかをすれば、透過伝送機構を変えられる。ボリュームレンダリングは以下のことを行う。

＊立体ボックスの軸が、視界の方向に対してほとんど並行に確定するため、現行で見える変形を調べる。
＊データ値を色付き、不透明にマッピングする軸に対してperpindicularである、多くの色付けされたスライスを作る。
＊裏にある色付けされたスライスを前面の順番にする。頂点のアルファ値は、各スライスの間及びその内部での移行が順調に行われるように補間、調整される。

５．６．７　速度の軌跡

速度の軌跡は、速度のトンネル内で立体ボリュームの煙の線の痕跡を通して空気の動きを追跡する。軌跡モードにするには、コントロールパネル上のTRAJECTORYラジオボタンを選択する。ポップアップウィンドウが画面のボタンの近くに現われ、立体カーソルがボックス内部に現われる。この立体カーソルは、新しい速度軌跡がどこに作られるべきか指定するために使われる。主コントロールパネル上のSTEPボタンも重要であり、それは軌跡を生み出す所のタイムステップを選択するために使用

されるからである。

速度の軌跡は一セットで処理され、８つのセットがある。各セットは、セット1、セット2、...セット8とラベル表示されたボタンを持つ軌跡ウィンドウ内で表現される。各セットは個別に表示され、色付け、或いは消去される。新しい軌跡を作成する時、地点、時間等に対応するセットにグループ分けするとよい。

軌跡作成の最初の段階は、立体カーソルで位置を選択する。右マウスボタンを使って立体ボックス内部の立体カーソルの周辺をドラッグする。立体カーソルは軌跡に並行な面の中で平面に動く。左マウスボタンでビューボックスを回転させるのと、右マウスボタンでカーソルを設定する動作を交互に行うことによって、立体カーソルはビューボックス内部のどこにでも設定可能になる。

次に、コントロールパネル上のSTEPボタンでタイムステップを選択しなければならない。軌跡が作成される時、それは現在のタイムステップから最後のタイムステップへと順に辿って行き、時間内は最初のタイムステップに向かって後ろ向きに遡るようになる。

最終的に、現在のカーソル位置とタイムステップにおいて軌跡を作成するには、立体ウィンドウ内部に向かう時の中央のマウスボタンを押す。ラインセグメントとして軌跡が現われる。ANIMATEボタンをONにすることにより、軌跡がどのように時間と場所にまたがり移動するかを観察することができる。概して立体カーソルの位置決めと軌跡のセットの作成のため、中央のマウスボタンをクリックする過程を繰り返す。

ANIMATEボタンがONの時に軌跡ができることによって、軌跡は唯一つではなく全てのタイムステップごとに作成される。これは全ての空気の包みの経路を表し、その包みは場所の中の単一地点を通過する。以下は様々な軌跡の機能の概要である。

＊立体カーソルの配置には、左マウスボタンでビューボックスを回転させるのと、右マウスボタンで立体カーソルをドラッグするのを取り混ぜて行うこと。
＊タイムステップの選択にSTEPボタンかANIMATEオプションを使用する。
＊現在のカーソルの位置とタイムステップにおける軌跡を作成するには、中央のマウスボタンを押す。
＊軌跡の表示の入切を切り換えるには、左マウスボタンでセットボタンをクリックする。
＊中央のマウスボタンでセットボタンをクリックすることにより、現在の軌跡のセットを選択する。
＊軌跡ウィンドウ内の'Delete Set'ボタンで軌跡セットは消去される。
＊軌跡ウィンドウ内の'Delete Last'ボタンをクリックすることによって作成された最後の軌跡を消去することができる。

速度の軌跡はラインセグメントかリボンの２つの方法で描かれる。軌跡ウィンドウ内のRIBONボタンをクリックすることによってリボンを選択することができる。RIBBONボタンの切り換えは、新しい軌跡が表示されるかを制御するのみであり、既に作成された軌跡には効果がない。

軌跡ウィンドウは、更にSTEPとLENGTHとラベル表示された２つのタイプイン規格を持つ。STEP値は、軌跡を辿るアルゴリズム内で使用されている工程のサイズを制御するために使われる。LENGTH値は軌跡の長さを制御するために使われる。1.0は各々に対する省略値の値であり、それらは各々乗数として作動する。もしも省略値のサイズの1/2の過程で統合するために等高線にする必要があれば、0.5の工程値を入力する。軌跡を省略値の長さと同じく2倍にしたければ、2.0の長さの値を入力する。

軌跡の色は同一表面に対するのと同じ方法で制御される。それは軌跡のセットが、他の物理的変数によって単一色か、または色付きのどちらかになる。そのカラーウィンドウを提示するために右マウスボタンで軌跡セットボタンをクリックする。

色でマッピングされた軌跡が見える時、軌跡の色が時間依存型であることは知られている。現在のタイムステップに対する他の変数に対する値によって、軌跡の頭部のみが色付けされる。軌跡の尾部がその他の変数の色によって色付けされる時、頭部はその位置にある。

５．６．８　速度の変数

省略値によって、速度の軌跡と速度ベクトルの一番目のセットは、U、V、Wと名付けられた変数によって計算され、その間に二番目の速度ベクトルがU2、V2、W2と名付けられた変数によって計算される。他の変数は、コントロールパネル上の"UVW VARS"ボタンを通して指定される。このボタンをクリックすると、ポップアップウィンドウが現われ、そこで軌跡、及び一番目と二番目の速度ベクトルのセットの計算のために使用される変数の名称を指定することができる。指定は単に新しい変数の名前をタイプするだけである。有効な名称を入力しないと、動画化シミュレーション・プログラムは選択した図形を計算しない。

新しい速度の構成要素の変数名を入力した後は、新しい値を使用してAPPLYをクリックし、かつウィンドウが見えるようにしておかなくてはならない。新しい値を使用しウィンドウを閉じるには、OKをクリックする。変換を廃止しウィンドウを閉じるには、CANCELをクリックする。更に動画化シミュレーション・プログラムの開始時に、コマンドライン上の速度構成要素の変数も設定することができる。

５．６．９　テキストラベル

テキストラベルは、立体ビューウィンドウ内でイメージの注釈を付けるのに使用される。これは図形表示の作成用に使用される。題目、名称、日付を追加することができ、日付、データの意味がウィンドウ内に見える文書の特殊な機能を強調する。テキストラベルを入力するには、モードをコントロールパネル上のLABELラジオボタンで選択する。

テキストラベルを作成するには、立体ウィンドウ内のどこかにマウスのポインタを位置付け、左マウスボタンを押す。縦のバーカーソルがその位置に現われた時、テキストをタイプすることができる。<Backspace>キーは間違いを訂正するために使用される。終了後、<Return>を押す。

テキストラベルを新しい位置に移動させるには、マウスでそれを指し、中央のマウスボタンを押さえドラッグする。マウスの動きに合わせ、マウスボタンを離すまでテキストの輪郭がポインタに引きずられる。

テキストラベルを消去するには、マウスでそれを指し、右マウスボタンを押す。ラベルを消去する場合、消去確認のプロンプトがないので注意を要する。消去してしまったら、再度タイプすることによって保管し直すしかない。コントロールパネル上のSAVEボタンは、作成されたテキストラベルのどのようなものでも保護する。異なったフォントを選択するには、'-font'コマンドオプションを使用する。

５．６．１０　データ検証

立体ボリューム内で様々な位置における個別のデータ値を検査するためデータ検証が役立つ。コントロールパネル上のPROBEラジオボタンをクリックすると、立体ボックス内に立体カーソルが現わ

れ、右マウスボタンを使用して動かすことができる。各物理的変数に対する現行のタイムステップの値が立体ウィンドウの左端にそって表示される。物理的な単位が変数に対して設定されたら、その値の隣に表示される。単位は、データ変換プログラム内のv5dSetUnits()機能で割り当てられる。

GRID#'sボタンをONにすると、その検証は一体型の格子座標に強制される。それは、カーソルが最も近い個々の格子座標に'snap'（はまる）ことである。

５．６．１１　新しい変数の作成

コントロールパネル上のNEW VARボタンは、ボタンマトリックスに対する新しい物理的変数を追加するのに使用される。３種類の追加可能な新しい変数は以下の通りである。

＊複製された変数：これらは実在する変数のコピーである。例えば、同じ変数の異なった同一表面を2つ同時に作成するには、この複製された変数を使用する。

＊外部機能の変数：実在する変数の機能として新しい変数を計算するには、外部機能（自分で書き込んだ）を呼び出すことができる。

＊計算された変数：実在する変数の値を格納する方式でタイプすることによって、新しい変数を計算することができる。

NEW VARボタンをクリックするとウィンドウが現われ、複製された変数と呼び出した外部機能を列挙し、新しい変数を計算するための方式でタイプさせる。新しい変数が作成された後は、新しい縦の行のボタンが、新しい変数としてコントロールパネルに追加される。他の物に似た変数の、同一表面、軌跡、スライス等を使用し作成できる。

（１）複製された変数

Uの速度素子の変数を複製する場合、その同一表面を+20と-20の両方にすることになる。最初は、NEW VARボタンをクリックし、それからポップアップウィンドウからUを選択する。複製された変数はU'と名付けられる。このU'は他のどの変数とも同じで、その同一表面を作成するために扱われる。

（２）タイプイン方式

タイプイン方式は、実在する変数の機能として新しい変数を計算する数学的な表現でタイプさせる。例えば、U、V、Wからの速度速度を計算するためには、以下の数式を入力する。

```
SPD3D=SQRT(U*U+V*V+W*W)
```

気温に対する露点(TD)の制限を計算するためには、以下の数式を入力する。

```
RATIO=TD/T
```

数式は実在する変数、番号、算術操作の+、-、*、/、**（パラメータ関数）等の名称、及びSQRT、EXP、LOG、SIN、COS、TAN、ATAN（弧の接線）、ABS（絶対値）、MIN,MAX等の関数の名称

を使用する。MINとMAXは、他の関数が全て1つの引数を取る間に2つの引数を取る。

OKボタンをクリックして新しい変数を計算するか、CANCELで数式を廃棄する。NEW VARポップアップウィンドウから再度それを選択することにより、後に数式を編集することができる。

（3）外部分析機能

外部分析機能は高度な機能である。外部分析機能はFORTRANで書き込まれており、実在する変数の機能として新しい変数を作成するために動画化シミュレーション・プログラムによって呼び出される。例で挙げたように、SPD3Dの機能を含んでおり、SPD3D=SQRT(U*U+V*V+W*W)のような速度の速度を計算する。外部関数機能は動画化シミュレーション・プログラムを経験済みのユーザーで、更にFORTRANの熟練プログラマに向いているといえる。

全ての外部関数は"userfuncs"（これは動画化シミュレーション・プログラム.h fileで変換される）という名称のディレクトリ内にあり、動画化シミュレーション・プログラム実行時に現行のディレクトリに対応する。例えば、"/usr/jones/data"内で常に動画化シミュレーション・プログラムを実行させるとするならば、分析関数は"/usr/jones/data/userfuncs"内になくてはならなくなる。更にこのディレクトリはスクリプト"externf"を格納しており、それは機能の編集に使用される。

外部関数の書き込みを行うには、提供されたサンプルの1つをコピーし、それを修正するのが一番良い。内蔵されている"userfuncs/example.f"は、この目的に対して説明がなされている。後に動画化シミュレーション・プログラム内部からその関数を呼び出す時に、それは各タイムステップ毎に1度起動される。その関数に通じる引数は以下の事項を含む。

＊データファイル内の物理的変数の番号
＊各変数の名称
＊立体格子のサイズ
＊タイムステップの日付と時間
＊マップ投射と縦の座標系の情報
＊各物理的変数に対するデータの実際の立体格子

その関数は計算を必要とするのに必要なものを見い出す物理的変数の名称のリストを操作する役割を果たさなくてはならなくなる。それは動画化シミュレーション・プログラムへ戻るデータの新しい格子を発生させる実際の計算を行わなくてはならない。これをどのように行うかを実演するサンプルを内蔵してある。関数引数の文書化を詳術するsample.fを特に見るとよい。マップ投射と縦の座標系引数は、v5dCreateライブラリ呼び出しと全く同じ方法で作動する。

その関数を"delta"と名付けたとするとFORTRANプログラムの名称が"delta.f"になる。"externf delta"とタイプすることによって関数をコンパイルすることになる。エラーが無ければ、実行可能なファイル"delta"が書き込まれる。その後、動画化シミュレーション・プログラム内でNEW VARを選択する時に、ポップアップウィンドウ内の機能一覧に"delta"が現われる。

外部関数に通じる格子データを得るために、動画化シミュレーション・プログラムに対して独自の圧縮されてないMcIDASファイルか、または圧縮されv5d/comp5dファイルの2つの位置がある。圧縮されてないMcIDASデータの方が、より精密であるが故に優れているといえる。McIDASファイルが見つからない場合は、動画化シミュレーション・プログラムがメモリ内に所持している圧縮されたデータが外部関数に通じることになる。

関数内部からデータ証明の位置と値を取り戻すことができる。データ証明の位置を得るのには以下の（コマンド）を使用する。

```
CALL PROBEPOS (ROW, COL, LEV, LAT, LON, HGT)
```

格子座標内の位置は行、列、標高へ戻る。地理的座標の位置は緯度、経度、高度へ戻る。

実際の試験位置とタイムステップにおけるどの物理的変数の値を得るにも、以下の（コマンド）を使用する。

```
VALUE = PROBEVAL(VAR)
```

VARが、どの物理的変数を要望するかを設定する地点である。

５．６．１２　画像ファイルの保管と印刷

コントロールパネル上のSAVE PICボタンは、立体ウィンドウ内の画像をファイルへ保管するために使用される。SAVE PICをクリックすると、ファイル書式とファイル名を選択できる所にポップアップウィンドウが現われる。ファイル形式の選択は使用している機種によって異なる。動画化シミュレーション・プログラムによって支援される書式は以下の通りである。

XWD	xwud、またはxvでは役割を果たさない、Xウィンドウダンプ
RGB	ipaste、またはxvでは役割を果たさない、SGI画像ファイル書式
GIF	xv、または他の多数のプログラムでは役割を果たさない、標準GIF書式
PostScript	ghostviewのようなプログラムで画面上に印刷、または写し出される
Color PostScript	ghostview状のプログラムで印刷、または写し出される

動画化シミュレーション・プログラム（GLを使用する）のirix4とirix5の機器構成は、直接RGBファイルの書き込みを行う。GIFファイルの作成には、togifプログラムが必須である。PostScriptファイルの作成には、topsプログラムが必要である。togif、tops、その他多くのRGBファイルの変換プログラムは、IRIXの標準的である。もしも、それらが/usr/sbinにおいて見つからない場合は、IRIX CD-ROMからインストールする。

動画化シミュレーション・プログラム（OpenGLを使用する）の他の全ての機器構成は、直接XWDファイルの書き込みを行う。RGBファイルの作成には、fromxwdプログラムが使用される。gifファイルを作成するには、fromxwdとtogifの両方（SGIシステム上でのみ使用可能）が必要である。無彩色スケールのPostScriptファイルを作成するには、xprの有用性（X11で規格化された）が求められる。PostScriptファイルを作成するには、topsプログラムが必要とされる（SGIシステム上でのみ使用可能）。

前述のどの有用性も持ち合わせてなければ、画像ファイルの変換のためにxvを使用してみるべきである。xvはcontrib/内のexport.lcs.mit.eduと、pub/ディレクトリ内のftp.cis.upenn.eduからのftpによって使用可能である。

動画化シミュレーション・プログラム画像を印刷するには、立体ウィンドウ越しにマウスポインタを設定し、Pのキーを押す。その動作の確認のためにプロンプトが来る。動画化シミュレーション・プログラムは、Post Script画像ファイルを省略値プリンター、またはPRINTER環境の変数によって指

定されるプリンタへ送るため、lprを使用する。

５．６．１３　テクスチャマッピング

テクスチャマッピングは、コンピューターグラフィック用語で、３Dの表面上に２次元の画像を表示することを指す。動画化シミュレーション・プログラムでは地形図（地形図が消えているときは、立体ボックスのボタン）に重ねて、衛星や地図といった画像を表示することができる。テクスチャマッピングは、SGIシステムかメサ・ライブラリを使用したシステムでのみ利用でき、テクスチャマッピング用の支援ハードウエアを使用することを勧める。Vis5dには３種類のテクスチャ／画像マッピングがあり、コマンドラインで指定できる。

-area N

NはMcIDASエリアファイルの数列の最初の記号である。ファイル読み出しの数は自分のデータファイルのタイムステップ数と同じになる。画像は全て同じ大きさにするのがよい。再マッピングする必要がある場合、Vis5dを使わなければならない。

例：仮に自分のデータファイルが４タイムステップで、エリア100と指定したとすると、
AREA0100、AREA0101、AREA0102、AREA0103がロードされ、かつ表示される。

このオプションには、SGIシステムでのみ利用できるMcIDASライブラリが必要である。

-sequencefile

これは-areaオプションのような働きをする。ただしデータが、McIDASエリアファイルではなく、非常に単純なファイル形式から来る場合はこの限りではない。ファイルは３int'sで始まり、この３int'sにはシーケンス中の画像の数、一画像に対するラインの数（ラインに対する画素の数）が含まれる。srcディレクトリのimage.cファイルにある読み込みread_texture_sequence機能は、このシーケンスファイルを読み込み、画像シーケンスファイルを作成したい人のためのファイル形式レファレンスとしての役割を果たす。

-texturefile

このオプションは画像を一つ指定し、それをタイムステップの間中、地形図にかぶせて表示するためのものである。このファイルはSGI RGB ファイル形式になっている。XVプログラムを使用して、自分の画像をRGB形式にコンバートすることができる。テクスチャマップが利用できるときは、コントロールパネル上のTEXTUREボタンで画像のオン、オフ表示を切り換えられる。

５．６．１４　Tclスクリプティング

動画化シミュレーション・プログラムにはスクリプト機能がある。つまり、Tcl言語のコマンドのテキストファイルで動画化シミュレーション・プログラムをコントロールできる。

５．６．１５　キーボードの機能

マウスのポインタが立体ビューウィンドウの内側に入り込んでいるとき、以下に述べるキーボードの機能を呼び出すことができる。

キー	機能
F1	コントロールパネルウィンドウを上下に移動する。このキーは-fullオプションのとき役に立つ。
F2	システム情報（使用中のメモリーも含む）の表示と計算されるグラフィック表示を切り換える。
P	現在のウィンドウの画像を印刷する。ポストスクリプトプリンタが使用可能でなければならない。自分のシェルのプリンタ環境変数を設定し、どのプリンタを使うか指定する。
S	動画の動きを遅くする。フレーム間の最小時間を10ミリセカンド増やす。
F	動画の動きを速める。フレーム間の最小時間を10ミリセカンド減らす。

自分独自のキーボード機能をプログラムしたい場合は、ファイル　src/gui.cの中のfunc1()、func2()、func3()等の機能を探すこと。これらは、対応するファンクションキーを押すと呼び出される。

５．６．１６　制限事項

動画化シミュレーション・プログラムのSGI版で複数のCPUを使用できれば、裏でグラフィックスの計算をし、動画化シミュレーション・プログラムの速度を速められる。他のシステムでは、動画化シミュレーション・プログラムは、ユーザーとのインタラクションでグラフィックスの計算を割り込ませようとする。

この方法では、グラフィックス計算の処理が終わるまで、ユーザーインタフェイスが遅くなる。動画化シミュレーション・プログラムのユーザーインターフェイスを言葉で記述するのは、複雑かもしれないが、実はこれでも簡単なものにしようと努力した結果である。サンプルデータを使って少々練習すれば、スムーズに感じるようになると思われる。

５．７　v5dimportのユーティリティ

v5dimportのユーティリティは格子ファイルをv5d形式にコンバートするための新プログラムで、複数のソースファイルを組合せ、新しい座標系へ再サンプリングし、変数やタイムステップを選別する。グラフィクス・ユーザーインターフェイスとラインコマンド・ユーザインターフェイスの両方がある。

例えば、v5dimportを使うとMcIDAS GR3DファイルとMcIDAS GRIDファイルを読み込み、全データのランバート・コンフォーマル投射への再サンプリング、CWATやVORT変数の省略、lambert1.v5dという名前の動画化シミュレーション・プログラムファイルへのデータの書き込みができる。
v5dimport使用時の事象の基本的な順序は以下のとおりである。

1)入力ファイルの読み込み。
2)タイムステップ、物理変数、射影、または垂直座標系に従って出力用の格子を選択する。
3)出力ファイル用に射影と垂直座標系を設定する。
4)出力ファイルを書き込む。このとき再サンプリングが行われる。
5)出力ファイル上で動画化シミュレーション・プログラムを起動してもよい。

v5dimportは以下のファイル形式を読み込むことができる。

McIDAS GR3DとGRIDファイル
v5dとcomp5dファイル

GRADSファイル
"UW vis"ファイル（ウィスコンシン大学で使用されている）
EPA MM4とRADMファイル（Clays上でのみ）

（1）v5dimportの操作

シェルからv5dimportを起動する。

```
v5dimport [-path pathname] [files]
```

ここでは、［files］は入力ファイルのオプショナルリストであり、［-path pathname］は、"pathname"というディレクトリが、現在のディレクトリの場所で、入力ファイルブラウザ用、あるいは出力ファイル作成用に、省略時値として使われるように指定する。v5dimportをスタートさせると、メインウィンドウが現われる。その構成は以下のようになっている。

1) 入力ファイルからスキャンされた全格子のリスト。このリストはスクロールする。
2) 変数名、タイムステップ、投射、あるいは垂直座標系に従って格子を選択　／選別するためのボタン。
3) 出力ファイルを記述し、作成するためのボタンやタイプインフィールド。

（2）インプット格子の読み込み

いつでも"Read file..."ボタンをクリックして、追加の格子ファイルをv5dimportに読み込ませることができる。ファイルセレクタを使って、自分のファイルを位置付けし、OKまたはCANCELをクリックする。

最初に入力ファイルを全て読み込んでしまうのがよい。というのは、新たなファイルを読み込こむたびに、全格子が出力用に選択され、その前に行った選択が無効になっていまうからである。

"Discard all grids"というボタンは文字通りの作業をするボタンである。v5dimportから出て再スタートさせることになる。各入力ファイルを読み込んだ後、ウィンドウの上半分に表示される格子のリストは時間順に再分類され、その中で変数名順に再分類される。リストの桁は以下の通りである。

Grid	格子ナンバー
YYDDD	格子の年月日
HHMMSS	格子の時間（時、分、秒）
Variable	変数名
Nr	格子の行数
Nc	格子の桁数
Nl	格子のレベル
Proj#	投射ナンバー（"Select by projection..."のウィンドウを参照する）
VCS#	垂直座標系ナンバー（"Select by VCR..."を参照する）
Filename	その格子が入っているファイルの名前

（3）出力用の格子選択

入力ファイルから、ある物理変数やタイムステップを捨てて、出力ファイルに書き込まれないようにしたいというケースがよくある。省略値によって出力用に全変数が選択される。

変数を選択／選別するには、"Select by variable.."ボタンをクリックする。全変数が載っているポップアップウィンドウが現われる。マウスを使って、欲しいタイムステップを選択、あるいは省略したいタイムステップを解除する。このボタンをホールドしている間は、マウスをドラッグするだけで、

幾つかのタイムステップを選択／解除できる。最後に、"Select by projection ...", "Select by VCS..."ボタンで、マップ投射もしくは垂直座標系に従って、格子を取捨する。

タイムステップ、変数、投射、VCSを選択するたびに、その影響を被った格子が、メイン格子リストで強調表示／強調表示解除される。"Select All"、"Select None"は文字どおり、全選択、無選択で実行する。

（4）出力ファイルの定義

読み込まれた最初のファイルから、出力ファイル（格子サイズ、投射等）用の省略時媒介変数が取り出される。自分の出力ファイルをつくる前に、媒介変数の値を変えなくてはならないことがよくあるため、必ずこの媒介変数を見直したほうがよい。

出力ファイル用の行、桁、レベルの数を指定するには、メインウィンドウにある"Rows", "Columns"及び"Max Levels"と表示されたタイプインフィールドを使う。省略値が正しくない場合、新たな値をタイプする。

出力ファイル用のマップ投射を見たり、変更したりするには、"Map Projection..."ボタンをクリックする。このポップアップウィンドウで、マップ投射の種類を選んだり、特定の投射媒介変数を入力することができる。"Guess"ボタンもあり、現在選択されている格子リストに与えられた出力投射で、適当なものを探すことができる。"Select by Projection"ポップアップウィンドウを画面に表示しておいて、出力投射と入力投射を比較することができる。

出力ファイル用の垂直座標系を見たり、変更するには、"Vertical Coodinate System..."ボタンをクリックする。このポップアップウィンドウで、垂直座標系の種類を選んだり、特定の媒介変数を入力することができる。このウィンドウには"Guess"ボタンもあり、適当な省略値を探すことができる。"Select by VCS"ポップアップウィンドウを画面に表示しておいて、出力VCSと入力VCSを比較することができる。

（5）出力ファイルの作成

メインウィンドウの底辺部にあるタイプインフィールドに、出力ファイルのファイル名を入力し、"Make"をクリックする。ファイルの変換が行われ、メッセージが印刷される。エラーが発生した場合、処理を停止する。入力格子座標系から出力用の新しい座標系へデータを再サンプリングする必要がある場合は、出力ファイルの生成には時間がかかることに注意する。

"Visualize"をクリックすると、ファイルができ、次に自動的にそのファイルの上に動画化シミュレーション・プログラムが立ち上がる（最初に"Make"をクリックしなくてよい）。タイプインフィールドにファイル名を入力する場合は、その名前を使用する。そうでなければ、自分のログインネームの後に".v5d"をつけたものを使う。動画化シミュレーション・プログラムコマンドで、コマンドラインオプションが欲しい場合、"vis5_option"というファイルに入れる（例　"-mbs 64"）。

（6）その他

オプションのウィンドウを利用するには、"Options..."ボタンをクリックする。

最初の項目は、"combining of co-located data"を統制する。幾つかの立体格子が出力用に選択され、そ

れが空間と時間に関して同じ場所に設置されることは少なくない。出力ファイルに入れる値を計算する場合、その位置において、より解像度の高い格子からデータの値を選んでもよいし、あるいはその格子位置での全格子の平均値を取ってもよい。

二番めの項目は、格子データの出力ファイルでの圧縮法を統制する。省略値によって、格子の値は整数で1-バイトに基準化される。そうでなければ、よりよい解像度を求めて整数で2-バイトに基準化できるし、あるいは4-バイトの浮動小数点値を選択し、圧縮も基準化も行わないということもできる。このオプションは、ファイルサイズや精度のトレードオフを表している。

（7）v5dimportのテキストインターフェイスの使用

Xが使用できないとき、あるいはスクリプトでv5dimportを作動させたいときに、v5dimportへのテキスト／タイプイン（text/type-in）インターフェイスを使う。以下のテキストモードを入力し、v5dimportを起動する。

```
v5dimport -t [-path pathname] [files]
```

ここでは、［files］は入力ファイルのオプショナルリストであり、［-path pathname］は"pathname"という名前のディレクトリが、現在のディレクトリの場所で、入力ファイルブラウザ用、あるいは出力ファイル作成用に省略時値として使われることを指定する。テキストインターフェイスを通せば、自分のシェルのインポートリダイレクション機能を使うことによりスクリプトでv5dimportを作動することができる。

```
v5dimport -t <script
```

-tオプションでv5dimportを呼び出すと、>>プロンプトが表示される。そのプロンプトでは以下のコマンドを出すことができる。

exit	v5dimportを終了する
help	オンラインヘルプ
list	格子、タイムステップ、変数、マップ投射、垂直座標系のリストを表示する
read	入力ファイルを読み込む
keep/omit	格子、タイムステップ、変数、マップ投射、垂直座標系を出力ファイルに入れたり、そこからはずしたりするか選択する。
info	出力ファイル用に媒介変数を表示する
rows	出力ファイル用に格子行の数を指定する
columns	出力ファイル用に格子桁の数を指定する
levels	出力ファイル用に格子レベルの最大数を指定する
projection	出力ファイルのマップ投射を指定する
vertical	出力ファイルの垂直座標系を指定する
make	出力ファイルを作成する
visualiz	出力ファイルを作成し、動画化シミュレーション・プログラムを起動させる．

v5dimportにテキストインターフェイスを使用するのは、戦略という点ではグラフィックインターフェイスを使用するのと同じである。

1）入力ファイルを読み込む
2）タイムステップ、変数、投射、and/orはVCSによって格子を選択する。この選択は一般的には一連

のリストによって行われ、コマンドを無視したり、続行したりする。
3) 出力ファイの媒介変数を設定／調整する。一般的には、インフォメーション、行、けた、レベル、投射、垂直コマンドといった一連のものである。
4) 出力ファイルを作成する。もしくは出力ファイルを作成し、動画化シミュレーション・プログラムを起動させる。

v5dimportのスクリプトは、v5dimportコマンドと引数だけのASCIIファイルである。最も簡潔なケースでは、たった2、3のコマンドだけで成りたっていることもある。

```
例：# read my file, omit two vars, write v5d file
    read mydata.dat
    omit var CW
    omit var RW
    make outdata.v5d
    exit
```

v5dimportがスクリプトを実行する際、各コマンドとその結果が印刷される。#で始まるラインは注釈とみなされ無視される。

（8）新ファイル形式のための追加支援

v5dimportは、新しいファイル形式を読み込むための追加コードが簡単なものになるように書かれている。v5dimportのソースコードはinport/サブディレクトリの中にある。comment/***ADD NEW FORMATS HERE***/を探し、新しいファイル形式を支援するには、どこにコードを追加すればよいかを見る。

基本的に、２つの新機能を書き込む必要がある。一つはgrid_info strucsのリストを構築するためのファイル形式を走査するもの。もう一つは、自分のファイルからgrid_info strucsに与えられた実際の格子データを読み込むものである。これらの機能は、read_foo.cという新ファイルに入れるとよい。ここでfooというのは、自分のファイル形式の名前である。そして、自分の機能を使用するために、file.cファイルをアップデートする。ガイドとして在るread_*.cファイルを使う。

（9）特定のファイル形式に関する注意

記号EPAは、EPAファイルを読み込むことができるシステム上にのみ、-DEPAというccコマンドラインに規定されている。EPAが提供するファイル読み込み機能はクレイコンピュータでしか作動しないので、現在はクレイシステムだけがEPAファイルを読み込める。

記号MCIDASは、libmcidas.a.ファイルを使用できるシステムにのみ、-DMCIDASというコマンドラインに規定されている。現在支援されているのは32ビット形式のSGIだけである。

6　まとめ

前章までに考察してきたように、地層処分ヴァーチャル・エンジニアリング・システムの開発に於けるシミュレーション機能の開発においては、従来からの工学的手法によるモデル化とプログラミングとは別な、ソフトウェア科学を手法の基盤としたシミュレーション手法が考えられる。例えば、オブジェクト指向によるモデル、複雑系やマルチエージェントモデル、そして、並列処理と協調計算のモデル等である。また、そのようなソフトウェアを開発する上で、ソフトウェア複雑化の要因を把握し管理することも重要である。従って、それらを応用し複数の計算を組み合わせて試行錯誤的にシミュレーションをするシステム（本研究では**モデル協調計算**と呼ぶこととした。これは、複雑系モデルである）を開発し、複雑な性能評価問題に対応するシミュレーション技術の目処を付けることが、今後の具体的な技術開発課題である。

連成問題ー＞モデル協調計算（複雑系）

例えば、地層処分システムの性能評価は、地震等の確率的に生じる事象による変化を除いて、地層中の環境が安定している場合、地層処分された廃棄体から放射性核種が放出され、地下水により生物圏まで輸送されるシナリオを基本としている。そのような安定した地層中での環境に添って核種移行を考えた場合、それだけでも非常に複雑な環境評価モデル開発と評価の問題であることが解かる。

領域１：ガラス固化体（崩壊熱モデル、放射性核種放出モデル、等）
領域２：オーバーパック（腐食モデル、腐食物ベントナイト化合モデル、等）
領域３：ベントナイト（２層多孔質媒体地下水流れ、熱・オーバーパック腐食化合モデル、等）
領域４：人工バリア周辺（人工バリアと岩盤との熱応力モデル、コロイド、核種収着、沈殿、等）
領域５：地層中（広域的地下水汚染物質移行モデル、亀裂水理、等）

地層処分ヴァーチャル・エンジニアリング・システムの操作では、実際に観測できる現象は、上記のモデルが複雑に組み合わさったものと考えられる。評価シナリオは、例えば温度場の変化が力学系に作用し、地下水流動を変更することにより、人口バリアの冠水や亀裂を生じ、結果として核種が滲出して輸送されるシナリオが考えられる。このようなシナリオにそったシミュレーションを、対話方式で試行錯誤的に行なうには、下記の操作機能がアプリケーション・システムに要求される。

操作機能１：シミュレーション・シナリオの一覧
　　シナリオは、充分に評価され、妥当なものとしてデータベース管理されていること。
操作機能２：シミュレーション条件の決定
　　シナリオの選択、個別評価モデルの選択、計算条件の設定等
操作機能３：シミュレーションの実行と制御
　　シミュレーションの実行、実時間視覚化、計算結果の解析等

また、シミュレーション・システムを開発する上で、対話型実時間可視化シミュレーション機能は、今後の性能評価シミュレーションの技術研究に於て重要な課題である。それは、**“個別の計算モデルを複雑に組み合わせ協調して計算させる”**プログラムの具体的なコーディング技術の研究であり、シ

ミュレーション機能開発の最も基礎的な部分である。

例えば、MIT X/11に代表される大規模で複雑なシステムプログラムの多くは、オブジェクト指向を開発戦略に取り入れている。それは、オブジェクト指向がソフトウェアの大規模化からくる複雑度の増大に伴う信頼性低下や、複数の計算モデルを組み合わせて実行するといった要求に対する、現在考えられる具体的な手法であるためと考えられる。従って、大規模で複雑な統合化シミュレーション・システムを開発するためには、オブジェクト指向の考え方を、物理工学シミュレーションで取られてきた要素還元論と合わせて整理することにより、計算科学の領域である大規模複雑系シミュレーションの技術開発戦略として期待できる。

オブジェクト指向

オブジェクト指向は、プログラム開発に関する新たな方法論である。この方法論によれば、メモリー上に展開されるプロセス、タスクといった従来の実行状態の手続は、オブジェクトと言う、より発展した概念に統合して考える。このオブジェクトの考え方に添って、物理・工学上の問題等をプログラミングすることにより、より複雑な問題を効率良くプログラムすることが期待できる。また、並列計算技術を併用することにより、より説得力のあるシミュレーションが期待できる。オブジェクト指向によるシミュレーション・プログラム開発には、新しい考え方（パラダイム）が必要であり、その影響はモデル開発にまで及ぶものと思われる。それは、シミュレーションの考え方を革新する思考方法と捉えることもできるであろう。オブジェクト・シミュレーションの考え方は、従来のシミュレーションの考え方とは異なったものである。それは、シミュレーション対象を極端に単純化するとともに、クラスとインスタンス、メッセージ交換という概念をプログラミングに採用し、移動や衝突を伴う対象をシミュレーションすることを可能にする。そのための最も基本的な手法は以下の通りである。

1：対象を構成する最も単純な最小要素を物理モデルとして考える。例えば、構造物はそれを構成する要素の組として考える。流体はそれを構成する分子等を代表的な粒子として考える。従って、流体は非常に単純な対象に分解できる。

2：対象とする物理モデルの運動について数理モデルを考える。対象が粒子であれば、移動や、摩擦、衝突等の扱いは単純なモデルとなる。このようにして、導いた数理モデルは外部から与えられたメッセージ（外力等）に応答して物理モデルの位置等決定する関数と考える。

3：対象となる物理モデルは、それを特徴付ける変量（インスタンス）と状態を決定する関数からなるプログラム・アルゴリズム（クラス）として記述する。

4：クラスに対して固有の現在座標等のメッセージを送り込むことにより、実行プロセス（オブジェクト）を必要な数だけ生み出す（C＋＋ではインスタンス化と言う）。

5：各オブジェクトは、互いにメッセージを交換し、時間と供に摩擦、衝突等の状態を確認し状態を変える。その状態を視覚化して観察することによりリアルなシミュレーション結果が得られる。

ただし、その様にして得られたシミュレーションは、場の状態や釣り合い等を記述する方程式から出発していない。従って、例えば、流れの問題を扱うことを考えた場合、従来のナビエ・ストークス方程式等の場の方程式から出発する有限要素法や有限差分法とは本質的に異なった考え方である。従来の手法が、どちらかと言うと観測された事実を数値的に再現するための定式化を目指していたこと

に比べ、オブジェクト指向は数値的に実験を行う手法と捉えることがより適切と考えられる。

協調計算プログラミング技術の課題

従来研究開発されてきた、科学技術のトップダウン・アプローチ（現象全体を確率論的モデルで記述したもの等）とボトムアップ・アプローチ（現象全体を個別の要素に還元して個別の決定論的モデルで記述していく手法）とは別な、第３の手法としてモデル協調計算という手法を考え導入した。その考え方は以下の通である。

> **コンピュータ中に組み立てられた仮想世界の中で、複数の数値計算モデルを協調させて並列的に実行することにより、従来のトップダウン・アプローチで押さえきれなかった、より精密な現象過程や、ボトムアップ・アプローチで記述が難しかった、ある程度の自由度をもった現象のシミュレーションを計算で実現する方法とした。このモデル協調計算は、理論、実験といった、従来の科学技術の方法とは別な、第３の科学技術の方法である計算科学技術の位置づけを明確にする科学思想とした。**

協調計算のプログラミングは並列処理を基礎とするであろう。その時、並列処理は単なる計算速度の向上を狙ったものではなく、モデルとアルゴリズムの並列化により、直列計算（従来型）では扱えない（表現できない）問題を取り扱うことを主眼とするべきである。例えば、手順並列法、結果並列法、専門家並列法といった並列問題のモデル化について、その特徴を良く把握し、協調計算プログラムのアルゴリズム開発に応用する必要がある。また、並列協調計算を実現する情報処理モデルの基本的な考え方として、オブジェクト指向のモデル化を導入することも大切である。更に、将来の人的リスク等の問題を考えた場合、独立した判断によって動作するオブジェクトとしてのエージェントという概念を把握し、モデルに組み込む手法を検討することも大切であると考えられる。そのようにして、目標とするシミュレーション機能は、マルチ・エージェント／マルチ・オブジェクトな並列協調計算を、マルチコンピュータで実現するものとしてアーキテクチュアを設計するべきである。

平成８年度のヴァーチャル・エンジニアリング・システムの開発プロジェクトの経験から、モデル協調計算を実現するための基礎技術は、オブジェクト指向によるシミュレーション、協調計算結果を実時間で３次元可視化する技術、異機種の複数計算機に機能分散を行ない１システムとして統合する技術を確立することにより、基本的に可能であると判断した

数理モデル開発の課題

並列協調計算を利用して、数理モデルを組み合わせた計算を実行することが考えられる。その時、個々のモデル空間座標は３次元であることが望ましいが、モデルとしては最も還元されたものを扱うことが良いと考えられれる。何故ならば、自由度の高い連成モデルは分割できないため、協調計算には向かないと考えられるためである（それらは、むしろ単体で利用すべきであろう）。但し、協調計算においては、個々のコードが、計算サイクルや時刻は他のコンピュータ（試作システムではＰＣ）のマスタークロックから受けること、連成させると物理量が繰り返し計算となるもの（例えば、圧力

Pは、熱を連成させると密度の変化と勾配の釣り合いを解く繰り返し問題になるであろう）には特別な、メッセージ交換技術の開発が必要と考えられる。

新しいパラダイムの受容

いずれにしても、本研究で取り上げているシミュレーション手法（マシンパラレル・モデル協調計算）は、全く新しいシミュレーションの考え方である。この考え方は、シミュレーションと捉えるより、仮想世界における数値実験と考えるべきである。そのような、新しい科学に対する考え方が受け入れられることが重要な課題である。

システム開発には個別の専門的知識より広範な知識が必要

システムの設計開発にあたり、現象評価に係わるシナリオ、個別現象過程、そして、ソフトウェア・エンジニアリングについて広範な知識が必要であると判断された。個別の専門領域に限定してきた従来の技術者教育の今後の課題として考えられる。

以上のように、地層処分性能評価研究を効率的に支援することを目的とした、平成８年度のヴァーチャル・エンジニアリング・システムの開発プロジェクトでは、複雑なモデルを試行錯誤的に組み合わせてシミュレーションを実行する、システム開発技術の基礎的な部分の試作と評価を行なった。その結果、以上の新しい概念とシステム開発技術の成果が得られ、これからの計算科学技術の新しい成果が期待できる考えられた。

動画化シミュレーション・プログラム

動作画面集

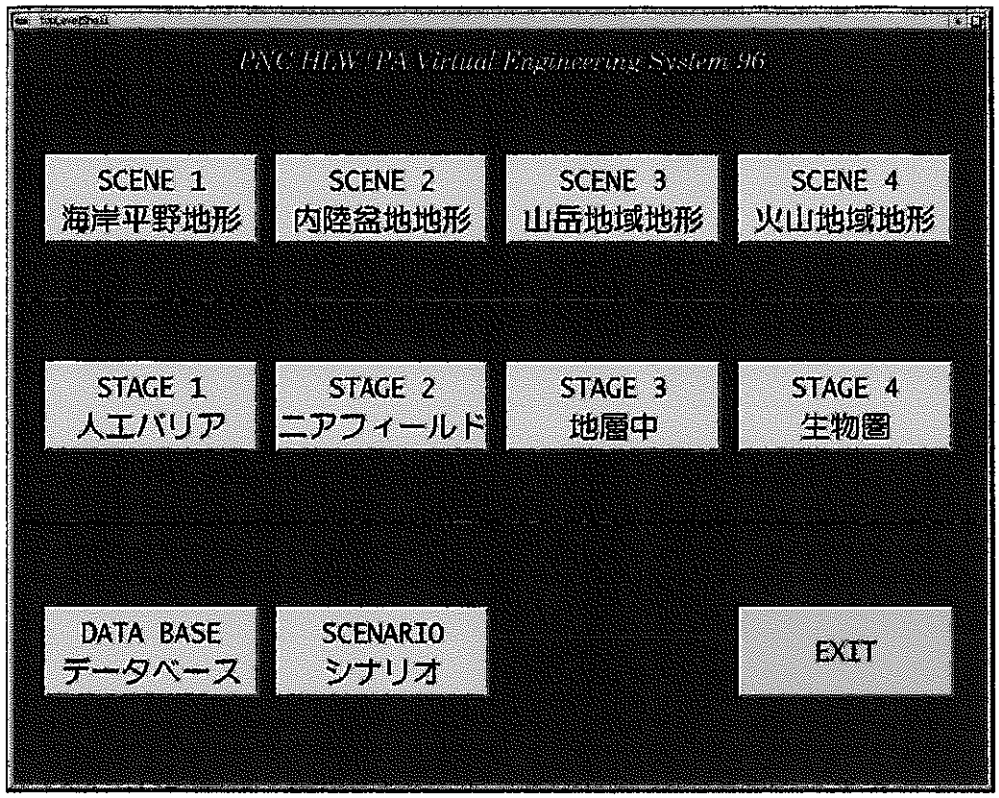

図１　主操作画面１

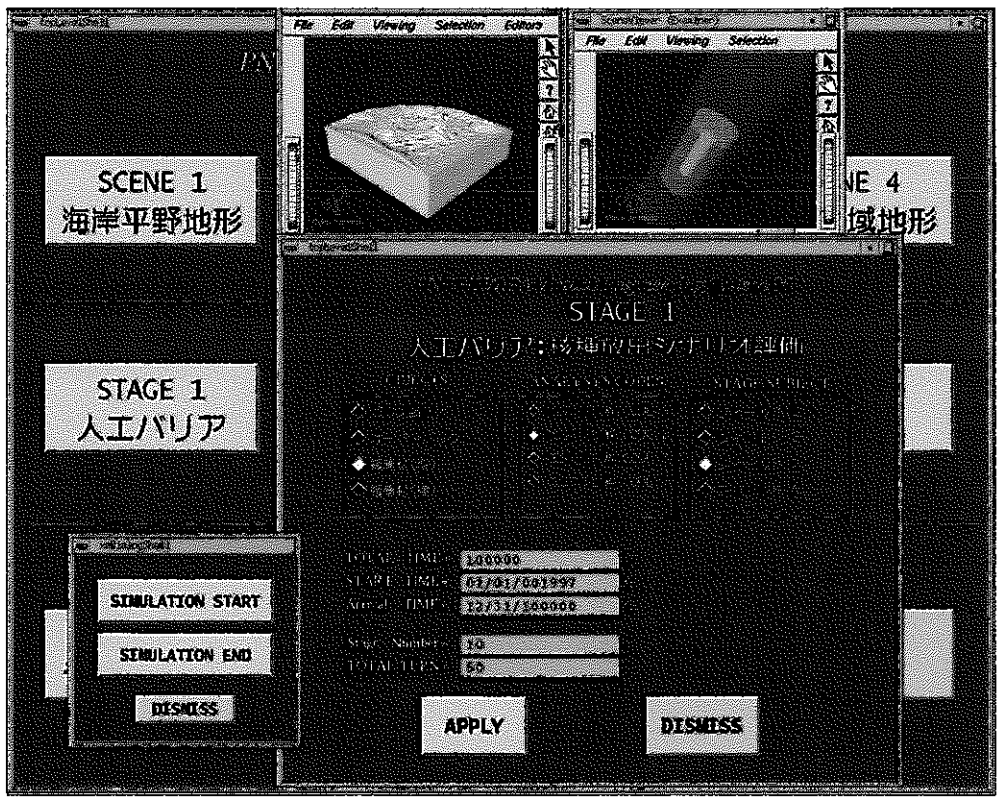

図２　主操作画面２

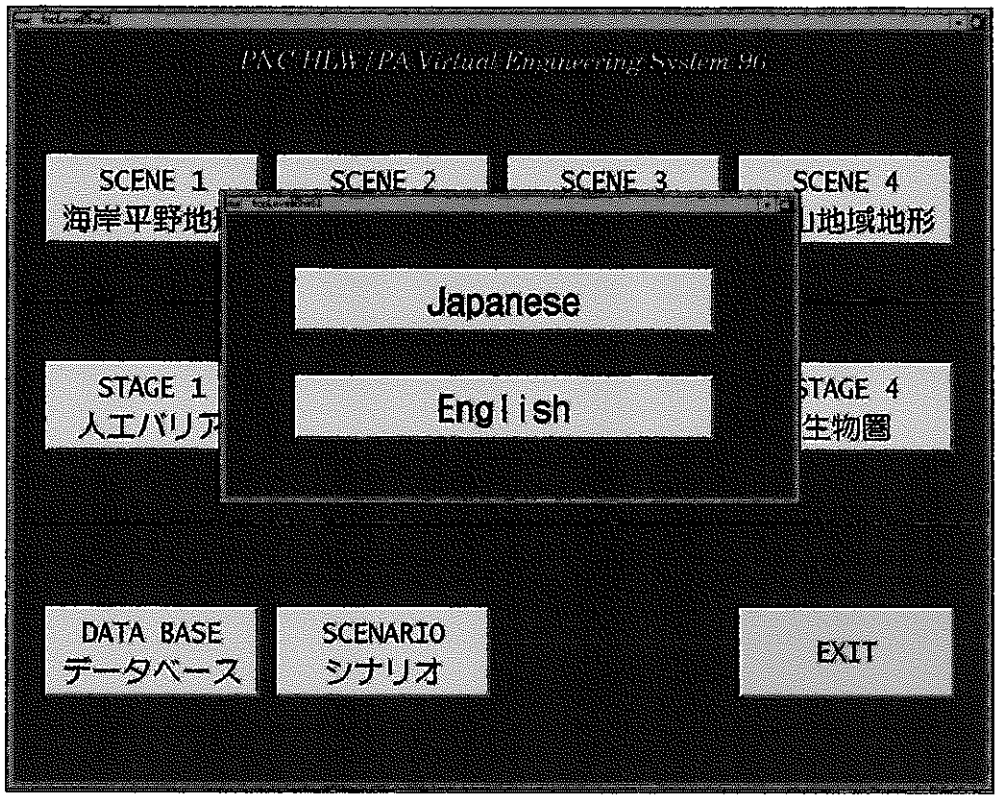

図 3　FEP操作画面 1

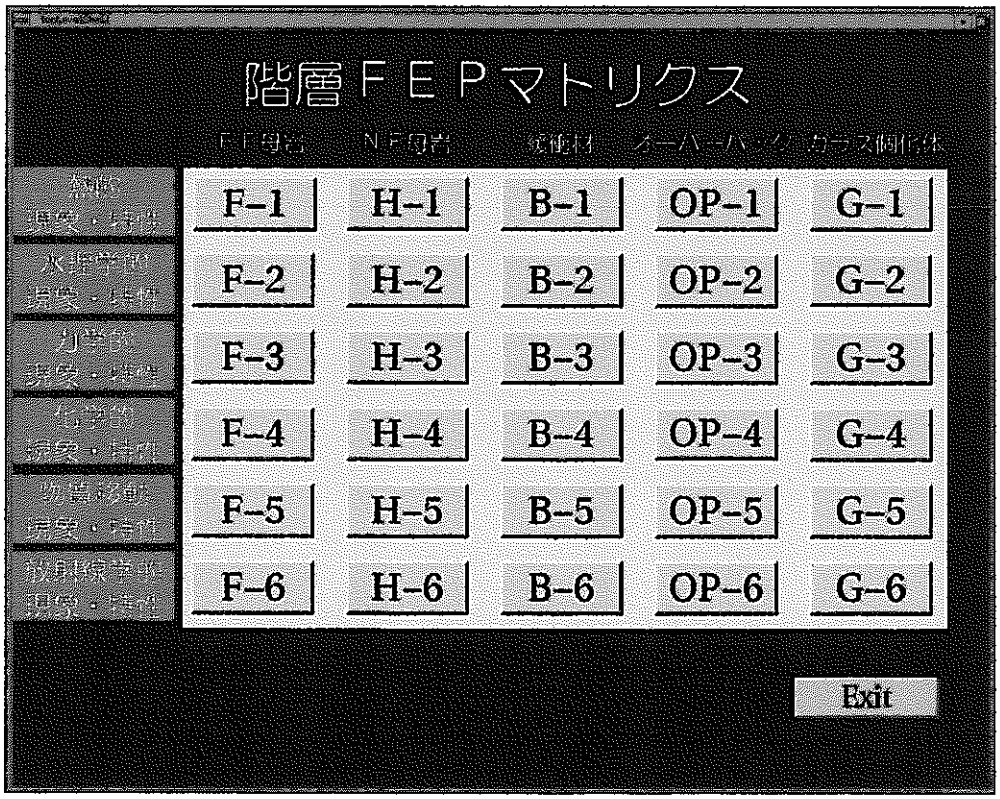

図 4　FEP操作画面 2

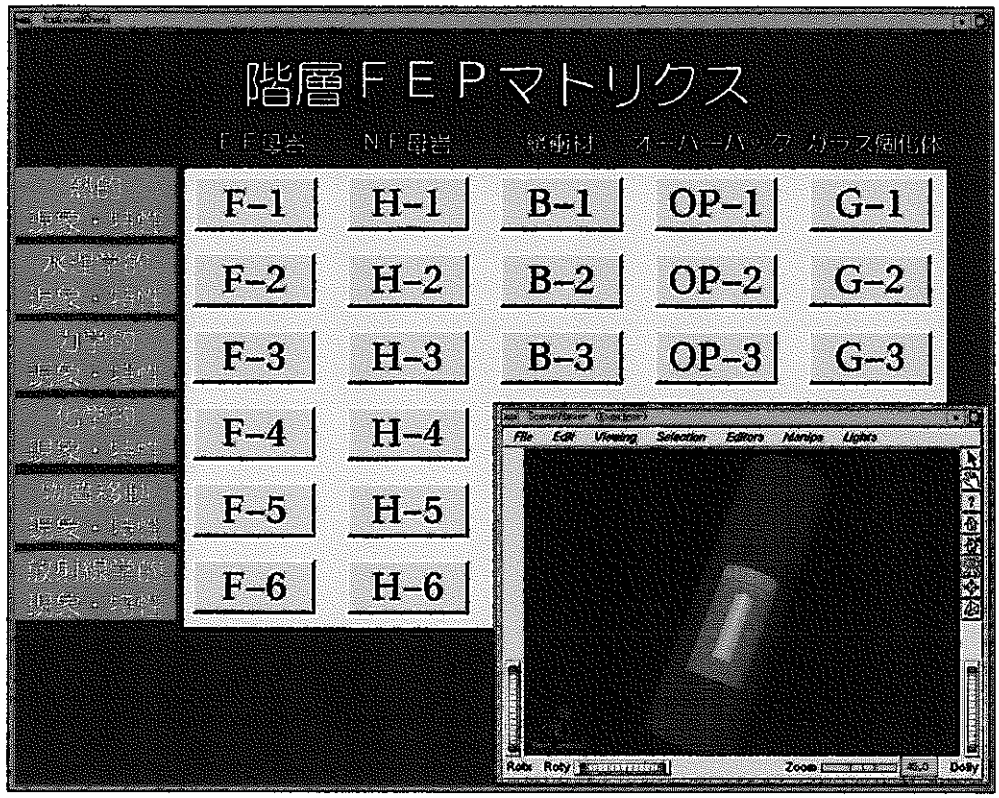

図５　FEP操作画面３

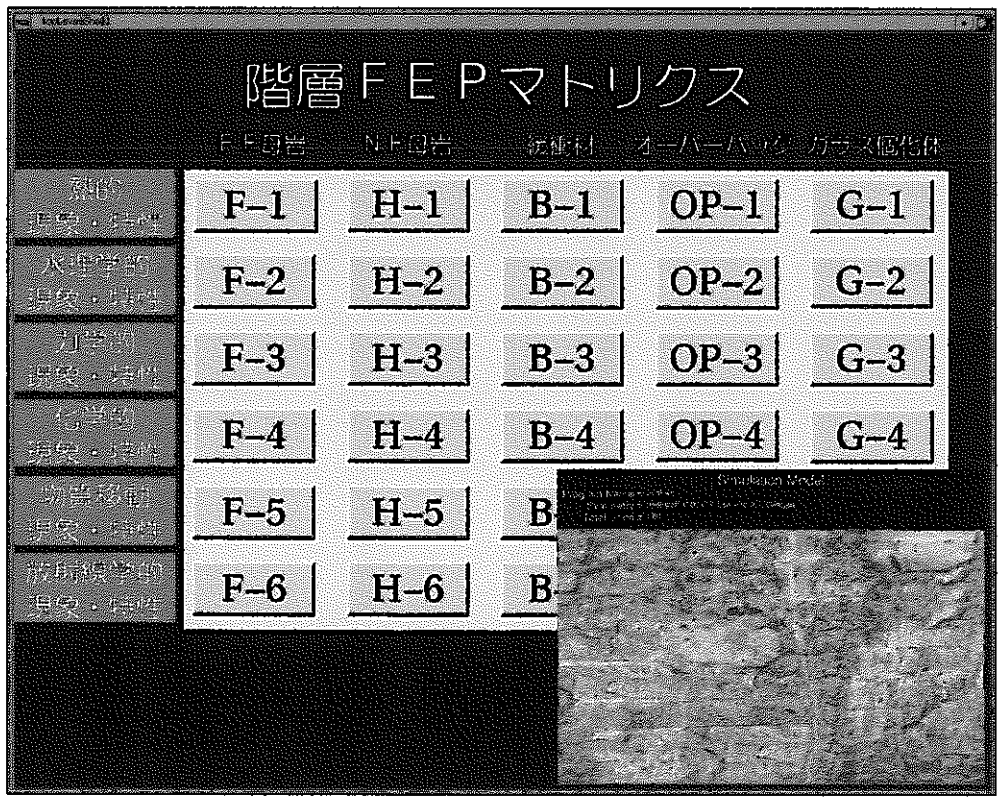

図６　FEP操作画面４

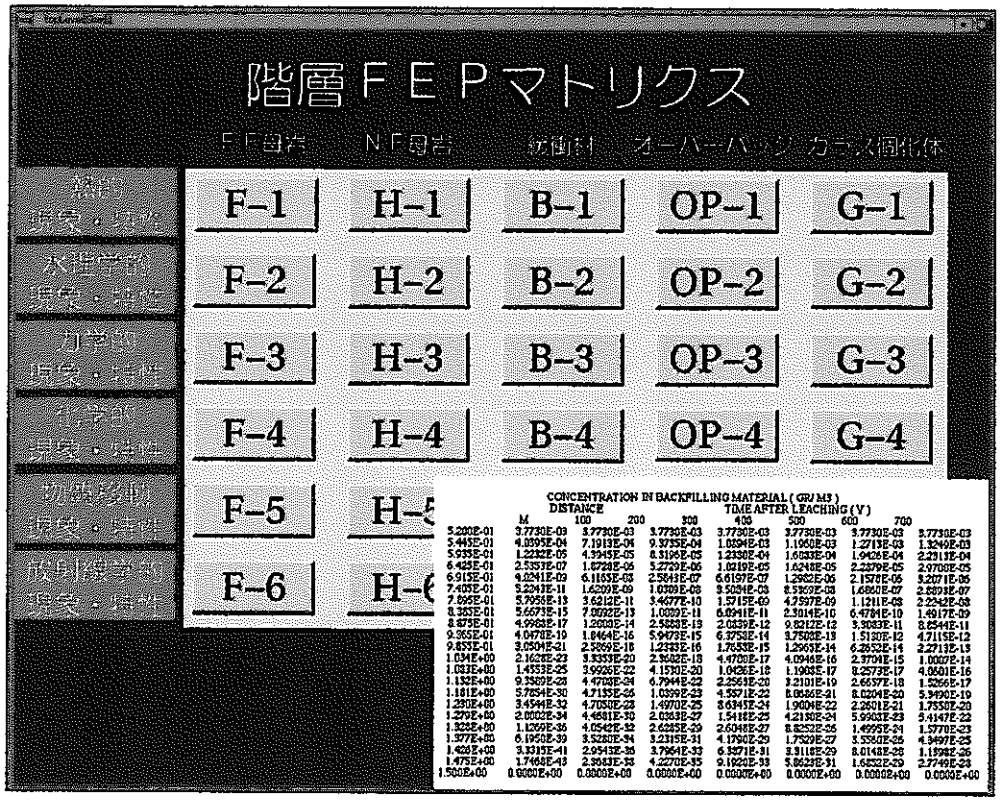

図７　FEP操作画面５

図８　FEP操作画面６

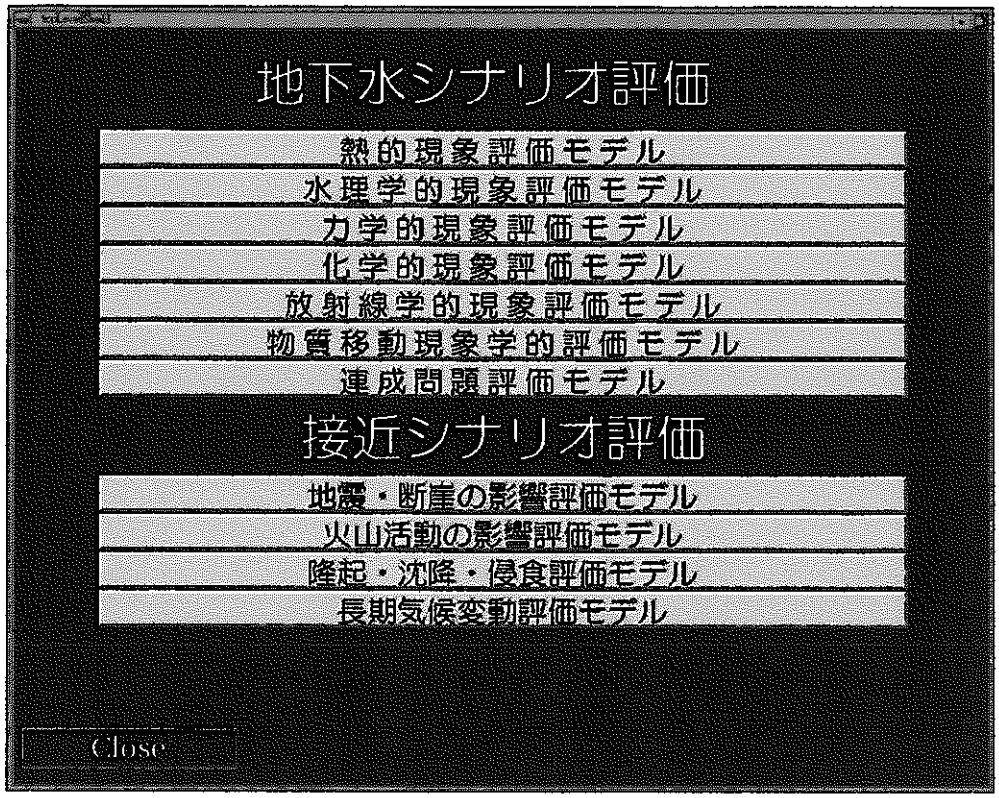

図９　データベース操作画面１

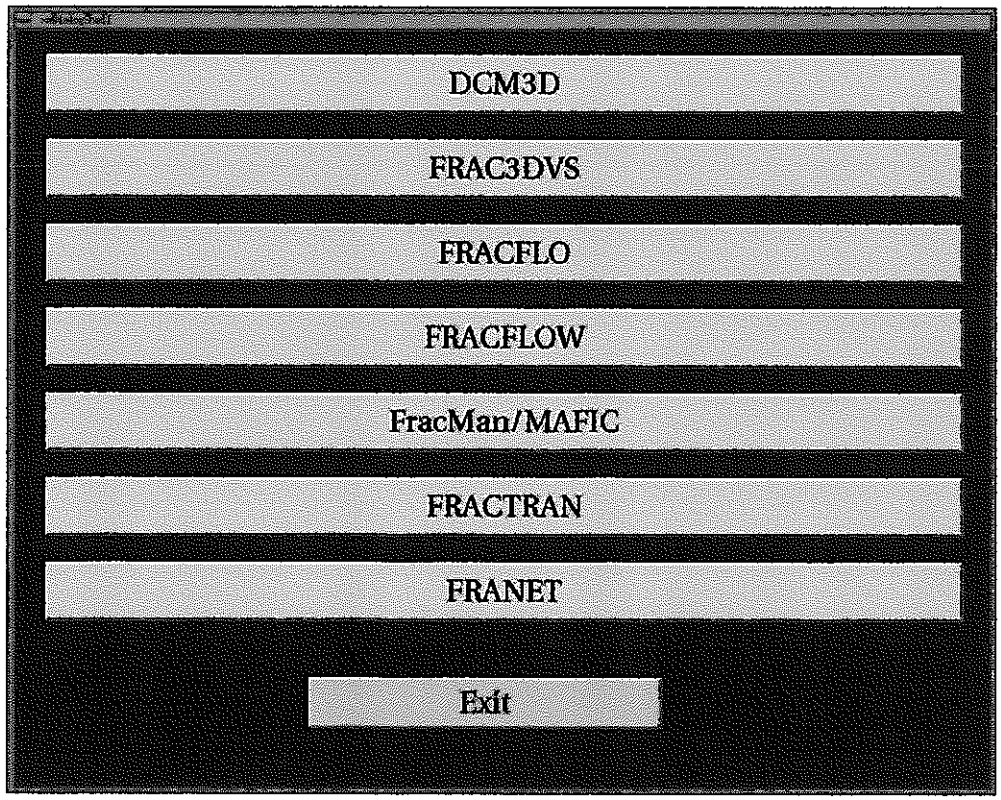

図１０　データベース操作画面２

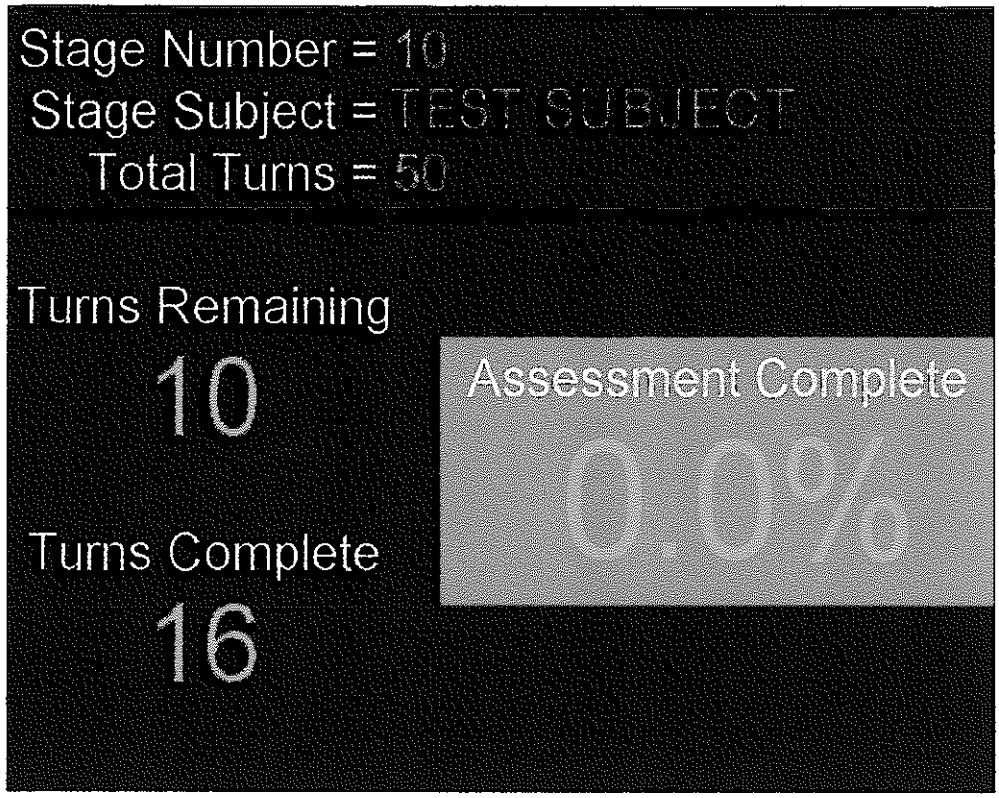

図１１　シミュレーション・パラメータ表示画面１

図１２　シミュレーション・パラメータ表示画面２

図１３　シミュレーション・パラメータ表示画面３

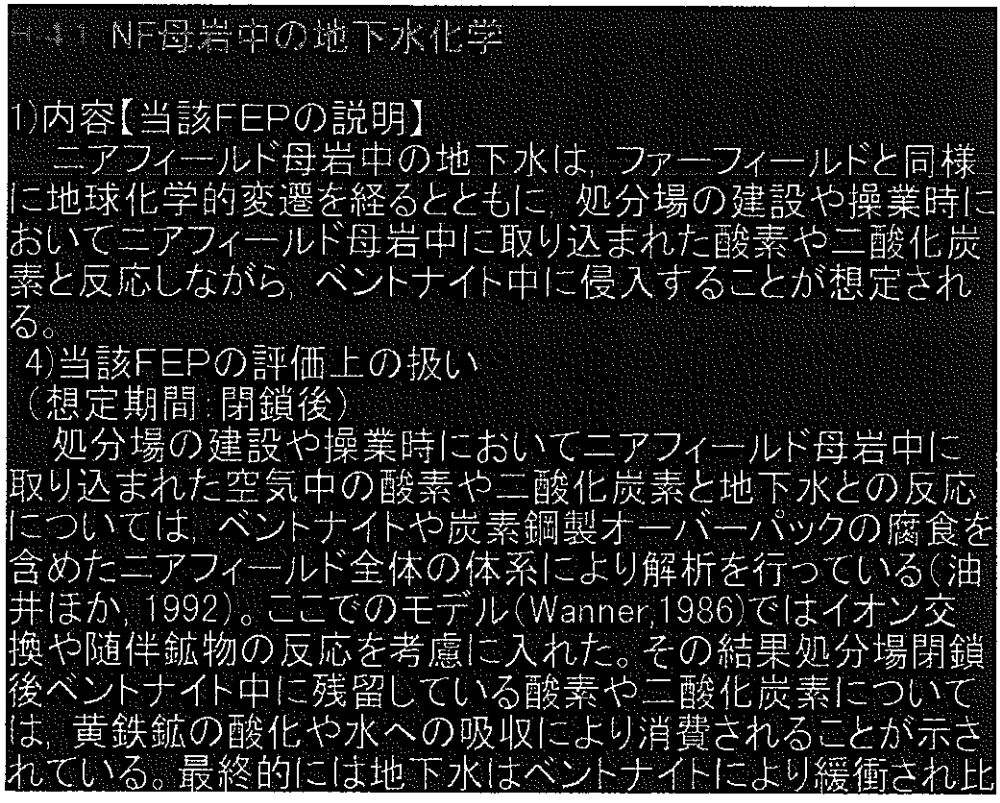

図１４　シミュレーション・パラメータ表示画面４

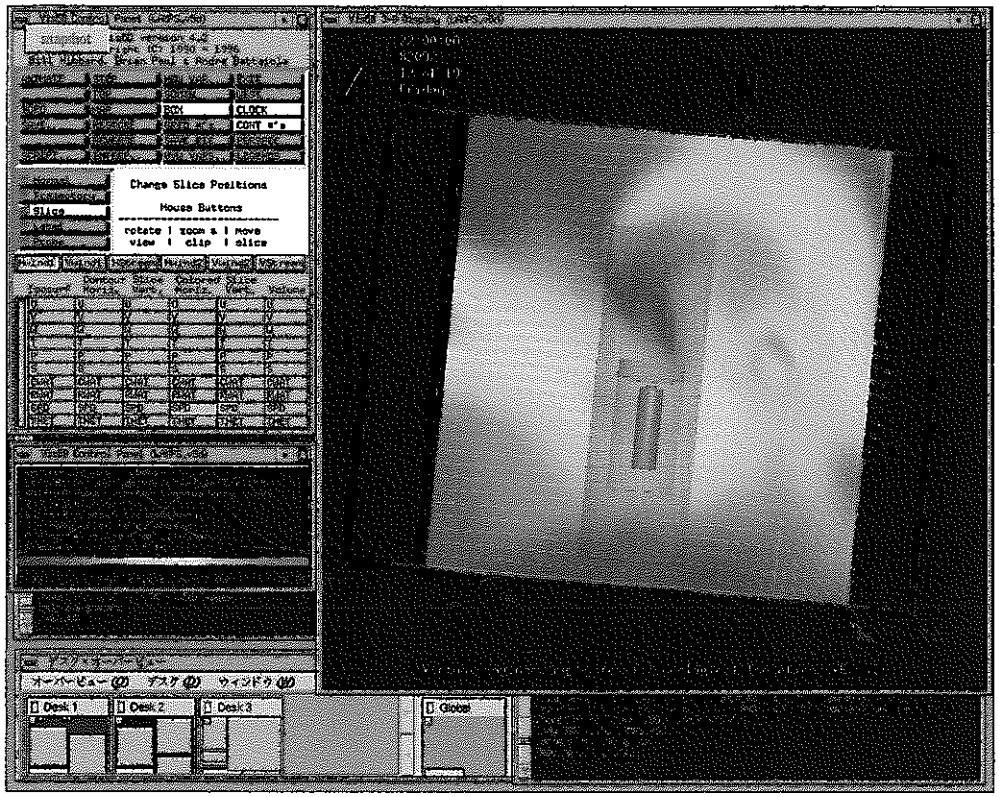

図１５　動画化シミュレーション表示画面１

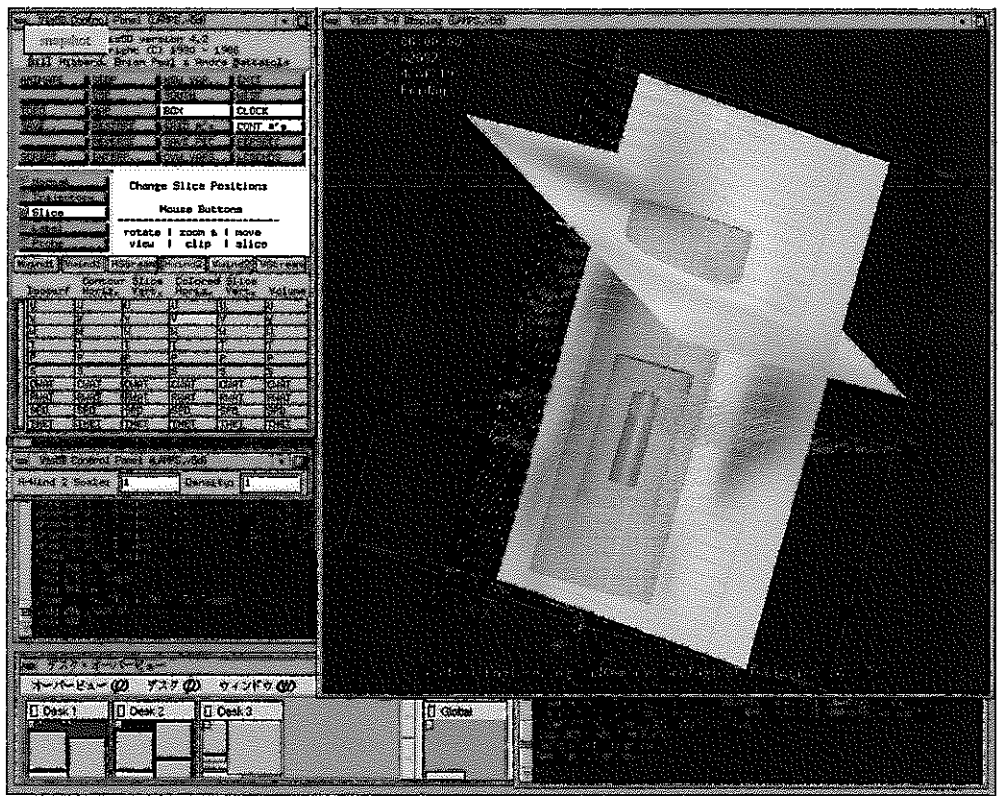

図１６　動画化シミュレーション表示画面２

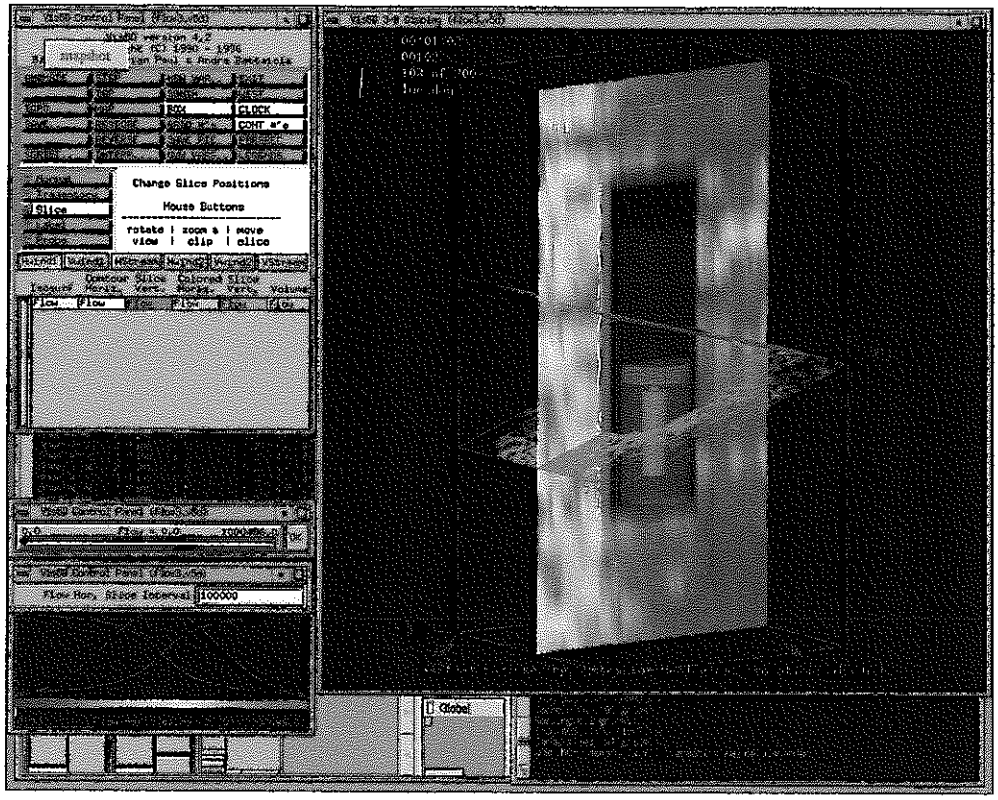

図１７　動画化シミュレーション表示画面３

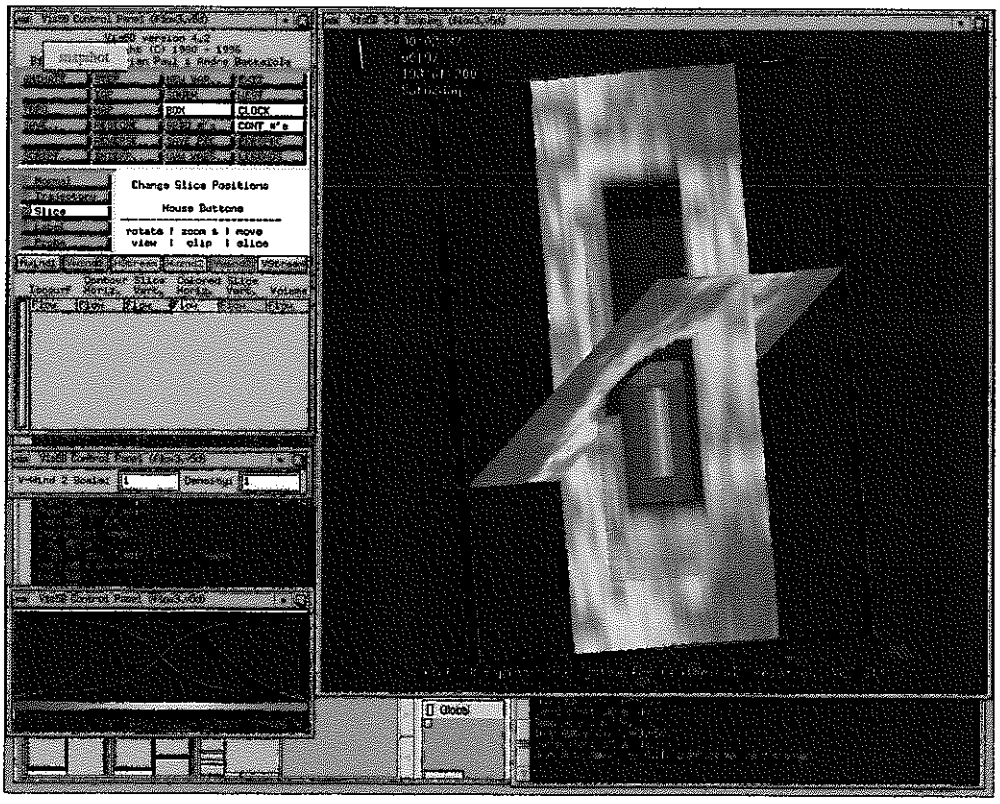

図１８　動画化シミュレーション表示画面４

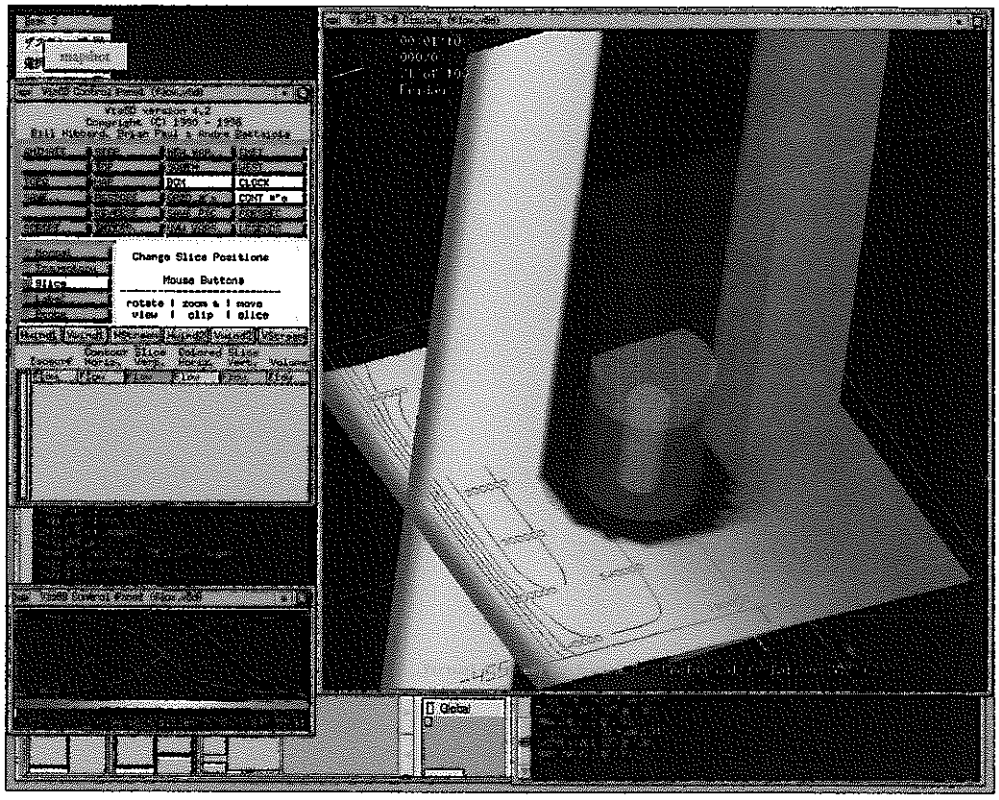

図１９　動画化シミュレーション表示画面５

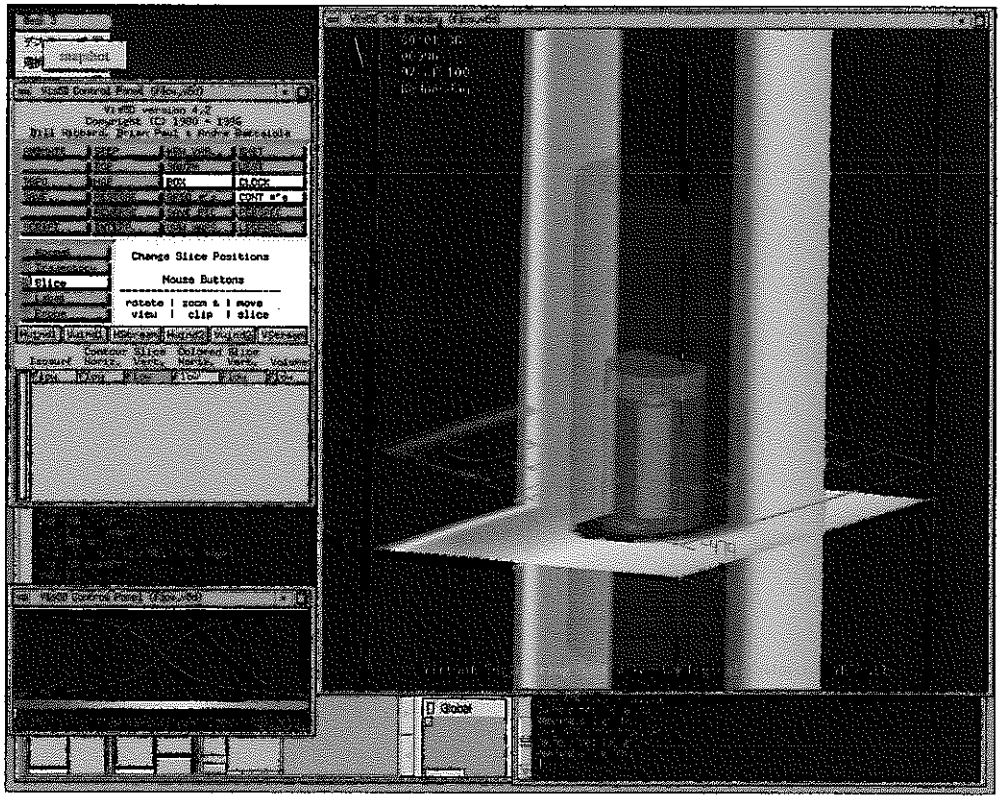

図２０　動画化シミュレーション表示画面６

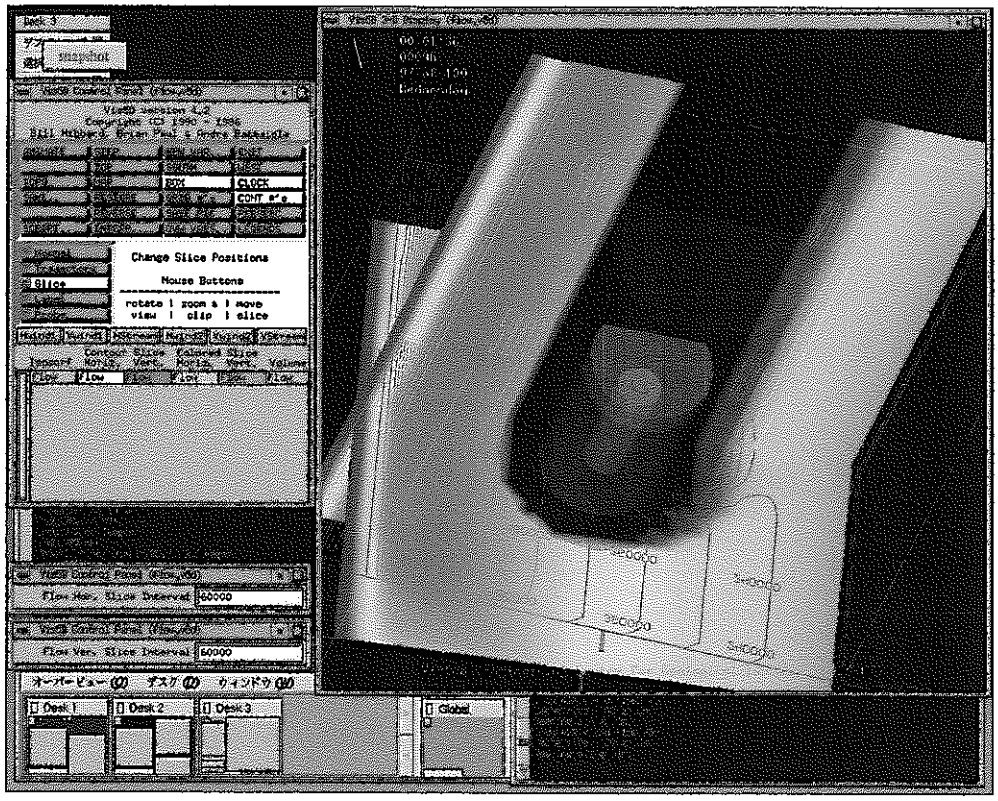

図２１　動画化シミュレーション表示画面７

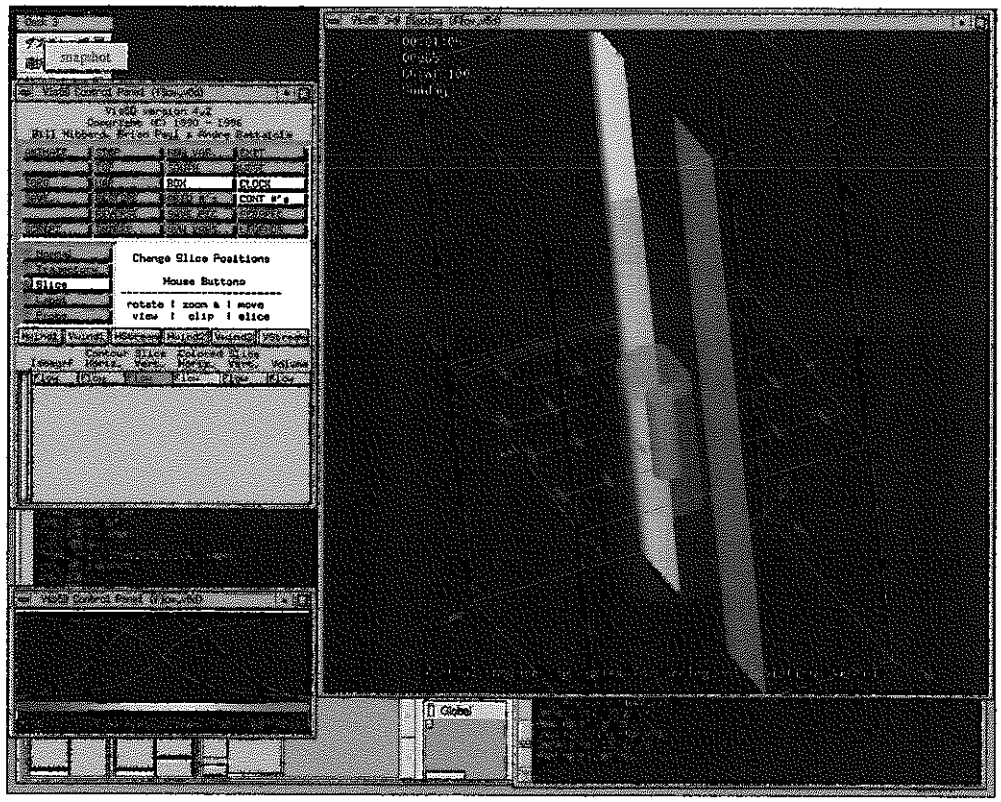

図２２　動画化シミュレーション表示画面８

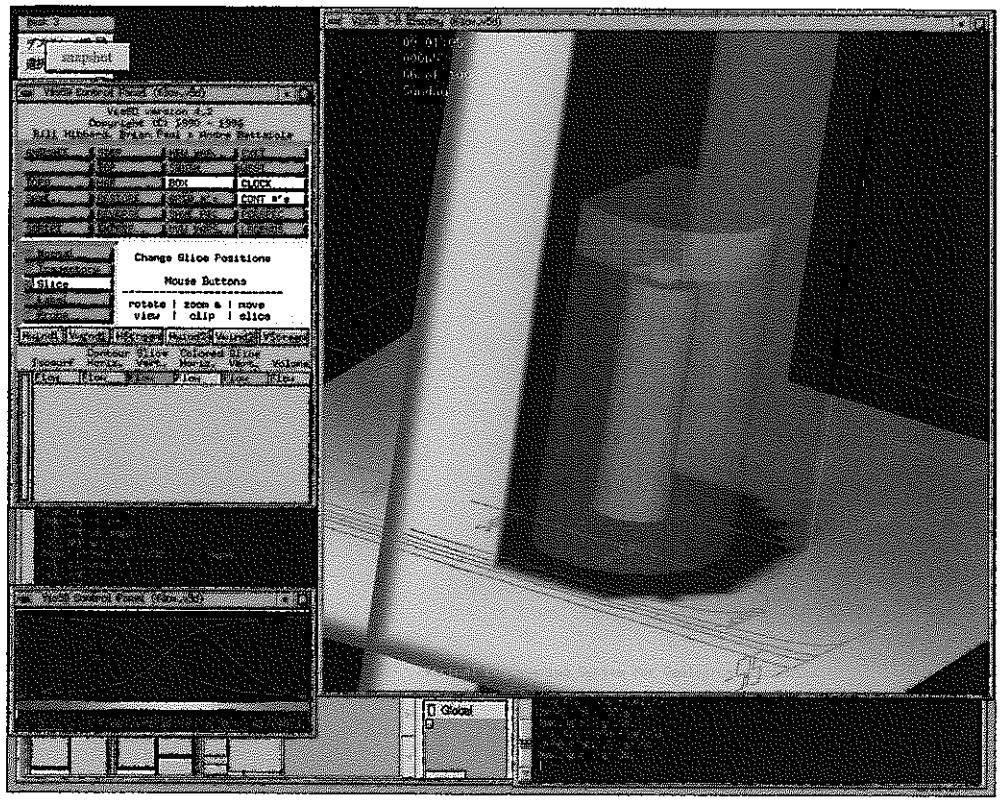

図２３　動画化シミュレーション表示画面９

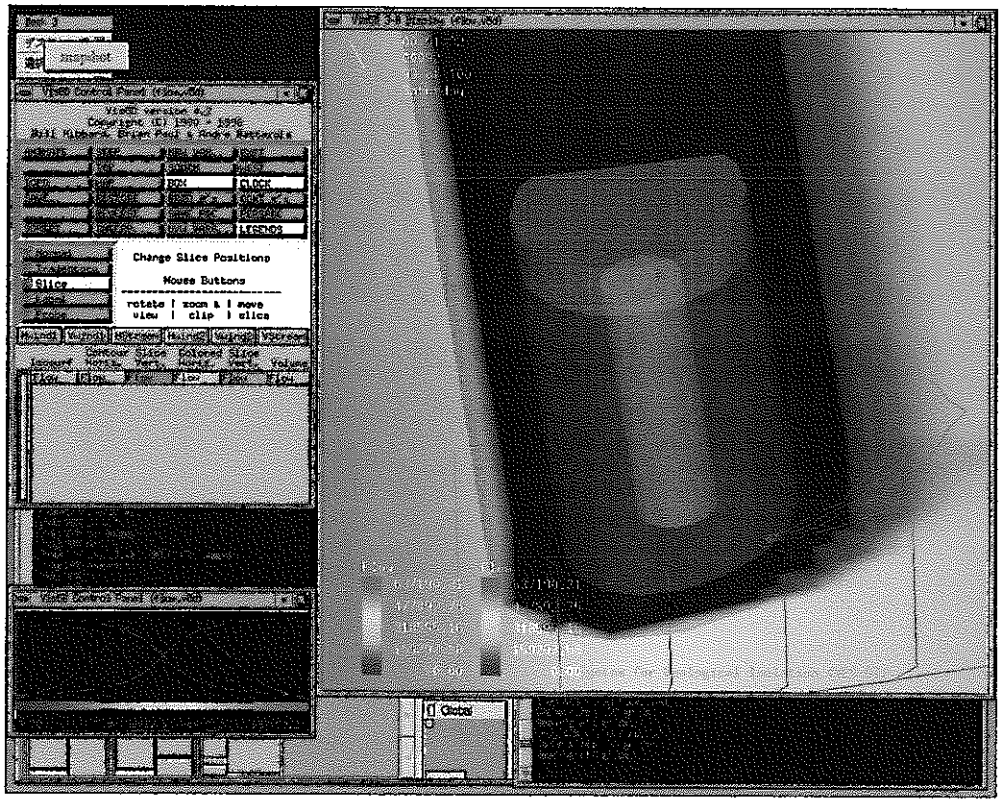

図２４　動画化シミュレーション表示画面１０

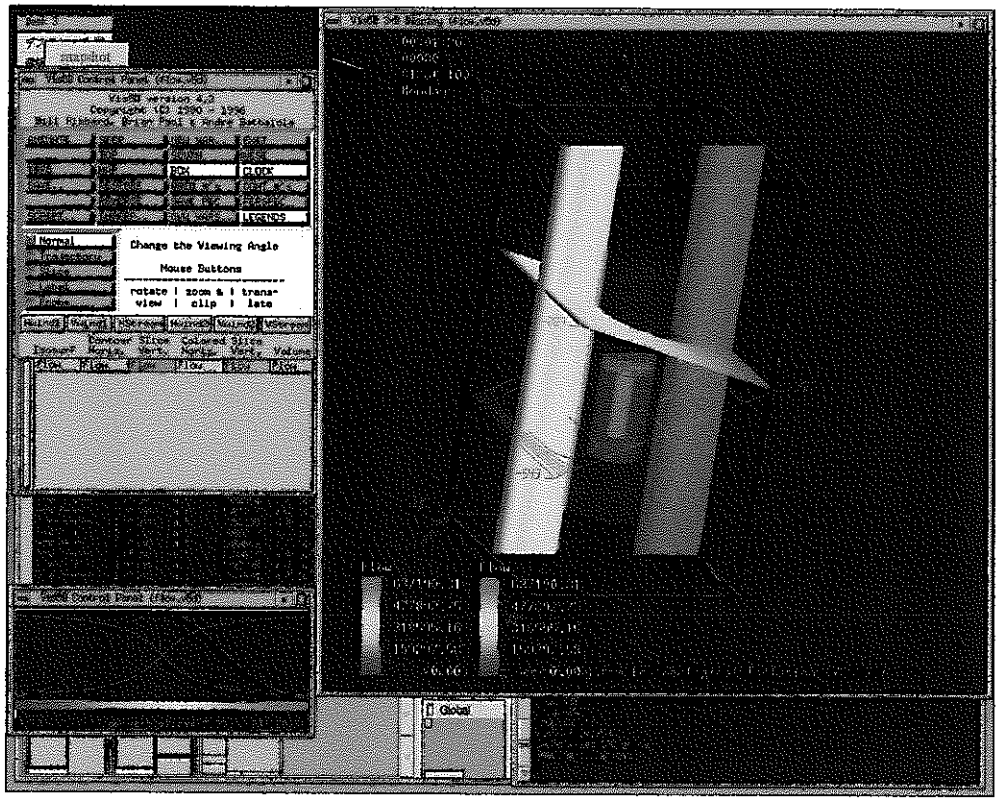

図２５　動画化シミュレーション表示画面１１

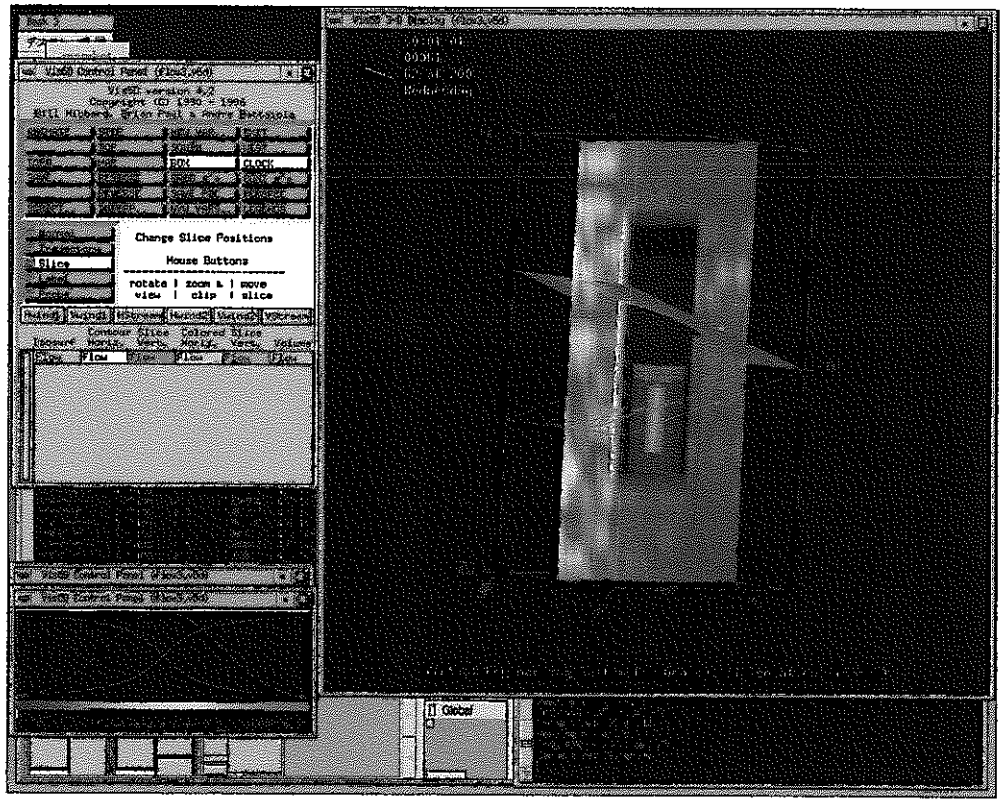

図２６　動画化シミュレーション表示画面１２

図２７　動画化シミュレーション表示画面１３

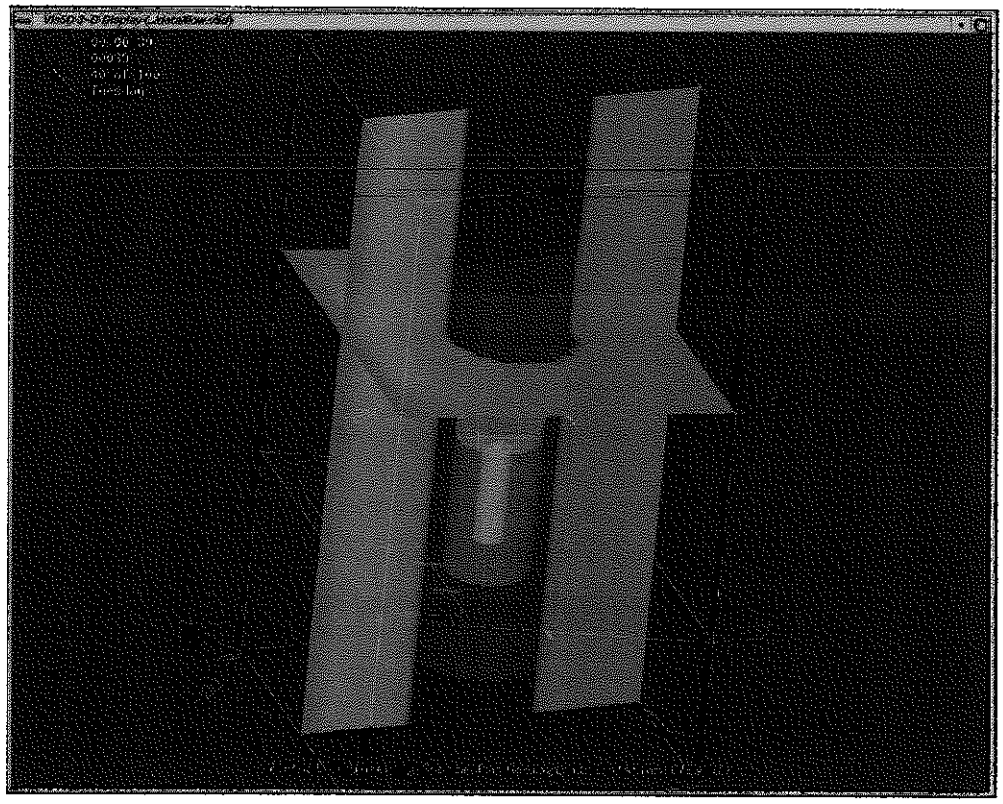

図２８　動画化シミュレーション表示画面１３

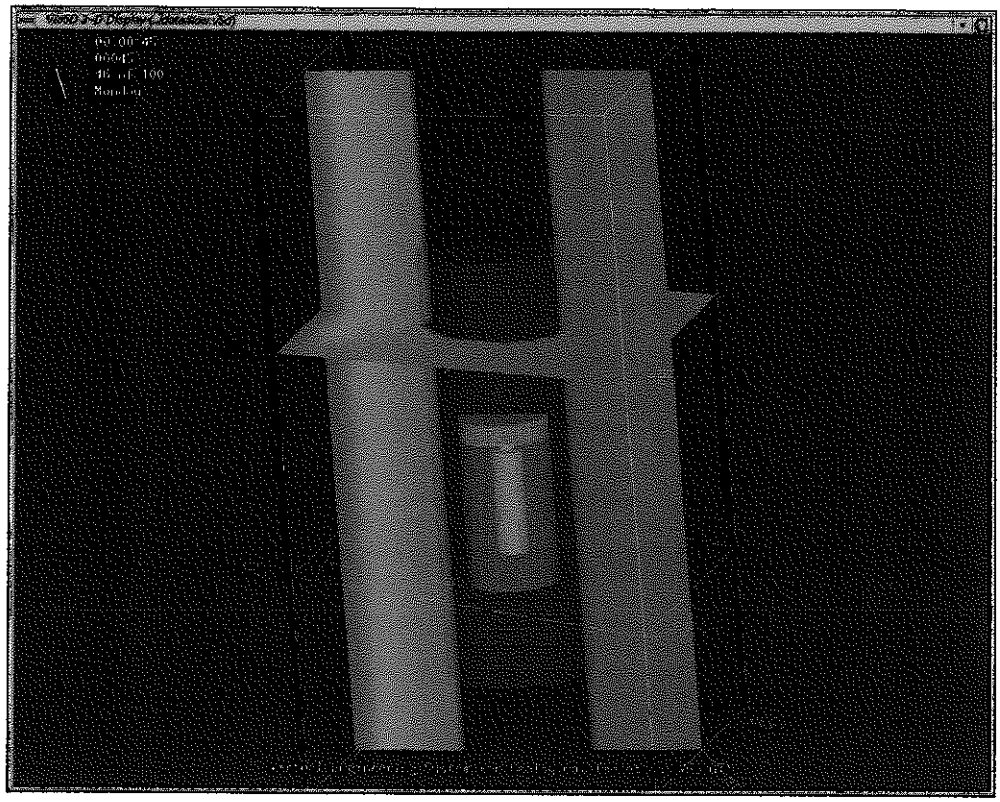

図２９　動画化シミュレーション表示画面１４

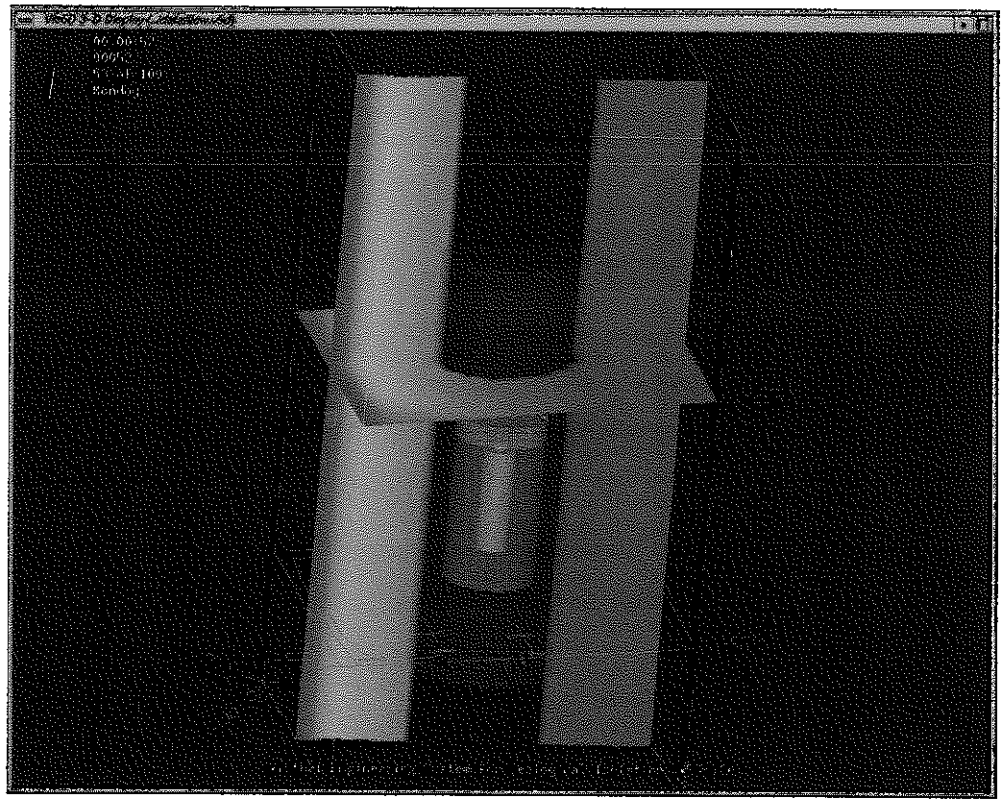

図３０　動画化シミュレーション表示画面１５

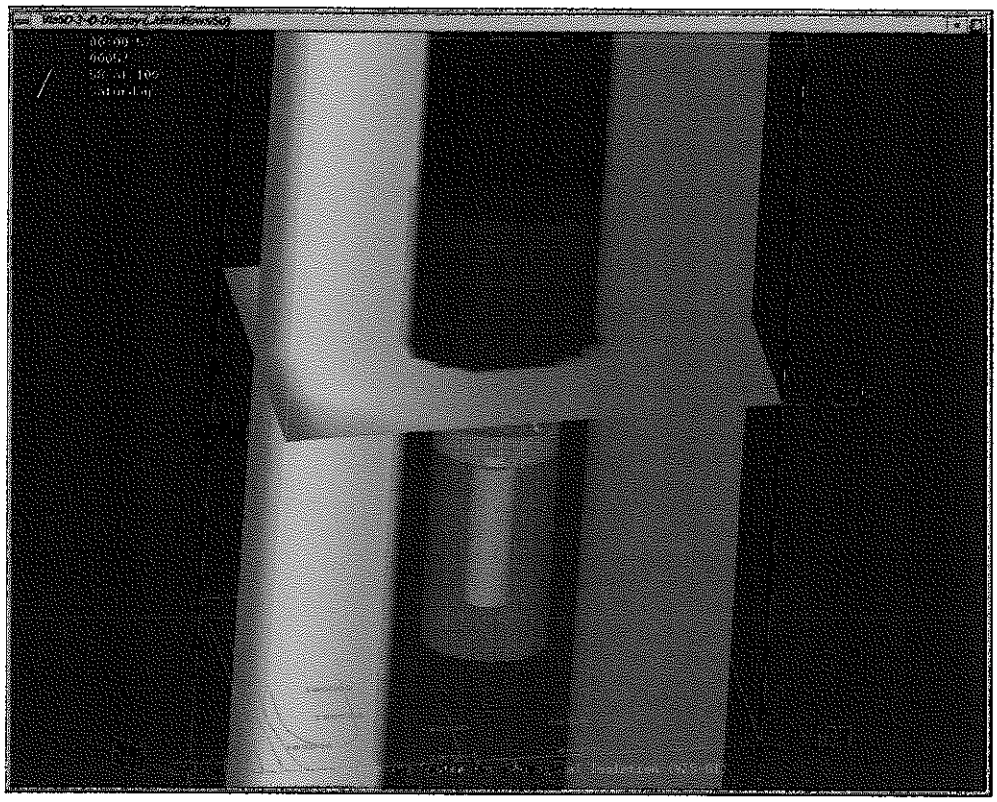

図３１　動画化シミュレーション表示画面１６

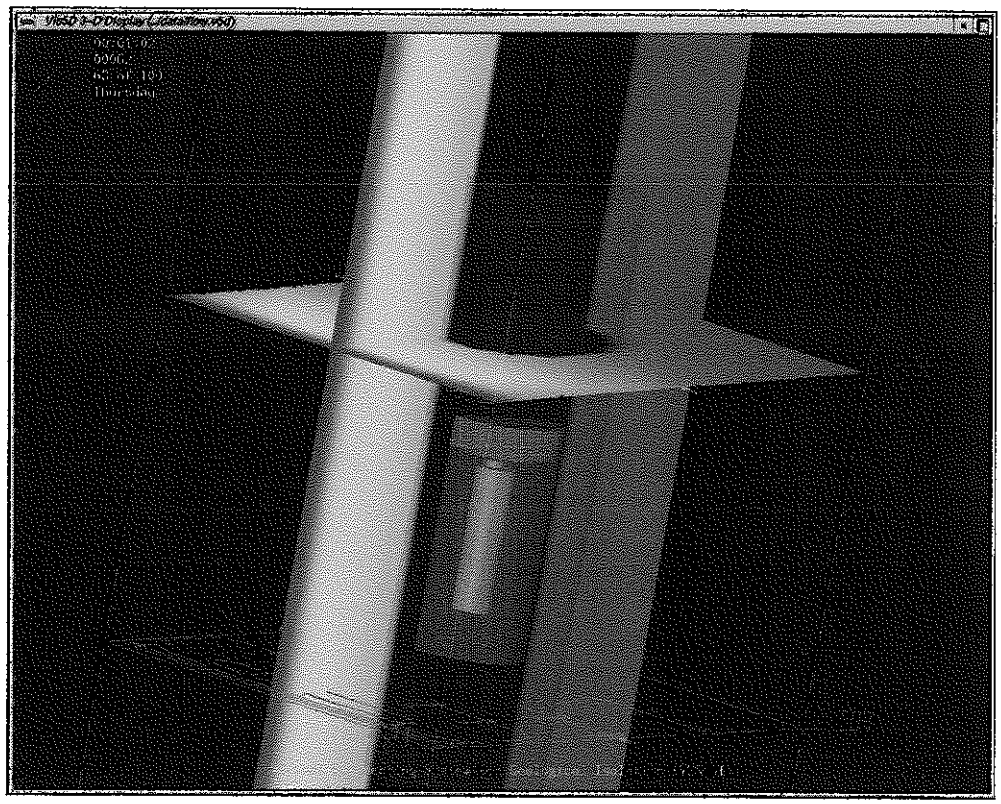

図３２　動画化シミュレーション表示画面１７

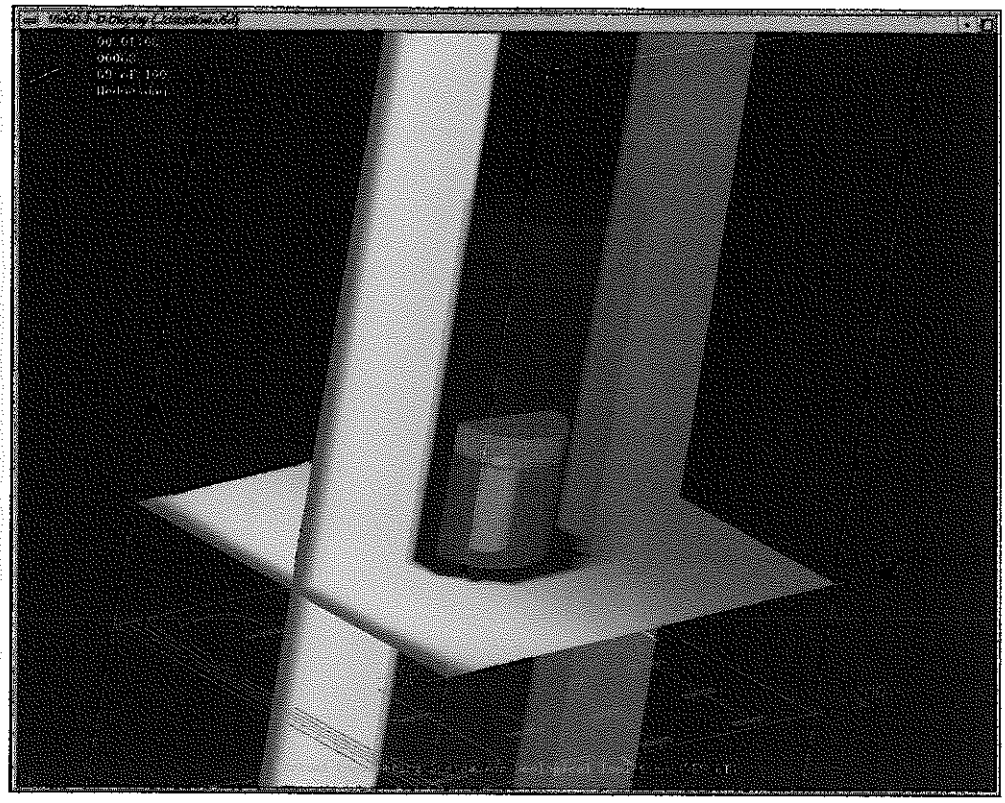

図３３　動画化シミュレーション表示画面１８

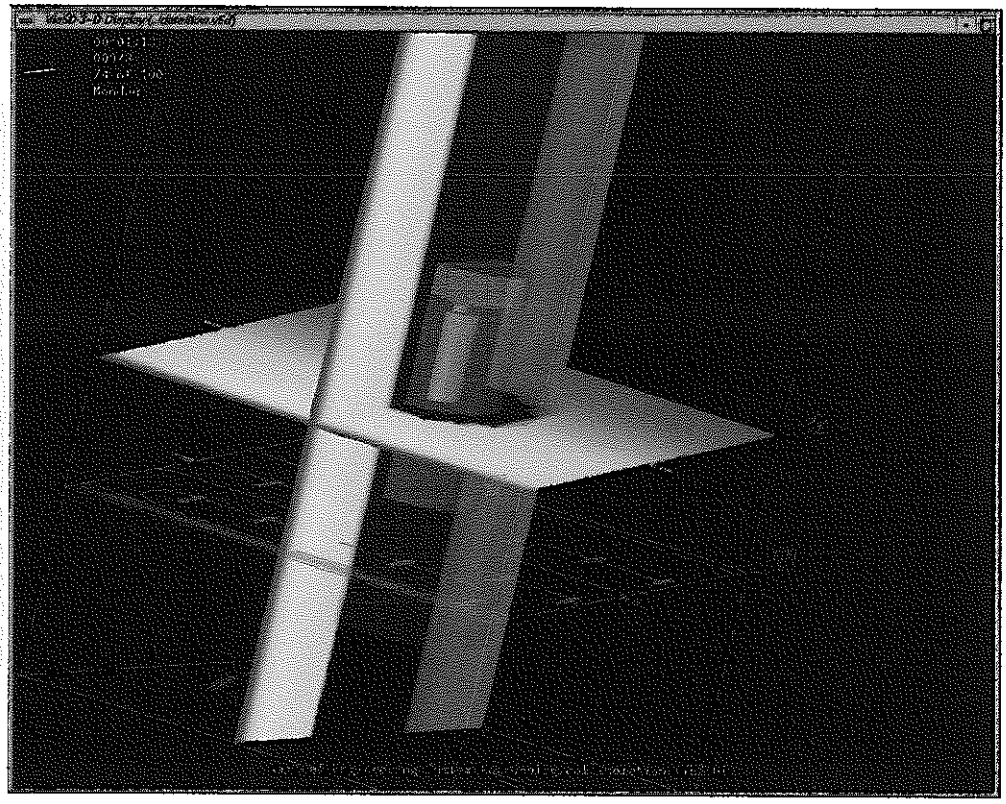

図３４　動画化シミュレーション表示画面１８

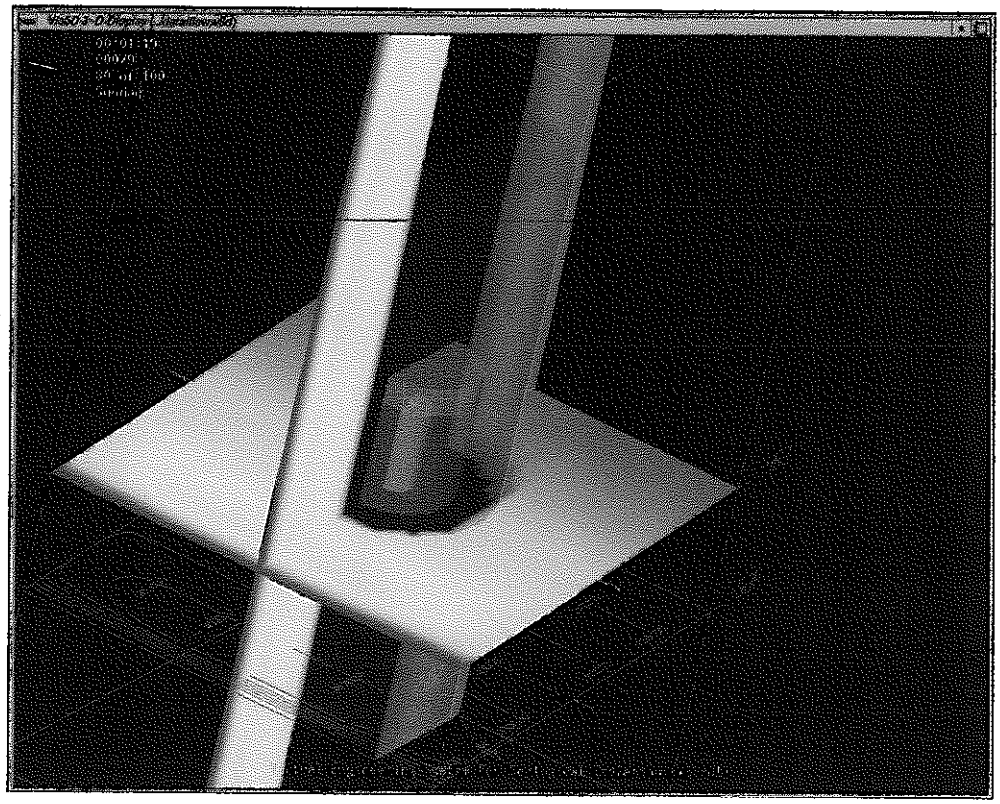

図３５　動画化シミュレーション表示画面１９

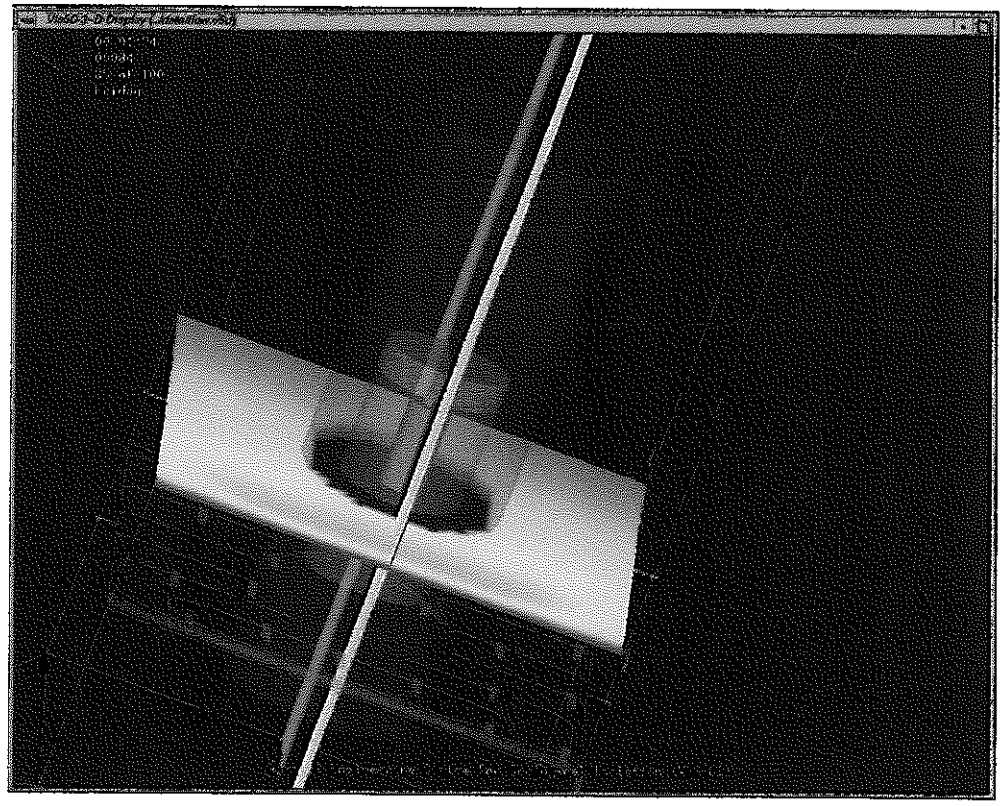

図３６　動画化シミュレーション表示画面１９

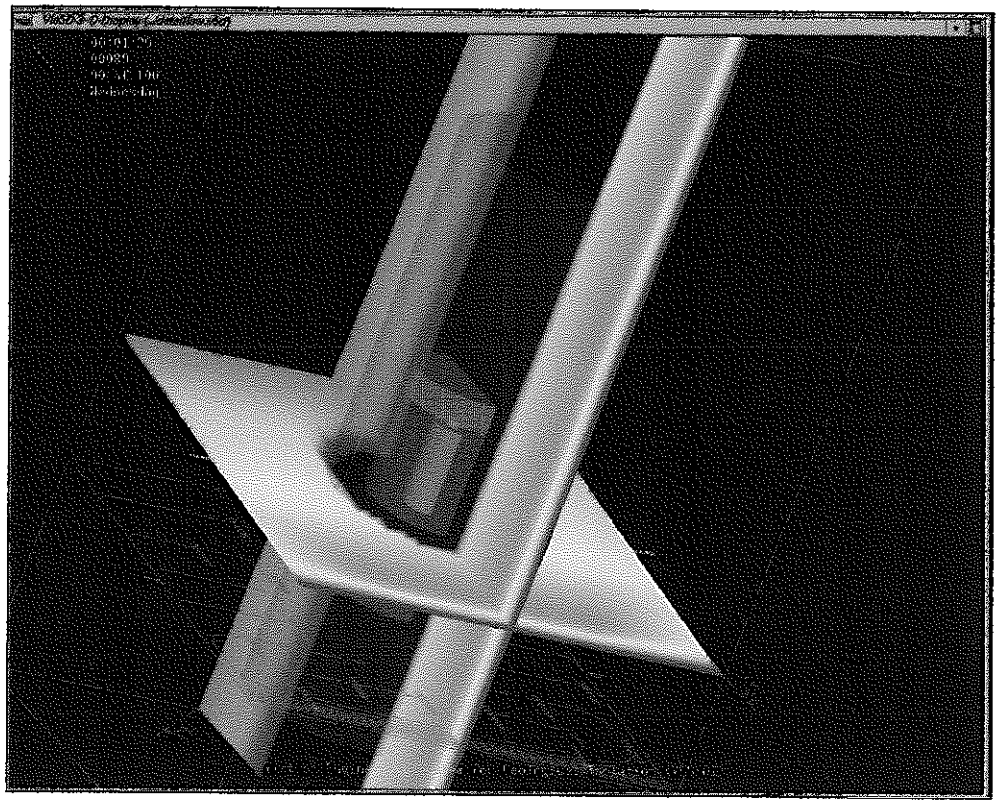

図３７　動画化シミュレーション表示画面２０

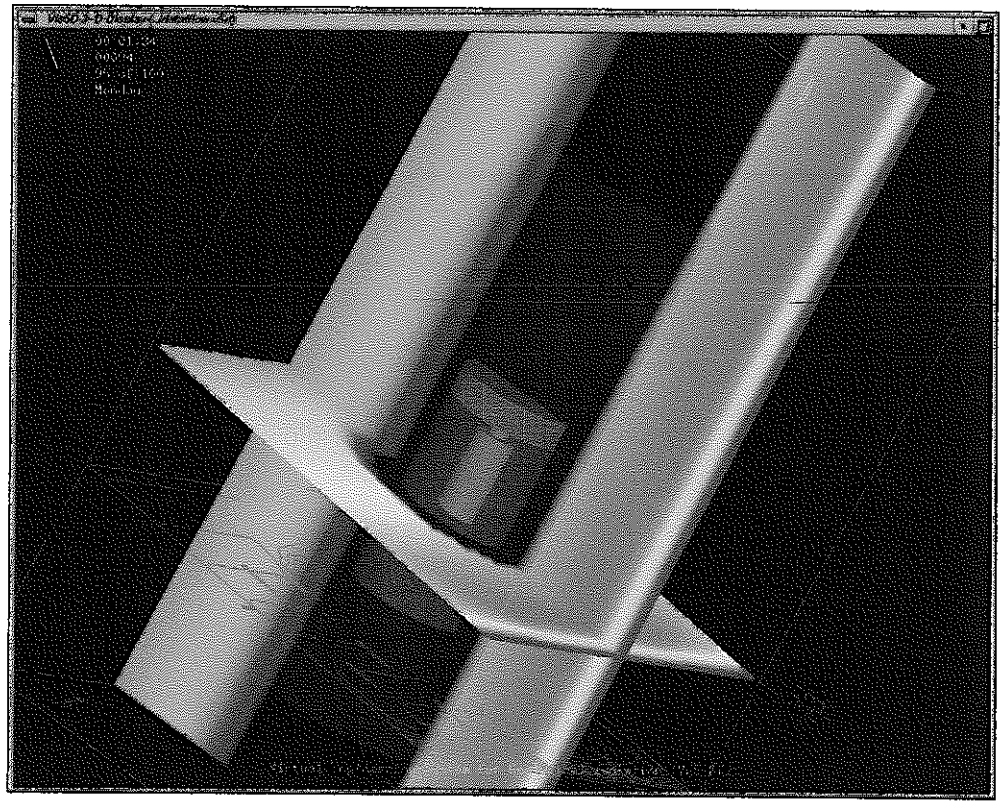

図３８　動画化シミュレーション表示画面２１

プロセス間通信

1　プロセス間通信

ネットワーク上の複数の計算機で動作している複数のプロセスが互いに通信しながら、協調して一つの仕事を成し遂げるものをネットワーク・アプリケーションという。つまり、ネットワーク・アプリケーションとは、ネットワーク上では計算機の処理単位であるプロセスとプロセスの通信をとおして行われ、そのプロセスとプロセスの間の通信をプロセス間通信という。

複数の計算機（ワークステーション、ＰＣ）を用いて並列処理させるためには、ネットワークを利用した計算機間の通信が必要となる。標準的に準備されているワークステーションにおけるプロセス間通信には、ソケット通信とリモートプロシジャコール（RPC）がある。RPCは、そのままでは並列処理が行えないので、ここでは、ソケットを用いてのプロセス間通信について記述する。

プロセス間通信を行うためには、通信するプロセス同士が通信路で結ばれていなければならない。このプロセスと通信路の接続点となるのがソケットである。

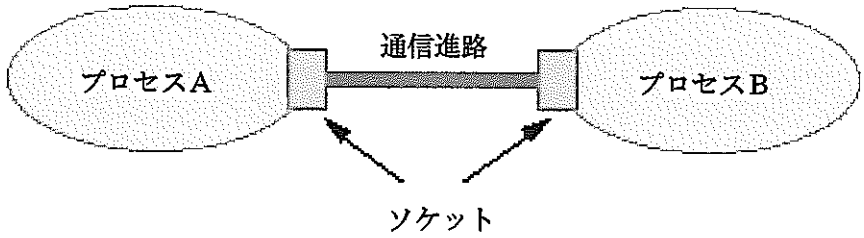

図１－１　プロセスとソケット

C言語には、ソケットを作ったり、ソケットを使用して通信を行ったりするためのライブラリが用意されており、そのライブラリで提供される関数群を使用してソケット・インターフェイスプログラムを作成することができる。

１．１　ソケットの接続概念

ソケットはワークステーション（UNIX）に標準に実装されているプロセス間通信のインターフェイスである。また、UNIXにはTCP/IP（Transmission Control Protocol / Internet Protocol）プロトコルが実装されている。TCP/IPはネットワークを介した計算機同士が相互に通信を行うための手法の１つであり、ソケットは、このTCP/IPプロトコルをUNIXシステム上で利用するために開発・実装された。そのことから、ソケットを利用することにより、同一計算機上のプロセスだけでなく、異なった計算機上のプロセスの間においても同じインターフェイスでプロセス間通信を行うことができる。

図１．１－１にネットワークを介したソケットによる接続概念を示す。

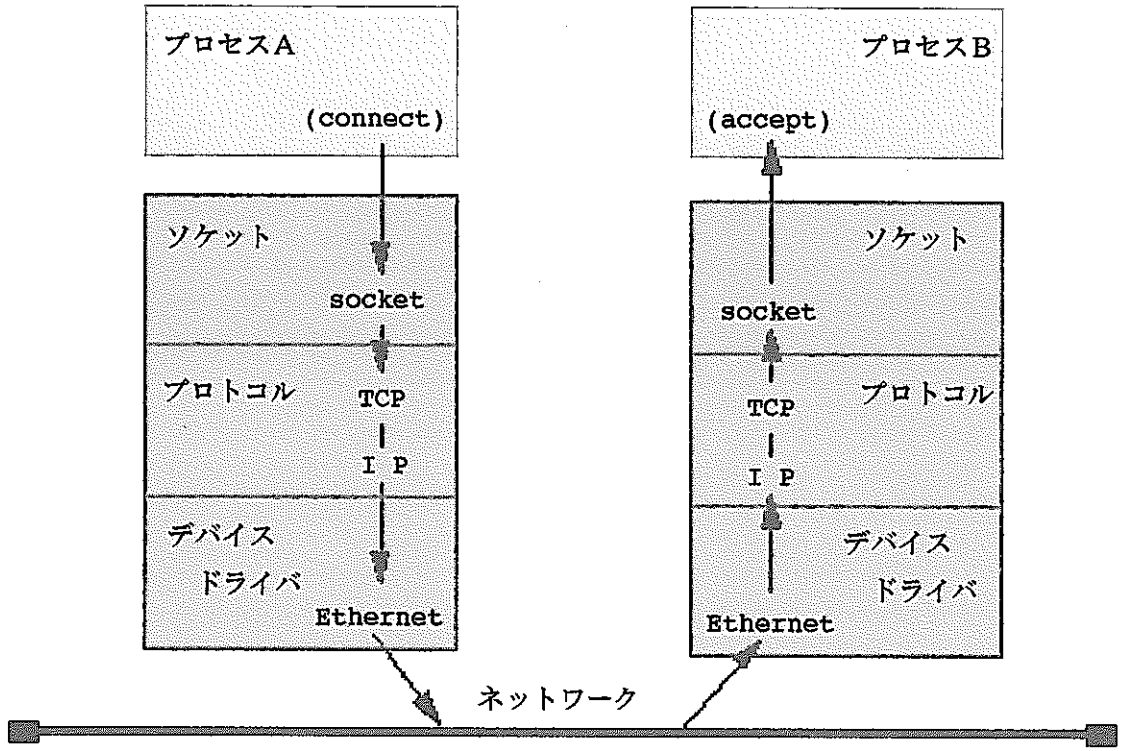

図１．１－１　ネットワークを介したソケットによる接続概念

ソケットによるプロセス間通信を行うには、当然送信側と受信する側の２つのプログラムを作成しなければならない。ソケットを作ったりソケットを使用してメッセージを送信したりするための特別のCライブラリで提供される関数が必要である。

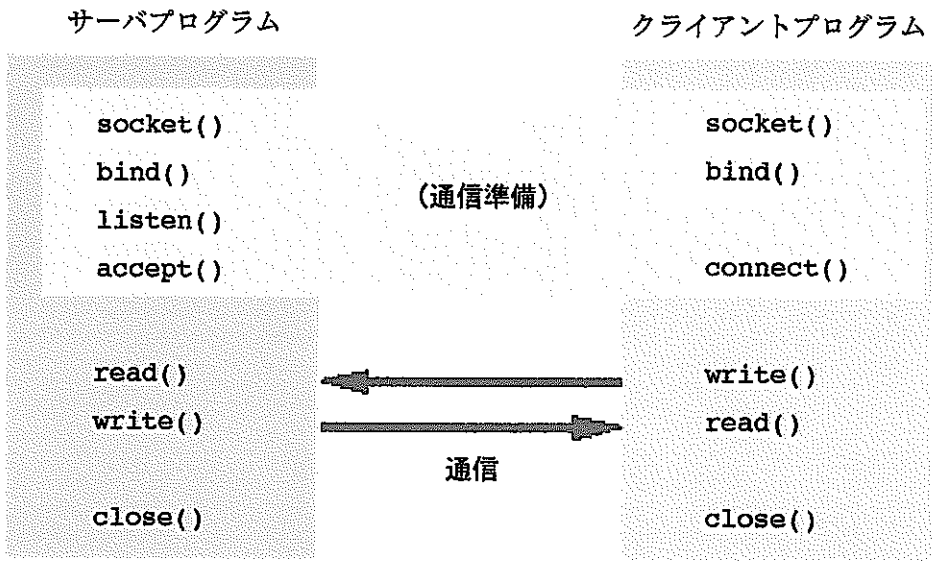

図１．１－２　ソケットプログラム関数

１．２　ソケットによるプロセス間通信手順

ソケットによるプロセス間通信手順を以下に示す。

サーバプログラムは、socketシステムコールによりソケット（socketインターフェイスを使用するためのシステム内に確保されるバッファ）を生成する。

サーバプログラムは、bindシステムコールにより接続相手を特定するためにソケットに名前を付ける。

サーバプログラムは、listenシステムコールを発行して接続受け入れ準備が整ったことをシステムに知らせる

クライアントプログラムは、connectシステムコールを発行してサーバへ接続する。

サーバプログラムは、acceptシステムコールを発行してクライアントの接続を受け入れる。

接続された両プロセス間では、その後readおよびwriteを使用することでメッセージのやり取りを行うことができる。

図１．２－１にソケットによる基本通信手順を示す。

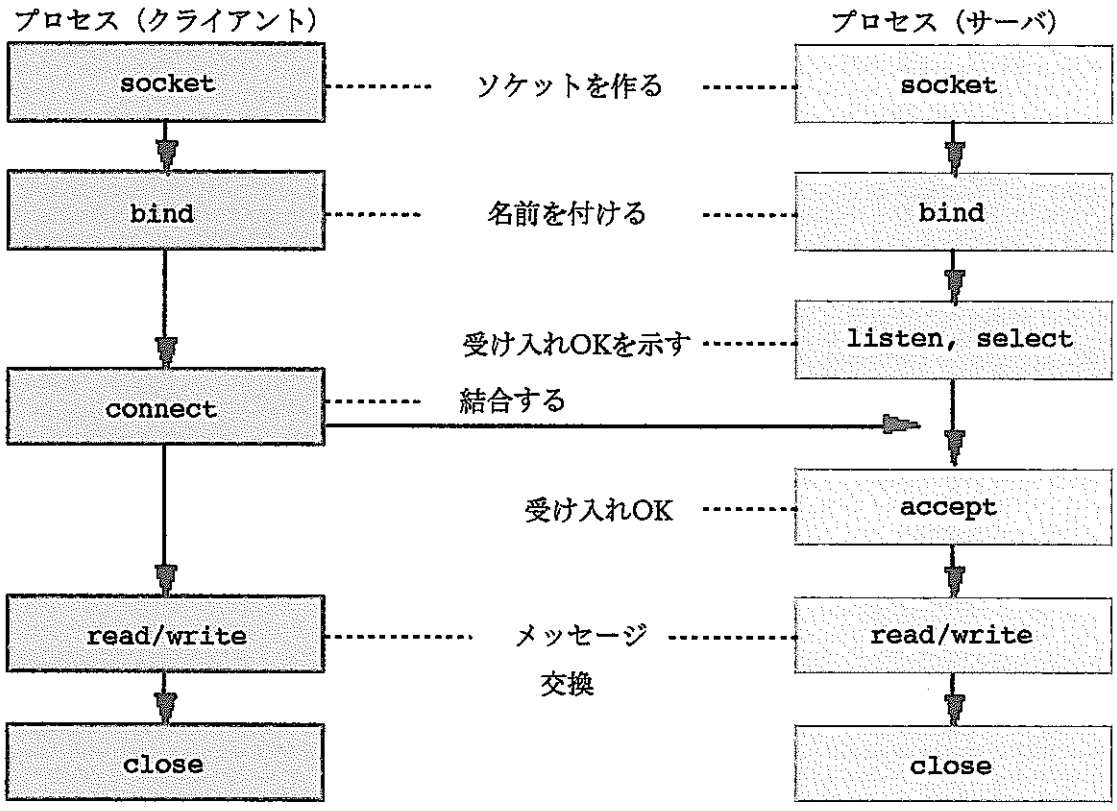

図１．２－１　ソケットの基本通信手順

ソケットによるプロセス間通信における主な利点として次のようなものが上げられる。

> **ネットワークに接続される計算機間でのデータ通信方法が標準であること。**
> **ネットワークに接続される計算機の機種が限定されないこと。**
> **ネットワークに増設される計算機に容易に対応可能なこと。**
> **データ通信（受送信）がオートマチカルなこと。**

ソケットは、UNIXドメインのソケットとINETドメインのソケットと呼ばれる2種類ある。

UNIXドメインのソケット

同じ計算機（ローカルホスト）内で動作しているプロセス同士で通信を行うためのソケット。他の計算機上のプロセスとは通信できない。

INETドメインのソケット

ネットワークで接続されていれば、離れた計算機上のプロセス同士で通信を行うことができる。

１．３　ソケット関連ライブラリ関数

ソケットは、C言語のサブルーチン形式（システム・コール）で提供されるプロセス間通信機能であり、ローカルホストのプロセス間およびローカルホストとリモートホストのプロセス間を、信頼性の高い仮想回線で結ぶEND to ENDのプロトコルである。仮想回線は複数張ることが可能で、どのプロセスと通信しているかをポート番号で識別している。

サブルーチン形式で提供されることから、制作するプログラム内に依存させることが可能で、運用に合致したデータ通信や操作員の関知する必要のないデータ通信をオートマチカルに行うことができる。

（１）socketシステムコール（ソケットの生成）

ソケットを通信を行うには、最初に通信を行うすべてのプロセスがそれぞれsocket()システムコールによりソケットを用意する必要がある。socketシステムコールは、プロセス間通信を行うためのソケットディスクリプタを得るために使用される。

socketには3つの引数が有り、それぞれドメイン、ソケットタイプ、そしてプロトコルである。このシステムコールで使用される様々な定数は、インクルードファイルsys/socket.hの中で定義されている。

```
int socket( int domain, int type, int protocol )
```

ドメイン引数

オペレーティング・システム（OS）に対して、どのドメインのものとしてアドレスが解釈されるべきかを指定する。同じ計算機上でのプロセス間通信では、UNIXドメインを、ネットワーク上の計算機とのプロセス間通信では、INETドメインを指定する。

ソケットタイプ引数

ソケットタイプを表わす引数は、そのソケットと使われるべき通信チャネルの型を指定する。プロセス間通信では、利用可能な通信チャネルの型がいくつかあるが、一般に必要なものは次の2つである。

SOCK_STREAM　この型の接続は通常、仮想回線と呼ばれる。これは、データが送り出された順番どうりに到達することを確実に保証する連続バイトストリームである。回線が確保されるまでは、どのデータも送ることができない。そして、その回線は通信が完了するまで残る。

SOCK_DGRAM　この型の接続は、データグラムと呼ばれる個々の情報パケットを送るのに使用される。データグラムは先方に送り出した順番に届くという保証はない。

プロトコル引数

プロトコル引数により、プログラマは通信チャネルを実現するのに使用するプロトコルを選択することができる。プロトコル引数に0が与えられると、オペレーティング・システムが自動的に適当なプロトコルを選択する。

socketはソケットがプロセスに接続された後に、readやwrite、その他のプロセス間通信システムコールに使用するためのファイルディスクリプタとなる整数値を返す。ソケットが作成できない場合には－1が返され、エラーの原因がerrnoにセットされる。

（2）bindシステムコール（ソケットアドレスの指定）

bindシステムコールは、ソケットに名前を付けるのに使用する。ソケットに、ある名前を付け終えるまでは、クライアントプログラムがそのソケットを通信対象として指定することはできない。bindには3つの引数がある。名前を付けるソケット、ソケットの名前へのポインタそして名前の長さである。

処理が成功すると0を返し、失敗すると－1を返す。エラーの原因がerrnoにセットされる。bindの第2引数は、sockaddrと総称される型を持った構造体であり、UNIXドメインではsockaddr_un型の構造体で、インクルードファイルsys/un.h中で定義されている。

```
struct sockaddr_un {
        short sun_family;    /* AF_UNIX    */
        char sun_path[108]; /* path name */
        };
```

sun_familyメンバーは、AF_UNIXにセットする。これは指定したいアドレスのアドレッシングファミリー（命名の方式）を示す。sun_pathは、これから使用するソケットのパス名を入れる。sun_path中

に指定された名前がソケットにbindされた時に、この名前のファイルが実際に生成される。インターネット（INET）ドメインでは、sockaddr_in型の構造体でインクルードファイルnetinet/in.h中で定義されている。

```
struct sockaddr_in {
        short sin_family;
        u_short sin_port;
        struct  in_addr  sin_addr;
        char sin_zero[8];
        }
```

sin_familyにはAF_INETが与えられる。これはインターネットドメインを指定するもので、sin_portメンバーには、接続されるポート番号（ネットワークバイトオーダで）を含まれなければならない。sin_addrメンバーは、ポートが存在する計算機のネットワーク番号を（ネットワークバイトオーダで）含んでいなければならない。

（3）listenシステムコール（接続準備）

サーバはlisten()システムコールにより、接続のための準備を行う。

```
int listen( int s, int backlog )
```

sは、socketの返り値で、backlogは接続要求をいくつ受け付けるかを指定する（通常のシステムではbacklogの値の上限を5に定めている）。。listenシステムコールは、指定したソケットへの接続要求が他のプロセスから自分のプログラムにくるはずだと、オペレーティングシステムに通知するためにサーバによって使用される。あるソケットを監視し、接続要求の通知を待っているサーバプログラムが存在しない場合は、そのソケットを指すアドレスに対する接続要求はすべて拒否されまう。listenシステムコールには、接続要求の発生をモニタしようとするソケットと、整数backlogの2つの引数がある。backlogは、そのソケットにいくつまでの接続要求を受け入れるかを示す値で、これにより、プログラムは、1つのソケットに対して一度に2つ以上のプログラムを接続できるようになる。オペレーティングシステムは、backlogで指定された数の要求を待ち行列に蓄えて、それらを1つずつプログラムに渡す。サーバがbacklogの容量がいっぱいになっても、その後に入ってくる接続要求は拒否されない。オペレーティングシステムが接続を要求するデータパケットを切り捨ててしまう。接続を拒否するのでなく、接続にタイムアウトの可能性を与えることを目的としている。クライアントがサーバがビジーで接続を拒否したか、サーバがダウンしているかの区別を付けるためである。

（4）connectシステムコール（接続要求）

クライアントがサーバに対してconnect()システムコールにより接続要求を出し、サーバはaccept()システムコールにより接続を受け入れる。

```
int connect( int s, struct sockaddr *name, int namelen )
```

引数は、bind()と同じである。connectは、クライアントプロセスがサーバとの対話を実現するために使用される。connectには3つの引数がある。オープンされたソケットディスクリプタ、接続相手や適切なドメインを含んでいるsockaddr構造体へのポインタ、そしてそのアドレスの長さを示す整数である。処理が成功すると0を返し、失敗すると－1を返す。エラーの原因がerrnoにセットされる。

（5）acceptシステムコール（接続）

サーバ側では、クライアントのconnect要求をacceptによって受け入れる。

```
int accept( int s, struct sockaddr *addr, int *addrlen )
```

sは、listenの時と同じでsocket()の返り値を渡す。nameには空の構造体のポインタを返す。accept()が成功すると、システムが自動的に書き込む。接続要求が完了するとaccept()はクライアントと接続された新しいソケットディスクリプタを返す。サーバはクライアントとの通信にこの新しいディスクリプタを用い、socket()コールの時に返された古いディスクリプタは必要なくなる。acceptシステムコールは、サーバがソケット上の接続を受け入れるのに使用される。受け入れ側のソケットはlistenによって、すでにモニタが開始されていなければならない。acceptが呼び出されると、システムコールはクライアントから接続要求が入ってくるまでプロセスをブロックする。接続要求が届くとacceptはクライアントプロセスに接続された新しいソケットディスクリプタを返す。このようにして、サーバはクライアントとの対話のために新しいディスクリプタを使う一方で、古いソケット（既知のアドレスのつながれたソケット）への接続を受け入れ続けることができる。

（6）read, writeシステムコール（通信）

接続が完了した後は、write()、read()システムコールにより通信を行なうことができる。

```
ssize_t write( int d, const void *buf, size_t nbytes )
```

sはソケットディスクリプタ、bufは送信したいデータである。

```
ssize_t read( int d, void *buf, size_t nbytes )
```

sはソケットディスクリプタ、bufは受信したデータを置くバッファである。
readシステムコールには3つの引数があり、読みだし用にオープンしたファイルディスクリプタと、

データを格納するバッファへのポインタ、そして読みだすバイト数を表す整数である。成功すると実際に読みだしたバイト数を返し、失敗すると－１を返す。ファイルの最後に達すると０を返す。writeシステムコールにも３つの引数があり、書きだし用にオープンしたファイルディスクリプタと、データを書き込むバッファへのポインタ、そして書き込むバイト数を表す整数である。成功すると実際に書き込まれたバイト数を返し、失敗すると－１を返す。

１．４　マルチクライアント・プロセス間通信プログラム

ここでは、前述したソケットの関数を用いてマルチクライアント・プロセス間通信プログラム作成例を示す。複数のクライアントを同時に扱うことのできるサーバプログラムとINETドメインのクライアントプログラムである。サーバプログラムをmserver.c、クライアントプログラムをclient.cとした。

サーバプログラム例

```
/*
 * server.c - Example of Multi Client Server
 */

#include < stdio.h >
#include < string.h >
#include < sys/types.h >
#include < sys/socket.h >
#include < netinet/in.h >
#include < sys/time.h >  /* selectシステムコール */
#include < unistd.h >

#define PORT 8765     /* サーバーが使うポート番号 */
#define SOCK_MAX 5    /* 最大ソケット数をここで決めた (最大32)*/
#define UNUSED (-1)

char *Modification(char *str, int length);

int main()
{
  int s[SOCK_MAX + 1]; /* array of socket descriptors */
  int max = 0;      /* max num of used socket */
  int n = 0;
  int len;
  fd_set readfds;
  int clilen;          /* client length */
  struct sockaddr_in saddr; /* サーバ側(自分)のアドレス情報 */
  struct sockaddr_in caddr; /* クライアント側のアドレス情報を入れるところ */
  char str[1024];       /* string buffer */
  int i, j;           /* loop counter */
  int msglen;
  /*
   * ソケットを作る。
   * このソケットはINETドメインでストリーム型(コネクション型)。
   */
  if ((s[0] = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
    perror("socket");
    exit(1);
  }
  /*
   * saddrの中身を0にしておかないと、bind()でエラーが起こることがある
   */
```

```
bzero((char *)&saddr, sizeof(saddr));

/* ソケットに名前をつける bind() */
saddr.sin_family = AF_INET;
saddr.sin_addr.s_addr = INADDR_ANY;
saddr.sin_port = htons(PORT); /* 定数PORTは#defineしたもの */

if ((bind(s[0], (struct sockaddr *)&saddr, sizeof(saddr))) == -1) {
  perror("bind");
  exit(1);
}

/* 接続されるのを待つ listen() */
/* 第2引数の値を大きくする */
if ((listen(s[0], SOCK_MAX)) == -1) {
  perror("listen");
  exit(1);
}
max = 1;  /* s[0] is ready */

/* メインループ */
while (1) {
  FD_ZERO(&readfds);  /* fdsの初期化 */
  printf("max: %d¥n", max);
  /*
   * すでにクライアントと接続されているソケットのビットを
   * セットしてselect()の検査対象にする。
   */
  for (i = 0; i < max; i++) {
    if (s[i] != UNUSED) {
      FD_SET(s[i], &readfds);
    }
  }
  /*
   * メッセージが到着しているソケットがないか調べる
   */
  if ((n = select(FD_SETSIZE, &readfds, NULL, NULL, NULL)) == -1) {
    perror("select");
    exit(1);
  }
  printf("select returns: %d¥n", n);
  /*
   * 1～maxのソケットに届くのは、すでに接続されているクライアントからの
   * メッセージである。
   * もしメッセージがあれば、接続されているすべてのクライアントに送る。
   */
  for (i = 1; i < max; i++) {
    if (s[i] != UNUSED) {
      if (FD_ISSET(s[i], &readfds)) {
        /*
         * s[i]のビットが立っていれば、s[i]にメッセージが到着している
         */
        printf("s[%d] ready for reading¥n", i);
        if ((msglen = read(s[i], str, sizeof(str))) == -1) {
          /* 受信失敗 */
          perror("read");
        } else if (msglen != 0) {
          /* メッセージの受信に成功 */
          printf("client[%d]: %s", i, str);
          Modification(str,msglen);  /* 作業 */
         /* 接続されているクライアントすべてにメッセージを送る */
          for (j = 1; j < max; j++) {
            if (s[j] != UNUSED) {
```

```
                write(s[j], str, strlen(str));
              }
            }
          } else {
            printf("client[%d]: connection closed.¥n", i);
            close(s[i]);
            s[i] = UNUSED;
          }
        }
      }
    }
    /*
     * 新たな接続要求があった場合は、s[0]のビットが立つ。
     * 以下では新たな接続の受け付けを行う。
     */
    if (FD_ISSET(s[0], &readfds) != 0) {
      printf("Accept New one.¥n");
      /* 接続を確立する accept() */
      len = sizeof(caddr);
      s[max] = accept(s[0], (struct sockaddr *)&caddr, &len);
      printf("%d = accept()¥n", s[max]);
      if (s[max] == -1) {
        perror(NULL);
        exit(1);
      }
      if (max < SOCK_MAX) {
        printf("client accepted(%d).¥n", max);
        max++;
      } else {
        printf("refuse connection.¥n");
        strcpy(str, "Server is too busy.¥n");
        write(s[max], str, strlen(str));
        close(s[max]);
      }
    }
  }
}

/* 大文字に変換する */
char *Modification(char *str, int length) {
 int i;
 for (i = 0; i < length; i++) {
  if (isalpha(str[i])) {
   str[i] = toupper(str[i]);
  }
 }

 return str;
}
```

socket descriptorとしてs[SOCK_MAX+1]を定義している。SOCK_MAXは、defineしたもので5としている。すなわち、全部で六つのソケットを用意したことになる。serverではこれらの六つのソケットを、

s[0]	新しい接続要求を待つためのソケット
s[1..4]	クライアントと接続され実際に通信をおこなうためのソケット
s[5]	接続拒否に使うの一時接続ソケット

として使うことにする。また、s[i]の値がUNUSEDの場合は、そのソケットは使われていないことを表すことする。次にfd_set型の変数を定義する。名前をreadfdsとし、selectシステムコールで利用する。最初のクライアントの接続は、acceptすればうまくいくはずである。しかし、次のクライアントのために、s[0]は再度acceptシステムコールで待ち状態にしなければならない。しかしながら、acceptシステムコールで処理をブロックしてしまった場合、最初に繋いだクライアントの応対をすることができなくなる。そこでselectシステムコールを使用する。これは、複数のソケットに対してのメッセージ待ちを可能にするもので、selectシステムコールでは検査対象のソケットを指定し、そのうちのどれかにメッセージが到着するまでブロックする。そしてどれかにメッセージが到着すると、どこに届いたかを教えてくれる。

s[0],s[1]のどちらかにメッセージが来たら知らせるようにselectシステムコールを設定し、処理が戻ると、例えばs[1]にメッセージが到着しているということがわかる。そのことから、それに従ってメッセージの到着したソケットに対して処理を行なえばいいわけである。selectシステムコールを使うにはfd_setという型の変数を使用する。この変数で検査対象となるソケットをあらかじめ指定するわけである。このfd_set型の変数は直接操作せず、マクロを使って操作する。まず、FD_ZERO(& readfds)でリストをクリアしておき、その後、検査対象はFD_SET(追加したいソケット,& readfds)を使って設定していく。selectシステムコール本体は

```
select(FD_SETSIZE, & readfds, NULL, NULL, NULL)
```

となる。FD_SETSIZEはシステムで許される最大の個数を指し、検査しないものにはNULLを指定する。最後の引数はタイムアウト時間だが、NULLを設定するとready状態になるまでブロックされる。

selectシステムコールから処理が戻ると、readfdsにはFD_SETで設定したなかで、メッセージが到着しているものだけがビットがセットされて返ってくる。これをFD_ISSET（調べるソケット, & readfds)を使ってどれがセットされているのか調べる。

serverでは、s[1～max未満]のソケットにメッセージが到着しているかどうかを調べ、到着している時は大文字に変換した後、接続中のクライアントすべてにメッセージを送り返している。

ソケットs[0]は、接続要求のくるソケットであるから、このソケットにメッセージが到着しているときにはacceptシステムコールで接続を受け付ける。また、接続を拒否する場合には、一旦acceptしたあとに、ソケットをcloseしてしまうという方法を使う。

以上でマルチクライアントのサーバが書ける。このserverでは、一度使ってしまったソケットを再利用していないが、これを再利用するようにすれば４クライアントのサーバとなる。

クライアント・プログラム例

```
/*
 * client.c
 *  ソケットを使用して、サーバーに接続するクライアントプログラム。
 *
 *  入力された文字列をサーバーに送り、サーバーが大文字に変換したデータを
 *  受け取る。
 *
 */

#include < stdio.h >
#include < string.h >
#include < sys/types.h >
#include < sys/socket.h >
#include < netinet/in.h >
#include < netdb.h >

#define PORT 8765

main(int argc, char *argv[])
{
  struct sockaddr_in  addr;
  struct hostent *hp;
  int  fd;
  int  len;
  char  buf[1024];
  int  ret;

  if (argc != 2){
    printf("Usage: client SERVER_NAME¥n");
    exit(1);
  }

  /*
   * ソケットを作る。このソケットはUNIXドメインで、ストリーム型ソケット。
   */
  if ((fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    perror("socket");
    exit(1);
  }

  /*
   * addrの中身を0にしておかないと、bind()でエラーが起こることがある
   */
  bzero((char *)&addr, sizeof(addr));

  /*
   * ソケットの名前を入れておく
   */

  if ((hp = gethostbyname(argv[1])) == NULL) {
    perror("No such host");
    exit(1);
  }
  bcopy(hp->h_addr, &addr.sin_addr, hp->h_length);
  addr.sin_family = AF_INET;
  addr.sin_port = htons(PORT);

  /*
   * サーバーとの接続を試みる。これが成功するためには、サーバーがすでに
   * このアドレスをbind()して、listen()を発行していなければならない。
```

```
    */
    if (connect(fd, (struct sockaddr *)&addr, sizeof(addr)) < 0){
      perror("connect");
      exit(1);
    }
    /*
     * 入力されたデータをソケットに書き込んでサーバーに送り、
     * サーバーが変換して送ってきたデータを読み込む。
     */
    while (fgets(buf, 1024, stdin)) {
      write(fd, buf, 1024);
      ret = read(fd, buf, 1024);
      buf[ret] = '¥0';
      /*write(1, buf, ret);*/
      printf("%s",buf);
    }
    close(fd);
    exit(0);
}
```

サーバ、クライアントプログラム MAKE例

```
CC=gcc
LFLAG=-lsocket -lnsl
CFLAG=

all:  client server

client: client.o
    $(CC) -o $@ client.o $(LFLAG)
client.o: client.c
    $(CC) -c client.c

server: server.o
    $(CC) -o $@ server.o $(LFLAG)
server.o: server.c
    $(CC) -c server.c

clean:
    rm server client *.o * ~
```

NC ZJ1064 97-001 Vol.3

限定資料

分置01

地層処分ヴァーチャル・エンジニアリング・システムの開発（Ⅰ）

データベース技術研究
ソフトウェア開発管理手法研究
専用コンピュータシステムの構築

（動力炉・核燃料開発事業団　研究委託内容報告書）

技術資料

開示区分	レポートNo.	受領日
Z	J1064 97-001 Vol.3	1998.4.14

この資料は技術管理室保存資料です
閲覧には技術資料閲覧票が必要です
動力炉・核燃料開発事業団　技術協力部技術管理室

１９９７年３月

日商岩井株式会社

この資料は，動燃事業団の開発業務を進めるため，特に限られた関係者だけに開示するものです。ついては複製，転載，引用等を行わないよう，また第三者への開示又は内容漏洩がないよう管理して下さい。また今回の開示目的以外のことには使用しないよう特に注意して下さい。

本資料についての問合せは下記に願います。
〒107　東京都港区赤坂１－９－13
動力炉・核燃料開発事業団
技術協力部　技術管理室

限　　定　　資　　料
PNC ZJ1064 97-001 Vol.3
１９９７年３月

地層処分ヴァーチャル・エンジニアリング・システムの開発（Ⅰ）

—品質保証—

梅木博之１）、　大屋昌之２）
David REED ３）、Trevor Sumerling４）
Geoff FREEZE５）

要　旨

地層処分ヴァーチャル・エンジニアリング・システムの中核となるシミュレーション機能開発にかかわる予備的な検討を行った。検討の結果、シミュレーション機能は評価シナリオが基本となって、実行に必要な複数の計算モデルに対応した独立したオブジェクトとして動作する個別計算プログラムが複雑に組み合わされ、

並列・協調して動作するアーキテクチュアが適当と考えられた。それにより、コンピュータ内部に構築された仮想世界において複雑な現象を数値的に実験し、新たな定式化手法として観察する可能性が期待できると考えられた。例えばそれは、コンピュータ内部に仮想的に構築された地層処分システムを対象に、シナリオに添って複雑にモデルを組み合わせて、それらを協調させた新しいシミュレーション技術開発課題と考えられる。

本研究では、そのような手法を、モデル協調計算手法と呼ぶことにした。モデル協調計算は、従来研究開発されてきた、トップダウン・アプローチ（現象全体を確率論的モデルで記述したもの等）とボトムアップ・アプローチ（現象全体を個別の要素に還元して個別の決定論的モデルで記述してゆく手法）とは別な、第３の手法として、次のような考え方とした。それは、コンピュータ中に組み立てられた仮想世界の中で、複数の数値計算モデルを協調させて並列的に実行することにより、従来のトップダウン・アプローチで押さえきれなかったより精密な現象過程や、ボトムアップ・アプローチで記述が難しかった、ある程度の自由度をもった現象のシミュレーションを計算で実現する方法であるとした。また、モデル協調計算技術の開発は、理論、実験といった従来の科学技術の方法とは別な、第３の科学技術の方法である計算科学技術の位置づけを明確にする科学思想とした。

このような協調計算機能については、複数のコンピュータを用途別に接続しそれぞれが協調して並列（平行）に動作し全体として一つの目的に沿ったシミュレーション動作を行うシステムの開発技術が必要となる。平成８年度は、それらの基礎的な機能を確認するためプロトタイプ・システムの試作を行った。試作の結果、計画する協調計算アーキテクチュアは技術的には開発が可能と考えられた。従って、次年度の課題として、個別のシミュレーション・モデルの連結機能研究と協調計算システム全体の基本設計が必要であると考えられた。

本報告書は日商岩井株式会社が動力炉・核燃料開発事業団の委託により実施した研究内容結果である。
契約番号：080D0330

１）動力炉・核燃料開発事業団　環境技術開発推進本部
２）高度情報化技研株式会社
３）University of Reading, Enterpris, U.K
４）Safety Assessment Management Ltd. U.K
５）INTERA Inc. U.S.A

COMMERCIAL PROPRIETARY
PNC ZJ1064 97-001 Vol.3
MARCH 1997

Developmemnt of the Virtual Engineering System for High Level Radioactive Waste Geological Disposal Performance Assessment

Simulation System Development

Masayuki OYA 1), Geoff FREEDS 2),
Mark SAWYER 3), Trevor Sumerling 4)

Abstract

Software Engineering has been studied in the context of the development of the Virtual Engineering System for Geological Isolation. Software engineering is a systematic approach to the management and quality assurance of the total life-cycle of a software product, covering requirements analysis, design, coding, implementation, testing, operation, and maintenance. The study shows that software engineering is essential for the development of the Virtual Engineering System for Geological Isolation, and therefore specific Software Engineering methodologies need to be developed for the system. In parallel with the above study, Object-oriented Database technology has been also explored as a key technique for the software development of the system based on the following understanding:

1) Due to its nature, the Virtual Engineering System will have to handle a huge variety of data, ranging from image, video, audio and textual data to numerical data and analytical codes; and
2) Such data are required to be organized graphically and utilized in an interactive GUI environment for simulating trial-and -error designing.

The findings suggest that object-oriented databases are effective systems managing and processing information.

In relation to a networked system of distributed processes and functions, a prototype system consisting of dedicated hardware and software has been developed to evaluate the feasibility of enabling technologies for such a system. Through work on the prototype system, a preliminary methodology was explored to determine key requirements such as those for distributed functions and collaborative computation, 3-D simulation capability and interactive operations.

Based on recent research, the next research targets should be the development of practical software engineering methodologies, design of practical databases and data collection systems, and the design of total system architectures.

Work perfoormed by Nissho Iwai corporation under contract with Power Reactor and Nuclear Fuel Development Corporation.
PNC Liaison : Radioactive Waste Management Project, Isolation System Research Program
Hiroyuki Umeki

1) NISSHO IWAI corporation
2) INTERA Inc. U.S.A
3) University of Edinburgh, Edinburgh Parallel Computing Centre. U.K
4) Safety Assessment Management Ltd. U.K.

【 目 次 】

1 ソフトウェア・エンジニアリング …… 1
2 データベース技術検討 …… 2
2.1 蓄積されたコード（機能的モデル）管理 …… 2
2.2 コード選択の基準 …… 3
2.3 解析コードの管理 …… 5
2.4 視覚化情報管理 …… 16
2.5 数値計算のためのデータの扱い …… 22
2.6 情報の処理 …… 25
2.7 データベース・システム要件 …… 29
2.7.1 オブジェクト指向データベース管理技術 …… 33
2.7.2 RDBMSとOODBMS …… 35
2.7.3 市販OODBMSシステムの特徴 …… 38
2.7.4 市販OODBMSシステムの比較 …… 39
2.7.4.1 実現言語、プラットホーム及び外部インターフェイス …… 39
2.7.4.2 基本データモデル …… 40
2.7.4.3 多重バリア持越機能 …… 42
2.7.4.4 複合オブジェクト機能 …… 43
2.7.4.5 動的スキーマ展開機能 …… 45
2.7.4.5.1 ノードの内容に対する変更 …… 45
2.7.4.5.2 エッジに対する変更 …… 46
2.7.4.5.3 クラスに対する変更 …… 47
2.7.4.5.4 ORION/ITASCA …… 47
2.7.4.5.5 ObServer/ENCORE …… 48
2.7.4.5.6 GEMSTONE …… 48
2.7.4.5.7 ONTOS …… 48
2.7.4.5.8 ObjectStore …… 48
2.7.4.5.9 VERSANT …… 48
2.7.4.6 オブジェクト記憶管理 …… 49
2.7.4.7 基本的トランザクション管理機能 …… 53
2.7.4.8 ロック管理 …… 55
2.7.4.9 変更通知 …… 57
2.7.4.10 バージョン管理 …… 59
2.7.4.11 同時性管理 …… 61
2.7.4.12 分散協調処理のための機能 …… 63
2.7.4.13 照会管理 …… 65
2.7.4.14 データベース許可および機密保護 …… 67
2.7.4.15 マルチメディア・アプリケーション …… 69
2.7.4.16 ユーザー・インターフェイス …… 69
2.7.5 まとめ …… 71

3　ソフトウェアの開発と管理の技術 …… ７２
3.1　ソフトウェアの一般的な管理活動 …… ７２
3.1.1　品質保証 …… ７６
3.1.2　構成管理 …… ７７
3.1.3　検証と確証 …… ７９
3.1.4　再利用可能性の追及 …… ８０
3.1.5　ソフトウェア・エンジニアリング環境要因 …… ８１
3.2　今後の技術課題 …… ８２
3.2.1　ソフトウェア開発プロセスの問題 …… ８３
3.2.2　手法に関する問題 …… ８５
3.2.3　自動化に関する問題 …… ８８
3.2.4　今後の課題 …… ９０
3.3　セイフティ・クリティカル・システム開発 …… ９１
3.3.1　ソフトウェア・ライフサイクル …… ９１
3.3.2　セーフティ・クリティカル・システムのソフトウェア、エンジニアリング… ９５
3.3.3　ソフトウェアの品質保証 …… ９８
3.3.4　ソフトウェア品質モデル …… ９９
3.3.5　コード評価 …… １０１
3.3.6　ソフトウェアライフサイクル管理 …… １０１
3.3.7　ソフトウェアの保守、バージョン及び要求事項の変更管理 …… １０３
3.3.8　独立型の検証及び確証 …… １０４
3.3.8.1　従来の検証及び確証手法 …… １０４
3.3.8.2　セーフティ・クリティカル・システムの検証と確証（Ｖ＆Ｖ）…… １０６
3.3.8.3　FORTRAN数値コードの検証と確認（Ｖ＆Ｖ）…… １０７
3.3.9　地層処分ヴァーチャル・エンジニアリング・システム
に関連した特別な問題 …… １０８
3.4　まとめ …… １０９
4　地下水モデル開発におけるソフトウェア品質管理手法事例 …… １１２
4.1　地下水モデル開発とソフトウェア品質管理 …… １１２
4.2　モデル開発過程における検討事項 …… １１４
4.2.1　モデル開発過程 …… １１４
4.2.2　ソフトウェア開発とメンテナンス …… １１６
4.2.3　コードデザインとコード開発のドキュメンテーション …… １１７
4.2.4　プログラム構造とコードの検証 …… １１８
4.2.5　モデル妥当性検査 …… １２０
4.2.6　検証と妥当性検査規準 …… １２２
4.2.7　ソフトウェアドキュメンテーション …… １２５
4.2.8　化学的技術的レビュー …… １２６
4.2.9　ポスト監査 …… １２８
4.3　モデルアプリケーションにおける検討事項 …… １２８
4.3.1　アプリケーション要件 …… １２８
4.3.2　データ収集 …… １３０
4.3.3　モデル公式化 …… １３０
4.3.4　モデルの較正と検証 …… １３１
4.3.5　感度解析 …… １３４

4.3.6 シナリオ解析 …… 135
4.3.7 結果の解釈とプレゼンテーション …… 135
4.3.8 ポスト監査 …… 136
4.4 品質管理組織構造 …… 136
4.5 品質管理プラン …… 137
4.6 品質管理アセスメント …… 138
4.7 品質管理とコード選択 …… 146
4.7.1 コード選択過程 …… 146
4.7.2 モデル開発必要性の公式化 …… 146
4.7.3 コード特性の記述 …… 147
4.7.4 コード選択規準 …… 147
4.7.5 コード選択における品質管理 …… 151
4.8 品質管理の終結 …… 152
4.8.1 序論 …… 152
4.8.2 文献レビュー …… 153
4.8.3 比較コードテスティングの必要性 …… 165
4.9 戦略と実例ベンチマーク問題のテスティング …… 166
4.9.1 モデル表現がテストされるべき典型的なフィールド条件 …… 167
4.9.2 統合的データセット1 …… 168
4.9.3 表4 統合的データセット2 …… 170
4.9.4 3次元溶質輸送コードのためのテストシナリオ …… 172
5 システムの試作 …… 174
5.1 システム開発方法 …… 174
5.1.1 問題解決へのアプローチ …… 174
5.1.2 CASE …… 185
5.1.3 システムの性格 …… 185
5.1.4 システムの規模 …… 188
5.1.5 システム開発者とシステムの利用者 …… 188
5.1.6 システム開発に潜在的な問題 …… 192
5.1.7 プロトタイピング手法の導入 …… 198
5.2 思考の道具としての情報システム …… 204
5.2.1 研究者の環境 …… 205
5.2.2 思考支援システムという考え方 …… 206
5.2.3 想像を活性化するシステムと言う考え方 …… 208
5.3 プロトタイピング手法 …… 212
5.3.1 概念の定義 …… 213
5.3.2 プロトタイピングの手順 …… 214
5.3.3 設計段階の技法 …… 216
5.3.4 プロトタイピング技法の分類 …… 217
5.3.5 プロトタイピング・ツール等 …… 218
5.4 プロトタイピング対構造化手法 …… 220
5.4.1 関連概念 …… 221
5.4.2 その他のアプローチ …… 223
5.5 試作システムの構成 …… 225
5.5.1 全体システム概念構成 …… 225

5.5.2 多重並列分散動作機能概念 …… 225
5.5.3 ビデオ信号制御仕様 …… 226
5.5.4 オーディオ信号制御仕様 …… 227
5.6 シミュレーション用システム・ハードウェア …… 227
5.7 操作用システム・ハードウェア …… 230
5.8 データベース用システム・ハードウェア …… 232
5.9 パラメータ・ディスプレイ用システム・ハードウェア …… 234
5.10 オーディオ・ビデオ用システム・ハードウェア …… 237
5.11 システム全景 …… 240

1　ソフトウェア・エンジニアリング

ソフトウェア・エンジニアリングでは、ヴァーチャル・エンジニアリング・システムに関連する総てのソフトウェアが対象とされる。例えば、地下水の流れ解析のように地層処分性能評価計算コードや専用のコンピュータ・システムとしてのヴァーチャル・エンジニアリング・システムを構築する為に必要な各ソフトウェアも含まれることになる（ただし、解析コードの数学的計算手法等はこの範疇ではない）。従って、ソフトウェア・エンジニアリングの質は、システム開発プロジェクトの質に直接影響する事は必須である。本章は、地層処分ヴァーチャル・エンジニアリング・システムを開発するために必要となる、ソフトウェア・エンジニアリングに関して以下に報告する。

（1）データベース技術

地層処分ヴァーチャル・エンジニアリング・システムの情報管理基盤となるエンジニアリング・データベース・システムについて検討した。作業にあたっては、評価用システムの試作開発を行い技術検討した。データ・ベースシステムの開発作業にあたっては、地層処分ヴァーチャル・エンジニアリング・システムの中核となる性能評価シミュレーションがシナリオにそって評価目的が選択され、また、並列に分散した評価解析コードが、操作（例えば、メンバー関数、メソッド等）としてバインドされ実行管理されるシステム制御を想定した。そして、そのための情報基盤となるデータベースとして要求される機能、アクセスモデル、アプリケーション、インターフェイス等、環境を含めて統合型データベース（マルチ・データベース・モデル）として検討した。データベースモデルの評価作業は、地層処分ヴァーチャル・エンジニアリング・システムの開発に必要なデータベースモデル（主にオブジェクト指向データベース）を検討した。

（2）ソフトウェア開発管理技術

地層処分システムの安全性評価など、人間の安全性が最も重要な問題となる機能等に係わる情報処理を行うシステムを、一般にセーフティ・クリティカル・システムと呼ぶ。例えば、航空機や核燃料サイクル施設の運転を制御する為に開発されたシステムが挙げられる。セーフティ・クリティカル・システムに係わるソフトウェアにとって最も重要な事項は、その品質である。

地層処分ヴァーチャル・エンジニアリング・システム開発作業においては、地層処分ヴァーチャル・エンジニアリング・システムをセーフティ・クリティカル・システムと想定し、ソフトウェアの品質を管理・保証するための手法について研究を進めた。研究にあたっては、地層処分ヴァーチャル・エンジニアリング・システムのプログラム開発に係わる要求分析から設計、プログミング、検査に到る、一連のソフトウェア開発と管理の標準化手法について研究した。地層処分ヴァーチャル・エンジニアリング・システムの設計段階、ソフトウェア生成、改良、バグ訂正等における信頼度評価モデルを開発し、地層処分ヴァーチャル・エンジニアリング・システム・ソフトウェア開発におけるトータルなライフ・サイクル管理手法を検討した。

（3）専用コンピュータ・システムの構築

地層処分ヴァーチャル・エンジニアリング・システム研究開発作業に伴う評価用プログラムの開発等、技術評価に利用するための専用コンピュータ・システムの構築を行った。このコンピュータシス

テムに、地層処分ヴァーチャル・エンジニアリング・プロトタイプ・システムとして地層処分ヴァーチャル・エンジニアリング・システムを試験的に実証整備し、具体的動作等の評価に資するものとして年度毎に高度化していくものとした。

2　データベース技術検討

地層処分ヴァーチャル・エンジニアリング・システムの将来における性能要求を定義付けするために、高レベル放射性廃棄物処分に関するモデル化の最新技術を包括的に検討した。現在開発されている技術から包括的な検討を行い、検討結果は後続の各節で触れている。地層処分ヴァーチャル・エンジニアリング・システムに要求される機能を以下に示す。

（1）蓄積されているコードから、数値計算コードを選択するための指針情報を提供すること。
（2）数値計算を行うのに必要な入力データを検索し、グラフィック処理により数値計算を開始すること。
（3）性能評価シミュレーション個別部品のために数値計算結果を視覚化すること。

2．1　蓄積されたコード（機能的モデル）管理

性能評価シミュレーション等のコードを管理する際の主要目的は以下の通りである。

（1）地層処分の安全性評価に係わるコードの分類と整理と行うこと。
（2）地層処分の安全性評価に係わるコードの出力を視覚化するソフトウェアと関連情報を体系化して整理すること。
（3）地層処分ヴァーチャル・エンジニアリング・システムの操作を支援するためのソフトウェアと関連情報を体系化して整理すること。
（4）地層処分ヴァーチャル・エンジニアリング・システムのシミュレーション用に選択された一つ一つのコードの機能に関する情報を整理すること。また、名称、著者、使用可能性、移植性、実験履歴などの追加情報を整理すること。
（5）参考文献について情報概略を提供すること。

表2．1－1に、地層処分ヴァーチャル・エンジニアリング・システムのシミュレーション・コードの現象別に分類している整理例を示した。実際の性能評価にあたっては、人工バリアやニアフィールド等の場所による区分、これらの現象による区分、非定型的な区分（そのつど区分条件が決められる）に対応することが望ましい。従って、モデルとその関連情報に関するデータベースは、基本的には非定型的な論理条件検索に対応するリレーショナル・データベースが現時点では有利であると考えられる。

表２．１－１　コードの整理例

A. MECHANSTIC SOFTWARE	B. VISUALISATION SOFTWARE/DATA PROCESSING
1 Inventory Evolution	1 Engineering systems
2 Corrosion	2 Geographical Information Systems
3 Leaching	3 Geological Structure in 3D
4 Geochemistry	4 Data Visualisation
5 Geomechanics	
6 Thermal Processes	
7 Groundwater Flow	
8 Gas Generation	
9 Gas Transport	
10 Chemical Transport	
11 Biosphere	
12 Site Evolution	
13 Surface Hydrology	
14 Tectonics	
15 Catastrophic Events	
16 Radiolysis	

性能評価研究には数多くの分野があり、今後のモデル開発に利用する新しいツールを開発することが必要と考えられる。例えば、地下水の流動や地球化学などのプロセスについては、シミュレーションするための各種のモデルが存在するが、それに比べ、ニアフィールドの熱・岩盤力学影響を評価するモデルは比較的少ない。このように、必ずしも各分野で一様にコードが研究開発されているわけではないため、コードを選択する際に参考となるような基準の確立が必要である。このような、追加情報をコードや関連データとともに管理することにより、体系化されたモデルのデータベースの構築が可能となると考えられる。

２．２　コード選択の基準

（１）　機能的な仕様（関連範囲、次元性、結合性及び互換性）

コード選択の際、モデルの適用範囲、関連性、及び将来性が重要であることは明らかである。モデルは、コードによって確実で適切に記述されなければならない。モデル間の相互作用やプロセスが相互に依存し合う問題は（例えば、地下水の流動、ガスの移動及び熱プロセスなど）、他のプログラムへ容易に連結できるように考慮することが必要である。

（２）　利用可能性

（パブリック・ドメイン／プライベート・ドメイン、情報源および参考文献へのアクセス）

一般に、情報源に容易にアクセス可能であれば、研究がより進展する可能性が高まるので、最初は、パブリック・ドメイン（公開財産）のコード（例えば、NEA/OECD原子力機関データバンクから入手可能な）が評価用に選択されると考えられる。どのコードを使えばシステムに要求される事項が満足されるのかを決定するために、他のソフトウェア（プライベート・ドメインのコードも含めて）と比較して評価することも重要である。例えば、並列化の研究開発における研究の見通しが最も高いと思われるような、より進歩したソフトウェアがプライベート・ドメイン（私的財産）である場合がある。

地層処分ヴァーチャル・エンジニアリング・システムのシミュレーション・コードに含められるようなソフトウェアは、研究開発団体以外の組織にも入手可能でなければならない。また、対外的にも適用されるように支援するべきである。研究の現場では、国際的に受け入れられているコードが最も好まれている。なぜならば、文書化が未完成なプログラムは、適用基準を満たすことはできないと考えられるからである。完全に文書化されたソフトウェアにおいては、コードの機能、変数、データ要求事項、及び出力オプションを正確に記述すべきであることは言うまでも無いであろう。

（3）実験履歴（受入テスト、分析的／数値的検証、実験に基づく確証）

放射性廃棄物処分の安全性の評価用ソフトウェアを用いる際には、ソフトウェアもそれに関連するデータも理論的に正当な手順をふまえていることを明白に示さなければならない。

一般に、コードの検証とモデルの確証は、地球科学と物理科学とでは多少意味が異なる。ここでは、受入テストは、「組織内」のプログラムに適用される最低レベルの品質保証を表わしており、予め定義付けされている一連の受入基準を満たしている事を意味する（このプロセスは、ソフトウェア・エンジニアリングのアプリケーションにおいては、一般に「検証」と称されている）。

検証とは本来、数学モデルのために書かれた計算をそのモデルが実行するテストのことである。このことは、大規模な計算によっても証明できるが、もっとも一般的な手法は性能評価試験である。それは、共通な問題に対して得られた複数の結果を比較することである。

確証とは、予測と自然の現象とがほとんど一致することを意味している。しかしながら、自然界に本来備わっている不確実性により、モデルを明白に定量化して確証することはほとんど不可能であり、多くの場合は現実的ではない。従って、確証は、研究所における実験結果と、観察結果をモデルが正確に反映していることを実証するような、定性的及び定量的な証拠とみなされることがある。

２．３　解析コードの管理

一般に前述の基準に基づいて、数多くのコードが開発蓄積される。コードについてのこれらの予備的な推奨事項の概要を表２．２－１に示す。

表２．２－１　推奨コード概要

CODE CATEGORIES		CODE RECOMMENDATIONS		
		Fall Back Code (Supporting Tools)	Preferred Approach	Coupled Processes Considered
A	MECHANISTIC CODES			
1	INVENTORY EVOLUTION	ORIGEN2	Determined by PNC	
2.	CORROSION	WAPDEG	*Required*	
3.	LEACHING (glass dissolution) (other waste forms)	Possibilities: GLASSL GLADIS WAPPA	*Required*	Corrosion and Leaching
4.	GEOCHEMISTRY	PHREEQE Supporting Tools: EQ3/6, MTDATA, MT3D, SOLISOL, HSC	New code based on PHREEQE/ PHRQPITZ with improved functionality	Speciation and Mass Transfer
5.	GEOMECHANICS	GEOSIM TOUGH2 with SPECTROM UDEC	Required	GW flow-thermal-mechanical (Thermal-GW flow)-(Thermal-mechanical) GW flow-thermal-mechanical
6.	THERMAL PROCESSES	TOUGH 2 SWIFT II CFEST	TOUGH 2	GW flow-thermal-gas transport GW flow-rn transport-thermal GW flow-rn transport-thermal
7.	GROUNDWATER FLOW	TOUGH2 CFEST (porous) (Supporting Tools): 3D FEMWATER (porous) FRACFLOW (fracture) MODFLOW	TOUGH2 (fracture/porous) SWIFT II (fracture/porous) (NAMMU) - porous (NAPSAC) - fractured	GW flow-thermal-gas transport GW flow-rn transport-thermal GW flow-rn transport-thermal GW flow-rn transport Groundwater flow

表２．２－１　推奨コード概要（続き）

	CODE CATEGORIES	CODE RECOMMENDATIONS		
		Fall Back Code (Supporting Tools)	Preferred Approach	Coupled Processes Considered
A	MECHANISTIC CODES			
8.	GAS GENERATION	GAMMON GLEAM	*Required*	microbial degradation-metal corrosion
9.	GAS TRANSPORT	TOUGH2 PORES	*Required*	GW flow-thermal-gas transport
10.	CHEMICAL TRANSPORT	CHEMTARD	LEHGC SWIFT II	Chemical transport - (flow) Hydrology - (transport)
11.	BIOSPHERE	BIOPATH/ PRISM	JESS + HSPF	Speciation + near surface processes
12.	SITE EVOLUTION	FFSM Supporting Tools: TIME4	*Required*	
13.	SURFACE HYDROLOGY	HSPF	JESS + HSPF	Speciation + near surface processes
14.	TECTONICS	Not available	*Required*	
15	CATASTROPHIC EVENTS	No code recommended	*Required*	
16.	RADIOLYSIS	Possibilities: FACSIMILE CHEMISIMUL	*Required*	

表２．２－１．３　推奨コード概要（続き）

CODE CATEGORIES		CODE RECOMMENDATIONS	
		Fall Back Code (Supporting Tools)	Preferred Approach
B	VISUALISATION SOFTWARE		
1.	Engineering Systems	AVS/EVS	AUTOCAD
2.	Geographical Information Systems	ARC INFO / REGIS	*Required*
3.	Geological 3D Structure	STRATAMODEL / EARTH VISION	*Required (VIS-AD?)*
4.	Data Visualisation	Various	Influence Explorer

表２．２－１では、各分野にそれぞれ２つのコードを選択することを試みた。短期的にソフトウェアを表わすのであれば、「Fall Back Code」が最適である。Fall Back Codeの適切な選択は、コードが即座に適用可能であるかとか、信頼性のあるものであるかどうかとか、望ましい実験履歴を有しているかなどに基づいている。「Preferred Approach」には、地層処分ヴァーチャル・エンジニアリング・システムにおいて研究開発が進む可能性が最も高いと思われるコードを選択した。

表２．２－２は、表２．２－１に関するより細かな情報の例である。これらの情報は、地層処分ヴァーチャル・エンジニアリング・システムを利用して問題解決にあたる場合に有益な情報となると考えられる。

表２．２－２　推奨コード情報

	CODE CATEGORIES	Databases	Data Requirements
1	INVENTORY EVOLUTION	Decay database *data associated with each disintegration of a nuclide* Fission database Cross section database	α, β and γ energy (MeV), the number of neutrons emitted by spontaneous fission or from (α, n) reactions in oxygen (in oxide fuel), the γ energy spectrum (MeV) and multiplicity (number of photons in a given γ energy band or group), radioactive decay constant, modes of decay, branching fractions. Fission yield data (fission yields of the fission products) Irradiation dependent cross-section libraries categorise cross section data in terms of actinide, fission product and (optionally) structural material nuclide libraries.
2	CORROSION	Geometric database	Repository depth determines hydrostatic pressure and temperature; cavern shapes, dimensions and extent of Engineered Damage Zone (EDZ) determines fracture density and principal stress orientations. The rock type and nature of backfill determines porosity and permeability.
		Water / gas phase properties	The characteristics of the gas and water phases, especially permeability and capillary pressure, which are strong functions of liquid saturation.
		Flow and transport properties	Porosity, hydraulic head and permeability of the host rock, of the disturbed zone, and of the repository lining and backfill. Diffusivities in the groundwater and backfill and water flow rates.
		Chemical Properties	Metal corrosion rates. Chemical data including solubilities of radionuclides and matrix and concentrations of corrosive agents, sorption coefficients, redox conditions, pH and complexing agents.
3	LEACHING	Geometric database Properties database	Waste package geometry, characteristics of the waste form, list of materials Radiation decay and shielding properties, thermal and mechanical material properties and empirical data for radiation damage, radiolysis, corrosion and leaching.

表２．２－２　推奨コード情報（続き）

	CODE CATEGORIES	Databases	Data Requirements
4	GEOCHEMISTRY	Thermodynamic database	
		Speciation and solubility data	Stoichiometry, log K, and K_{sp} constants, standard enthalpies, heat capacity and redox potentials, element concentrations for the aqueous phases, any required reactant mineral or gas phases, temperature.
		Surface complexation data	Specific surface area of the sorbent, capacitances of the surface layer, concentration of sites and a set of equilibrium constants for the surface reactions.
		High salinity data	Specific interaction coefficients.
		Temperature data:	Heat capacities, standard enthalpies or the entropies of species at 298.15K.
5	GEOMECHANICS	Geometric database	Waste package geometry, characteristics of the waste form, list of materials
		Stresses database	In-situ stress measurements as a result of gravitational, tectonic, structural and residual effects.
		Geological characterisation	Data on rock types from historical site records. Fracture density, orientation, roughness amplitude, mineralogy and hardness of fillings etc from site tesfing.
		Geophysical characterisation	Rock stiffness, strength and failure, moduli of deformation, Poisson's ratio, coefficient of friction and temperature effects (thermal conductivity and coefficient of thermal expansion of the host rock)
			Effects of excavation and concrete / backfill characteristics
6	THERMAL PROCESSES		Material properties of the medium, including density, specific heat capacity and conductivities.
			Heat source and hydraulic parameters, initial temperature distribution.
			Parameters for relative permeability and capillary pressure functions.

表２．２－２　推奨コード情報（続き）

	CODE CATEGORIES	Databases	Data Requirements
7	GROUNDWATER FLOW	Geometric database	1D, 2D or 3D geometry, repository dimensions
		Matrix properties	rock type, fracture density and orientation, porosity, permeability mineralogy, backfill properties.
		Water properties	Quantity, density, viscosity, diffusion coefficient, compressibility, salinity, fluid velocity, temperature, pressure.
			Hydraulic properties of each geological layer (hydraulic conductivities, specific storage)
8	GAS GENERATION	Radiolysis database	See Radiolysis.
		Corrosion database	See Corrosion data specifically related to corrosion of metals (steel).
		Microbial degradation	In most models, the formulation of equations is treated empirically and requires many parameters. Data includes carbon sources, pathways and sinks in microbiolgical food chain; effects of pH, temperature and moisture content; hydrolysis rate of waste, growth rates of microbes and mass of biomass carbon formed per mass of substrate metabolised.
9	GAS TRANSPORT	Geometric database	Repository depth determines hydrostatic pressure and temperature; cavern shapes, dimensions and extent of EDZ determines fracture density and principal stress orientations. The rock type and nature of backfill determines porosity and permeability.
		Gas properties	Density, viscosity, solubility in water, binary diffusion coefficient, diffusivity coefficient, saturation vapour pressure, and compressibility. Gas properties are a function of temperature and pressure
		Water properties	Quantity, density, viscosity, diffusion coefficient, compressibility, conductivity, salinity.
		Matrix properties	Stress state, rock type, density, fracture density and orientation, porosity, permeability, mineralogy, strength, modulus of deformation, Poissons ratio, coefficient of friction, thermal conductivity, coefficient of thermal expansion, specific surface area, cement/backfill properties.

表２．２－２　推奨コード情報（続き）

CODE CATEGORIES	Databases	Data Requirements
10 CHEMICAL TRANSPORT	Chemical data	Equilibrium constants for aqueous complexes and surface reactions, stoichiometry, solubility products, redox potentials, concentrations of aqueous species/ elements, constants for ion exchange and thermal models.
	Transport-decay data	Dispersivity and fluid velocity, species dependent molecular diffusion coefficient, charge, radioactive decay half-life / radioactive decay constant.
11 BIOSPHERE		Transfer coefficients (rate constants) between the specified compartments in the biosphere (first order); radionuclide(s) half-life; parent nuclides; source of activity; masses and volumes of compartments; dose function specifications; number of exposed groups for which doses shall be calculated; biological data (uptake factors); dose conversion factors; consumption data; irrigation data; population data. Speciation and solubility data - see Thermodynamic database
12 SITE EVOLUTION	Site definition database	
	Point Data	Coordinates, topography, stratigraphy;
	Climatic Data	4 global climate states considered (interglacial, glacial-temperate interstadial, glacial-interstadial, glacial-stadial); mean annual precipitation, temperature pdfs and degree of seasonal variation required;
	Scalar	Single variables, eg. Diapirism rate, repository depth.
	Polynomials	Describe relationship between interactive processes in the submodels (eg. Sea-level and global ice volume).
	Probability Density Functions	Required for each temporally changing parameter.
	Constant Data	Time invariant parameters such as time of change of stress regimes.

表２．２－２　推奨コード情報（続き）

CODE CATEGORIES	Databases	Data Requirements
13 SURFACE HYDROLOGY		Meteorological records of precipitation and estimates of potential evapotranspiration are required for watershed simulation. Air temperature, dewpoint temperature, wind, and solar radiation are required for snowmelt. Air temperature, wind, solar radiation, humidity, cloud cover, tillage practices, point sources, and (or) pesticide applications may be required for water-quality simulation. Physical measurements and related parameters are required to describe the land area, channels and reservoirs.
14 TECTONICS		Specific data unknown; will include seismic forecasting data displacement (frequency and magnitude) and associated disturbance to solid geology and hydrogeology.
15 CATASTROPHIC EVENTS		Specific data unknown; will include forecasting of human intrusion (accidental - engineering, deliberate - terrorism/war), meteorite impact (probabilistic forecasting).
16 RADIOLYSIS		G values for α, β and γ radiation (rate of radiation production per energy input) - temperature dependent. Dose rate (energy input per unit time), radiation yields, radioactive decay constant, activity, decay heat of isotope / heat production per unit time. Chemical reactions and kinetic rate constants for each chemical reaction.

現在の時間スケール内では、すべての利用可能なソフトウェアを決定的に評価することは出来なかった。しかしながら、地下水の流動に関するコードや地球科学的なコードが、選択基準（上記した）の適用方法を詳細に説明するものとして検討された。新規開発コードの機能仕様をより詳しく説明していくことが長期的に必要とされるであろう。選択された各コードの機能仕様は、以下の情報を提供することが望ましい。

- Program Summary
- Modeling Capabilities
- Testing History and Applications
  - Verification and Validation
  - Types of Use
- Program Operation
  - Program Language
  - Input Requirement
  - Output Facilities including Graphic Display
  - Machines on which the program has been used
  - Coupling
- Program Availability
- Known Users
- References

（1）核種インベントリーとインベントリー変化

核種インベントリー・コードは、多種の核種（アクチニド、核分裂生成物）のモル（グラム分子）数の経時変化を算出するものである。例えば、次の2つコードがある。ORIGEN2（パブリック・ドメイン）及びFISPIN（プライベート・ドメイン）である。本質的には両コードとも同じ機能であるが、FISPIN（ウェブスター1995年）は、特定のアプリケーション用に開発されたコードで、英国における核種インベントリーに焦点を合わせたものである。これに対して、ORIGEN2（CROFF 1980年、1983年）は、より大規模なソフトウェアシステムの一部分である。モデル化能力、適用性、及び順応性を有するORIGEN2の方が、放射性廃棄物処理に関する研究用に適したコードである。

（2）腐食

好気的環境では、ステンレス鋼も炭素鋼（軟鋼）も腐食し、水酸化（第三）鉄を生成する。嫌気的な環境では、腐食の発生後に磁鉄鉱（Fe3O4）もしくは水酸化鉄が生成すると考えられる。両方の環境において、それに対応する陰極が半減し、水素ガスが発生する。嫌気的な環境における、鋼鉄の一般的な腐食に関するデータを入手することは可能であるが（Marsh 1988年a, b）、温度、湿気、塩分などの要素によって決定される信頼できる予測に関しては、さらに研究が必要である。このことは、地層処分ヴァーチャル・エンジニアリング・システムにそのまま応用できるような包括的なモデルが存在しないことを説明している。

（3）浸出

数多くの一般的な廃棄物パッケージの劣化コードが適用可能である。これらのコードには、WAPPA（インテラ社1983年）やWAPDEG（LEEとATKINS1995年）が含まれ、両コードともさらに研究を進めていくことが必要とされている。より機械的なモデルは、ガラス溶解やセメント固化廃棄物体の劣化などのような、特定の処分概念に限定されている。これらのモデルは、定常的な平衡化学コードを修正したものに依存する傾向がある。

（4）地球化学

長期的には、ニアフィールドの化学は、人工バリアシステムの性能に大きな影響を及ぼすと考えられる。金属製容器の材質の腐食、粘土埋め戻し材の安定性、ガラス廃棄物体の溶解、溶解した放射性核種の可溶性／分種化などが、ニアフィールドの化学的条件の直接的な機能である。地球化学モデルの重要な構成要素には、平衡状態での分種化、溶解度の推定、及び反応経路の算出などが含まれている。表2．2－2に、検討された地球化学コード概略が示されている。

PHREEQE（PARKHURST 1980年）は、機能性に優れ、適用性の高さから、地球化学のモデル化に最も広く使われているコードの一つである。そのコードのバリエーションには（例えば、BROWNなど1991年; ARTHURとTYRER 1996年）有機錯化、表面における錯化（吸着）、塩水におけるシミュレーション、及び温度上昇などのプロセスも含まれている。

PHREEQE以外の地球化学コードについては、鉱物（無機化合物）の溶解もしくはバリア材の劣化などについて動力学的に算出することが必要とされている。PHREEQE以外の地球化学コードで最も広く用いられているのが、EQ3/6である（WOLERY 1979年、1983年、1992年）。これらのコードや同様の機能を有した数多くのツールが世界的プロジェクトであるCHEMVALプロジェクトにおいて、広範囲に試行されている。

（READとBROYD 1989年; READ 1990年、1991年、1994年; READとFALCK 1996年）。

（5）地球力学

安全性の分析での重要な側面として、岩体の安定性、地殻変動による応力、処分場における地下水の流動及び熱負荷などの連結状態の評価が挙げられる。廃棄熱が割れ目に及ぼす影響や、それが岩体の透水性にどのような影響を及ぼすのかを把握するために、熱・水理学的・機械力学的（THM）連成コードを適用することが推奨されている。国際的な共同研究プロジェクトである、DECOVALEX（熱・水理学・機械力学連成モデルの開発及び核廃棄物の隔離に関する実験の確証）（JINGなど1995年）が開始された。DECOVALEXの全体的な目的とは、放射性核種の処分場から生物圏への流出及び移行に関してTHM連成プロセスがいかに重要であるか、そしてこれらのプロセスが数学的モデルによってどのように記述されるのかについて、一層理解を深めることである。このプロジェクトによって開発される、ベンチマークテストやケース問題によって、地層処分ヴァーチャル・エンジニアリング・システムに使われるコンピュータ・コードを適切に選択することが出来るようになると考えられる。

（6）熱プロセス

廃棄物容器や周囲岩石の中の温度を予測するには、熱伝達の計算が必要となる。温度は、これらの閉じ込め性能及び流出性能に直接的かつ間接的な影響を及ぼす。熱伝達に左右される地下水のシステムを評価するために、いくつかの限定されたコードを適用することが出来るが、その中には、TOUGH2（PRUESS 1991年）、CFEST（GUPTAなど1987年）、SWIFT II（REEVESなど1986年）、SPECTROM（SVALSTAD 1991年）などが含まれている。これらのコードは、応力、流れ、放射性核種の移行、及びガスの移動能力などを連成した利点を備えている。

（7）地下水の流動

概念化した多孔質媒体を通るダルシー流動のモデルについては、多くの文献があり（ASHTONなど1993年）、今回のプロジェクトの初期的評価段階に使用可能な候補には事欠かない。亀裂における流動ツールについては、まだ開発がそれほど進んではおらず、種々の能力が異なるコード（パブリック・ドメインにおいてもプライベート・ドメインにおいても）を検討した。その結果、選択されたコードの能力、制限事項、有用性をリストにまとめた。亀裂が生じている多孔性媒体における流動体の流れを説明する概念的なモデルは、近接した多孔性媒体の貯蔵力と流動能力に基づいて分類することが出来る。主な4つの概念モデルを以下に示す。

a．明瞭で不連続な亀裂（例えば、SWIFT）により、亀裂と多孔質媒体間の流体の潜在的な流動や勾配を最小限の非物理的なパラメータで説明することが出来る。しかしながら、数多くの亀裂に関するデータを得る事は容易でない。
b．二重連続体モデル（例えば、TOUGH2）では、亀裂の生じていない岩体は媒体の空隙率（貯蔵力）が高く、透水性（流動）は非常に低いとする観察に基づいている。反対に、亀裂の生じた岩体の空隙率はごくわずかではあるが、透水性は高い。多孔性媒体と亀裂は、それぞれ2つの別々のものであるが重複している連続体としてみなされる。この2流動体間での流体質量移転は、亀裂・多孔性媒体の境界で起きている。
c．非連続的亀裂ネットワーク（例えば、NAPSAC HERBERTとSPLAWSKI 1990年）は、流動が亀裂に限定されている、二重連続体としてのモデル分類がなされている。
d．単一等質連続体モデルの重要性は高いとみなされ、平均的には、透水性は亀裂と多孔性媒体の透水性の合計と考えられている。

流体流動を概念化することに共通事項として挙げられるのが、問題に関連した時間的空間的尺度を決定することである。ニアフィールドの問題において、亀裂における流動について個別のモデルが必要とされる一方で、ファーフィールドの問題には、単一の連続体モデルが適用される。

TOUGH2（PRUESS 1991年）は、不飽和、もしくは部分的に飽和しているマルチフェーズの流動解析を扱うプロジェクトにおいて、処分対象地層の様々な特性を検討し、またその特性を高レベル放射性廃棄物処分場の性能評価に適用する場合に、適した解析コードとして使用されてきた。TOUGH2は、また並列化の点でも発展の可能性が高い。

また、亀裂の生じている多孔性媒体中の地下水飽和流動についての解析では、SWIFT97が、地層処分ヴァーチャル・エンジニアリング・システムやその他の一般的な廃棄物処分研究に推奨される。

（8）ガスの発生

次の2つの理由から、放射性廃棄物処分場におけるガスの発生について考えることが必要である。まず第一に、高圧ガスが通風孔のない塞がっている廃棄物キャニスタの中で発生して人工バリアを破壊してしまうかもしれないことである。第二に、溶解ガスや分離ガス相によって、移行時間が短縮するため、核種移行能力が増大してしまうことである。

ガスの発生の重要なプロセスには、金属の化学的腐蝕（好気的及び嫌気的）、微生物による劣化、熱による劣化及び放射性分解などが含まれる。ジョーンズ（1990年）の調査文献によると、ニアフィールドにおけるガスの発生をシミュレーションするためのコンピュータモデルを適切に文書化したものが一般的には不足していることがわかっている。一般的に、ガスを発生させるかもしれないプロセス毎にモデルは限定される。そして、場合によっては、有機分解のプロセスに関連したガスの発生について明確に考慮されていないこともある。最近、GAMMON（PURDOMとAGG 1993年）というコンピュータプログラムが開発されたが、それは微生物による劣化と腐蝕のモデルを組み合わせようとするものであった。

（9）ガスの移動

処分場内部でのガスの発生と移動が地圏における放射性核種の流動性に影響を及ぼす恐れがあることが考えられる。処分場を経由したガスの移動や処分場からのガスの移動には、相互に関係し合う広範囲のメカニズムが寄与している。これらのメカニズムを扱う手法は、次に述べる3つの分析方法に分けられる。

- **ガスの移動及びマルチフェーズ流動**
  ガスが移動する方法は、処分場の環境内の水圧に強く影響を受ける。
- **地球化学的影響**
  毛管流動が主要なガス移動メカニズムであるならば、毛管圧力が現存の亀裂をさらに拡大させることになるかどうかを知ることが必要である。さらに処分場の周囲の地下水とガスの流動経路は、岩体の応力分布よって決定され、また、処分場掘削時やその後の密封などに対して岩体の体積が時間の経過とともに反応することによっても、処分場の周囲の地下水とガスの流動経路は決定される。従って、岩体や埋め戻し材の性質について、詳しく理解することが必要になる。
- **生物地球化学的影響**
  溶存した状態もしくは分離状態のガスの移動が、化学的吸着により遅れる可能性もある。

ガス移動メカニズムを完全にモデル化することができれば、亀裂が生じたストレスの変化の影響を受けやすい代表的な岩体マトリックスを介して、溶存状態もしくは分離状態のガスが移動出来るようになる。その際、吸着のような遅延要因も考慮に入れられている。現在のところ、このタイプの概念モデルを適用することは出来ない。システム全体の個別の現象を解析するためのモデルは存在する。適用可能なコードのうち、TOUGH2とPORES（PONTINGなど1983年; SCOTT 1990年）の2つのコードがガス移動のマルチフェーズ的な流動をモデル化するためとして認識されている。両方のコードともパブリック・ドメインのコードであり、亀裂の生じている岩体をモデル化するのに、二重空隙率モデルか透水性モデルかどちらかを選択することができる。PORESは、油層をシミュレーションするものであるが、上記に関連した二相の流動に関する問題にも適用可能であろう。

（１０）汚染物質の移行

放射性汚染物質の移行をシミュレートするためには、同時に作用している地球化学的プロセスと水理学的プロセスを正確に理解することが必要である。

現在のところ適用可能なツールは、CHEMVALプロジェクト（READ 1991年; READとFLACK 1996年）において評価されている。また、CHEMTARAD（BENNETTなど1991年）は、適切に文書化されており、広範囲に検証された化学連成移行コードである。また、このコードは、パブリック・ドメインで入手可能である。しかしながら、初歩的な数値解析アルゴリズムに基づいており、一次元的なシミュレーションに限定されている。LEHGC（YEH 1994年）は、飽和媒体及び不飽和媒体中の地下水化学的移行を解析する最新の有限要素法モデルである。CHEMTARDとLEHGCの両方のモデルを使用して検討可能なプロセスは、地下水の流動（移流、分散／拡散）、アクア錯化、イオン交換、吸着、沈殿、溶解などである。HYDROGEOCHEM（YEHとTRIPATHI1991年）によって後者のコード（LEHGC）が適用されて以来なされた改訂・改良は適切に文書化されている。LEHGCは、並列化に適用しやすいコードであり、将来発展することが期待される適切なコードと思われる。

（１１）生物圏

放射性物質に関する評価の全体的枠組みでの生物圏モデル化は、空洞や地圏の進展をモデル化することよりも関心が低い。一般的に、現在の方法論では、簡単な分画的なモデルが用いられ、多くの生物圏の経路やプロセスが削減されて、前もって決められている個体群への被曝量を概算するための線量換算要因となっている。

適用可能で柔軟性のあるツールのモデルの中から、BIOMOVSプロジェクト（BIOMOVS2 1996年）において、開発及び実験が継続中のBIOPATH/PRISM（BERGSTROM他 1982年; GARDNER他 1983年）が評価用に選択された。しかし、複雑な生物圏の経路が厳密な構造の中で簡素化されているために、これと同様の分画モデル制限が多いものになっている。従って、生物圏に関する問題に分種化に基づいた手法を適用して、各分画（例えば、土壌、堆積、入江など）における主要な放射性核種の移動性や有毒性を限定してはどうかと考えられている。

高度な分種化モデルシステムである、JESS（分種化についての共同専門システム）（MURRAYとMAY 1995年）は、生物圏における化学作用をモデル化するための強力で、柔軟性のある予測ツールとして推奨されている。このシステムは、非常に包括的なデータベースによりサポートされている。気候、植物、土壌及び水理学との複雑な相互作用について検討する必要性があるので、機械論的方法で近地表環境に適用できる統合化モデルを構築するために、JESSと適切な地表水理学パッケージを組み合わせる提案が出されている。

（１２）サイトの変化

サイトの変化モデルの目的は、廃棄物隔離システムににあるひとつ以上の自然バリアの有効性に影響する一定の事象とプロセス（自然及び人間によって引き起こされる）を定量的に表わすことである。２つのサイト変化コード、すなわちTIME4（DAMESとMOORE 1991年）とFFSM（インテラ社1983年b）を検討した。

TIME4は、気候によって引き起こされる環境の変化については検討しているが、地殻変動や生物発

生のプロセスによる影響及び人間によって引き起こされる環境の変化などについては検討されていない。TIME4のパラメータの大部分は、生物圏の生物学的特性よりはむしろ物理的特性について多く記述している。

FFSM（ファーフィールドにおける状態モデル）は、確率論的枠組みの自然バリアにおいて自然、及び人間によって引き起こされる変化に適用される比較的古いタイプのモデルである。将来、多種の時間枠（例えば、腐蝕プロセスについては０年から１００年、再飽和については１００年から１、０００年、氷河作用については１、０００年から１０、０００年）における重要なプロセスを表記するために、規模を拡張した変化モデルを適用することが望ましいと考えられる。

（１３）地表水理

地表付近の水理コードを選択する際、一般的なコード選択基準（有用性、実験履歴、次元性など）を越える多数の追加機能を検討した。特に一連の気候条件下での地表水のプロセスをシミュレーションするために、モデルが使用可能かどうかを検討した。合計１２個のよく知られているコードから、詳細に検討するために２つのコードとは、SHETRAN（EWEN 1990年）とHSPF（BICKNELLなど1993年）を選び出した。SHE（欧州水理学システム）はヨーロッパにおいて広範囲に適用されているが、その最も高度な形態のコードはパブリックドメインにて公開されていない。HSPFは、USGSとUSEPAによって承認されたパブリック・ドメインにて公開されている地表水理コードであり、現段階においては地層処分ヴァーチャル・エンジニアリング・システム用のコードに利用可能と考えられる。

（１４）地殻変動

現在までのところ、地質時代における地殻変動による圧縮が土壌に及ぼす影響をモデル化する数値コードは確認されていない。もし、その必要性があると考えられるのであれば、この分野における研究及びコードの開発をさらに進めていく必要がある。

（１５）天災事象

地震や火山の噴火などの天災事象が引き起こす地圏への影響をモデル化するのに適しているコードは、地層処分ヴァーチャル・エンジニアリング・システムプロジェクトの第一段階では検討していない。この目的のために「既製品」コードがあるならば、あるいは開発が必要とされているならば、さらに調査研究を進めていく必要があると考えられる。

（１６）放射線分解

高レベル放射性廃棄物処分場において、放射線分解は、廃棄物容器の近辺の地下水の化学的性質を変化させることがある。その結果、容器の腐蝕率や廃棄物の溶解度、及び放射性核種の可溶性に影響を及ぼすこともある。放射線分解により劣化しやすい材質は、イオン交換樹脂、アスファルト、プラスチック、コンクリートなどである。コンクリートは空隙水が劣化しやすい。高温高湿下の一連の酸化還元条件下では、放射線分解によるガスが発生しやすく、特にコンクリートについては、水の劣化によりガスが発生しやすい。水の放射性分解によって、基やイオンが結合して反応することにより、数多くの末端元素（過酸化水素、水素、酸素などを含む）が形成される。これらの新しい元素の発生

は、吸収されたエネルギーとの関係で、G値を使用して定量化される。これらの値は、100Evのイオン化した放射線を吸収した結果、形成されるある一定の種の分子の数として定義付けされる。

このアプリケーションには、標準的な機械的モデルはない。モデルによって一般的に検討されるのは、簡単な同種のものから構成されるシステム（純粋な水の場合が多い）で、拡散については検討されていない。適用される一般的な手法によって、重要な反応（反応の再結合及び劣化）や適切なG値が限定される。これらの反応は、割合を表わす方程式によって表わされ、形成及び（時間の経過と共に）消費される分種（ガスを含んだ）の量がその等式によって算出される。一般に、このタイプのモデルにおいては、検討されるのは溶液だけで固体については検討されていない。放射線分解を表わす微分方程式を解くために推奨されるコードが、FACSIMILE（非公開）である。この場合、リアクタントや生成物の化学的性質の他にその動きについても検討されている。

２．４　視覚化情報管理

本項では視覚化ソフトウェアの実例を挙げて検討し、視覚化情報の管理に関する技術について示す。

（１）ARC/INFO

ARC/INFOは、本質的に異なるデータを統合し、新しい機能を受け入れることができることからも、地理学的な情報システム（ESRI 1995年a）のおそらくリーダー的存在といえる。ARC/INFOによって、視覚化及び図式化されたデータを単一の地質処理システムを用いて統合し、地理学的な背景知識と対比させて、すべての空間的な参考データを考察することが出来る。GISによって、マルチメディアの相互作用が可能になり、それによってグラフィック化されていないタイプのデータ（例えば、ベクトルデータ、地表、ビデオ、走査イメージ及び文書など）を見ることが出来る。ユーザーは、また、CADシステム、アースビジョン、文書管理システムなどの他のアプリケーションを用いて、分析を行うことができる。また、出力されるデータが空間的な性質をもっているのであれば、マップと対照させて結果を考察することも出来る。

ARC/INFOの中枢となるのが、別々に限定された一連の層や適用範囲に地理学的情報の概要を取り入れた地理学上のデータモデルである。その一つ一つがそれに密接に関連した地理学的な特性を表わしている。この手法は、主に、関連したデータベースの管理システム（RDBMSs）の概念から取り入れられたもので、その手法はそのような管理システムによって厳密に統合されたものになっている。

いくつかの選択自由なソフトウェアの拡張機能により、ARC/INFOにアプリケーション上の特定の機能が加えられる。例えば、ArcStorm（Arc Storage Manager:アーク記憶処理）は、空間的な情報の記憶及び処理装置である。この特徴指向の連続的な地理学的なデータベースは、ORACLE、INGRES、INFORMIXやSYBASEなどの関連データベースシステムに統合される。ArcCADは、データ構造が根本的に異なるARK/INFOとAutoCADを結び付ける。AutoCADデータモデルの拡張機能として、ArcCADは、GISとCADを統合する際に以前生じていたいくつかの制限条件を取り除くことができる。ArcViewデスクトップ作図及びGISバージョン3.0（ESRI白書シリーズ1996年11月）は、マップの表示及びそれについての質問、データ分析、編集、作図などに関する強力なツールとしての特徴を有する。

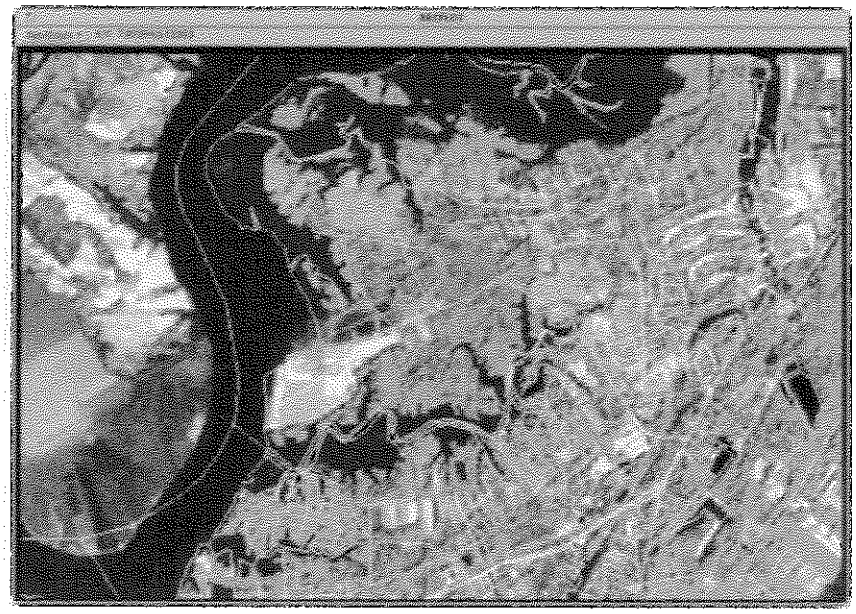

図２．４－１　ARCINFO表示例

プロジェクトの進行に伴い、自動作図、地理学的な情報システム、イメージ処理などの様々な技術を用いることによって、膨大な量の空間に関するデジタル化されたデータが、基礎的な研究やサイト調査プログラムなどから蓄積される。クライアント／サーバー計算やデータベース管理技術の発展により、空間に関するデータベースエンジンを使って、そして地理学的及び地質学的データを管理し、空間分析に関する質問に基づいて選択されたあらゆるアプリケーションに対して情報を普及させていくことがユーザーにとって可能になる。

データ管理に加えて、エンジンによってクライアント／サーバー共同処理アーキテクチュアが構築されなければならない。その際には、アプリケーションの効率的な性能は保証されていなければならない。環境システム研究機関の空間に関するデータベースエンジン（ESRI 1995年b）が、論理上の起点である。相互に作用する視覚化技術により、ユーザーと数値計算の間に重要なインターフェイスが形成される。アルゴリズムの内部作用を明白にすることによって、画面上で実験をすることが可能になり、post priori感度分析の手法を適用するよりもより効果的な方法で計算結果を得ることが出来る。

（２）Vis-5D

Vis-5Dシステム（HIBBARDとSANTEK 1990年）は、このシステムのモデルの履歴ファイルが５Ｄ矩形のデータ（２Ｄの配列と３Ｄの空間的グリッド）であるという事実からその名称がつけられている。２Ｄの配列は、時間とモデルフィールド（例えば、温度、圧力、塩分、風など）によって表わされる。これらのデータは、ユーザーが相互制御する形で３Ｄの動画で表わす事も出来る。システムはデータグリッドを変換して、３Ｄベクトルと多角形から構成される原線として図式化される。原線を図式化して表わすことによって、「仮想上の地球環境」が作り出され、データグリッドをスライスが移動することによって、あるいは図式化されたイメージを回転させたり、拡大縮小させたりして調査

することができる。

Vis-5Dシステムは、特に、天気予報のモデル化や大気及び海洋の視覚的なシミュレーションのために開発されたものである。それに関連したシステムである、Vis-AD（アルゴリズム展開のための視覚化）システム（HIBBARDなど1992年）は、より一般的なアプリケーション用に開発されたシステムである。Vis-5Dが単にポストプロセッサーとして働くのに対して、Vis-ADは多種多様な視覚的実験を支える実行環境としての役割を果たす。例えば、Vis-5Dにおいては、データは５Ｄの矩形から構成されるとされ、３Ｄの図式化されたスペースはいつも３Ｄの物理的スペースを表わしているとされる。一方で、Vis-ADにより、より幅広い分野のアルゴリズムをサポートするために、ユーザーは前述のデータ構造を定義付けし、図式化されたスペースを抽象化することができる。Vis-ADシステムは、データモデル、コンピュータモデル、ディスプレイモデル、及び図式的なユーザーインターフェイスを組み合わせるものである。（HIBBARDなど1994年）

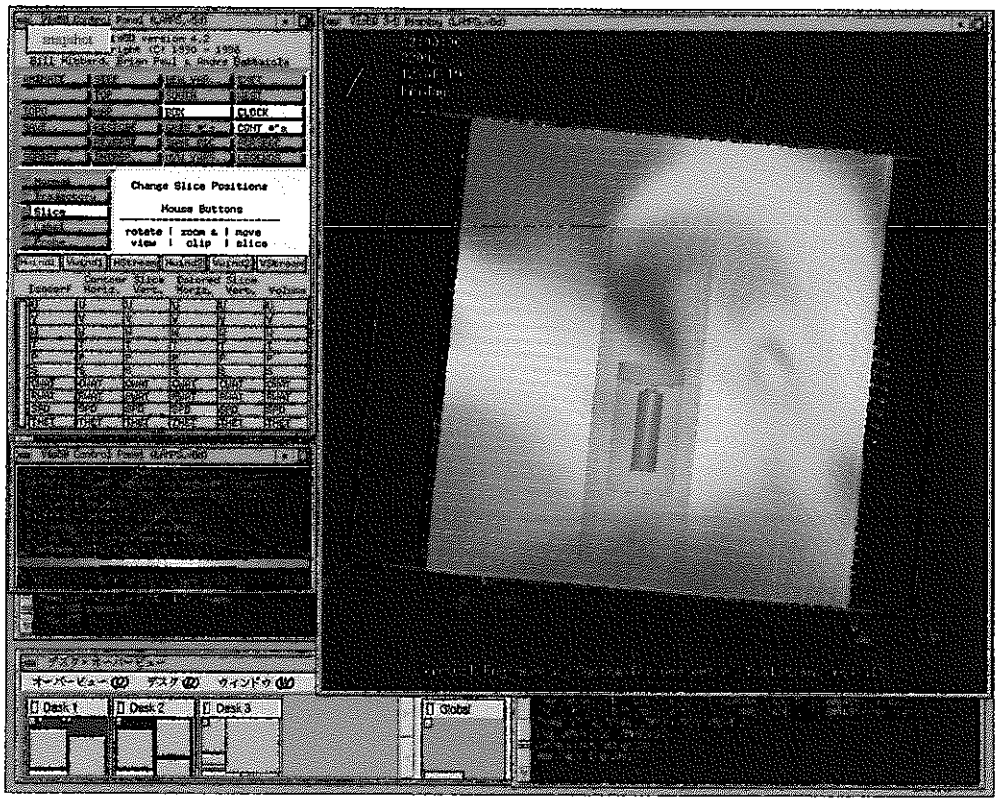

図２．４－２　Vis-5D表示例

（３）影響ダイアグラム

データ視覚化の見地からは、現在適用されている影響ダイアグラムや相互作用マトリックス手法の中には、本来非常に便利なものもあるが、そうではないものもある。例えば、岩盤工学的手法（HUDSON 1992年）の個々のループは多くの情報を伝えることが可能である。しかしながら、ループの定量的及び定性的解釈が簡単ではないので、ユーザーが理解するにはかなり難しいように思われる。「ループゲイン」をある程度まで視覚的に表示することは、価値のあることである。もし、ループゲイン

が相互に作用し合うものであったならば、表示方法はかなり改善されたと考えられる。次に、図２．４－３に示されているように、ダイアグラムの中の経路によって表わされる多くの相互作用には、強力な数学的枠組みがすでに存在している。より多くの明白な情報を得るためにこれらの相互作用を利用することができる。

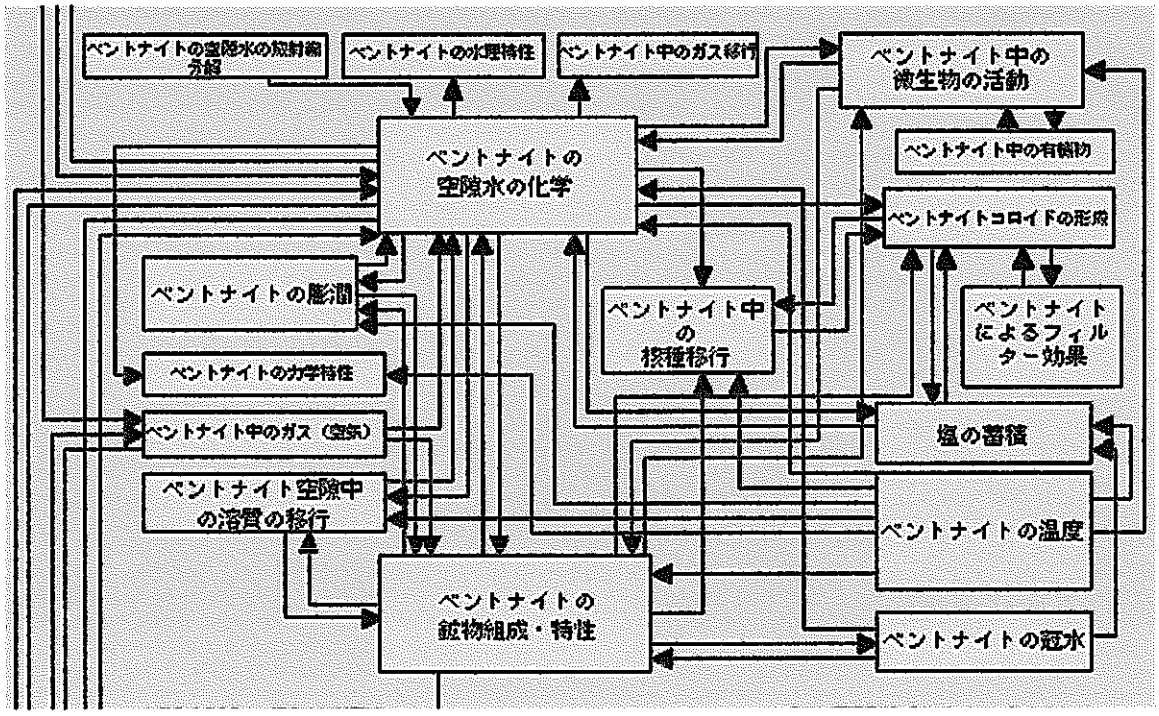

図２．４－３　影響ダイアグラム表示例

（４）影響探索

「インフルーエンス・エクスプローラ」（TWEEDIEなど1994年）は、ユーザーが多くの複雑なデータを相互的に調査可能になるように開発されたツールである。データベースにおけるそれぞれの属性は、ヒストグラムの形で表わされる。例えば、図２．４－４は、２つの異なる属性であるＸ１とＸ２のデータの分布を表わしている。バーの高さは、ヒストグラムのスケール上の各間隔の範囲内に値が入っているデータベースの項目数を表示している。このようにして、データ項目の値を示すために、ユーザーはバー上のひとつの点を選択することが出来る。そして、データ項目の値は、ディスプレイ上で強調される。

ユーザーは、ヒストグラム軸にスライダーを用いる事によって、選択することもできる。このことによって、選択されたヒストグラムだけでなく他のすべての関連ヒストグラム上のスライダーの限定範囲にあるすべての項目が強調される。例えば、図２．４－５においては、低い値のＸ１を有するデータ項目はＸ２の値も低くなる傾向があることがわかる。スライダーをドラッグすることによって、ユーザーは、一つの属性における変化が他の属性にどのように影響を及ぼすのかについて理解することが出来る。図２．４－６は、選択したスライダー（Ｘ１）がスケール上を移動した場合の結果を表している。このシステムに特有の反応フィードバックによって、ユーザーはＸ１とＸ２が相互に関連

しているということが分かる。同様に、異なる属性の間の関係を調べることによって、ユーザーは大量の複雑なデータベースを迅速に調べることが出来るのである。

図２．４－４　影響探索図表示例（Selecting a sigle data item）

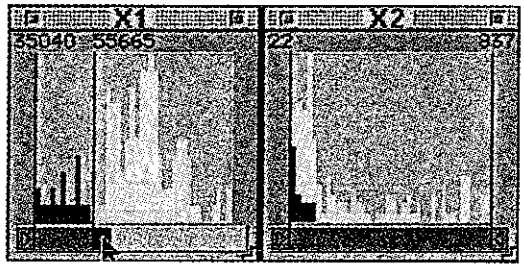

図２．４－５　影響探索図表示例（Selecting a range using a slider）

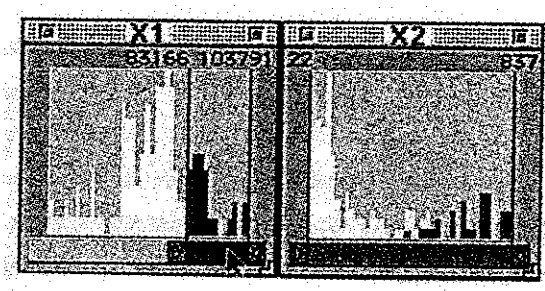

図２．４－６　影響探索図表示例（Dragging a slider range）

ユーザーは、2つ以上のスライダーを組み合わせて選択したいと思うかもしれない。

例えば、図２．４－７においては、ユーザーはＸ１、Ｘ２、Ｘ３のスライダーの範囲を選択している。この場合、感度に関する情報を加えるためには色でコード化する手法が適用される。両方のスライダーバーで設定された限定範囲内のデータ項目は黒でコード化される。それに対して、どれか一つのスライダーの限定範囲でも外れてしまったデータ項目はダークグレイの色にコード化される。これらの項目は、ユーザーの要求事項をほとんど満たしており、選択された属性のわずかな変化によって選択されるようなシステムにおける潜在的な変化もこれらの項目により表示される。これらのダークグレイの点により、ユーザーが将来どのようにスライダーを調整したいと思っているのかについての

有用な情報を得ることができる。２つ以上のスライダーの限界範囲を満たすことができないデータ項目が、ライトグレイの点によって表わされる。

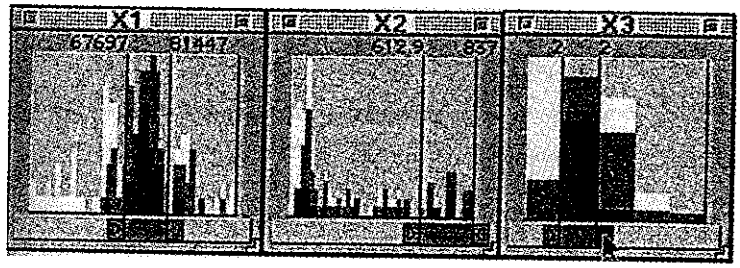

図２．４－７　影響探索図表示例（Combining a number of different slider limits）

視覚化データベースに加えて、相互作用ツールもまた、数学的モデルの研究に適用することできる。相互対応ヒストグラムは、それに関連したツールである「インフルーエンス・エクスプローラ」（SPENCE他1995年）に適用されている。この場合、入力パラメータ値のサンプルをとって、出力結果を算出することによって、データが得られる。前もって算出されたこれらのデータは保存され、その結果、相互対応ヒストグラム技術を用いて、データは考察される。このようにして、ユーザーは複雑な問題を理解するために、入力変数と出力変数の関係を質的に調べることが出来る。

出力パラメータの設計を最適化しようとする際、図２．４－７に示したように感度をコード化することによって、ユーザーはその出力パラメータから仕様を選択することも出来る。生産や構築では設計の変更があるため、ユーザーは、要求事項を満たすような単一の設計を求めているだけではない。その代わりに、例えば廃棄物キャニスタや埋め戻し材などのような信頼性の高い製品を作製出来るような、既知の公差が含まれる設計仕様などを必要としている。このような目的を達成するために、入力パラメータヒストグラム上のスライダーを調整することによって、ユーザーは入力パラメータの公差を設定することが出来ると考えられる。生産高やコストなどの設計全体の目標を実現するために、このような公差を、最適化することが可能である。図２．４－８では、最小のコストで最大の生産高を達成するための一連の公差がユーザーにより最適化されている。それは、公差範囲の幅によって表わされる。

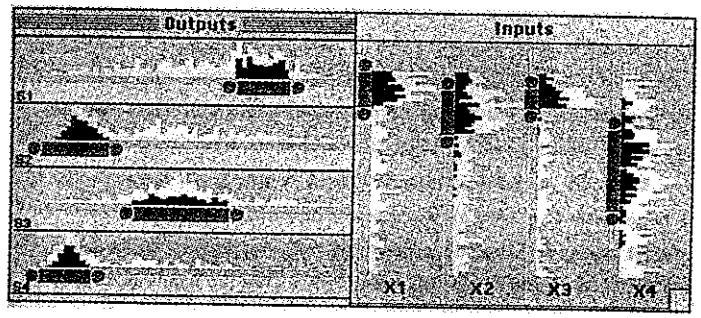

図２．４－８　影響探索図表示例（Setting range on input data and output）

特に相互対応の結果が直ちに現れる時（すなわち反応的である時）、ユーザーがダイアグラムと相互対応が可能であるならば、ディスプレイの背後に「隠されている」数学的な関係を洞察する能力は飛躍的に高まる。そうすれば、ユーザーは、「～したらどうなるか」という質問をすることが出来る。また、パラメータ値を自由に変更し、その結果を認識することが出来るのであれば、新しい事実を発見することも可能である。

２．５　数値計算のためのデータの扱い

処分場の挙動やその環境への影響を予測するためには、システムの物理的、化学的及び生物学的特性を説明する代表的なデータを用いて、機械的な方法でシステム全体をモデル化することが非常に重要であると考えられる。このようにして、予測シミュレーションコードに必要とされるデータベースを選択し、管理することは、コードそのものを選択するのと同じくらい重要である。コードもしくは専門的なシステムに関連しているすべてのデータにあらゆるフォーマットでオンラインでアクセスすることは困難である。すべてのデータを保存・管理するためには、様々なレベルにおけるサポートデータベースが必要となると考えられる。

モデル化調査に基づく数値的な予測は、適用されるデータの品質に大きく左右される。このため、データの標準の必要性が高くなり、特に性能評価計算の場合はデータベースを選択し易くするための一連の基準も必要となる。最も重要なことは、放射性物質の評価研究結果をまとめたデータベースは完全であり、内部的に整合性があり（データが体系的に標準化された）、追跡可能な情報源を参照したもので、最新のデータ処理技術に対応可能なこと、より現実的な予測が行われるように包括的に実験されていなければならないことである。上記のような要求事項をすべて満たすことはできないので、より実用的な手法が（そのデータの適用性やリソースが限られていることなどから）これまで適用されてきた。

（１）データベースの管理

膨大な量の数値データを保存・管理するための包括的な方法は非常に重要な要件である。表２．２－２は、機械的なコードの各カテゴリーのデータに要求される事項、及びそれに関連するデータベースの一覧表である。コードの各カテゴリーに関連するデータベース間にはある程度重複する部分があり、その結果、対応するデータベースのデータも重複していることが認識できる。

理論的には、必要とされるすべてのデータは、標準的な参照の情報源から得られるか、もしくはそれらのデータは特定サイトのもので、マトリックス特性や水理学的特性などの処分場の設計特性に左右される。移行分種に特有のデータは、温度や圧力それに塩分、pH、Eh等のような空隙水に関するその他の特性などの主要変数に左右される。このような要因は、処分場の深さや地質的位置選定などに大きく影響される。情報源である共通のデータや依存パラメータを表示するために、ガスの移行に関連したガス、水及び岩石マトリックスデータをまとめる試みがなされている。従って、数値コードによって参照されるすべてのデータを整合性を持たせ確実に保存するために、データベースを統合して、適切なデータベース管理システムを開発する必要がある。データベースシステムによって、特定のタスクに関連したすべてのデータを見つけ出し、引き出すことが可能でなければならない。

最新版のデータが主要情報源である、特定のデータベースの各バージョンを電子的に保存すること

が提案されている。以前にまとめられたデータベースと、より新しいバージョンのデータベース（未検討である）が比較研究や研究開発のためにそれぞれ保持されると考えられる。例えば、固体やイオンの複合体の水溶液における安定性を説明する熱力学的データベースについては、可溶性や分種化を算出するためにただ一つの熱力学的モデルが用いられると考えられている。これまでは、他の研究者たちによる研究成果と直接比較するために、他のコードを使って熱力学的なデータを変換させる事が必要であった。

従って、地層処分ヴァーチャル・エンジニアリング・システムにおいて、熱力学的データを2つの形態で保持する方法が考えられている。

第一の熱力学的データとは、選択されたコードと共に直接使用するのに適しているデータであり、決定論的なあるいは確率論的なシミュレーションをする際、データに迅速にアクセスすることが可能になる。第二の熱力学的データとは、第一のデータと同じ熱力学的データを有していると同時に、特定の測定が行われた実験状態の概略などの、重要な情報や文献目録データを有しているフラットファイルのことである。これによって、文献目録的なデータベースの動的リンクが可能になり、情報源となる文書にアクセス可能となる。そして、二次的なデータベースを形成するような数値的なソースファイルもまた作製される。後者（第二の熱力学的コード）の重要性は二重構造になっており、同一の基本的なデータを用いて予測コードを比較することは可能であるが、すべてのコードの使用に適しているフォーマットですべてのデータを保存し管理する必要はないと考えられる。

このように、別個のモデルの成果を比較することは必要とされるべきであり、それによって適切なデータベースを形成することが出来る。データベースを一つのフォーマットから、他のコードとの使用に適した形態に直接変換するために数多くの小規模のコードが書き込まれてきているが、この方式を本プロジェクトに適用することが望ましいと考えられる。

（2）データ選択の基準

特定の数値シミュレーションのためにデータを選択する際は、ソースデータの信頼性と互換性を評価するために一連の基準を適用することが必要とされる。すべてのデータ型に関連してこのような基準を認定することが本来のプロジェクトである。例えば、熱力学的なデータを決定選択する際に考慮しなければならない重要な基準について以下の項に取り上げている。同様のデータ評価は、地層処分ヴァーチャル・エンジニアリング・システムにおけるすべてのデータベースに必要とされる。システムに関連している化学的、熱力学的、及び動力学的データに関する現行のデータベースについての調査が、原子力エネルギー機関（NEA）によって1994年に開始された。NEAによる研究調査（NEA 1996年）においては、熱力学的なデータベースは次のように分類されている。

- 「純粋な物質」（例えば、CODATA;COX他1989年）についてのデータを有しているデータベース。
- 金属・金属、金属・酸素、酸素・酸素の各システムに関連している特定の「解法データベース」
- 水システムおよび地球化学システムをシミュレーションするためのデータベース。

NEA（NEA1996年）によってリストされたデータベースの比較や、厳密な評価はなされていなかったが、表向きは同じ用途に開発されたツールに共通性が欠けていることは明白である。このことは、深刻な問題であるので、この研究分野に関してさらに検討する必要がある。

放射性物質を包括的に評価するために設計されたあらゆる組み合わせのデータの品質も変化しやすいことは、避けられない事実である。脱落のリスクよりは損害が少ないと思われているので、測定された定数は理論的もしくは化学的類推に基づいた推測によって補足される。ウラニウム及びアメリシウムの熱力学を包括的に考察したことによって、適用可能なすべての情報が厳密に評価され、特定の情報を選択した理由が明白にされた。（GRENTHE他1992年, SILVA他1995年）適用された実験手順を検討することや、標準的な状態の温度、圧力及びイオン強度に従って生データが転写もしくは再計算された方法を検討することに基づいて評価が進められた。

ある特定の分野のアプリケーションについては、科学的に厳密な選択を行ったり、データを削減したりすることができる。しかしながら、資源やデータに互換性があるにもかかわらず、地化学的な計算のためのデータベース範囲内では、そのように徹底した手法を適用する事は出来ない。CHEMVALプロジェクト（1987年から1994年）において、データベースの検討や拡張（FALCK他1996年）のために、下記事項に関する一連の手順が適用された。

- **実験的な手法の妥当性／感度**
- **反復分析の精度**
- **データ推測の際に、関連しているすべての平衡状態を含有すること**
- **実験条件の選択及び範囲**
- **算出方法とデータの削減**
- **使用される補助データの信頼性及び互換性**

補助データの中には、最初の実験で明示的に研究されていないが、実験データを引き続き評価する際には適用される定数も含まれている。データを選択する際は、想定モデルとそれに関連する適切なデータがデータベースの他の部分と矛盾しないようにしなくてはならない。研究開発において「凍結データセット」を確立させるため形式化した手順が適用され、その結果、CHEMVALの６つの管理バージョンが作成される。（FALCK他1996年）このような手順は、モデル化による計算をその後確証するのに必要不可欠なものである。

水、ガス及び固体の各相に分布されている多種の種を含んでいる自然システムをモデル化するためには、水種と「純粋な物質」混合物のデータが両方必要とされる。従って、共通参照状態を適用する際には、誤った結果が得られることのないように注意を払わなければならない。異なるデータベースからのデータを混合すると、また、厳密に評価されたデータと推測データ（実験的な論拠となるものが限定されている）が混合してしまうと、地化学的な計算が不正確になものになると考えられる。

FALCKとREADは（1997年）、パブリック・ドメインにおける数多くのデータベースの内容をイオン強度、温度正法、種、相及び熱力学的な定数の観点から比較した。検討されたデータベースの中からCHEMVALの最新バージョンが、放射性物質の評価に適用できる独立型のフレキシブルな選択肢として推奨された。しかしながら、NEA（原子力機関）とCEC（欧州共同体委員会）に於いては、これも新しいデータベースにとって替わると考えられる。

（3）地球座標システム

地層処分ヴァーチャル・エンジニアリング・システムによって生成されたり処理される大半のデータは、採取場所が表示されたタイプのものである。例えば、遠隔探査、土地の測量、試錘孔ログ、数値的モデル化及び処分場の設計などの多くのソース（情報源）から得られるデータは、適用されるデータベースあるいはソフトウェアパッケージに特有の局所的な座標システムに左右される。この空間的な情報（動的なシミュレーションのための情報や時間に関するデータ）を管理する際には、地層処分ヴァーチャル・エンジニアリング・システムの構成要素間をその情報が移動する間、この情報が確実に維持されるために厳密な品質管理を行うことが必要となる。

このために、地球座標システムを最初から適用する方法が考えられる。地球座標システムを適用するための方法とは、国際的な基準を参照することである。従って、すべての空間に関する情報を次のように表わすことが望ましいと考えられる。緯度と経度を実際の北極と南極、赤道及びグリニッジ子午線に従ってそれぞれ表わし、海抜の基準面は国際的な基準による平均海面によって表示する。

地球座標を適用することにより、システムの構成要素において適用されている局所的な座標に関わりなく、地層処分ヴァーチャル・エンジニアリング・システムにおける空間に関するすべてのデータの位置が明白になる。基準となる地球の枠内の空間情報をシステムが追跡することになるので、多くのコードに共通の自動メッシュ生成ルーチンを適用することが出来る。さらに、地層処分ヴァーチャル・エンジニアリング・システムによる予測出力は、マップや試錘孔ログなどのようなベースとなるデータに直接関連したものになり、広範囲にわたる後処理は必要なくなると考えられる。

2．6　情報の処理

地層処分ヴァーチャル・エンジニアリング・システムの情報データベースの機能要件は広範囲なものである。技術的な専門家をサポートするために最初から科学的な情報を提供する必要があるが、システムの適用範囲を拡大しなければならない。

例えば、教育的活動を援助するような一般的な情報の提供や、合法的な調査研究及び公衆の合意を得るための研究などにおいては、複雑性を有する多様なタイプのデータが必要とされる。従って、情報データベースの研究開発の結果、一つのソフトウェアだけが形成されるということはなく、システムが発展して数多くの別々のモジュールから構成されるシステムとなる。このようにして、新しいデータ管理技術が発展するにつれて、システムがそれほど大きく破壊されることなく、旧式のモジュールとは別のモジュールに取って替わられると考えられる。プロトタイプ（原型）システムの概要については後述の通りであるが、それはこの第一段階において重要と考えられているモジュールを説明するものである。

（1）プロトタイプ・システム

地層処分ヴァーチャル・エンジニアリング・システム・プロジェクトにおいては数多くの様々なタイプのデータを管理することが必要とされるであろう。例えば、前述したように、熱力学的及び流体力学的なデータのような数値上の情報を必要とするシュミレーション・コードにそういった情報を送らなければならない。このようなデータはすでに発表されている参考文献から得られるであろう。従って、システムには強力な文献目録上のデータベースが必要となるであろう。地図、写真、遠隔探査

などのような空間的な情報によって、文献目録上の及び管理上の問題が認識されるようになるであろう。そして、そうした問題に迅速にアクセスしたり、フィルムやオーディオ及びビデオでの記録などのように文献として保存されていないものに迅速にアクセスすることが可能になることによって、システムに対する需要が高まるであろう。

公衆の需要についての研究をサポートするために、業界に関連しているグループの政策や技術的な立場についてよく理解することが必要である。このために、プロジェクトが進行している間、重要とみなされている問題を記録するモジュールを考慮に入れてはどうかと提案されている。このことによって、ユーザーは例えば、ガスの発生によって処分場を保全することが脅かされることなどの特定の問題について、特定のプラントのオペレーターの意見と環境問題に関係する圧力団体の意見を比較することが出来るようになるであろう。

同様に、日本と海外における一つ一つの研究プログラムの進行の様子を監視する事も価値のあることである。一つ一つの研究プログラムの進行の様子を監視した結果の概要をまとめたものからプロジェクト全体の進行の様子を概略することにより、業界の発展の様子を特別に記録することができるであろう。そして、発展を推進させるような技術的な問題を監視することにより発展について先取りして論じることができるために、地層処分ヴァーチャル・エンジニアリング・システムの展開が可能になるであろう。

最後に、一般的な情報のデータベースに「索引」サービスを提供できるような一般的なツールが必要不可欠である。このモジュールは、本質的には特に情報を持っている必要はないが、派生モジュールに保持されている情報にユーザーを導いて、その中のデータに迅速にアクセス出来るようにしなくてはならない。

（2）文献モジュール

文献のソフトウェアがさらに発展して、前述したような文書として保存されていない項目に迅速にアクセス出来るようになることが必要とされている。よく使われる製品を予備的に検討すると、地層処分ヴァーチャル・エンジニアリング・システムに適用可能な、価値がある競合製品が数多くあることがわかる。「EndNote」(Reference)は幅広く使われているマルチプラットフォーム型の製品で、非常に柔軟性に富んでいる。それは、主要オンライン及びCD-ROMのソースから参考データを直接取り入れるものである。それはまた特定の定期刊行物のために前もって規定されているひな型もしくはユーザーによって指定されているフォーマットで選択された参考データを決まったスタイルで提供するものでもある。「ProCite」(Reference)は、広くユーザーから支持されている、このクラスでは最も長く定着している製品の一つである。「EndNote」で可能な広範囲のプラットフォーム選択はないが、「ProCite」には同様の機能があるので、これを選択することも可能である。

上記のこれらの製品の適用の主な制限として挙げられることは、特定のデータ項目を検索するために非常に大きなデータベースを操作する際、テキスト列全体と比較して反応が遅いということである。「CAIRS」（食品研究団協会、英国レザーヘッド）は、文献作業上及び関連作業で、英国環境庁が使用しているDOS（唯一の）製品である。その検索手法は迅速で、データエントリーの項目を検索をするのにデータベースフラッグのシステムに依存している。それは、メーカーからの技術援助（商業ベースで）を受ける事が可能なので、非常に柔軟性に富んだ製品であると考えられる。その結果、この

ソフトウェアは（多くの点で多少時代遅れとも言えるが）、英国の放射性廃棄物の管理プログラムにおいて、他のアプリケーション用に開発が進められている。

上記の製品が参考文献や概要の登録を扱うのに適しているのに対して、文献の形では保存されていない資料に迅速にアクセスするのには不向きである。これらのタイプのデータには、ユーザーが必要な情報を迅速に見つけ出すことが出来るように他のツールを開発することが必要不可欠になると考えられる。例えば、地図やフィルムを示している従来の文献データベースに概要を登録したとしても、ユーザーが必要とする情報をハイライトすることはない。概要を拡張して情報を1画面以上にすると、それが読み取られないというリスクが生じる。このために、代替的な技術をこれらのタイプのデータに適用する方法が考えられている。MuMMI（McCRINDLEとDOGGETT 1996年）の様なマルチデータを扱えるツールを適用するとこれらの作業を容易に対処することが出来る。例えば、マップの情報をデジタル保存することにより、ユーザーはウィンドウの中のマップを見ることによって必要な領域に導かれる。ハイパーメディアが他の情報につながることによって、ディスプレイの各「層」においてサイト調査報告、掘削孔あるいは遠隔探査に関するデータを得ることが出来るために、ユーザーはデータを参照したり、時には能動的にデータを呼び出すことが可能になる。同様に、オーディオ、ビデオ、新聞の切り抜きなどが文献のデータベースに機能的に結合されることにより、あるいは（そうすることが適切であるならば）それぞれがお互いに結合することによって、それらのデータを保存することも可能である。このような手法によって、基本的なデータを変更することなく数多くのタイプのデータを機能的に結合したり相互に参照させたり出来るので、この手法の潜在的価値は非常に高いと考えられる。

データ管理の主な改革的前進とは、関連データベース管理システムにおける空間、CAD及びイメージなどのデータを保存し、管理するための空間的データベースエンジンSDE（Spatial Database Engine：ESRI 1995年b）である。SDEは単一のデータベースに他のすべてのデータと共に空間データを保存し、（例えば、Oracle, INFORMIX, SYBASEなど）ユーザーによって限定される空間的ではないデータと目的を透過的にリンクする。SDEは、多数の空間的な特性を管理し、局所的及び広範囲のネットワークにおいて他のどの空間的な技術を適用した場合よりも迅速にデータにアクセスし検索することが出来る。

（3）論争問題データベース

データベースに原子力業界に関連する社会問題や技術的な問題等を記録することは、非常に有効な手段になり得ると考えられる。例えば、ユーザーは一つの組織における特定の問題についての見解がどのように発展しているかについて調べたり、2、3の組織の見解を比較することも可能になる。公衆の合意に関する研究においては、この方法によって、原子力に関して展開している討議について独特の洞察を行ったり、原子力業界において作成された数多くの文献を通して歴史的なつながりを見いだすことが可能になると考えられる。関連団体による報告書の中の重要な文献の概要を作成し、特定の問題に関連するキーワードを記録することによって、上記の要求機能が満たされることになる。このようにして調査すると、ある組織の政策について言及した特定の問題に関連するすべての文献が明らかになると考えられる。

（4）データベースの管理及び索引付け

地層処分ヴァーチャル・エンジニアリング・システムのプロジェクトが進行している間、大量の有効・無効なデータが集められてくると予想される。そのような大量のデータからユーザーが必要なデータを検索することが出来るようにデータを注意深く管理し、適切に検討することが必要である。数値データベースの場合には、この作業は比較的簡単なものになる。それは、問題の記録及び特定のデータセットの検討や最終的な受容については、年代順に記録されることが多いからである。

従って、データベース間の相違点を強調して、どのデータベースが保持されているか（文献目録上の参考資料と共に）をユーザーに知らせたり、特別なことについて調べる時には、特定の現象もしくは重要な反応を示す特定のデータ・エントリーをユーザーに知らせるために、データベースを管理するためのツールが必要となる。

しかしながら、データベースの中にはより複雑な特質を持つものもある。別々に発展し、再び結合するサブセット（部分集合）が生成されることによって、ファミリー・ツリーに複雑な再帰ループが取り入れられる。同じ事がモデル化コードについても当てはまる。モデル化コードにおいては、一人一人の研究者による様々な研究開発によって統合化がはかられ、その結果派生物の数が減少することにつながると考えれる。文献のモジュールから得られるキーワード（及び発行日）を再度利用することによって、研究開発を年代順に追っていくことができなければならない。コードやデータベースの展開についての記録がそれ自体文献として記録されているならば、特定のコードの展開はデータ管理システムを簡単に調べるだけで決定できると考えられる。あるいは、展開の歴史的な過程がこのように文書化されていることをユーザーがすでに知っていると仮定すると、文献データベースに直接アクセスすれば、同じ記録を再生することができると考えられる。

データベース管理モジュールは、主要な骨格部分とみなされ、それによってユーザーは他のデータベースを調べることができる。このようにして、キーワードという形式で調べることによって各データベースにアクセスすることが可能となる。この最も高度なレベルにおいて簡素なブール論理機能を働かせることにより、規模の大きいデータベースを参照する際に必要不可欠とされる選択性を有したデータをユーザーは検索することが出来るであろう。

最後に、能率的で効果的なインターフェイスは、最低２つの理由から必要となるであろう。１つ目は、大量で多種多様のデータベースに容易にアクセスできるようにするため、２つ目は、より多くの研究者、一般の人に、地層処分ヴァーチャル・エンジニアリング・システムを議論のための共通の土俵として提供するためである。インターフェイスの設計は、まず要求される機能によって表現が変わることは言うまでもないが、多くのデータを扱うため、視覚的に分かり易い構成になっている必要があるであろう。データを入力画面では、複雑にそして相互に絡み合うパラメータ群の値範囲の設定や計算範囲の設定等のインターフェイスを通じて行う機能が要求されるであろう。この事は、より範囲の大きな連成問題を扱う際に重要となることが予想される。また、出力画面では研究者や利用者の要求に合わせた結果の表現方法の選択機能が必要である。一般の人には、より分かり易い３次元的表現が要求され、また、正確な値（数字）の読みとりが必要な研究者には、従来のように数字のアウトプットを表示させる等の機能がそれに当たる。

（5）シナリオデータベース

シナリオは、主に、天然自然事象のような別々の事象の影響を分析する目的のために検討することが望ましいと考えられる。しかし、説明的な目的、「～したらどうなるか」という調査を行う際のシステム性能を決定するには、全体的なシミュレーション手法も適用されている。ある一定のシナリオを構築する際に行われた重要な仮定や決定は、連続した事象の複雑性に応じて事象／フォールトツリー、影響ダイアグラムもしくは相互作用マトリックスを用いて図式的に表わすことが出来る。より高い視覚的な効果が望まれたり、あるいは一連のシナリオを相互に調査する必要がある場合には、例えば「ポートフォリオ・エクスプローラ」のような適切なIVA（インタラクティブな視覚化手法）アプリケーションを適用することが出来ると考えられる。相互対応を図式的に表示することによって属性値（この場合にはFEPを構成する要素）や各属性の重要性に基づいて、数多くのシナリオから一群の（ポートフォリオ）シナリオを選択、選別して分析することができるようになると考えられる。

2．7　データベース・システム要件

ここでは、まず最初に、エンジニアリング設計支援用データベース・システムの様々な要件について議論し、その目的にOODBMSがいかに適しているかを明らかにする。次に、CADアプリケーションの場合のRDBMSに対するOODBMSの利点についての考察を行なう。さらに、考慮の対象としたシステムの紹介を行ない、それ以降の節では、その様々な特徴の批判的検討および比較を行なう。最後に、考慮の対象とした各システムの客観的比較順位を示す。

大規模コンピュータ支援用設計またはエンジニアリング（CAD／CAE）は、一般的には、相互間の緊密な対話や設計データ交換および情報の動的シェアリングによって複雑な設計を完成するために、分散形ワークステーションによって協力しながら作業を行なう設計技術者のグループに必要とされる。このような設計環境では、強力なデータ・モデル化、シェアリング、管理ツール、ならびに、高度な対話処理等の同時プロセス管理のため、柔軟性のあるトランザクション・ネットワークが必要になる。このようなアプリケーションの支援のために要求されるデータ管理システムの基本的特徴は、以下のように要約することができる。

（1）複雑な情報のモデル化能力

モデル化および設計しなければならない物理的システムが複雑であるために、エンジニアリング・データ表示は通常は複雑なものになる。設計エンティティも、（関係の構成要素または相補要素のような）様々な機能的または構造的リンクによって、複雑な相互関係を有している。エンジニアリング・システムの設計には、シミュレーションも含まれ、これによって、設計技術者は、最終設計の前に、プロトタイプの挙動テストを行なう。このためには、エンティティ従属性の表示、メッセージ・パッシング、スケジューリングのような、様々なシミュレーション能力が必要である。オブジェクト指向データ・モデルによって、ユーザーは、あらゆるレベルの複雑なデータ構造を極めて柔軟に定義することができるようになる。オブジェクト間のメッセージ・パッシング（情報交換）によって、オブジェクト指向プログラミング（OOP）は、シミュレーションにとって理想的なものとなる。

（2）意味論的スキーマ設計

エンジニアリング・データの複雑さのために、大きなデータベース・スキーマによって、設計方針および階層を反映しなければならない。そうでなければ、維持および拡張に問題が出て来ることになる。これが特に重要になるのが、複雑なオブジェクト相互関係やオブジェクト同士の依存性をスキーマに表示しなければならない場合である。

（3）動的スキーマ展開

エンジニアリング設計アプリケーションの場合には、アプリケーションの停止なしで、データベース・スキーマを、つまり、属性および方法のクラス定義、持越し構造および仕様を動的に定義および修正するに際して、かなりの柔軟性が要求されることになる。これが必要であるのは、設計が増分プロセスであり、時間とともに展開するからである。

（4）厳格な制約管理

エンジニアリング・データベースの規模および複雑さのために、データが展開するにつれて厳格な設計制約を課すことによって、データ状態の整合性を維持しなければならない。整合性制約は、数種類のデータのタイプ、有効範囲、設計条件、安全限界、などをとることができる。このような制約を課す最善の方法は、データ更新の都度余分な整合性のプログラム・チェックを行なうよりは、制約チェック・メカニズムをデータ構造それ自体に組み込んでおくことである場合が多い。オブジェクトの形のデータが、このスキーマを実現するのに極めて便利な方法であるのは、オブジェクトの各属性を更新するためのクラス法によって、更新の都度、制約および整合性チェックを行なうこともできるからである。

（5）大容量データの管理

エンジニアリング設計アプリケーションは、高度データ集約的であり、膨大な量のデータの効率的な管理が必要である。エンジニアリング計算の分散的性格のために、データベースを、ネットワーク全体にわたって遠隔接続された分散形超不均一プラットホームとしなければならない場合がある。設計グループのニーズに応じてデータベースを区分することも可能でなければならない。これについては、簡単に検討を行なう。

（6）データに関するデータ管理

CADは高度データ集約的であり、増分的に展開していくために、DBMSの場合には、設計データ自体に関する入念なデータ記録が必要である。この記録は、所有権、作成の時間およびオブジェクト、更新記録、クライアント従属性（つまり、どのクライアントがデータにアクセスしたか）、ロック状況、バージョン、ならびに、トランザクション管理と関連したいくつかのその他のアドバンスト問題のような情報を含んだものである。さらに、設計データに関するこれらのデータを設計オブジェクト自体の中にカプセル化し、それによって、記録の更新および維持を自動化し、しかも、タスクを、外部メカニズムによって制御するのではなく、様々なオブジェクトに分散することもあると考えられる。

（7）データ・シェアリング

エンジニアリングにおける重要問題の１つが、各設計技術者間のデータ・シェアリングである。所有権、使用、作成の時間およびオブジェクト、データ相互関係、ネットワーク全体にわたる分散、または、その他の有意義なオブジェクトのような判定基準に基いて、データを区分およびグルーピングすることが可能でなければならない。クライアントが利用するデータは、対話も可能でなければならない。オブジェクト自体の中に判別基準がカプセル化されるために、オブジェクト指向データ・モデルによって、クライアント間のオブジェクトシェアリングが容易になる。メッセージ・パッシングによって、設計オブジェクトが、実ワールドで表示するエンティティとして対話を行なうことが可能になる。

（8）データ・バージョニング

異なるバージョンがデータベース内に共存し、データ更新の結果、古いデータへの重ね書きが起こることがないように、データは、バージョニングも可能でなければならない。オブジェクトが、データの効率的な記憶およびバージョン管理の便利な単位となる。データのバージョニングは、同時性を大きく推進する（というのは、様々な設計技術者が、お互いの作業の終了を待つ必要がなく、同じオブジェクトの異なるバージョンで作業を行なうことができるからである）。オブジェクトがそのバージョン記録を記憶することができるために、設計の展開の軌跡をたどるのにも役立つことになる。いずれの段階であれ、設計に欠陥があると思われた場合には、設計を有効なデータ状態までロールバックすることが可能でなければならない。

（9）クライアント間通信

CADデータベース環境の最も重要な要件の１つが、現在のDBMS環境には明らかに欠けている特徴である、設計技術者間の効果的な通信プロトコルである。設計技術者は、相互の開発状況を知らないことが多く、それが、協調の欠如、比較の減少、相当の時間と資源の浪費、さらに、データの解釈の間違いによる欠陥設計を招くことがある。クライアント間通信は、活発な情報交換、データ修正またはデータ状況の更新、設計インターフェイス横断的な作業の同期化の改善、同時性の向上を促進し、これらは何れも協力開発には不可欠なものである。以下の各種の通信モードを含むことができる。

- **ロック通信モード**
  ユーザーに対してオブジェクトのロック状況を知らせ、これによって、もし要求したオブジェクトがたまたま別のクライアントによってロックされている場合には、ユーザーは、何も知らずに待つ必要はなく、それ以外の有益な作業を予定することができる。
- **更新通信モード**
  該当するユーザーに対して、オブジェクトの変更および更新を知らせる。
- **コンフリクト通信モード**
  ２つ以上のトランザクション間のコンフリクト（例えば、デッドロックまたはコミット故障）の場合に、ユーザーに対して通知を行なう。
- **ネゴシエーション・モード**
  相互に合意可能な設計に関するクライアント間の交渉のための、より高度な通信プロトコル。

（１０）柔軟なトランザクション・フレームワーク

コエンジニアリングとは、結果を共同利用することができるように、作業の各単位で対話を行なわなければならない、という概念に基づくものである。これは、同時トランザクションが、そのプロセス中に、結果を相互に伝達可能でなければならないことを意味している。CADのトランザクションは通常は期間が長いために、ロッキング機構は、同時性の向上を可能にするだけの柔軟性を有するものでなければならず、そうであれば、あるトランザクションが終了するまで、処理が相互に待つ必要がなくなるわけである。それに加えて、サブトランザクションを（設計タスクの細目を表示する）別のサブトランザクション内にネストさせ、活動データ・シェアリングのオブジェクトのためにグルーピングする機能が存在しなければならない。全てのこれらのアドバンスト・スキーマは、トランザクションの性格とは無関係にデータベースの整合性を維持する、逐次性という伝統的概念に抵触するものとなっいる。しかしながら、同時性を大きく低下させ、CAD環境には明らかに不適当である。従って、トランザクションおよびデータ・タイプの意味論を考慮に入れたその他の手段によって、データの「正確さ」を保証しなければならない。オブジェクト指向データベースは、このスキーマを使用して、より高い同時性をもたらす機会を提供するが、それは、タイプに関するデータベース・オペレーションがその定義域に関して判定を下すからであり、さらに、単なる読み書きよりは意味論的な内容を有している。

（１１）高速データ・アクセスおよび検索用の効果的な記憶機構

対話形グラフィックス・ベースCADトランザクションは、永続性データに対する高速検索およびアクセスを必要とする。関連データは、様々な判定基準（例えば、機械部分の全てのコンポーネント、または同じデータの全てのバージョン、その他）に応じて、二次記憶装置に隣接集合させることが必要であると考えられる。多くのCADアプリケーションには、グラフィックスおよびマルチメディア情報の利用が含まれ、それには、特別なデータ圧縮、記憶、マッピングおよびアクセス機構が必要である。（オブジェクト見出しおよびオブジェクト間参照に基づく）関連検索が、もう１つの特徴であり、これによって、オブジェクトベースのデータベース内での照会のスピードアップを助けることができる。分散形計算も、異質プラットホーム横断的なネットワークへのデータの分散を必要とすることがある。

（１２）計算面で完璧なデータベース・プログラミング言語

大部分のエンジニアリング・アプリケーションは、データの複雑な数学的操作、または、計算面で完璧または十分に低レベルなデータベース定義およびプログラミング言語を必要とするシステム・レベル・プログラミングを含んでいる。例えば、SQLを使用してエンジニアリング設計を行なうのは困難である。

（１３）互換性、拡張性および統合

エンジニアリング用OODBMSは、より大きな別のCAD環境に統合されるツールとして使用することをオブジェクトとしている。従って、これらのコンポーネントを相互に簡単にインターフェイスさせることが非常に重要である。共通のデータ表示またはデータ操作言語を持つということが、互換性

を実現する１つの方法である。OODBMSシステムは、アプリケーション開発者のニーズに合わせるために、カスタマイジングも可能でなければならないと考えられる。

（１４）グラフィック環境

最重要要件ではないが、大部分のCADアプリケーションの場合には、グラフィック環境に対する支援が必要である。これは、データ構造および従属性の操作検索と修正、エンジニアリング図面およびチャートの表示、さらには、アプリケーション開発の全般的な簡素化に有益である。

２．７．１　オブジェクト指向データベース管理技術

オブジェクト指向データベース管理システム（OODBMS）の分野は、いくつかの個別の研究の収斂の結果として登場してきたものである。オブジェクト指向プログラミング言語、人工知能、ソフトウェア・エンジニアリング、データ抽象化理論および複雑データ管理という分野の全てが、データベース領域でのオブジェクト指向技術の利用に貢献している。OODBMSを必要としてきた応用領域としては、大規模コンピュータ援用設計・エンジニアリング（CAD／CAE）アプリケーション、分散形協力計算、コンピュータ支援ソフトウェア・エンジニアリング（CASE）、オフィス情報システム（OIS）がある。OODBMSの開発は、コンピュータ援用設計を支援する高性能グラフィックス・ワークステーションおよびネットワーキング環境が利用可能になった結果、促進されたのである。

OODBMSは、従来のデータベース・システムの全ての機能、つまり、以下の機能を提供するデータベース管理システムである。

データモデル

情報をモデル化するために使用されるテンプレートまたは構造のセット、それらを操作するためのオペレーションのセット、および、データの整合性状態を定義するための制約のセットを提供する。

永続性

データは、非持久メモリではなく、永続記憶装置に常駐しており、複数のセッションを横断的に使用することができる。

同時性

複数のユーザーが、同じデータベースへのアクセスおよび使用を同時に行ない、相互に対話することもできる。

トランザクション管理

データの整合性または正確さを保証するためにデータベース相互関係をモニターするプロセス。

回復

クラッシュから定義された安定状態への回復。

照会言語

情報に体系的にアクセスするための高レベルで、使いやすい言語。

パフォーマンス

大量の永続性データを二次記憶装置から検索するための効率的なアクセス構造およびアルゴリズム。

機密保護

無許可アクセスからの情報の保護。しかし、OODBMSはオブジェクト指向システムでもある。

データ抽象化

これによって、抽象的で、論理的に複雑かつ柔軟なタイプのデータの開発および利用を行なうことができる。

強力な情報モデル化能力

情報が、コンピュータ環境内での実ワールド・エンティティの構造、および挙動を表示かつ補足するクラスおよびオブジェクトの形で、モデル化され、シミュレーションおよび設計オブジェクトに理想的なものとなる。

オブジェクト見出し

このシステムは、オブジェクトに関する識別名を定義し、維持する。これによって、「等しい」オブジェクト（同じ属性および等しい属性値を有するもの）が共存することが可能になる。ユーザーは、エンティティ事例毎に単一のキーを定義する必要がなくなる。

カプセル化およびデータ隠し

オブジェクトは、そのタイプに関して定義されたオペレーションによって操作され、その中身は外部アクセスから隠されている（あるいは、特定の言語実現の場合のように、アクセス不能なこともある。これらのオペレーションの実現は、その使用を無効にせずに変更することもできる。

活動（インテリジェント）データ

データとともに手順をカプセル化することによって、その定義域、整合性、有効性、その他を判定する能力が与えられる。これは、インテリジェント・オペレーションのためのデータベースに関するトリガを定義する能力を強化することになる。

持越し

これによって、既存のものとの関連における新しいクラス構造の再使用および／または増分再定義が可能になり、導出オブジェクトに対する情報の「継承」ができるようになる。以下の２種類の持越しがある。

特殊化

エンティティ構造および挙動は、クラス導出によって、継続的に特殊化することができる。

一般化

共通の属性および挙動を有する（つまり、類似の）エンティティは、共通のスーパークラスから共通の特性を持ち越すために、一般化することができ、一方、類似性のない属性は、各子クラス内で特殊化することができる。

同質多形データおよび機能

データが、実行時間中に決定される様々な形態を動的に仮定するを可能にする特徴である。

構成

オブジェクトは、他のオブジェクトの一部とすることもできる。これは、セットおよびアレイのような群オブジェクトを基準とする集合とは異なっている。

メッセージ・パッシング

相互の方法の呼出しによるオブジェクトの対話ーシミュレーションを強化する特徴。

拡張性

アプリケーションに対して使用可能なオペレーション、構造および制約のセットは、伝統的なDBMSと違って、固定されていない。ユーザーは、アプリケーションに対する増分として、新しいオペレーションやタイプを定義することができる。要するに、DBMS機能と組み合わされたオブジェクトデータ表示の柔軟性とパワーが、エンジニアリング情報のモデル化、調整、記憶、操作のための強力な手段となっている。

２．７．２　RDBMSとOODBMS

伝統的なリレーショナル・データベース・システムの設計は、典型的なビジネス・ニーズに合わせて、多くが決定されてきた。これらのシステムは、テーブル作成が可能で（従って、固定構造を有し）、非常に高度の照会言語によるアクセスや修正が可能なフラットなデータを記憶や操作するのには優れた装備を有する。しかしながら、リレーショナル・データ内の構造、オペレーションおよび制約のセットは限定され、固定されている。その結果、アプリケーションが必要とするあらゆる構造やデータを、この限定されたセットの中にマッピングしなければならない。アプリケーションが次第に複雑になるにつれて、このマッピングは面倒で、非現実的なものに成ってしまうのが一般的である。

コンピュータ支援用設計アプリケーションは、高度データ集約的であり、設計しなければならない複雑なエンティティの構造や挙動を模擬するために、複雑なデータ表示を含むものとなっている。エンティティ間の関係は、複雑または微妙であり、リレーショナル・モデルによってモデル化することはできない。それに加えて、相当な量の推論知能が、実現性、整合性、または、設計プロセスが課すその他の制約の形で、設計データと結び付けられている。これらの制約は煩雑であり、リレーショナル・システムで使用される制約のシステムを使用した場合には、実現が不可能なこともある。要するに、フラットな「ダンプ」関係およびRDBMSの関連制約の形で表示するには、エンジニアリング情報がそれほど簡単ではないことが多いのである。従って、エンジニアリング・データが複雑であることから、アプリケーション専用データ意味論、関連システムが貧弱なツールであるにすぎないデータベース構造の増分開発のメカニズムを補足するために、適正なレベルの拡張性が必要となる。以下に、RDBMSに対するOODBMSに利点を示す。

（１）よりな現実的データ・モデル

OODBMSにおける設計クラスやオブジェクトは、実世界の設計エンティティを表示し、RDBMSの場合のようにフラットなテーブルを作成するよりは、問題の技術的側面をはるかに明瞭に実感させる。さらに、リレーショナル・モデルには、挙動（機能）をデータと関連付けるメカニズムがないが、これが、オブジェクトによって容易に実現される。

（2）より強力なデータ・モデル

オブジェクト・データ表示は、柔軟性が高く、ほとんど制約なしで、ユーザーがカスタマイズすることができる。データを同質多形で高度化することが可能になる。持越しやスキーマ展開の機能によって、設計を増分発展させることが可能である。データの正規化や不要な重複のような、RDBMSで通常発生するような問題は存在しない。オブジェクトモデルが取扱うことのできるデータの複雑さや関係の複雑さの方が、リレーショナル・モデルの場合よりもはるかに優れている。

（3）より容易なスキーマ開発

OODBMSにおいては、一般化や持越しによって、スキーマをより構造の優れた、より直感的なものにし、スキーマがアプリケーションの意味論を補足することが可能である。さらに、共通の属性を除外することができるために（一般化）、持越しによって、スキーマもより小さくなる。設計環境は連続的な変化に特徴があるので、設計技術者が問題に対する理解を深めるにつれて、タイプやスキーマ定義を修正するのに適している。従来のRDBMSシステムは、変更をタイプのレベルで適応させるという限定的な機能だけしか有していない。OODBMSはもっと柔軟性があり、いくつかのシステム（例えば、ORION）は広範なスキーマ展開機能を提供する。

（4）RDBMSにおけるインピーダンス不整合

RDBMSを使用して複雑なアプリケーションを開発する際の問題の１つが、データベース操作言語（ＤＭＬ）と、アプリケーションの残りの記述に使用される汎用プログラミング言語との間のインピーダンス不整合である。この不整合には、以下の２つの局面がある。

- 例えば、SQLのような宣言言語とCのような命令プログラミング言語との間のような、プログラミング・パラダイムの相違。
- タイプ・システムの不整合で、これによって、インターフェイスで情報の損失が起こる。大部分の関連システムにおいては、ＤＭＬは、エンジニアリング設計に共通の複雑な数学的操作を表現するための計算完全性を欠いている。一方、大部分のOODBMSは、有限要素解析のような大規模プログラムに典型的な複雑さを取扱うことが可能なプログラミング言語（例えば、C++、Clos、Smalltalk）を計算的に完成するための拡張データベースを提供する。

（5）オブジェクト見出し

関連データベース・モデルは数値ベースであり、同じ（または等しい）数値を２つ以上の関連オブジェクトに組み込むことによって、２つのオブジェクトの関係を表現する。ユーザーは、類似のエンティティを区別するための単一キーを定義しなければならないことが多く、その結果、データが重複することになる。オブジェクト指向システムは見出しベースであり、２つ以上のオブジェクトをその組み込み数値とは無関係に関係付けることができる（もし必要があれば、OODBMSには、数値ベース・フレームワークも組み込むことができる）。見出しは、いたるところにコピーを保持するのではなく、様々なポイントから同じオブジェクトを参照することによって、シェアリングを容易にする。

（6）強力な統一的知識表示

情報のオブジェクト指向表示は、データ（属性）と知識（手順）を統合するための均一な枠組みを提供する。OODBMSには、いくつかの強力な人工知能技術が登場する。実例としては、階層別の分類や特殊化（持越しが、各関係を正確かつ簡潔に補足し、知識の伝播を可能にする）、挙動の委任、方法へのメッセージの同質多形や実行時間結合、自動不要情報収集やメモリ管理戦略、ならびに、エンティティのインテリジェント挙動がある。大部分のエンジニアリング・データベースは、独立のものとしてではなく、例えば、グラフィックス・パッケージやエキスパート・システムを備えた、より大きなCAD環境に統合されるツールとして使用することをオブジェクトとしたものである。データ構造としてしてオブジェクトを利用することによって、エンティティ・定義域知識がオブジェクト自体の中に組み込まれているために、「知識」と「データ」の区別がファジーになり、別々の表示が不要になる。RDBMSにはこれに匹敵する機能はない。OODBMSの方が、エキスパート、またはインテリジェント・データベースのような、次世代統合ツールの開発に際して決定的に重要な役割を演じるための装備が優れている。

（7）より優れたトランザクション管理および同時性

現在のRDBMSは、データベースの整合性を維持するために、同時トランザクションの逐次性を強制する。OODBMSは、伝統的なアプローチによって可能であるよりも高い同時性をもらす機会を提供する。これは、オブジェクト指向アプローチの場合の方が、データベース・システムが、実行中のオペレーションについてより多くのことを知っているからである。単なる読み込みまたは書き込みではなく、むしろ、より大きな意味論を有している。従って、読み書きコンフリクト・アプローチを使用するのではなく、アプリケーション意味論に基づいて、「正確さ」を強制することができる。

（8）より表現度の高い照会言語

大部分のRDBMSは、素晴らしい標準化高レベル照会言語を支援する。現在のOODBMSは、高レベル標準照会言語を欠いている。しかしながら、様々なオブジェクト指向照会形式が現在開発中である。オブジェクトSQL（またはOSQL）は、SQLに対するオブジェクト指向アプローチである。OSQLは、オブジェクト参照によって、照会ステートメントの形での機能呼出し、連想検索を可能にする。これによって、単一照会は（例えば、呼出し機能によって複雑な計算を行なうことによって）、限定されたセットの複雑なオペレーションしか行なうことができない標準SQL照会よりも、はるかに多くのことができるようになる。オブジェクト識別名や照会の機能的構成は、SQL照会の場合には必要なことがある不経済なジョインも不要にすることがある。それに加えて、オブジェクト間参照（「ポインター・チェーシング」という呼び方に人気がある）によるデータへのアクセスは、関連リンクによるナビゲーションよりもはるかに有効である。

（9）より優れた協力作業支援

OODBMSは、協力作業用の環境支援の候補としては、RDBMSよりも優れている。OODBMSは、より複雑なデータ取扱いを可能にし、多数の細分性レベルで人工品のシェアリングの支援を行ない、バージョンや代替案の支援をよりうまく行ない、さらには、多人数プロジェクトに伴う大量の管理情

報ー例えば、スケジュール、タスク従属性、注釈、記録、設計経歴、設計決定、さらには、プロジェクト固有方針ーを補足することができる。RDBMSおよびOODBMSの有効性の比較解析の事例研究が、簡単なコンピュータ統合製造システムについて行なわれている。その結果によれば、オブジェクト指向プロトタイプの方が優れたスキーマであり、情報への便利なアクセスを提供することが可能であり、拡張および維持もより容易である。

2．7．3　市販OODBMSシステムの特徴

考察の対象としたシステムは、ORION、ObServer／ENCORE、GEMSTONE、ONTOS、ObjectStoreおよびVERSANTである。ORION／ITASCA、GEMSTONE、ONTOS、ObjectStoreおよびVERSANTは市販品であり、一方ENCOREは研究プロトタイプである。ONTOS、GEMSTONEおよびENCOREは、エンジニアリング・アプリケーション開発用として、IESLにおける広範な使用実績があり、さらに、ObjectStoreおよびVERSANTは評価用として使用された。ORIONはIESLにおける使用実績はないが、本研究に含めたのは、その完成度と極めて高度な特徴のためである。従って、その他の既存のシステムの評価用のベンチマークとして使用することができるのである。

（1）ORION／ITASCA

ORIONは、1985年にマイクロエレクトロニクス・アンド・コンピュータ・テクノロジー・コーポレーション（MCC）の高度コンピュータ・テクノロジー（ACT）計画によって開始された一連の次世代データベース管理システムから構成されている。ORIONは、AI、CAD／CAMおよびオフィス情報システムのようなシステムにおけるアプリケーション用にオブジェクト指向プログラミング言語とハイエンド・データベースを統合するための研究手段である。現在、ORIONシリーズは、以下の3つのシステムを含んでいる。ORION-1、単一ユーザー多重タスク処理システム。ORION-1 SX、ローカル・エリア・コンピュータ・ネットワークで働くように設計されたクライアント・サーバー・システム。ORION-2、完全分散形DBMS。現在、ORIONが利用可能であるのは、ACT計画の企業スポンサーに限られている。ITASCAという名前のORIONの市販バージョンが、現在イタスカ・システムズ社から発売されている。

（2）ObServer／ENCORE

ObServerは、米国ブラウン大学で開発中の汎用オブジェクトサーバである。二次記憶、クラスタリング、オブジェクトロッキングおよび各種のトランザクション・プロトコルの管理を行なう。ENCORE（拡張可能および自然共通オブジェクト資源）は、ObServer用のフロントエンド・データ定義および操作環境であり、オブジェクトデータ・モデルおよびその機能性を定義する。

（3）GEMSTONE

GEMSTONEは、セルヴィオ社が開発した市販のOODBMSである。最初期の市販のOODBMSの1つである。生成されるオブジェクトに対して永続性およびデータベース機能を与えるために、Smalltalkに類似したプログラミング・パラダイムを拡張する。

（4）ONTOS

ONTOS、かってのVBaseは、オントロジック社が開発した市販のOODBMSである。ONTOSは、オブジェクト指向プログラミング言語として、C++に生成されたクラスおよびオブジェクトに対して永続性を与え、その機能のいくつかを拡張する。

（5）ObjectStore

ObjectStoreは、オブジェクト・デザイン社が開発した市販のOODBMSである。C++をフロントエンド・データ定義および操作言語して使用し、さらには、C++を数種類の方法で拡張する。

（6）VERSANT

VERSANTは、ヴァーサント・オブジェクト・テクノロジー社が開発した市販のOODBMSである。VERSANTは、C++クラスおよびオブジェクトに対して永続性およびデータベース管理機能を与える。さらには、C「クラス」の定義および「オブジェクト」の生成を可能にするCプログラミング言語に対するアプリケーション・インターフェイスを有している。

２．７．４　市販ＯＯＤＢＭＳシステムの比較

比較のオブジェクトで考察の対象とする主要な特徴としては、実現言語およびオペーレーショナル・プラットホーム、外部言語インターフェイス、ポータビリティ、データ・モデルおよびデータ抽象化のパワー、動的スキーマ展開機能、トランザクション管理の柔軟性、共同作業のための資源、通信プロトコル、照会、データベース機密保護、ユーザー・インターフェイス、ならびに、マルチメディア能力がある。

２．７．４．１　実現言語、プラットホームおよび外部インターフェイス

（1）ORION／ITASCA

ORION／ITASCAは、コモンLISPで実現されており、Symbolics 3600　LISP機で最もうまく動作する。最近、UNIXをランするSUN3ワークステーションにポートされた。ORIONは、LISPプログラミング言語とのインターフェイスが容易である。ITASCAは、Sun、Apollo、Hewlett-Packard、Silicon Graphics、DECStation、その他を含めた大部分のUNIXシステムで動作する。LISPに加えて、ITASCAはCに対してインターフェイスする。さらに、ITASCA法へのFORTRANコードの動的ロードの支援も行なう。

（2）ObServer／ENCORE

ObServerもENCOREも、Cを使用して開発されたものであり、Sun 3、Sun 4およびその他のUNIXワークステーションで動作する。ただし、このシステムはまだテストおよび完全なデバッグを行なう必要がある。しかしながら、カスタマイジングおよび強化のために、ソース・コードが利用可能である。ENCOREはCにインターフェイスする。その他の言語へのインターフェイスは現在開発中である。

（3）GEMSTONE

GEMSTONEは、Cを使用して開発されたものであり、Sun 3、Sun 4、Apolloのような大部分のUNIX環境、DEC（VAX/VMS）、Tektronixおよびその他のミニコンピュータで動作する。Smalltalkに対して容易にインターフェイスする（GEMSTONE－Smalltalkインターフェイスは、グラフィックであり、Smalltalkユーザー・インターフェイスと類似している。インターフェイス・ルーチンのセットによって、C言語を使用して、OPALオブジェクトに対するアクセスおよび操作を行なうこともできる。しかしながら、OPALオブジェクトとCデータ構造の結合はゆるやかであり、一方から他方への翻訳は面倒である場合が多い。それに加えて、クラスおよびオブジェクトは、OPALだけでしか生成することができず、Cインターフェイスによる生成は不可能である。C++インターフェイスが現在開発中である。しかしながら、OPALおよびC++クラスとオブジェクトとの間のマッピングの意味論は明確ではない。

（4）ONTOS

ONTOSは、C++を使用して開発されたものである。Sun 3／4、ApolloのようなUNIX環境で動作し、現在VAX/VMSおよびOS/2プラットホームに対してポート中である。ONTOSは、C++をフロントエンド言語として使用し、C++のパワーおよびポータビリティを提供し、さらに、その能力のいくつかを強化する。

（5）ObjectStore

ObjectStoreは、C／C++を使用して開発されたものであり、ＡＴ＆ＴのC++支援する。C++を数種類の方法で拡張する。例えば、（プログラム・ヒープにも物理的記憶を配分する）C++新演算子の永続性バージョンを提供する。さらに、C++とともに使用することもできる（永続性オブジェクトを生成し、物理的記憶を配分する永続性演算子のような）新演算子の導入も行なう。大部分のUNIXプラットホームで動作し、Sun 3／4、Apollo、DEC（Ultrix）、HPのような異質なクライアントの支援を行なう。

（6）VERSANT

VERSANTは、C／C++で実現される。現在のところは、SunOSを有するSun 3およびSun 4だけでしか動作しない。VERSANTは、C／C++とC言語のいずれに対しても、アプリケーション・プログラミング・インターフェイスを有している。

２．７．４．２　基本データモデル

各システムのオブジェクトデータ定義および操作言語（DDML）の基本的特徴の検討を行なう。多重持越し、複合オブジェクトおよびスキーマ展開のような、より高度な特徴については、以降の節で検討を行なう。

（1）ORION／ITASCA

ORION／ITASCAのクラスおよびオブジェクトは、LOOPS（LISPオブジェクト指向プログラミン

グ・システム）に類似した言語によって、注釈的に定義されている。それは、オブジェクトの中の共用値（全ての場合に同じ値になる）および省略時値（もし指定がない場合には、省略時値を仮定する）変数を支援する。ORION／ITASCAは、データのタイピングと非タイピングの両方を支援する。さらに、集合体、セット、コレクション、その他のような、ユーティリティ・クラスのライブラリの供給も行なう。ORION／ITASCAは、登録ユーザーがInternetを通じて遠隔地からアクセスすることもできるObject SHare Libraryと呼ばれる再利用可能オブジェクトの共用リポジトリィを支援する。

（2）ObServer／ENCORE

ENCOREのタイプは、データ抽象化、持越し、オペレーション（メンバー機能）および特性（メンバー属性）を規定するカスタム言語によって定義されている。タイプは、設計オブジェクトまたは事例の形で裏付けることができる。ENCOREにおいては、クラスという用語は、特定のタイプの事例のセットに相当する。全てのオブジェクトは永続性があり、もしバージョン変更可能なクラスから導出された場合には、固有にバージョン変更可能なものとすることもできる（バージョン管理については、後続の節で検討を行なう）。ENCOREのデータは協力にタイプ化されており、タイプ・チェックがコンパイル時に行なわれる。

（3）GEMSTONE

GEMSTONEのDDMLは、Smalltalkに極めて類似したOPALである。Smalltalkの場合と同様に、OPALは純粋オブジェクト指向注釈言語であり、そこでは、全てのデータがオブジェクトである。OPALのオブジェクトは、事例変数（その値は事例に対して局所）およびクラス変数（全ての事例によって見られる）を支援する。OPALは、Cほどには強力にタイプ化されておらず、タイプ・チェックが動作時に行なわれる（データ属性のタイプの制約を課すことによって、コンパイル時タイプ・チェックを行なうこともできる）。セット、連関、アレイ、バッグ、その他のような、いくつかの核クラスが、アプリケーション開発のために供給される。

（4）ONTOS

ONTOSは、C++をDDMLとして使用し、従って、ハイブリッドオブジェクト指向プログラミング・システムを供給する。クラスおよび機能は、ONTOSのクラスをポータブルにする標準C++コードによって定義されており、（その他のアプリケーションからの）既存のC++コードを容易にONTOSに組み込むこともできる。C++クラスは、静的データ（あるクラスの全ての事例によって見られる）および静的機能メンバー（裏付けなしであるクラスから直接呼び出すこともできる）を支援する。それに加えて、C++は、（私用および保護変数の形での）データ隠しによって、真のカプセル化を可能にする。検討対象となったその他の言語とは違って、C++は静的プログラミング言語であり、柔軟な動作時環境を供給することはない。C++は、強力にタイプ化された言語でもある。ONTOSは、アプリケーション開発用のC++ユーティリティ・クラスのライブラリを供給する。

（5）ObjectStore

ObjectStoreは、パラメータ化クラスを含めて、C++の全てのデータ・モデル化機能を支援する。さ

らに、複合オブジェクトおよびコンフィギュレーションの若干の概念も支援する（後に検討を行なう）。ObjectStoreは、アプリケーション開発用のC++ユーティリティ・クラスのライブラリを供給する。

（6）VERSANT

VERSANTで使用されるDDMLは、CでもC++でもよい。Cインターフェイスに装備されているインターフェイス機能は、C++に類似した生成および操作データベースオブジェクトに対するオブジェクト指向アプローチを供給する（カプセル化および多重持越しを支援する）。さらにVERSANTは、NIHクラスのクラスに基づくC++クラス・ライブラリを供給する。照会方法およびバージョン管理機能も利用可能である。

2．7．4．3　多重持越し機能

（1）ORION／ITASCA

ORION／ITASCAは、名前コンフリクトの解決のために、いくつかの戦略によって多重持越しを支援する（2つ以上のスーパークラスが同じ名前の属性を有する場合）。

・選考持越し。ユーザーは、持越しのために、スーパークラスを選択することもできる。
・持越し後再命名。相反する名前を有する持越し変数は、派生クラスにおいて再命名することもできる。

（2）ObServer／ENCORE

ENCOREは多重持越しを支援するが、しかしながら、（同じ名前の持越し属性に関する）名前コンフリクト解決戦略は明確ではない。

（3）GEMSTONE

OPALは多重持越しを支援しない。

（4）ONTOS

C++は多重持越しを支援する。名前コンフリクトの場合には、前のベース・クラス属性がその他をオーバーライドする。C++における多重持越しとの関連でのコンフリクト解決のための様々なルールで検討が行なわれている。ONTOSは、多重持越し永続性クラスを支援する。しかしながら、仮想および私用ベース・クラスは許容されない。

（5）ObjectStore

ObjectStoreは、C++に実現された形の多重持越しを完全に支援する。

（6）VERSANT

VERSANTは、C++に定義された形の多重持越しを支援する。しかし、多重持越しを可能にするためには、VERSANTのベース・クラスであるObjectから、全てのクラスを持ち越さなければならない。

２．７．４．４　複合オブジェクト機能

エンジニアリング設計では、設計エンティティが、いくつかのコンポーネント・エンティティまたは部分から複合または構成されている場合が多い。例えば、自動車はいくつかの部分から構成され、その各部分自体が小部分を有している。オブジェクト指向パラダイムにおいては、そのようなエンティティは、オブジェクトのセットが単一論理エンティティとして取扱うことができる別のオブジェクトに所属する（またはその部分である）複合オブジェクトとして有効にモデル化することができる。複合オブジェクトは、複合オブジェクト階層と呼ばれる排他的コンポーネントオブジェクトの階層を有するオブジェクトである。

（１）ORION／ITASCA

ORIONデータ・モデルは、複合挙動、スキーマ、持越し、ロッキング、コンフリクト解決、オブジェクト従属性、その他を定義するための詳細な意味論によって、オブジェクト間におけるの部分関係を支援する。ORIONにおいては、複合オブジェクトは、複合クラス階層の事例である。その階層は、ルート・クラス（例えば、自動車）、および、いくつかのコンポーネント・クラス（例えば、車体、エンジン、車輪、その他）から構成されており、コンポーネント・クラスは、サブコンポーネントを有していてもよい（例えば、車体は、ドア、ウインドシールド、その他のようなコンポーネント・クラスを有していてもよい）。複合オブジェクトは、これら全てのクラスの事例から構成されている。コンポーネントオブジェクトは、その存在が全体に依存しており、複合オブジェクトが削除された場合には、全てが削除される。ORIONが提供する高度複合オブジェクトモデル化特徴の幾つかは以下の通である。

・数値伝搬

これは、コンポーネントオブジェクト間における事例変数の値のシェアリングを可能にする。例えば、自動車の部分であるドアの（事例変数である）省略時カラーは、自動車の場合または車体の場合と同じになる。数値伝搬はユーザー特有であり、スーパークラスからの持越しをオーバーライドする。

・クラスタリング

全てのコンポーネントに直ちにアクセスするのが適当であるので、ORIONは、複合オブジェクトを、高速アクセス用ディスクにおける関連オブジェクトのクラスタリングのための単位であると考える。

・スキーマ再定義

あるオブジェクトを、複合オブジェクトから（独立の存在を有する）非複合オブジェクトに、あるいは、その逆に変更することもできる。コンポーネントオブジェクトを、複合オブジェクト間において交換することもできる。

・ロッキング

８タイプの特別ロックのセットが、複合オブジェクトロッキング用に装備されている

また、コンポーネントオブジェクト間に、実行時参照、複合参照、複合排他的および複合従属排他的のような関係を与えるために、ITASCAがORIONの複合オブジェクトモデルを拡張する。

（2）ObServer／ENCORE

ENCOREは、オブジェクトおよび特性間におけるの部分関係を支援する。さらに、数値伝搬（回復機能委任と呼ばれる）、バージョン管理、および、複合オブジェクトのロッキングに対して、意味論も供給する。

（3）GEMSTONE

OPALは、複合オブジェクトの概念の明示的支援は行なわない。しかしながら、あるオブジェクトを別のオブジェクトのメンバー属性とすることはできる。

（4）ONTOS

GEMSTONEの場合と同様に、ONTOSも形式的には、オブジェクト間におけるの部分関係の支援は行なわない。しかしながら、あるクラスが、そのメンバー属性の１つとして、別のクラスの事例を有することは可能であり、その事例がそのクラスの範囲だけでしか利用できないために、それをそのままの状態で「所有する」。さらに、ONTOSは、データベースの中でC++クラスを表示するオブジェクトであるそのタイプオブジェクトインターフェイスを通じて、逆関係の定義も可能にする。

（5）ObjectStore

ObjectStoreは、オブジェクト構成およびコンフィギュレーションの若干の概念を供給する。さらに、オブジェクト間の構成関係をモデル化し、そこでは、各関係は、特定の形で相互に整合性を有するように制約を課せられているオブジェクト・データ・メンバーから構成されている。関係を構成しているデータ・メンバーに関するこれらの制約は、ユーザーが宣言を行なう。このようにして、ObjectStoreは、オブジェクト間に参照完全性を確立する。さらに、オブジェクト間において逆関係の支援も行なう（例えば、もしあるオブジェクトが削除された場合には、逆関係ポインターも削除され、それによって、ダングリング・ポインターが取り除かれる）。の部分関係は、コンポーネントに対する組込みポインター（子部分と呼ばれる）および含有オブジェクトに対するポインター（親部分と呼ばれる）を含むクラス部分によって実現される。コンポーネントのデータ属性間の挙動制約の仕様は、ユーザー定義でなければならず、暗黙の取扱いをObjectStoreが行なうわけではない。

（6）VERSANT

C++クラスの属性は、別の事例に対するリンク（ポインター）である、または、別の事例（組込みオブジェクト）を含むと宣言することができる。VERSANTにおいては、オブジェクト・リンクは、特別の構文によって、つまり、リンクされたオブジェクトを保持するために使用される生成クラスである<タイプ>リンクによって宣言される。リンク・クラスは結果として無オブジェクトのレベルになるが、リンクを横断する際の透明性を高めることになる。基本的には、C++におけるメモリ・ポインターの使用と類似している。親事例の削除が、集合リンクによってリンクされた事例の削除につながるという点が、オブジェクトリンクとは異なっている

２．７．４．５　動的スキーマ展開機能

エンジニアリング設計アプリケーションの場合には、データベース・スキーマ、つまり、属性および方法のクラス定義、持越し構造および使用を、アプリケーション遮断の必要なしで、定義および修正するためには、相当の柔軟性が要求される。これが必要であるのは、設計が増分プロセスであり、時間とともに展開するからである。スキーマ修正は、バージョン管理の問題とおよそ以下の通に密接に結び付いている。スキーマに対する典型的な変更には、以下のものが含まれる。

・ノードの内容（クラスまたはその事例）に対する変更。
・エッジ（つまり、クラス間の関係）に対する変更。
・クラス格子の中のノードに対する変更。

２．７．４．５．１　ノードの内容に対する変更

（１）事例およびクラス変数に対する変更

事例変数は、どんな事例によって持越しおよび修正を行なってもよい、一方、クラス変数の値は、持越しはできるが、事例またはサブクラスによって変更することはできない。設計技術者がクラスおよび事例に対して行っても構わない変更のタイプは、以下の通りである。

・新しい事例変数を加える

追加設計情報を保持するために、新しいスロットを加えてもよい。例えば、床スラブを支持する梁の数を、事例変数として、クラス床に加えることもできる。

・事例変数を落とす

設計が進んだ段階では、若干の情報が冗長または不要になることがある。従って、設計オブジェクトのサイズを減らすために、その情報を保持するスロットを削除してもよい。例えば、もしスロットｘを含むクラスが、同じデータを含む別のクラスのサブクラスまたはコンポーネント・クラスにされる場合には、全ての個々の事例に記憶装置を割当てるのではなく、持越しによって、情報を検索してよい。従って、スロットｘは削除してよい。

・事例変数の名前を変える

例変数を別の名前によって識別することが必要になることが多い。持越し変数による名前コンフリクト、または、将来の参照を容易にするためだけの名前コンフリクトが存在することがある。例えば、ユーザーは、梁オブジェクトのWポンド単位重量スロットを重量に変更すことを希望することもできる。

・変数の定義域を変更する

変数（スロット）が記憶することのできる情報のタイプ（例えば、整数、実数、文字、その他）を変更する必要が存在することがある。

・事例変数の省略時値を変更する

もし設計仕様をいずれかの段階で変更しなければならない場合には、これが必要であると考えられる。例えば、全てのコンクリート梁の設計の完了後に、設計技術者が以降に例示するオブジェクト毎に数値を設定しなくても済むように、全ての梁オブジェクトの変数材料ータイプの省略時値を鋼に設定してもよい。

・事例変数の持越し仕様を変更する

設計技術者が、スロットのために導出された情報をオーバーライドし、自分独自の仕様を指定することを希望する場合には、これが必要である。さらに、複数の親が同じ名前のスロットを有する多重持越しの場合には、ユーザーは、希望のスーパークラス（スーパータイプ）からスロットを持ち越すことを選択することもできる。例えば、鉄筋ーコンクリートー梁およびプレストレストーコンクリートー梁が梁のスーパークラス（親）であり、梁の材料スロットの省略時値に関する持越し仕様が鉄筋ーコンクリートー梁に設定されていると仮定する。比較的遅い段階であれば、この持越し仕様を鉄筋ーコンクンリートー梁からプレストレストーコンクリートー梁に変更することも可能であり、その結果、材料はプレストレストーコンクリートー梁から省略時値を持ち越すことになる。

・クラス変数を加える

これによって、クラスが設計された時には想定されていなかった新しい変数を加えることが可能になる。例えば、新しい変数（スロット）鉄筋ー率を梁オブジェクトに加えることができる。

・クラス変数を落とす

設計オブジェクトにとってもやは不要であると考えられた場合には、クラス変数をが落とすこともできる。

・クラス変数の省略時値を変更する。

これが必要であるのは、設計仕様がある程度遅い段階で変更され、この変更をクラスの全ての事例に知らせる必要がある場合である。

（2）方法に対する変更

・増分設計開発を可能にするために、新しい方法をクラスに加える。様々なサブコンポーネントの解析または設計のための手順を随時加えることもできる。例えば、（特に予備設計中に）予備コスト見積を行なう方法が存在していてもよい。後になって、詳細コスト見積を行なう方法が利用可能になっても構わない。これらの方法は、該当する設計オブジェクトに加えることができる。
・もやは役に立たなくなった場合には、既存の方法を落とす。
・識別を容易にするため、あるいはその機能性が変化した場合には、方法の名前を変更する。
・方法の原始コードを変更する。これが行なってよいのは、エラーが存在する場合、あるいは、何か別のタスクを実行するためにその方法が再実現される場合である。
・方法の持越しを変更する。変数の場合に類似した環境の下で、これを行なうことができる。

2．7．4．5．2　エッジに対する変更

これらの修正は、持越し格子の中のクラスの階層の変更に関するものである。それには、以下のものが含まれる。

・あるクラスを別のクラスのスーパークラスにする。

これを行なうのは、いくつかのクラスの共通の属性を1つのスーパークラスに抽象化するため、または、新しいサブクラスがいくつかの追加属性を新しいスーパークラスから持ち越すことを可能にするためである。例えば、クラス梁の全ての事例に対してグラフィック能力を与えるために、梁を、オブジェクトのグラフィック・ディスプレイのための手順を含んでいる幾何学的ーObjnoサブクラスにすることもできる。

・クラスのスーパークラスの１つを除外する。
これを行なってよいのは、ある特定の持越しスロットが必要であるとは感じられない場合か、あるいは、別のスーパークラスから持ち越された属性とのタイプ・コンフリクト状態にある場合である。

・クラスのスーパークラスの順序を変更する。
これは、クラス階層の順序の再シャフリングを含むものであり、設計仕様の大きな変更を反映するために行なわれる。

２．７．４．５．３　クラスに対する変更

クラス・タイプに対する以下の変更を支援する必要がある。

・新しいクラスを加える。
これを行なうのは、設計の任意の段階で、新しい設計オブジェクトのクラスを導入するためである。例えば、梁オブジェクトのサブクラスとして、新しいクラスＴ－梁を加えることもできる。

・既存のクラスを落とす。
これが行なわれることがあるのは、名前コンフリクトを回避するため、または、識別を容易にするためである。

ORIONは、整合性および参照制約の維持を試みながら、実行時におけるスキーマの修正に最大の柔軟性を与えるように思われる。これは、ORIONがLISPベースで、その実行時注釈環境を最大限に利用するからである。これに対して、ONTOSは、C++プログラミング言語の静的性格のために、スキーマ修正に関しては極めてリジッドである。ORIONのより高度な特徴のいくつかについて簡単に検討を行なう。

２．７．４．５．４　ORION／ITASCA

ORIONは、スキーマ修正の結果としての曖昧さを解決するために、クラス格子の不変量と呼ばれるルールのセットを規定する。

・クラス格子不変量
クラス格子はルート付きであり（ルート・クラスを有しており）、連結有向非循環グラフである。つまり、（ａ）孤立（非連結）ノードがなく、（ｂ）持越し構造に循環がなくてもよい。

・明瞭な名前不変量
あるクラスの全ての変数および方法は、明瞭な名前を有していなければならない。

・定義域互換性不変量
あるクラスにおける持越し変数の定義域（またはタイプ）は、持ち越されてきたベース・クラスのものと同じでなければならない。

ORIONは、上記のスキーマ変更の全てを支援し、それらの変更の結果として生じることがある不

整合性を解消するためのルールおよび戦略を供給する。ORIONは、変更された場合のスキーマのバージョンを生成するために、スキーマ・バージョニングも支援する。

２．７．４．５．５　ObServer／ENCORE

ENCOREは、ORIONの場合ほど広範ではないが、多種多様なスキーマ展開機能を供給する。それに含まれるのは、タイプの修正、階層におけるタイプの追加および削除などである。スキーマの変更に伴って、スキーマのバージョニングが行なわれる。ENCOREは、例えば、削除されたクラスのスロットに対するアクセスを試みる際には、これらの変更の結果としての不整合性に対処するために、例外取扱メカニズムも供給する。

２．７．４．５．６　GEMSTONE

GEMSTONEは、実行時における新しいクラスの定義、クラスおよび事例変数の追加、削除、その他のような、スキーマ展開機能の限定セットを許容する。例えば、持越し階層、または、クラスの名前およびタイプの変更のような、大部分のその他のタイプの変更は、再コンパイルを必要とする。修正されたタイプの古い事例に対するデータ移行は不可能である。つまり、以前の構造および挙動が保持されることになる。

２．７．４．５．７　ONTOS

ONTOSは、主としてC++言語の静的性格のために、そのスキーマ展開機能が極めて限定されている。新しいクラス（既存のクラスのサブクラス）は、データベース・タイプオブジェクトを使用して、実行時にプログラム定義することができ（ONTOSユーザー・マニュアルを参照）、そのクラスのためのC++コードは自動的に生成させることができる。もし事例がない場合には、クラスの構造は実行時に修正しても構わない（データ移行は支援されない）。スキーマ展開および修正に関しては、ONTOSは、グラフィカル・スキーマ設計技術者に付属している。

２．７．４．５．８　ObjectStore

ObjectStoreは、そのグラフィカル・ユーザー・インターフェイスを通じて、限定的なスキーマ展開機能を供給する。スキーマの中で新しいクラスを定義することが可能になる。クラスを修正することができるのは、既存の事例がない場合に限られる。

２．７．４．５．９　VERSANT

VERSANTは、いくつかの限定的なスキーマ展開機能を供給する。データベース内のクラス定義を指定スキーマ・ファイルと比較するスキーマ変更ユーティリティ・プログラムが提供される。新しいクラスを挿入することができる。属性の追加および削除が許容されるのは、リーフ・クラス内に限られる。リーフ・クラス以外のクラスに対する変更は、そのクラスおよび全てのそのサブクラスおよび事例を落とす結果となる。

２．７．４．６　オブジェクト記憶管理

OODBMSの記憶サブシステムは、ディスクへのオブジェクトの二次記憶の管理、および、クライアント機内のインコア・メモリの管理に責任を負う。これには、ディスクにおけるページの割当ておよび割当て解除、ディスクおよびクライアント機メモリに対するページの出し入れ、コレクションに関するオブジェクトクラスタリングおよびインデクシング、メモリ内でのオブジェクトバッファリング、その他が含まれる。ここで考察した全てのシステムは、データ記憶および分散に関しては、クライアント・サーバー・アーキテクチャを採用しており、素晴らしいオブジェクト記憶管理能力を提供する。

（１）ORION／ITASCA

ORIONが生成する全てのオブジェクトは、組＜クラス・ｉｄ、事例・ｉｄ＞から構成された単一識別子（UID）によって識別される。ORIONは、デュアル・バッファ管理スキーマを採用している。オブジェクトは、メモリ（オブジェクトバッファ・プール）とディスク（ページ・バッファ・プール）内では異なる記憶フォーマットを有しており、オブジェクトをディスクから、および、ディスクに対してフラッシュしなければならない場合には、変換に若干のオーバヘッドがある。

オブジェクトバッファ管理システム（クライアント機メモリに常駐）は、オブジェクトUIDを含む常駐オブジェクトテーブル（ROT）および常駐オブジェクト記述子（ROD）を維持する。RODは、各オブジェクトに関して、そのUID、ディスク上の物理的記憶位置、ロックおよびメッセージ・キャーシュなどの関連情報を含んでいる。

ORIONは、高速検索のために、同じディスク・セグメント上で、あるクラスおよびコレクションオブジェクトの全ての事例のクラスタリングを連続的に支援する。さらに、Ｂ＋樹木インデックスに類似した手法を利用して、クラス・階層インデックスと呼ばれる二次インデックスも維持する。このスキーマにおいては、インデックスは、ユーザー指定クラスに定着している階層の全てのクラスに共通な属性に関して、定義が行なわれる。このフレームワークは、クラスの削除、または、クラスの名前の変更、その他のような、特定のタイプのスキーマ修正の場合には、維持することが困難ではあるが、大きなデータベースに関してはうまく機能する。

ITASCAは、私用および共用（または大域）データベースの形で、データベース区画を支援する。共用データ区画は、全てのユーザーがアクセス可能であるが、一方、私用データにアクセス可能であるのは、許可されたユーザーまたは局所トランザクションに限られている。データは、データベースに出し入れ（チェックインおよびチェックアウト）することができる。複合オブジェクトは、１つの単位として、チェックインおよびチェックアウトされる。バージョン化可能なオブジェクトは、私用データベース内にチェックアウトされると、バージョン化され、その結果、親バージョンは、その他のクライアント用として、共用データベースに残る。共通トランザクションおよび共同設計におけるデータベース整合性の維持には、データベース区画化が有益である。

（２）ObServer／ENCORE

ENCOREの全てのオブジェクトは永続性があり、単一識別子（UID）によって識別される。各々のオブジェクトが、１つの外部UIDおよび１つ以上の内部UIDを有する。外部UIDは、クライアントに対して、オブジェクトに対する一定の参照を提供する。外部UIDは、オブジェクトをディスク上に物

理的に位置決めするためにシステムよって使用される１つ以上の内部UIDに対してマッピングされる。削除されたオブジェクトは、その外部UIDが、削除されるのではなく、ツームストーンUIDに対してマッピングされる。これによって、削除済みのオブジェクトに対する参照を探知し、問題を取扱うための適当な例外を設けることが可能になる。

ObServerは、様々な判定基準に従って、例えば、あるオブジェクトの全ての事例、コレクションの形のオブジェクト、複合オブジェクト、特定の値の属性を有するオブジェクト、その他のような、オブジェクトをディスク・セグメント内にクラスタリングすることを許容する。連続セグメント内へのクラスタリングが、ディスク・アクセスおよび検索を大きく改善する。同じオブジェクトを異なるカテゴリーにおいてクラスタリングすることも可能であるために、オブジェクトを異なるセグメントにおいて（多重内部UIDによって）複製することもできる。複製オブジェクトテーブル（DOT）は、外部UIDから内部UIDへのマッピングを維持する。オブジェクトの複製の更新は不経済であるが、サーバが全てのコピーの更新を原子的に保証する。オブジェクトはセグメントの単位でクライアント機メモリにロードされ、主メモリ・セグメント・テーブルのような、様々なハッシュテーブルが、オブジェクトをトラッキングするために使用される。

（３）GEMSTONE

GEMSTONE核クラスオブジェクトのサブクラスの全ての事象は永続性がある。つまり、ディスクに記憶される。オブジェクトは、Sun３機の長整数として実現された単一オブジェクト指向ポインター（OOP）によって識別される。ディスク上のオブジェクトの物理的記憶位置に対してOOPをマッピングするために、様々な一次（インコア）および二次オブジェクト・テーブル（OT）が使用される。

GEMSTONEは、ディスクと機械メモリ間で（ユーザーに対して透過的に）オブジェクトをシャッフルし、大きな動的仮想データ・スペースを生成する。各々のクライアントトランザクションに対しては、ディスク上の直近の関係オブジェクトテーブル（OT）の蔭のコピーを含む作業空間を（クライアント機に）提供する。トランザクション中のオブジェクトに対する変更は、蔭のオブジェクトテーブル（SOT）に記録され、その他のセッションにはアクセス不能なページに配置される。トランザクションが関係する場合には、変更はサーバにおいて更新される。

GEMSTONEは、オブジェクト・識別子マッピングを含む記号辞書のセットを使用し、OPALコードのコンパイル中に記号を解読するために使用される。各々のユーザーは、ユーザーが私的にアクセス可能なデータ・スペースを定義する専用の私用記号辞書を有している。それに加えて、GEMSTONEは、ユーザーのグループがアクセスおよび共用することが許されるオブジェクトを含んでいる共用辞書を提供する。共用の範囲については、ユーザーが定める。全てのシステム核オブジェクトは、全てのユーザーにのみ読み取り・アクセス可能な、ユーザーグローバルズと呼ばれる大域辞書の中に常駐している。GEMSTONEは、例えば、コレクション、事例のセット、その他のような関連オブジェクトの連続的な物理的記憶を可能にするための様々なタイプのクラスタリング手法を提供する。

（４）ONTOS

ONTOSのオブジェクトは、永続性にも非永続性（ディスク表示なし）にもすることができる。その他のシステムの場合と同様に、オブジェクトはUIDによって識別される。ONTOSは、２種類のメ

モリ・ベース参照－透過性（TRefと呼ばれる）参照および直接参照－を提供する。直接参照は、機械メモリ内にオブジェクトをロードするために、明示活動化呼出しを必要とする。透過性参照は、核クラスTRefの事例に対するポインターである。TRefによって指し示されるオブジェクトは、参照と同時に自動的に活動化される。オブジェクト非活動化はオブジェクトを現在のトランザクション・バッファ・プールにフラッシュさせ、そのトランザクションが始まる場合には、書き込みが行なわれる。オブジェクトは、単独またはクラスタの形で（例えば、全集合オブジェクト）、活動化または非活動化させることができる。ONTOSは、ディスクへのデータ転送に関して、以下の３つのオブジェクトバッファリングスキーマを提供する。

- **バッファリングなし。この場合には、各々の「プット」呼出しの度毎に、直ちにサーバへの転送が行なわれる。**
- **省略時バッファリング。この場合には、オブジェクトはバッファされ、通常は「プット」呼出し１０回毎に、小グループでサーバに送信される。**
- **開始までバッファ。この場合には、トランザクションが開始されるまで、オブジェクトはバッファされる。**

その他のシステムの場合と同様に、ONTOSにおいては、ユーザーの仕様に従って、ディスク上でオブジェクトのクラスタリングを行なうことが可能である。小さなオブジェクトの大きなグループに関するデータベース・オペレーション（ネットワーク転送、ロッキング、記憶）を最適化するために、グループ管理システムを事例化することができる。ONTOSの欠点の１つに、データベース機密保護の、つまり、オブジェクト所有権または許可の完全な欠如がある。データベース・ファイルに対する適当なアクセス特権を有する人であれば、誰でもデータを修正することができるのである。

（５）ObjectStore

ObjectStoreにおいては、データ（その他のC／Cプリミティブ）は、以下の数種類の方法によって、その永続性を宣言することができる。

- **ObjectStoreの永続性新演算子を使用する。**
  ObjectStoreは、永続性記憶を取扱うための、C++オペレータ新のオーバーロード・バージョンを提供する。これは、以下の構文を使用して、オブジェクトの物理的（およびインコア）記憶を特定のセグメントに割当てる。新（データベースまたはセグメント、オブジェクト参照）タイプ永続性新演算子はその補数を有しており、永続性削除演算子はデータのインコアおよび物理的記憶の割当てを解消する。

- **ObjectStore演算子永続性を使用する。**
  これによって、変数またはオブジェクトを永続性にすることが可能になる。例えば、永続性<データベース＝タイプ・仕様変数またはオブジェクト名。

- **機能objectstore－割当て（）を呼び出す。**
  これによって、指定されたデータベース内に永続性オブジェクトが生成される。

上記のスキーマは、単にC++クラスのオブジェクトだけではなく、あらゆるタイプのデータをデー

タベース内で永続性にすることが可能になる。これによって、既存のCコードからのObjectStoreへのインターフェイスが容易になる。

ObjectStoreは、コア内のオブジェクトを物理的表示のためにディスクにマッピングするため、または、その逆のために、全ての仮想メモリ機能を実行する。アプリケーション内のオブジェクトに対するアクセスは、ユーザーに対する透過性がある。つまり、すでに仮想メモリ内にない場合には、ユーザーはデータを永続性記憶から明示検索する必要がない。その他の全てのシステムと同様に、ObjectStoreは、例えば、コレクション、複合コンフィギュレーション、その他のメンバーのような、関連データの連続記憶のためのクラスタリング機構も提供する。ユーザーは、データを共同配置することができるセグメントを指定することができる。ObjectStoreのアプリケーションは、多重データベースの生成、アクセスおよび操作を同時に行なうことができる。これによって、参照の維持（核クラス参照の事例を使用）、および、データベース相互間のデータ移行が可能になる。

ObjectStoreは、協力作業のための局所および大域共用作業空間の支援も行なう。あるユーザーの私用作業空間は、その他の全員にはアクセス不能であり、個人的作業用である。様々なユーザーが、その作業空間を同じ共用作業空間から持ち越すことによって、共通の作業空間を（協力作業のために）共用することができる。そうすれば、相互共用のために、データをこの親データ空間からチェックアウトすること、および、この親データ空間にチェックインすることができる。作業空間は任意の深さの親・子階層を形成することができ、その結果、多くの程度の共用および私用を存在させることができるようになる。作業空間は、後の節で検討するバージョン・コンフィギュレーションと密接に結び付いている。

（6）VERSANT

VERSANTは、単一横断多重データベースである、論理オブジェクト識別子（LOID）の名前で知られている、永久識別子によって定義される。オブジェクトのクラスは、事例に含まれている別の識別子によって定義される。記憶オブジェクト識別子（SOID）の名前で知られているもう1つの識別子が、オブジェクトの物理的記憶位置を提供する。VERSANTにおいて使用されるオブジェクト管理機構は、ORIONの場合に類似している。LOID、メモリ・ポインター、および、ロック状況を含んでいるオブジェクトキャーシュ・テーブル（OCT）が、各アプリケーションについて、VERSANT管理システムによって維持される。OCTの各項目も、メモリ内のオブジェクトの物理的記憶位置を指し示すキャーシュトオブジェクト記述子を指し示す。キャーシュ・テーブルの中にない、あるいは、スワップ・アウトされたオブジェクトの検索の結果、VERSANT管理システムは、VERSANTサーバに対して、オブジェクト検索要求を送ることになる。VERSANTサーバは、オペレーティング・システムとのインターフェイスによって、低レベル記憶管理を提供する。

VERSANTは、物理的記憶に関しては、異なるクラスの事例のクラスタリングを同時に行なうことを可能にする。省略時機構は、1つのクラスの全ての事例のクラスタリングを同時に行なう。ORIONの場合と同様に、VERSANTは、共用および私用データベースの概念を有している。アプリケーションはいつの時点でも多重データベースに結合することができる。

２．７．４．７　基本的トランザクション管理機能

この節では、各システムのトランザクション管理フレームワークの基本的機能のいくつかについて検討を行なう。ロック管理、バージョンおよび通知管理、同時性管理、その他のような、このフレームワークの様々な構成要素については、以降の各節で検討を行なう。

（１）ORION／ITASCA

ORIONは、正規および仮想の２つのタイプのトランザクションを支援する。正規トランザクションは、開始された場合には、データベースが更新される結果となる。仮想トランザクションは、常に打ち切りを行なうトランザクションであり、従って、トランザクションによって生じた変化がデータベースに記録されることはない。その結果、仮想トランザクションは、複雑な実験的変更、さらに、データベースの更新を伴わない「フォット・イフ」解析に役立てることができる。

ITASCAは、長いおよび短いの２つのタイプのトランザクションを支援する。短いトランザクションは原子的で、独立しており、実行の際には直列化される。長いトランザクションは私的データベースで動作し、異例に長い期間続くことがある。長期間トランザクションは、共用データベースから私用データベースにオブジェクトをチェックアウトすることができる。オブジェクトを共用データベースにチェックバックすることによって、そのオブジェクトに関する長期間トランザクションが終了する。

ITASCAは、ユーザー間での共用トランザクションを許容する。多数のクライアントが、協力作業のために、同じトランザクションに参加することができる。ITASCAは、セッションの概念を支援する。セッションは、共通の入力および出力ファイルを有するトランザクションのシーケンスから構成されており、ワークステーションにおける独立したタスクを構成している。同じセッションに参加している多数のプロセスは、同じデータ、ロック、などを共用している。

（２）ObServer／ENCORE

ObServerは、最も柔軟性のあるトランザクション管理システムを有している。グループ化およびネストされたトランザクションを支援し、この場合には、１つ以上のトランザクションがもう１つのトランザクションの中に組み込まれている。これによって、トランザクションの内部でタスクを細分化することが可能になり、クラッシュの場合には、ロールバックまたは回復の程度が小さくなる。ObServerは、複雑なCADトランザクションが広範な対話を行ない、非直列的に実行されることを可能にする柔軟なタイプのロックのセットを提供する。

（３）GEMSTONE

GEMSTONEのトランザクションは、実行時に注釈的に定義される。アクセス・コンフリクトの場合には、全く警告なしでトランザクションを打ち切る傾向がある。

（４）ONTOS

ONTOSのトランザクションは、コンパイル時にC++コードで静的に定義される。ONTOSによるトランザクションは、127レベルの深さまでネストすることができる。ONTOSは、アクセス・コンフリ

クト、および、トランザクション待ち、打ち切りまたはデッドロックの可能性を取扱うために、例外のセットを提供する。これらの例外によって、アプリケーションが問題を適切に取扱うこと、あるいは、別のトランザクションの終了を待っている間に何か別のことをすることが可能になる。ONTOSのトランザクションの実行時定義のための動的C++リンカーは、IESLで開発されたものである。

（5）ObjectStore

ObjectStoreは、読取り専用および更新の2つのタイプのトランザクションを支援する。読取り専用トランザクションは、そのトランザクションの範囲内の何らかのデータを更新することはなく、データベース内の要求されたデータのスナップショット・コピーを得る。更新トランザクションは、データを実際に更新することも、しないこともあるが、アクセスされたデータに書込みロックを獲得する。

ObjectStoreは、ネストされたトランザクションをいずれかの任意の深さまで支援することはない。ONTOSの場合と同様に、トランザクション・コンフリクトまたは打ち切りを取扱うために、例外を提供する。システム指令トランザクション打ち切り（例えば、デッドロックによる）の場合には、システムが、ユーザーが定義することのできる特定の回数まで、トランザクションを自動的に再試行する。ObjectStoreは、共用およびグループ化トランザクションを支援することはなく、長いトランザクションおよび短いトランザクションの概念は全くない。

（6）VERSANT

VERSANTは、適度な効果のあるトランザクション管理を提供する。VERSANTは、長いおよび短いの2つのタイプのトランザクションを支援する。長いトランザクションは永続性があり、任意の期間継続することができる。多数の短いトランザクションおよびプロセスをつなぐこともできる。短いトランザクションは、性格が原子的である。短いコミットまたはチェックポイントは、分散形データベースの二段階コミット・プロトコルに従う。各々の短いトランザクションは、長いトランザクションに付属させられている。様々なタイプのロックが、同時性制御のために提供される。これらについては、後に説明を行なう。長いトランザクションは、短いトランザクション間でロックを共用する方法を提供する。短いトランザクションによって得られるロックは、コミットの後に解除する必要はないが、次の短いトランザクションで使用することができる。短いトランザクションで得られる暗黙ロックには、もしシステムがクラッシュした場合に、ロック情報が失われるという点で、永続性はない。

短いトランザクション中に行なわれた変更は、「チェックポイント」を与えることによって、永久的なものにすることもできる。これによって、全てのロックおよびオブジェクトがキャーシュ・メモリの中に保持されることになる。短いトランザクション中には、セーブポイントを与えることもできる。これは、変更を永続的なものにすることはないが、トランザクションが復帰することのできる状態を記録する。

２．７．４．８　ロック管理

（１）ORION／ITASCA

ORIONは、以下の５つのタイプのロックを有する精巧なロック管理システムを提供する。

・共用（Ｓ）
読取りロックと等価であり、ロックが解除されるまでは、ロックされたオブジェクトの更新を防止する。

・意図共用（ＩＳ）
より自由な（概念排他ロックとの互換性がある）読取りロックであり、必要な場合には、Ｓ（読取り）ロックにグレードアップすることもできる。

・排他（Ｘ）
他のユーザーによる読取りまたは更新を防止する。

・意図排他（ＩＸ）
Ｘロックにグレードアップすることもできる。

・共用意図排他（ＳＩＸ）
オブジェクトはＳモードで暗黙ロックされ、更新を行なう場合には常にＸモードで暗黙ロックされる。

ORIONは、事例およびクラスレベルのロッキングの両方を提供する。事例は、ＳモードまたはＸモードのいずれかでロックすることができる。クラスは、全モードでロックすることができる。クラス・ロックに関するロックは、当該のモードにおいて全ての事例をロックする。

ORIONでは、スキーマ展開のためのクラス格子ロッキングも可能である。あるクラスに関して行なわれたロックは、持越し階層を下に伝搬させることもできるし、あるいは、ロックおよびトランザクションのタイプに応じて、様々なレベルへのスーパークラスのロッキングを要求することもできる。

ORIONにおける複合オブジェクトは、詳細なロッキングプロトコルを有しているーオブジェクトは、全体または選択的な部分として、８種類のロックモードでロックすることができる。（クラス・ノードまたはオブジェクトに関する）論理ロッキングに加えて、ORIONは、例えば、あるディスクのページの全てのオブジェクトに関して、物理ロックも可能にする。ORIONにおける全てのロックは、先入れ・先出し（FIFO）ベースで待機および承認される。ロックは動的にセットされ、トランザクション中は随時変更することができる。

（２）ObServer／ENCORE

ObServerは、以下の４つのタイプのロックを支援する。

・非制限的読取り（ＮＲ－Ｒ）
クライアントが、その他のクライアントのアクセス特権を制限しないで、オブジェクトの読取りを行なうことを可能にする。

・制限的読取り（Ｒ－Ｒ）
その他のユーザーがロックの継続中にオブジェクトの書込みを行なうのを防止する。

・非制限的書込み（ＮＲ－Ｗ）
クライアントのために書込みロックを獲得し、その他のクライアントのオブジェクトの読取りは、ＮＲ－Ｒロックによるものだけを可能とする。

・制限的書込み（ＮＲ－Ｗ）
その他のクライアントによるオブジェクトへのアクセスを不可能にする。

全てのロックは通信モードと関連付けられており、これによって、ロックが要求された場合、オブジェクトに関するロック状況が変化した場合、あるいは、オブジェクトが更新された場合には、いつでもクライアント間通信を行なうことが可能になる。ObServerにおける通信モードに関しては、後の節で検討する。

ObServerは、ディスク・セグメントの中のオブジェクトに関する物理レベルロッキングも許容する。そのUIDによってオブジェクトに明示的に要求されたロックは、ハード・ロックと呼ばれ、一方、セグメントの中のその他の連続的オブジェクトに関して獲得されたロックは、ソフト・ロックと呼ばれる。全てのハード・ロックは待ちであるが、ソフト・ロックは、もし要求時に直ちに承認ができない場合には、待たない。

（３）GEMSTONE

GEMSTONEは、読取り、書込み（その他のクライアントが読取りを行なうのは許容するが、オブジェクトを修正することはない）および排他（その他のクライアントはオブジェクトにアクセスことができない）の３つのロッキング・モードを有する。各々のオブジェクトのロック状況、および、ロックを行なうクライアントの詳細な記録が維持される。しかしながら、GEMSTONEは、ロック要求を待つことはなく、コンフリクト状態のトランザクションを単に打ち切るだけである。

（４）ONTOS

ONTOSは、読取り、書込み、書込み意図およびロックなしの４つのタイプのロックを有する。読取りロックは、読取りアクセスのみを示す。書込み意図ロックは、修正されやすいオブジェクトに対して適用される。書込みロックは、アプリケーション用には使用されず、ONTOSサーバによって内部的に使用される。ロックなしは、オブジェクトのロックなしでのオブジェクトの読取りを許容する。通知ロックは、ONTOSで支援されることになる。ロックの強さは、トランザクションによって決まる許容度に応じて、システムによって決定される。許容度レベルとしては、以下の２つがある。

・読取り・書込みコンフリクト
書込み意図がすでに承認済みのオブジェクトに関して読取りロックが承認されるのを防止する。

・読取り・書込みなしコンフリクト
書込み意図ロックされたオブジェクトに関する読取りロックを許容し、トランザクション・コミット時にお

けるアクセス・コンフリクトの解決を試みる。

ONTOSは、クライアント間通信を支援することはないが、ロック要求を待ち、ロックまたはアクセス・コンフリクトの場合には、選択された例外を設ける。しかしながら、アプリケーションに対してオブジェクトのロック状況を随時提供することはない。

（5）ObjectStore

ObjectStoreは、書込みと読取りの2つのタイプのロッキングを支援する。永続的データ項目の読取りが行なわれる場合には、そのセグメントには暗黙の書込みロックが行なわれる。ネストされたトランザクション中に獲得されたロックは、そのトランザクションがネストされている最高レベルの（または、ネストされていない）トランザクションが終了するまでは、解除されない。明示的アンロック・プロトコルはなく、2段階ロッキングが厳密に実行される。ロックは、例えば、個々のオブジェクト、コレクションまたはコンフィギュレーションに関して、様々なレベルの細分性においてセットすることができる。

（6）VERSANT

VERSANTによって提供されるロックのタイプは、読取り、書込み、スナップショットおよびモニター・ロックと多様である。スナップショット・ロックは、オブジェクトのロック状況とは無関係に、オブジェクトのコピーを生成する。モニターは、もしそのような事象が発生した場合には、電子メールによってユーザーが通知を受けることになる事象の指定を可能にする。これらの事象に含まれるのは、その他の人々によるロックの解除とその結果としてのオブジェクトの利用可能化、ロック獲得の意図、ソフト・ロックの破壊、読込み、更新、削除、および、バージョン化である。ロックは、ハードとソフトの2つの優先権を有する。ソフト・ロックは、ハード優先権を有するロックを獲得するコンフリクト要求によって破ることができる。ロックを獲得する際には、時間切れまで待つ、ロックが使用不能になった場合には直ちに放棄されるブロックなし、ならびに、オブジェクトに関する場所の予約および使用可能になった場合のユーザーへの通知のような、各種の待ちオプションも提供される。ロックは、クラッシュに耐え抜く永続性ロックにすることも、永続性のない短いロックとすることもできる。ロックは、オブジェクトレベルで設定および解除することができる。セレクトのようないくつかのデータベース機能によって暗黙セットされるが、明示的にセットおよび解除することもできる。

2．7．4．9　変更通知

設計技術者が膨大な量のデータを共用し、従って、その状況について情報の提供を受ける必要があるために、クライアント間通信は、CAD／CMAのような分散形エンジニアリング作業においては極めて重要である。

（1）ORION／ITASCA

ORIONは、クライアントに対する変更通知のための様々な機構を提供する。

・メッセージ・ベース：このスキーマは修正メッセージを全ての関連ユーザーに定期的に送る。
・フラッグ・ベース：これはデータ構造を連絡なしで更新し、その変更による影響を受けたオブジェクトにユーザーがアクセスする場合に限ってユーザーに通知を行なう。
・即時：影響を受けるユーザーには、変更の発生後直ちに通知を行なう。
・遅延：ユーザーが指定した若干遅れた時点で、ユーザーへの通知を行なう。

オブジェクトにおける変更を探知および記録するためには、タイム・スタンピング機構を使用する。ITASCAは、変更通知を行なうことを可能にする総称的通知可能クラスも提供する。変更通知には、受動と能動の２つのタイプがある。受動通知は、更新または削除のような事象のセットを、通知可能クラスの事例に対して割当てる。もしその事例に関して事象が発生した場合には、それの参照は参照不整合となる。（該当するユーザーによる）これらの変更の承認が、参照整合性を回復する。能動通知事象は、更新、削除、新バージョン作成、チェックインまたはチェックアウトのいずれかとすることができる。これらの事象のあるオブジェクトに関して発生した場合には、変更の内容および変更を行なったクライアントを記述した電子メールメッセージが、（該当するユーザー宛に）送られる。（通知可能クラスとして実現された）この挙動は、電子メールの発送の前に、データベースによるその他のオペレーションを行なうために、洗練することもできる。

（２）ObServer／ENCORE

変更または更新が行なわれる場合には、その変更によって影響を受けるユーザーに対して、電子メールメッセージが送られる。すでに検討した各々のロック・モードは、クライアント間通信を容易にするいくつかの通信モードの１つと組み合わせることができる。通信モードには、以下の５つがある。

・更新通知
ロック・ホルダーにオブジェクト更新を通知する。

・読込み通知
その他のクライアントがオブジェクトの読込みを要求した場合には、ロック・ホルダーに通知を行なう。

・書込み通知
その他のクライアントがオブジェクトの読込みを要求した場合には、ロック・ホルダーに通知を行なう。

・読込み・書込み通知
その他のクライアントがオブジェクトの読込みまたは書込みを要求した場合には、ロック・ホルダーに通知を行なう。

・通知なし
いずれの事象であっても、通知は行なわない。

（３）GEMSTONE

通知機構は提供されない。

（4）ONTOS

通知機構は提供されない。

（5）ObjectStore

通知機構は提供されない。

（6）VERSANT

指定された事象が発生した場合には、電子メールによる通知を受け取るために、オブジェクトに対してモニター・ロックを行なうことができる。

2．7．4．10　バージョン管理

実験は実験的かつ増分的プロセスであることが多く、その範囲が時間とともに変化するために、バージョン管理がエンジニアリング・データベースにとっては重要になる。設計オブジェクトの展開、および、様々なトランザクションによって設計に対して加えられた変更をフォローしておくことが必要である。これは、協力作業にとっても重要である。というのは、様々なクライアントが、お互いのトランザクションの終了を待つのではなく、同じオブジェクトの異なるバージョンで同時に作業を行なうことができるからである。ORIONおよびObServerは両方とも素晴らしいバージョン管理機能を提供する。

（1）ORION／ITASCA

ORIONには、以下の3つのタイプのバージョンがある。

- 過渡　：広範にわたって作成中の中間的設計であり、広範な修正を行なうことができる。
- 作業中：公表またはプロトタイプ前のより安定した状態の設計。
- 公表　：その他のユーザーと公的に共用されるオブジェクトの最終設計段階。

各々のバージョン化可能オブジェクトは、総称バージョン化可能クラスの事例、または、そのサブクラスの事例である。各々のバージョンオブジェクトは、バージョン・カウント、次バージョン数、バージョン導出階層、その他に関する情報をカプセル化する。オブジェクトのバージョンに加えて、ORIONは、スキーマ修正を保持するためのスキーマのバージョン、および、オブジェクトをそれが所属する該当のスキーマにマッピングするための精巧なアルゴリズムも提供する。

ITASCAは、共用データベースから私用データベースにチェックアウトされた場合には、バージョン化可能オブジェクトのバージョン化を必ず許容する。オブジェクトのバージョンを参照するためには、動的バインディングが使用される。省略時参照は、生成された直近バージョンが対象となる。私用省略時（私用データベースにおける直近バージョン）、共用省略時（共用データベースにおけるもの）および大域省略時（私用データベースと共用データベースの両方における直近バージョン）の3つの省略時バージョン参照がある。ユーザーは省略時参照を選択することができる。ITASCAは、ユーザーが、生成、削除、チェックイン、チェックアウト、その他のようなバージョン管理手順を修正およびカスタマイズすることも許容する。

（2）ObServer／ENCORE

ENCOREは、事例およびクラスの両方をバージョン化する機能を提供する。全てのバージョン化可能オブジェクトは、バージョン展開の記録を維持するバージョン化可能タイプの事例である。複合オブジェクトのバージョン化には特別の考慮が払われており、コンポーネントオブジェクトの変更は、包括的オブジェクトに伝搬され、包括的オブジェクトの新しいバージョンを生成する。

タイプのバージョン化によって、修正されたタイプの古い事例が、ユーザーによって消去されるまでは、そのタイプの新しい事例と共存することが可能になる。例えば、タイプ定義から消去されたオブジェクトの属性へのアクセスを試みる際に、タイプの修正の結果生じることがある不整合に対処するためには、例外ハンドラーが使用される。バージョン経歴を維持するために、精巧なタイムスタンピング機構が提供される。

（3）GEMSTONE

バージョン管理機能はない。

（4）ONTOS

バージョン管理はバージョン２．０では利用不能である。将来のバージョンでは支援が行なわれることが予想される。

（5）ObjectStore

ObjectStoreは、ORION／ITASCAの場合に類似した強力なバージョン化機構を支援する。これによって、直線および分岐バージョン樹を生成するために、私用および共用作業空間に関するオブジェクトのチェックアウトおよびチェックインが可能になる。

ObjectStoreは、コンフィギュレーションの概念も支援する。コンフィギュレーションは、バージョン化、ロッキング、クラスタリング、その他のための単位として取扱うべきオブジェクトをグループ化する働きを行なう。コンフィギュレーションは、ObjectStore核クラスコンフィギュレーションの事例によって表示される。それらは、典型的には、複合設計オブジェクトを作り上げているコンポーネントオブジェクト、または、例えば、クラス自動車のコンポーネントのような、ともに展開するオブジェクトから構成されている。コンフィギュレーションは、ネストする、つまり、親コンフィギュレーションに組み込むこともできる。ObjectStoreは、更なるバージョン化が望ましくない場合に、バージョンを組み合わせ、コンフィギュレーションをフリーズする機能も提供する。

（6）VERSANT

VERSANTにおけるバージョン管理は、VERSANTがスキーマ修正のバージョン管理を支援しないという点を除いては、ORION／ITASCAの場合に類似したものである。バージョン化機能には、開放、作業および過渡バージョン、バージョン導出グラフ、総称事例、動的および静的結合、ならびに、チェックインおよびチェックアウト中の多重データベース横断的なバージョンの統合および協調の概念が含まれている。

２．７．４．１１　同時性管理

データベースの整合性および完全性を維持するためには、同時性管理が必要である。伝統的DBMSシステムは、原子的トランザクションの直列化を強制する傾向があり、１つのトランザクション全体を単位としてのデータベースの更新を許容するーそのトランザクションの中の全てのオブジェクトがコミットされるか、全くコミットされないかのいずれかである。これは、短期の財務またはビジネス・タイプのトランザクションの場合には好都合であるが、トランザクションが一般的には複雑で、以上に長く、しかも、対話型かつインターリーブ型である同時的エンジニアリングまたはCADアプリケーションの場合には、制約が大きすぎることが多い。OODBMSは、ObServer／ENCOREによって具体化されているような、直列化によって命令されるものよりも柔軟な同時性管理環境を提供する必要がある。

（１）ORION／ITASCA

ORIONは、数人のクライアントによって開始される多重同時トランザクションを支援する。しかしながら、トランザクションを直列化する試みを行なう。つまり、あるトランザクションを別のトランザクションの影響から完全に隔離する。これによって、トランザクション間の活動データ共用および協力が平衡されることになる。最近では設計技術者がグループで作業を行ない、共通のデータ・プールを共用することを許容するトランザクション・グループ用のフレームワークが提案されているが、ORIONでそれが実現されているかどうかは明らかではない。

ITASCAは、ユーザー間の協力的共用トランザクションを支援し、さらには、同時性を高めるために、データベースの区分および（私的データベースにおける）トランザクションの局所コミットメントを許容する。

（２）ObServer／ENCORE

ObServer／ENCOREは、大域的データベース整合性を維持するための手段として、直列化を迂回する傾向がある。その代わりに、アプリケーション意味論の観点からデータベース整合性を定義付けるアプローチを強調する。例えば、エンジニアリング設計においては、オブジェクトのたった１つの大域バージョンによって作業を行なう必要がなく、様々なユーザーが同じオブジェクトを違った風に読み取り、同時にその修正を行なうことを許容することが可能であるかもしれないし、あるいは、その方が望ましいとさえ言えるかもしれない。これによって、同時性が改善され、適当な通信プロトコルの使用によって、トランザクション間での活動データ共用および対話が促進される。ObServer／ENCOREは、このフレームワークを支援するいくつかの新規な機能を提供する。

・トランザクション・グループ

対話およびデータ共用のために、様々な関連および協力トランザクションがグループに組織化される。各々のトランザクション・グループは、ユーザー定義特権およびプロトコルのセットによって支配される（それ自体がいくつかのサブトランザクションを有することもできる）いくつかのトランザクションから構成されている。コンポーネント・トランザクションは相互に活発に対話を行ない、さらには、その他のグループのその他のトランザクションの好ましくない影響から遮断することも可能である。様々なグループにおけるコンポーネント・トランザクションは、１つの単位として、個々の「親」を通じて、その他の外部トランザク

ションと通信を行なう。トランザクションは、別のトランザクションの内部にネストすることもできる。従って、トランザクション管理用のフレームワークは、階層的であり、全てのトランザクションが独立の単位を構成し、データ・アクセスの際には相互に競合する伝統的システムの場合よりも高く簡素な抽象化レベルにおける抽象化が可能になっている。

・オブジェクト登録

これによって、オブジェクトをデータベースにコミットさせる必要なしに、その他のクライアントがオブジェクトの最新の状態を見て、使用できるようにするために、オブジェクトに関する書込み（または排他）ロックを保持することが可能になる。これによって、設計技術者は、トランザクションへのコミットまたはオブジェクトに関するロックの解除を行なう必要なしに、その結果を瞬間的にその他のユーザーと共用することができるようになる。この機能は、クライアント間のデータ共用を促進する。

・通信プロトコル

通信ツールが、アプリケーション開発およびデータベース更新の状況を常にユーザーに伝える。これが、ユーザーの行動がデータの状態およびその他のトランザクションの結果に依存する対話型トランザクションに多いに役立つ。

（3）GEMSTONE

GEMSTONEは、全ての同時トランザクションの直列化の試みを行ない、トランザクション単位でのデータベース更新を許容する。アクセス時よりはむしろトランザクション・コミット時にアクセス・コンフリクトのチェックを行なう楽観的同時性管理スキーマが採用される。

（4）ONTOS

ONTOSも全てのトランザクションの直列化を行なう。トランザクションに関して定められた許容のレベル（読取り・書込み・コンフリクト・なし、または、読取り・書込み・コンフリクト）に応じて、楽観的および悲観的同時性管理スキーマの両方を提供する。悲観的状況では、トランザクションの開始時に、アクセス・コンフリクトのチェックが行なわれる。これによって、同時性は低下するが、結局終了した場合には、トランザクション・コミットが保証される。ONTOSは、コミット前のトランザクションのチェックポイントを許容する。チェックポイントは、ロックを解除せずに、トランザクション・バッファ・プール内の全てのオブジェクトを直ちにデータベースにコミットさせる。これによって、データ共用を促進することができる。

（5）ObjectStore

ObjectStoreは、同時性管理のために、全てのトランザクションの直列化の試みを行ない、2段階ロッキング・プロトコルを強制する。しかしながら、データのバージョン化が可能になり、このデータのバージョン化を、共用および局所作業空間と併用することによって、より同時性の高い環境を実現することができる。

（6）VERSANT

VERSANTは、十分柔軟性のある同時性管理を提供する。数タイプのロックが提供され、様々なロ

ックの利用法がある。従って、もし必要であれば、同時性を高めるために、直列化を迂回することもできる。この場合には、データベース整合性は、アプリケーションによって維持しなければならないことになる。グループおよび私用データベースとバージョン化の併用が可能であることによって、同時性を高めることのできるようになる。

２．７．４．１２　分散協調処理のための機能

従来型データベースは、トランザクションの直列性を強制し、そこでは、１つのトランザクションは、その他の同時トランザクションの影響から完全に隔離されている。しかしながら、（分散形複雑CADアプリケーションのような）協力設計は、結果を共同利用することができるように、各作業単位が対話を行なわなければならないという概念をベースにしたものである。これには、非常に柔軟性の大きな同時性管理スキーマが必要であり、さらには、データベースの整合性および「正確さ」の概念を、アプリケーションの要件によって定義することが必要である。これは難しい問題であり、検討対象とした各システムは、アプリケーションのニーズに合わせるのに適したフレームワークを実現するためのに、若干の機能をユーザーに提供する。

（１）ORION／ITASCA

ORIONは、協力活動に役立つ以下の機能を提供する。

- 優れたデータベース更新通知スキーマ。
- 優れたバージョン管理機能。
- ロック要求を承認されるまで待たせるか、あるいは、デッドロックの場合にはフラッシュすることを許容するロック待ち管理。さらに、ロックはトランザクション中に随時解除または修正することができる。
- 実験作業および再設計を許容する素晴らしい動的スキーマ展開機能。

ITASCAは、協力作業を支援するために、以下の追加機能を提供する。

- 多数のユーザーによるトランザクションの協力的共用。
- 同じセッションへの参加という形での異なるプロセスによるデータおよびロックの共用。
- 短期トランザクションと長期トランザクションの区別。
- 共用エリアと私用エリアへのデータベースの区分。
- より優れたオブジェクト変更通知戦略。
- インテリジェントでカスタマイジング可能なバージョン管理スキーマ。
- 優れたデータベース許可および機密保護措置。

（２）ObServer／ENCORE

ObServer／ENCOREは、以下の機能を有するという点で、コエンジニアリング設計に最適であると考えられる。

- 様々なトランザクションによる同じオブジェクトの同時的読取り・書込みを許容する、柔軟性の非常に大きなタイプのロックのセット。ロックはトランザクション中に随時獲得および解除が可能

・オブジェクト登録
・素晴らしいクライアント間通信プロトコルおよび更新通知スキーマ
・トランザクション・グループおよびネストされたトランザクションに対する支援
・優れたバージョン管理能力
・適正なスキーマ展開機能

（3）GEMSTONE

GEMSTONEは、協力作業に対して適正な支援を行なう。しかしながら、機能の大部分は、ユーザーが実現しなければならないことになる。その関連機能の中には、以下のものがある。

・オブジェクト共用のためにグループを作ることを全てのクライアントに許容し（設計グループの全てのメンバーが、特定のオブジェクトにアクセス可能）、各ユーザー間の共通データ・ロックアップのために共用記号辞書を提供する。
・トランザクションが実行時に動的に定義される。
・システムが、オブジェクトに関する全てのロックおよびロック・ホルダーの詳細な記録を維持する。

GEMSTONEは、ネストされたトランザクションまたはグループ化トランザクションを支援することもないし、何らかの更新通知スキーマを提供することもない。　バージョン管理および更新通知機構のようないくつかの機能が、MITのIESLで開発されたMAGPIEGBRRIDGEと呼ばれるシステムにおいて、GEMSTONEを通じて実現されている。

（4）ONTOS

現状では、ONTOSが、協力型アプリケーション開発には最も不適であるし、上記の高度な機能をほとんど何一つ提供しないのである。別の欠点として、トランザクションが静的に定義されることがあり、その結果、（設計の進行に応じた）実行時における変更を行なうことができない。さらに、C++の静的コンパイルおよび実行環境のために、再設計および実験作業が困難で、時間がかかるのである。公的および私的データベース区画の支援は行なわず、データ機密保護の完全な欠如が危険である。

IESLでは、オブジェクトバージョン管理、クライアント間通信モード、更新通知、および、トランザクションの実行時定義のための増分リンカーのような高度な機能が、既存のONTOSフレームワークを通じて実現されている。

（5）ObjectStore

ObjectStoreは、ONTOSおよびGEMSTONEよりも優れた協力作業環境を提供する。その関連機能の中には、以下のものがある。

・チェックインおよびチェックアウト機能を備えた優れたバージョンおよびコンフィギュレーション管理スキーマ。
・データ共用のための私用および共用作業空間に対する支援。
・ネストされたトランザクションに対する支援。

・ObjectStoreは、通信および更新通知機能の提供は行なわず、柔軟性のあるロッキング・スキーマを欠き、動的トランザクション定義環境を欠いている。

（6）VERSANT

VERSANTは、コエンジニアリング作業に最適の市販のOODBMSの1つである。その優れた特徴には、以下のものがある。

・柔軟性のあるタイプのロックおよび使用。
・通知機構が利用可能。現在は、通知は電子メールによって行なわれているが、将来は、プロセス間通知機構が予定されている。
・長期トランザクションおよびネストされた短期トランザクションに対する支援。
・グループおよび私用データベースに対する支援。
・グループおよび私用データベースのチェックインおよびチェックアウトと統合された優れたバージョン管理。

2．7．4．13　照会管理

現状では、大部分のOODBMSは、RDBMSにおけるSQLのような簡素で効率的な照会言語を欠いている。いくつかのシステムについては、（オブジェクトSQLと呼ばれる）SQLのオブジェクト指向拡張言語が開発されているが、その他は独自の照会フォーマットを使用している。近い将来OODBMSより普及するにつれて、何らかの形態の標準化オブジェクトSQLが使用されるようになることが考えられる。

（1）ORION／ITASCA

ORIONは、独自の専用照会言語を提供するが、その構文はSmalltalkに極めて類似している。照会ステートメントは、次のようなオブジェクト指向メッセージ送りフォーマットで作成される。

```
(Receiver  Selector  arg1  arg2  arg3. . . )
```

ここで、Receiverは、通常は、メッセージの送り先のオブジェクトまたはクラスである。Selectorは、メッセージの名前、または、メッセージを実現する方法の名前である。アーギュメントarg1、arg2、arg3、その他は、オブジェクト、または、オブジェクトに対する評価を行なうコードのブロックである。典型的なメッセージは、select、some、all、ref、および、（>、<、≦、≧、=、その他のような）その他の比較演算子を含んでいる。

ORIONは、特定の照会を行なうのに最適の経路を自動的に決定する照会オプティマイザを提供する。この照会オプティマイザは、クライアントバッファ（仮想メモリ）およびディスク内のオブジェクトを探索するために様々な戦略を採用する。これらの戦略には、以下のものがある。

・シングル・バッファ評価スキーマ。
この場合には、クライアントオブジェクトバッファ内の該当するオブジェクトがまずサーバページ・バッファ・プールにフラッシュされ、サーバのところで探索が行なわれる。

・デュアル・バッファ評価スキーマ。
この場合には、探索述語がオブジェクトに対して2回評価される。

1回目はクライアント・バッファ・プールにおいて、次には、サーバのところで、クライアント・オブジェクト・バッファ・プールにはすでに存在しないオブジェクトに対して。ITASCAは、照会の範囲をユーザーが定義することを許容する。照会の範囲には、私用、共用および大域の3つのタイプがある。私用照会は、ユーザーの現在の私用データベースに対して行なわれる。選択されたオブジェクトを私用データベースから共用データベースにチェックインすることもできる。共用照会は、共用データベースに対してのみ行なわれ、選択されたオブジェクトを共用データベースから私用データベースにチェックアウトするのに使用することもできる。大域照会は、ユーザーの現在の私用データベースおよび共用データベースに対して行なわれる。

（2）ObServer／ENCORE

ENCOREも独自の照会言語を提供する。照会作業の中には、select、image、union、contained-in、member、その他が含まれる。照会は、指定されたタイプの事例のセットに対して行なわれ、事例のセットまたはブール値のいずれかを戻す。様々な手法（例えば、等価の照会への照会変換、データのタイプおよび特性に関するインデクシング、その他）の利用によって、照会の最適化が行なわれる。オブジェクト間ジョイン（Ojoinと呼ばれる）、射影、および、その他のセット・タイプ演算（合併、差および共通部分）のような、高等照会代数用のオブジェクト指向フレームワークは、現在実現されつつある。

（3）GEMSTONE

GEMSTONEの照会言語は、Smalltalkの拡張であり、メッセージ送りを使用して定式化される。照会は、あるクラスの事例のセットに対して、または、オブジェクトのコレクション（集合体）に対して行なわれる。照会機能には、select、detect、in、reject、その他が含まれている。アクセス速度を高めるために、コレクションオブジェクトにはインデクシングを行なうことも可能であり、その際には、事例変数の値がキーとして使用される。インデクシング・スキーマには、同一性と相等性の2つがある。同一性インデクシングは、ある述語がある目標値と同一であること、つまり、共通の内部表示を有すること、あるいは、メモリー内の同じ位置にあることに依存したものである。相等性インデクシングは、ある述語の値をある目標値と突き合わせるだけである（例えば、2つの異なる「自動車」オブジェクトの「色」属性が、同じまたは等しいかもしれない）。GEMSTONEの照会管理は、簡素ではあるが、効率的である。

（4）ONTOS

ONTOSは、そのオブジェクトベースに対してプログラムSQLインターフェイスを提供する。さらに、select、from、where、and、orのような基本的SQLコマンド、ならびに、between、not between、is in、is not in、その他のような標準ブールおよび関連演算子のいくつかを許容する。ONTOSオブジェクトSQLの利点の1つは、1つまたは複数の照会のパラメータを与えるために、手順（またはメンバー機能）を照会ステートメントの中に呼び出すことができることにある。それに加えて、オブジェクトの直接

参照を、照会表現のあらゆる深度に連鎖させることができる。というのは、例えば、以下の形式、つまり、自動車、車体、ドア、色の形式で属性を使用することは正当だからである。オブジェクトSQLによる連想検索によって、オブジェクトSQLはリレーショナル・モデルの場合よりも強力で、表現力豊かになる。

ONTOSは、SQL表現をストリング・アーギュメントであると考え、その評価を行なうライブラリ・クラスQueryIteratorによって実行される。照会は、クラスの拡張部分（その事例のセット）および集合オブジェクトに対して行なわれる。全ての照会は、ONTOSが提供する低レベル・アクセスおよび反復子機能を利用して、C++によってユーザーが行なうこともできる。さらに、辞書およびアレイ集合オブジェクト内のデータ要素に対するキー方式アクセスも許容する。

（5）ObjectStore

ObjectStoreは、データベースからのデータの航法および連想検索用の独自の専用照会言語を提供する。照会フォーマットは、以下によって与えられる。

コレクション表現［：整数表現：］

コレクション表現は、（セットのような）コレクションの名前であり、整数表現は、照会の選択判定基準である。比較演算子の実例としては、＞、＜、≦、≧、＝、その他がある。ObjectStoreは、コレクションに関するインデックスの維持を許容し（インデックスは動的な変更または脱落が可能である）、効率的な検索のために照会最適化を行なう。

（6）VERSANT

照会は、VERSANTにおいては、pClassに定義されている選択法によって行なうことができる。pClassは、クラスに関する情報を記憶する「メタクラス」である。－＜クラス＞によって定義されるpClass事例は、定義された各クラス毎に生成される。この選択法は、連言的に満足される述語のリストとすることができる探索条件を採用する。これらの述語は、そのクラスの属性に関して定義されており、リンクを横断することができる。サブクラスの事例を繰返し探索するために、照会を指定することもできる。パフォーマンスを向上させるために、B樹およびハッシュテーブル・インデックスを、クラスの属性に関して定義することができる。

２．７．４．１４　データベース許可および機密保護

無許可アクセスおよび改竄からデータを保護するために、データベース機密保護措置が必要である。データ許可機構が、（特権クライアントのみによる）データへの選択的アクセスを管理し、オブジェクトの所有権および使用の追跡を行なう際に役立つことになる。

（1）ORION／ITASCA

ORIONデータベースは、アクセス・コードおよびパスワードによって、機密保護が行なわれ、しかも、データベースのエリア毎に、アクセス・プロトコルおよび機密保護措置が異なっている。

ITASCAは、ユーザーのオペレーションおよびデータへのアクセスを管理するための精巧な許可スキーマを提供する。許可範囲は、role（つまり、ユーザーまたはユーザーのグループ）、object（データベース・エンティティ）およびtype（読取り、書込み、生成、削除、その他）の3つの定義域から構成されている。許可は、プラス（可）であっても、マイナス（不可）であってもよい。オブジェクト許可の定義域は、以下のカテゴリーに分けられる。

- **クラス許可：与えられる許可に含まれるのは、読取り・クラス・定義、更新・クラス・定義、生成・サブクラス、生成・事例、読取り・全・事例、および、更新・全・事例である。クラス許可は階層的である。つまり、階層内の特定のクラスに対する特権が、スキーマ内のその全ての降クラスにも適用される。**
- **事例許可：許可されるオペレーションは、更新・事例および読取り・事例である。**
- **属性許可：属性に関する許可は、更新・属性および読取り・属性である。**
- **方法許可：方法実行が、与えられる許可の唯一の形態である。**

各々のユーザーは、いずれの時点でも最大1つの私用データベースだけしか所有することができず、それ関するあらゆるオペレーションを行なう権限を有する。私用データベースの所有者は、データ共用のために、その他のユーザーに対してアクセスを承認することができる。

（1）ObServer／ENCORE

データベースは、パスワードによって機密保護が行なわれる。

（2）GEMSTONE

GEMSTONEも、データベースへのアクセス用のロジン・パスワードを要求する。ユーザーは、ユーザーによる適切な特権の承認がない限りは他人がアクセスすることのできない私用データ空間を有することができる。個々のオブジェクトへのアクセスは、データの開放の前のユーザーのidをチェックするアプリケーション専用アクセス機能によって、手順的に保護することができる。

（3）ONTOS

現状では、ONTOSはデータベース機密保護は行なっていない。データベース・ファイルに対して読取りアクセスによって、誰でもデータを読み取ることができ、書込みアクセスによって、自由に更新を行なうことができる。

（4）ObjectStore

ObjectStoreデータベースは、パスワードによって保護されている。

（5）VERSANT

機密保護機構は基本的にはONTOSと類似しており、アクセスはデータベース・ファイル・アクセス権によって決定される。しかしながら、異なるグループ・ユーザーのアクセスは制限することができる（読取りまたは書込みのいずれか）。グループ・データベースに結合する場合には、データベース内のオブジェクトをロックする許可を指定することもできる（例えば、RLOCKは読取りロックの

配置を示している）。

２．７．４．１５　マルチメディア・アプリケーション

マルチメディアは、データベースのパワーおよび表現性を高め、ユーザーとシステム間の通信の帯域幅を大きく拡大する。マルチメディア・データ管理がエンジニアリングにとって重要であるのは、多くのエンジニアリング・データが、画像および音声のような様々な形態のマルチメディア表示に適しているからである。

（１）ORION／ITASCA

ORIONは、（テキストまたは音声のような）線形および空間（例えば、ビット・マップ画像）マルチメディア情報の記憶、補足およびプレゼンテーション用の完璧なマルチメディア情報管理機構（MIM）を提供する。全てのこのような情報は、実際の装置のオペレーションおよび挙動を識別および管理する（プレゼンテーション・装置クラス、補足・装置クラス、補足・オブジェクトおよびディスク・ストリーム・クラスのような）クラスおよびオブジェクトの形で表示される。磁気ディスクに大きなオブジェクトをコンパクトかつ効率的に記憶するために、いくつかの手法が利用される。例えば、同じオブジェクトのバージョンは、共通の記憶ブロックを共用し、複製されるよりは、どこか他の場所に記憶される特定の変更に対するポインターを有する。

（２）ObServer／ENCORE

現在は不明。

（３）GEMSTONE

現在は不明。

（４）ONTOS

現在は、ONTOSは、マルチメディア・アプリケーションの支援は行なわない。ただし、（CAD図面のような）ビット・マップ画像をオブジェクトとしてデータベース内に記憶することはできる。

（５）ObjectStore

現在は不明。

（６）VERSANT

現在は不明。

２．７．４．１６　ユーザー・インターフェイス

（１）ORION／ITASCA

ORIONは、プログラミング用のLISP類似コマンド解釈プログラムを提供する。グラフィカル・スキーマ・エディタを提供するが、このエディタ上のドキュメンテーションは利用不能である。ITASC

Aは、データベース・スキーマ走査検索および開発用のOSF（オープン・ソフトウェア・ファウンデーション）／モチーフ追従グラフィカル・ユーザー・インターフェイスを提供する。

（2）ObServer／ENCORE

ENCOREは、Interviewsを使用して開発されたグラフィカル・ユーザー・インターフェイスを提供する。いくつかのデータベース・トランザクション機能に加えて、スキーマ走査検索および変更を許容する。

（3）GEMSTONE

GEMSTONEは、Smalltalkプログラミング言語に対する（GEMSTONE－Smalltalkインターフェイス、GSIと呼ばれる）Smalltalk類似ユーザー・インターフェイスを提供する。GSIは、強力な走査検索、編集およびデバッグ機能を備えた完璧なオブジェクト指向環境である。GSIは、OPAL用の単純なコマンド・ライン解釈環境を提供する。

（4）ONTOS

ONTOSは、X・モチーフ・ツールキットを使用して開発された（DBDデザイナーと呼ばれる）グラフィカル・スキーマ・デザイナーを提供する。クラス階層の走査検索、オブジェクト編集、新しいクラスの動的生成、C++コード生成、ならびに、コミットおよびロールバックのようないくつかのデータベース・トランザクション機能を許容する。

（5）ObjectStore

ObjectStoreは、Xウインドー・ベースのモチーフ追従エンド・ユーザー・インターフェイスを提供する。これには、定義されたクラス用のC++コード、および、データベース内のエンティティの値を表示するデータベース走査検索機能を含んだものである。さらには、クラス階層（およびクラスと結合されたコード）を表示するデータベース・スキーマ・デザイナーを提供し、（新しいクラスおよびオブジェクトの生成、オブジェクト・属性値の修正、その他のような）データベース・スキーマ定義および修正を許容する。ObjectStoreは、C++コードのデバッグ用の名前アンマングリング機能を備えたGNUGDBデバッガーの仮編成済みバージョンも提供する。

（6）VERSANT

VERSANTは、X・モチーフ・ツールキットをベースにした走査検索機能を提供する。走査検索パラダイムは、Smalltalkで使用されるものに類似している。要素のセットまたは要素が、さらに走査検索を続行することができる感知可能フィールドを有数パネルに表示される。VERSANTは、クラスおよびその属性の検索、事例の検索および編集、ならびに、事例のセットの処理を許容する。走査検索機能によって、多重データベースへの接続が可能になる。

２．７．５　まとめ

現在利用可能なOODBMSの主要な機能について、さらには、エンジニアリング設計管理にとってのその相対的長所および欠点について包括的な説明を行なった。OODBMSの方がRDBMSよりも優れていることを強調したが、それは、複雑なデータのモデル化および抽象化能力、ならびに、柔軟性のあるCADトランザクションのハンドリングに示されている。以下は、検討対象とした各システムの特徴を簡単に要約する。

ORIONは、極めて優れた機能を有する超高度プログラミングおよびデータベース環境であり、しかも、LISP環境が実行時効率にある程度のオーバヘッドをもたらすとはいっても、LISPプログラマには比較的使い易くなっている。ITASCAは、現在イタスカ・システムズ、Inc.から入手可能なORIONの市販バージョンである。

ObServer／ENCOREは、共有定義域で利用可能な素晴らしい研究プロトタイプである。ORIONの高度な機能の多くを有しており、おそらく、協力作業の処理用の最も柔軟性の大きなトランザクション管理フレームワークであろう。大学の製品であるために、まだデバッグを完全に行なう必要があり、ドキュメンテーションが不完全である。しかしながら、カスタマイジング、デバッグおよびエンハンスメントには、原始コードが利用可能になっている。

GEMSTONEは、これまでに開発された中で最初期の市販のOODBMSの１つである。十分な耐久性があり、Smalltalkプログラマには使い易くなっている。ORIONのより高度な機能のいくつかが欠けており、実行時効率はあまり高くないが、エンジニアリング設計用としては十分に妥当なツールである。

ONTOSは、比較的新しい市販品であり、プログラミング言語としてのC++のパワーおよび移植性に大きく依存している。C++の効率性にもかかわらず、ONTOSには、その言語の静的性格および硬直性に伴う欠点がある。大部分の高度データベース管理機能はまだ開発中である。

ObjectStoreも、オブジェクト指向プログラミング言語としてのC++の移植性および人気を活かした新しい市販のOODBMSである。コエンジニアリング用の環境としては、ONTOSよりも優れたものを提供し、いくつかの高度トランザクション管理機能を有する。

VERSANTは、C++フロント・エンド、および、NIHに類似したライブラリー・クラスのセットを支援する新しい市販のOODBMSである。多重分散形データベースを備えた強力な環境を提供する。その高度トランザクション管理は、その他の市販システムの場合よりも柔軟性が大きく、従って、VERSANTの方が、エンジニアリング・システムには適している。

3　ソフトウェアの開発と管理の技術

我々が日頃使用しているソフトウェアには様々な種類がある。たいていの人が一度は使用したことがあるワープロ・ソフトウェアや表計算ソフトウェア等のパソコン・ソフトウェアをはじめとして、大型計算機上で研究者が使用する科学的数値計算コード、また、銀行のＡＴＭ、飛行機の操縦制御、コンビニエンス・ストアＰＯＳシステムのように、日常生活には直接的・間接的を問わず様々な場所で使用されている。

これら多種多様のソフトウェアは、その使用の目的により開発の手法も異なる。飛行機の自動操縦制御プログラムは多くの人命を直接左右する可能性がある理由により、本質的にワープロ・ソフトウェアの開発とは全く異なる開発手法が用いられている。

地層処分ヴァーチャル・エンジニアリング・システムの開発では、その機能・目的において様々な種類のソフトウェアを扱うことになると予想される。例を挙げれば、人工バリアや地下水の流れ等の性能評価を行う解析コード、多種多様な情報（データ）を管理するデータベース、また、解析結果を表示させるための視覚化ソフトウェア、これらソフトウェアの総合的とりまとめを行うマネージメント・システム・ソフトウェア等、様々なソフトウェアが考えられる。地層処分ヴァーチャル・エンジニアリング・システムは、飛行機の自動操縦制御プログラムのように直接人命に係わる機能が要求されるシステムとは考えられないが、人間生活の安全性に係わる事象を評価する機能を提供することから、システムの各部に於いて、高レベルなソフトウェアの品質管理対応が理想であると考えられる。

本章では、まず一般的な製品ソフトウェアの開発と管理技術について解説を行い、その後、地層処分ヴァーチャル・エンジニアリング・システム開発に係わる同様の技術検討を論じた。

3．1　ソフトウェアの一般的な管理活動

すべての努力を台無しにしてしまう可能性が一番高いのは管理業務である。その第１の理由は、ソフトウェア管理における決定が不確定な要素に基づいて行われるということである。不確定さの要因は、ソフトウェア要件の不完全性、作業範囲の変更、最初の想定の誤り、生産性の偏りなどである。この他、信頼性と効率性、保全性と効率性などの取捨選択も不確定性の要因である。これらの取捨選択は、システムの効率性、開発の期間とスケジュール、ライフサイクルの維持管理に大きな影響を与える。「完全な」ソフトウェア製品を作り出すには資源と時間が無限に必要だが、それは無理な話である。したがって、ソフトウェアの管理担当者にとってこれらの取捨選択は重大な問題である。管理担当者は計画作成、監視、管理（コントロール）という一般的な３つの手段を用いて確定要因を減らしていく必要がある。

（1）計画作成

計画作成とは、ソフトウェア製品の開発目標を達成するために必要な資源を見積り、それら資源の編成と割振りを決定して開発作業が実施できるようにすることである。見積りでは、開発するソフトウェアのサイズと複雑さ、開発スタッフの技量と生産性のレベル、開発資金の割振り、開発期間の割振り、潜在的な問題点、開発手法をはじめとする支援用資源の能力と正確さなどを考慮しなければならない。計画は、プログラム全体の統合的な開発と、開発に関わるすべてのグループの協力体制を前提として作成しなければならい。計画作成には、極めて重要な課題が２つある。１つは製品の実体

（すなわちソフトウェア要件）を理解すること、もう１つは製品開発が成功するための必要条件を知ることである。見積り作業が終わり、その結果がソフトウェア開発計画書にまとめられると、開発作業の実施に必要な資源の編成と割振りが行われる。

（２）監視

監視とは、作業の進展状況、製品の開発時点での品質、開発プロセスの質、基準への合致の度合いを識別する作業である。管理担当者は、プロジェクトの最終目標を達成するために必要な労力の投入状況や製品の作成スケジュールに常に気を配らなければならない。また、作業がスケジュールどおりに進行していなければ、それを是正する処置を講じる必要がある。それには、管理面と技術面の両方について方針変更の影響を評価・分析しなくてはならない。

そして、基準を満たさない場合はやり直しが求められる。進行状況は定期的にチェックされ、その時点までの成果が計画と対照される。技術面での進行状況と使われた資源は、開発プランの予算やスケジュールと対照され、合わないところがあれば訂正が求められる。製品保証技法は、開発プロセスの質と基準に対する適合度の評価にも用いられる。検査、監査、ウォークスルー等の技法は、それぞれの段階における製品の完全性、整合性、最新性を確かめるためのものである。基準に沿って開発を進めることは、エラーの発生防止、エラーの早期発見、開発者の生産性の向上という意味で不可欠である。

（３）管理（コントロール）

管理（コントロール）とは、開発プロセスのコントロールと各段階のソフトウェア製品自体の管理とを目的としたものである。プロセス・コントロールのねらいは、利用可能なあらゆる資源を使用して、（開発プランで設定された）目標や目的を達成することにある。そのためには、各開発者が自己の職務をタイムリーに遂行することができなければならない。それには、組織構造を通じて各グループに権限を付与し、同時に責任も割り当てる必要がある。これによって、組織構成要素間の摩擦が解消され、コミュニケーション用のチャンネル（組織構成要素間の）が確立される。

製品の管理は、ソフトウェア開発プロセスの各段階で作成される製品（文書）を対象とするものである。これには構成管理という技法が用いられる。構成管理とは、マスター・コピー（ベースライン）の作成を通じて、製品を無秩序な変更から守ることを意味する。これによって、ソフトウェア要件の定義、仕様の文書化、設計、プログラム作成、プログラムに対する変更というソフトウェアの開発工程の整合性が保証される。

（４）防衛的管理

管理担当者が計画作成、監視、ソフト管理（コントロール）をどのように行うかは、ライフサイクルの各段階ごとに異なる。これは、ソフトウェア製品の開発が進むにつれて組織内のより下位の構造が管理業務を担当することになるので、管理担当者の視角が変わるためである。いい換えれば、管理は次第に専門的で権限の少ない人たちが担当するようになるということである。

ソフトウェアの管理担当者は、開発プロセスは完璧なものではなく、変更は避け難いものであることを知っておかなければならない。つまり、開発プロセスは不安定なものだということである。管理

プロセスにはすべてを台無しにする可能性のある不確定要素がいくつも潜んでいるため、そこでの決定は製品を良くするよりもダメにする可能性が強い。

そこで、開発努力を台無しにしないためにどうすればよいか以下の事項を自らに問いかけることである。

- 何かうまくいかない可能性があるか？
- うまくいかなかったときに、どうやってそれを知ったらよいのか？
- うまくいかなかったときに何をしたらよいか？
- それを未然に防ぐにはどうしたらよいか？

要するに、ソフトウェア要件や仕様や設計などが正しくない、あるいは完全でない場合にそれを見分けるのがソフトウェア管理担当者の主要な役割である。したがって、ソフトウェアの管理担当者は、不十分な情報を基に決定を下さなければならない環境の中で、それを識別するためのスケジュールを立て、変更が必要な事項の早期発見を促進し、将来の変更を可能にする製品設計の柔軟性を確保するよう努力しなければならない。

（5）資源管理

資源管理とは、資金、時間、要員、機器というソフトウェア開発に用いる４つの主要な資源を管理することを意味する。要員や機器は資金とひとまとめにして「費用」として扱われるのが普通である。資源管理は、ソフトウェア要件分析段階が実際に始まる前に、すなわちシステムの開発の最初の時点から始まる。まず、最初にソフトウェアの開発に投入する資金の総額が決定され、ソフトウェア・プログラムの管理担当者に初期的な予算が示される。プロジェクトが進展して詳細な情報が得られるようになると、資金総額と当初の予算が見直される（通常は上方修正される）。資源を正確に見積もることと現実的なスケジュールを決めることは、ソフトウェア開発計画の重要なポイントである。これをきちんと行っておかないと、実際の開発作業においていろいろなことがうまくいかなくなる恐れがある。開発がスムーズに進行しない原因の代表的なものをいくつかあげている。

- 同時に達成できないソフトウェア要件があること
- 仕様書に欠陥があること
- 検査やテストが多すぎること
- 顧客の考えが不明確であること
- 新しいテクノロジーや設計手法が使われること
- ソフトウェア要件の解釈がひと通りではないこと

上記の事柄はいずれも、最近の高費用のソフトウェア・プロジェクトのほとんどが抱える共通の問題である。

（6）資源の見積り

費用やスケジュールを見積もることは非常に難しく、数多くの手法が提案されてきた。これまで提

案されてきた費用／スケジュール見積り手法の問題点は、それらが過去の経験に基づいていることにある。過去のデータが不十分である場合や、データが小規模のソフトウェア製品開発について集められたものである場合は、それを基にした大規模プロジェクトのスケジュール見積りが妥当であるとはいい切れない。費用見積りは、ミクロ見積りとマクロ見積りの２つに分けることができる。ミクロ見積りは、ソフトウェア製品の規模、開発開始日、各段階の所要期間を規定することから始まる。これらは、要員の経験や製品の複雑さに応じてあとで調整される。

（７）見積りの精度

どの手法を用いるにしても、費用見積りモデルで得た結果を利用する場合には十分な注意が必要である。実際には、外部要因がスケジュールを左右するケースがよくあるからである。

たとえば、OS／360オペレーティング・システムの開発には２年間かかったが、これはハードウェアの製作に必要な期間が２年間であったためである。開発要員は置き換えがきかないことも問題である。相対関係はあっても、２人の要員の能力が完全に１対１に対応することはないということが、すでに実証されている。

もう１つ問題がある。それは、人員を追加投入したからといって、その分だけプロジェクトの進行が早まるわけではないということである。プロジェクトの進行が遅れているからといって、やみくもに新しいスタッフを投入すると、余計に進行が遅れる場合もあるということである。プロジェクトの開発が見積りどおりに進めば、資源の管理は非常に楽である。だが、見積りどおりにいかない場合は、大変難しくなる。多くの研究者は、この問題を解決する最前の方法は管理プロセスを自己修正型のプロセスとして捉えると述べている。たとえば、開発の開始が遅れた場合には以下の対応をする。

- **製品のソフトウェア要件の範囲を狭める**
- **ある仕事から別の仕事に資源を割り振り直す**
- **開発期間と要員を（妥当な範囲内で）増やす**
- **そのプロジェクトを打ち切る、などのことが可能である**

ただし、補正が過度になり過ぎたり、開発プロセスの不安定さを増幅させる結果になったりしないように注意しながら、何をなすかを決定する必要がある。スケジュールはすべての開発要員の判断に大きな影響を与えるため、これを変更すると要員たちの問題の捉え方が変わってくる可能性がある。これを避けるためには、変更に関する情報を関係者全員に確実に伝達しなければならない。

（８）計画書の作成

資源管理の最後の要素はマスター・プラン、あるいは一般にソフトウェア開発プランと呼ばれるものの文書化である。この文書には、初期費用、スケジュール、要員、機器などの見積りと、開発にあたる組織の編成や利用する設計アプローチなどが記載される。この文書は、製品開発が進展するにつれて必要に応じて改訂されていく。

管理担当者は、ソフトウェアの開発に携わるスタッフ全員に特定のアプローチの使用を徹底させる手段として、また開発に関する疑問点を提起し解決するためのコミュニケーション手段として、この文書を利用することができる。

（9）製品保証

次に、「製品保証」という見出しの下で各種の監視活動について考察することにしよう。製品保証とは、製品の保全性を高める、すなわち製品の構造をユーザーや顧客の実際のニーズに合うようなものにするための作業である。そのためにソフトウェア製品が特たさねばならない条件は、以下のものである。

- ユーザーが求める機能が備わっていること
- 開発のライフサイクルを通しての追跡が可能なこと
- 性能の要件を満たしていること
- 費用が予想どおりのものになること
- 予定の納期に間に合うこと

製品の保全性を保証する責任は開発要員全員が持つのが原則だが、実際的には多くの担当グループに製品保証の個別的な職務（権限と責任）が割り当てられる。その職務とは、品質保証、構成管理、検証および妥当性、検査などである。これにはテストも合まれるが、これは基本的にライフサイクルの特定の段階を指すもので、ソフトウェアの開発プロセス全体に関わるものではないことに注意する必要がある。また、製品保証に関する用語は人によってさまざまな使われ方をするという点にも注意が必要である。たとえば、品質保証は、ある要員にとってはテストと評価のことであったりするからである。

3．1．1　品質保証

品質保証とは、品質の管理を通じて、開発された製品が仕様書どおりであるか、あるいはそれより優れたものであることを保証する事である。品質保証は、IEEEのソフトウェア品質基準では「製品またはプロジェクトが設定された技術的要件を構たしていることを十分に保証するために必要な計画的かつ体系的な活動」と定義されている。

いい換えれば、品質保証はあらかじめ設定した基準を製品が間違いなく満たすようにする活動である。製品の品質とは、機能と特性を合わせたものであり、それによってユーザーのニーズを満たす能力が生み出される。そして、品質の管理とは、製品の機能と特性を評価し、それを仕様書の内容と比較するために必要な作業である。

品質とは、後から付け足すというわけにはいかないものである。したがって、製品保証は、製品のライフサイクルの最初から始めねばならない。そして、その製品が使われるまで続けなくてはならない。それは、ライフサイクルのどの段階にも、エラーが新たに発生する余地が必ずあるからである。質の高いソフトウェア要件記述書を作成したからといって、質の高い仕様書が作成できるという保証はない。管理担当者が粗悪な仕様書の作成を見直し、それが設計段階に持ち込まれたならば、一層粗悪な設計が出来上がる恐れが強い。そして、プログラム作成段階ではさらに悪い結果がでるはずである。通常使われる品質保証方法は、見直し、検査、ウォークスルーである。見直しとは、不確かな情報や余計な情報が入っていないかどうか、また必要な情報が抜けていないかどうかを調べるためのもので、これによって、製品のソフトウェア要件分析、仕様書作成、設計、プログラム作成、およびテストの各々に関する情報内部の完全性と整合性が判別される。また、作成した情報が前段階の情報と

一致しているか否かを判別する場合も、見直しが行われる。

見直しは、開発要員、管理担当者、ユーザー、それに外部の専門家を中心とする広範囲の人々によって行われる．見直しを担当するグループば固定的なものではなく、必要に応じて編成され、ミーティングを通じて作業方針が決定される．各見直しグループは、それぞれ異なる目的やテーマを持っており、それに応じた見直しを実施する。見直しで発見されたことは、該当する開発担当グループに伝えられる。そして、何らかの処理が取られることになる。検査とは、構成要素レベルで仕様、設計、コーディング、テスト計画、テスト結果が正しいか否かを判別する作業である。検査は見直しよりも定式性が高〈厳密なものである。検査では、製品に生じる可能性のある障害のチェックリストを用いるといった具合に、焦点をはっきり絞った検査項目リストを用いて作業が行われる。検査グループは、全員が同レベルの技術を持つきわめて専門的なグループである。

ウォークスルーは検査や見直しとは異質のものである。この場合は、製品保証に責任を有する者による製品の「ウォークスルー（通り抜け）」検討会が開かれる。これは、できるだけ多くの情報をできるだけ多くの関係者に伝達することを目的としている。この「非公式な検討会」の場で、ソフトウェア製品の全体的な改良方針が示される。これらの技法は、管理担当者が製品の状況について偏りのない情報を得るためのものである。これらが実施されれば、各開発グループは各自の開発を安心して進めることができる。

また、客観的な立場で評価が行われることにも大きな意味がある。ソフトウェアの管理担当者は、これらの技法を通じて、問題が起こった場合に論理的でしかも妥当性のある決定を下すための情報を入手することができる。品質保証方法は、追跡可能性と整合性の向上に役立つという利点はあるが、その性質上自動化には向かない。

3．1．2　構成管理

構成管理（Configulation Management：CM）とは、ライフサイクル全体について体系的な変更の制御や保全性と追跡可能性の監視を行うために、持続的にシステムの構成を識別することを目的とした製品保証の1つの手段である。ソフトウェアの管理担当者は、これを通じて、自分たちが作成しようと考えているものが何であるか、自分たちが作成しているものが何であるか、どんなものが作成されたか、などをその時点で知ることができる。構成管理は下記の4つの作業要素から成っている。

（1）識別

識別とはソフトウェア製品のベースラインの定義と記録を意味する。ベースラインというのは、ソフトウェア製品を定義するために各時点で作成される基本情報のことであるこの情報が更新されて新たな基本情報が作成される（前のベースラインが形成される。要するに、ソフトウェアの識別とは、これらのベースラインに名前を付けてその親子関係を追跡する作業である。

（2）コントロール

ソフトウェア構成のコントロールとは、ソフトウェア製品のライフサイクル全体について、ベースラインの変更に関する提案の形式設定、評価、承認・不承認をコントロールするメカニズムである。変更提案のコントロールの核を成すのは以下の活動である。

・変更の提案とその変更の内容の定義を目的とする技術的変更提案書と呼ばれる文書の作成
・ソフトウェア構成コントロール委員会と呼ばれる変更提案の公式の評価と処理を行う機関の設置
・ベースラインに対する変更をコソトロールする手順の作成

変更の要求は、ソフトウェア要件を変更する場合や文書の欠陥を訂正する目的で出されることが多いが、どの要求も認められるというわけではない。要求が出されると、技術面からみた変更の実現可能性とスケジュールや予算に対する影響についての考慮がなされる。この際には、ベースライン、製品の保守容易性、柔軟性等に対する影響も検討される。

変更の承認・不承認の判断は、それの価値と費用を基になされる．変更が承認された場合は、変更が実施され、それが完全に検証され、そのあとベースラインの修正が行われる。

（3）監査

ソフトウェア構成の監査とは、ソフトウェア製品の現在の状況がソフトウェア要件記述文書や仕様書とどの程度合致しているかを判別する手段である。また、これを通じて正式なベースラインが確立される。どんな文書も、監査をパスするまでは、ベースラインとは認められない。

ベースラインの変更についても同様である。監査は、構成の検証と確証という２つの作業を通じて実施される。構成の検証とは、あるベースラインに指定した構成項目について意図したことが次のベースラインにきちんと反映されていることを確かめる作業である。一方、構成の確証とは、構成項目が顧客やユーザーの要求に合致することを確かめる作業である。この２つを実施すると、ソフトウェア製品の品質が向上し、またソフトウェア管理担当者に非常にわかりやすい形で問題が浮き彫りになる。

（4）状況報告

ソフトウェア構成の状況報告とは、ベースラインと実施または承認された変更を記録する作業である。変更は承認されてもすぐに実施されるとは限らないので、ベースラインの保守性を維持すること、それをきちんと追跡することが必要である。この情報は、将来新たな変更提案について考慮する場合にも参考にされる。ソフトウェア構成の状態報告では、他の３つの構成管理作業の記録も作成される。

構成管理作業、特にコントロール作業は、広い範囲にわたって自動化されている。この自動化は、各構成管理作業でそれぞれ自動化された手法のセットが用いられ、それらがまとまって１つの構成管理システムが構成されるという形をとる。

大規模なソフトウェア製品の開発では、構成管理システムの自動化は不可欠な条件である。構成管理の利点は多い。製品のエラーが減少し、保全性が保たれ、変更が必要な箇所だけに限定される。また、製品開発の記録が維持されることも大きな利点であるが最大の欠点は、費用がかかることである。構成管理にどれだけの費用をかければ、最終的にどれだけの費用が節減できるかを判別することは極めて難しいので、構成の管理はそれほど厳密に行われない場合が多い。たとえば、構成の監査では、次の金曜日までに出来たものをとにかく新しいベースラインにしようというような決定が下されることがときどきある。これは危険なことだが、このようにすれば追跡可能性をある程度のレベルに保つことができ、費用もそれほどかからない（あとで問題が生じなければ）。

3．1．3　検証と確証

品質保証と構成管理はハードウェアにも関係する作業だが、検証と確証（V&V）はソフトウェアの開発だけに関わる作業である。検証と確証は、ソフトウェア製品が機能および性能の要件を持たしていることを確認し、ソフトウェア要件が正しく記述されて正しく解釈されたことを保証するためのものである。これらの作業の狙いは、顧客の立場に立って、納入した製品が顧客の注文どおりのものであることを確かめることにある。

ソフトウェア・エンジニアリングの用語を定義したIEEEの用語集によれば、検証とは、「ソフトウェア開発サイクルのある段階の製品（文書）が、前の段階で設定された要件を満たしていることを確かめるプロセス」である。

確証とは、開発プロセスの最後にソフトウェアの評価を行うことを意味する。これは、ソフトウェアが当初の要件を満たしていることを確かめることを狙いとしたものである。いい換えれば、製品が目標を達成して正しい任務を遂行することを確認することを目的としている。

V&Vは「製品を正しく作成しているか」と「正しい製品を作成しているか」を判別する手段だと考えられることもある。検証と確証を実施する際の評価基準は、完全性、整合性、実現可能性、テスト可能性である。

完全性とは、製品の構成要素がすべて開発され、存在していることを表す。これは、必要な技能や文書が1つも欠落していないことを意味する。整合性とは、各段階での構成要素が前の段階からの出力を誤って解釈して生成されたものではないことを表す。これは、製品の開発プロセスが追跡可能であることを意味する．実現可能性とは、妥当な費用の範囲内で製品が作成できることを意味する。テスト可能性とは、仕様との合致を確かめる製品テストが妥当な費用で実施できることを意味する。

（1）V&Vがカバーするライフサイクルの範囲

他の製品保証手段と同様に、検証と確証のプロセスもソフトウェア製品のライフサイクル全体をカバーしなくてはならない。作業内容は凡そ以下の通である。

- **完全性、整合性、実現可能性、テスト可能性の要件を検討して一定の基準を作成する。**
  これらの要件は厳密に定義されていない場合が多いので、点線の矢印が示してある。
- **完全性、整合性、実現可能性、テスト可能性の基準がソフトウェア要件レベルで設定できない場合は、仕様書に基づいて設定する．**
- **仕様書がソフトウェア要件を反映したものであることを確かめる。**
- **設計が仕様書の内容を正しく反映したものであることを確かめる。**
- **設計の分解が製品設計の原則に反映するものでないことを確認する。**
- **およびソフトウェア・プログラムの各構成要素が、詳細設計の内容を反映したものであること。**

そしてその製品が、ソフトウェア要件や仕様書の内容を正しく反映したものであることを確かめる。

検証と確証は、ソフトウェア開発プランに必ず盛り込むべき要素であり、これまで見てきた製品保証技法と同じように、独立したチームによって遂行する必要がある。検証と確証を行えば、必ず誤りの発生件数が減るというわけではないが、誤りの検出に要する時間は間違いなく減少する。これによ

って、高い品質の製品が作り出される可能性が高まるが、構成管理と同様に、検証と確証にはかなりの費用がかかる。検証と確証のやり方は品質保証の場合と同じである。その違いは視点にある。

品質保証はあらかじめ定められている基準を製品が満たしていることを確かめることを主眼としているのに対し、検証と確証は製品がどの程度うまく機能するかを判別することを主眼としている。検証と確証を支援する手段はいろいろある。

３．１．４　再利用可能性の追求

再利用可能性とは、単に、あるソフトウェア要素が再び使用できることを表している。再利用し得るソフトウェア要素はたくさんある。たとえば、データ、設計、ソフトウェア要件、仕様、プログラム、共通システム、コード、モジュールなどは再利用が可能である。

再利用可能なデータは、再利用できる情報の基本要素である。再利用できる情報とは、標準データベース、光学、音声、図形、音楽などの情報である。これらをフルに活用するには、標準的なデータ交換形式を用いる必要がある。再利用可能な設計とは、コンパイラや整序（順序を整えるための）アルゴリズムなどのように、いろいろな形で利用できるが、結果が同じになるような設計のことである。

再利用可能なソフトウェア要件と仕様は設計より前段階の再利用可能な記述である。同様なアプリケーション・プログラムを多く開発する―たとえばレーザー・システムのアプリケーション・プログラムを開発するような―場合には、一般的なソフトウェア要件は皆同じになる。再利用可能なプログラムや共通システムとは、たとえば再利用可能なオペレーティング・システムやスプレッドシートといった完全な形の製品のことである。再利用可能なコードやモジュールとは、他のソフトウェア製品の作成に使用できるソフトウェアの構成部分のことである。これの例はサブルーチンや演算機能などである。これらの再利用可能な要素は、再利用可能なソフトウェアが何によって構成されているかを明確化するために、下記のように別のやり方で分類されることもある。

- **数多くのアプリケーション・プログラムに再利用できる要素（たとえば演算機能）**
- **後続のバージョンで使えるもの（たとえば前のバージョンの基本部分）**
- **プログラムが実行されるときにいつでも再利用できるもの（たとえばコンパイラ）**
- **１つのプログラムの中で再利用できるもの（たとえばサブルーチン）**

再利用可能性の追求がソフトウェア管理担当者の仕事である理由＝なぜこれらがソフトウェア管理担当者の仕事なのかという質問が出るかもしれない。その理由は、ソフトウェアの要素を再利用する機会がソフトウェアのライフサイクル全体にわたって数多くあること、そしてソフトウェア製品が可能なかぎり再利用できるとすれば、ソフトウェアの開発コストが減少し、生産性や信頼性が向上するなどの大きなメリットが生じるということである。管理担当者は、ソフトウェア製品の再利用の決定権を持っているだけでなく、技術要員によるソフトウェアの再利用を促進する責任も有している。このやり方が特に効率的であるというわけではないが、現状はそうなっている。

再利用可能性の問題は、検証と確証と同じように取り扱うべきである（実際、再利用可能性は検証と確証のもう１つの評価基準になり得るものである）。すなわち、各開発段階で、製品やその製品の一部が再利用できるか否かを問うてみるべきである。このようにすれば、再利用可能性も検証と確証における他の項目と同じようにはっきり見えてくるはずである。ソフトウェアの再利用を実践してい

る例がまったくないというわけではない。たとえば、NASAのあるアプリケーション領域では既存のソフトウェア製品の75～90%が再利用されている。この数字は別の調査・研究によっても裏づけられている。別の見方をすれば、毎年作成されている２５０億行に上るコードのうち75～85%が再利用可能であるにもかかわらず、毎年コーディングし直され、その費用が支払われているということになる。これは大変な金額である。それでは、コスト削減の責任は誰にあるかといえば、それはソフトウェアの管理担当者にあるのである。

（１）再利用の問題点

余分なコードにこのような膨大なお金が浪費されているというのに、ソフトウェアの再利用が大歓迎されないのはなぜだろうか。明言することはできないが、技術と人間の姿勢、の両方に問題があると推察される。ソフトウェアが真に再利用可能であるには、それの性能、特性、適用範囲、抽象化レベルに関する情報が利用できなければならない。この情報が利用できるケースもなくはないが、多くのケースではまったく利用できないか、あるいはすぐ利用できる状態になっていない。これは再利用を阻む大きな障害になっている。いい換えると、ソフトウェアが真に再利用可能であるということは、単に既存のコードが再利用できるというだけでなく、そのソフトウェアの構成要素が別の製品の新たな作成に利用できることも意味する。これが可能であれば、生産性が大幅に向上するはずである。

もう１つの問題は開発スタッフの姿勢にある。皆が自分のやり方に固執するのである。非常に難しい問題に遭遇したり、やさしいと思われるのにどこから手を付けていいのかわからなかったり、途中でひっかかって先に進めなくなったりした場合には、何か盗める（利用できる）ものはないかと探し始めるだろう。だが、そのような場合以外は、誰も他人のものに目を向けようとはしない。

したがって、管理担当者は、再利用できるものを見つけてそれを利用するのが当然という考え方を開発要員全員に定着させなくてはならない。

３．１．５　ソフトウェア・エンジニアリング環境要件

ソフトウェア・エンジニアリング環境が管理について果たすべき要件は、他の開発段階よりも多少多い。管理担当者の一般的役割は開発プロセスとソフトウェア製品自体を計画、監視、管理（コントロール）することである。したがって、環境はこれらの活動を支援するものでなくてはならない。管理担当者の中心的な仕事は製品保証である。

たとえば、その中の製品の検証を確実に行うには、ソフトウェア要件から仕様書へ、仕様書から設計へという具合に開発段階を順次追跡することが可能でなければならない、同様に、ライフサイクルの各段階での製品に対する変更についても、追跡し、管理（コントロール）することができなければならない。環境は、ライフサィグルの最初の時点からこれらの活動を支援する必要がある。ソフトウェアの管理に関するソフトウェア・エンジニアリング環境のもう１つの使命は、ソフトウェア管理担当者が理にかなった決定を下すために必要な情報を集められるようにすることである。このことは、ソフトウェア・エンジニアリング環境が、予算の作成や費用の追跡を支援するばかりでなく、予算、スケジュール、人員等の変更の影響を分析する手段を提供することも必要だということを意味する。

ソフトウェア・エンジニアリング環境にはもう１つ使命がある。それは、再利用が可能な部分の識別を助け、再利用可能な構成要素のライブラリを維持することによって、構成要素の再利用を促進す

ることである。

ソフトウェアの管理はソフトウェア製品のライフサイクル全体に関わるものである。管理の仕方が悪いと、ソフトウェア製品の開発がすぐ台無しになってしまう。したがって、管理担当者は、決定のための正しい情報を入手し、十分に注意しながら決定を下し、決定が間違っているとわかった場合はすぐにそれを撤回することが必要である。さらにまた、ソフトウェアの管理には、時間や費用の管理だけでなく、人間の管理も含まれる。組織を円滑に機能させ、人間とモノをうまく調和させるのがソフトウェア管理担当者の務めである。

3．2　今後の技術課題

今日の環境テクノロジに対する認識は、FORTRNコンパイラが初めて導入される直前（1950年代中頃）のコンパイラ・テクノロジに対する認識と似ている。現在では、環境を作り出すのに必要な要素はすべて存在している。だが、どのように要素を組み合わせたら環境がうまく機能するかをまだ誰も知らないのである。

（1）コンパイラと環境

コンパイラと環境を比較することは多くの点で非常に有意義である。今日のプログラム生成環境はざまざまな面で1950年代初期のアセンブラと同じ役割を担っている。当時、アセンブラは多くのプログラマにとって最先端技術であった。それらのアセンブラは利用可能なものだったが、普及はしなかった。アセンブラのなすべき仕事に対する基本的な認識は十分にあったが、その基礎理論がまだ曖昧模糊としていたからである。

このギャップを埋めたのが「プログラム自動生成機能」（つまりコンパイラ）であった。そして、このコンパイラの開発がすべてを良い方向に向かわせることになった（ちょうど今日のソフトウェア・エンジニアリング環境がそうであるように）。1950年代初期に模索されたのは、プログラマが英語に似た高水準言語を使って問題を「迅速に」記述してプログラムを作り出すということであった（ソフトウェア・エンジニアリング環境の利用はこの考えとほぼ同じである。「問題」を「ソフトウェア製品」に代えて読み直していただきたい）。この先は読者もよ〈ご存知であろう。言語が英語に似たものになり、プログラミングは容易になった。しかし、コードの自動生成に関しては、すべてが計画どおりにうまくいくわけではなかった。３０年後の今日においてもこの問題の解決が待たれている。とはいっても、３０年の間にコンパイラについて非常に堅実な論理が構築されており、今日ではアセンブラの作成は高校のコンピュータの授業でも行われるようになっている。ソフトウェア・エンジニアリング環境が理解されるまでには、コンパイラがそうであったように、あと３０年が必要なのだろうか。一方、プログラム自動生成機能はどうなるべきかという議論がかつてなされたように、ソフトウェア・エンジニアリング環境はどうあるべきかという議論が今日活発になされている。

（2）許容できる環境

理想的なソフトウェア・エンジニアリング環境では、多くの仮定がなされ、解決が必要な問題についてもはっきり定義されているので、管理担当者は誤った判断を下すことはなく、開発要員はソフトウェア要件の分析、仕様書の作成、設計といった段階において行うべきことがはっきりとわかる。ま

た、この環境には、開発における問題（予算の削減とか特別なモジュールの機能の仕方を知っている人員の離脱などの）を発生させる原因になるような要素はまったくない。だが、実作業においては、理想的な環境で言われているようには事は進まない。このため、現実に合わせた解（答え）を見い出す努力が必要だが、この努力は、環境が許容できるもの（理想的な環境ではなく）になるまで続けなければならない。しかし、この努力をどれだけ続ければ解が見つかるかは定かではない。

ソフトウェア・エンジニアリング環境に関する論議は許容できる解とはどんなものか、また、その解を見つけるにはどうすればよいのかを中心に展開されている。そして、使用するプロセス・モデル、利用できる手法、適切な自動化といったことが論議のテーマとなっている。前項述べたように、環境をめぐる論議の最大の問題は、環境に求めるものが人によってさまざまに異なることである。すべての人々の希望を満たすには、各人が求める可能性のあることをすべて環境に盛り込まなければならない。そのために環境が備えるべき特徴は以下のものである。

- **包括的であること**　：ライフサイクル全体が支援できること
- **適応性があること**　：様々なタイプのコンピュータ・アーキテクチャに適用できること
- **調整可能であること**：各種のライフサイクル・プロセスに適合するように調整できること
- **拡張可能であること**：手法またはツール（あるいはこの両方）を簡単に追跡できること
- **整合性があること**　：ユーザーにとって「継ぎ目がない」ように見えること
- **効率性が高いこと**　：性能が良いことまた、良い環境には以下の特徴も備わっていなければならない
- **凝集性があること**　：手法の統合が可能であるということ
- **追跡性があること**　：１つ１つの情報項目の作成と動作の道筋がたどれること
- **描象性があること**　：環境の活動の内部的な詳細の大部分がユーザーの目に触れないようになっていること
- **回復性が高いこと**　：ソフトウェアやハードウェアのエラーからの回復が容易なことリストアップすべき特徴は他にもある

３．２．１　ソフトウェア開発プロセスの問題

ソフトウェア・エンジニアリングおよびその環境に関する論議は、ソフトウェア開発のプロセス・モデルを中心に行われている。今日利用できる開発プロセス・モデルは、ウォータ・フォール型、プロトタイプ早期作成型、内部動作重視型、情報ベース型の４タイプである。この４つのうちソフトウェア製品開発に最も適しているタイプはどれかということが盛んに論議されている。しかし、残念なことに、この論議では、それらの開発プロセス・モデルが許容できるものであるか否か、またソフトウェア製品の開発プロセス全体をモデル化しているか否かという点が見逃されている。

（１）集団作業を前提とする開発プロセス

中規模以上のソフトウェア製品の開発には多くの人員が必要である。そして、実際に大きなグループによる集団作業が実施されている。まず、この視点に立ってソフトウェア開発を見る必要がある。開発にたずさわる人々は、グループ間で、また個人間で資源の獲得競争を展開するが、この競争は時として激しい争奪戦に発展することもある。現実のソフトウェア開発は混沌とした状況にあり、開発プロセスに人的エラーの原因が内包されている。

大規模なソフトウェア製品の開発の過程で実際に何が生じるかについて詳しく調べた者はいない。したがって、この点が明確化された開発プロセス・モデルはまだ存在しない。４つのタイプのライフ

サイクル・モデルは、いずれも開発グループが変則的な活動を一切行わないという前提に立ったものである。これらのソフトウェア開発モデルと実際の開発との食い違いを調べて今後は、集団作業に問題があることがわかるはずである。とすれば、現行の開発プロセス・モデルをよりよいものに改良するために今日なされている多大な努力は果たして有益なのだろうか、という疑問も湧いてくる。

プログラム作成のモデル化ではなく、ソフトウェア開発プロセス全体のモデル化が緊急の課題である。ソフトウェア・エンジニアリング環境用に新しいモデルを作りだすことができれば、その影響は甚大なものとなろう。

（2）許容できるプロセス・モデル

現行のモデルは、満足できるものではないとしても、許容範囲の最低レベルには達している。ウォータ・フォール型のライフサイクル・モデルを利用して何十億、何百億行ものコードが作成され、管理されてきた。そして、同タイプのモデルで今後やはり何十億何百億行のコードが作成され管理されることは間違いない。したがって、ウォータ・フォール型を改良して新たな開発プロセス・モデルを開発することがどうしても必要である。このプロセス・モデルのどこを改良すべきかということはわかっている。

プロトタイプ早期作成型および内部動作重視型のライフサイクル・モデルは、これらの問題を克服することを目的として生まれたアプローチである。ところで、モデル化に際して、ソフトウェア開発プロセスの中で実際に起こっている事をモデル化すべきか、それとも起こって欲しいと願う事をモデル化すべきか、という別の次元の問題もある。この問題に対する答えは重大な意味を持つ。なぜなら、この答えがモデルの構築方法と利用方法を決定づけるからである。

では、幾つかの考察を行ってみよう。個々の開発段階に関する議論の中には、仕様書作成段階は実際にはソフトウェア要件分析段階の一部であり、設計段階は実際には仕様書作成段階の一部であり、プログラム作成段階は実際は設計段階の一部であり、テスト実施段階は実際はプログラム作成段階の一部である、といった主張がある。これらが本当なら、テストの実施とソフトウェア要件の分析とはまったく同じ段階に属することになる。開発段階のいずれかを統合して環境を構築すれば、当然手法の統合の仕方に影響がでる。仕様書作成段階と設計段階とを一緒にするとしないのとでは、作成されるソフトウェア製品に差がでる。

また、必要な手法と自動化にも差が生じる。次に、開発段階のどれかとどれかを上述のように結合すると、ソフトウェア開発に対する視点が変わる。この結果、環境のユーザーに対する影響が及ぶのば必至である。開発段階同士の統合には良い場合と悪い場合とがある。これは、ユーザーがどの程度厳密に開発プロセスを管理（コントロール）したいかによって決まる。より厳密な管理を望む場合は、自動化された手法同士の統合の度合いを高める必要がある。統合の度合いが高い場合は、ユーザーが選択できる作業の進め方が少数に制限される。

一方、それほど厳密な管理を望まない場合は、統合の度合いを低くする必要がある。どちらの方針を選ぶかによって、環境の構築方法は大きく変わってくる。環境構築のためのアプローチ：高レベルの統合を望む場合は、環境をトップ・ダウン方式で構築しなくてはならない．（ただし、自動化の最終目標は各要素内でどのような取捨選択が必要であるかを決定することにある）。厳しい管理が必要ではない場合は、自動化された手法や技法を数多く用意して、それらがプロセス・モデルを支援する

形に（プロセスからそれらを導き出すのではなく）環境を作り上げることができる。

つまり、開発プロセスから手法が導き出され、それらの手法が高度に統合されることになる。このようにすれば、ユーザー（またはプロジェクトの管理担当者）の必要に応じて「自由に」変形することが可能である。さまざまなアプローチのどれを利用するかによって、開発の各段階で作成される情報の追跡性、整合性、完全性（もちろん品質にも）差が生じる（これらは、手法の統合の度合いが高いほど高くなる）。

最終的には、創造性に重きを置いてユーザーの自由性を高くするか、厳密に秩序立てられたアプローチを用いるかの択一を迫られる．ソフトウェア製品の規模が大きくなれば、当然のことながらより厳しい規律が必要となるが、「自分流」のやり方で作業が進められる柔軟性が欲しいと考える人も多くなってくる。そして、ソフトウェア製品にとって良いことを大切にするのか、それとも開発者にとって良いことを大切にするのか綱引きが絶え間なく続けられる。開発段階どうしを統合しようとする多くの試みは、上述の問題を部分的に解消することを狙ったものである。すでに述べたとおり、ソフトウェア要件分析段階とテスト実施段階が統合されて１つのものになり得るなら、たしかに問題点は減少するだろう。

プロセス・モデルの定義に使われる考え方は、コンパイラを作成するときのアプローチと同じである。コンパイラの場合は、コンパイルの各段階で何をなすべきかが総合的に定義される。さらに、コンパイルの各段階へのインタフェースが厳密に定義されるため、個々の段階にどんな手法（適切なものなら）を用いても構わない。また、最適化段階などのいくつかの段階を省くことも可能である。これは、コンパイルの速度を高めるためにコードの効率性などを犠牲にすることを意味する。

ところで、改良プロセスをこのような考え方でモデル化することは極めて有効である。改良段階ではソフトウェア製品の強化作業が行われるが、このようなモデルを利用すればライフサイクル全体をやり治す必要がないからである。要約すれば、ソフトウェア・エンジニアリング環境に関する技術的な問題は、結局、ソフトウェア開発に利用される、プロセス・モデル周りにあるということである。

開発プロセスをどれだけうまくモデル化できるかが、他のどの要因よりも製品の品質を強く左右されることになる。優秀な開発プロセス・モデルは、そのプロセスへの理解を深める手助けをし、ソフトウェア開発にたずさわる人々におのおのの任務と作業内容を知らせるとともに、開発の管理担当者に決定を行うための基準を提供する。ただし、適切なモデルを選び出すのには困難な作業である。

３．２．２　手法に関する問題

ソフトウェア・エンジニアリング手法に関する技術的問題は、手法の選択と統合に関するものが中心である。主要な問題は、個別的な手法を多数備えるべきか、それともそれらを統合すべきか、といったことである。

（１）手法の統合

統合されていない数多くの手法が利用できるようにすることは、統合に向けての初歩的なアプローチであり、また既存の多くの環境の基本的なアプローチである。手法が統合されていなければ、それぞれの手法は他の手法と無関係に働くため、環境構築の要件は極めて少なくなる。また、他の手法に対する影響に配慮することなく手法を追跡することもできる。このアプローチの欠点は、ある手法で

作られた情報を別の手法で利用することができないこと、情報の追跡が困難なこと、整合性や正確性をチェックする能力が低いことである。手法が統合できれば、追跡可能性、正確性、整合性のレベルが引き上げられ、また統合された手法の数とそれらがどの程度統合されているかによって追跡可能性、正確性、整合性の高さを厳密に測ることができる。

手法の統合に関する第１の問題は、ある手法と別の手法と統合し得るかということである。この問題は単純に見えるが、環境の構築を困難にする大きな原因である。なぜなら、すでにある手法はほとんどが他の手法との統合を前提とせずに個別に開発されたものだからである。各手法の観点や表記法、媒体、およびガイダンス情報がまちまちであることは大きな問題である。２つの手法が統合し得ることがわかったら、この２つの手法を相互連関させるために、拡張が必要か否かを判別しなければならない。

また、統合によって手法を構成している基本部分が影響を受けるか否か、手法の有用性が低下するか否かもチェックしなければならない。手法の統合の仕方は２種類ある。１つは情報だけを統合するやり方である。この場合、一方の手法で作成されたデータはもう一方の手法でも利用できるが、この２つの手法は基本的に別々に利用される。もう１方は、２つの手法を合体させるというものである。この場合、一方の手法はもう一方の手法がなければ利用できない。どちらの統合化アプローチの方が良い結果を生む確立が高いかという質問が出されるだろうが、それには利用される手法次第であるとしか答えようがない。統合される手法が増加すればするほど、統合は複雑なものになっていく。ここで注意が必要なのは、手法によって作り出された情報を、なんらかの方法で記憶しなければならないこと、そして、それがどの手法によって作成されたか、また別のどの手法による操作が許されるのかを明示する必要があることである。そこで、この仕事を行うデータベース機能の存在が重要になってくる。さらにまた、データの表現法を統一する必要が生じてくる。統一的な表現法がない場合は、個々の手法間のインタフェースを定義しておかなくてはならない。

N個目の新しい手法を導入すると、N−1だけ必要なインタフェースの数が増加することになる（Nは統合される手法の総数を表す）。統合される手法が増えれば環境の能力はある程度高まるが、同時に環境構築における複雑さも増大する。また、情報共用のレベルにおいて全手法が１つに統合されていない場合は、ある手法セットにより作成された情報を他の手法セットが再利用できるようにする作業が必要である。

この他、手法の統合が部分的にしか行われていない場合には、ユーザーの使用している手法が他のどの手法と併用できるのかを判別できるようにしておく必要がある。ユーザー自身がこれを判別しなくてはならないとしたら、まったく手法が統合されていない方がむしろ楽である。完全に定義された方法論を利用すれば、この種の問題が解消される。このような方法論では、すべての手法が完全に統合されており、手法を利用するステップが詳述されている。この方法論の利点は、すべてが定義されているため環境の構築が容易になることである。

データベース機能は、統合された手法とそうでない手法とを区別する必要がなく、ユーザーは、多くの異なる手法を使い分けるという煩雑さから開放され、1つの「手法」を認識するだけで済む。この方法論では、追跡性、整合性、完全性に特に重きが置かれている。この方法論にも弱点はある。何かを試そうと思っても、それを途中から開始することができないのである。方法論の中のある地点より前にある情報が必要な場合、必ずスタート地点まで引き返さなくてはならない。これは明らかに柔軟

性を低下させる原因となる。さらにまた、この方法論がアプリケーションにぴったり適合しない場合、その有用性がかなり低下することも問題である。もう１つ、既に使われている手法間の依存性が非常に強いため、全く混乱をきたさずに方法論に新しい手法を追跡することができないということも大きな弱点である。上述のことからわかるように、環境設計者は、環境構築の容易さ、手法の使用面での柔軟性の高さ、その環境によって作り出されるソフトウェア製品の品質などについて数多くの取捨選択をしなければならない。

（２）手法の組み込み

ソフトウェア・エンジニアリング環境の構築に関するもう１つの問題は、どのような手法を組み込むかということである。ライフサイクルのすべてがカバーされるように手法を選択することが大切である。これには、ライフサイクル全体をカバーする方法論（すべての手法が統合された）の利用が有効である。手法がまったくまたは一部しか統合されていない場合はカバーしきれない部分が生じることになる。これは重大な問題である。

なぜなら、プロセスに手法によって支援されない部分がある場合は、そこをユーザーが埋めなくてはならないからである。埋められない部分があれば、そこの部分は遂行されないことになってしまう（このようなケースは珍しくない）。

また、ソフトウェア・エンジニアリング環境が部分的にしか統合されない場合は、最初に統合された手法がプロセスに対する環境の視点を決定づけることになる。たとえば、手法として最初に言語とコンパイラと設計用の手法を選択した場合は、構築が進むにつれて言語的視点が強くなる。また、ソフトウェア要件分析用の手法と仕様書作成用の手法を最初に選ぶと、製品情報的な視点が強くなる。最初のやり方（これはボトム・アップ方式である）は、プログラム作成に利用される言語の視点からデータ・オブジェクトを見ようとするものであり、開発の各段階における手法間のデータの受け渡しは言語的色彩が強くなる。もう一方のやり方（これはトップ・ダウン方式である）は、開発の各段階において作成される製品情報を中心とするものであり、手法間のデータの受渡しは、この視点に立って行われる。トップ・ダウン方式とボトム・アップ方式とで最終的な結果が同じになるかどうかはわからない。多くのプログラミング言語が利用される環境の場合は、ボトム・アップ方式のアプローチは困難である。ただし、言語を一種類しか利用しない場合は、このアプローチを利用するとプログラム作成と設計との間に存在する概念上の谷間をそれほど問題なく越えることができる。

（３）手法の欠陥

手法に関する技術的な問題は他にもある。各開発段階の作業を実施する際のプロセスを詳細に調べれば、今日の手法に非常に多くのものが欠落していることがわかるはずである。また、ここ１５年以上、ソフトウェア・エンジニアリング分野では新しい原理が１つも生まれていない。ソフトウェア・エンジニアリングに関する理論が限界まで発展しているのでない。ならば、新しいアプローチを生み出すさらなる努力が必要である。さらにまた、原理に対するユーザーの基礎的な理解を深め、手法に原理がどのように利用されているかがはっきりわかるようにすることは、研究の大切な一分野であり、環境の能力をうまく引き出すことができるか否かに大きな影響を与える。実践面では、ライフサイクルのほとんどすべての段階、特に開発の上流工程（プログラム作成より前の段階）に新しい手法が必

要である。

たとえば、仕様書作成段階においては、仕様書の内容を正確に伝える優れた手法が必要である。ソフトウェア要件分析および設計段階では、開発要員が頭に描くイメージを引き出す優れた手法と引き出された情報を処理する優れた手法が求められている。テスト段階では、テストする情報を作り出す手法とテスト手法を直接結びつけることが必要である。また、開発管理作業においては、特にソフトウェアのサイズ見積り、コストの適正な配分、プロジェクトの流動性といった点についての支援強化が必要である。データベース領域では、複合的な視点が定義できるような手法が必要である。また、タイムリーにデータが利用できるようにする方法や性能を向上させる方法も求められている。要約すれば、利用される手法は、作成されるソフトウェア製品の品質だけでなく、環境構築の難易度にも大きな影響を与えるということである。

3．2．3　自動化に関する問題

自動化は、ソフトウェア・エンジニアリング環境に影響を与える一連の問題を内包している。環境内で使用されるプロセスや手法に関する問題は、環境の潜在的な利点を暗示するものだが、プロセスと手法とを自動化したからといって、これらの問題が解決される（利点が引き出せる）とは限らない。また、自動化によって手法の成し得ることが変わってしまい、その手法に対するユーザーの期待が裏切られる場合もある。

一方、自動化が達成されれば、追跡性が高まり、整合性と完全性のチェックが容易になる。小規模のプログラムは例外だが、どのように手法を統合しようとも、自動化なくしては、情報の追跡は不可能である。自動化は、それなしでは不可能なことを成す能力を手法に与えるだけではなく、その手法を用いて作成されたものの正確さを飛躍的に向上させるトップ・ダウン方式でソフトウェア・エンジニアリング環境を構築する利点は、開発プロセスの厳密さが格段に増すことであろう。そうなれば、開発プロセスを支援するために必要な手法の識別と定義が容易になり、自動化も容易になる。

（1）環境のタイプ

手法の自動化と統合をうまく利用すれば、さまざまなソフトウェア・エンジニアリング環境を構築することができる。これによって構築できる環境の基本的なタイプは4つだが、それらを組み合わせればいろいろな環境を構築することができる。

4つの基本的なタイプとは、閉鎖型、開放型、統合型、定形型である。閉鎖型のソフトウェア・エンジニアリング環境とは、ソフトウェア製品の開発要件を満たす自動化された手法の非常に強固な（固定化された）セットを指す。閉鎖型環境は、拡張や変更が不能である（それが必要な場合は環境全体を作り直さなければならない）。方法論を中心として構築される環境はこの閉鎖型に属す。閉鎖型環境には、整合性や完全性のチェックが簡単に行えるといった利点がある。また、ある閉鎖型環境で作成されたものが同じ方法論を使用する別の閉鎖型環境に容易に移植できることも、（自動化という観点から見た）利点である。これの弱点は、手法間の相互連関が多いため、環境の性能（能率）が低下してしまうことである。開放型環境とは、拡張や改良がいつでもできるものをいう。統合されていない手法を中心に構築される環境はこの開放型に属す。拡張可能性の他に、手法間の相互連関が少ないため高い性能が望めるという利点がある。

開放型環境は独自性が強い（類似する環境がほとんどない）ので、移植性が低いことが弱点である。環境が特築された時点で同一の環境がもう１つ別にあったとしても、その環境が長期間にわたり変化しないということはほとんど考えられない。開放型環境には、整合性と完全性を自動的にチェックする機能や情報を追跡する機能は備わっていないが、この種の機能が実施できるように手法を自動化することは不可能ではない。

統合型環境とは、自動化されたすべての手法が連係して機能し、情報を共用するものをいう。部分的に統合された手法によって構築された環境もこの統合型に属すると考えてよいだろう。プログラムの移植性が高いこと、また、整合性や完全性が自動的にチェックできることなどが統合型の利点である。情報の共用方法はあらかじめ定められているわけでない。これには、自動化された手法間のインタフェースを定義する前処理プログラムやデータベースを利用すればよい。環境の性能が高くなるか低くなるかは、手法間の相互連関の数とその方法によって決まる。

定形型環境とは、値のセットとそれらの値に関する操作のセットが定義されているものをいう。「仮想的に」統合された手法と、オブジェクトを基本とする視点を備えた環境が定形型とみなされる。これは、自動化された各手法に対して情報が透明である（そのまま利用できる）ことを意味する。ソフトウェアの移植性が高いこと、整合性と完全性を高いレベルでチェックできること、情報の追跡が容易なことがこのタイプの利点である。この他、柔軟性が極めて高く新たな手法が追跡しやすい、という利点もある。弱点は環境の性能があまり高くないことである。どのデータがどこで使用されるのかをデータベース機能が追跡しなくてはならないということがネックになっている。

インタフェースは、緩目のないインタフェースとしてマクロ・レベルで定義されるのではなく、各データ・オブジェクトと一緒に「運ばれる」。これによって性能が低下するのか向上するのかを判別するのは厳しい。理想的な環境は定形型だが、ユーザーはこれから閉鎖型環境を作ることもできる（開放型環境の構築も可能だが、その場合は整合性のチェックなどを実施する能力が著しく低くなる）。このように、それぞれのタイプの環境には利点と弱点がある。この他、環境構築の難易度にも差がある。閉鎖的な環境は、方法論が定義されていれば、簡単に構築できる。しかし、ライフサイクル全体を完全にカバーする方法論はまだ確立されていない。開放型環境の構築は容易であり、今日の多くの環境はこのタイプである。

統合型環境の構築は、手法がまだ部分的にしか統合されていないため難しい。定形型環境の構築は、手法により許されるデータの操作がきちんと定義されていれば、統合型環境より簡単である。しかし、完全な開発モデルを基礎とする定形型環境はまだ構築されていない。どのタイプにも、部分的な環境はあるが、ライフサイクル全体をカバーするものは今のところ存在しない。

上記の４つの環境カテゴリをミックスすれば、ざまざまなタイプの環境を作ることが可能である。その場合は、どのような環境を作るかが問題だが、それはソフトウェア製品のアプリケーションの性格に基づいて決めるべきである。信頼性に重きが置かれるアプリケーションならば、製品の品質保証が十分に支援できる閉鎖型または統合型の環境を基にして適切な環境を作ることが良いだろう。

信頼性にあまり重きが置かれないアプリケーションならば、開放型環境のように品質保証にそれほど重点を置かない環境を選ぶべきだろう。また、環境のユーザーの経験のレベル（閉鎖型環境ではその利用法について多くのガイダンス情報が提供されるが、開放型の場合はそうではない）、環境性能（開放型は高く、閉鎖型は低い）、環境の構築に要する時間（開放型は少なく、閉鎖型は多い）など

も環境を選ぶ際に考慮する必要がある。

（2）自動化とプロジェクトの流動性

自動化がプロジェクトの流動性にどの程度対応できるかということも重要な問題である。情報の伝達は、たとえば設計者からプログラム作成者へといった具合に、さまざまな人の間で行われる。情報伝達の形態には、口答によるものや、文書や手法を介するもの、あるいはこれらを組み合わせたものがある。ところが、手法が自動化されると、今までのように開発関係者のいくつものグループが１つの問題について討議し合うことがなくなり、関係者間のコミュニケーションの量が実質的に減る可能性がある。また、手法が自動化された環境では、プロジェクトの関係者がグループごとに隔離されて、環境を通じたコミュニケーションしか図れなくなる恐れもある。

さらにまた、環境が問題の解決者のように見られるようになると、開発者は時間をかけて良い仕事をしようとしなくなる可能性がある。「それは環境にまかせておけ」といった言葉が将来ソフトウェア開発者の口から吐かれるようになるかもしれない。事実、この種の現象がいくつかのケースに現れ始めている。たとえば、自動化、特にプロトタイプの利用によって、問題解決に向けてのアプローチを見つける訓練をほとんど受けていない開発者が増え始めている。

もう１つの問題は、環境が、ソフトウェア製品を開発する際の無秩序な状態または混乱した状態を改善するための手段と見なされがちなことである。この傾向は望ましいものではないので、助長されないように配慮しなくてはならない。これを回避する簡単な方法は、ユーザーに環境の継ぎ目を見せること、すなわち環境の構成要素がどのような形で連係しているかをユーザーに認識させることである。また、ユーザーが環境と統一的な形で対話ができないように環境への共通的なユーザー・インタフェースを提供しないのも１つの方法である。この他、環境の一部の性能を低下させて、ユーザーにフラストレーションを与え、自分で仕事を成し遂げようという意欲を起こさせるのも１つのやり方である。要約すれば、自動化は環境の潜在的な利点を引き出す非常に有効な手段である。ただしどのタイプのソフトウェア・エンジニアリング環境が必要か、また構築されるべきかを決定する際には、考慮すべき事柄がこの他にもたくさんある。さらにまた、自動化はソフトウェア・エンジニアリング環境に調査価値を持たせる要素であるが、その価値を高めるためにはプロセス・モデルを完全に支援しなくてはならない。そうでなければ、自動化の意味が半減してしまう。

３．２．４　今後の課題

ソフトウェア製品開発においてさまざまな目標を達成しようと思えば、必ず二律背反の矛盾に陥ることになる。環境自体がソフトウェア製品である、ということを思い起こす必要がある。ソフトウェア製品の開発についてこれまでに論じたことは、同時にソフトウェア・エンジニアリング環境にも当てはまることである。環境がうまく構築されず、ユーザーのニーズに応えるものにならなければ、優秀なソフトウェア製品がそこから産み出されることは有り得ない。残念なことに、環境の構築を支援する環境というものは存在しない。我々は、今日利用できる手法と自動化支援機能を自らの手で組み合わせて最高の環境を作り出すしかないのである。手法の統合とかプロセス・モデルの完全な支援とか技術的な問題はたくさあるが、ソフトウェア・エンジニアリング環境は、大規模なソフトウェア製品、特に製品保証に重点を置いた製品の開発には極めて有効である。メリットば大きいがそれを引き

出すためには、確固とした方針の下に総合的な努力をする必要がある。解決しなくてはならない重要な問題がもう１つ残っている。それは新しいテクノロジの利用に対するユーザーの消極性の問題である。

３．３　セイフティ・クリティカル・システム開発

人間の安全性に重大な影響を及ぼす機能に係わる情報処理システムを一般に、セーフティ・クリティカル・システムと呼ぶ。例えば、航空機や核燃料サイクル施設の運転を制御する為に開発されたシステム等が挙げられる。セーフティ・クリティカル・システムに係わるソフトウェアにとって最も重要な要件はその品質である。地層処分ヴァーチャル・エンジニアリング・システムはセーフティ・クリティカルな機能が直接的に要求されるシステムとは考えられないが、人間生活の安全性に係わる事象を評価する機能を提供することから、ソフトウェアの品質管理はセーフティ・クリティカル・システム・レベルの対応が理想であると考えられる。

ここでは、地層処分ヴァーチャル・エンジニアリング・システム・ソフトウェアに関連していると考えられるソフトウェアの品質管理（安全性を決定する大規模で複雑なソフトウェア・システムの開発に重点を置いて）に関して報告する。

３．３．１　ソフトウェア・ライフサイクル

ソフトウェアの開発プロセスにおいて、初期段階から長期間にわたって支配的概念となっているのが、ウォータ・フォール・モデルである。1968年の学術会議において「ソフトウェア・エンジニアリング」という言葉が初めて造り出されてから２年後の1970年に、ROYCEによって初めてこのウォータ・フォール・モデルが記述された。ウォータ・フォール・モデルによって、ユーザーの要求事項を定義付けする初期段階から、システム操作し、維持する最終段階までの、連続的な局面のカスケードとして（一般的及び一次的な製造プロセスとの類似によって）、ソフトウェアの開発プロセスを説明することができる。（図３．３－１）それぞれの段階はさらに細分化されるかもしれないが、一次的なプロセスが本質的な概念となっており、各段階は次の段階が開始する前に完了するようになっている。例えば、表３．３－１に列挙されているように、それぞれの段階において文書化される。

上記のモデルが、特にプロジェクト責任者の間で好評を博してきたのは、文書化により明らかに画期的な進展を遂げてきたことによる。

表３．３－１　文書の種類

Stage	Document
Requirements	Software Requirement Specification
Design	Software Design Specification
Coding	Prototype or Preliminary Source Code
Testing	Program Testing Report
Operation	Program User Manual

実際問題、開発プロセスの各段階は、次に述べるステップとして解釈することができるであろう。

- 検討や文書化によって、クライアントの要求事項をカプセル化すること。
- 要求事項を高レベルのソフトウェア設計に変換すること。
- 高レベルのソフトウェア設計をより詳細なモジュラーソフトウェア設計に変換すること。
- 詳細な設計を高級レベル言語コードに実現する。

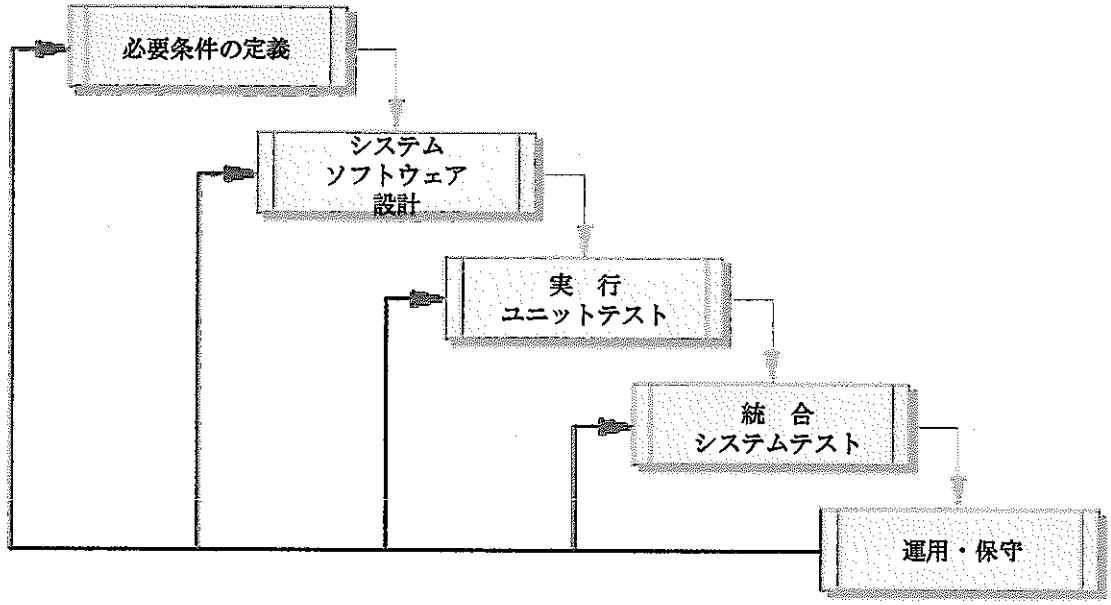

図３．３－１　ウォータ・フォール型開発

プロジェクトの中間段階において、最終的なシステムを定義付ける重要な情報は多くの形態をとる。その形態には、自然言語、構造化言語、フローチャート、プログラム構造ダイアグラム、擬似コード及び最終コードなどがある。それぞれの段階において、検証を行ったり、正確さを追究するために、前方及び後方の比較をすることが必要となるであろう。この理由から、ウォータ・フォール・モデルはしばしばＶ字型のコンフィギュレーションで書き込まれる。そのコンフィギュレーションに直線や矢印が付け加えられて、試験計画やその実行、検証及びトレース経路を示している（HAMBLING 1996年）。この形態によって、ソフトウェアのライフサイクル・モデルを表わすことができる（図３．３－２）。図３．３－１に示されている最初の5段階はさらに9段階に分類される。

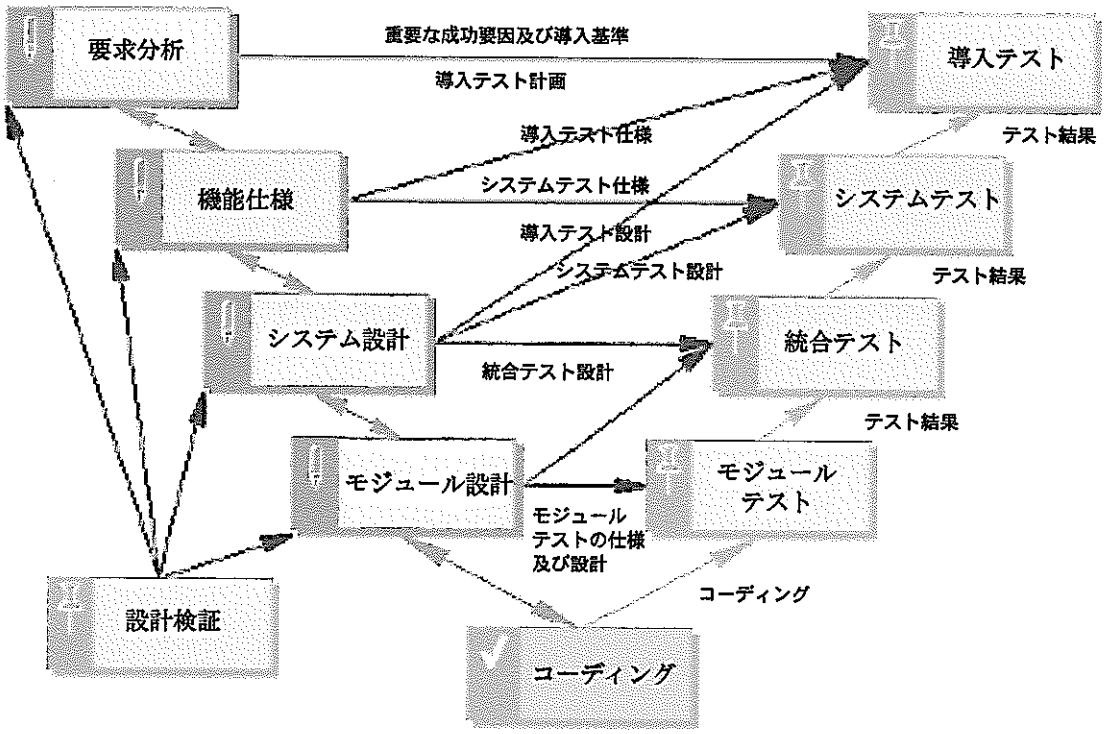

図３．３－２ 一般的ソフトウェア・ライフサイクル・モデル

ライフサイクル・モデルには、通常、「操作」、「要求分析」間の付加的な関連性も含まれているが、それはクライアントの要求事項を修正することによって生じる展開及び変化の状態を示すためのものである。そして、その結果、新しいバージョンもしくは高品質のシステムを開発することができる。

ウォータ・フォール・モデルには、次に述べるようないくつかの短所があると考えられている。ソフトウェア開発プロセスが、線形的で逐次的であるとする仮定は正しいものではない。例えば、設計と要求事項の間及びコード化と設計の間に包括的なフィードバック・ループが生じるのは避けられないのが常識である。しかしながら、高度に相互作用を及ぼし合うプロセスの働きによって、管理上の概念的なモデルはそれに都合のよい計画や記録のチェックポイントと相容れなくなってしまう。その結果、プロセスを反復している間にタイムリミットが来てしまい、引き起こされたエラーはシステムに「凍結」され、繰り越されることになる。プロジェクトの最も難しい段階は、一般的には最初の段階であると考えられている。すなわち、ユーザーの要求事項を把握し分析することである。この段階におけるエラーや脱落をあとで修正するのには非常に多くの時間と費用を要する。

最近、コードを一連のプロトタイプとして開発するという概念に基づいて、ユーザーの要求事項に含まれているすべての機能を徐々に把握していく手法のソフトウェア開発モデルが提唱されている。このモデルを用いると、多数のバージョンやプロトタイプを迅速に開発することが可能となり、適切なシステムが生み出されるまでクライアントが関与し、また、各段階毎でフィードバックが必要となる。このような考えは、プロトタイプ化の次の段階に入る前の「リスク分析」という概念と一緒に、

ソフトウェア開発におけるBoehmのスパイラルモデルに埋め込まれている。（BOEHM 1988年1986年）。図３．３－３を参照。

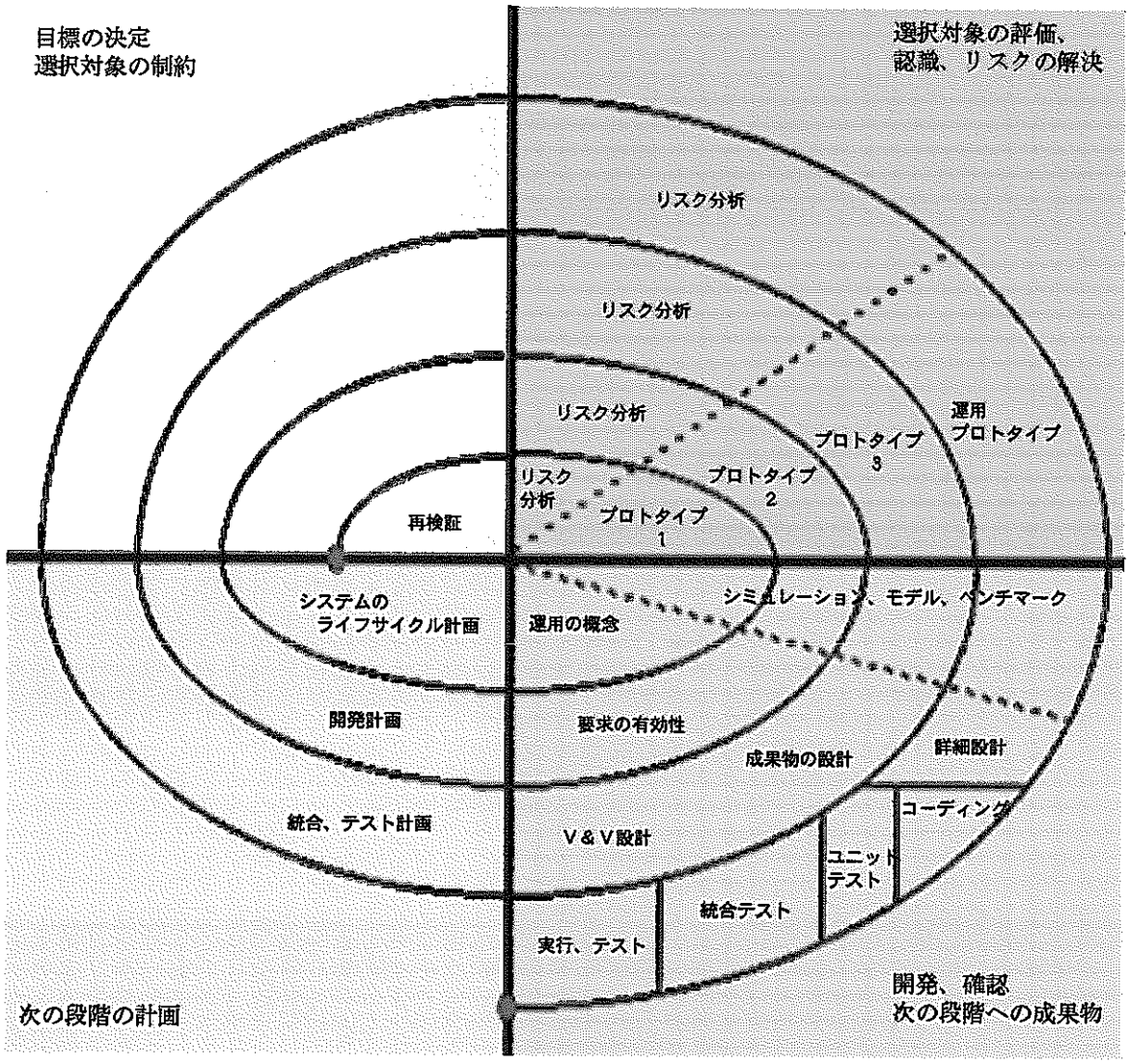

図３．３－３　スパイラルモデル

この非線形プロセスは、当初、プロジェクト管理者たちには好まれていなかった。何故なら、プロジェクトの進行状況を把握することが、どの段階においても明白でなかったし、作業工程が文書にほんの一部しか連結していないためであった。しかしながら、柔軟性と言う点では非常に優れた開発モデルであるといえる。ユーザーの要求を完全に記述していないことによって、「凍結」されることはないであろうが、漸次的に修正されていくことになる。さらに、品質やリスク要因を早期に評価することも可能である。里程標として定義付けされることは、二次的な文書化（研究者が「誤った」システムを構築してしまったならば、適切に文書化したとしてもまぎれのないエラーや誤った認識が隠されてしまうかもしれない）よりもむしろ、ソフトウェア製品自体に重点を置くことに関連しているようである。

３．３．２　セーフティ・クリティカル・システムのソフトウェア、エンジニアリング

セーフティ・クリティカル・システムとは、適切に動作しない場合、人の生命が脅かされたり、環境に多大な損害を与えたりするシステムと定義してよいだろう（SUMMERVILE 1996年、SMITH 1995年）。地層処分ヴァーチャル・エンジニアリング・システムは、間接的もしくは二次的な意味でセーフティ・クリティカル・システムの範疇に入ると考えられる。危険性があると判明している、実際の処分場設計における特性は、工学シミュレーターにおけるエラーの影響を受けたかもしれないが、全体的な意志決定や設計プロセスにおいては（例えば、感度／不確実性解析もしくは追加的な実験や実地調査などによって）そのようなエラーを考慮にいれておかなければならない。

セーフティ・クリティカル・ソフトウェアの開発者は、安全性を要求されないシステムを開発している研究者よりも多くの難しい問題に直面することになる。セーフティ・クリティカル・ソフトウェアの開発者は、通常は、国家的なもしくは国際的な標準や安全性を規制する当局に従う必要がある場合が多い。プロジェクトが安全性に関連しているために、クライアントの要求事項の他に新たに要求事項が加えられるであろう。

セイフティ・クリティカル・システムの特性により、要求事項の分析から確証作業に至るまでの開発プロセス総ての段階にて、特別な注意が必要となる。前述した、通常のソフトウェアのライフサイクルに「安全性ライフサイクル」の概念を加えることを提唱する研究者もいる。図３．３－４には安全性ライフサイクルが例示されている（SUMMERBILLE 1996年）。開発工程初期段階で、危険性を分析するには、システムエンジニアや各分野の専門家および安全性に関するアドバイザーなどによる水平思考（常識や既成概念に基づかない考え方）が必要となる。確認可能な各々の危険性は、フォルトツリー分析などの手法によって、根本的な原因が詳細に分類される。危険の重大性や蓋然性などの要因を用いて、リスクを定量的に評価することが可能になる。

セーフティ・クリティカル・システムの短所を最小限にするたの手法は多数存在し、以下の様な手法例として挙げられる。

- 「例外」を処理するのに優れている形式的な構造化言語を適用すること（例えばAdaもしくはPascal）。
- 最近の軍基準（Military Standard）において強制されている、形式的仕様やそれに関連した確証を適用すること（REESとODDY 1991年；OULD 1990年）。
- 故障を許容できる手法を適用すること。それには、エラー検知、識別や再配置などが含まれている。

形式的手法には、要求事項の形式的仕様、形式的な設計方法論および自動的なコード確証方法などが含まれている。それらは、VDM、Ｚスキーマ及びOBJ法などのようなデータ指向の手法や、CSMもしくはCCSなどのような事象が動因となる方法論や、Petriネットなどのような状態に基づいた手法などに分類される。（SUMMERVILLE 1996年; SMITH 1995年;JOHNSON 1995年）

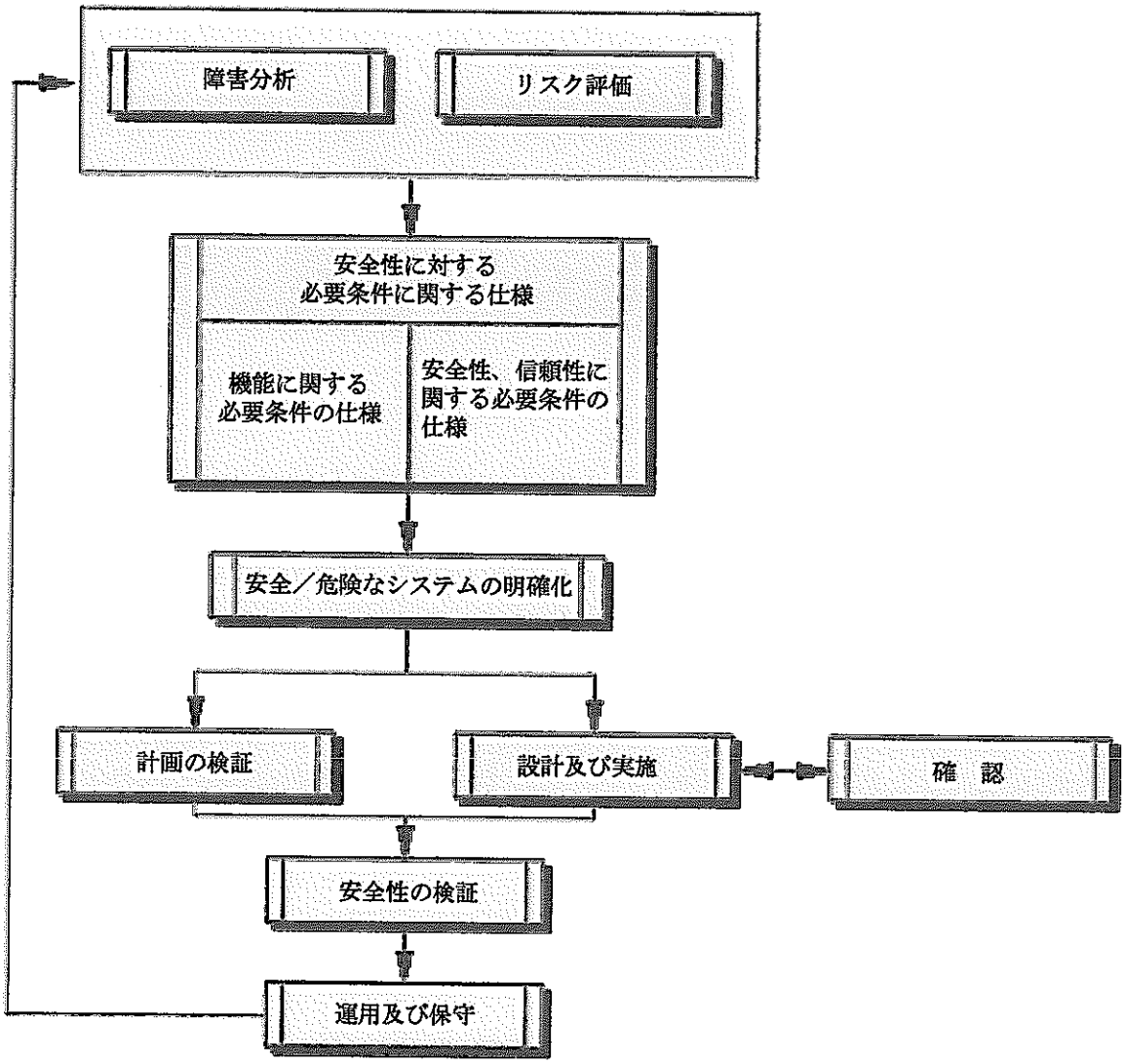

図３．３－４　ソフトウェア安全管理のライフサイクル

SUMMERVILLEは1996年、これらの手法の費用効果性については疑わしいとしている。しかしそれと同時に彼は、これらの手法は現在のところ、おそらく最も効果的な適用方法であろうと考えている。その他にも数多くの考え方があり、次に示す文献に述べられており、その多くは上記の手法に好意的である。（例えば、CRAIGEN他1995年、GERHART他1994年、THOMAS 1993年、FROOME及びMONAHAN 1988年、BOWEN及びHINCHLEY 1995年）。

次の項では、航空宇宙業界や原子力業界において、セーフティ・クリティカル・システムの開発を行っている研究者の現況について紹介する。

（１）航空宇宙業界

ITT Avionics社のHeilは、米国航空宇宙業界におけるリアルタイム埋め込み型システムのための高品質のソフトウェアを開発する際に生じたいくつかの問題について論じている（Heil 1992年）。その

ようなシステムを採用しているものとして、ICBM（大陸間弾道弾）に対抗する陸上戦闘用のシステム、衛星監視システム、機上通信、機上航行、機上コントロールなどのシステムが挙げられる。これらのシステムの多くは、安全性を決定するものとしてよりはむしろ使命を決定するシステムとされている。すなわち、ソフトウェアが適切に働かないことによって、必ずしも人命や環境に危険を及ぼすことにはならず、それよりはむしろ目標とする使命が果たせなくなるということである。それにもかかわらず、設計規模とテストの際に生じる問題は一致する。

HEILは1992年、スパイラルモデルに基づいた迅速なプロトタイプによる反復手法が、従来のウォータ・フォール・モデルよりも優れていることを明確にした。このために操作上及び管理上のいくつかの問題が生じることになった。何故なら、DCD-STD-2167やMIL-STD-1571B等の合衆国における初期のDOD（米国国防省）のソフトウェア規格やその規格が要求する文書はウォータ・フォール・モデルの概念に基づいていたからである。

旧式のシステム開発モデルによって規定されている手法は、関連性がないモニタリング及びテスト手法であった。このことは、TRW社やRationalソフトウェア社に関するDEVLINとROYCEの論文（1996年）においても明らかにされている（Rationalソフトウェア社がウォータ・フォール・モデルの創作者であることは注目に値する）。

これらの研究者たちは、柔軟性に富むライフサイクルの管理という目標に到達するために、Ada言語やO-O（オブジェクト指向）設計概念と共に迅速なプロトタイピング手法を採用した。迅速なプロトタイピング手法の主な利点は次のとおりである。

- **将来の変更に対応できる堅固で柔軟性に富むアーキテクチュアを開発できる能力を有していること。**
- **クライアントによる完全な要求事項が自然に形成され、初期の不完全なソフトウェア・プロトタイプを使ってきた経験から迅速なプロトライプの適用方法をよく理解していること。**
- **非常に軽量化された言語によって、アーキテクチャーに係わるソフトウェア設計の決定はハードウェア／プラットフォーム選択やオペレーション・システムから切り離すことができる。これらの決定を、プロジェクトのライフサイクルの最適段階に後回しにすることも可能であろう。**

最初のうちは、保守的な経営者や現行の開発工程基準を支持する人たちによる反対はあったものの、1987年頃からスパイラル・モデルや迅速なプロトタイプの適用が合衆国の航空宇宙ソフトウェア業界において、事実上一般的になってきていることは明らかである。現在では、スパイラル・モデルの処理の容易さ、コスト管理及び最終的なシステムの性能などの点で優れた手法であることが業界において認められている。さらに、DEVLIN及びROYCE（1996年）は、ソフトウェアの開発にAda/O-O設計／迅速なプロトタイピングを組み合わせた手法を適用することの経済的な利点であるとして次のようなことを強調している。

- **既存のコードを最大限に再利用できること。**
- **複数のシステム上で新しいコードを最大限に適用できること。**
- **開発プロセスにおいて、開発にかかる期間及びコスト、機能性、性能などのどの要素を優先させるべきか否か、等について正確にかつ柔軟に考慮できること。**

しかしながら、この様な利点の反面、非線形プロセスモデルを適用することによって、バージョン

もしくはコンフィギュレーションの管理に関する問題がに生じる可能性はあると言われている。

（2）原子力業界

韓国やカナダの国家的な原子力業界の研究者によって発表された数多くの論文（KANGなど1995年；KIMとHAN 1995年；ICHIYENとJOANNAU 1995年；DAIとSCOTT 1995年；OSTROFF 1995年）は、動力炉の運転を管理するためのセーフティ・クリティカル・ソフトウェアを開発する事、特に自動制御による動力炉閉鎖について考察したものである。韓国においてもカナダにおいても、「合理的な設計プロセス」として知られている機械の状態に基づいた形式的な設計方法が採用されている。

ICHIYENとJOANNOU（1995年）は、その採用が成功していると主張している。設計用CASEツールの開発（DAIとSCOTT 1995年）、動力炉管理の自動テスト及び閉鎖システム（OSTROFF 1995年）の分野における技術も発展している。

3．3．3　ソフトウェアの品質保証

ソフトウェアの品質の正確な定義について全体的に一致するにはいたっていないが、ソフトウェアの品質といった概念については広範囲に論じられている（SUMMERVILLE 1996年；HAMBLING 1996年；GRADYとCASWELL 1987年；SCHULMEYERとMcMANUS 1992年；SMITH 1995年；SCHULMEYER 1992年）。ソフトウェアの品質を管理することは、最終的に発表されたシステムにおける欠陥の数をただ単に最小限にすると言う考えよりも更に広い範囲の概念である。それは、開発プロセス総ての段階において、計画を立て、監視し、品質を維持することに意味している。

品質の概念は多元的であり、異なる定義やモデルが提唱されている。しかしながら、品質モデルが一旦定義されると、適切なライフサイクル・モデルに加えて、ソフトウェアの品質についての全体的な管理プログラムを定義することができるであろう。品質管理は、主に次の3つに分類される。

- **品質保証、すなわち高品質の最終的なソフトウェア製品を作り出す体系的な標準を確立すること。**
- **品質計画、すなわち現在進行中のプロジェクトの性格に合わせて手順を選択し、実行すること。**
- **開発プロセスのすべての段階における品質を監視し管理すること。**

最終的な製品の品質については、SMITH（1995年）が、品質保証に関連する業務活動には次の2つの異なる種類の活動があると述べている。

- **欠陥を認識し取り除くことを目的とした試験のような業務活動**
- **欠陥が発生する可能性を最小限にする設計もしくは要求分析に関する手法を改良するための業務活動**

ソフトウェア・メトリックスとは、ソフトウェア・システム、プロセスもしくは文書に対して行われる様々な種類の評価のことである。SOMMERVILLE（1996年）は、管理メトリックス及び予測メトリックスを区別している。前者は、管理者の意思決定に用いられ、後者は信頼性もしくは複雑性などのように一部のソフトウェアの品質に関連した、いくつかの特質を評価するのに用いられる。両方のメトリックスとも適切な品質モデルを使ってソフトウェアの品質を定量的に予測するためのものである。とりわけ、SOMMERVILLE（1996年）、SMITH（1995年）、GRADYとCASWELL（1987

年）、SCHULMEYERとMcMANUS（1992年）及びSCHULMEYER（1992年）が、様々な実用的なメトリックスについて述べている。

３．３．４　ソフトウェア品質モデル

過去２０年間に、数多くの異なるソフトウェア品質モデルが提唱されている（KIMとHAN 1995年；ICHIYENとJOANNAU 1995年；DAIとSCOTT 1995年；OSTROFF 1995年；SCHULMEYER 1992年；McCALLなど1979年；KITCHENHAM 1987年；KITCHENHAMとPICKARD 1987年；DROMEY 1991年）。すべてのモデルには次に述べるような共通した特徴がある。

（１）ライフサイクルの各段階における品質に寄与するものには、限られた数の外的な品質特性、もしくは高レベルの品質特性があると考えられている。例えば、運用段階においては、McCALLなど（1979年）の定義に対応する効率性、正確性、信頼性、使いやすさ及び安全性などがその品質特性が含まれる。また、システムもしくはコードの低レベルの品質特性、内的な品質特性は数多くあるとも考えられている。これらの品質特性は、今度は、システムやコードの評価可能な特性として表現することができる。そのために、メトリックスが存在するか、もしくはメトリックスを開発することは容易なのである。一般的には、内的な品質特性を外的品質特性に結び付ける重要な要素を定義付けすることが必要とされている。ソフトウェアのライフサイクルのそれぞれの適切な段階において、属性のひとつひとつを定量化することも可能となるであろう。

（２）図３．３－１には、McCALLなど（1979年）のモデルに適用される品質属性及び特性が示されている。概して言えば、このモデルはBOEHM（1978年）、KITCHENHAM（1987年）及びKITCHENHAMとPICKARD（1987年）などによるモデルと類似している。これらのモデルを用いる際に実際に次のような問題が生じる。何組かの外的な属性もしくは内的な特性に語義的な関連性が生じるのは避けられないということである。どのモデルも完璧なものはなく、それらを選択して組み合わせると少々主観的になる。（例えば、正確性や信頼性など）さらに、品質に影響を与える特性と、システムもしくはコードを直接評価ができる特徴との間に有効な数値的な関係が確立される場合もある。

（３）DROMEY（1991年）によって、最近提案されているモデルは、同様な考えに基づいているが、もっと複雑なものである。Dromeyは、データ型、定数、変数及び記録からオブジェクト（抽象的なデータタイプ）、インタフェースそしてプログラム全体にいたるまでのオブジェクトの段階的分類もしくは構造形態について考察している。それぞれのレベルにおいて、正確性、構造、モジュール性、説明の４つの特性が（優先順位もこの順序で）認識されている。それぞれの特性の種類やオブジェクトのレベルにおける欠陥の全体的な数を評価することによって、品質が定量評価される。欠陥は最も低いレベルの構造形態と結合し、その構造形態においてその欠陥は属性となる。上記のDromeyの見解は、非常に興味深い。いろいろの点で、Dromeyのモデルを用いた品質評価は、コンパイラ言語の厳密性に左右される。このことによって、ソフトウェアの開発環境における品質に関する責任の少なくとも一部が、プログラマーから設計者に転嫁されるものとも考えられる。

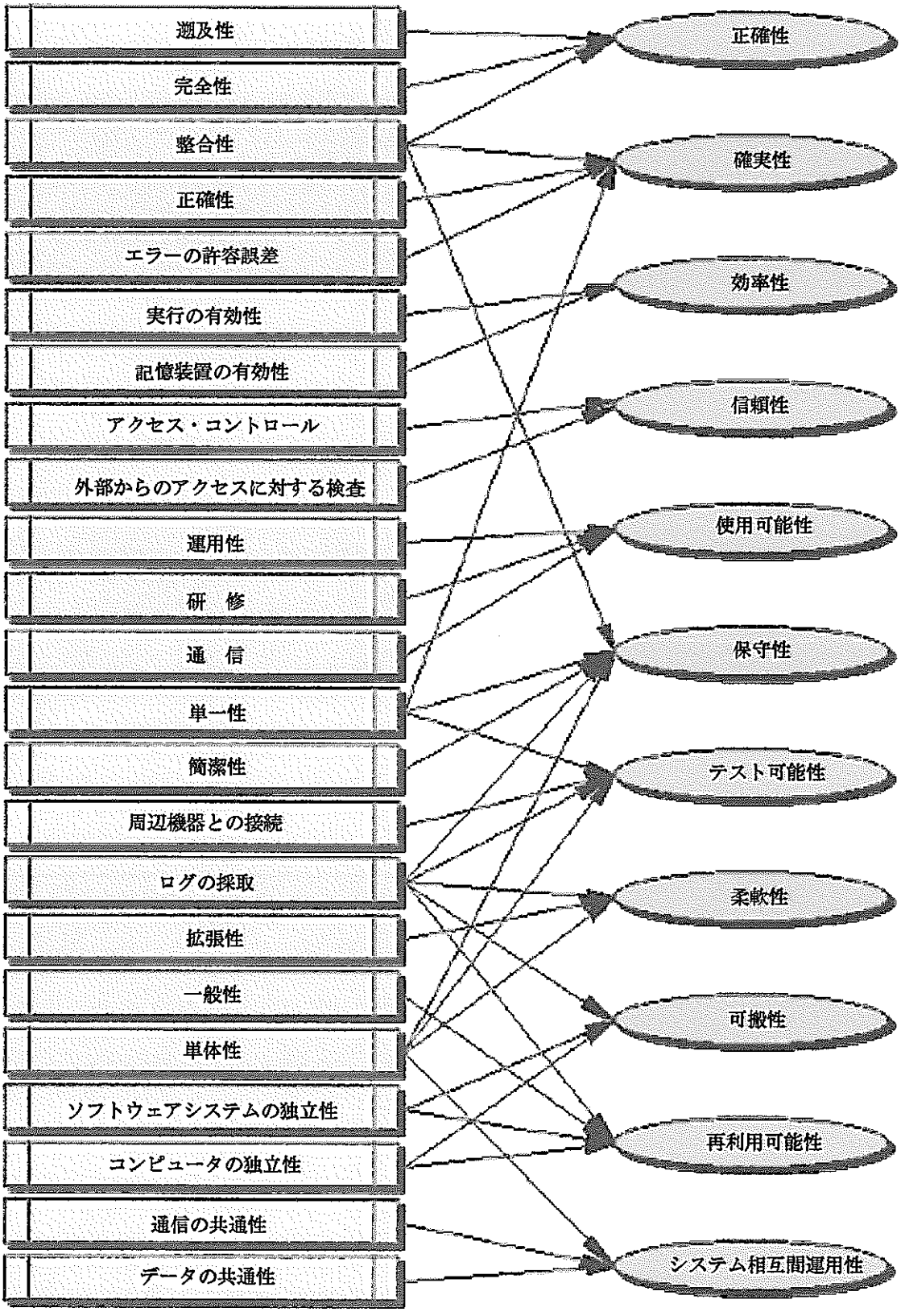

図３．３－５　ソフトウェアの品質に影響する要求の属性と課題

オブジェクト指向の設計手法を適用することにより、ソフトウェア・ライフサイクルの管理や要求事項の変更管理の点で有利になると考えられるが、ソフトウェアの品質管理活動を行うことは難しくなると考える人もいる。しかし、HAMBLING（1996年）は、これらの困難でも特別なO-O（オブジェクト指向）ソフトウェア・ライフサイクル・モデルを適用することにより克服できるであろうと述べている。同様に、O-O（オブジェクト指向）手法を迅速なプロトタイプ手法と共に用いる際、スパイラルモデルは品質という観点では最良のライフサイクルモデルと定義するのは難しい。

３．３．５　コード評価

ある一定のタイプのソースコードを検証することができるソフトウェア・ツールは数多くある。これらのソフトウェアツールを使用すれば、前述した価値のある特性に関連している一連のメトリックによる静的分析及び定量的な評価が可能になる。プログラミング・リサーチ・アソシエーツ社の製品、QAFORTRAN、QAC, QA C++などは言及する価値のある製品である。

動的に分析を行ったり、実行中のコードの性能を監視することのできるツールも数多くあり、それを支援するいくつかのコンパイラもある。動的な分析によってチェックされる代表的なものとして挙げられるのが、変数の適用、違法アレイ（配列）インデックス（索引）の適用、情報のプロ・ファイリングなどで、それらにおいて、コードはすでに分割され、条件付きの分岐もおこなわれている。新しいコードが開発されている時、このようなツールは開発研究者と試験者の両者が入手できるものでなければならない。地層処分ヴァーチャル・エンジニアリング・システムの性質が多種多様であるために、プロジェクトの様々な面を支援するための特別な用途のソフトウェア・ツールを開発することが必要となろう。

３．３．６　ソフトウェアライフサイクル管理

（１）要求事項の定義付け及びトレーサビリティ

クライアントの要求事項を正確に定義付けすることは、おそらくソフトウェア開発のすべてのプロセスにおいて、最も難しい仕事であろう。従来のウォータ・フォール・モデルにでは、上記の定義付けを開発の初期の段階で完了しておかなければならないとされている。ソフトウェアの開発プロセスにスパイラルモデルを適用することにより、クライアントと開発研究者間において、要求事項に関する共通認識を持つ事がが可能になり、長期間にわたって同意が継続し、互いに最大限の協議をおこなうことができるようなる。とはいえ、ソフトウェアの開発をする前に一定の箇所において、ソフトウェア要求仕様書（SRS）あるいはそれに相当する文書が存在しなければならない。この文書が、クライアントと開発研究者間の同意事項の基盤となる。この文書にはクライアントの要求事項がすべて含まれていなければならない。

SOMMERVILLE（1996年）は、必要事項を機能的なものと機能的ではないものとの２つのカテゴリーに分類している。最初のカテゴリーでは、ソフトウェアが実際に機能しなければならないことが規定されている。２つめのカテゴリーでは、ハードウェアの能力、品質及び安全性に関する要求事項そしてリアルタイムメッセージや並列処理などを含んだその他の規準などの様々な要求によってシステムが受ける制限事項が説明されている。

特に、セーフティ・クリティカル・ソフトウェアの開発プロジェクトにおいては、安全性に関する

要求事項が、分析及び編集されなければならない非機能的な必要事項の大部分を占めている。要求分析の主な目的は、クライアントの要求事項を確実に完璧なものにし、首尾一貫させ、妥当で実現可能なものにすることである。

開発研究者がソフトウェアの機能要求事項を日常言語よりも明確な形で規定することは一般的なことである。これは、構造化された自然言語もしくは形態に基づいた要求事項からすべてのプログラムを説明する言語まで適用されている。形式的手法のように、これらの手法は要求事項のトレーサビリティを補助するものであり、要求事項の発展や変更の際に、特別に重要となり、必要とされる。原則的には、システム設計の一部にそれぞれの要求事項を示すことができなければならない。密接に関連している要求事項は、変更の影響が系統立って扱われるようにソフトウェア要求仕様書（SRS）において相互参照されなければならない。

WATKINSとNEAL（1992年）は、正確なトレーサビリティがきわめて重要であることを強調している。彼らは、完全なトレーサビリティとは、どのバージョンのソフトウェアも、どのソースあるいはバージョンから指令されたものなのか仕様書や検証文書によってさかのぼることができることと指摘している。完全なトレーサビリティを実現するために、非常に系統立った手法を文書化やバージョンのナンバリング、及び分類に適用することがきわめて重要である。その構造には次のような特徴がある。

- 要求事項が文書において首尾一貫していること。
- 個々の要求事項の毎に、パラグラフに基づいた独特のナンバリングシステムを用いていること。
- 終わり頃に文書のパラグラフへの追跡や修正があること。
- 削除されたどのパラグラフの番号も「削除」タグと共に控えていること。

それらのトレーサビリティ機能は、マイクロソフト社製の「FoxPro」リレーショナル・データベースに基づいたあるCASEツールに組み込まれている。

MARTINとTSAI（1990年）は、安全性に関連したプロジェクトにおいて、クライアントの要求事項を異なるチームによってN割りに検査するのが有利であると提唱した。数多くの研究者たちは、特に安定性を決定するような複雑なシステムに合わせて、クライアントの要求を分析する自動化手法を開発している（WARDLE 1990年；BIRCHとWHITELEY 1989年；HAYLEYとPIRBHAI 1987年）。その他の手法としては、システムユーザーのあらゆる種類の見解を分析することも含まれている（KOTONYAとSOMMERVILLE 1992年）。

（2）ソフトウェア・ライフサイクル管理構造

ソフトウェア・ライフサイクルの管理に関連している活動の多くは適切な文書の管理（様々なバージョンのコードそのものも含めて）やそれらの文書変更に関係している。このために、前述したように、正確に定義付けされた管理構造が必要とされる。

地層処分ヴァーチャル・エンジニアリング・システム・プロジェクトにおいて、地質学的、化学的及び工学的な情報を含んだデータベースや、固有の解析コードもしくはモジュールを取り入れた大規模なシステムを管理する必要がある。セイフティー・クリティカルと言う視点から言えば、これは副次的なものであろう。従って、次に述べるようなソフトウェア管理構造をそなえていることが望まし

いと思われる。

- すべての開発に関する機能は、「品質管理者」によって管理され、品質管理者は「プロジェクト・ディレクター」に直接報告する。
- 「ソフトウェア及びデータベース・ライブラリアン」は、図書（品質保証や安全性に関する文書及びデータベースが含まれているすべての関連文書やプロジェクト・コードのバージョンを管理する）とシステム開発の様々な段階において適用されるような一連のCASEツールの両方を担当する。
- データベースもコンピュータ・コードも管理、開発及び確認する必要があるので、これらの役割を担当する二人の責任者を選定することが望ましい。
- 「開発責任者」は、コードの生成及びインテグレーションの責任を担うプログラマー、設計者及び技術的な専門家から成るチームを率いる。この開発活動は、すべてのV&V（検証・確認）、モニタリング及び管理活動とは分離されなければならない。
- 安全性に関する責任者は、安全監査のチームを率いて、安全性に関連するすべての事項において（規制事項や標準に従うことも含まれる）その要求事項を確実に満たすようにしなければならない。この活動は、コードの開発やデータベースの管理に関する活動とは別個のものでなくてはならない。

確立したコミュニケーション形式が適切に遵守されているならば、ある程度の活動を組み合わせることも可能であろう。しかしながら、コードやデータベース等の開発から管理、確証及び検証を分離し、プロジェクトを進行させることは必要不可欠なことである。

何人かの重要な人材を「ソフトウェアバージョン及びコンフィギュレーション管理委員会（SVCCC：以下「承認委員会」）」に任命する必要があるであろう。この委員会によってのみ、コードの変更が正式に認可され、あるいは数多くの変更があると、それを新しいバージョンとして更新することが認可されるのである。このような委員会の機能については、後述することにする。

次に、ソフトウェアの維持、最終的にはライフサイクルの展開ということについて（例えば、クライアントの要求事項の変更に対応するシステムの修正）考察する。

3．3．7　ソフトウェアの保守、バージョン及び要求事項の変更管理

ソフトウェアシステムの開発過程において、システムやプログラム単体の試験、確証、検証の結果、数多くの変更が必要とされるであろうと予測される。ソフトウェアを運用する際、時々、他の欠陥が発見される事もある。そして、これらは、単なるコード化の際のエラーもしくは根本的な設計上のエラーかもしれない。現行のバージョンのシステムを運用したことによって、あるいは現行のシステムの変更等の外的な理由から、要求事項を修正することに対応して、かなり広範囲にシステムを修正する必要性が最終的には生じる場合もある。必要とされる、このタイプの（ソフトウェアの保守）変更をすべて行う際には、変更によってシステム全体に対してどれほどの影響が及ぼされるのかを考慮しなければならない。

新しいバージョンのシステムの開発を決定する前にある程度の変更がそれに先だって行われるのが一般的である。要求された変更事項はすべて文書化されなければならない。修正することが承認委員会に承認されると、モジュールは「改訂中」とされ、プログラマーはこの作業を担当することになり、新しいバージョンのシステムを評価するという仕事が一人一人の担当者（検査者）に課せられる。評価の結果、承認委員会によって承認されると、そのコードは更新され、「改訂中」というフラッグは

新しいバージョンのコードから取り外される。

一連のテストプログラムが更新され、「固定された」バージョンのコードが有効になるような新しいテスト方法もそれに含まれるようになると、ソフトウェアシステムの修正は完了する。同時に、どんな修正がおこなわれたのかをはっきり示すために、すべての関連文書が変更されなければならない。変更によって品質や安全性に影響が及ぼされる場合には品質や安全性に関連する文書も変更されなければならない。例えばシステムの複雑性に関連しているメトリックスのような、いくつかのソフトウェア・メトリックスによって、操作中にみつかった欠陥などから生じる、ソフトウェア保守の際の相対的なコストを半定量的に予測することが可能である。HINLEY（1996年）は、高い保守コストの原因解明という理由から、ソフトウェアの保守に新しいプロセス指向のモデルを提案している。

ソフトウェア・バージョンやコンフィギュレーション管理に関していくつかのCASEツールが存在し、その中にはUNIXプラットフォームで操作できるソースコード・コントロール・システム（SCCS; ROCHKIND 1975年）と改訂コントロールシステム（RCS;TICHY 1985年）が含まれる。前者は、すべてのバージョンのモジュールを記憶し、更新し、検索するための装置や、更新を管理したり、一つ一つの記録された変更に対しての「だれが、いつ、どこで、なぜ」を記録するための装置を提供するものである。そして、すべてのバージョンのシステムを永久に記憶することができる多容量の“worn"（write-once-read-many：一度の書き込みで多くを読み取ることのきる）記憶装置も備えている。これらは、タイム・スタンピングによって、アクセス可能である。この手法によって、正常なシステムのバックアップ方法が完全なものになる。

３．３．８　独立型の検証及び確証

３．３．８．１　従来の検証及び確証手法

「検証」と「確証」という言葉は業界によって異なる文脈で使われている。科学的、工学的解析コードのような現象をシュミレーションする解析コードでの意味とワードプロセッサ・ソフトウェアや表計算プログラムのような非科学的ソフトウェアにおけるそれらの意味は、かなり異なる。そこで、ここでは、ソフトウェアエンジニアリング業界において広く普及している概念を紹介することにする。つまり、検証とはソフトウェアが、プログラミングの技術上、要求された仕様に従っていることを意味し、確証とは出来上がったプログラムが本当にユーザーの要求事項を満たしているかどうかを検査する事を意味する。しかしながら、しばしば、この２つの言葉は同義語として扱われることもある。

ソフトウェアのテスト方法は大きく２つの種類に分類される。すなわち静的なテスト方法と動的なテスト方法である。静的なテストは検証に関連しており、つまりすべての段階において要求事項が正確に解釈されているかどうかをチェックすることである。（図３．５－１）（確証のための）動的なテストは、実行可能なプロトタイプの動きを調べるために、その全てもしくは一部を作動させて実行される。

動的なテストは、統計的なテストと欠陥テストに細分化することができる。統計的なテストにおいては、システムの性能の信頼性を評価するのに平均故障間隔（MTBF）のようないくつかの信頼性に関するメトリックが用いられるのに対して、欠陥テストは欠陥を認識し、それを取り除くことを意図したものである。欠陥テストはたいていユニットやモジュールの初期段階において適用されるもので、それが完全なシステムテストを形成し、その最終的な段階がカスタマー受入テストである。この段階

的アプローチは特にオブジェクト指向のシステムに適用される。

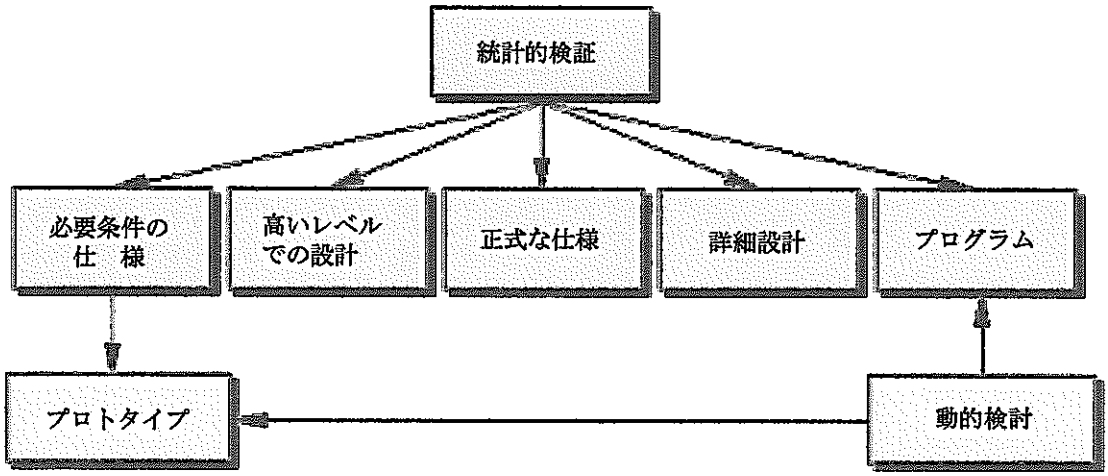

図３．３－６　静的、動的テスト

テストにかかる費用が高いということは、多くの非機能的な制限事項を有する複雑なリアルタイムのシステムを徹底的にテストすることはほとんど不可能であることを意味している。従って、費用対効果を最大限にするためには、全体的なソフトウェアの品質管理の一部であるテスト・プランを入念に作成しなければならない。適用される設計手法に応じて、テスト手法はトップダウンもしくはボトムアップ方式にすることが可能であろう。後者は統合テストとも称される。欠陥テストは次のようにさらに分類することができる。

- **機能的なブラックボックス・テスト**
- **構造的なホワイトボックス・テスト**
- **インタフェース・テスト**

このことについては、とりわけSOMMERVILLE（1996年）とJACOBSON（1992年）によってより詳しく検討されている。信頼性のチェックについては、ストレスや誤用をシミュレーションできるような状態で、すなわち極端な状態もしくは違法入力状態でソフトウェアがテストされることが重要である。

システムを初めて公表する際には、一つ一つの構成要素とシステム全体の性能を確証するような広範囲の適切な一連のテストが必要である。これらのテスト結果は、それにふさわしい専門家の承認を受けなければならない。このテストは、構成要素が修正されたり更新されたりする度に、その変更が何らかの副作用を及ぼしていないことを確認するために、毎回おこなわなければならない。さらに、システムに何らかの修正が加えられる時には、関係組織（委員会等）によって既に証明されたテスト結果と共にそれに対するテスト行われる。テスト結果にいかなる変動があっても、承認委員会が認めない場合は、更新する事が出来ない。テストが削除されるのは、そのテストの機能がシステムにおいて必要ではないことが確認されたり、もしくは適切な専門家によって同等の別のテストが受け入れられた場合だけである。これは、多くの時間を要する方法である。しかしながら、この方法によって自

動的に多くの比較が行われる。この方法を開発することを優先しなければならないであろう。感度テストは、浮動小数点出力が分析されている場合、各モジュールが達成しなければならない精度を示す。

自動的な比較データが最も難しいタイプのデータは、視聴覚データである。この能力を有している強力な汎用パッケージは今のところ存在しないと思われる。地層処分ヴァーチャル・エンジニアリング・システムは、マルチメディア・インタフェースに重点を置いているので、この分野には注目していかなければならないであろう。

3．3．8．2　セーフティ・クリティカル・システムの検証と確証（V&V）

セーフティー・クリティカル・システムの場合、検証・確証作業をより厳密に実行することが必要不可欠となる。理想的には、確認や検証作業の後、ソフトウェアは高度な信頼性を有していなければならない。しかしながら、CHUDLEIGH（1989, 1990年）によると、完全な信頼性をもつソフトウェア開発モデルは存在しない。また、彼は、発生しうるシステムの状態の数は天文学的な数になってしまうため、大規模なシステムの入力／出力状態をすべて徹底的にテストすることは不可能である、とも述べている。よって、そのため、「正確なソフトウェア」よりは「堅固なソフトウェア」、すなわち「故障に耐えられる」ソフトウェアを作り出すことを目指す方が現実的と考えられている。

故障に耐えられる動作は次に述べる２つの種類に分類できる。

- **ソフトウェアの局地的な環境における故障から生じるエラー**

  例えば、無効入力データ、あるいはセンサーの誤動作による不正確な読み取りもしくはハードウェアのエラー、等が挙げられる。ある程度までは、これらの動作を予測し、例外対処技術によって処理することは可能であろう。

- **未発見のソフトウェア・エラーによって生じる欠陥**

  予測できない一般的な欠陥は、SMITH（1995年）によって説明されているように、様々な手法によって絶対的に安全なものにすることができる。この手法の中には、多重プログラミングや様々なエラー訂正及び復旧手法などが含まれる。

その他の研究者たちは、セーフティ・クリティカル・システムの動作を評価するために、欠陥を慎重に導入する技術などを適用している（VOASとMILLER 1995、1996年）。セーフティ・クリティカル・システムをテストする従来の手法の限界の一つとして挙げられるのが、セーフティ・クリティカル・システムの故障と安全性には影響がないシステムの故障を区別できないことである。例えば、セーフティ・クリティカル・ソフトウェアをテストする際には、初期の要求事項を定義付けする段階において行われる危険分析及びフォールトツリー（過失系統樹）分析などを適用するべきである。その中には、FMEA（fault mode and effect analysis：故障モード及び効果分析）のような手法も含まれている（CHUDLEIGH 1989年）。その他の研究者たち（HUGUEとSCALZO 1995年）は、もし出来ることならば、セーフティ・クリティカルな構成要素をコードの他の部分から切り離して、出来る限り単純な構造にすることを主張している。

最近報告されたことによると、MUNSON（1990年）は、欠陥はコード全体に均一には配分されず、語義的に複雑なもしくは意志決定が複雑な部分に集中しがちである。この概念を認識することによっ

て、テスト手法がより効率的になるであろう。形式的設計手法が適用される場合、最終的なコードの正確性を確証するために統合された自動的な承諾テストツールが開発される（WARD 1993年）。しかしながら、このことによって必ずしも正確性や有効性を確認できるわけではない。複雑なソフトウェアシステムにおける、安全性を決定する局面をテストするには、他の従来のV&V手法は、不適切であることは、広く認められている。要求分析もしくは設計上のミスから生じる、安全性を決定する論理上のエラーによって多くの欠陥が生まれる可能性もある（WARD 1993年；GOWENとCOLIOFELLO 1994年；GOWEN 1993年；LUTZ 1993年）。このような場合、最良の方法はおそらく統合されたCASEツール環境においてソフトウェアを設計し、テストすることであろう。様々な不完全な手法があることは報告されているが、必要なすべてのテストをおこなうことができるような、十分に洗練された使途の多い一連のツールはないようである。

3．3．8．3　FORTRAN数値コードの検証と確認（V&V）

少々異なる一連の問題については、「検証」と「確証」にはしばしば異なる意味が付けられる。数値的には、検証は、入力条件が同一であるならば、モデル化されるシステムの動作に関するモデルの絶対的な正確性をチェックすることを意味している。ソフトウェア・エンジニアリング的な意味合いのV&Vはすでに完成しているように思われる。すなわち数値エラーを引き起こしそうな欠陥はすべて取り除かれている。

浮動小数点計算については、数値の端数を四捨五入することによって生じるエラーに関連した問題は残っている。コード化を適切にすることによって、これらのエラーを最小限に抑えなければならない。例えば、二倍精度変数の適用などである。GUILANとVIGNES（1994年）は、ディーゼルエンジンからの熱の放出をモデル化するコードにおけるこのタイプのエラーを最小限にするのに、確率計算及び感度解析を適用することについて言及している。

IDA（1996年）は、いくつかの電磁的計算を「確証」する際に、多重コードを使って規範的もしくは分析的な解法と比較した。LEEたち（1990年）は、割れ目のはいった岩石における核種の移行に関する新しいモデルを確証することについて考察している。そして、彼らは、他のコードや限定された分析的な解法から得られた結果とこのモデルをも比較している。

数値モデルの他の手法として挙げられるのが、実験データと比較することである。ORESKESたち（1994年）の興味深い論文においては、地球科学における数値モデルの確認について考察されている。この論文において指摘されていることは、数値モデルが閉鎖的なシステムでなかったとしても、もしくは必要とされている入力データが不完全である場合には、完全に検証することはできないかもしれないということである。実際問題、このような状況は、熱力学上の基本的なデータが入力できない場合もしくは動的な要因を無視することが出来ないとき、平衡地球化学モデルにおいてしばしば発生する。同様に、水文地質学的流動モデルが検証されるのは、関連しているすべての機能の関係（例えば、岩石の透水性と空隙率の関係など）がはっきりしていて、問題に関係しているすべての領域において検証がおこわれている場合だけである。

３．３．９　地層処分ヴァーチャル・エンジニアリング・システムに関連した特別な問題

地層処分ヴァーチャル・エンジニアリング・システムプロジェクトには、標準的なソフトウェア開発プロジェクトの領域を少々越える数多くの特徴がある。その中で、特に顕著な２つの特徴が、そのプロジェクトの持続期間と適用範囲である。プロジェクトの開発期間が１０年以上になると、標準的なソフトウェアサイクルよりも非常に長いものになるであろう。このような長期間にわたって、すべての技術分野において（ハードウェアとアーキテクチュア、言語とコンパイラ、モデル化とアルゴリズム）注目に値する研究開発が進められていくであろう。

プロジェクトの目的は、適切な研究手法を取り入れていく一方で、技術の発展に伴って必要とされる作業の繰り返しを回避することである。これらは矛盾している目的であり、重要な設計に挑戦してみることにもつながる。良質のソフトウェア・エンジニアリングは、状態を改善するための一つの方法である。しかしながら、ソフトウェア・モジュールの多くは様々なソースから取り入れた既存のものであり、実行できる時だけ適用できるという場合もあることが予測される。このことは、ソフトウェアを適切に適用することとはいえない。

遺産的ソフトウェアは根本的な問題を引き起こす要因の一つである。進歩的で実際的な唯一の方法は、一つ一つのサード・パーティのモジュールの場合を非常に注意深く考慮して、修正や代用が比較的容易なインタフェースを設計することであるように思われる。このことを可能にする一つの方法として挙げられることが、モジュールの構造における変動に容易に対応することができる、抽象データモデル（ADM）を用いることである。ADMは、データセットの典拠、品質に関する情報及び並列アーキテクチュアへのマッピングなどのメタデータをサポートすることができる。ADMへアクセスすることによって、地層処分ヴァーチャル・エンジニアリング・システムにとって重要な課題である連成問題が解決され、システムにおけるエラーの伝播を管理することもできる。

テストされていないデータの経路もしくは修正を受けるシステムの領域へのアクセスを管理するというシステム保守機能の一部としてもADMを適用することができる。そして、プロジェクトにおいては、視覚化、動画及び音声などのマルチメディアデータの重要性についても強調されている。特に、プロジェクトに関わる様々な試みを管理するために自動的な検証手法が必要とされるので、データを統合したり、テストすることにより、困難な問題が生じる。

最終的には、特にプロジェクトの性質にも関係し、異なるサイトにおいて開発が進められるという事実にも関連するセキュリティの問題がある。システムがいったんその信用を落とすと、それを取り戻すのは難しいので、リスクの評価／受入に関する調査が注意深く、プロジェクトの早期段階で行われなければならない。これらのリスクを最小限にくいとめるための、暗号化、確証及びデジタル記号を用いた数多くの戦略がある。しかしながら、確実なシステムにかかるコストは非常に大きく、レベルも適切に設定する必要がある。パブリック・ドメインに公開されたPGPは、多くの便利な機能を有しており、プロトコルの確証や伝達にそのシステムを取り入れることができる。多くの国々において、データの暗号化に関して法的な規制が行われているということも心に留めておく必要があるであろう。

３．４　まとめ

地層処分ヴァーチャル・エンジニアリング・システムプロジェクトにおいて実施れるソフトウェア・エンジニアリングに関し適当と考えられる推奨事項を示す。推奨事項は次の４つに分類される。

Ｄ：ソフトウェアの設計に関する推奨事項
Ｑ：ソフトウェアの品質管理に関する推奨事項
Ｌ：ソフトウェアのライフサイクルの管理に関する推奨事項
Ｖ：ソフトウェアのテスト及び確証に関する推奨事項

（１）設計に関する推奨事項

Ｄ１：順次システムプロトタイプを開発して、スパイラルプロセスモデルを地層処分ヴァーチャル・エンジニアリング・システムに採用することが推奨される。

Ｄ２：既存のサード・パーティ・コードが数多く取り入れられると思われるので、形式的な設計手法の適用やAdaやその後に使われる言語などの特定の安全性指向の言語の採用を推奨することは不適切であろう。

Ｄ３：同様の理由から、新しい数値コードや旧式の数値コード、新しい視覚化システムとデータベースを組み合わせて適用していることからも、それがオブジェクト指向の設計戦略にたやすく移行することはないであろう。しかしながら、オブジェクト指向の手法によって、システムにおいてGUIやデータベースなどの新しい構成要素が形成されるであろう。オブジェクト指向の手法によってシステムの維持性は高められなければならない。

Ｄ４：検知されていないバグや無効なもしくは不正な入力データの両方に関して最大限に故障に耐えられるように、新しいコードのモジュールは設計されなければならない。

（２）品質に関する推奨事項

Ｑ１：メトリックスやモデル（特にエスプリ財団によるREQUESTプロジェクトにおいて1987年に開発された。KITCHENHAMのCOQUAMOモデル及びDROMEYによって段階的に分類されたオブジェクトに基づく品質モデル：1991年）を検討してから、地層処分ヴァーチャル・エンジニアリング・システムの研究開発グループは、グループ独自の優先された品質保証手法を定義付けしなければならない。

Ｑ２：製品ができるとすぐに品質保証グループ（管理者）を決めなければならず、そのグループ及び管理者は、システムをテストし、製品とプロセスの品質保証をするという二重の役割を果たさなければならない。

Ｑ３：品質グループは管理上（出来れば地理的にも）ソフトウェア開発チームから独立していなければならない。

（３）ライフサイクルに関する推奨事項

Ｌ１：図３．４－１に示されているようなタイプの管理構造がつくられなければならない。ライフラリアンとそのスタッフはすべてのバージョンのコードとそれに関連した文献を保管する責任がある。

L２：例えば次に述べるような事柄についても考察されているように、すべての文献やすべての例のソフトウェアの変更を管理するために厳密なプロトコルが定義付けされなければならない。

・エラーがライフラリアンに報告される。
・エラーの重大性が専門家によって評価される。
・ライフラリアンはユーザーにその問題を伝える。
・SVCCCが訂正方法を認可する。
・専門家がエラーを訂正し、バグをチェックする。
・品質／テストグループによって除染が別個に検証される。
・新しいコードが受け入れられ、もし必要であれば、新しいレベル／バージョンが認可される。

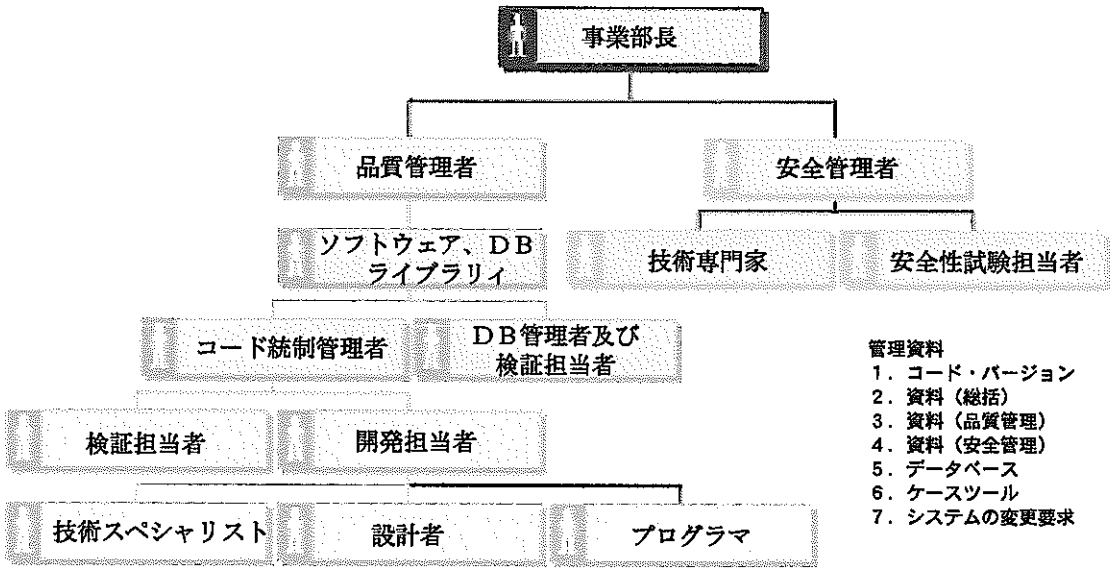

図３．４－２　本プロジェクトに推奨できる体制

L３：ソフトウェアの開発をサポートするための広い範囲のCASEツールがすべての適切なサイトに適用可能でなければならない。新しいコードの受入基準の一部では、静的及び動的テストの両方において、妥当な最小限の基準が満たされていなければならない。これらのテストは開発研究者によって及びSVCCによって指名された人たちによっても行われなければならない。適切なツールがない場合には、そのようなツールを開発するよう努力しなければならない。

L４：プロジェクトの初期段階において、完全なリスク分析／受入に基づいて一連の防御手段及び認証手順を遂行しなければならない。

L５：SCCSやRCSのようなCASEツールもしくはそれよりも優れた代替的なツールをコードやバージョンの管理に適用しなければならない。その際には、特別な用途の長期保存用ハードウェアやバックアップ手法を適用しなければならない。

（4）確証／テストに関する推奨事項

プログラムにおいては、新しいコードと継承コードが併用されるであろう。データベースの典拠や信頼性は大きく変動すると思われる。このことによって、確証やデータの保全性に関して様々な問題が生じるであろう。次に述べるような事柄が推奨される。

Ｖ１：新しいコードはプロジェクトの品質基準を満たしていなくてはならず、そして構成単位として及びシステムの一部として徹底的にテストされなければならない。

Ｖ２：継承コードは厳密に選別されなければならない。継承コードの典拠や文書化されている事前のテストによって、どれだけのテストが追跡されなければならないかが決定されるであろう。上記の決定は、その分野の専門的な知識を有する研究者による技術的な判断に基づく。（統合化された最終的なシステムのテストがいつも必要であることは当然である。）

Ｖ３：全体的なシステムのテストには、次に述べる事柄が含まれるであろう。

・入力／出力状態を最大限に網羅するように設計されたテスト
・ストレスのテスト、すなわち、非合法的なもしくは不正な入力、
・あるいは導入されてしまった欠陥を検討すること

Ｖ４：抽象データモデル（ADM）は、データのフォーマット化や統合に関する問題を単純化するのに用いることが出来る。そして、一つ一つの数値モジュールにおいて、それぞれの指令におけるアプリケーションコードとADM間のフォーマットを変換するために、前処理プロセッサーや後処理プロセッサーが必要とされるであろう。

Ｖ５：ADAMの実行の際には、データ保存に関する情報を処理し、次に述べる事柄に関連するサポートを強化しなければならない。

・並列計算機及び高性能コンピューターのためのフォーマットと適切な分散処理
・エラーの許容と伝播（特に関連問題について）
・テストされていないコードやデータを組み合わせたものへのアクセスを管理すること

4　地下水モデル開発におけるソフトウェア品質管理手法事例

本項では、ソフトウェアの品質保証に関する背景となる情報を提供し、地層処分性能評価のための地下水モデル開発における品質保証と品質管理の役割を検討した。作業にあたって先行していると判断される米国環境保護局の研究報告を参考とした。そこでは、機能的で実用的な品質保証の方法論が紹介され、品質保証にかかわる判断の論拠となる技術・手法を必要とするモデル開発ユーザー等の立場から検討した。ソフトウェア品質保証の重要な部分は、コード試験と性能評価である。参考として、米国・国際地下水モデル開発センター（the International Ground Water Modeling Center）によって開発された3段階のコード試験手順を参照した。

4．1　地下水モデル開発とソフトウェア品質管理

地下水管理や廃棄物処分等の環境評価作業において、地下水モデルの開発は重要な手法である。モデル開発問題において、コンピュータ・シミュレーション・コード（以下、地下水シミュレーション・コードと記す）を開発することは効果的手段として続けられてきた。それは、定性的な一つの科学的手段である。成功したモデルを扱うコンピュータ・シミュレーション・コードは、科学原理、数学的方式、そしてモデル開発過程における、数理解析とプログラミング技術にかかわる専門的洞察力、そして対象問題としてのサイト特性記述の知識が統合化されたものである。従って、それは多方面の専門家、研究者の協調から生まれてくる。ヨーロッパ水協会（the European Institute for Water）によって組織された「水管理のモデル開発」研究会の参加者（Como, Italy, May 21-22, 1987）が、このアプローチについて下記の通りに説明している。

> **「地下水モデル開発は、関係者全員に目標、規範、束縛、適切な過程、パラメータ等を明確にする規則を強いるものである。」**

地下水モデル開発は、地下水システムのメカニズムと管理、そしてそれらの品質に影響を与える過程と要因、特に地下水システムにおける人工介入物（汚染物質等、社会地球化学的影響）が原因による特性変化等、地下水を対象とした現象理解を得るための解析手法を組み立てることである。水資源の管理者には、地下水モデルは地下水の水質特性にかかわる管理作業の基本的方針を立案するための有効な判断論拠を与えることができる。これは特に、地下水資源開発、地下水保護、そして帯水層復元等、現実の地下水管理プロジェクトで利用されている。

地下水管理の成功は、技術的・科学的に確かなデータ収集、情報処理、そして評価方法の使用に基づいて決定が行われ、これらの方法が整然と統合されていることを必要とする。コンピュータ・シミュレーション・コードは基本的なモデル管理の手段であるため、それらのコードが計画や意思決定の道具として用いられる前に、コンピュータ・シミュレーション・コード特性の体系的試験や評価を通して、開発者から独立し、信頼性を確立（検証）することが極めて重大である。

地下水管理のためのコンピュータ・シミュレーション・コードは、その開発初期段階においては非公式な開発環境で開発利用されたが、現在ではそれによる判断があまりにも社会的に重要になりすぎてしまった。さらに、サイト特定問題における地下水モデルのアプリケーションは、体系的で包括的な手順や詳細な報告が、結果の持つ社会的異議から求められるようになってきた。そのような背景か

ら、現在地下水モデル開発について論じる際には、モデル開発とコンピュータ・シミュレーション・コードの間の区別がなされなければならない。モデル開発は、以下の3つの構造要素から成り立っていると考えられる。

- 研究対象である地下水システムの定量的理解を得るための研究作業
- ソフトウェア開発と管理作業
- モデル開発と評価手法の開発作業

モデル開発、特にコンピュータ・シミュレーション・コード開発は、地下水資源管理業務において、即時に、かつ長期的予測問題として実行される。コンピュータ・シミュレーション・コードの開発は、サイト、または特定の問題を解決することを目標とした、より大きい作業の一部であり、データ収集、評価と蓄積、システム概念モデルデザイン、代替の問題解決シナリオ、設計、数値解析などの作業を含むものである。地下水モデル開発は、何を必要とするかという多数の意見が存在するにもかかわらず、ソフトウェア工学による「モデル」の定義は曖昧である。水文地質学における「地下水モデル」という用語は、概念的地下水モデル、数学的地下水モデル（解析モデルと数値モデルを含む）と同意語になった。

地下水モデルは、簡易化された数学的説明であり、境界条件、パラメータの形でコンピュータプログラミング言語にコーディングされると考えられる。様々なサイト、または特定問題のシミュレーションに使用可能な汎用コンピュータ・シミュレーション・コードは、（コンピュータ）シミュレーションコード、または総称シミュレーションモデル等と呼ばれる。地下水モデル研究は、ある特定の地下水管理問題を解決するための地下水モデルの開発と使用（すなわち、コンピュータ・シミュレーション・コードとデータ）として定義付けられる。時には、そのような地下水モデルは、一般化された地下水管理問題への一つ、またはそれ以上のコンピュータ・シミュレーション・コードの適用であると考えられる。また時には、モデルは用いられた数学的解析技術の見地から述べられる。最も一般的に使用される用語は、「解析モデル」、「半解析モデル」、そして「数値モデル」である（van der Heijde et al., 1988）。解析モデルは、問題の解（と境界条件）が解析式として与えられる数学モデルである。数値モデルは、一般に、与えられた基礎式から解析式としての結果を導くことが出来ない問題（解析解が導けない式）に関し、時間と空間に離散近似を行い、適当な条件の基に数値計算により解くモデルである。これは、有限差分法、有限要素法等の離散方程式により与えられる。半解析モデルはその両者を混同させた手法である。

効果的で信頼できるソフトウェアを開発し、そのような道具を地下水管理に適用するには多数の段階があり、各段階は念入りに行い、注意深くレビューされなければならない。作業は、モデル開発とコンピュータ・シミュレーション・コード開発過程の全段階において、体系的に管理・統制されたアプローチが必須である。そのためには、同作業に関する適当な品質保証手法の開発と適用が重要である。

本章では、地下水モデルと関連コンピュータ・シミュレーション・コードの品質保証に関する背景となる考え方等を検討し、地下水モデル開発における品質管理の役割を定義付けている。

（1）地下水モデル開発における品質保証

地下水モデル開発における品質保証とは、全てのモデル開発に基づいた解析が検証可能で正当と認められ、研究に含まれた全てのプロジェクト作業の適当な実行を技術的・科学的に証明するために、モデル開発研究を管理する協会が手順と操作において構成したものである。地下水モデル開発における品質管理は、モデル開発とモデル使用の両方において極めて重大であり、モデル開発過程の全面に適用されるプロジェクト計画の必須の部分である。品質保証の主要な2つの要素は品質管理（QC）と品質評価である。品質管理とは最終生産品の品質を保証する手順についてである。これらの手順は、コンピュータ・シミュレーション・コードの開発と利用のための適切な方法論の使用、十分な検証や妥当性検査の手順、そして選択された方法やコードの正しい使用を含むものである。品質管理手順を監視し、研究の品質を評価するために品質評価が適用される。各プロジェクトには、品質保証計画（品質管理計画）がなければならず、そのプロジェクトの品質目標を達成するための基準を挙げなければならない。

「品質保証」は様々な分野や環境で使用されている用語である。その意味や実現は分野によって異なってくる。例えば、ソフトウェア工学における品質管理、ソフトウェア品質保証（S品質管理）、そして工業生産品における品質管理の間では大きな違いがある。また、データ品質管理とソフトウェア品質管理手順の間にも大きな違いがある。多くのモデル開発研究は、適切な品質管理の準備が欠如したまま実行されている。正式な品質管理計画、すなわちプロジェクト計画全体のある部分が欠如していることがある。品質管理アセスメントは、プロジェクトが最終段階に入るまでたびたび延期されている。これはサイト特定問題を解くために、あるモデルが適用された研究においては特に重要なことである。しかしながら、大きな支持者層に影響を及ぼす政策決定がモデル開発評価に基づく場合、使用されたモデルと、そのコンピュータ・シミュレーション・コードの両方は、研究結果が受け入れられる前に批判的評論を受けさせられることがある。従って、現在の地下水汚染問題における財政上・刑事上の責任等に影響の及ぶモデル開発作業においては、モデル開発者がモデル開発プロジェクトの全段階に厳密な品質保証手順を履行する事が基本要求とされている。

4．2　モデル開発過程における検討事項

4．2．1　モデル開発過程

モデル開発は、観察、仮説作り、そして推論された関係の検証を通して、自然界の新しい定量的知識を習得する科学的過程と密接に関連しており、観察された現象の確かな理論的構造の立証という結果になる。従って、地下水システムの基本的理解とは、理論によって統合された研究の産物である。

そのような研究の目的は、現実世界の多要素地下水システムにおける選択された要素を含む原型自然システムである。研究のための特定原型システムの選択は、管理側の必要性や研究者の個人的興味によって動かされ、現実世界の通り一遍の標本抽出に基づいた統合されたシステムの初期概念化に影響されている（図4．2－1）。

選択された地下水システムの概念モデルは、システムとその環境の様々な構成要素の間の不定の関係を把握するための基礎をつくる。これらの関係は数学的に定義され、数学的モデルになる。数学方程式の解法が複雑であったり、繰り返しの多い計算が必要な場合はコンピュータの使用が必須である。これは数学問題の解のコーディングをプログラミング言語で行うことを要求し、コンピュータコード

になる。概念公式化、数学的解説、そしてコンピュータコーディングが原型モデルを組成する（図４．２－１）。

モデルドキュメンテーションの準備は、数学的モデルの公式化の段階で行われ、コードデザインやプログラムコーディングの段階で続けられ、他の全てのモデル開発段階が完了した後に終了しなければならない。

地下水モデルが、計画や意思決定の道具として使用される前に、モデルの正確さの体系的テスティングやモデルの性能特性の評価によってその信用証明が立証されなければならない。利用できる２つの主要なアプローチのうち、評価またはレビューの過程は現実に定性的であり、コードテスティング結果は定量的性能の測定によって表わすことができる。性能特性は、信頼性（例えば、解アルゴリズムの収束や安定性、そして端末不足が無いこと）、コーディングされたアルゴリズムの効率（達成された数値の正確さ対メモリー要求とコード実行時間の見地から）、そしてモデルのステップアップのために必要な資源（例えば、入力準備時間）によって表わすことができる。

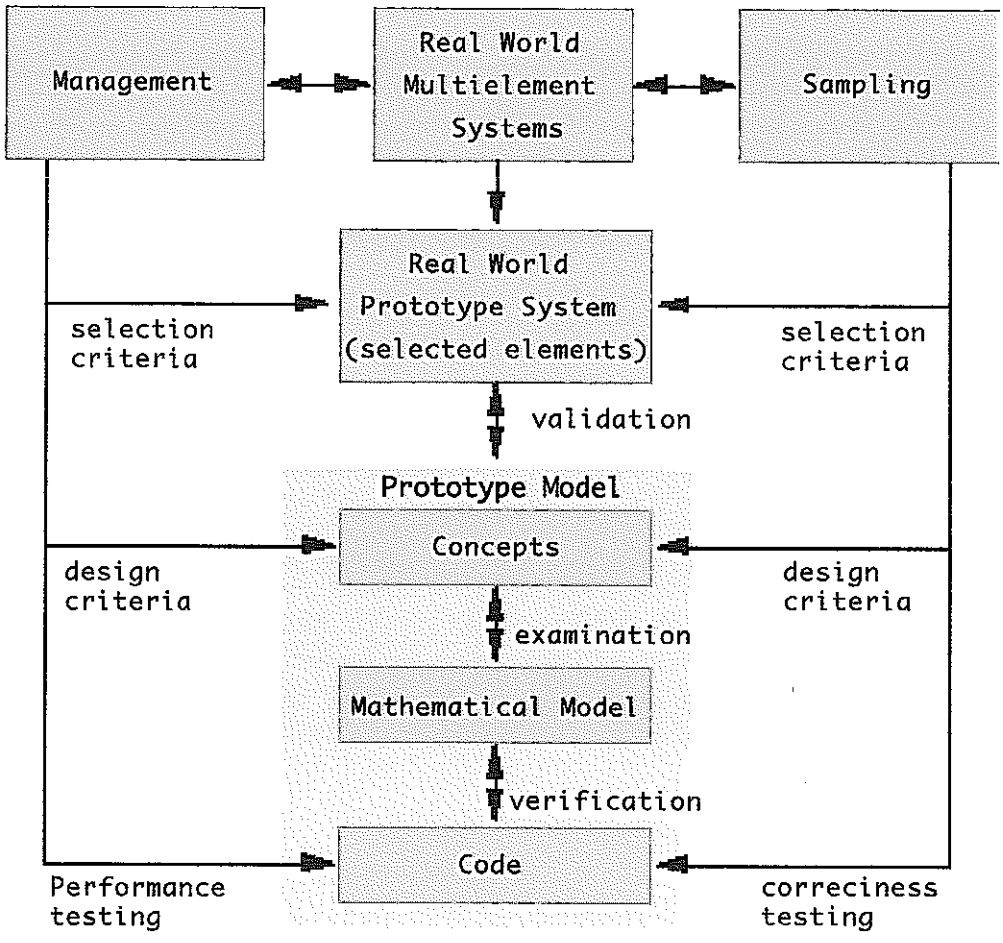

図４．２－１　モデル開発の概念（van der Heijde et al., 1988）

コードがシミュレートするためにデザインされた全範囲のパラメータやストレスの性能特性が決定される必要性がある。地下水資源管理者は、そのようなコードが本来のデザイン規準を超えて、また

は既にテストされた適用範囲を超えて使用された場合の結果を決定するために更に進んでコードのテストをすることに興味を持つかもしれない。包括的で体系的なコードテスティングやモデル評価を通して、コードの適用性への確信が増大する。

このように、コードテスティング（またはコード検証）は、注意深く選択された実例テスト問題やテストデータセットの実行を通して、プログラミングエラーの発見、組み込まれたアルゴリズムのテスティング、そしてコードのオペレーション特性を評価することを目標とし、コードテスティングとモデルテスティングの違いを区別することが重要である。コードテスティングは、コンピュータコードにデザインされた規準や、要求に関連したその正確さを立証することに限定されている（例えば、数学的モデルを表わすため）。

モデルテスティングまたは（原型）モデル妥当性検査は、そのモデルがシミュレートするためにデザインされた現実世界原型システムによって得られた定量的関係の妥当性を把握するための最終段階を表わすため、コードテスティングよりもより包括的（そして、より回避的）である（図４．２－１）。

４．２．２　ソフトウェア開発とメンテナンス

ソフトウェア品質保証は、ソフトウェアのライフサイクル全体に亙る手順、技術、そしてのアプリケーションから成り、それは生産物が要求に適合することを確実にするためである（Bryant and Wilburn, 1987）。これはソフトウェア開発プロジェクトの初期段階において、適当な手順（例えば、品質管理計画の開発、記録、プロジェクトの品質管理組織の確立）、技術（例えば、監査、デザイン検査、コード検査、エラー傾向の解析、機能テスティング、理論テスティング、方向テスティング、レビュー、リハーサル）、そして道具（例えば、テキストエディタ、ソフトウェアデバッガ、ソースコードコンパレータ、言語処理プログラム）が確認されなければならず、ソフトウェア設計規準が決定されなければならない。

現在、多くの地下水モデル開発コードがそのような厳密な品質管理アプローチを受けていない。理想的には、研究または資源管理のいずれにおいても、S品質管理は現在使用中のコードと、これから開発されるコードの全てに適用されるべきである。

ソフトウェア・ライフサイクル概念の使用は、ソフトウェアの開発、使用、そしてオペレーションの品質管理要求を決定することに成功したことを証明した（Bryant and Wilburn, 1987）。このソフトウェア開発ライフサイクルは、主に３つの段階から成り立っている。それらは、開始の段階、開発の段階、そしてオペレーションの段階である（NBS, 1976）。この概念によると、開始段階はソフトウェアのための目的や要求を定義し、事業化前調査や費用便益分析を行うためにある。開発段階では、ソフトウェアとドキュメンテーション要求が決定され、プログラムデザインが公式化され、コーディングは履行されテストされる。

実行段階においては、ソフトウェアはそれがデザインされた目的のために使用される。この段階においては、ソフトウェアは維持され、定期的に評価され、追加の要求の確認に応じて変更される。メンテナンスの調整が必要とされなくなった場合（例えば、使用されているコンピュータ環境の変更、オペレーションの要求の変更、またはソフトウェアデザイン目的の変更のため）、1970年代に開発された地下水シミュレーションコードの多くに適用できる状態、ソフトウェアライフサイクルの終わりに到達する。これらの各ソフトウェア・ライフサイクル段階の品質管理要求における詳細な論議がBr

yantとWilburnによって提示されている（1987）。コード開発とメンテナンスにおける最も重要な品質管理手順は（van der Heijde, 1987）以下の通りである。

- コードデザインとコード開発のドキュメンテーション
- プログラムの構造やコーディングの検証
- 完全なソフトウェア製品の妥当性検査（モデル妥当性検査。図４．２－１参照）
- コード特性、可能性、そして使用についてのドキュメンテーション
- 科学的で技術的なレビュー
- 管理上の監査

地下水モデル開発コードを体系的で一致した方法で評価するため、国際地下水モデル開発センター（the International Ground Water Modeling Center：IGWMC）は、モデルレビュー、検証、及び妥当性検査の手順を明瞭に述べた（van der Heijde et al., 1985）。この手順の一部として、IGWMCは3段階のコードテスティング戦略を開発した。

シミュレーションコードに何らかの修正が行われた場合、例えばある特定のアプリケーションのためにコードを適合するため、そのコードは再度テストされなければならない。また記録、テスティングと報告を含むコード開発のための全ての品質管理手順が適用されなければならない。修正されたコードは、適用可能なところで、前バージョンでテストされたテストデータでテスト実行されなければならず、その前バージョンのテスト結果と比較されなければならない。全ての新しい入出力ファイルは、検査や起こりうる再利用のために保存されなければならない。

４．２．３　コードデザインとコード開発のドキュメンテーション

コード開発やメンテナンスの品質管理は、コードデザイン目的の公式化に始まり、規準や標準、実現戦略に続く体系的なアプローチを意味する。実現戦略は、コード構造のデザインやコードに適用されるソフトウェア工学原理の方法の説明を含む。この計画段階においては、コードデザインと実現、コーディング手順の記録、各コード区分の目的や構造の記述（機能、サブルーチン）、及びコード検証過程の記録の完全なドキュメンテーションを確実にするための測定を行う。コード開発手順の必須の部分は、ユーザー向けドキュメンテーションの準備である。モデル開発における品質管理のペーパートレーラは、モデル開発における報告やファイルから成り、以下の事項を含まなければならない。

- （標準化され承認された）プログラマーのノートを含むコードの開発に関する報告。
- 使用された検証（または妥当性検査）シナリオ、パラメータ値、境界条件と初期条件、及びそれらのソース、検証結果、そして適応可能であればベンチマーク（データ、コーディング、ソース）を含む検証報告。
- 使用された検証グリッドの特質とグリッドデザインの調整。
- ベースライニング後のコードにおける変更との変更の検証。
- 妥当性検査の場合は、もし適用されたならば、較正手順の詳細。

それに加えて、要求された品質管理のレベルによっては、あらゆるファイルが保持され（ハードコピーと、より高いレベルの場合にはデジタル形式）、以下を含まなければならない。

・コードのベースラインバージョンの実行できるイメージとソースコード
・各検証実行の入出力
・妥当性検査の入出力

4．2．4　プログラム構造とコードの検証

ソフトウェア工学における検証とは、ソフトウェアの一貫性、完全性、そして正確さを表わす手順である（Adrion et al., 1986）。ATSM（1984）は、検証を「コンピュータコードにおける数値技術の調査であり、それが本当に概念モデルを表わし、解を得るために先天的問題が無いことを確定するためのものである」と定義している。地下水モデル開発においては、コード検証には、以下の２つの観点がある

・方程式を解くために使用された計算アルゴリズムの正確さや確実さを確かめるため
・コンピュータコードが完全に操作可能だということを確実にするため

理論的原理の正しいコーディング、コード理論や主なプログラミングのエラー（「バグ」）のコードを調査するためには、コードは特別にデザインされた問題を使用して実行されている。このコードテスティングへのアプローチは、信頼性の評価（例えば、解アルゴリズムの安定性、端末不足の無いこと）、コーディングされたアルゴリズムの効率と内外部のデータ転送（例えば、数値の正確さ対計算時間によるコード性能、メモリ使用と貯蔵要求）、そしてコード実行準備資源の需要（例えば、入力準備時間）を考慮する。首尾よく構造されたコードは、全体的なプログラム構造に加えて、独立したコード区分、サブルーチン、機能やモジュールの効率的なテスティングを考慮する。この段階のコードテスティングは、グリッドデザイン、様々な主要手順、そしてパラメータ値の広い選択に対するコードの感度を評価するためにも使用される（Huyakorn et al., 1984a; Sykes et al., 1983; Ward et al., 1984; Gupta et al., 1984）。

この時点で、左右する方程式の解析解法に基づいたシミュレーションモデルと数値解法を含むモデルとの間の区別をしておくことが必要である。コーディングされた解析解法の検証は、同じ数学的表現を用いて独立して計算された結果の比較、例えば手動計算、第三者のプログラマーの団体によって独立してコーディングされたコンピュータプログラムの結果からの比較、またはMathematicaR（Wolfram, 1991）やMathcad3.0。R（Mathsoft, 1991）などの、一般的な数値計算用コンピュータソフトウェアシステムの使用に限定されている。数値モデルを表わすコードの検証には以下の形が考えられる。

・解析解法との比較
・同じ総称シミュレーションモデルを表わすコード間のコード相互比較。
・統合的データのセットを使用する。

数値モデルのために頻繁に使用される検証アプローチでは、テスト問題は空間と時間の領域の各点においてストレスやパラメータが決定されるシステムを表わし、そのようなシステムの反応は左右する微分方程式に対する閉じた解によって表わされる（解析解法、IGWMCテストレベル１）。テスト

される数値モデルは、空間や時間における離散した点の限定された数で同じ方程式に解を与える。コーディングが正しいと仮定して、解析解法と数値モデルによって表わされたシステム反応の違いは、主に関連した数値方法の概算的性質とコンピュータの正確さの限界のためであり、一般的にはランダムに散布されていない。

多くの場合、これらの違いの絶対値は、計算方式に使用された離散における解像と関係がある（Lapidus and Pinder, 1982）。理論的には、時空の段階の大きさがゼロに接近するほどに解像が増大した場合、数値解法と閉じた解法の差は無くなるはずである。

このような種類のテスティングにおいて、解析解法のコンピュータコード実現が使用された場合、結果コードは数値モデルと同じ厳密な品質管理が必要であるということを注目しなければならない。解析結果と簡易化した状態の結果を比較することによって、数値モデルをテストしても完全にデバッグされたコードは保証されないにもかかわらず、首尾よく選択された問題のセットは、コードのメインプログラムと、そのほとんどのサブルーチンがテストで使用されているということを確実にし、使用されるプログラムや、あるパラメータフィールドにおいては、コードの計算性能に対して批判的になるかもしれないものを含む。検証練習の効率は、検証問題のデータを使用したコンピュータプログラムオペレーションの着実な解析であるリハーサルによって高めることができる。

コード検証手順の一部として、簡単な解析モデルには表わされない特別な計算特性をテストするために、仮説的な簡易化された問題を使用することができる（IGWMCテストレベル2）。明確にすると、このようなタイプのテスティングは、変則的境界、時間変動ストレス（ソース、シンク）、帯水層特性の不均質や異方性、そしてグリッドオリエンテーションと幾何学を扱ったコードの性能を評価することを目標とすることができる。このような仮説的問題は、単一フィールドのサイトの特性、または様々なサイトの特性の総称的混合に基づくかもしれない。結果として生じるデータセットは統合的テストデータセットと呼ばれる。

このようなテストに使用される統合的または仮説的システムは、統合的システムパラメータやシステムストレスによって定義され、単独的に観察されたシステム反応は無い。従って、テスティングは数値一貫性や安定性に関するコード反応の評価、または他と得られたものとテストされた数値モデルとともに得られた離散的予測の比較によってテストは行われ、それは高い解像の時空離散方式を使用した立証された数値モデルが好ましい。これらの仮説的問題の絶対的「事実」は不明のため、あるモデルの比較の検証だけが得られる。このアプローチを使用することによって、相対的ベンチマークを得られる。このようなタイプのコード検証は時々「ベンチマーキング」と呼ばれる。

シミュレーションが、相互比較が互いに注目に値するほど迂回しないコードの結果に終わった場合、「相対的」または「比較的」妥当性が立証される（van der Heijde et al., 1988）。しかしながら、違いが起こった場合、両方のコードによって行われたシミュレーション実行の結果の詳細な解析が必要となる。

国際地下水モデル開発センター（the International Ground Water Modeling Center）は、コーディングされた解析解法と短期的流動の2次3次元の体系的ベンチマークと、飽和されたものとされていないものの浸透性メディアにおける溶解運搬を収集している。ベンチマーク予測を得るために、立証された様々な2次3次元モデルが使用されている。ベンチマーク問題のデザインは、線形と非線形の地下水流動と運搬方程式を解くために使用される様々な数値方法の知られた特性に到達している。

ほとんどの地下水シミュレーションコードは、そのコーディングのある区分、またはシミュレートできるようにデザインされた全条件のサブセットによってのみ検証されてきた。以前は、上記に概説された原理に沿った完全で体系的な検証を受けたコードは極少であった。

コード検証の段階は、様々な主要手順、そしてパラメータ値の広い選択に至るまで、例えばグリッドデザインへのコードの感度など、コードのオペレーション特性を評価するため使用できる（できなければならない）ことが注目されなければならない。最後に、「モデル検証」という用語は、時々サイト特定設計モデルの更正と関連して使用される（Anderson and Woessner, 1992）。

4．2．5　モデル妥当性検査

モデル妥当性検査とコード妥当性検査や、コード検証を等式化することによって多くの混乱が招かれた。「妥当性検査」という用語は、それが使用されているフィールドによって定義される。ソフトウェア工学においては、全米標準局（the National Bureau of Standards）によって採用されているように、コード妥当性検査は**「ユーザーの必要性や要求に関する最終的ソフトウェア商品の正確さの決定」**と定義されている（Adrion et al., 1986）。化学製品の環境の結果モデルの評価をするための標準実行について論じるため、米国テスティングと物質協会（the American Society for Testing and Materials）は、モデル妥当性検査を「実験室の実験または環境の観察から独立して得られたモデル結果と数値データの比較」と定義付けている（ASTM, 1984）。

Van der Heijde（1987）は、地下水モデル妥当性検査の目的を、モデルが開発されたプロトタイプ世界の地下水システムの計算されたものと、独立的に観察された原因結果反応の間の「相関の度合い」に関して、モデルの理論的設立とコンピュータ実現が実際のシステム反応をどの程度表わすか決定することと定義している（図4．2－1）。この定義は、サイト特定設計モデルは妥当性検査を達成するために開発されてはいるが、サイト特定設計地下水モデルには当てはまらず、総称的シミュレーションモデルに当てはまる。

「モデル妥当性検査」という用語は、サイト特定設計モデル開発にも使用される。AndersonとWoessner（1992）は、サイト特定設計モデルデザインとアプリケーションプロトコルが「与えられたサイト特定設計モデルが意味深長な結果をもたらすことが可能だということ、すなわちモデルが妥当であるということの裏付けをする」と議論する。彼等の見地からすると、モデル妥当性検査の過程は全モデル開発プロトコルを包含する。さらに、サイト特定設計モデルがその予測において正確であるとポスト監査に証明された場合、そのモデルはそのサイトにおいては「妥当化された」とみなされる（Anderson and Woessner, 1992）。彼等は、「各サイトが独特であるため、モデルは理想的には各サイト特定設計のアプリケーションごとに妥当化されなければならない」と述べている。

相関の度合を量化または定性的に表わすには様々な方法が存在する（Koch and Link, 1980）。モデル入力、システムパラメータとシステム反応の実際に測定されたデータは、本物のシステムと先天的に混合されたエラーのサンプルであるということを注意しなければならない。従って、このように立証されたモデル妥当性検査は常に主観的である（NRC, 1990）。

さらなる複雑性は、フィールド妥当性検査のために使用されたデータが、そのフィールドから直接収集されず、それ以前の研究でしばしば処理されるということにある。その結果、それらは不正確で、情報に欠け、解説上の先入観、精密さの欠如、そして転送や処理のエラーに陥りやすく、妥当性検査

過程において使用されるデータの基本的な劣化の結果に終わる。

妥当性検査過程における較正の役割に関していくつかの議論が行われてきた。物理的パラメータの慎重な調整後にフィールドデータを当てはめるために、較正がコード（と総称モデル）の力量をテストする限りにおいては、著者の何人かはそれを妥当性検査のより弱い形とみなす（Ward et al., 1984）。この見地に対する弁明は、利用できるシステムパラメータや観察された従属変数の不確実さによって本当の妥当性検査は可能ではないということである。

妥当性検査の厳密な形は、較正を除いたモデル予測に基づいている。システムストレスやシステムパラメータは、現存するストレスへのシステム反応とともに分かっているとみなされる。このアプローチは、パラメータの独立したモデル開発推定を使用して実験データに当てはめるモデル力量のテストである。原則的には、これが妥当性検査への正しいアプローチである。しかしながら、フィールドデータの無効と不正確さは、そのような硬化した妥当性検査アプローチを実際のフィールドシステムに適用することを防止することがある。典型的には、フィールドデータの一部分が較正データとして表示され、パラメータ値の適度な調整によって較正されたサイトモデルが得られる。フィールドデータのもうひとつの部分は妥当性検査データとして表示される。

比較のための類似したデータをシミュレートするために、較正されたサイトモデルは予測モードで使用される。従って、このようなテストの品質は、その基づかれている較正データを「ストレス上」のサイトモデルの程度によって決定される（Ward et al., 1984）。ユーザーの見地からすると、総称的地下水モデルの妥当性を決定することは次のような問題への解答と関連する.

- **概念モデルはそれが表わす原型システムに対して妥当か？**
- **数学的モデルは概念モデル、関連している過程、そして様々なデザイン条件におけるストレスを本当に表わしているか？**
- **コードは本当にモデルの数学的構造を表わしているか？**
- **モデルは、ユーザーが調査しようとするサイト特定設計のストレスシナリオの原型システム反応を本当に表すことができるか？**

従って、完全なモデル妥当性検査は、そのモデルがデザインされた全範囲の条件におけるテスティングを必要とする。総称モデルの開発は、新しい研究結果、技術開発、そしてユーザー要求の変化に反応する進化的過程である。モデルレビューと妥当性検査は、このダイナミックな過程に従う必要があり、モデルが修正される度に適用されなければならない。

多くの地下水モデルのタイプにおいて、それらの妥当性検査のために要求される広範囲な高品質のデータのセットが欠如している。さらに、利用できるデータセットは、システムのサンプルにおいて起こる様々な条件やストレスに関連して限定されることがある。

従って、モデルのテスティングは一般的に、現存する解析解決や体系的ベンチマークを使用した妥当性検査、コードの相互比較、そしてサイト特定設計モデルアプリケーションの詳細なポスト監査に限定されている（NRC 1990）。

妥当性検査過程における第一段階は、妥当性検査シナリオの定義付けである。フィールド妥当性検査の計画と実行は次の段階を踏まなければならない（Hern et al., 1985）。

・妥当性検査のデータ必要性を定義し、利用できるデータのセットを選択、または研究するサイトを計画する
・データ品質を正確さ（測定エラー）、精密さ、そして完全さの点から評定する
・モデル性能または受け入れ規準を定義する
・感度解析の戦略を開発
・妥当性検査実行を行ない、モデル性能と立証された受け入れ規準を比較する
・妥当性検査実例を詳細なドキュメンテーションにする

様々なタイプの地下水モデルのフィールド妥当性検査のため、データベースの更なる開発が必要である。これらの研究データベースは、広範囲の水文地質学的状況や地下水保護問題を説明し、そのフィールドに現存する様々なタイプの流動、運搬、そして変形メカニズムを反映しなければならない。データベースは、水文地質学、土壌、地球化学、そして気候学の特性に関する広範囲に亙る情報を含まなければならない。そのようなデータベースの開発や標準モデルテスティングと妥当性検査過程の採用によって、フィールドアプリケーションに使用されるモデルはかなり改善される。

モデルの絶対的な妥当性は存在しないということを注意しなければならない。ある特定のアプリケーションにとってモデルが妥当であるかどうかは、事前に決定された性能規準を適用することによって評定され、これは妥当性検査または受け入れ規準と呼ばれることがある。計画や意思決定の様々な使用が予測できるならば、様々な性能規準が定義付けできるかもしれない。そしてユーザーは、モデルが使用されようとしている目的の妥当性を注意深く確認しなければならない。

４．２．６　検証と妥当性検査規準

コード検証と妥当性検査の重要な見解は、評価または性能規準として使用される有益で有効な測定の定義付けである。そのような測定は、モデルに適切な全空間と時間の領域の中の数値モデルと、モデルが開発されたシステムに直面するかも知れないパラメータの全範囲から得られた解の正確性と安定性を特性付けなければならない（van der Heijde, 1990）。

これまで、検証や妥当性検査の受け入れは、主にテストされたコードや使用されたベンチマークに対する予測変数のグラフ説明の視覚検査に基づくものであった（図４．２－２）。そのような検査や、それについて得られるどのような定量的測定のセットでも、絶対的限界点と予測変数の値の関連の両方において、最大偏差、平均偏差そして偏差の散布を含む全体的な適合度を必要とする。これらの測定は、時間と空間の領域、そして全領域と領域の代表的部分の両方に適用されなければならない。測定は、様々な時空の尺度や解像に対するモデル反応の評価を促進し、関係している変数の偏差の傾向や振動を示さなければならない。

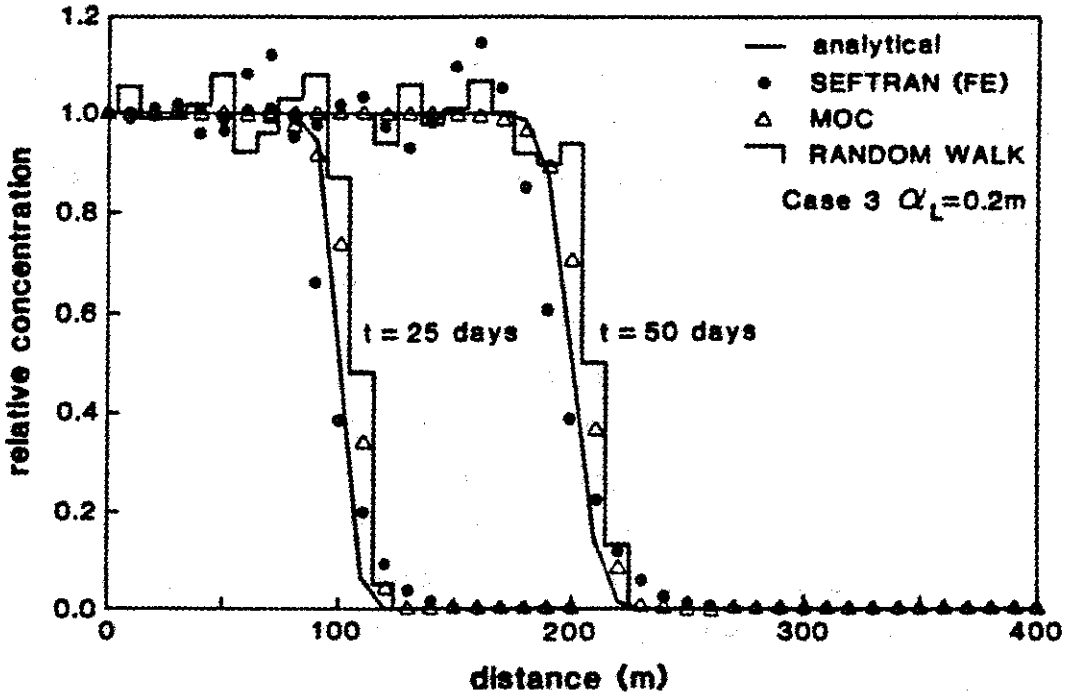

図４．２－２　解析解法と様々な数値解法における比較の例（Beljinより 1988）

ほとんどのグラフの最良の測定は１次元グラフ、例えばヘッド対時間またはヘッド対距離を使用する。これまで、複数次元のグラフは主に妥当性検査、例えば輪郭化された偏差の形で使用されてきた。

これら複数次元グラフの技術を検証過程に拡張することによって、適合度を判断する能力がかなり高められるかもしれない（図４．２－４）。DonigianとRao（1986）は、コードの性能の定量的測定を与える３つの主な手順を挙げた

- ひと組になったデータ性能－時間や空間の正確な位置におけるシミュレートされたデータと観察されたデータの比較
- 時間と空間の統合された、ひと組のデータ性能－時空の平均されたシミュレートされたデータと観察されたデータの比較
- 振動数領域性能－シミュレートされたものと観察された振動数配列の比較

彼等は、レビュー中の地下水コード性能の定量的測定を生成するため、２つの統計モデルを選択した。

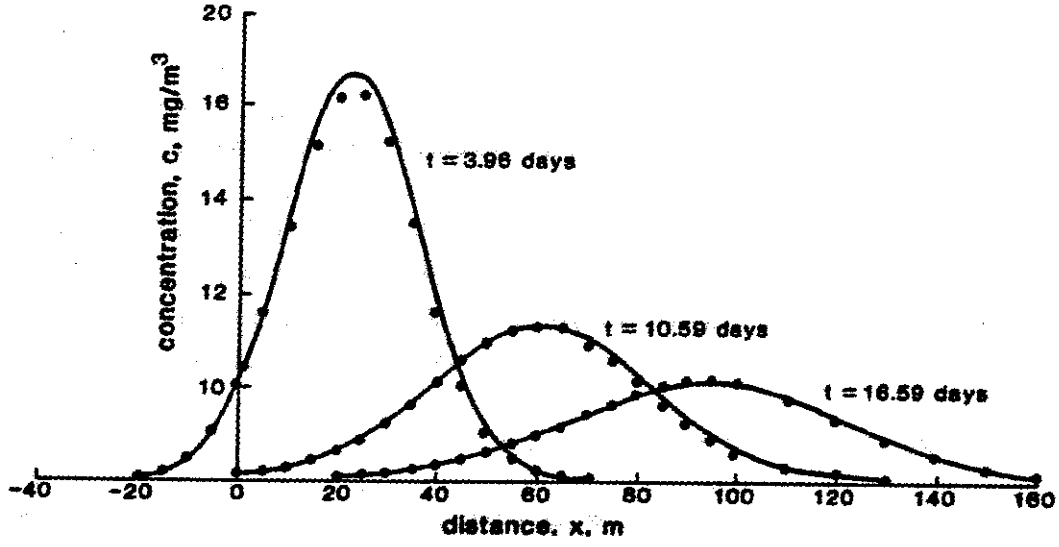

図４．２－３　空間領域における、解析解法と数値解法の接近した一致の例
(from Huyakorn et al., 1984a)

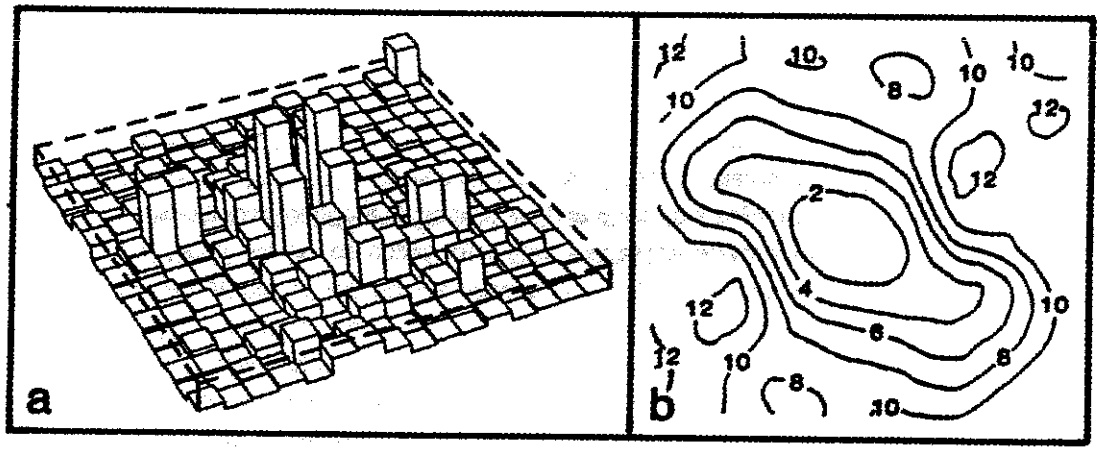

図４．２－４　２次元検証偏差のグラフ説明。a) ブロック図表、b) 輪郭の偏差。

IGWMCによって採用された３つのテスティングの各段階は、これらの測定の定義付けに対して別々のアプローチを必要としている。テスティングの最初の２段階においては、検証の段階と統計測定は不適当であるが、統計に類似した測定は有益と証明されるかもしれない。例えば、ATSM（1984）は偏差係数DCの説明を表わしている。

$$DC = [(x - y) / x]\ 100\%$$

$$DC = \sum_{x-1}^{n} [(x_1 - y_1) / x_1]\ 100\%$$

x= 予測された値、y= 測定された値、n= 計算された偏差の合計数。

式４．２－１

モデルテスティングの第３段階であるフィールド妥当性検査は、統計分析される標準データ適合手順に類似しており、Donigian and Rao（1986）によって使用された統計測定によって説明のため公開されている。

４．２．７　ソフトウェアドキュメンテーション

コンピュータコードのドキュメンテーションとは、コンピュータアプリケーションのデザイン、開発、そしてメンテナンス中に記録された情報であり、目的、方法、理論、関係、能力、そして限界を含むデータ処理システムの適切な面を説明するためのものであると定義付けられる（Gass, 1979）。モデル開発の労作に関係している者、例えばコード開発者、コードメンテナンススタッフ、コンピュータシステムオペレーターやコードユーザーにとって、それはソフトウェアの全ての面において伝達するための主要な道具である。ソフトウェアドキュメンテーションの主な目的はGass（1979）によって次のように述べられている。

- システムやプログラム変更をより早く効率的にすることを可能にする技術情報を記録するため
- ソフトウェアオリジネータ以外にもプログラマーやシステムアナリストがプログラムを使用したり研究することを可能にするため
- 何のプログラムで何ができるかということをユーザーが理解できるよう手助けするため
- プログラムの共用ポテンシャルを増加させるため
- プログラムオペレーションの監査や検証を促進するため、すなわち、コード評価
- マネージャーたちに重大な開発についてレビューするための情報を与え、プロジェクトの要件は揃い、資源を使用し続けるべきだということを決定させるため
- 人事異動による分裂の影響を減らすため
- ソフトウェアのオペレーションにおけるメンテナンス、トレーニング、そして変更についての情報をマネージャー、開発者、プログラマー、オペレーター、そしてユーザーの間に与えることによって理解を促進させるため。
- 他のポテンシャルユーザーにとってソフトウェアが彼等の要求を満たすかどうか決定できるよう、その機能や可能性を知らせるため。

包括的なソフトウェアドキュメンテーションの準備のための詳細なガイドラインが、米国連邦コンピュータ性能評価とシミュレーションセンター（the Federal Computer Performance Evaluation and Simulation Center）（FEDSIM, 1981）によって述べられている。

この出版物は、マネージャー、ユーザー、アナリスト、そしてプログラマーのモデル情報を提供す

る４つのタイプのマニュアルのために推奨された構造を論じている。FEDSIM（1981）によると、マネージャーの概略マニュアルは、モデル説明、モデル開発履歴、実験報告、そして現在と未来のアプリケーションについての議論を含まなければならない。現在ASTM（米国テスティングと物質協会：American Society for Testing and Materials）は、この特定の目標のために標準地下水コード記述を開発している（Ritchey, 1992）。ユーザーマニュアルは、広範囲なモデル記述、モデル入力データ記述とフォーマット、得られた出力データのタイプ、コード実行準備指示、サンプルモデルの実行、そして障害探索ガイドを含まなければならない。プログラマーマニュアルは、モデル指定、モデル記述、フローチャート、ルーチン記述、データベース記述、ソースのリスト、そしてエラーメッセージを含まなければならない。コードそのものが首尾よく構造され、内部的に良くドキュメントされていなければならない。パラメータ、変数、サブルーチン、そして機能の名前が使用されなければならない。

典型的には、地下水モデル開発コードのドキュメンテーションは、コードの基づかれている総称モデルによって表わされる理論的構造、適用されたコード構造と標準言語、そしてモデルセットアップやコード実行パラメータに関するコード使用の指示の説明を含まなければならない。

良質なドキュメンテーションは、総称モデルが基づかれている方程式、根本的な仮定、モデルに混合された境界条件、方程式を解くために使用された方法やアルゴリズム、そして選択されたアプローチから生じる限界条件の完全な処理法を含んでいる。ドキュメンテーションは、コードの実現やオペレーション、そしてデータファイルを準備するためにユーザーの指示も含まなければならない。それは入出力ファイル記述が完全なモデル公式化の例（例えば、グリッドデザイン、境界条件の指定）を与えなければならず、広範囲なコード検証と妥当性検査の報告を含まなければならない。最後に、プログラマー向けのドキュメンテーションは、コード修正とメンテナンスのための指示を与えなければならない。

ソフトウェア開発プロジェクトの最初にドキュメンテーションは開始されなければならないとはいえ、それはプロジェクトが完成になるまで放置されていることがある（Gass, 1979）。それは逆に、古い記録や消えていく記憶の調査を要するためドキュメンテーションをより一層困難にする。時たま例として言及される不完全なドキュメンテーションは、そのモデルが本来は研究道具であり、その後、政策評価の中心に登用されたからである。

４．２．８　化学的技術的レビュー

一般的に、完全な化学的で技術的なレビューの手順はその特質において定性的であり、以下から構成される（van der Heijde et al., 1985; Bryant and Wilburn, 1987）。

- 選択されたモデル概念、左右する方程式、そしてアルゴリズムの考察
- ドキュメンテーションと基本的な使用の容易さの評価
- プログラム構造とプログラム論理の検査
- エラー傾向の解析
- コンピュータコーディングの検査

検証または妥当性検査の実行が行われた場合、レビューの過程はこれらの過程の評価も含まなければならない。

総称モデルの徹底的なレビューを促進するためには、モデルの詳細なドキュメンテーションと開発歴史、同様に検査のためのソースコードが利用可能であることが必要とされる。それに加えて、検証や妥当性検査結果の再現可能性の独立的評価を確実にするため、レビューする者にとって、コード検証や、妥当性検査に使用されたオリジナルのテストデータを含むファイルとともに、コードが利用できる、または少なくとも使用のためにアクセス可能でなければならない。

モデル開発のレビューは、少なくとも地下水モデル開発の理論的な見解について知識のある化学者と、ソフトウェアエンジニアを含む経験豊富なチームによって行われるべきである。モデルの理論的構造は非常に複雑なため、流体力学、水文地質学、地球化学、汚染菌化学、水中微生物学、及び確率的数値解析等のレビューは個人よりは各専門分野協力のチームで行われるべきである。

（1）モデル考察

モデル考察は、原型システムにおける初期の概念化で何か基本的なものが省略されなかったかどうかを決定する。そのような手順は、研究中のシステムの特質をモデルの根本的な概念が十分に表わしているかどうかを決定し、モデルの意図された使用方法に適切な過程や動作を確認する。考察は、モデルの適応性の範囲において、様々な手順を表わす方程式が妥当であるかどうか、モデルの使用に意図された範囲においてこれらの方程式が数学的に適合されるかどうか、そして選択された解法アプローチが最も適切であるかどうかも決定する。モデル考察は選択された初期境界条件の適切さを決定し、モデルの適用性範囲を確立する。

複雑なモデルには、モデルを表わすために適切な数値方式が採用されているかどうかを決定するために、実現されたアルゴリズムの詳細な考察が要求される。この段階は、数値解法の非独特さ、数値パラメータの不十分な定義、これらのパラメータに使用される誤った、または最適ではない値、数値散布、振動や異なる解法などの数値不安定さ、そして多量の滞留に関する問題などの先天的数値問題を公開しなければならない（ASTM, 1984）。

それに加えて、モデルの適切なアプリケーションのための特定ルールは、その意図された使用方法の見地から解析されなければならない。そのようなルールの例は以下の通りである。

- 有限差分法のノード中心またはブロック中心のグリッド構造によるデータの指定
- 統合された有限差分法と有限要素法における要素の大きさや形
- グリッドサイズの変動
- 井戸など特異なものの処理法
- ２次元水平モデルまたは積み重ねた３次元モデルにおける垂直の平均化へのアプローチ
- 解析要素法における概算解法の算入
- 境界条件の処理法

数学的方程式、解法手順、及び最終結果が物理的に解釈できる容易さにも考慮する必要がある。

（2）コンピュータコード検査

モデルレビュー手順の一部がコンピュータコード検査である。この検査では、コード構造、プログラミング標準のコンプライアンス、プログラミング言語の効果的使用、そして内部ドキュメンテーシ

ョンに関して新式のプログラミング原理が適用された方法に注目される。この段階では、検証実行において検出されるのは難しい検出されていないプログラミングや理論エラーを示すかもしれない。

（3）総称モデルドキュメンテーションの評価

総称モデルのドキュメンテーションは、視覚検査、存在するドキュメンテーション標準とガイドラインとの比較、そして検証や妥当性検査実行を準備または実行するためのガイドとして使用することによって評価される。

（4）使用の容易さの評価

モデル開発者によって与えられたデータファイルは、テスト実行過程を通して、コードのオペレーションとユーザーガイドを評価するために使用されている。この段階においては、コードを操作するために必要なルールと制限（「策略」例えば、適用性の制限を克服するため）やコードの使用の容易化の見解について、特別注目されている（van der Heijde and Beljin, 1988）。

4．2．9　ポスト監査

総称シミュレーションモデルの妥当性を決定するために重要な役割は、ポスト監査に残されているかもしれない。ポスト監査は、主としてモデルアプリケーションの成功率を決定するために使用されるにもかかわらず、首尾よく実行されたポスト監査解析の明確な結果は、較正と関連して使用されている場合でもモデルの予測能力の受け入れに貢献する（例えば、Anderson and Woessner, 1992）。ポスト監査の使用を成功させるためには、概念化、仮定、そしてシステムパラメータやストレスを評価し、もし必要であればアップデートし、最近観察されたシステム反応の予測の比較を促進するためにモデルを再実行させなければならない。

4．3　モデルアプリケーションにおける検討事項

4．3．1　アプリケーション要件

サイト特定設計問題に対する総称シミュレーションモデルのアプリケーションは、「モデルアプリケーション」または「（コンピュータ、シミュレーション）コードアプリケーション」と呼ばれることがある。総称モデルのサイト特定設計条件へのアプリケーションは、首尾よく構成されたモデルアプリケーション原型を従わなければならない。そのような原型は、Mercer and Faust（1981）, van der Heijde et al.（1988）及びAnderson and Woessner（1992）によって説明されている。このようなタイプの地下水研究における品質保証は、総称モデル開発プロジェクトで論じられたパターンと同じものに従い、適切なデータの使用、データ解析手順、モデル開発方法論と技術、管理手順と監査から成り立つ。大部分においてモデル開発研究の品質は、モデル開発や品質評価チームの専門的技術によって決定されるということに注目しなければならない。コードアプリケーションにおける品質保証は、モデル開発手順の全ての面を指し、以下の論点を含む。

・解かれる問題の正確で明確な方程式
・プロジェクトの説明と目的
・プロジェクトのモデル開発アプローチのタイプ
・モデル開発が利用できる最良のアプローチであるか、そしてそうであるならば、選択されたコードが適切で費用対効果的であるか
・水文地質学的構造、境界条件、ストレス、そしてコントロールを含むシステムや手順の概念化
・明白なものと、無条件なもの両方の仮説や単純化の詳細な説明（批判的な仲間のレビューに従属する）
・データ獲得と解釈
・コード選択の考慮、または存在するコードの修正または新しいものの開発のための調整
・モデル準備（パラメータ選択、データ入力または再フォーマット、グリッディング）
・モデルアプリケーションに使用されたパラメータ値の妥当性
・特に感度の高いパラメータのため、ガイダンスを与えるためのパラメータ概算とモデル較正の原型
・コンピュータ出力の情報レベル（表示された変数やパラメータ。フォーマット。レイアウト）
・較正目標の確認やそれらがどの程度達成されたかの評価
・感度解析の役割
・ポストシミュレーション解析（輪郭を示すことに関しては、結果の合理性、結果の解釈、不確実性の解析、そして手動または自動データ処理技術の検証を含む）
・適切な性能目標の確立（例えば、6フィートヘッドエラーは、250フィート帯水層の厚さではなく、20フィートヘッドの傾斜または降下と比較されなければならない）。これらの目標はデータの限界を認知しなければならない
・結果の発表やドキュメンテーション
・管理者側から挙げられた質問に対して、モデル開発の結果がどの程度しっかりと解答しているかについての評価。

モデルアプリケーションにおける品質管理は、モデル開発過程の各段階の完全な記録を含まなければならない。品質管理のためのペーパートレーラは次の項目を挙げる報告やファイルを含まなければならない。

・仮定
・パラメータ値とソース
・境界と初期条件
・グリッドの特質とグリッドデザイン調整
・コードにおける変更と変更の検証
・実際に使用された入力
・モデル実行の出力と解釈
・モデルの妥当性検査（または少なくとも較正）

コード開発品質管理に関しては、監査やプロジェクト後の再利用のためにモデル開発研究で使用された以下を含む全てのデータファイル、ソースコード、そしてコンピュータソフトウェアの実行可能なバージョンが保持されなければならない（ハードコピーや、高レベルにおいてはデジタルの形で）。

・ソースのバージョンと使用されたコードの実行可能なイメージ
・較正の入出力
・検証の入出力
・アプリケーションの入出力（例えば、研究された各シナリオのもの）

モデル開発研究に使用されたコードに何らかの修正がされた場合、そのコードは標準テスティングプロトコルによって再度テストされなければならない。コードは、正確な記録や報告を含むコード開発のための完全な品質管理手順を受けなければならない。全ての新入出力ファイルは、現存するファイル、記録、コード、そしてデータセットとともに検査や再利用のために保存されなければならない。

モデルアプリケーションの比較研究において注目された主要な違い（例：McLaughlin and Johnson, 1987; Freyberg, 1988）やモデル開発結果の基本的な確信の欠如に照らして、効果的な品質保証のために、解析を少なくとも２つの独立したモデル開発チームによって行うことまで要求できるかもしれない。その場合は、第３モデル開発チームまたはレビューする者が両方のモデル開発労作の結果を比較し、その違いの重要性と特質を評定しなければならない。

４．３．２　データ収集

モデル開発プロジェクトにおける重要な段階（時には準備的）はデータ収集である。プロジェクトのこの段階における品質保証は、モデル入力の準備のために収集された技術データの取得、レビュー、照合、変換、そして統合において行われるべき段階や準備的な技術結論を定義付ける。

プロジェクトチームによって得られたデータは、利用できる様々な形の一つで、レビューやドキュメンテーションのあらゆるレベルにあるかもしれない（品質管理）。追跡可能性を保証するため、ドキュメンテーションのソースや記述の情報を含む受信の記録は、オリジナルデータや書類とともにファイルされなければならない。追加のデータ処理の間、動作やその後の発見の記録は保存されなければならない。そのようなデータ処理の例は、ユニット変換、輪郭作成、ノード割り当て等である。レビューする者は、変換が正当であるということを認められなければならない。記録は、実例計算や変換、そしてデータ処理のためのソフトウェア参照を含まなければならない（例えば、ソフトウェア名、プロバイダー、バージョン）。

４．３．３　モデル公式化

モデル公式化またはデザイン段階においては、システムや可能な解析手段の様々な面に注目が置かれている。それはシステム概念化（即ち、水文地質学的構造、適切な過程、そして現存する境界条件の定義付け）とモデル概念化（即ち、モデル開発研究の目標と一致するようにシステムを単純化するために、プロジェクトチームは何を正しいとみなすか）を含む。この段階における品質管理は、以下のような全ての問題点における解析のドキュメンテーションを要求する。

・地下水と地上水の間の相互作用、状態の本質的関係や方程式の決定。
・時間の尺度の決定（安定状態対過渡状態）。
・システムが飽和されているか、または不飽和か。
・モデルの次元。

・境界条件。
・そしてシステムの初期状態。

モデルを使用すると決定された後、研究目的やモデル開発チームの人材と物質資源の限界を熟慮しながら、存在するコードのよく定義され品質保証された特性とモデル開発要求の詳細な記述を突き合わせることによって、適切なコードが選択される（van der Heijde, 1987）。モデル要件とコード特性の間に良い突き合わせが見つからない場合は、突き合わせ規準のいくつかの免除、存在するコードの修正、または新しいコードの開発を考慮してもよい。別のセクションにて詳細なモデル選択を説明し、品質管理コードの選択を強調する。

どこか他から得られなければならないコードが選択された場合は、コード転送と実現において特別な扱いが適用される。ここにおいては、品質管理とはドキュメンテーションやコンピュータファイルの完全性の評価、転送メディアフォーマットのドキュメンテーション、そして受信通知や受け入れサインオフのファイリングを意味する。

地下水モデルの適用における決定的段階のひとつは、グリッドのデザインである（Mercer and Faust, 1981）。一般的に、グリッドが良質であればあるほど、解は正確になる。選択されたモデルに従属して、グリッド過程には様々なルールが適応されるかもしれない（van der Heijde et al., 1988）。以下に実例が見られる。

1）井戸のフィールド中心の井戸またはその付近のグリッドノードの場所。
2）詳細な解析が要求され、高度な正確さが重要な良質グリッドのデザイン（解像であり、正確さは問題にしない）。または空間的特性に生じるかもしれない大きな変化（これはモデルの他の部分において正確性に影響を与えるかもしれない）。
3）他の異方パラメータまたは水力伝導率の主要な方向とグリッド軸を一直線に合わせること。

他のルールは、例えば有限差分グリッドにおける不規則な境界の処理や、大きい要素の隣の小さいグリッドセルの使用などの特定の方法に適用される。グリッド過程における他の考慮は下記の通りである。

・空間的尺度と離散の関係
・（相対的）正確さと離散のトレードオフ
・プロジェクト目標を達成するための空間平均や集合の役割
・時間と空間における種々の正確さの要件

モデルアプリケーションにおける次の段階は、選択されたコンピュータコードの実行に適切なノードまたは要素の値を指定することによって入力ファイルを準備することである。

4．3．4　モデルの較正と検証

較正とは、結果として生じる仮説が観察されているデータに適度な当てはまりを見せるまでモデル入力を調節する過程である（NRC1990）。較正のほとんどは安定状況の条件で実行されるが、過渡なデータのセットに対して2度目の較正を含むこともある。普通は、較正は測定や結果として起こる

データ解析に基づいたモデル入力の最良の概算値から始められる。

初期のシミュレーション結果は、システムの概念を改善するため、またはパラメータ値を修正するために使用できる。モデル較正の成功は基礎となるモデル公式化の妥当性に従属している。モデル構造が重要なソースを無視した場合、地質学的不均質、物質的過程、または化学反応、パラメータ概算、及びモデル較正は、有効な入力に適切な公式化を補う（普通は不十分に）ことを強いる適合問題に減らされる（NRC 1990）。

較正の過程においては、より多くのデータが必要になることがある。なかには、コードが初めにデータ収集プログラムをデザインするために使用されることもある。新しく収集されたデータは、システムの概念化を改善し、予測的シミュレーションを準備する両方のために使用できる。較正の際に修正を受けやすいモデル入力は、以下を含む。

- 本質的係数とパラメータ（例えば、水力伝導性、散布係数と分割係数）
- 強制技術用語（例えば、水または汚染物質のソースやシンク）
- 境界条件（特定されたヘッド、密集、流動）

較正の際に、感度解析は較正手順の結果にモデル入力のどの要素が最も決定的であるか確認するのを補助するために行われる。どのパラメータにおいても受け入れられる調整の度合いは、一般的に、その値や詳記の不確実性に対して直接比例し、予想された範囲の値または確信間隔に限定されている（Konikow and Patten, 1985）。

普段は、満足できる正確さのレベル内において履歴データを再生した場合、モデルは較正されたことになり、較正問題に先立って決定される。特に履歴的適合が少量のデータに基づくならば、または予測をするために要求されているモデルの可能性をテストしないならば、履歴的データへの適合は良い予測を保証しない、ということを注目しておかなければならない（NRC 1990）。

較正は、モデル入力データの試行錯誤の調整によって達成されることがある。無関係な要因の多くがモデル出力に影響を与えるため、これはかなり主観的な手順となる（Konikow and Patten, 1985）。自動パラメータ確認手順が、この主観性のいくつかを削除するのを助けるかもしれない。モデラーの水分学的経験と判断力が、モデルを正確に効率的に較正するための主要な要因になる。

較正の重要な品質管理面は較正目標の使用である（Anderson and Woessner, 1992）。例えば、流動においては、ヘッドや流動のフィールド測定（例えば、流れの出発点、泉の流れ、衰える流れからの浸透、水地層からの蒸発）がサンプル情報または較正値をつくる。較正値とその関連したエラーが較正目標をつくり、これはモデルを較正する前に決定されなければならない。関連したエラーは以下を含む可能性がある。

- 測定エラー
- 尺度のエラー（モデル変数の測定の典型）
- 補間エラー（測定された情報からノード値への転送）

流動モデルのための追加の較正情報は、速度や解決した散布によって得ることができる。そのような較正情報の追加は「独特の」解を得る可能性を高くするかもしれない（NRC, 1990）。

通常使用される較正測定は特質上、定性的か定量的である。較正進歩の定性的測定は以下に基づく

かもしれない。

・エラーの空間的散布の情報を与える、測定とシミュレートされた変数の輪郭図の比較
・較正エラーの輪郭化（剰余）
・各セルまたは要素における較正エラーのグラフ説明（３次元棒グラフと比較可能）
・測定されたもの対シミュレートされた変数の散布図表。直線からの点の偏差はランダムに散布されていなければならない（線形回帰の確信間隔を含むかもしれない）
・測定とシミュレートされた変数の製表化、そしてそれらの各ノードの偏差

定量的測定は、較正規準と呼ばれることもあるが、以下を含む（Anderson and Woessner 1992）。

・平均誤差（ME）は測定されたヘッドとシミュレートされたヘッドの平均差である。

$$ME = \frac{1}{n}\sum_{i=1}^{n} (h_m - h_s)_i$$

式４．３－１

nは較正値の数である。MEは較正されたパラメータの様々な値に対してグラフによって表わすことができる。負と正の差の両方がMEに合体しており、互いに削除できる。

・平均絶対誤差（MAE）は、測定されたヘッドとシミュレートされたヘッドの違いの平均絶対値である。

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |(h_m - h_s)_i|$$

式４．３－２

・標準偏差の平均平方根（RMS）は、測定されたヘッドとシミュレートされたヘッドの二乗の差の平均である。

$$RMS = \left[\frac{1}{n}\sum_{i=1}^{n} (h_m - h_s)_i^2 \right]^{0.5}$$

式４．３－３

較正の成功の全体的アセスメントは、較正レベルによって表わされる（Anderson and Woessner 1992）。

レベル1。シミュレートされた値は、目標範囲内に入る（較正の最高度合い）
レベル2。シミュレートされた値は、較正目標と関連したエラーの2倍の範囲内に入る
レベル3。シミュレートされた値は、較正目標と関連したエラーの3倍の範囲内に入る
レベルn。シミュレートされた値は、較正目標と関連したエラーのn倍の範囲内に入る

較正されたモデルにおいて使用されたパラメータのセットがフィールド値を正確に表わさないかもしれないため、較正されたパラメータは別のセットの境界条件や水文学的ストレスのシステムを表わさないかも知れない。モデル検証が、較正されたモデルの較正と予測の可能性において、より大きな確信をつくることを補助する（Anderson and Woessner, 1992）。

較正データから独立したテストによって、その正確性と予測可能性が受け入れられる範囲内のエラーにあると証明されたならば、サイト特定設計モデルは「検証」されたことになる。概して、モデル検証は過渡データのセットを使用して、例えば、間伐または長期的ポンピングへのヘッド反応によって行われる。単一の時系列しか利用できない場合は、そのシリーズを２つに分割してもよく、ひとつは較正のため（例えば、ストレスにおける自然な周期的変動へのシステム反応の説明）、そしてもう１つは検証（例えば、灌漑開発などの原因によって起こる大きく変化したストレス状態のシステム反応の説明）のためである。そのようなデータが入手できないならば、検証は第２の「独立した」安定状態のデータセットによって行うことができる（即ち、以前には較正として使用されていない）。検証中にパラメータが変更されたならば、この問題は第２の較正となり、第１の較正は変更の説明をするため繰り返さなければならない。第２の問題はもはや「検証」として考慮されない。

品質管理の見地から、首尾よく定義されたモデル較正手順への執着は決定的である。レポートは、較正の目標、較正の測定、較正された変数、較正アセスメント（即ち、較正された変数の適切さの評価と確立された分類による較正測定と較正目標の比較）、そして可能であれば、モデル検証結果の説明を含まなければならない。

４．３．５　感度解析

モデルの重要な特性は、変動への感度と入力パラメータにおける不確定さである。感度解析は、モデル予測の正確さと精密さを決定するための最も影響を与えるパラメータを確認するために使用される。この情報は、データの質と量の機能としてモデルアプリケーションにおける正確さと精密さを確立させなければならないため、ユーザーにとっては重要である（Hern, et al. 1985）。感度解析は、ある特定パラメータまたはパラメータのセットにおける選択されたモデル性能アセスメント測定（または中間変動）の従属状態を定量的または半定量的に定義付ける（Intera, 1983）。感度解析は、シミュレーションモデルの単純化の方法を決定するためにも（例えば、使用された手順、次元、時空の変動、境界条件等）、較正手順の効率性を改善するためにも使用可能である。

モデル感度は、入力のユニット変更によって起こる選択された出力の相対的変化として表わされる。入力における変更が出力に大きな変更をもたらす場合、モデルはその入力に敏感ということになる。感度解析方法は、ほとんどが非統計的で、その特質は直観的でさえある。これに関連して、Fjeld et al.（1987）によって説明された感度指数が注目されている。

感度解析は、典型的には毎回１入力パラメータを変更し、従属変数の散布の結果を評価することによって行われる。例えば水力ヘッドである。選択された入力パラメータの公称値、最小値、そして最

大値が特定されている。従属変数の散布はその公称値を決定されており、Ct nomとなる。そして与えられたパラメータの最小値と最大値は、そ公称値の代理となり従属変数における限界をもたらす。すなわち、パラメータpの最小値に相当するCt pmhと、パラメータpの最大値に相当するCt pm4である。感度指数Stは以下のように定義付けられる（Fjeld et al. 1987）。

$$S_t = \frac{C_t - C_t^{nom}}{C_{nom}^{max}}$$

式４．３－４

これにおいてCtは、Ct pmhまたはCt pm4である。Cmax nomはパラメータの公称値に基づいた従属変数の最大瞬間値である（すなわち、最大時間における従属変数の公称値である）。

汚染物質のマルチメディア的運搬のようにモデルが結合された場合、その後のシミュレーションによる伝播エラーや不正確さの増加が感度解析の一部として解析されなければならない、ということに注目しなければならない。

詳細な感度解析は骨の折れる多くの時間を要する作業ではあるが、それは通常、おそらく直感によって、確認されたいくつかの決定的な入力に集中する小さい規模の探究的な解析が実行可能である。感度解析は、較正戦略をデザインするために較正を始める初期に実行されなければならない。較正が完成された後で、帯水層パラメータ、ストレス、そして境界条件の不確実さによってもたらされた、較正されたモデルの不確実さをはかるために、より精巧な感度解析が行われる。

首尾よくドキュメンテーションされた感度解析は、シナリオ解析の段階において実行されたシミュレーションの様々な面を評価する最高の機会をレビューする者に与える。従って、感度解析の結果は定性的に説明されなければならない（Anderson and Woessner 1992）。

４．３．６　シナリオ解析

ほとんどのモデルアプリケーションの目標は、提案された管理者の意思決定を支持する代理シナリオのシナリオ解析とその後のスクリーニングである。従ってシナリオ解析段階は慎重に実行され、広範囲にドキュメンテーションされなければならない。そのようなドキュメンテーションの重要な要素は、使用されたコードの実行オプション、入力データの完全なセット（タイプとフォーマット）、そしてシミュレーションの直接的な結果の概略（操作されていない、または処理後）を含む。

４．３．７　結果の解釈とプレゼンテーション

この段階における品質管理は、結果の形成や計算結果とポストシミュレーション解析の意味と合理性についての検査を含む。予測のコントロール計算やポストシミュレーションの妥当性検査（ポスト監査を参照）を行うことが品質管理構造における重要な手段である。

確定的モデルによって行われた予測と確率的モデルによって得られた予測の妥当性検査の間には重大な違いがあることを注目しなければならない。不確定解析の数学的モデルの妥当性検査、例えば、

性能アセスメントにおいて評価された各シナリオの確率散布計算を確証しなければならない。計算された確率は、シナリオの性能測定の反応を予測し、制御するパラメータが曖昧な値の範囲内にて変化することを認めるならば、結果が出る（Brandstetter and Buxton, 1987）。曖昧な値の範囲から繰り返しランダムにパラメータ値を選択することによって、再サンプリング研究は行われなければならない。計算は新しい各パラメータのセットと行われなければならない。

４．３．８　ポスト監査

もうひとつの重要な品質管理手順はポスト監査研究に見られる。モデル化されているシステムに関するフィールド情報を更に入手する機会があるたび、モデルにおける工夫や改善をし、前の解析を修正しなければならない。そのような機会がポスト監査という形によって提供されることがある。ポスト監査とは、初期予測の正確さの評価であり、オリジナルのモデル開発研究が完成した数年後に行われる（Anderson and Woessner, 1992）。モデルの予測が正確であったならば、そのモデルは、あるサイト特定設計と実際のストレスにおいては「妥当」とみなされる。ポスト監査は、予測された変数において新しいフィールド観察を要求し、それは予測された変化にシステムが適応する機会が与えられた後に収集される。Konikow（1986）、Person and Konikow（1986）、及びLewis and Goldstein（1982）による研究が、新しい情報が利用可能になった時、またはある一定時間が過ぎた後にシステムの予測された変化の結果が明白になった時にモデルに基づいた予測を評価する必要があること論証した。

４．４　品質管理組織構造

品質管理は、プロジェクトチーム（品質管理と内部監査）と契約、または監督する組織（品質評価）の両方における責任である。品質管理はプロジェクトの方向を推進または管理してはならず、品質管理は技術データの事後ファイリングを意図していない。ソフトウェア開発またはモデルアプリケーション研究を遂行する組織内の品質管理構造には以下の２つの段階がある。

- **完全な品質管理管理方針、目標や目的を持つ常任組織があること**
- **一般的品質管理方針や研究課題がプロジェクト目的に向けて詳細であること**

プロジェクト品質管理組織である、管理の上のレベルにおいては、品質管理がソフトウェア開発やモデル開発研究の欠くことのできない部分であるということを認識しなければならない。上位の管理者におけるそのような認識は、品質目標を掲げる方針、品質管理機能の確立、そして仕事を実行するために必要な人材、資金、機材などの資源を通して公約に変えなければならない（Bryant and Wilburn, 1987）。

品質管理組織は、明らかにされた組織の各要素に特許が無ければならず、責任の大要が必要であり、品質管理の責任者は、ソフトウェア開発またはモデルアプリケーションの責任者から独立していなければならない。品質管理組織は、プロダクト開発や配信の何においても従属的であってはならない（Bryant and Wilburn, 1987）。

品質管理プログラムを成功させる鍵（キー）は、スタッフの競合にある。品質管理のスタッフは一緒に働くプロジェクトスタッフに対して尊敬がなければならない。彼等は、彼等が品質を保証する仕事が実際にはどのように達成されているかを理解しなければならない。成功した品質管理プログラム

の実現は、関係者全員が、品質管理とは何か、なぜそれが行われ、どのような利益があり、彼等に何が期待され、各個人の責任とは何か、そして良質な品質管理はモデル開発手順を妨げるよりもむしろ促進させるということを理解することを要する（Bryant and Wilburn, 1987）。彼等は品質管理の便利さと管理によって与えられた重要性を確信しなければならない。

プロジェクト完成の成功と同様に、成功した品質管理は、モデル開発チームの可能性に従属している。良いモデル開発チームは各専門分野協力のもので、かなり訓練された、幅広い経験を持つ先任スタッフを含み、効果的な内部のコミュニケーションが見られる。このようなチームは、プロジェクトに関係した様々な学問分野の概略を知り、管理者の質問を技術的プロジェクト目標に直すことができ、管理者側にモデル開発結果を説明できるような人によって管理されている。

４．５　品質管理プラン

モデル開発またはアプリケーションプロジェクトの初めには、完全な品質管理手順のセットを含むプロジェクトプランが立てられなければならず、これは品質管理プランと呼ばれている。これらの品質管理手順は、規定された品質目標を達成するための規準のリストを含む。品質管理プランは技術的作業の着手の前に承認が必要である。プロジェクトプランや品質管理プランのそのような品質管理部分の主要な要素は以下のとおりである。

- 妥当性、不確実さ、正確さ、完全性、そして比較可能性に関する要求された品質レベルや品質管理目標の公式化
- 適切なソフトウェア開発やモデルアプリケーション研究を行うためのオペレーション手順や標準の開発
- 品質標準が維持されたということをドキュメントにするために品質管理作業のペーパートレールを確立
- 内外部の監査やレビュー手順のための品質管理上重要点の洗い出し

品質管理プランは、これらの目標達成のための個人的責任や品質評価段階において、万が一問題が確認された場合の補修または訂正の行動手順の概要の特定をしなければならない。

総称的コンピュータ・ソフトウェアメンテナンスと品質保証プランの説明的実例が、Wilkinson and Runkle（1986）によって与えられている。サンディア国立研究所（Sandia National Laboratories）における高レベル廃棄物管理プログラムの実行がHarlan and Wilkinson（1988）によって説明されている。このプランは、ソフトウェアメンテナンス過程の詳細な記述と同時にソフトウェア保管とドキュメンテーションの要件を含む。

サンディア国立研究所（Sandia National Laboratories）は、品質管理プランを実行するためのコンピュータ化されたソフトウェア管理システムを確立した。このシステムの目標は次の通りである。

- 現在、そして最近に使用されたソフトウェアのバージョンの貯蔵所（追跡可能性と検索可能性のため）にする
- 報告書、エラーのドキュメンテーション、修正や強化、検証シミュレーション結果等々を含むドキュメンテーションの貯蔵所にする

これに関連して、検索可能性とは、ある一定時間の経過によってハードウェアと、またはシステムソフトウェアに変化があり、そのため実行不可能かもしれないにもかかわらず、適当な時間において開発されたソフトウェアの以前のバージョンにアクセスできることである。追跡可能性とは、計算に使用されたソフトウェアの実際の構成を確認することができることである（Wilkinson and Runkle 1986）。

サンディア品質管理プランは、米国核規制委員会（the U.S. Nuclear Regulatory Commission）（Codell and Silling, 1984）による「オペレーションソフトウェアのための品質保証プラン」に従って準備され、NRCのNUREG-0856（Silling 1983）に応じることを目的としている。それは、ソフトウェアテスティング、原稿レビュー、ソフトウェア問題報告、コンピュータコード要約記述、アップデートログ（データとバージョンを含む）、そしてソフトウェアアップデート記述における標準フォームへの準備やファイリングの説明を含む。

典型的なモデルアプリケーション品質管理プランは、プロジェクトにおける全ての主要な作業の標準的フォームへの準備、レビュー、ファイリングと、全ての適切なプロジェクトデータ、解析、そしてプロダクト（レポート、電子ファイル、地図等）の記述を提供する。それはプロジェクトにおける責任の連鎖と適応された品質管理組織を表わさなければならない。含まれているフォームの例は以下のとおりである。

- 適応されたモデル開発手順の全要素のチェックリスト（4．6項）
- 第3者からのデータやソフトウェア転送のフォーム
- 標準化されたプロジェクトノートの使用（概念化の仮定、データ評価、コード実現、モデルセットアップ、入力ファイルの準備、ポスト処理、その他に関連した手書きまたはワープロのノートを含む）
- データのフォーム、シミュレーションと報告のレビュー
- 標準化されたソフトウェア問題報告やコンピュータコード要約記述（データとバージョンを含む）。

全てのフォームは、ファイリング日時と品質管理承認のために余白を含まなければならない。

4．6　品質管理アセスメント

品質評価は2つの要素から成り立つ。監査と技術的レビューである（van der Heijde, 1987）。監査とは、プロジェクトに規定された品質管理のレベルと同じ度合いで品質管理要件とのコンプライアンスの程度を評定するためにデザインされた手順である。コンプライアンスは、記録の追跡可能性、有効性（責任者のスタッフからの承認）、そして品質管理プランにおける公約の実現から測定される。技術的レビューは、プロジェクトの技術的・化学的基礎の独立した評価と、プロジェクト結果の有益さから成り立つ。

品質管理アセスメントは、手順が正確に適用されたかどうかをチェックするだけではなく、プロジェクトのオリジナルの目的の全体的成功を定量的に確立することにも関係する。モデル開発とアプリケーションの両方において、品質評価の様々な面が存在する。まず最初に、ソフトウェア開発またはレビュー、そして責任のある研究者によるコントロール計算の実行におけるレビューとテスティングであり、時にはプロジェクトに参加していない他のスタッフ、または組織外の招かれた専門家が含まれる。プロジェクトを着手した組織による品質評価も考慮されなければならない。再び、プロジェク

ト、プロジェクトレビュー、またはテスティングにおいて3段階に分類される（van der Heijde, 1987）。

1）プロジェクト管理者またはプロジェクトモニターによるレビューまたはテスティング。
2）管理または資金を出している組織内のプロジェクトには関係していない技術専門家によるレビューまたはテスティング。
3）外部の同僚のパネルによるレビューまたはテスティング

品質評価は、一般的には技術と管理のレビューの形を取る。論評はメモやレポートの形、またはレビューされたドキュメントの許可済みのコピーの余白に注釈として表わすことができる。もし、このレビューが追加の研究や訂正の作業のための重要な推薦に至った場合、プロジェクトチームによる追従と追加のレビューが取り計られドキュメントにされなければならない。

ソフトウェア開発のように、モデルアプリケーションプロジェクトのレビューはますます様々な化学的・工学的分野と関係するようになったことに注目しなければならない。そのようなプロジェクトにおいては、各専門分野協力のレビューチームまたはパネルを設置することが最も適切なアプローチかも知れない。

（1）モデル開発アプリケーション目標

01 管理側の決定目標とモデル開発目標は明確に特定されなければならない。
02 管理側の決定目標は、サイトの物質的特性（例えば、水文地質学的システム）や汚染のソースと場所についての存在する情報に基づかれなければならない。
03 モデルの機能（例えば、データ構造、システムの理解、代わりの補修の追加のフィールド特性記述の計画または評価）はモデル開発目標の開発中に明らかにされなければならない。
04 モデル開発着手の前に、評価されるべきポテンシャル解法（例えば、包含と補修解法）は確認されなければならない。
05 要求される解析のレベル（例えば、数値モデル、解析モデルまたは図解技術）は、モデル開発目標を明らかにする際に決定されなければならない。
06 管理者は、専門の地下水科学者との相談をもとに、モデル予測に必要な時間（例えば、1年、10年、または100年）を特定しなければならない。
07 モデル開発結果に要求される確信のレベル（定量的または定性的）が明確にされなければならない。
08 モデルアプリケーションの性能目標（例えば、許容ヘッドエラー）が明確にされなければならない。
09 これらの研究目標の発展と関連する増分コスト（例えば、研究領域の大きさ、モデル化された補修技術の数、またはモデルの性能目標の拡張）と、管理決定目標を支持する結果として起こる増分改善についての解析が行われなければならない。
10 管理側の決定目標は、モデル開発過程の全般にわたって再確認されなければならない。
11 モデル開発着手の前と概念モデルの開発後に、それらが管理側の決定目標を支持していることを確証するために、モデル開発目標はレビューされなければならない。
12 要求される解析レベルは、プロジェクトの経過の間にレビューされ、そして必要であれば修正

されなければならない。

13 モデルの機能は、プロジェクトの経過の間にレビューされ、そして必要であれば修正されなければならない。

（2）プロジェクト管理

14 実際にモデル開発を実行し、モデル開発労作の管理や同僚のレビューを行っている人達は、プロジェクトに要求される地下水モデル開発経験を持たなければならない。明確には、プロジェクトにおける彼等の役割のため、各自が以下のような適当なレベルを持たなければならない

- モデル開発と水文地質学における正式な訓練
- 物質的システムをモデル開発する職務経歴
- サイト水文地質学の特性を表わすフィールド経験
- モデル開発プロジェクトの管理経験

15 これらの各個人は、よく明確にされた役割、責任、そして参加レベルを持つ結合力のあるモデル開発チームとして編制されなければならない。

16 チーム編制はアプリケーションに適切でなければならない。

17 プロジェクトの最初に、独立した品質保証（品質管理）過程が確立されなければならない。

18 この品質管理手順は、以下に対する進行する同僚のレビューを含まなければならない。

- モデル開発目標開発
- 概念モデル開発
- モデルコード選択
- モデルセットアップと較正
- シナリオのシミュレーション
- ポストシミュレーション解析

19 この品質管理過程は実現されなければならない。

20 モデルアプリケーションをドキュメントにするため、手順が確立されなければならない。

21 ドキュメンテーションは以下に関する説明を含まなければならない。

- 一般的なサイトのセッティング
- 関心のある物質的システム
- 評価すべきであるポテンシャル解法
- モデル予測のためのモデル開発目標と時間の骨組み
- 品質保証と同僚のレビュー過程
- モデル開発チームの構造
- データソースとデータ品質
- 概念モデル。水層序学、地下水流動システム、水文学的境界、水力学的特性、流動体ソースとシンク、汚染物質のソースの船荷と地域の広がり、そして汚染物質の輸送と変形過程
- コンピュータコードの選択。コード、信頼性、使用性、輸送能力、性能、公衆領域対所有モデル、限界、そして関連したアプリケーションの記述
- 地下水モデル建設。コード修正、モデルグリッド、水力パラメータ、境界条件、そして単純化した仮定
- 較正、感度解析、そして検証
- 予測シミュレーション。シナリオ、シナリオの実現、そして各実行結果の説明
- 不確実性の解析

・決定目標に公式化されたような管理側の情報要求の結果に関する説明
・実行上の要約（決定目標による）
・参照
・入出力ファイル

22 ドキュメンテーションは、独立したレビューする者がポストアプリケーションアセスメントを完成するために必要な情報を提供しなければならない。

（3）概念モデル開発

23 どのようなコンピュータモデル開発の前にも、局地と広域の水文地質学的システムの初期概念モデルが開発されなければならない。

24 概念モデルは、以下の情報を含む他の定性的データと同様、フィールドデータの定量化に基づかなければならない。

・帯水層システム（帯水層と半透水層ユニットの散布と相関位置）。関連したユニットの厚さと連続性、地域の広がり、そしてユニット間の相互連結
・水文学的境界。帯水層システムの物質的広がり、地下水システムをコントロールまたは衝撃を与える水文学的特性、地下水分割、そして地上水体
・水力学的特性（何に関連し、均質で等方性の特性を含む）。伝達可能性、間隙涵率、水力学的伝導性、蓄積可能性、そして特定産出
・ソースとシンク。帯水層への涵養（例えば浸透）、蒸発散、排水溝、地下水流量（例えば、地上水体への流動）、そして井戸（例えば、給水量、注入または灌漑井戸、水平井戸）
・流体ポテンシャル（例えば、電位差計の表面、各モデル層内の水力学的傾斜の振幅と方向）
・汚染物質。ソース、流送、地域の広がり、物定性的特性、化学相互作用、そして生物変態
・土壌

25 フィールドデータの量、質、そして完全性は概念モデルの開発の一部として解析されるべきである。

26 データにギャップがある場合（例えば、水位または水力学的伝導性の情報の欠如）、これらのギャップを埋めるための追加のフィールドワークや他の試みをドキュメントにする必要がある。

27 データにギャップがある場合、追加のデータを取得する際のコストと結果として起こる管理側の決定目標を満たす改善の間から解析されなければならない。

28 データソースはドキュメントにされなければならない。

29 データの品質は調査されドキュメントにされなければならず、プロジェクト結果における品質の影響は評定されなければならない。

30 他の物質的システムとの全ての可能性のある相互作用（例えば、地上水システムまたは農耕システム）はモデル開発の初めから、水予算、大量化学の見積り、または他の解析学的技術という手段で評価されなければならない。

31 存在する、または未来の工学（例えば、井戸、スラリー壁）が数値または解析モデルに表わされなければならないということは、概念モデルに明確に含まれなければならない。

32 水柱中の汚染物質の塊を説明するため、十分な汚染物質ソースが確認されなければならない。

33 境界条件の場所、タイプ、そして状態の明確な報告書。それらの公式化の調整、そして境界条件の情報のソースが概念モデルの一部として含まれなければならない。

34 数値モデルの較正の始まる前に、全ての概念モデルパラメータと適度なパラメータ範囲が特定されなければならない。

（4）モデルコード選択

35 選択されたモデル（コード）は、その流動と輸送過程、数学、水文地質学的システム表示、境界条件と入力パラメータに関連して説明されなければならない。

36 モデル（コード）の信頼性は下記のレビューを含めて評定されなければならない。

- 同僚によるモデル理論のレビュー（例えば、地下水の専門的知識または同僚にレビューされたジャーナルでの理論の出版によって一般に認められた個人または組織による正式なレビュー過程）
- 同僚によるモデルコードのレビュー（例えば、地下水コンピュータモデルを評定する専門的知識のため一般に認められている個人または組織による正式なレビュー過程を受けなければならない）
- 検証研究（例えば実験室テスト、解析解法またはよく受け入れられたモデルに反対したモデル結果の評価）
- 関連したフィールドテスト（すなわち、広範囲なデータのセットが利用できるサイト特定設計条件のモデルのアプリケーションと評価）
- 使用の量とタイプによって立証されたようなユーザー社会におけるモデル（コード）の受容性

37 モデル（コード）の使用性は以下の利用可能性を含めて評価されなければならない

- モデルの2進コード
- モデルのソースコード
- 前後のプロセッサー
- 存在するデータ資源
- 標準化されたデータフォーマット
- 完全なユーザー指示マニュアル
- サンプル問題
- 必要なハードウェア
- プラットフォームを横断する輸送能力
- ユーザーサポート

38 トレードオフは、モデル（コード）性能（例えば、正確性と処理速度）とモデル開発を実行するために要求されている人材とコンピュータ資源の間で解析されなければならない。

39 モデル（コード）は、公衆領域にあるか、少なくともすべての関心を持つ全ての団体に即座にアクセス可能でなければならない。もしそうでなければ、モデラーは、アクセス不可が研究目的や調整機能過程から品質を落とさないことを説明しなければならない。

40 モデル（コード）における仮定は、モデル開発目的の衝撃に関して解析されなければならない。

41 モデル開発要件の間の全ての不一致（決定目標、概念モデル、そして利用可能データによって示される）と選択されたモデルの可能性が確認され調整されなければならない。
モデラーは、何故モデル開発目標と、または概念モデルが修正される必要があったのか無かったのか、説明しなければならない。例えば、1、2、3次元モデル開発を支持する選択されたコードの実現。安定対不安定状態のモデル開発の提供。または概念モデルの単純化の要求が説明されなければならない。

42 そのような不一致によってモデル開発目標が修正された場合、それらの修正はドキュメントにされなければならない。

43 モデルソースコードが修正された場合は、テストを実行し、テスティング方法論と結果を調整しなければならない。

・信頼性のテスティング（＃36参照）

・使用性の評価（＃37参照）

・性能テスティング。

（5）モデルセットアップと入力概算

44 全体的なグリッド解像（例えば、100フィート対1000フィートの間隔の平均）は管理側の目標を満たすために要求されている従属変数の正確性に関連して解析されなければならない。例えば、グリッドは、水力学的傾斜が正確に表わされることを可能にするほど良質でなければならない。

45 有限要素または有限差分グリッドデザインは、正確に井戸の場所を見つけたりモデル開発したりする必要性、存在するものと未来の工学、または汚染物質のソースと水柱などのモデル開発目標と関連して解析されなければならない。

46 グリッドは物定性的システムに関連してデザインされなければならない。例えば以下の通りである。

・主要グリッドオリエンテーションは、水力学的伝導性と、または伝達可能性の主要な方向と一直線に合わせられなければならない。

・結果が必要な場所（例えば、最も汚染されている場所または水位降下）、または伝達可能性の変化、水力学的ヘッドの変化、または密集した傾斜のある場所では、より良質なグリッドが使用されなければならない。

・データが乏しいところや、研究地域の特に関心のない部分においては、より粗いグリッドが使用されなければならない。

47 グリッドの間隔とタイムステップの大きさは数の正確さに従って解析されなければならない。例えば以下の通りである。

・変数グリッドの間隔がおかれた有限差分モデルが利用された場合、そのグリッドは離れた境界に向かって2の因数に拡大されなければならない。

・有限要素モデルが使用された場合、モデルアプリケーションの数値の正確さの衝撃に関して次のものが解析されなければならない。各要素の長さと幅の割合、隣接する要素の多きさの違い、そしてペクレ数（Pe=v・ΔX/D）。

・クーラン数（Cr=v・Δt/ΔX）は<1でなければならない。

D＝散布係数［L2/T］

Δt＝時間の段階［T］

V＝速度［L/T］

ΔX＝グリッドの間隔［L］

・もしランダムウォーク粒子のトラッキングが使用されたならば、グリッドの間隔と粒子の質量は汚染物質解像と関連して解析されなければならない。

48 グリッドにおける境界条件の場所の地図作成は評価されなければならない。例えば以下の通りである。

・モデルにおける不安定さを弱らせるため、境界は対象となっている場所から十分離れたところに位置し

なければならない

・グリッドに表わされている境界の形は、グリッドの良質さ、幾何学の正確さと境界条件の正確さを確証しなければならず

・有限要素グリッドにおいては、内外の境界が要素境界と一致しなければならない

49 井戸のノードは井戸の物質的場所の近くになければならない。

50 モデル入力データのデータソース、データ収集手順、そしてデータ不確実性は、プロジェクトレポートまたはファイルに評価されドキュメント化されなければならない。

51 全てのモデル入力は、それらが測定、概算または仮定のどれに当てはまるか明確にされなければならず以下を含む。

・構造する係数とパラメータ（すなわち、一般的には観測することができないが他の変数の観測から推測されなければならないパラメータ、例えば伝達可能性や特定蓄積の散布）

・強制用語（例えば、水や溶解した汚染物質のソースとシンク）

・境界条件

・初期条件

52 データがモデル入力に変換される入力概算過程は（例えば、時空の補間と補外またはクリギング）説明されなければならず、補間を行うために使用された空間の場所とデータの連合した値は地図に示すか表にて提供されなければならない。

53 入力概算過程と連合した不確実さは明確にされ、説明され、ドキュメントにされなければならない。

54 モデルの較正をしなければならない。

55 モデルが較正されていない場合、モデルを較正しなかった理由を説明しなければならない。

56 較正過程を終結させるために使用されている規準は（観測値とモデル化された値の間の適切な組み合わせの定義付け）モデル開発目標と関連して調整されなければならない。

57 較正は一般的に受け入れられる形で行わなければならない。明確には以下の通りである。

・較正において調査されるべき主要パラメータと境界条件を決定するために、感度解析が行われなければならない

・較正はシミュレートされた値と測定された値の間の誤差の計算を含まなければならない

・較正は時空の誤差の評価を含まなければならない

・較正は物質的特性に関連して行われなければならない（例えば、土手、陥没、または地下水流量および涵養を含む地下水輪郭のパターンに関連して誤差は解析されたか）

58 水予算が展開された場合、モデルを較正する際のそれらの結果や使用が説明されなければならない。

59 較正による初期モデルパラメータ値の全ての変化はその合理性に対して調整されなければならない。

60 概念モデルにおける較正されたモデルパラメータと概算されたパラメータ範囲の間のあらゆる不一致は調整されなければならない。

61 モデル較正の結果、概念モデルが修正された場合、概念モデルにおける全ての変更は調整されなければならない。実行できるときはいつでも、較正されたモデルは独立したフィールド観察のセットによって検証されなければならない。

（6）シナリオのシミュレーション

62 各モデル開発シナリオにおいて、モデル入力とモデルグリッドにおける特性の場所は調整されなければならない。たとえば以下の通りである。

- あるノードにおいてポンプ式井戸が位置してなかった場合、隣接するノードの井戸陥没の配置が調整されなければならない
- スラリー壁が代わりの補修の場合、壁の幾何学的水力学的特性のモデルの表示が調整されなければならない
- 掃除の時間が計算されている場合、汚染物質の場所、量、状態に関する全ての仮定は調整されなければならない
- 井戸摘出などの補修行動が流動に影響を与える場合、各モデル化された摘出井戸の沈滞点への下降傾斜と側面範囲を含めてそのような影響は決定されなければならない。

（7）ポストシミュレーション解析

63 サイトシナリオをシミュレートする際のモデルアプリケーションの成功は評価されなければならない。

64 このアセスメントは以下の解析を含めなければならない。

- モデル開発シミュレーションが現実的であったかどうか
- シミュレーションが正確にシナリオに反映したか
- 水文地質学的システムが正確にシミュレートされたか
- 概念モデルのどの見解が成功してモデル化されたか

65 モデルの感度は、サイト特定設計パラメータにおいて結果として不正確さが生じ、モデル較正におけるエラーのレベルは調査され量化されなければならない。例えば、モデル開発シナリオは、より感度の高い水文地質学的パラメータの可能な値の範囲にシミュレートされなければならない。さらに、モデル結果についての結論を出す際に、モデル較正におけるエラーの範囲が考察されなければならない。

66 ポスト処理は、それがモデル開発結果を正確に表わしているということを確証するために解析されなければならず、適切なところで補間やスムージング方法はドキュメントにされなければならない。

67 ポスト処理結果は、それらがモデル開発目標を支持するということを確実にするために解析されなければならない。

68 最終発表は、モデル開発結果を効果的に正確に伝達しなければならない。

69 実行できるときに、未来にモデルのポスト監査の実行または計画をしなければならない。

（8）全体的有効性

70 モデルアプリケーションにおいて遭遇したどのような困難もドキュメントにされなければならない。

71 モデルアプリケーションは管理側が意思決定のために捜し出した情報を提供しなければならない。

72 モデルアプリケーション結果は他の関連した団体にも受け入れられなければならない。

73 モデルアプリケーションはタイムリーで効果的な決定を支持しなければならない。

74　モデル開発労作の見解で、後学として別の方法で行われたかもしれない部分はドキュメントにされなければならない。

4．7　品質管理とコード選択

4．7．1　コード選択過程

モデルアプリケーション過程において、コード選択はプロジェクト労力と結果の間の最善のトレードオフを確実にする決定的な段階である。この結果は一般的に、予測正確性に関するモデル開発労作の期待された有効性と表現される。労力は最後にプロジェクトのコストによって表わされる。そのようなコストは、フィールドデータ取得のそれと独立して考察されてはならない。モデル開発コストの適当な評定のため、新しいコードの開発または存在するコードの取得、実行、メンテナンス、コードのアップデート、及びデータベースの開発とメンテナンスの測定が考察される必要がある。

コード選択は、本質的にはモデル開発必要性の詳細な説明と存在するコードの首尾よく定義付けられた特性との組み合わせのために、適切なコードを選択することは、存在するシミュレーションコードの体系的特性記述と同様に、モデル開発必要性の公式化を必要としている。モデル使用において、コード信頼性は主要な問題である。従って、十分なレビューとテスティングを受けた適確なシミュレーションコードの使用を確実にするため、選択過程においては特別な注意をしなければならない。モデル開発プロジェクトの成功のためには、適切なモデルの選択が決定的である。

4．7．2　モデル開発必要性の公式化

能率的にシミュレーションコードを選択するためには、そのようなモデルの評価と受け入れのため、管理者向けの規準を開発する必要がある。そのような規準を得るためには、着手すべき管理側の目標と捜し出された解析のレベル（関連している間違った、または不正確な解答、またはリスクに対するプロジェクトの感度に基づく）が把握されなければならない。さらに、研究されている物質的システムの適切な知識が存在しなければならず、研究に利用できる人材と物質的資源における制限の解析を完成させなければならない。選ばれた選択規準のセットは以下に基づかなければならない。

- コードの実行コスト（要求された解析レベルのデータ取得を含む）と正確性の間のトレードオフ
- モデルユーザーのプロフィールと要求されるユーザーフレンドリネスの定義
- 努力、コスト、制限に関するアクセス可能性
- 受け入れられる時空の尺度と集合のレベル

これらの考察に基づいて、ユーザーは研究されている問題に関連したコード特性を公式化し、存在するコードの特性を地図に入れるためにこれらの特性を優先しなければならない。コード特性は、その選択過程に集中されることがあるが、モデルの目的、表わされた地下水システムの特質、そして使用された数学的方法を含む。さらに、ユーザーはモデルの正確さ、ドキュメンテーションの完全性、そしてソフトウェアの品質、性能、及び使用性に関心があるかもしれない。ユーザーがコード選択において関連があると思うかもしれない他の情報は、それが実行するコンピュータシステムとその使用のための条件である。

4．7．3　コード特性の記述

地下水モデルには様々なタイプがあり、それらは様々なタイプの地下水システムをシミュレートするためにデザインされ、いろいろなシステム変数を計算することができる。ある特定モデルの主要な特性を確認するためには、標準化され定義されたモデル記述子を使用した記述的システムが必要である。地下水モデル開発コード記述子のデータベース開発の一部として（MARS：モデル注釈探索と検索）、国際地下水モデル開発センター（the International Ground Water Modeling Center）は何百もの段階別に分類された記述子を含む体系的なリストを開発した（van der Heijde and Williams, 1989）。この記述的システムのために採用された詳細のレベルは、ユーザーの関心のあるシミュレーションコードを確認し、準備的な使用性アセスメントを行うことを可能にする。

シミュレーションコードの記述は、モデルの目的、表わされた地下水システムの特質、及び使用された数学的方法の目的を網羅する。更に、それはモデルの正確さ、ドキュメンテーションの完全性、及びソフトウェアの品質、性能、使用性を指定する。記述的システムMARSに含まれている他の情報は、ソフトウェアが実行するコンピュータシステムとその使用のための条件に関連する。

最終選択が行われる前に、ユーザーはコードドキュメンテーションを入手し、コード特性を詳細に評定しなければならない。また、ユーザーは入手できる様々な数値方法の一般的性能特性を認識しなければならない。各方法はそれぞれ長所と短所があり、それは左右する方程式、適用可能な境界条件、そして関心のあるパラメータの散布の条件に含まれた種類と関連していることがある。

例えば、Kinzelbach（1987）は、主に4つの解法アプローチを表わす2次元トレーサー溶質運搬コードを評価した。彼は、離散の条件が満たされなかった場合、標準有限差分法と有限要素法は信頼できない結果をもたらすことを論証した。粒子のトラッキング方法、例えば、特性の方法やランダムウォーク方法は、振動と非混合の問題を示す。

選択されたモデルとされるべき計算の正確さが決定的な場合、明確にフィールドサイトにおいて遭遇すると期待される状況のタイプを指定して、検証テストのためにコードのドキュメンテーションや関連出版物を評価しなければならない。そのような検証はKhoury et al.（1989）に見られる。彼等は貯水池のデザインを解析するため、USGS MOCコードを使用した。貯水池のための不透水な基礎を自然に起こる粘土の層が供給しなければならない。しかしながら、粘土の層が殲滅する貯水池のある部分とダムの最大の高さに隣接した部分で、その粘土の層は欠如している。ダムの上流の場所は、密な粘土の脈を使用して補修されるべきである。貯水池からの漏出が、粘土の層の下にある限定された帯水層に過多のピエゾメーター的な土手をつくり、それがダムのデザインや谷間の安定性に衝撃を与え、他の問題を引き起こす可能性があるという懸念が続いた。コードの適用性と正確さを評定するために、単純化された不均整の幾何学による類似した問題の予測が解析解法（Khoury et al., 1989）と、独占的2次元有限要素コードによって得られた解と比較された。

4．7．4　コード選択規準

コードの選択における主な規準は以下の通りである。

- コードが意図された使用に適確であること
- コードに信頼性があること
- コードが効率的に適用できること

様々な問題が解決されなければならない場合には、１つ以上のコードが必要になるかもしれないし、あるいは１つのコードが１つ以上の能力に使用されるかもしれない。そのような場合、提出された各問題のモデル要件は選択過程の最初に明確に定義付けられなければならない。ある部分には、プロジェクトの様々な段階の同じシステムをモデル開発する場合においても事実である。コードは問題を広げたりシステム概念化を補助するためにプロジェクトの初期段階に選択されることがある。データ使用可能性における時間と資源の限定は、初期に公式化を「シンプル」なモデルにするかもしれない。システムとデータの使用可能性の理解の増加は、増大する複雑さや結果として生じる様々なコードの選択の成功を導くかもしれない。そのような場合、候補コードの柔軟性または様々なレベルの精巧さの統合されたコードセットの利用可能性が重要な選択規準になるかもしれない。

上記に述べられたように、コード選択における主な規準は、確認された作業の適確性、その信頼性、そして効率性である。コードの信頼性は、その開発中に適用された品質保証のレベル、その検証、フィールド妥当性検査、及び性能評価によって決定される。コードの信頼性は、広範囲に認められるレビューやテスティング手順を適用することによって確立されなければならない。

様々な組織が、それぞれ独自の手順を確立したにもかかわらず、そのような手順はまだ採用されていない（例えば、Wilkinson and Runkle, 1986; van der Heijde, 1987）。それに加えて、米国試験・材料協会（ASTM：the American Society for Testing and Materials）の小委員会D-18.21の地下水モデルのセクションにおいて、地下水シミュレーションコード選択の標準的ガイダンスを開発するため検討が始まっている。

コードの効率性は、ソースまたは実行時間バージョン、ドキュメンテーションの使用可能性、オペレーションに要求される人材やコンピュータ資源と関連した使用性、移植性及び修正可能性によって決定される。

シミュレーションコードの受け入れは、技術的・科学的確実性、効率性、法的そして管理上の考察によって基づかなければならない。モデルの効率性は、コードへの急速なアクセス、良質なドキュメンテーションがあること、ユーザーサポートへのアクセス、そしてその信用性、真実性、使用性、移植性、修正可能性、実行時間資源の利用状況、及び全体的な節約によって決定される。これらの規準のより詳しい説明が下記に述べられており、van der Heijde and Beljin（1988）によって、より広範囲な処理法の論述がある。後に述べた出版物には、これらの各規準の提案された評価システムも含まれている。

（１）利用可能性

総称的シミュレーションコードは、それに関連したプログラムコードがソースコードまたは実行可能なバージョンとして得られる場合、またはプログラムがポテンシャルユーザーによって簡単にアクセスできる場合に利用可能と定義付けられる。地下水ソフトウェアは、公衆領域もしくは所有権を持つステータスにある。米国においては、連邦や州の政府機関、あるいはそのような機関からの資金による大学によって開発されたほとんどのソフトウェアは、その使用と配布において無制限に利用可能であり、従ってそれは公衆領域に入っているとみなされる。他の国々においては状況が異なり、たとえ政府援助によって開発されても、ほとんどのソフトウェアは所有権を持つステータスにあるか、またはステータスが明確にされていない。そこでは、コンピュータコードは、その使用、複製、配布の

ある制限のもとで入手またはアクセスが可能になる。

コンサルタントや私営の企業によって開発されたコードは所有権を主張するものがある。これはいくつかの大学や私営の研究機関によって開発されたソフトウェアにおいても同様だと言えるかもしれない。一般的に、所有権を主張されたコードは、版権の法律によって保護されている。いくつかの総称モデルのソースコードがテキストなどの出版物に発行され、出版社からのテープやディスケットによって入手可能ではあるが、その使用と配布は発行の版権に制限されている。

コードが、数学的またはグラフィックのサブルーチンなど、所有権を主張する第3者ソフトウェアを含む場合、さらなる制限が生じる。公衆領域コードにおいては、そのようなルーチンは外部的であり、プログラムを成功して実行させるためにホストコンピュータでの存在が要求される。

公衆領域と所有権を主張するソフトウェアの間には、フリーウエアまたはユーザーサポートされたソフトウェアと呼ばれるものがある。フリーウエアは自由に複写し配布されてもよいが、ユーザーはこのようなソフトウェア開発を任意の寄付によって援助するよう奨励されている。地下水モデル開発においては、このようなタイプの配布方法は稀である。

公的資金によって開発されたいくつかのコードにおいては、例えばソフトウェアが輸出されたり、広範囲なメンテナンスとサポート設備が作成された場合、配布制限が実施されている。後に述べたケースでは、品質保証のされていないバージョンの使用防止、ソースコードの確認されていない修正を予防するため、及びコントロールされたユーザーグループに効率的なコードのアップデートをサポートするために、制限が施行されている。

（2）ユーザーサポート

サイト特定設計問題のために、あるコードが選択された場合、ユーザーは利用できるコンピュータシステムにおいてコードを実行する際に技術的問題に突き当たるかもしれない。そのような困難は以下の結果から生じる可能性がある。

- **コードの開発されたコンピュータとユーザのコンピュータの間の互換性の問題**
- **オリジナルのプログラムにおけるコーディングエラー**
- **データ入力やコード実行におけるユーザーのエラー**

コードにより精通することによって、ユーザーに関するエラーを減らすことができる。ここでユーザーに良質なドキュメンテーションが役立つのである。注意深いコード選択の後に、実現や実行に問題が起こり、ドキュメンテーションから答えが見い出せなかった場合、ユーザーはそのコードのオペレーションと使用に関して知識のある人の支援が必要となる。ソフトウェア実現と実行サポートまたはソフトウェア・ユーザーサポートと呼ばれるそのようなアシスタンスは、一般的にはコードの使用やモデル開発の適切なトレーニングの必要性にとって代わることはできない。コード開発者からの補助のリクエストは、ユーザーサポートがコンサルタントサービスまたは職場内訓練になってしまうほどの広範囲な割合とみなされるかもしれない。この可能性は一般的にはコード開発者によって認知されているが、ソフトウェアのユーザーにはいつもそうとは限らない。

（3）真実性

地下水シミュレーションコードの使用における主要な問題はその真実性にある。コードの真実性、そしてそれが表わす理論的構造は、その証明された信頼性とユーザーの受け入れに基づかれている（成功したアプリケーションの数の一部に基づいている）。モデラーや地下水管理者は、その頻繁に適用されるシミュレーションコードに確信を持つことがある。この見解は、よくレビューされた成功したアプリケーションが発行されたときに強化される。

プログラムの信頼性は、ソフトウェアエラーによって起こる可能性のあるローカライズされたエラーや端末失敗と関連しているため（Yourdon and Constantine, 1979）、そのような本来は広く使用されているプログラムにもあるエラーのほとんどは見抜かれ、訂正されているとみなされている。長期間使用とアップデートの歴史にもかかわらず、依然としてプログラミングエラーの無いコンピュータプログラムは存在しない。いくつかのエラーは絶対に見抜かれることは無く、プログラムのユーティリティーを影響しないか、またはほんの少しの影響しか与えない。他のエラーは例外的状況またはテストされていない状況においてのみ発生する。

シミュレーション結果に基づいた決定は、使用されたコードが適切なレビューとテスティングを行われているときのみ実行できる。しかしながら、フィールド妥当性検査（もし存在するならば）やコードアプリケーションの感度に頼りすぎるのは、いくつかの科学的に確かな、首尾よくドキュメントにされた、徹底的にテストされたコード、そして手元にある最も問題解決に効率的なものさえも除外することになるかもしれない。

コードが多人数によって使用された場合、これが重要なユーザーの確信の証拠となるということが論じられるかもしれない。そのような広範囲な使用は様々なタイプの地下水システムへのコードの適用性や様々な管理側の問題に着手する能力を反映させることがある。それはコードが比較的に使用しやすいということを意味するかもしれない。最後に、もしコードが大きなユーザーの基盤を持つ場合、知識豊富な人間と特定のアプリケーションについて討論する機会が多くなる。

（4）使用性

シミュレーションコードがユーザーのコンピュータシステムに実現された場合、様々な問題に直面する。そのような困難は、ハードウェアの不適合性や、コードのインストール、データ入力、またはプログラム実行などのユーザーのエラーのコーディングによって発生するかもしれない。オペレーション上の理解と知識を実行するプログラムで使用が簡単なものは、ユーザーフレンドリーと呼ばれ、その使用性によって定義される（van der Heijde and Beljin, 1988）。そのようなプログラムでは、一般的に広範囲なよく編集されたドキュメンテーションが強調されている。簡単な入力準備とコード実行で、オンライン補助システムのある（インテリジェントな）プリプロセッサの使用が望ましい。そしてよく構成された有益な出力。適切なコードサポートとメンテナンスはコードの使用性も強める。

（5）移植可能性

ひとつの実行環境から他へ簡単に転送できるプログラムは移植可能プログラムと呼ばれる。プログラムの移植可能性を評価するためには、ソフトウェアとハードウェアの両方の従属性が考慮されなければならない。新しいコンピュータ環境でプログラムを実行することが必要な場合、その修正可能性

が重要である。

（6）修正可能性

コンピュータプログラムの有益な寿命の間に、ユーザーの経験と管理側の変化する要件はソフトウェアの機能上の詳記の変化に導かれることがある。それに加えて、科学的開発、変化する演算環境、及びエラーの持続がプログラムの修正を必要にする。ソフトウェアが、ある一定期間以上使用される場合は、そのような成り行きにペースを保てるように継続して修正できるようデザインされなければならない。修正が難しいコードは、壊れやすく維持能力が欠如しているものと呼ばれる。その様な困難は、地域的変化のグローバルな広範囲プログラムの実現によって生じる可能性がある（van Tassel, 1978）。

（7）ランタイム資源効用性

ランタイム資源効用性とは、モデル開発計算を実行するために必要とされている計算資源を指す。主な資源は、要求されるハードウェア装置やシステムソフトウェアの種類や数、コンピュータコアメモリ（RAM）使用、プログラム実行（CPU）時間、入出力処理（I/O）時間、そして質量蓄積要件を含んでいる。コンピュータコアメモリの使用は、プログラマーによってコードの中で前もって決定される場合がある。他の場合においては、ユーザーはセットアップされた問題を特定することによって、コアの使用を間接的に決定する。質量貯蔵スペースの需要とともに、プログラム実行とI/O時間は、代表的ベンチマークのひと組みを実行させることによって測定が可能である。

4．7．5　コード選択における品質管理

要求された特性と利用できる総称モデルに存在する特性の間には完璧な組み合わせは稀であるということを認識しなければならない。ある特定のアプリケーションに重要と判断された選択規準は主観的もしくは調整が弱いこともあり、それはモデル化されるべきシステムのある特性の重要性を確立するためのプロジェクト内の選択段階におけるデータが不十分であるからである。組み合わせの入手が困難な場合、これらの規準と選択過程における、それらの相対的比重の再アセスメントが必要である。このために、コード選択はまったく巡回過程であり、成功は熟練した判断によるのである。

コード選択における品質保証は、コード開発やモデルアプリケーションの品質管理の見解において説明されたこととと同様のパターンに従う。コード選択品質管理における主要な要素は以下の通りである。

- 上記に説明されたように注意深く一歩一歩の選択手順に従うこと
- この手順に執着した詳細な報告
- 各段階に行われた考察、仮説、そして決定のドキュメンテーションである

明確には、研究されている問題のために選択されたコードが、なぜ適切であるかということが詳細な調整として示されなければならず、研究されている問題に関連した選択コードの制限が説明されなければならない。

地下水モデル選択に関する情報は（Rao et al. 1981; Kincaid et al. 1984; Boutwell et al. 1985; Simmons

and Cole 1985）に発表されている。

利用できる地下水モデルの概略は、van der Heijde and Beljin（1988）、van der Heijde et al.（1988）及びvan der Heijde and Elnawawy（1992）によって提供されている。

４．８　品質管理の終結

文字どおりに言えば品質保証とは、商品（コード、モデル）または関連している作業（モデル開発）の品質を保証することである。より活用できる記述は、品質管理（モデル開発において）は、モデル基盤の解析やアドバイス（意思決定者に対して）の品質が定量的品質規準や測定を満たすことを保証する。品質管理に隠された主要な考えは責務であり、主要なメカニズムは全ての作業や結果の記録の維持（ハードコピーと電子ファイル、レポート）であり、より適切な用語は品質ドキュメンテーションである。それは、広義で、品質管理は我々の利用できる技術に対する現在の理解の範囲内において最善を尽くすための方法論的で管理上の構造を与える。

品質管理が、コード開発プロジェクトまたはモデル開発研究の受け入れられる品質を常に確証するということは無駄な期待である。しかしながら、品質管理は不完全なコードや不適当なモデル開発に対する予防手段を与えることはできる。モデル開発に基づいたアドバイスが完全に正しいということを保証する方法は何も無く、使用されたシミュレーションコード（または、それについてはどのような科学的モデルでも）が、これらの用語の最も正確な意味においても、証明、検証、妥当にできないということを調整者や意思決定者は理解しなければならない。むしろ、モデルは本物のシステムに関する独立して得られた予測の不一致によってのみ不当にできる。

品質管理の主要な役割は、モデラーとその同僚の間、そしてモデラーと意思決定者の間のコミュニケーションを提供することであり、後者にはモデラーのアドバイスの正確性、不確実性、及び信頼性の意味を与えるということを注目しなければならない。従って、品質管理は下級のモデラーの仕事にのみ適用してはならず、熟練したモデラーにも執着しなければならない。

様々な注意されるべき点がある。経験者のモデラーが以前には探究されていない新しい手段を取ることを思いとどまらせたり、プロジェクト予算の大部分が官僚の要求にこたえるために不適切に消費されてしまうほど、品質管理は息苦しくなってはならない。品質管理規則が官僚の形式主義になってしまうと、品質管理の時間とコストは、データ収集や問題解析作業から貴重な資源を取り去るかもしれない。更に品質管理が堕落し、モデル開発結果の誤った自信をインストールするただのチェックリストになるというリスクが存在する。

４．８．１　序論

地下水モデルに基づいた予測可能なシミュレーションの有益さは、そのようなモデル結果の信頼性を表示し、量を測ることができないことで限定されることがある。研究者たちは、水資源管理者が意思決定過程における不確実さを説明できるように、モデル予測の確信レベルを評定するための様々な技法を開発してきた。例えば、Freeze et al.（1990）は、水文地質学的環境における工学デザインの決定分析のアプリケーションに基づいた方法論を発表している。その方法論は、地下水流動と汚染物質輸送のシミュレーションモデルであるリスク費用対効果の目的関数に基づいた決定モデルと、地質学的不確実性とパラメータの不確実性を包含する不確定モデルの結合を含む。

重要な分野の一つは、コードの使用したシミュレーションの真実性と、それらが表わす総称モデルである。前セクションで説明したように、シミュレーションコードの真実性における重要な面は、その信頼性である。コードの信頼性は、包括的で体系的なレビューとテスティング過程を適用することによって確立される。次のセクションは、解析解法とベンチマークとしての統合的データセットの使用に基づいた体系的なコード検証と性能テスティングプロトコルについて述べる。報告はいくつかの実例テスト問題を提供するが、実際のベンチマークは含まない。

4．8．2　文献レビュー

地下水システムのコンピュータに基づくシミュレーションの初期段階に始まって、コードの検証はコード開発活動の一部であった（例えば、Pinder and Bredehoeft, 1968; Prickett and Lonnquist, 1971; Pinder and Frind 1972）。最近では、コード開発者によるそのようなコード検証は、かなり精巧なものとなってきた（Ward et al., 1984; Reeves et al., 1986; Gupta et al., 1987; Sims et al., 1989; Faust et al., 1990; Zheng, 1990）。

いくつかの場合においては、コード検証とそのコードの性能評価は、コード開発チームとは別のテストチームによって実行されてきた（例えば、Beljin, 1988; Watson and Brown, 1985）。そのような第三者によるテスティングは、ある特定の取り締まられた構造内でコードが使用するのに受け入れられるかどうかを決定するため、取り締まり機関よって着手されることがある。

地下水シミュレーションコードの包括的な検証と妥当性検査の必要性は、一連の国際協力プログラムの成功によって明らかにされている。それらは、INTRACOIN、HYDROCOIN及びINTRAVALである。これらの研究は、放射性廃棄物のための処理場の性能アセスメントに関連して、地下水流動と放射性核種輸送の概念モデルと数学モデルの評価を試みた（Nicholson et al., 1987; INTRAVAL, 1990）。これらは、スウェーデン原子力検閲団（the Swedish Nuclear Power Inspectorate）（SKI）によってコーディネートされた様々な国から参加しているモデル開発グループによって実行された。個々のチームは、それぞれの国の政府機関によって支援された。

最初のプロジェクトである国際核種輸送コード相互比較研究（the International Nuclide Transport Code Intercomparison：INTRACOIN）は1981年から1986年まで及んだ。これは水文学的コード相互比較研究（the Hydrologic Code Intercomparison：HYDROCOIN）によって継承され、1990年に完成した。第3のプロジェクトである、地圏輸送モデルの妥当性検査を研究するための国際プロジェクト（International Project to Study Validation of Geosphere Transport Models：INTRAVAL）は、1987年に創始され現在も活動中である。これらの研究は、コンピュータコードの数値の正確さ、根底にある概念モデルの妥当性、そして感度と不確定解析のための様々な技法をテストした。

INTRACOIN研究は、地質学的媒体における放射性核種の輸送を説明する22の異なる計算コードの比較から成り立っていた（INTRACOIN, 1984）。この比較は以下を確立することを目的とした3段階で実行された。

- コードの数値の正確さ。
- コードのもとの場所の測定を説明する能力。
- 典型的な処理場シナリオアセスメントにおける核種輸送計算の、様々な物質的現象の定量的影響である。

この報告の第２節で定義された技術用語を使用して、INTRACOIN第１レベルはシミュレーションコードの検証、第２レベルは総称モデルの妥当性検査、そして第３レベルはフィールドパラメータや境界ストレスにおける摂動のためのコードの感度解析を通した性能評価に関係している。

INTERCOIN第１レベルでは、研究者は存在する解析解法と数値モデルからの計算結果を比較、または様々な数値方法を利用するコードからの計算結果の相互比較によってコードを検証した（INTRACOIN, 1984）。この研究は、１と２次元ファーフィールド放射性核種地圏輸送モデルに限定された。いくつかの関係しているコードは３次元問題を扱うことができたが、「そのような問題を扱える関係したコードがほとんど無かったため」、研究は３次元テスト事例を含まなかった（INTRACOIN, 1984, p. 11）。７つのテスト問題は、一定のパラメータをもつ浸透性の媒体における単純な１次元輸送から、岩石マトリックスへの拡散をもつ断裂した媒体における２次元輸送の範囲に及んだ。

- 1次元移流分散、一定移動パラメータ（すなわち、地下水速度、保存因数や分散性は一定に保たれる）及び一定の浸出率。
- 層にした媒体における1次元移流分散（区分的一定移動パラメータ）。
- 連続して変動するパラメータをもつ1次元移流分散。
- 一定の保存因数と分散性をもつ２次元移流分散。
  平行流動フィールドと放射状の分散。
  ２次元流動フィールドと放射状の分散（２つの井戸の間の流動フィールド）。
- 岩石マトリックスへの分散をもつ1次元移流分散。
- マトリックス分散がある２次元移流分散。
  平行流動フィールドと放射状の分散。
  ２次元流動フィールドと放射状の分散。
- 線形の集団転送キネティックスと一定移動パラメータをもつ1次元移流分散。

含まれていた22のコードは５つのグループに分割された。

1）1次元移流分散モデル
2）区分的一定パラメータをもつ1次元移流分散モデル
3）マトリックス分散を含むモデル
4）非線形化学影響を含むモデル
5）水文学と熱輸送の方程式を含む結合されたモデル

である。モデルの５つ目のグループは、INTRACOIN研究のテスト目的には含まれていなかった水文学と熱輸送の方程式を入れた。また、地下水流動のモデル、生物圏や線量計算における輸送、及び「フィールド近く」のモデルも含まれなかった（INTRACOIN, 1984）。一般的に、各コードは選択された問題のみに実行された。最初の問題のみがほとんど全てのテストチームに使用された。コード比較は３つの規準を使用して実行された。

- 時間の関数としての移動パスの最後にある核種密集。
- 最大密集、最大量の回数、密集が最大密集の半分である場合の回数。
- 実行時間（CPU時間）とコンピュータ設置での１つの精度浮動小数点数乗算にかかる時間。

比較の結果は表とグラフの形によって表わされた。様々なINTRACOIN第1レベルテスト問題の結果は、関係したコードの間で概してよく一致した（INTRACOIN, 1984）。結果の変動は様々な原因に帰している。いくつかのコードは、主に移流の、または分散性の問題を扱うために最初にデザインされたアルゴリズムを使用し、従って全てのパラメータ値において無条件に確実な結果を出すことは期待できない。低いペクレ数（Pe<10）を使用することによって、この問題の繊細な相互比較を供給する。しかしながら、それた結果のほとんどは、境界条件の履行における不一致または入力データの履行における不一致によって説明されるか、打切り誤差（空間または時間の領域における粗野な離散化）の結果を考察した。

この研究は、モデル領域の離散化を変化させる、収束テスト研究と結合された慎重にデザインされたベンチマークが、数値コードの品質保証の重大な一因となるという結論に達した。明確には、この研究は品質保証過程における「最大限の重要さ」の精密でよくプログラムされた解析解法の開発を考察した。今後のベンチマーク研究は、堅実にデザインされたテスト事例を使用し、以下を含めて実行するよう推奨された。

- 境界条件とその数値履行の説得力のある定義。
- 連続して変化する属性の離散化の説得力のある定義。
- 比較において最適の感度を与えるほどの選択されたパラメータ値。

さらに、そのような研究においては、正確性の測定が定義されることも必要である。離散の区分のランダムウォークアルゴリズムまたは粒子トラッキングに基づいたコードは、信頼できる結果をもたらすために、ユーザーから「近づきやすいフィーリング」が必要らしい事が注目された。

INTRACOIN（1986）は、もとの場所の測定（INTRACOIN第2レベル）や感度解析（INTRACOIN第3レベル）活動を使用して妥当性検査の結果を表わす。もとの場所の測定を説明するコードの能力は、地質学的媒体における物理化学的放射性核種輸送の様々な概念と、実験所のデータによって補充されたフィールド試験を比較することによって評価された。モデル計算との比較のために2つの慎重に選択されたフィールド実験は、1つは断裂の媒体から、もう1つは浸透性の媒体から選ばれた。INTRACOIN第2レベルのテスティングに関係した研究は、一般的には関係のあるコードは、フィールド実験結果を適度によく再生することができたが、十分な妥当性検査のレベルが期待されているならば、より良い詳細な実験が早急に必要であるという結論に達した（INTRACOIN, 1986）。これに関して、今後の妥当性検査サイトが不均質や流動水路に関して特性を明らかにすることが極めて重大であるとみなされた。この報告は、フィールド実験は実験状況を解釈する際の自由度を減らすための妥当性検査を目的として直接デザインされることを推奨した。

水文学的コード相互比較研究（the Hydrologic Code Intercomparison：HYDROCOIN）は、INTRACOIN研究に対する流動向けの補充として設立された。1984年に着手されたこのプロジェクトは、地下水流動をシミュレートするために使用される解析と数値方法に集中した（HYDROCOIN, 1988）。HYDROCOINの目的は、「核廃棄物の最終処理場の安全性アセスメントのために、地下水モデル開発のための様々な戦略の影響について改善された知識を保つこと」である。この研究は3つの問題を評定することを目標とした。

・地下水流動計算の異なる解法アルゴリズムの影響。
・様々なモデルのフィールドと実験室の実験を説明する能力。
・地下水流動計算における様々な物質的過程を結合させた際の影響である。

HYDROCOINは、INTRACOIN研究のために公式化されたようなコード評価と同様の３つのレベルを使用した。HYDROCOIN第１レベルでは、コードの数値の正確さが計算結果と解析解法の比較、またはINTRACOINについて説明した前述のものと同じ方法でコードの相互比較をすることによって決定された。７つの仮説的テスト事例が展開された。

・単一の断裂を含む浸透性のある媒体における堀削孔からの過渡流動（解析解法）。
・互いを交差する２つの浸透性の断裂ゾーンを含む２次元領域における安定状態の流動。
・高い浸透性と低い浸透性の水成岩が代わり合う一連を通した部分的に飽和した流動。
・衰退する熱出力をもつ球状のソースから熱が放出されるシステムのための熱伝達（解析解法）。
・流体の濃度が塩の濃縮によって決定される、２次元領域における流体の流れと塩の輸送。
・総称的な層にされた塩の地質学的セッティングを表わす、（弱い）３次元領域における安定状態の流動。
・粘土質の媒体の浅い土地を埋めた サイトを通した安定状態の流動。

テスト事例は、現実的な廃棄物処理場シナリオを表わすよりも、むしろ主としてモデルの感度テストをもたらすためにデザインされた。これらのテストシナリオに29のシミュレーションコードが提示されている。比較のために使用された計算済みの実体は、地下水圧力、塩の濃縮と温度、及び流動速度と弾道の分布を含んだ。HYDROCOIN（1988）報告は、４つの線形のテスト事例（事例1、2、6及び7）において計算された主要な実体の間の一致は十分であったと結論を出した。

観察された偏差は、時間と空間における様々な離散化の密度の使用、または様々な数値の特性をもつ要素タイプの使用の結果によって説明された。しかしながら、計算された速度フィールドと弾道の間の一致は、スカラー実体よりも確信が低かった。これは主に、様々なチームによって使用されたポスト処理アルゴリズムにおける違いの産物とみなされたため、そのようなアルゴリズムの選択は、追加の精査に委ねられなければならない。

非線形の事例（事例４）の解決は簡単で、計算された主要なスカラー実体との間の一致が達成された。２つの大胆な非線形の事例（事例3と5）、明確には、事例3は、指定された浸透性の対照は湿気の容積が領域内で不連続に終わったことでシミュレーションの中の多くの問題を示した。最後に、この報告はテスト結果が空間と時間における適切な離散化にかなり委ねられていると結論を出した。HYDROCOIN研究の参加者たちは、正しく適用された際の有限差分と有限要素の結果から明確な違いを見ることはできなかった。有限差分ソルバの方が有限要素ソルバより実行するのための費用がかからないが、有限要素を使用して複雑な幾何学的領域を正確にモデル化する方が簡単であると指摘されている。米国核規制委員会（the U.S. Nuclear Regulatory Commission）（1988）によって、いくつかのコードの実際のテスティングのより詳細な説明が提供されている。後述の報告は、とりわけ異なるテスト事例、グリッドデザイン、反復規準の選択、及び境界条件と特異点の履行のためのモデルのセットアップに関する詳細を提供する。

HYDROCOIN第２レベルのテスティングでは、関係しているモデルがフィールド測定や実験室で

の実験を説明する能力が評価された（HYDROCOIN 1990）。この局面の始めに、関連した実験からのデータセットの広範囲な捜索が着手された。理想的なデータセットにおいては、データの解釈の自由度を減らすために、全てのモデルパラメータの測定された値は存在しなければならない。しかしながら、そのような理想的なデータセットは見つからなかった。選択された５つの実験とフィールドサイトは、多種の時空スケール、いくつかの重要な過程、そして多種の重要な媒体を表わす。これらの５つの事例は、22の関係しているコードを使用して解析された。

事例研究を実行するに当たって、参加している研究チームは地下水システムの複雑性に関連した問題に遭遇し、それは適切な水文学的過程または水文地質学的構造の代わりの公式化や流動パラメータ同次を表示させる。さらに、研究チームは実験データの解釈に困難を要した。明確には、参加しているモデル開発チームの間で境界条件の解釈と公式化において主な違いが現われたことである。これらの困難にもかかわらず、モデルは結局は５つのテスト事例の実験結果をシミュレートするよう「適合」された。いくつかの概念モデルとパラメータセットが比較できる結果をもたらし、実験に基づく発見やモデル開発結果の間の偏差について最もらしい解釈を見つけた。

1987年に第３のプロジェクトである、地圏輸送モデルの妥当性検査を研究するための国際プロジェクト（International Project to Study Validation of Geosphere Transport Models：INTRAVAL）が始まった。このプロジェクトの焦点はモデル概念の妥当性である。それはモデル妥当性検査過程を解析的に研究するため、実験室とフィールド実験の結果や自然な類似の研究の結果を使用する（INTRAVAL, 1990）。

他のベンマーキング、妥当性検査、及び検証研究が実行されてきたが、INTRACOIN、HYDROCOIN及びINTRAVAL研究で使用されるほどの体系的なものではない。Potter et al.（1987）は、同次な媒体を通して変動しやすく飽和した流動のための数値モデルの検証研究を行なった。彼等は異なる数値方法を用いて、６つの流動モデルの安定状態の性能を調査した。含まれた方法は以下の通りである。

- **有限差分単一公式化。**
- **有限差分ダブル公式化。**
- **線形ベースを用いた有限要素単一公式化。**
- **線形ベースを用いた有限要素ダブル公式化。**
- **エルミート行列ベースを用いた連続した有限要素公式化。**
- **エルミート行列ベースを用いたダルシーの連続有限要素公式化。**

これらの数値モデルは、変動して飽和する同次な土のコラムを通る流動のための、リチャードの方程式の新しく開発された２つの安定状態にある解析解法と比較された。著者たちは、ダブル公式化有限差分と有限要素方法の正確さが、単一公式化モデルのものよりも高いことを発見した。また、有限差分と有限要素結果を比較して、有限差分方法の方がより正確であることが分かった。ダルシーの連続有限要素モデルは、連続したモデルやその他のモデルよりも重要な増加を示した。この研究は、特定システムの物質的特質の正確さ、複雑性、パラメータ概算及び知識を考察して、探究された事例については、有限差分法を使用することが最も論理的な選択であるかもしれないと結論付けた。著者たちによると、「有限差分概念は、有限要素過程よりも単純で率直、そしてより広い聴衆に理解されて

いる」（Potter et al., 1987, p. 442）。

2つの研究が、通気帯ゾーンにおける殺虫剤の'死'をシミュレートするためにデザインされた、コンピュータコード化されたモデルを評価した。

Watson and Brown（1985）は、SESOILモデル（Bonazountas and Wagner, 1984）の包括的な評価を行ない、Carsel et al.（1986）は、PRZMモデルを評価した（Carsel et al., 1984）。これらの研究は、Donigian and Rao（1986）によって説明されており、不飽和ゾーンの流動と溶質輸送モデルのテスティングと評価を説明している。この報告における定義に関して、Watson and Brownの研究は、妥当性検査と検証の両方を含んでいる。彼等は、土のコラムから地下水のアルディカーブの浸出についてのSESOILモデル予測とJones et al.（1983）による予測の比較を通して検証を行ない、他の3つのモデルを使用した。PRZM（Enfield et al., 1982）、PESTAN及びPISTON（Donigian and Rao, 1986, p. 104）である。

検証は、SESOIL結果と解析解法からの、それとの比較も含んだ。また、妥当性検査はSESOILモデル予測と関連する公表されたフィールドデータの比較を含んだ。Carsel et al.（1986）による研究は、PRZMモデル結果と利用できるフィールドデータの比較によって行なわれる（部分的な）妥当性検査として説明することができる。Donigian and Rao（1986）が指摘するように、Carsel et al.（1986）の研究は、モデル評価のために限定されたのではなく、むしろモニター用のデータとして収集されたフィールドデータを使用したモデル性能テスティングで遭遇する問題の良い例である。これらのフィールドデータセットは、モデル評価のために重要である特定のタイプのデータが不足していることがある。また、多くのフィールドテストが短期のものである。妥当性検査に関しては、Donigian（1983）は観察データが収集された際にモデルが実際のフィールド条件に可能な限り近づいて実行される場合に条件をつくり、フィールド条件を表わす際のモデル予測や制限を感知することを推奨している。彼は、意思決定を補助するツールとしてモデルを使用するためには、モデルテスティングは「通知された疑い」と質問的なアプローチで実行されなければならないと提案している。

Hecox et al.（1989）によって行なわれた研究は、疑似3次元流動と飽和輸送モデルを較正し、検証した。このモデルは内部的にリンクされた流動と輸送コードである、PLASM（Prickett and Lonnquist, 1971）とRANDOM WALK（Prickett et al., 1981）から成り立っていた。この研究は流動と輸送メカニズムの理解を評定するため、修正システムのデザインを補助するため、そして未来の水柱移動方向と密集を予測するためにモデルを使用することを計画した。

流動検証は、周期的注入水位変動の実際のものとシミュレートされたものに加えて、帯水層ポンプ式テスト水位降下の実際のものとシミュレートされたものの比較を含んだ。溶質輸送検証は、1つのわずかに遅延した混合物と2つの異なった遅延した混合物の溶解した前部の移動率と密集分布の実際のものと観察されたものから成り立っていた（Hecox et al., 1989）。Hecoxの報告が説明するように、水位降下の流動検証は3週間のポンピングテストを通して達成され、周期的注入水位変動の流動検証は放水をシミュレートすることによって達成された。その結果は、各流動検証の観察されたレベルにかなり接近しており、違いは合理的に説明された。溶質輸送検証のための実行は、3つの異なる水柱を使用して行われた。溶質輸送のための較正は、遅延の要因にのみ焦点をおいた。報告は、シミュレーションが分散性または有孔性の較正無しで観察された水柱で多くの一致を見せたため、モデルは検証されたと論じている。

Kinzelbach（1987）は、空間と時間において等しい離散化を使用して、代表的な４次元コード履行をいくつかのテスト事例に適用することによって、４つの主な数値解法方法を比較した。選択されたコードは、以下のアプローチを表わす。

- 空間の中心または風上の違いをもつ明確な有限差分公式化。
- 時間が異なるフォワードの三角状の要素と直線補間をもつ有限要素公式化。
- 特質の方法（MOC）。
- Ito-Fokker-Planck理論に従って、完全な速度の用語を加えるオプションを持つランダムウォーク方法。

テスト事例（図４．８－１）は、数値解法技法の１つまたはそれ以上を持つ期待された数値問題を基礎として選択された。最初のテスト事例は、縦方向に0.5のペクレ数をもつ軸に平行な流動における不変の溶質ソースと、クーラン規準に従うタイムステップの特徴を描いた。明確な有限差分モデルとMOCモデルにおいて、タイムステップはNeumann安定性規準に従うためにも選択された。テスト事例２は、2より小さいペクレ数に終わる分散の値を使用した。３番目のテスト事例では、流動フィールドは座標の軸に45° 回転され、高い分散性の異方性が選択された。最後に、テスト事例４と５はソースの1000m下流に１つの井戸を持つ不変のソースを扱っている。事例４と５の違いは、分散性の値の選択である。結果はグラフの形で表わされ、密集対縦方向または横軸の距離、空間における輪郭化された密集分布、そして密集対時間を描いたブレークスルー曲線を使用した。

1980年代の半ばに国際地下水モデル開発センター（the International Ground Water Modeling Center）は、３段階の地下水シミュレーションコードテスティング過程を開発した。その過程と方法論は、van der Heijde et al.（1985）によって発表され、Huyakorn et al.（1984a）とBeljin（1988）によって２次元流動と溶解輸送コードに適用されている。最初のレベルでは、コードは図式化され単純化されたシステムに対してテスティングすることによって検証されている（IGWMCレベル1）。２番目のレベルは、クローズした形の解析解法が存在しない問題に対処するために、特別にデザインされたコードの特徴をテストすることを目標としている。ここでは、特定のモデルタイプまたはコード特徴をテストするためにデザインされた、仮説的もしくはかなり単純化された現実世界システムを表わす複雑な統合的データセットを使用して検証を試みている。（例えば、特異点の付近［井戸］とパラメータフィールドの断絶［流動障壁、水文地質学における突然の変化］）。第３レベルでは、モデル（とそのコード）は別個に得られたフィールドまたは実験室データに対して検証される（IGWMCレベル3）。

Huyakorn et al.（1984a）の研究は、IGWMCテスティングアプローチを２次元有限要素コードSEFTRAN（Huyakorn et al., 1984b）に適用した。第１レベルテスティングでは６つの解析解法が使用された。ガレルキン公式化とそのコーデイングのいくつかの異なった特徴の検証を可能にするため問題が選択された。

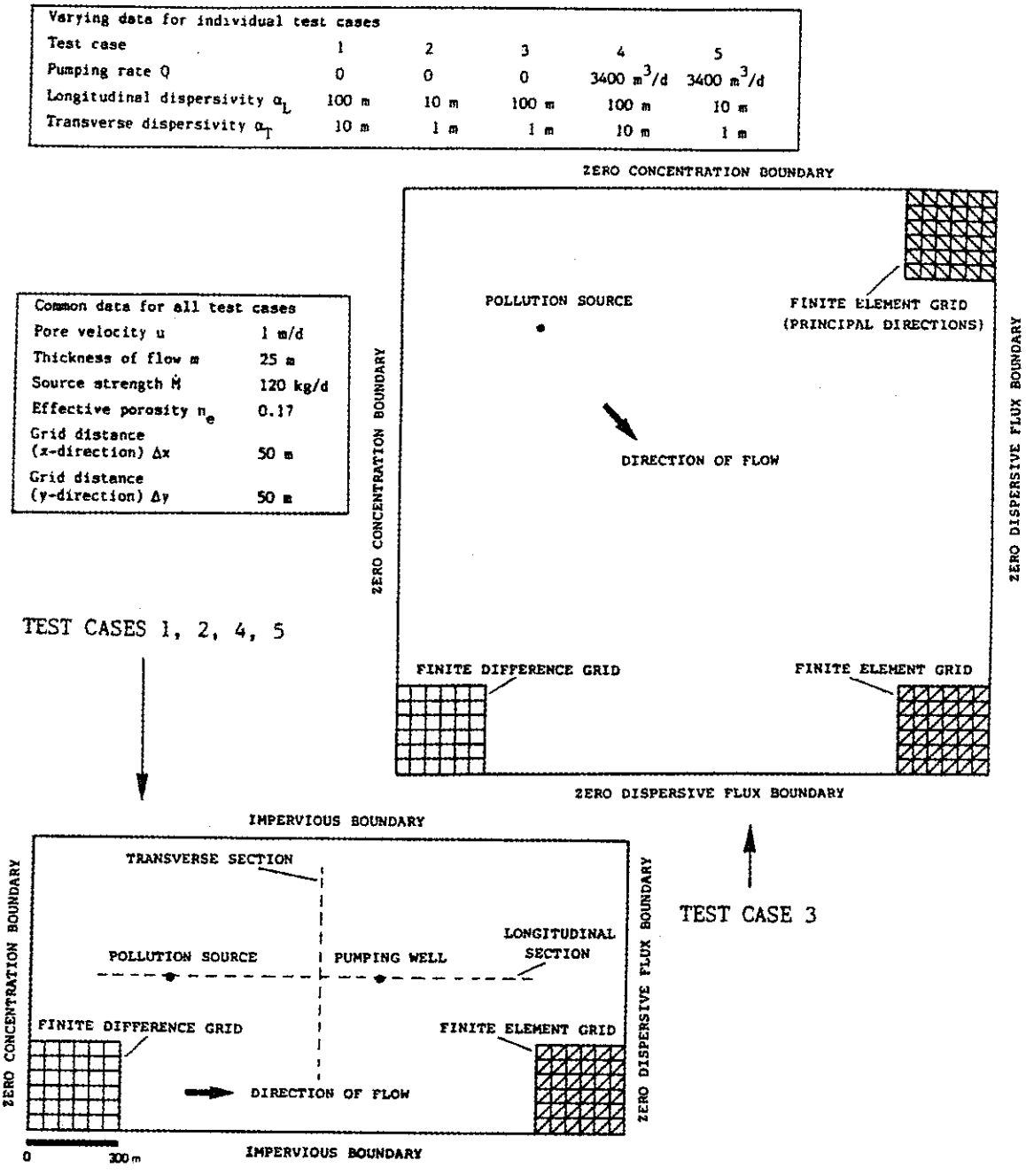

図４．８－１　２次元溶質輸送モデルのテスト問題（Kinzelbachより、1987）

各問題に対して、様々なポテンシャルなアプリケーション条件におけるコードの数値反応を評価するため、流動と輸送パラメータの現実的な値の範囲が選択された。６つの問題は、一定ではない流動フィールドの輸送に関連した単純な１次元的問題から比較的に複雑な２次元的問題の範囲を考察した。

・ファーストタイプ（ディリクレ）入り口境界条件をもつ半無限のコラムにおける遅延する溶質の輸送。

・サードタイプ（コーシー）入り口境界条件をもつ半無限のコラムにおける遅延し

衰退する溶質の輸送。

・一定地下水流動フィールドにおける連続ポイントソースからの輸送。

・一定地下水流動フィールドにおけるポイントソースから解放されたスラッグの輸送。

・注入井戸によって発生した放射状の流動フィールドにおける輸送。

・注入-流出井戸の組によって発生した一定ではない流動フィールドにおける輸送。

各問題について、報告は詳細な問題記述、テスト目的、入力指定、時空の離散化とシミュレーション結果を提供している。期待されたように、コードは高いペクレ数（図４．８－１）、粗いグリッドの離散化（図４．８－２）、そして小さい分散性の値をもつ１組（注入-流出井戸の組）をもつシミュレーションから注目に値する数値問題を見せた。

コードの第２レベルテスティングでは、２つの仮説的フィールド状況が使用された。これらの問題は、異質的で異方的なシステムに加えて、変則的な幾何学と複雑な境界条件を含んだ。最初の状況は、埋立地の下に位置する無制限の帯水層システムへの汚染物質輸送の断面図解析を含む。２番目の状況は、処分する穴からポンプ式井戸への汚染物質輸送の地域解析を含む。テスト目的は以下を含む。

・帯水層属性の変則的な幾何学と帯状の変動をコードが供給する能力があるかどうか決定するため。

・連合した、時間に依存した流動と輸送問題をコードが扱う能力を持つかどうか評価するため。

・コードが複雑な幾何学の地域を表現できるような効率性を評定するため。

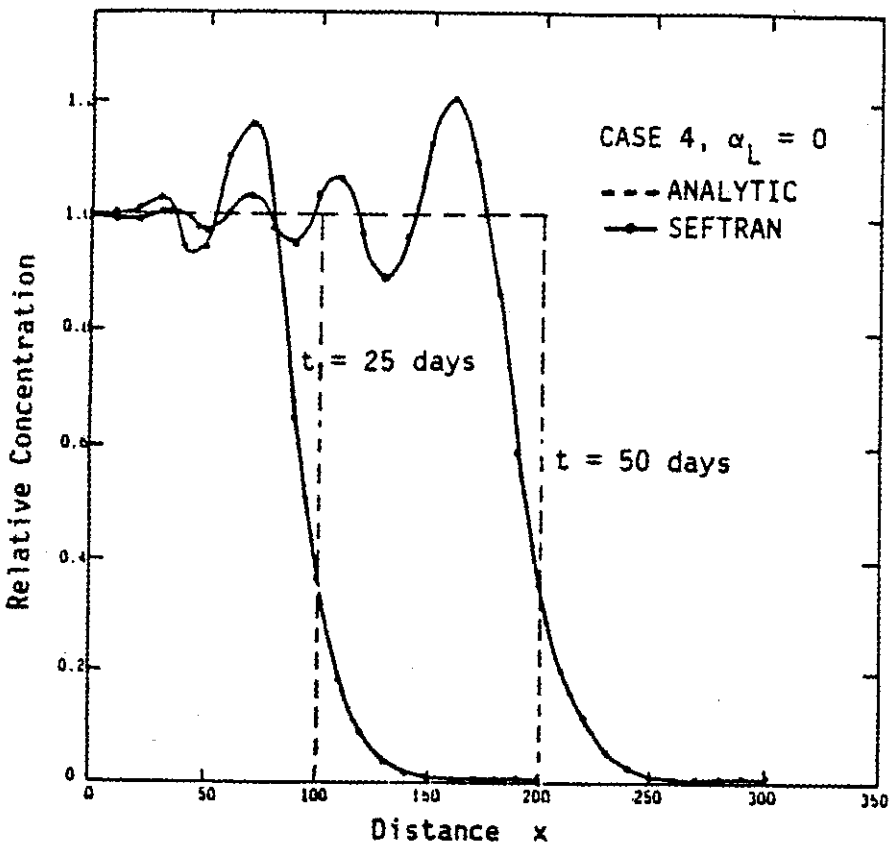

図４．８－２　問題1.1の解析解法と数値解法の比較、
事例4で Pe=∞（すなわち、純粋な移流）（Huyakorn et al., 1984aより）

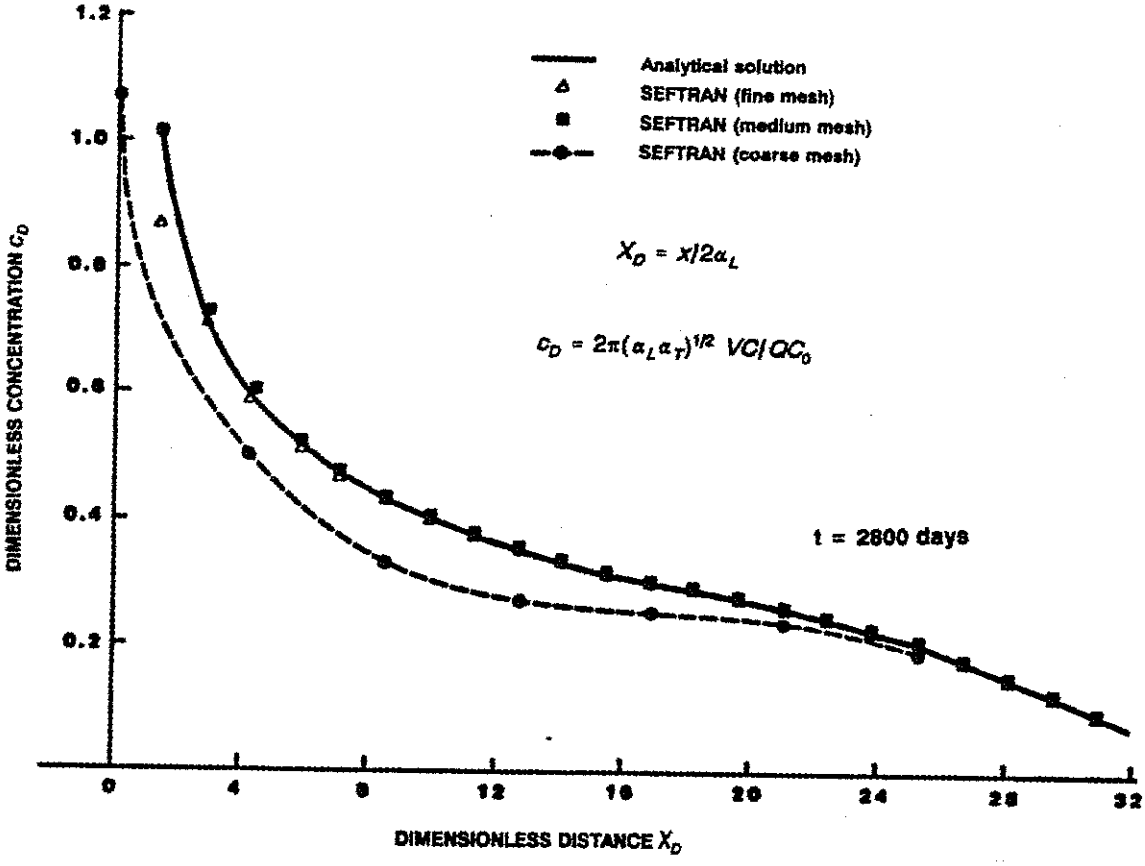

図４．８－３　問題1.3、事例1の解析解法と数値解法の比較を見せる
X軸に沿った密集分布（Huyakorn et al., 1984aより）

これらのテスト目的の評価は、シミュレーション結果の現実性のむしろ直観的で定性的なアセスメント、そして変則的反応の数値解法を調査することによって限定された。

第３レベルのテスティングでは、Huyakorn et al.（1984a）はカナダにあるオンタリオのカナダ空軍基地ボーデン埋立地の塩化物水柱の動きを説明する実験データセットを使用した。研究チームによって得られたシミュレーション結果は、他のモデル開発チームのフィールド観察や予測と比較された。

Beljin（1988）は、異なる数値解法技法を表わす３つの２次元溶質輸送モデルに、IGWMCテスティングアプローチを適用した。SEFTRANコード（Huyakorn et al., 1984b）は、有限要素技法の形を表わし、USGS 2D-TRANSPORT/MOC/KONBREDコードは、特質の方法と有限差分解法技法を結合させ（Konikow and Bredehoeft, 1978）、RANDOM WALK（Prickett et al., 1981）は、流動のための有限差分概算と輸送のためのランダムウォーク粒子移動技法を使用する。概念的アルゴリズムの数値の正確さはシミュレーション結果と溶質輸送の５つの解析的結果を比較することによって検証された。

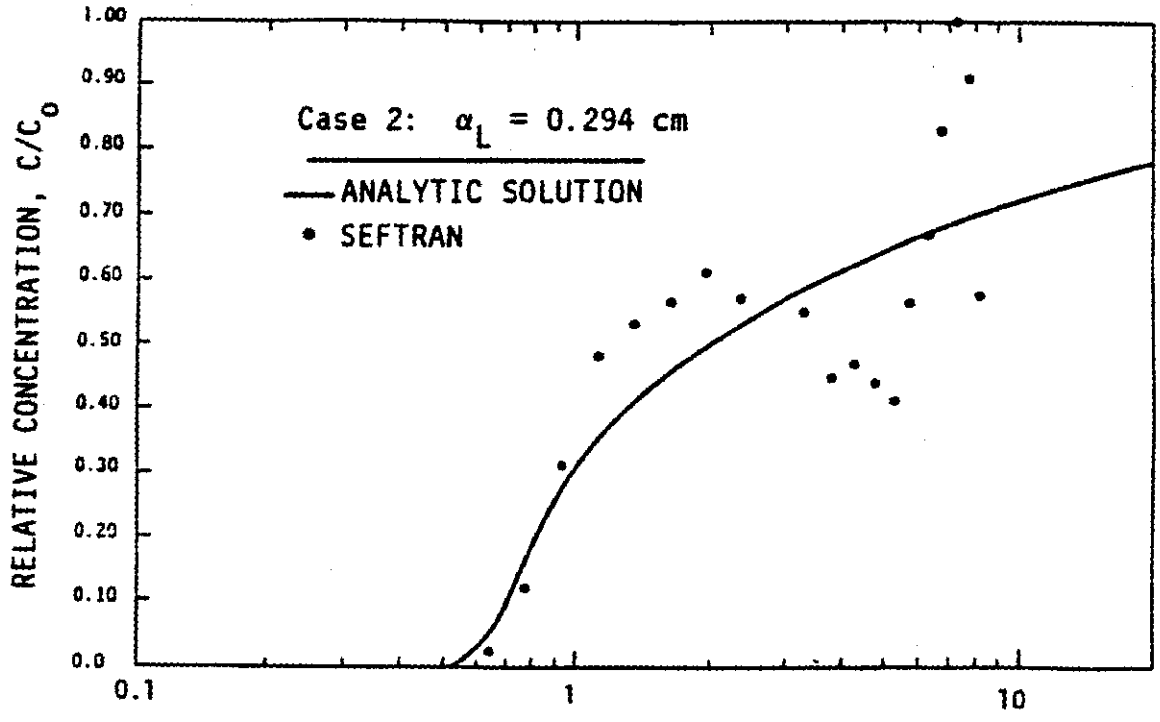

図４．８－４　問題1.6、事例２の解析解法と数値解法の比較を表わすポンプされた井戸の時間対密集（Huyakorn et al., 1984aより）

時間と空間の離散化に加えて、様々なパラメータへの各コードの感度を評価するための検証も使用された。各テスト問題で、明確なテスト問題、数学的公式化、及び使用されたデータセットが挙げられている。Beljin（1988）は、Huyakorn et al.（1984a）によって公式化された６つの第１レベルテスト問題のうち、最初の５つを選択した。表とグラフの両方で、コードの数値結果は解析解法と比較された。結果のグラフ表示は、解放の初めからの特定時間における相対的密集対距離（図４．８－４参照）と、ソースからの特定距離における相対的密集対時間（図４．８－５参照）であった。

報告は適合度を定性的に定量的に表わした。定性的には、「劣等」、「適度」、「受け入れられる」、「良い」、「大変良い」等の属性によって説明されている。定量的には、適合度は密集の解析的に数値的に計算された値の間の平均根２乗誤差によって表わされている。

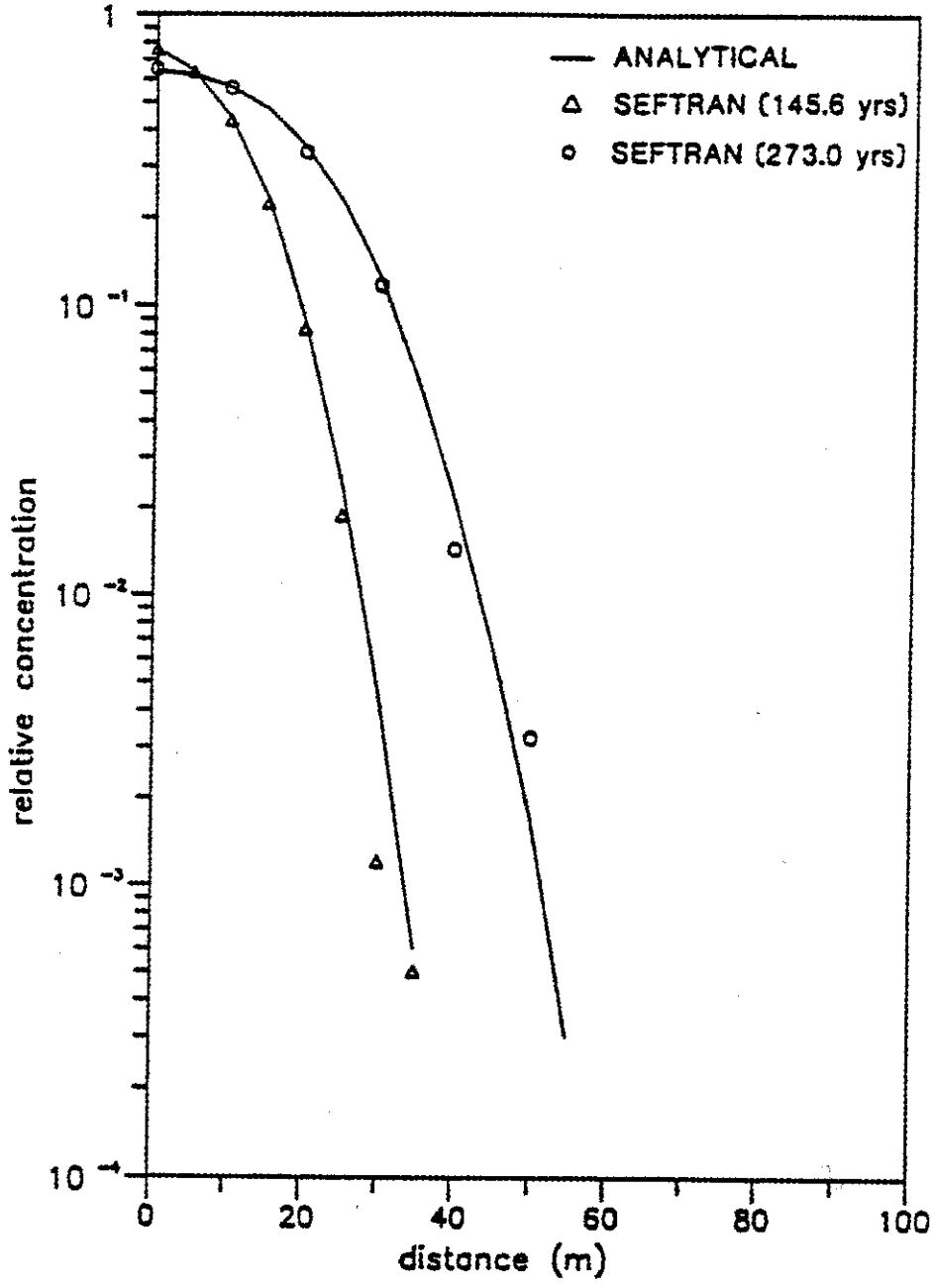

図４．８－５　問題BM-1.2の解析解法と数値解法の比較
相対的密集対距離（Beljinより、1988）

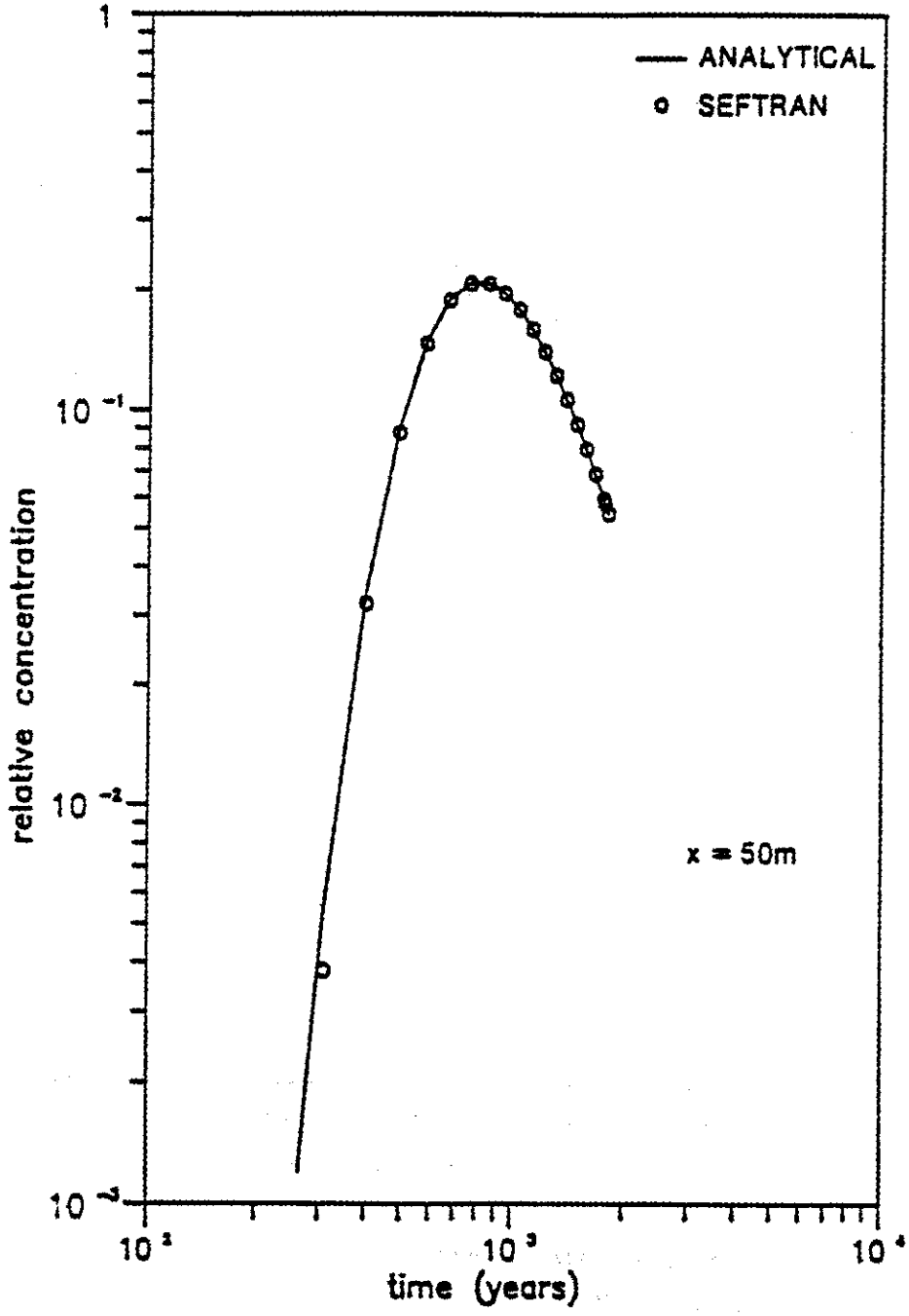

図４．８－６　問題BM-1.2の解析解法と数値解法の比較
相対的密集対時間（Beljinより、1988）

４．８．３　比較コードテスティングの必要性

前述の文献レビューが、コードテスティングにおいて現在進行中のいくつかの作業を説明するにもかかわらず、統一された首尾一貫したガイドラインや過程とともに、より体系的なコードテスティングアプローチが必要とされている（Donigian, 1983）。そのようなテスティングアプローチは、広範囲

のコードタイプに適用でき、全てのポテンシャルコード性能問題分野を挙げるベンチマーク問題（すなわち、問題記述、データセット及び解法）を提供しなければならない。更に、そのようなテスティングアプローチは、検証と妥当性検査過程の発展の評価と、コード性能特徴を確立するために定量的定性的測定を提供しなければならない。コードが、そのような体系的で包括的な過程に委ねられている場合、コードのドキュメンテーションとオペレーション上の特徴を評価する機会が存在することに注目しなければならない。

Streile and Simons（1986）に指摘されたように、多次元理論や概念化をテストした実験研究がほとんど無いため、シミュレーションモデルの検証や妥当性検査を目的にした研究が早急に必要となっている。Engesgaard and Christensen（1988）の化学輸送モデルの研究においても、そのようなモデルにおける輸送過程や化学変換のさらなる検証や妥当性検査研究を確証している。より複雑なモデルの開発よりも、むしろモデルテスティング研究にさらなる労力が向けられなければならない、というのが彼等の主張である。一般的には、現在のコンピュータモデルの可能性は、それらの可能性を広げる前に、その真実性が確証されなければならないと考えられている。

４．１０　戦略と実例ベンチマーク問題のテスティング

コードをテストする際には、それは増加する複雑な問題をシミュレートする際の性能の段階的評価に委ねられなければならない。IGWMCテスティング戦略は、最初に第１レベルに焦点をおき、完成に成功したら第２レベルのテスティングに進むことによってこれを試みている。最後に、適切なフィールドまたは実験データセットが利用可能な場合、コード（とその根底にある理論的モデル）は第３レベルテスティングに委ねられることによって、さらなる真実性を得るかもしれない。

飽和された流動コードにおいては、テスティングの焦点は（空間と時間における）水力学的ヘッド、ヘッド勾配、全体的な水バランスと個別の境界流動、流れの速度パターン（方向と振幅）、流動パスライン、キャプチャーゾーン、及び移動回数の分散の計算である。溶質移動コードでは、そのような評価は帯水層における（空間と時間の）密集分散、全体的質量バランス（種ごと）、観察点やシンク（井戸、小川）のブレークスルー曲線と関係する。

第１レベルでは、コードはその個別の可能性と機能について解析される。テスト問題は、これらの機能のオペレーションを検証するためデザインされている（機能性テスティング）。流動をシミュレートするコードにおいては、境界幾何学と条件、放水路への流動、小川からとそれに向かう流動、その他を扱うコードの可能性を含む。とりわけ、流動フィールドは数値対称のため、そして数値解法は反復発展のために評価されている。溶質輸送については、時間と空間における密集分散は、対称、数値の正確さ、そして解法過程の効率性に関連して評価されている。ほとんどの数値の溶質輸送モデルが各タイムステップの速度フィールドの計算を要求するため、コードの輸送部分に使用されている速度の方向と振幅の正確性を評価するために特別な注意が必要である。これは、第１レベルに水力学的ヘッド分布を提供するだけでなく、速度分布も計算するベンチマークを要求する。

数値モデルの第１レベルのベンチマークは、勾配解析に基づく単純計算からダルシーの法則と質量バランス、そして複雑な解析解法の範囲に及ぶ。このテストレベルでは、モデルのアルゴリズムのみではなく、グリッドオリエンテーションと解像の感度、及び時間の離散化もチェックされる。

飽和された流動のための解析解法はHarr（1962）、TNO（1964）、Edelman（1972）、Huisman

(1972)、Bear (1979) 及びStrack (1989) から見られるかもしれない。不飽和流動のための解析解法は、Potter et al. (1987) に見られる。溶質輸送のための追加の解析解法は、Lester et al. (1986) とWexler (1992) により提供されている。

第２レベルでは、コードは特定コード反応を研究するためにデザインされた仮説的問題に適用されている。例えば、パラメータフィールドにおける特異点や断絶のまわりや、内部と境界幾何学における変則の近くである。「根拠のある事実」を与える解析解法が無いため、コードのテスト結果は予期せぬ説明されていない反応をチェックされ、コードの相互比較は性能の相対的な測定を得るために使用される。このような形のチェックは、コード反応についての多くの要因の影響を研究するために使用できる。そのような要因は、様々な水文地質学的条件（帯水層の成層と同次など）、物理化学的過程とそれらの個々のパラメータ、境界と初期の条件、従属変数の勾配における大きな変動（例えば溶解した土地）、そしてソースとシンクを含む。

第１レベルの場合のように、第２レベルでは関連した数値分散や振動に加えて、グリッドオリエンテーションと解像の影響が評価に含まれている。この目的のため、テスト問題はペクレ数とクーラン数の臨界の範囲を使用して解かれる。これらの臨界条件、高い解像数値グリッド、及び小さいタイムステップを効率的に扱うために、知られているコードを使用して正確な数値解法が生成される。このアプローチは、空間と時間において離散化が小さいほど、左右する部分的な微分方程式の本物の（未知の）解法を概算数値解法が表わすという考えに基づいている（Huyakorn and Pinder, 1983）。ベンチマークは、粗い解像グリッドと大きなタイムステップから高い解像グリッドと小さなタイムステップまで、段階的な方法で開発される。各実行の後、計算に関する違いがモニターされる（実行時間も同様に）。さらなる精製が、例えば2の因数と計算に関する違いにおいて重要な変化を与えない場合、ベンチマークは確立される。高い分解の統合的データセットと、その解法がモデル開発者やユーザーにとってベンチマークとして利用可能になる。第２レベルにおいてベンチマークを確立するためには、選択されたコードは第１レベルと第２レベルの完成を成功していなければならない。更に、成功したフィールドアプリケーションは、コードの真実性や、それとともに確立されたベンチマークの受け入れを高める。第２レベルテスト問題の実例が４．１０．２と４．１０．３に示されている。

４．１０．１　モデル表現がテストされるべき典型的なフィールド条件

（１）境界条件

・指定されたヘッド（時間に一定または変動する）
・リチャージ（沈殿から、または合流として。時間に一定または変動する）
・流れが無い
・指定された流れ
・ヘッドに従属した流動（例えば、小川、貯水池、放水路、泉の流れ、蒸発散）
・指定された密集
・指定された溶質流動
・密集に従属した溶質流動

（２）ソース／シンク

・井戸（個々の井戸、井戸のフィールド）

・放水路（開いた放水路、地下の放水路）
・池、貯水、湖
・発掘、開いた鉱山の穴
・ポイントが溶質のソース
・制限された範囲の地域の溶質ソース
・ポイントの無い溶質ソース（例えば、沈殿によって導入される化学薬品）

（３）帯水層条件

・限定されたもの
・漏れの限定されたもの
・浸潤層のもの
・同次（例えば、単一の帯水層、帯水層-半透水層システム、浸透性の低い不連続のレンズ、帯水層と制限された層の消滅、スケール依存、高度に対比された隣接したパラメータゾーン）
・異方性（水力学的伝導性、分散性）

（４）流動の特徴

・一律な流動フィールド
・安定状態、対称的に湾曲した小川のライン
・不安定な流動フィールド
・変動する密度
・浸透性の媒体の流れ
・離散の断裂
・二重の有孔性

（５）輸送過程

・移流に支配された輸送
・移流分散性の輸送
・遅延（線形平衡の吸収）
・ファーストオーダー衰退

４．１０．２　統合的データセット1（図４．１０－１参照）

（１）一般的見解

この問題は、制限された範囲における連続した地域のソース汚染物質の水柱から、総称的であるが現実的な制限の無い地下水システム分割、２つの不浸透性の平衡した岩石の露出と完全に透入した小川の輸送に関係している。

（２）目的

この問題は、一律でない制限されていない流動フィールドにおける輸送をシミュレートする輸送コードの能力を検証する。即ち、それはコードの流動と輸送をシミュレートするモジュールをテストする。グリッドオリエンテーションと解像影響は、この定住している問題において探究されている。

次の３つのタイプのグリッドオリエンテーションが使用されている。

・本流の方向に平行

・本流の方向に対して45° 以下の角度

・本流の方向に対して60° 以下の角度

問題は次のように実行される。

・地域のリチャージと溶質ソース

・活動中の井戸と溶質ソース

・地域のリチャージと活動中の井戸の両方

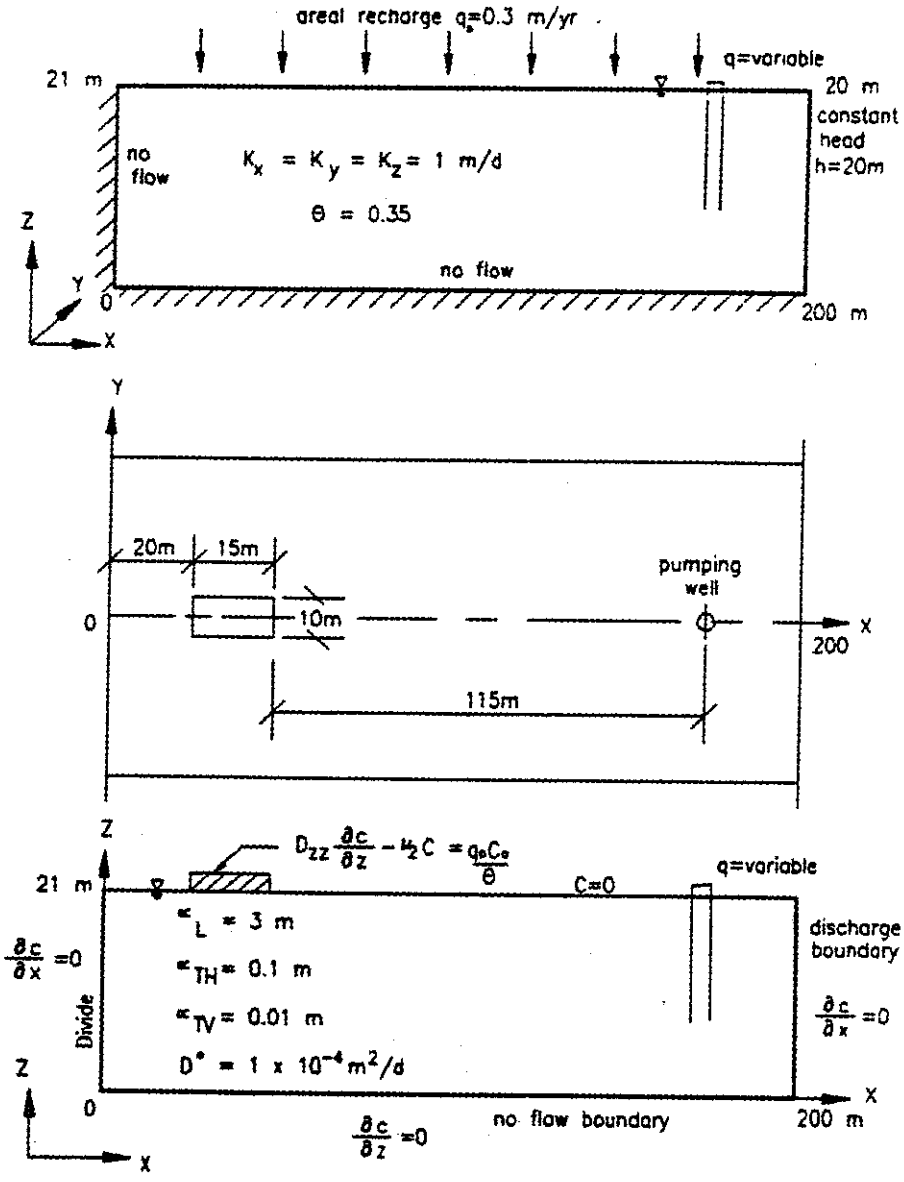

図４．１０－１　データセット１

（3）仮定

・水力学的伝導性に関しては、帯水層は同次で等方である

・地下水の流れは安定している

・流体密度と粘度は一定である

・溶質ソースは流動フィールドに影響を与えない

（4）パラメータの指定

単一のポンプ式の井戸（400m3/d）と汚染物質のソースは、200mと200mの幅で21mの厚さの帯水層に位置している（ソースの次元や位置と井戸の場所については図４．１０－３を参照）。シミュレーションを行なう際に次の物質的パラメータが使用される。

Kx = Ky = Kz = 1.0 m/d（細砂から中砂の水力学的伝導性）

θ = 0.35（完全に同形な砂粒の有孔性）

D* = 1 × 10-4 m2/d（分子拡散係数）

図４．１０－２　統合的データセット1

一律ではない、制限のない流動フィールドにおける連続したソースからの汚染物質水柱の移動。

４．１０．３　表4. 統合的データセット2（図４．１０－３参照）

（1）一般的見解

この問題は、連続しない2mの中粒砂層が位置する沈泥状の砂で作られた，地域的にリチャージされた制限のない帯水層における汚染物質の移動と関係している。境界は性質上は統合的データセットのそれと同じである。この問題はSudicky（1989）に基づいている。

（2）目的

この問題は、垂直方向への不均質な帯水層属性をもつ一律ではない、制限のない流動フィールドにおける輸送をシミュレートする輸送コードの能力を検証する。即ち、高い浸透性のレンズをもつ層にした帯水層における流動と輸送をテストする。グリッドオリエンテーションのシナリオは統合的データセット1に使用されたものと同じである。

（3）仮説

・水力学的伝導性に関しては、帯水層は等方である

・地下水の流れは安定している

・流体密度と粘度は一定である

・溶質ソースは流動フィールドに影響を与えない

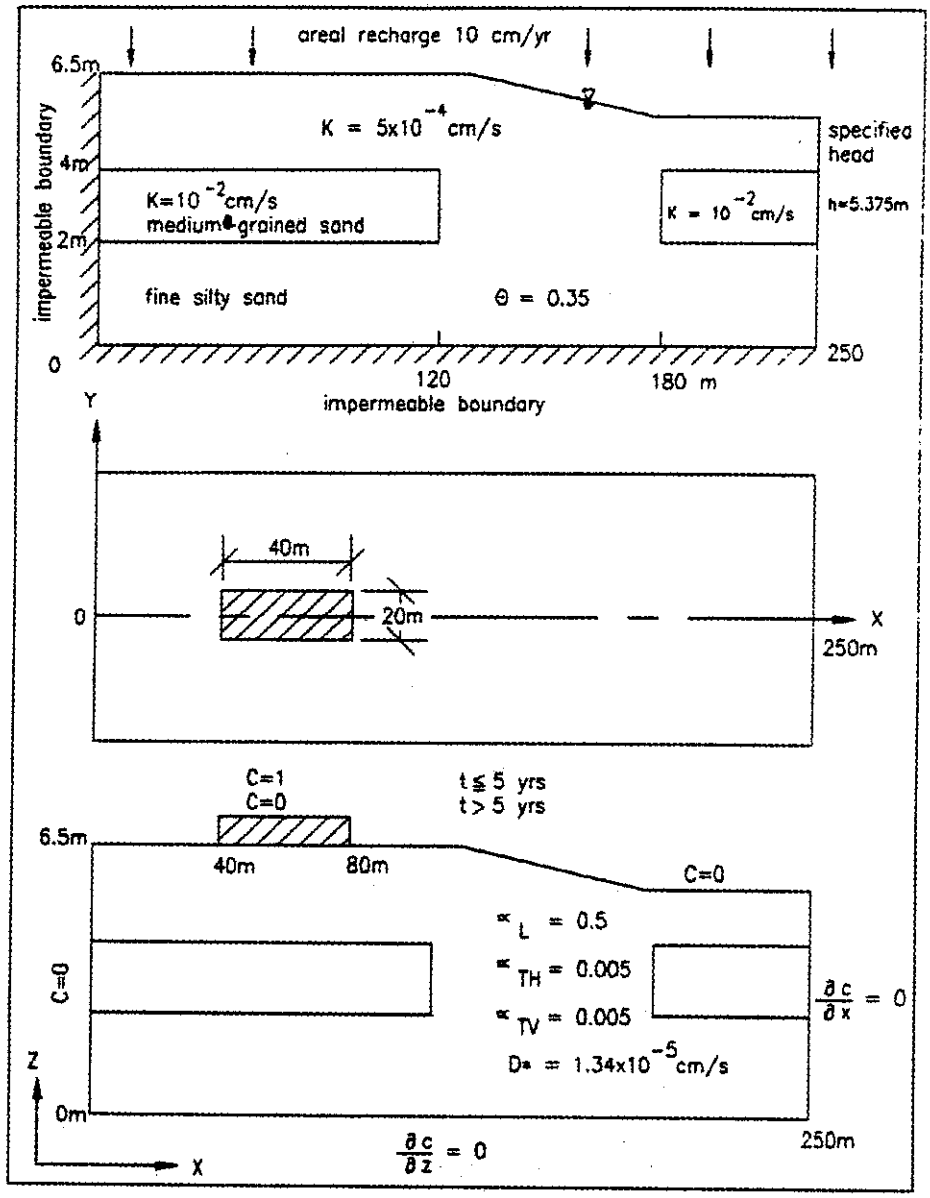

図４．１０－３　ケース２

（４）パラメータの指定

汚染物質ソースが、250mと250mの幅の制限のない帯水層に位置している（ソースの次元や位置と、境界の位置と境界条件については図４．１０－４を参照）。シミュレーションを行なう際に次の物質的パラメータが使用される。

Kx1 = Ky1 = Kz1 = 5 × 10-4 cm/s（良質な沈泥状の砂の水力学的伝導性）

Kx2 = Ky2 = Kz2 = 1 × 10-2 cm/s（中粒砂の水力学的伝導性）

θ = 0.35（完全に同形な砂粒の有孔性）

D* = 1.34 × 10-5 cm2/s（分子拡散係数）

dispersivity values

	α_L	α_{TH}	α_{TV}
1	1.0	0.1	0.01
2	1.0	0.01	0.01
3	1.0	0.01	0.001
4	1.0	0.5	0.1
5	5.0	0.1	0.1

α_L = longitudinal dispersivity

α_{TH} = transverse horizontal dispersivity

α_{TV} = transverse vertical dispersivity

図４．１０－４　統合的データセット2。不均質な帯水層における汚染物質の輸送

前述した通り、テスト問題の順序には論理的な進歩がなくてはならない。「コードテスター」は、次のテストに進む前に各テストの結果を評価しなければならない。必要であれば、様々なパラメータやストレス条件におけるコードの反応を探究するため（即ち性能評価である）、現在のテスト問題を異なる入力値で繰り返されなければならない。４．１０．４項は３次元溶質輸送コードのテスティングのためにデザインされた順序を表わす。

４．１０．４　３次元溶質輸送コードのためのテストシナリオ

1. 大きな同次で等方の帯水層の安定状態で一定の流動フィールドにおける溶質輸送（解析解法が利用できる）
2. 大きな同次の帯水層の一定ではない安定状態の流動フィールドにおける溶質輸送（解析解法が利用できる）。
3. 大きな同次で等方の帯水層の一定でない流動フィールドにおける溶質輸送（解析解法が利用できない）
   - 3.1. 安定状態の流動フィールドをもつ。
     - 3.1.1. 異なる溶質ソースパターン
     - 3.1.2. 異なる境界条件
     - 3.1.3. ソースやシンク（流動と溶質輸送のため）と結合された異なる境界条件
   - 3.2. 不安定な流動フィールドと
     - 3.2.1. ソースとシンク（流動と溶質輸送のため）
     - 3.2.2. 異なる境界条件
     - 3.2.3. 両方
4. 不均質で異方な帯水層の一定ではない流動フィールド（解析解法は利用できない）
   - 4.1. 層にしたシステム
     - 4.1.1. 安定状態の流動フィールドと
       - 4.1.1.1. ソースとシンク
       - 4.1.1.2. 異なる境界条件
       - 4.1.1.3. 両方
     - 4.1.2 不安定な流動フィールドと
       - 4.1.2.1. ソースとシンク
       - 4.1.2.2. 異なる境界条件

4.1.2.3. 両方

4.2. レンズの不均質

4.2.1. 安定状態の流動フィールドと

4.2.1.1. ソースとシンク

4.2.1.2. 異なる境界条件

4.2.1.3. 両方

4.2.2. 不安定な流動フィールドと

4.2.1.1. ソースとシンク

4.2.1.2. 異なる境界条件

4.2.1.3. 両方

4.3. ランダムな不均質

4.3.1. 安定状態の流動フィールドと。

4.3.1.1. ソースとシンク

4.3.1.2. 異なる境界条件

4.3.1.3. 両方

4.3.2. 不安定な流動フィールドと。

4.3.2.1. ソースとシンク

4.3.2.2. 異なる境界条件

4.3.2.3. 両方

全タイプの地下水モデルの包括的で体系的なテスティングと、検証と妥当性検査プロトコルの確立が早急に必要とされている。地下水管理の決定は、データ収集、情報処理、及び解釈の科学的にも技術的にも確かな方法に基づかなければならない。多次元理論、概念化、及び関連したコンピュータコードをテストした実験機関がほとんど無いため、重要な地下水モデルのための検証と妥当性検査研究の開発と実行を目標とした今以上の研究を行うことが大変重要である。地下水管理の見地からは、より複雑なモデルの開発より、むしろモデルテスティング研究に更なる努力が向けられるべきであると論じられるであろう。

近年、IGWMCは、包括的な品質保証プログラムを履行するための努力の一部として、モデル評価のためのテスティング過程と方法論を開発した。現在のプロジェクトは科学的考察を体系的に解析し、そのような方法論の履行のための技術的要素の収集を試みている。次のステップは、この包括的な方法論を実際のコンピュータコードに適用することである。

5　システムの試作

5．1　システム開発方法

過去10年の情報テクノロジー分野における根本的な発展は、非常に急激であったと特徴づけられる。情報システム開発のための新しい支援方法が普及し、その支援方法による直接的な効果により、情報処理に関する問題は解決可能になりつつある。現在利用できる多くのシステム開発戦略、構造化開発手法及びソフトウェア・ツールは、情報処理問題の解決のための方法を選択する際に必須なものとなっている。

情報に関する問題を解決するには様々な選択肢があるが、十分考慮してこの選択を行なうことにより、プロジェクト成功へのチャンスが大いに増すということはあまり認識されていない。一般に情報システム部門は1つの開発手法に集中する。もし、この手法が望ましい結果をもたらさないと（情報問題に対し、タイムリーに有効な結果を生み出さない）、すべての原因は、使用した戦略及び手法にあるとされることが多い。しかし、その原因は、使用した開発戦略や手法が現在取り組んでいる問題に不適当であったということであり、手法の質そのものに対しての評価はそれほどなされていない。実際には、開発戦略や手法に本質的な良し悪しはほとんどなく、特定の問題に対しての適用性があるかないかということである。

5．1．1　問題解決へのアプローチ

情報に関する問題が発生した場合、まず誰がその問題解決に最も適しているか、つまりエンドユーザーか情報システム部門かを決める必要がある。場合によっては、先進的で親和性のあるツールを使用することによって、エンドユーザーが自主的にアプリケーションを開発することも可能であるが、多くの場合、ユーザーが自分自身のシステムを開発することは不可能である。やはり、大規模で複雑な、しかも質の高い情報システムを構築できる情報部門の要員で組織されている情報システム部門が、広い観点からこのようなシステムを見ることが必要である。

有効な情報システムを効率的に開発するため、専門家であるシステム開発者は、あらゆる戦略、手法及びソフトウェア・ツールの使用が可能であり、それらの戦略・手法は、プロジェクトを計画し、統制するプロジェクト管理のためのマネージメント支援、及びアプリケーション・システム開発担当者のための開発支援と大別することができる。プロジェクトの初期段階で、プロジェクト・リーダはプロジェクト・モデル（必要な作業をフェイズごとに分ける）と、開発方法、手法、技法及びツールを選択しなければならない。この選択は、問題とその内容の性質に左右されるが、すべての情報システムを同じ開発支援を使って開発する必要性はない。

プロジェクト・リーダが正しい開発戦略を選択するためには次の開発戦略がある。それらは、パーソナルコンピュータ上でエンドユーザーが開発する、メインフレーム上で第4世代言語の助けを借りてエンドユーザーが開発する、および情報システム部門が開発するといった戦略である。以上の3つの戦略からの選択は、たとえば同時に使用するユーザーの数、システムを利用する場所の数、及び格納されているデータに関心をもつユーザーの数といった開発すべき情報システムについての具体的な特質を基に行なわれる。

開発戦略に対するのと同様に、プロジェクト管理に対する最良のアプローチは単純なものではない。プロジェクトの特質に基づいてプロジェクトを計画し統制する方法は、現在利用できるものの中から

選択する必要があるが、その方法はプロジェクトのリスクを基に選択を行なうことが考えられる。このリスクとは、プロジェクトの予算やスケジュールの超過、利用するシステムの満足度、または最終的に全プロジェクトが失敗に終わるという可能性が、どの程度起こるかということである。

プロジェクトのリスクは、相対的な規模、採用するテクノロジーの経験量、プロジェクト構造（問題の構造化度合い）により決定される。与えられた問題の状況に最も適した戦略、手法、ソフトウェア・ツールを選択するための具体的なガイドライン作りは、現在様々な分野で行なわれている。

プロトタイピングと伝統的アプローチ

要求定義段階での１つのアプローチはプロトタイピングである。ユーザーと開発担当者にとって、プロトタイピングの概念は有益と考えられている。新しい問題が起こるたびに、その利点と欠点を考慮せずに、この方法を選択する傾向があるが、プロトタイピングは万能ではない。つまり、プロジェクトの初期段階で、伝統的なダイアグラム技法とプロトタイプとを組み合わせることが意味があるかどうかを検討すべきである。プロジェクト・リーダは個々の状況に応じて、完全で、厳密で、確固たる要求定義を行なうため最良のアプローチを決めなければならない。

プロトタイピングは、ユーザー・インターフェイスのモデル化に限定される可能性がある。従来の手法では、プロトタイプのユーザー・インターフェイスを作成しない場合、システムの一面が現実には無視されることになる。スクリーン・ペインタのようなツールを使用することにより、ユーザー・インターフェイスは紙の上でモデル化するよりさらに迅速に作成が可能になり、また修正を加えることが可能になる。したがって、ユーザー・インターフェイスが、開発中のアプリケーションにとって重要である場合や、ユーザーがユーザー・インターフェイスやシステムの最終的な形に関して見解を持っている場合に限られるが、ユーザー・インターフェイス・プロトタイピングは実用的なものとして推奨される。

プロトタイプを採用すべきか否かは機能型プロトタイピングを考えた場合、選択が難しくなる。機能型プロトタイプと従来の作図技法を使って作るモデルとは、システムの同じ側面を描くことになり、そのようなプロトタイプを作るのは費用的に問題が生じるからである。

（1）プロトタイピング手法

ある特定の状況におけるプロトタイピングの使用は、経済的な観点から検証することが望ましい。つまり、予測される利益は予測される費用より大きくなくてはならないが、容易に推測することができない。確かな計数的なデータを得るには、伝統的な方法とプロトタイピングを使って開発した方法の多くのシステムについて開発費用を測る必要がある。そのシステムは“ノイズ”を濾過するための統計手法を用い、管理が行き届いた状態で使用されることが必要である。プロトタイピングの相対的費用、あるいは利益は、ライフサイクルの費用をそれぞれのカテゴリごとに比較した結果、推測することができる。しかし、このような実験には費用がかかり、長期間にわたって続ける必要性があることから、この種の実験的なデータを得ることは困難である。

ソフトウェア開発の費用は使用する戦略だけでなく、他の多くの要素とも関係する。開発費用の見積りは、プロジェクトの初期に適用できるものが望ましい。この分野の研究では、ソフトウェア開発費用は開発担当者に大きく依存する。つまり問題となるのは、手法に関係なく、その手法を適用する

担当者ということである。

さらに、より先進的な言語は開発費用に多大な影響を与えるということが考えられる。プロトタイピングの費用と従来のアプローチによる費用との比較調査をする場合、他の重要な費用推進力の影響が小さくなることが考えられる。この種の研究は、どの方法によって最も生産性が向上するかを判定する代わりに、システム開発関数の生産性に対するプロトタイピングの影響を確立することを目指している。

（2）プロトタイピング手法の恩恵

プロトタイピング・アプローチを採用するメリットは、主として情報問題の本質についての不確実性と、その情報問題を解決するためにユーザーが提示する要求の不確実性を減少させることにある。“不確実性”とは、ある問題に関する既存の知識と、その問題解決に到達するために必要となる知識との差である。伝統的な技法も、この不確実性を排除することを目指しているが、成功するまでには至っていない。

本項では、不確実性を低減することにより生じる諸々の費用の節約について、及びプロトタイピングの他の利益について述べる。

保守費用の削減

情報問題についての不確実性と、その問題を解決する情報システムの保守に要する費用とは直接関係する。つまり、不確実性を減らすことによって保守費用は下がることになる。このことから、情報システムの全ライフサイクル費用の50%は要求定義の質の悪さが直接の原因となっている。したがって、プロトタイピングを利用した代表的な例では相当な節約が可能となる。しかしながら、ユーザーの要求と要望について不確実性がない場合、つまり明確としていれば、要求定義に対するこのアプローチの利点はなくなる。この状況では、従来のアプローチもプロトタイピングも同じく（あまり）保守費用が減ることになる。個々の状況でプロトタイピングの潜在的な利益を査定しようとすると、どの程度の不確実性が存在するかを明らかにしなければならない。

不確実性はシステムの外的側面だけでなく概念的側面とも関係する。実際、概念レベルについての不確実性は非常に深刻な影響をもたらす。この場合、システムの機能について、またデータベースの構造について明快さを欠くことになる。この分野における誤りや欠陥は、一般に根本的なそして費用を要する修正が必要となる。システムが取らねばならぬ外観についての不確実性は、費用面で重要な影響をもたらすことにはなりそうにはないが、満足感に対しては重大な影響を及ぼす。この分野で明快さを欠くと、ユーザーとの親和性には乏しくなるが、要求されたことを一応行なうシステムにはなる。ユーザー・インターフェイスを修正することは、システムの機能変更を行なうほど根本的な問題とはならない。以下に示す規準を基に、ある状況で概念の不確実性が存在するか否かを判定するのは可能である。

(a) 支援が必要な仕事から要求を演繹する。

支援を必要とする仕事を分析することから、情報システムへの要求が出てくる場合が多くある。そこでは以下の特徴が見受けられる。

新しいシステムが支援しなければならない工程は完全に定義される。
このことは、仕事が定型的な、あるいは繰返しが多いということと直接関係する。
(Mann and Watson、1984)
支援が必要な組織の役割は、構造面においても運用面においても堅固である。
(Davis、1982)

情報システムへの要求が、支援を必要とする組織の役割を分析することによって作成できるとなると、つまり支援を必要とする仕事から直接要求を作り出せるとなると、不確実性の問題が深刻になるということはほとんどない。またこのような場合、開発者とユーザーとの間のコミュニケーションに問題が起こることはなく、システムの仕様はかなり堅固なものとなる。

(b) ユーザーおよび分析者の知識と経験のレベル

新しいシステムの要求定義がすべて支援を必要とする仕事から出てこないとなると、新しいシステムを使うことになる将来のユーザーや、そのシステム開発に携わる情報分析者の見識に頼らざるを得なくなる。問題の場面に係わっている人々をよく理解すればするほど、“本当の”情報問題が明らかとなるチャンスが増え、またユーザーがそのシステムを使い出した後に要求定義の変更が起こるといったことは少なくなる。

情報問題とその解決策に対するユーザーと情報分析者の見識は特に以下の点に左右される。

関係するアプリケーション分野に対する知識と経験
同じような、情報システムに対する以前の経験
一般的に自動化された情報システムについての経験
支援を必要とする組織の役割の安定性
開発するシステムの複雑さ

ユーザーも情報分析者も、いずれも問題の状況について十分な知識を持ち合わせていない場合、プロトタイピングは唯一の正しいアプローチとなる。しかし、少なくともどちらかが問題状況について、また開発するシステムの要求について十分な見識をもっていれば、要求を習得するためのモデルを利用する必要性はなくなる。情報問題が理解されると、ユーザーがそのシステムを使用し始めた後、システム要求定義が変わる危険はほとんどなくなる。このような場合、プロトタイプはコミュニケーションの方法として、依然として有用な役を演じることができる。

(c) コミュニケーション問題の可能性

ユーザーと情報分析者が互いに徹底的にコミュニケーションする必要が生じるのは、情報の必要性についての不確実性を同じ言葉で話せない双方の当事者が原因となって起こるからである。この場合、不確実性は情報問題を理解できないからではなく、理解するためのコミュニケーション能力が低いからである。コミュニケーション問題は次の場合危険度が高まることになる。

ユーザーが使用される分析用言語に経験がない。
ユーザーが情報分析者の使う言語に否定的な態度を示す。
ユーザーが抽象的思考能力を前提とする言語にうまく対処できない。
情報分析者が関係するアプリケーション分野とその分野の用語についての経験をもっていない。

概念レベルの不確実性はシステム要求が仕事の分析からでなく、ユーザーも情報分析者もアプリケーション分野の知識と経験がなく、そして彼らの間でのコミュニケーションが取れない場合に最大の問題となる。このような環境では、プロトタイピングは相当な時間的節約となるのはほぼ間違いない。

その他の利益

不確実性を減らし保守費用を削減できるほかに、プロトタイピングを使うことによって、より一層の節約が可能となる。従来の直線的に進められるプロジェクトと比較すると、テスト（機能）期間は短縮できること、また費用も要せずに済む（これはプロトタイピングを用いることにより、このフェイズの費用がなくなるからではなく、要求定義フェイズにテストのほとんどが移ってしまうからである）。新しいシステムの組織への導入がより円滑に進むことから、プロトタイピングはさらに利益をもたらすことになる。

第1に、一般的にシステムは組織に適度に取り入れられ、次にプロトタイピング作業に係わったユーザーは、既にそのシステムを熟知していることになる。それらから、教育費用は極端に下げることが可能となる。プロトタイプを本番のシステム（一部の）のベースとして使用できれば、保守費用の削減に加え、設計および実現フェイズにおいても相当な節約となる。この場合、情報システムの重要な部分はプロトタイピングの後には既に使えるようになっていることある。この状況は、プロトタイプが本番システムで使われるのと同じ言語を用い、そして同じ（技術的）環境で構築される場合、特にその影響がある。多くの第4世代開発環境は、プロトタイプと本番システム両方に対し優れた実現化のための機能を提供している。量的に、しかし恰も現実のように測定することは困難であるが、実際に開発費を要する必要がないことから諸々の節約が生まれてくる。ユーザーは、初期の時点で自由にプロトタイプを使って十分有効なシステムを構築することができる。

このことは相当な節約となる。たとえば、効果的な物流システムを迅速に構築できれば、在庫量の節約が可能となる。プロジェクトのスループット時間を短縮でき、組織の柔軟性を高めることができれば、企業の競争優位が高まることになる。企業がその競争相手に対しどれほどの優位性をもてるかは、利益を具現化する速度に左右される。つまり競争に停滞はないということである。情報システムの完成時期が企業の競争優位を獲得するのに非常に重要であるという例は、いくらでも挙げることができる。

（3）プロトタイピング手法の経済性

プロトタイピングの費用は、使用するツールの品質と先進的な機能に大いに左右されることは明らかである。ユーザー・インターフェイスをスクリーン・ペインタとレポート・ライタで作成した場合、その費用は従来の開発ツールを使うよりも何倍も安価にできる。強力な手続き型言語および非手続き型言語を統合化された開発環境の一部として使用することにより、プロトタイプの開発および変更の

両方の費用低減が可能となる。

優れたリレーショナル（あるいはセミ・リレーショナル）データベースを使用できる場合は、データベース構造の変更やデータベースの再編成を相当簡単に、しかも安価に行なえる。ネットワーク型データベースを再構築するには何倍も余計に費用を要し、もし階層型のデータベース管理システム（DBMS）だけしか利用できないのであれば、この仕事に取り掛かる前に少なくとも2回はよく考えることが賢明である。

階層型DBMSは、データベース構造をプロトタイピングの助けを借りなくとも正確に厳密に指定できる開発プロジェクトには適している。さらにプロトタイピングの費用は、プロトタイプを受け入れることができるようになるまでに必要となる繰返し作業の回数にも関係してくる。この繰返しの回数は、プロトタイピング工程の開始時におけるシステム要求の確実性がどの程度かに左右される。優れたツールと完全に構築されたプロトタイプがあれば、変更作業の費用は高くなることはない。先進的なツールが使える場合ですら、プロトタイピングに関係する費用は伝統的な分析費用よりも嵩むことになる。

従来の方法であれば、不確実性の程度が非常に高いと、要求は不確実なまま残るという結果になる。同じような状況でプロトタイピングを使った場合、やらねばならない仕事としては従来のプロジェクトの場合と同じである。しかしこの場合、この不確実性を減らし受け入れ可能なレベルまで到達するには、システムの側面、つまり部分について複数のプロトタイプを構築すること（そしてユーザーに評価させること）も必要となる。

（4）プロトタイピング手法の費用対効果

プロトタイピングによってもたらされる諸々の節約は、主として保守フェイズでなされ、プロトタイプが本番システムのベースとして機能する場合でも、設計および構築フェイズではそれほどではない。情報問題とその解決策について不確実性がある場合にのみ節約が可能となる。要求定義フェイズでは、プロトタイピング・アプローチのほうが余計に費用を要す。このようにプロトタイピングは新たな費用パターンを生み出すことになる。その原因を以下に示す。

a. 要求定義フェイズでは従来のアプローチを用い、本番システムの構築には第3世代言語を用いる。
b. 要求定義フェイズではプロトタイピングを、そして本番システムの構築には第3世代言語を用いる。
（プロトタイピング作業の終りでプロトタイプは棄てられる）。
c. 要求定義フェイズでは従来のアプローチを、そして本番システム構築は第4世代言語で行なう。
d. 要求定義フェイズでプロトタイピングを、そして本番システム構築には第4世代言語を用いる。
（プロトタイピング作業の終りにプロトタイプは棄てられる）。
e. 要求定義フェイズではプロトタイピングを、そして本番システムのベースとしてプロトタイプを使用する。

プロトタイピングを正しく理解するには、シナリオ（a）はシナリオ（b）と比較し、シナリオ（c）はシナリオ（d）と、シナリオ（d）はシナリオ（e）と比較しなければならない。第4世代言語の利用によって変化した費用パターンは、シナリオ（a）と（c）、およびシナリオ（b）と（d）を比較すると明らかになってくる。図5．1－1では、プロトタイピングが適用され、また第3世代言語による従来の開発（シナリオ（a））の代わりに本番システムのベースとして残る場合（シナリオ（e））、

いかに節約が最大となるかを示している。しかし、これらの節約がすべてプロトタイピングの利用によるものだけではない。つまり、部分的には第４世代ツールの利用によるものである。

このことはLangle他（1984）による調査からも明らかで、この中で多くの組織になぜプロトタイプを使うのか（あるいは使おうとするのか）と尋ねている。この質問に対する最も一般的な答えは“開発期間の短縮”である。これは強力な開発ツールの利用によるものであり、間違いなくプロトタイピングの利用によるものではない。図５．１－１を理解するには、この図が単に１つの徴候を示しているにすぎないことに留意しておかねばならない。絶対的な意味がヒストグラムの棒の最高位にあるということではない。この図は多くの仮定を基に作られており、経験的なデータによりその一部のみが認められているにすぎない。もちろん何らかの変更がこれらの仮定になされると図も変わることになる。これらの仮定は以下の通りである。

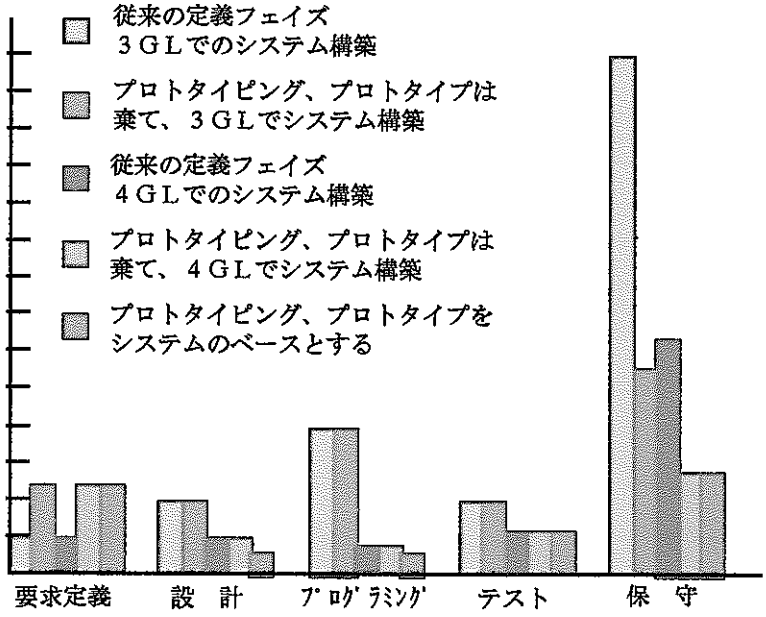

図５．１－１　第４世代言語利用のプロトタイプ費用予想

・情報システムの全ライフサイクルの費用の中で、60%は保守の費用である（システムの出荷後この費用が発生する）。この数字はIBM社の最新の調査からのものである。

・第４世代言語を使用するが分析と設計が従来の方法で行なわれる場合（シナリオ（c））、保守費用は依然ライフサイクル費用の60%となる。システム要求定義の質も同程度であることから、この数字が下回ることはない。

・従来のプロジェクト工程の様々な開発フェイズにわたる費用の分布はZelkowitz他（1979）からのものである。しかし、これらの数字はそれぞれの研究者によって異なっている。というのも、研究者それぞれ各自異なるフェイズを定義しており、また費用のパターンは情報システムの種類と規模に大いに左右されるからである（Boehm、1981）。

・第４世代言語を使った結果生まれる節約は、Rudolph（1983）によって調査されている。図の中

で示されている節約は（シナリオ（a）と比較した（c）とシナリオ（b）と比較した（d））、Rudolphの研究結果と比較すると控え目に見える。

・要求定義について多くの不確実性があれば、この不確実性を除去するために複数のプロトタイプを作る必要がある。これらの費用は第4世代言語で本番システムを設計し実現化するために要する費用を越えてしまう。

・全保守費用の60%はシステム要求定義が不十分なことによる（IBM）。プロトタイピングを通して、これらの費用を除くことができる。このことは、プロトタイピングが終わるとシステム要求定義は、エンドユーザーの要求と希望を完全な形で表現でき、そして情報システムを長期間使ったとしても利用者のニーズは変わらないということを前提にしている。

個々の問題でプロトタイピングの経済的評価が求められる場合、まずプロトタイプを本番システムのベースとして使用するのかどうか、あるいは捨て去られてしまうのかどうかを判断しなければならない。もし前者であれば、プロトタイピングはほとんど常に推奨するのに値するものとなる。何も不確実なことがないような状況だとすると、プロトタイピングは単に本番システムを構築する、また別の1つの方法であるにすぎない。

要求定義フェイズでの余分な費用は、設計と実現フェイズの費用が安くなることで埋合せができる。このような場合、要求定義フェイズでの繰返し回数は多くなることはない。情報システムについて不確実性があると、プロトタイピングはその不確実性を減らし、したがって保守フェイズにおける節約に非常に役に立つことになる。これに対して、要求定義フェイズの費用の増加を比較検討しなければならないが、多分繰返し作業が何度も必要になろう。

しかし、この段階でも、なお情報システムにかなり容易に変更を加えることができる。保守フェイズで変更することは何倍も費用を要することになる（システムが第4世代環境で実現されたとしてもである）。これは実際運用されているシステムが性能やセキュリティも含めてすべての要求を満たさねばならないからである。このような要求を満たすため取られる方法はシステムの複雑さを増し、したがって柔軟性が乏しくなる。反対にプロトタイプは一般に不完全であり、多くの制約を無視することができ、作成されたプロトタイプは複雑ではなく本番システムより変更がやさしい。本番システムの変更に伴う費用が高くなるもう1つの理由は、そのようなシステムは既に組織の中にがっちりと立場を固めているという事実である。出荷後のシステム変更は、しばしば組織上の手続きまで変えねばならないことがある。

プロトタイプをプロトタイピング作業の終了後捨て去る場合、費用便益性が多くの注目を浴びることになる。この場合、要求定義フェイズでの費用の増加は、設計および実現化のフェイズの節約とバランスがとれていないことから、プロトタイピングでの余分な費用は保守費用に見込まれる削減額を越えることはできない。情報問題の不確実性の度合いが大きいと、保守に予想される費用を相当節約できるとの観点から、プロトタイピングのアプローチは評価されることになる。もし不確実性が何もなければ、プロトタイピングに従うことは必要ないし、また賢明なことでもない。したがって、要求定義フェイズでの費用の増加は保守費用の削減によって埋合せをすることはできない。

このように、もしプロトタイプが本番システムのベースとすることができないならば、“厳しい”費用と利益の観点から、要求についての本質的な不確実性がある場合にのみプロトタイピングの方法を利用することが賢明であろう。しかしながら、プロトタイピング・アプローチに従いたいという欲

求にはさらに別の動機がある。

（5）効果的問題解決

プロトタイピングの有形の利益に加え、要求定義にこのアプローチを用いると、さらにいくつかの否定しがたい効果があるが、これらを測るのは特に難しく不可能ですらある。これらの無形の利益は、もちろん要求定義の方法を選択する場合も考慮に入れる必要がある。

ユーザーの態度

プロトタイピングに関する多くの出版物の中で、このアプローチがエンドユーザーの開発工程と情報システム部門の両方に対する態度に実際及ぼす良い影響について取り上げている。

ユーザーの態度の肯定的な変化は、主としてプロトタイピング工程へユーザーが積極的に参加することと、システム開発担当者とエンドユーザー間のコミュニケーションの改善によるものである。

ユーザーは高度なレベルでの参加により、開発工程の進展具合とその成果の両方に対して強い影響を及ぼすことができる。ユーザーは、情報システムの形態と備えるべき機能をほとんど決めることができる。この結果、ユーザーは自分自身がシステムの“所有者”であるとわかる。理想的には、ユーザーと開発担当者が一緒にユーザーの要求と要望をできる限り最も正確に記述するために共同で作業を行なうことである（Dearnley and Mayhew、1983）。また、ユーザーの参加の度合いが高くなると、開発プロジェクトの進展に容易について行くことができるようになる。プロトタイプが早い時点で利用でき、開発担当者とユーザー間の接触がプロトタイピングの過程で拡がると、プロジェクトはより見えやすくなる。これらの要因すべてによって、ユーザーは情報システム部門に大きな信頼を置くようになる。

プロトタイピングについて書いてある本は、一般には論理的なものが多い。このアプローチについての実際的な経験は依然かなり限られており、経験的な研究についての出版物はまだ殆どない。しかし、プロトタイピングがユーザーの態度に肯定的な影響を与えるだろうとの予想は現実に確認されている。

この分野での実際の経験を述べている著者は少ないが、このような肯定的な影響が確かに出てきていることに同意している（たとえば、Earl、1982；EDP Analyzer、1985参照）。

もちろん、それは態度と認識の問題であることから、これらの効果について明確な証明をすることはできない。引用した本は、やはり主観的な観点から報告している。開発工程に対するユーザーの認識の観点から、伝統的なシステム開発方法とプロトタイピングを、より公平に比較を試みた数少ない研究の1つがAlavi（1984 a）によってなされている。彼は実験室の経験を報告しており、そこでは伝統的な方法とプロトタイピングの方法の両方を使って1つの情報システムを何度も開発している。開発方法は独立変数としての役目を果たす。従属変数の1つの集合は開発工程に対するユーザーの認識と態度である。これらは以下の観点から測定されている。

- システム開発者とユーザーとの間のコミュニケーションの容易さ度合い
- ユーザーが工程に参加するレベルについての満足度
- システム開発者とユーザーとの間の摩擦、あるいは衝突の認識度合い
- 開発に対するユーザーの理解度

ユーザーにはLikert型の5点方式で（1＝低、5＝高）、これらの項目に関する認識の度合いを質問している。

表５．１－１　開発工程へのユーザーの認識

	コミュニケーション	参加	衝突	理解
プロトタイピング・アプローチ	3.86	4.20	1.36	3.66
伝統的アプローチ	3.00	3.21	2.86	2.78
確　率	0.13	0.0014	0.00149	0.10

出典：Alavi (1984a) を改作。(c)1984 Association for Computing Machinery, Inc.

表５．１－１がその結果である。見出しの欄は上に示された従属変数を指す。この表の最初の２つの欄は、これら４つの変数の平均得点である。統計上のテストを使うと（Mann - Whitney U Test）、様々な欄の得点の対が偶然によるものか、あるいは２つの戦略が関心のある点で本当に異なるのかをテストすることができた。表の最後の欄は測定誤差の確率である。つまり、0.05より小さければ、その差は統計的に有効となる。プロトタイピングに参加したユーザーは、伝統的な方法で固めたユーザーと比較して、開発工程をあらゆる面でより好ましいとみなしているのは、表５．１－１から明らかである。統計的に有効な結果としては、ユーザーの参加により高い満足度を示し、また開発担当者とユーザーとの間の摩擦には明らかにより低いレベルを示している。他の２つの点についての結果は統計的に有効ではないが、プロトタイピングにより開発担当者とユーザー間のコミュニケーションが容易になっており、また開発工程へのユーザーの理解が高まっていることを示唆している。この研究の結果は、比較的サンプルの規模が小さく、調査が実験室的な性格であることに影響を受けているかもしれないが、同じ著者が報告した現場でのインタビューから得られた結果と非常によく合致している（Alavi、1984 b）。このように、プロトタイピングは開発工程に対するユーザーの態度に肯定的な影響を与えることをこの調査は明らかにしている。

情報システム利用者の満足度

開発中のシステムの機能とユーザー・インターフェイスがユーザーに及ぼす影響が大きいことにより、プロトタイピングによりユーザーと親和性のあるシステムを作り出すことが期待されることになる。この面についてもやはりAlaviによって研究がなされている。多種の中から次のもので測定がなされた。

- **完成したシステムに対する全体的な満足度**
- **システムが作成するレポートの正確性**
- **情報システムのユーザーに対する親和性**
- **システムが作り出すレポートの有益性**

表５．１－２　Mann - Whitney U Testの結果

	満足度	正確性	ユーザとの親和性	有益性
プロトタイピング・アプローチ	4.40	4.06	3.46	4.21
伝統的アプローチ	3.21	2.64	3.07	3.14
確　率	0.011	0.002	0.345	0.007

出典：Alavi (1984a) を改作。(c)1984 Association for Computing Machinery, Inc.

表５．１－２は上記の点についての平均得点とMann-Whitney U Testの結果を示したものである。システムにより大きな満足を得ようとすれば、伝統的な方法よりプロトタイピングであることをこの結果は示している。システムの出力に対する正確性と有益性については両方とも非常に高い得点を示した。この結果はLangle他（1984）の調査で確かめられており、この中で経済誌フォーチュン500の企業の投票を基に、プロトタイピングがユーザーと開発担当者の両方のシステムに対する満足度を高めることを立証している。

しかし、Alaviの研究の中で少々驚かされるのは、それぞれの戦略に従って開発されたシステムが、ユーザーとの親和性という観点からはあまり異なるところがないということである。これは、プロトタイピングが伝統的な方法よりユーザー・インターフェイスにより多くの注意を払っていることから期待のパターンとは一致しないからである。ユーザーとの親和性における違いが現われないのは、実験が実験室の性格をもつことによるものと思われる。グループは、すべて立派なシステムを作るように高い動機付けがなされており、そして従来の方法に従う側は、現在では当然と思われていることよりも、もっと徹底的なユーザーとの会話とコミュニケーションを行なった結果ということは有り得ることである。

プロトタイピングは、このように前の経験からの結果としてユーザーとシステム開発者との間で形成された不和を癒してくれる助けとなる。この要求定義のアプローチを使うことにより、開発工程に対するユーザーの態度の向上性が増し、長期的にみても同様な効果が期待できる。プロトタイピングにより、保守に必要な労力を大幅に下げることができる。

保守の負荷は、新規アプリケーションを長時間待たす大きな原因となる。プロトタイピングをより広い範囲で使用すれば、最終的にはより多くの力を新しい開発へ向けることができる。そして情報システム部門は新規の開発要求に対しもっと柔軟に、もっとすばやく対応できるようになる。

（6）プロトタイピングの境界

これまで、プロトタイピング・アプローチを利用するのは賢明であるか望ましいかを判定するための基準について個々の問題状況に応じて述べてきた。さらにプロトタイピングが可能かどうかを調べることも必要である。

5．1．2　CASE

プロトタイピングが成功するには、RDBMS、スクリーン・ペインタ、能動型／統合化データ・ディクショナリ、VHLL（すべてが1つの開発環境に統合されていることが望ましい）といった強力なCASEツールを利用できることが必須である。このようなソフトウェアの支援がなければ、プロトタイプの開発は多大な費用を要し、またユーザーの要求や希望変更への対応が遅くて受け入れられないことになる。対応が迅速であることはプロトタイピング工程の有用性に著しい影響を与える。

さらに、構造化手法および技法を支援するためにCASEの機能が求められる。分析者用ワークベンチがなければ、作業現場で構造化手法・技法を正しく使うことは現実的に不可能である。分析者用ワークベンチは、システム要求定義を迅速に、しかも頻繁に変更できることが重要視されることから、プロトタイピング工程に適用される。プロトタイピングを使用するからとて、要求定義フェイズの最終結果を正しく文書化しなくてもよいということには決してならない。どのCASEツールを本当に必要とするかは、当然、システムのどの側面をプロトタイピングするかによって決まる。もしプロトタイプがユーザー・インターフェイスに限られるなら、スクリーン・ペインタあるいはレポート・ジェネレータで十分である。しかし、データも格納しなければならないならば、データ・ディクショナリをもつDBMSが必要となる。また手法を支援する機能が常に必要とされる。

第4世代環境にも分析者用ワークベンチにもスクリーン・ペインタ、レポート・ライタ、ダイアログ・マネージャ、それに作図機能と開発データの一貫性を維持する機能が備わっている。一般に、第4世代環境に見られるスクリーン・ペインタ、レポート・ライタ、ダイアログ・マネージャは、分析者用ワークベンチのそれよりも質的に優れている。一方、分析者用ワークベンチは、構造化手法・技法を支援する点においてはより多くの機能をもっており、この状況は急激に変化している。つまり、第4世代言語の提供業者は、プロジェクトの初期フェイズを支援するための機能を自社のツール群に加えつつあるからであり、また分析者用ワークベンチの提供業者は（簡単な）DBMSと第4世代言語を含むプロトタイプ作成のためのツールを新たに加えつつあるからである。さらに分析者用ワークベンチとCOBOLジェネレータといったツール間のインターフェイスもできつつある。

プロトタイピングで使うツールを選択する場合に最も重要な基準は、本番システム（の一部）のベースとしてのプロトタイプをそのツールで構築できることである。一般に、プロトタイプも本番システムも両方実現するために適した技術的環境（たとえばADABAS、Natural、Predictなど）を選択することが賢明である。あるいはまた、プロトタイプをベースに、別の技術的環境で実行できる本番システムの一部を生成するツールを提供するプロトタイピング環境を選択することもできる。

5．1．3　システムの性格

プロトタイピングは適切なツールなしでは不可能である。このことは、要求定義にこのアプローチを適用できるシステムの数が著しく限られることを意味する。というのも、現時点で購入できるCASEツールの大部分は、データ指向のユーザー・インターフェイスに力点を置く強力なオンラインの性格をもつアプリケーションのプロトタイピングに特に向いているからである。

力点が、データそれ自体でなくデータの処理、つまりアルゴリズムに置かれる場合、プロトタイピング工程を支援する現在の言語およびツールは一般に要求を満たしていない。複雑な操作を非手続きの方法で正確に記述することは普通はできない。処理の結果は処理自体の言葉でしか表わせないこと

もある。言い換えれば、結果に到達するには、実行しなければならないステップを正確に示すことによってのみ記述できるのである。プロトタイピングを検討する場合、データの収集とその保守の機能（挿入・修正・削除）は、開発すべきシステムの主要な構成要素でなくてはならない。

現在の第４世代開発環境には、ユーザー・インターフェイスを迅速に定義し、実現するツールが見受けられる。開発環境の中には、データベースの構造が定義された後、データベースからのデータを抽出したり保守したりするのに必要な画面を生成するものがある。この簡単なユーザー・インターフェイスで初期のプロトタイプの一部を作ることができ、そして後の繰返しの中で画面定義機能の助けを借りて迅速に変更を加えることもできる。

プロトタイピングがなぜユーザーとシステム間のインターフェイスを重要視するシステムに特に向いているのか、その最大の理由は、多分実用モデルがプロトタイピングの作業期間コミュニケーションの方法として役に立つからであろう。システム開発者とユーザーはシステムの外部表示について議論し、それによって外部レベルだけでなく概念レベルに対しても理解を深めることになる。システムとユーザー間のインターフェイスが重要な構成要素ではない情報システムにとって、このインターフェイスを議論の特に重要な伝達手段として使用する要求定義は適切なアプローチではない。

決定階層による分類

プロトタイピングの適用は、開発するシステムのタイプに非常に関係すると文献の中で述べられることがある。一般に用いられているシステムの類型学は組織内で認識されている決定階層（図５．１－２）を基にしている。以下にそのカテゴリを示す。

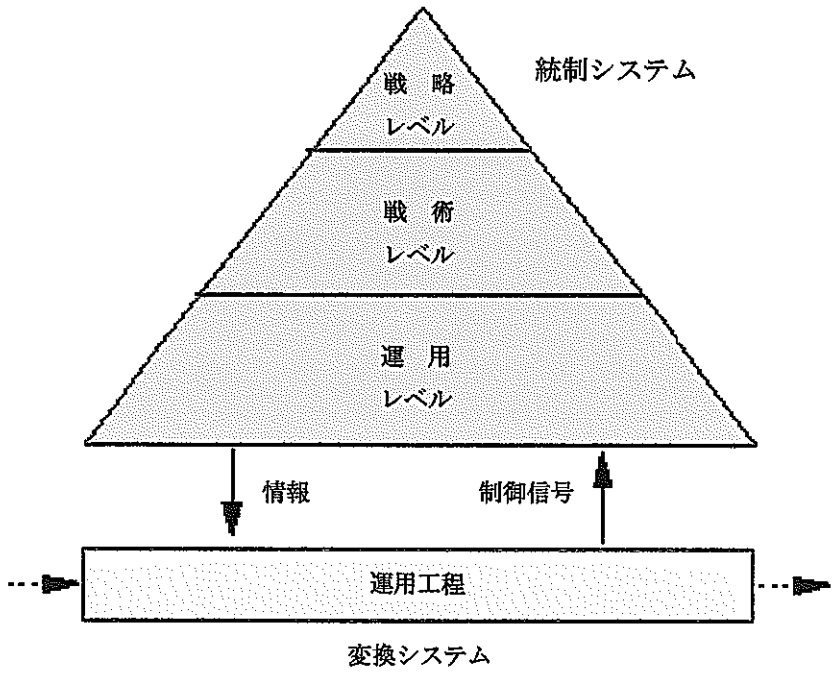

図５．１－２　組織内のシステム決定階層

・運用工程（変換工程とも呼ばれる）を支援する、あるいは自動化するシステム

・運用工程を制御するのに必要な運用管理を支援するシステム
・組織の戦術層で運用工程計画化を支援するシステム
・戦略的経営を支援するシステム

決定ピラミッドを基にすると、これらの意志決定問題は比較的容易に形式化できるし自動化できる。変換工程で必要な情報は安定しており、定型化されており、その範囲は決まっている。上位層になると、決定の範囲（組織の決定の結果）は広がり、決定者は組織外からの情報（経済発展、競争相手などについて）に左右されるようになる。戦略層になると、要求は組織の下部での事態を一目で把握できる高度に圧縮され選び抜かれたデータとなる。

戦略・戦術・運用管理間の違いは、このモデルのように明確でないことが多いが、これによって組織の情報インフラを理解しやすく管理しやすい部分に分割するのに役に立つ。しかし、プロトタイピングを持定の状況で使用するかどうかを判断するために、この分類が有用であるかは疑問である。どの組織のどの決定階層においても、情報システムが満たさねばならない要求は明確ではない。決定ピラミッドの底辺では、情報問題は構造化される傾向にある。つまり、システム要求にまつわる不確実性は、ユーザー・インターフェイスの周囲に集まることになる。ピラミッドが高くなると、概念レベルで、特に決定者がシステム内にもちたいと思う機能について確実性が失われて行く。組織のすべての階層においては、たぶん戦術、戦略層のほうが多いと思われるが、要求と要望については不確実であり、そこでプロトタイピングは重要な役割を演じることができる。

要求定義のアプローチを選ぶために、この分類の利用に限りがあるのは、いろいろな著者が決定基準としてシステム・タイプを採用していながら、逆のことを勧めている事実からも明らかである。したがって、Boar（1982）は意志決定支援システムにプロトタイピングを利用するのに反対しており、このアプローチは構造化問題に取って置くべきだと提案している。一方、Naumann and Jenkins（1982）は、プロトタイピング・アプローチに合うのは明らかに意志決定支援システムであると述べている。もちろん、このような逆のことが勧められるのは多くの著者が“プロトタイピング”、あるいは“意志決定支援”をそれぞれ違って解釈していることによる。

事務情報システム 対 技術情報システム

要求定義アプローチを選択するのにしばしば用いられるもう１つの類型学は、事務（あるいは経営）情報システムと技術情報システムとを分けることである。事務工程を支援する情報システムは大量のデータを管理する比較的単純な処理形態が特徴となっている。技術情報システムでは、リアルタイムの側面がしばしば重要で、処理をモデル化することがデータ構造をモデル化するより、はるかに重要である。組込み式ソフトウェアの場合さらに問題があり、いったんこのソフトウェアが特定のマシン上で開発されると、別のハードウェアに移植せねばならぬ必要性が出てくる。“技術的”なカテゴリに含まれるソフトウェアのもう１つ別の例として、線形プログラミングのような数学的なソフトウェアがある。技術的な情報システムをプロトタイピングするためのツールは現在ほとんどない。したがって、プロトタイピングのアプローチは、このようなシステムには現在適していないといわざるをえない。

技術情報システムのユーザー・インターフェイスは、ある程度無視されることがよくある。ユーザーとマシン間のインターフェイスが技術情報システムの重要な構成要素であれば、この分野について

より深く検討するため、このインターフェイスをプロトタイプすることは役立つことになろう。

5．1．4　システムの規模

プロトタイピングは、大規模で複雑なシステム開発には役に立たないという誤解が広く流布している。この誤解の主な原因は、多分プロトタイピングに皆がいだくイメージが型にはまったもので、このイメージとは、システム開発者とユーザーが画面の前に一緒に座り完全なシステムをすばやく作成するというものである。この解釈によると、どんな簡単なアプリケーションでもプロトタイピングに従うことはできなくなる。しかしながら、本項で唱えるように大規模なアプリケーションがプロトタイピング・アプローチに適さないという理由はない。その主な特徴はよく知られた構造化手法と技法を繰り返し行なうことと、ヒューリスティックなプロトタイピングとの工程と組み合わせるところにある。この方法が、問題の分析とその解決策の構造化が大いに注目を浴びているのである。

大規模アプリケーションのプロトタイピングが考慮に値すると思われるもう1つの要因は、第4世代開発環境であれば大規模なアプリケーションですら迅速に実現できるということである。実際、現在あるツールを使えば、25から35の画面をもち40から60のオンライン・プログラムから成る完全に文書化された初期のプロトタイプを4週から6週間で作成することは可能なのである（Boar、1984）。

以上に述べたように、情報システムを開発する場合、同じ方法で全システムを開発する必要はない。大規模システムは一般に非常に性質の異なるユーザーとサブシステムから成っている。このような場合、プロトタイピングはこのアプローチに適した（この章で述べた基準に従い）システムのその部分に対して適用することになる。

5．1．5　システム開発者とシステムの利用者

ユーザーの能力と動機付け

プロトタイピングを用いてシステム要求定義を行なうことができるのは、以下のようなユーザーがそのモデルを評価できる場合だけである。

- **全ユーザーを代表しているとみなされるユーザー**
- **システム要求定義に関して決定権をもっているユーザー**
- **要求定義について賢明な判断ができる能力をもっているユーザー**

プロトタイピングは、要求定義工程でユーザーの中心的役割に非常に大きな力点を置く。多分、プロトタイピングが成功するための最も重要な前提条件は、ユーザーが自分に与えられた役目を喜んで果たし、またその能力も備えていることである。ユーザーは自分自身だけでなく、自分の仲間にも受け入れられることができる人間環境工学的に健全なユーザー・インターフェイスに到達できる能力がなくてはならない。そのシステムが機能的な観点から完全であり、またシステムの導入後もユーザー・インターフェイスに何ら修正を必要としないことを確認するのはユーザーの責任である。ユーザーは誰よりも問題の分野をよく知っているが、どのユーザーもシステム要求について賢明な判断が行なえる能力をもっているわけではない。結局のところ、ユーザーは専門の分析者でも設計者でもないのである。プロトタイピングの進行を正しい線に沿って導いて行くには、システム開発者の役目について相当の専門的能力が求められる。開発中の情報システムを今後使うことになるユーザーが、そのシス

テムが備えるべき機能や形態について様々な相反する意見をもっているとすれば、プロトタイピング・アプローチに従うのは意味がない。プロトタイピングは構造的な意見の相違に対して万能策とはなりえない。しかし、もし観点の相違がその性格上構造的なものでなく、システムの機能と外見の不確実性によるものならば、プロトタイプの開発はやはり考慮すべきである。このプロトタイプはユーザー間での議論のベースとして役立つし、ユーザーはこの方法で共通の立場に立つよう努力すべきである。

将来のシステムを使用するユーザー代表者が見つからないことについては、いくつかの状況が考えられる。誰がそのシステムの今後のユーザーか正確にわかっていない、あるいは非常に大多数のユーザーが関係する場合はこの状況が起こりうる。たとえば、アプリケーション・パッケージの開発の場合である。

プロトタイピングはこの工程にかかわるユーザーの時間と注意力を大いに必要とする。ユーザーはこのための準備をしなければならない。ユーザーはモデルを評価し、再調査するために進んで時間を割り当てねばならず、管理職側はそのことに時間を割けるようにしなければならない。誰もそのプロトタイプの評価に手が掛けられないなると、プロトタイピングの工程は軋んで停止ということになる。

アプリケーション・システム開発者は、情報システムの導入に反対するユーザーがプロトタイピング工程の使い方を誤り、重大な遅れを引き起こすことに常に注意しなければならない。この徴候としては要求や要望が常に変わり、重要なときにユーザーは時間が取れないといったことが挙げられる。このような場合、繰返しの工程が実りある結果を得るまでに長い時間がかかることになる。そこでシステム開発者は反対の原因を追跡し分析し、その改善策模索に努めるほうがよいということになる。プロトタイピングは、この工程で積極的な役割を積極的に引き受け、しかもその能力がある意識の高いユーザーを必要とする。

システム開発者の能力と動機付け

プロトタイピングは情報分析者に多くのことを求める。分析者は、もちろん立派な分析能力、構造化分析と技法についての知識、正しくコミュニケーションできる能力、そしてなるべくならアプリケーション分野の専門知識をもつべきである。情報分析者の経験的な技能に加え、プロトタイピングには現在使えるソフトウェア・ツールで何ができ、何ができないかについて堅固な知識が必要である。プロトタイプ作成者は現在使えるものの中から最適のソフトウェア・ツールを選び出し、それを使って完全に構造化されたプロトタイプをすばやく作り上げることができなければならない。

プロトタイプ作成者はまた、技術的な設計フェイズで持ち上がる問題、たとえば本番システムの一部となるべきプロトタイプの部分を伝統的な製造ラインに従い並行して開発されたサブシステムと、あるいはすでに運用されているものと統合しなければならないといったことを承知しておかねばならない。特に重要なことは、プロトタイプでもモデル化したものが本番システムで実際に実現できることを保証するため、目標とする環境のどのような制約についてもプロトタイプ開発者は承知しておかねばならない。

したがって、伝統的な工程では、いくつかの機能分野（情報分析者、技術的なシステム設計者、プログラマ）に分かれていた技能をプロトタイピングは混ぜ合わせて１人の人に求めることになる。しかし、もしそのような人がいなければ、情報分析者と専門の設計者からなるプロトタイピングのチームを導入することを考えることになるかもしれない。

プロトタイピングでは、ユーザーに関して肯定的な態度がシステム開発者に求められる。過去、ユーザーは開発工程における邪魔な付属物とみなされたのは稀なことではなかった。設計と実現工程が複雑であるということは、情報問題を明らかにすることにではなく、開発の後の工程に力を注ぐことを意味していた。

ユーザーは自分が何を望んでいるのかを表現できないし、まして未知識ことを前提にシステム開発者は問題の本質について、またその解決について自分自身の仮説を立てていたが、この状況は変わりつつある。システム部門は、他の部門へサービスを提供すべきだということが広く受け入れられるようになってきた。ユーザーの重要な役割は認識されているというのも、ユーザーだけが本当の情報がどこに存在するかを知っており、また最終的にそのシステムとともに働くことになるからである。この認識は、ユーザーの参加が基本であるアプリケーション・システム開発環境に対する戦略となる。つまり、これこそプロトタイピングの主要な目的である。これらの戦略に沿って仕事をうまく進めるためには、システム開発者は“ユーザー指向”でなくてはならない。開発者は、これらのアプローチを特徴付ける非常に密度の濃いユーザーとの会話に肯定的な態度を取るべきであり、またユーザーがかなりの範囲まで開発工程を管理できるように準備をしておかなくてはならない。プロトタイピングを用いる場合、プロトタイプ作成者は、このアプローチの特徴であるシステム要求が常に変わることを肯定的に受け止めねばならないし、さらにそれらの変更を勧めることも行わなければならない。プロトタイプを繰り返し変更するには、システム開発者の高い意識付けが必要である（Podolsky、1977）。

現在、情報システム開発者には開発戦略、構造化手法・技法および自由に使えるソフトウェア・ツールが用意されているであろう。新たな情報問題が起こると、このコレクションの中から賢明な選択をすることができる。今日、情報システム開発に携っている専門家は様々な戦略と手法に存在する基本的な相違を詳しく知っていることが期待されている。専門のシステム開発者は、自分が自由に使えるツールを多く保持していなくてはならず、またどんな問題にも状況に応じた正しいツールを選び、使うことができなくてはならない。

プロジェクトの最初にプロジェクト・リーダはこの章で述べた基準を基に、要求定義フェイズに従来の方法かプロトタイピングのアプローチかどちらを採用するか決めなければならない。

費用 対 便益

プロトタイプを本番システムのベースとして残すことができれば、経済的な優位性はほとんど常にプロトタイピング・アプローチとなる。もしプロトタイプを捨てるのであれば、プロトタイピング・アプローチは、いまから開発する情報システムに対してユーザーの要求が不確実である場合にのみにすべきである。この不確実性は以下の事柄によって決まる。

- **支援を必要とする仕事から要求を演繹できるレベル**
- **ユーザーおよび分析者の知識と経験のレベル**
- **コミュニケーション問題の危険度**

プロトタイピングは多くの無形の利益をもたらすが、この無形の利益は伝統的方法かプロトタイピング・アプローチかどちらかを選択する場合考慮すべきである。この中で最も重要なことは、プロトタイピングが開発工程と情報システム部門の両方の観点からユーザーの役割により積極的な姿勢をも

たらし、そして情報システムに対するユーザーの満足度を高めることである。

自動化ツールの可用性

CASEツール（ワークベンチ、コード・ジェネレータ、VHLLなど）は開発工程の効率を高め、プロトタイプには要求や要望の変更に対し迅速で柔軟な対応を可能にしている。

CASEツールの最大の見返りは、プロトタイピング・アプローチがこれらのツールを使うことによって得られるということである。したがって、プロトタイピングとCASEツールは連携しながら進むことになる。

開発するシステムのタイプ

現在使用できる第4世代開発支援には制約があることから、プロトタイピングは、データ指向のユーザー・インターフェイスを非常に重視するオンラインの性格を強くもったアプリケーションに特に適している。

データとデータ構造の代わりに、データが従わねばならぬ処理に力点を置くのであれば、適当なプロトタイピングのツールはまだ広く使えるようにはなっていない。プロトタイピング・アプローチは現在経営情報システムには十分応えることができるが、ツールの欠如のため、技術的なアプリケーションにはそれほど、あるいはまったく応えることができない。

情報システムは組織上の意志決定階層のためであるが、その階層を基にした選択基準は、要求についての不確実さが組織のあらゆる階層に散在するためほとんど価値はない。

ユーザーとシステム開発者

プロトタイピングに参加するユーザーは、ユーザー社会の代表者であるべきで、システム要求定義についての決定ができ、また積極的にその決定をしなければならない。システム開発者の側では、専門の“プロトタイプ作成者”、つまり従来のシステム開発では何人かの専門家が分担したような技能を合わせもつ“ユーザー指向”の人が工程を管理しなければならない。

プロトタイピングを導入し利用しようとすると、様々な問題が持ち上がる。プロトタイピングの目的を十分知らされていないユーザーは、自分達のニーズを十分満たしていない最初のプロトタイプに懐疑的な反応を示す。つまり、プロトタイプを使うのは非常に難しく、繰返し作業を迅速に行なうことなどできはしない。また長い骨の折れる開発工程を経た後では、本番のシステムが待てなくなり、プロトタイプを手放そうとはしなくなるといったことである。プロトタイピングを導入し利用する間に起こる問題はほとんど、ユーザーおよび開発担当者双方がその概念について十分な知識をもっておらず、一貫性のあるフェイズごとのアプローチについていけないことに起因する。

別の違った状況も想像できる。この場合、開発担当者はユーザーと緊密に協力し合ってシステムの実用モデルを何度も繰り返し作成することから、ユーザーとのコミュニケーションは良くなり、要求定義は向上し、ユーザーの役割に対する考え方もさらに好ましいものとなるといったことである。

５．１．６　システム開発に潜在的な問題

（１）開発担当者間における開発要求の誤解

開発担当者がイメージしているプロトタイピングは、彼らが慣れ親しんでいるアプローチと非常に違っているということである。彼らが積み重ねてきた知識や経験が、要求定義への新しいアプローチの到来によって突然価値がなくなってしまうことを懸念している。プロトタイピングは彼らの身分と地位に対する脅威と見えるのである。経験ある開発担当者がプロトタイピングを拒絶することになることにもなる。彼らはこのアプローチに非常に懐疑的になり、それを妨害しようとすらする。そこで、変化に対して当然起こる抵抗は克服されねばならない。プロトタイピングの導入によって起きる自動化社会への抵抗は、概念を正しく理解していないことによることが多い。教育と訓練により、この問題に対し機先を制することができる。プロトタイピングは革命的な新しい開発方法ではなく、むしろ現在の要求定義方法の改良であることを明らかにしなければならない。構造化手法と技法で培われた技能は、プロトタイピングの出現によって捨て去られることはない。逆に、このアプローチとうまく組み合わされ、本当のプロトタイピングの基本的な部分となる。プロトタイピングは、情報分析者が自由に使えるひと揃いの戦略、手法、技法に付け加えられるべきもので、それらの代わりになるものではない。

このことは、もちろんプロトタイピングが何も変えはしないといっているのではない。プロトタイピングを適用しようとすれば、開発担当者は新しい技能とツール（たとえば第４世代ツール）を修得しなければならない。さらに開発工程と、そこでユーザーが果たす役割の両方についての考えを変えなければならない。開発担当者がプロトタイピングで成功を収めたいと熱望するならば、その変化に備えなければならない。しかし、（プロトタイピングに関する多くの出版物からすると）変化はまず最初に予想するほど大きくはない。したがって、教育と訓練が十分練られていれば、多くの抵抗の源を克服することができる。

（２）管理者間における開発要求の誤解

プロジェクト予算に責任をもつ管理者は、プロトタイピングとは何度もシステムを構築しては改訂し、あるいは置き換え、最後には時間とお金の立派な無駄使いだということで捨ててしまう、といった方法（ただの単純化された実用モデルであるが）とみなすかもしれない。逆に費用の膨張効果だけを見る管理者とは対照的に、プロトタイピングにより開発工程の能率が相当向上し、したがって費用も低減できると固く信じている管理者もいる。プロトタイピングが純粋に経済的な観点から正当化できるかどうかは、主としてプロトタイプ（の一部）が本番システムのベースとして使える見込みがあるかないか、またシステムへの要求にどれだけ不確実性が存在するかその程度にもよる。さらに、プロトタイピングは目に見えないが様々な現実的な利益をもたらし、これらも考慮に入れるべきである。プロトタイピングが効率を上げることに結び付くか否かについては疑問がある。プロトタイピングの主要な目的は開発プロジェクトの有効性を高めることである。つまり、多くの人が認識できる効率の向上は、実際には第４世代言語の使用によるものである。担当の管理者は、確固たる根拠の基にこのアプローチの採用を決定するため、プロトタイピングそれ自体の確かな費用と利益について理解しておくべきである。

（3）ユーザー間における開発要求の誤解

プロトタイピングへの誤った期待がユーザーの間にあるが、それは大部分プロトタイピングの目的と効果について十分な知識が欠如していることに起因している。この不十分な知識のためプロトタイピングを選んだり、拒んだり、またこのアプローチに失望したりすることになる。

プロトタイプの本番システムへの適合性

適切な手引きがない限り、経験のないユーザーは“最終的な”プロトタイプを運用可能なシステムとして期待しがちである。ユーザーはなぜ本番用のシステムが使えるまでに、かなりの時間待たねばならないか多分はっきりと理解できない。

ユーザーは既に最終的なプロトタイプに満足しており、またあらゆる面からそれは完成しているとみなしてしまうことがよくある。このことは開発担当者にそのプロジェクトを終わらせ、他のシステムに目を向けさせる圧力になる。これは実際、しばしば起こる問題である（EDPAnalyzer、1984）。

早まってプロトタイプ・システムをユーザーに渡すと、望ましくない結果になる。品質改良フェイズを省略すると、本番システムを成功裏に導入し、使用するのに基本となる多くの要因が無視されることになる。以下にその要因を挙げる。

- 性能
- 例外および誤り状況とその処理方法
- バックアップと回復
- セキュリティ
- 信頼性
- 保守性
- テスト
- ユーザーおよび技術支援のための文書
- 運用コスト

これらの面に適切な考慮を払わずに本番システムとしてプロトタイプを使うと、ユーザーは当初満足するがしばらくすると不満をいだくようになることが非常によく起こる。その失敗は、普通このプロジェクトで使われるアプローチに原因がある。

プロジェクトの最初の時点で、プロトタイプの限界をユーザーに示すことはプロトタイプ作成者の責任である。プロトタイピングの作業が完了した後、特にプロトタイプを捨てる場合、本番のシステムが使えるまでにはやはり長い時間を要することを明確とさせておかねばならない。まずプロトタイプをシステムの要求定義とみなし、システム自体であるとみなすべきではない。

プロトタイピング工程の運営

プロトタイピングに失望するもう１つの原因は、その手順と目的が十分理解されていないことにある。プロトタイピングの目的は、開発担当者が実用モデルを使ってユーザーの要求と要望を深く理解できるようにすることにあることをユーザーに明らかにしておかなければならない。もし明らかにしておかないと、ユーザーは初期のプロトタイプと期待しているシステムとがほとんど一致していない

ことから、好ましくない態度をとるかもしれない。この工程にユーザーが参加しようとすれば、ユーザーはプロトタイピングの間システムのある選別された側面、あるいはサブシステムだけに重点を置き、そして工程の一部としてプロトタイプを“あれこれと検討”するであろう。むしろ、必ずしなければならないことを認識する必要がある。プロトタイプは迅速に容易に変更できることから、経験のないユーザーが本番システムも同じ早さで変更できるという誤った結論を引き出すことがある（Dearnley and Mayhew、1983）。

これはプロトタイプと運用システムとの間の相違を明確にしなければならないもう1つの理由である。本番システムがプロトタイプから格上げされたものであっても、第4世代言語で書かれていたとしても、一般的に本番システムはプロトタイプより何倍も柔軟性に劣ることになる。

ユーザー参加に求められる時間

ユーザーはプロトタイピング工程で重要な役割を果たし、作業を行ない、そして参加するには、労力あるいは時間が必要であることを十分認識しておかねばならない。

プロトタイピングは、ユーザーに積極的に参加しようという意識があり、システムの要求を決定する機会も能力もあり、さらにプロトタイプを評価するのに十分な時間がある場合にのみ価値がある。さらにユーザーの管理者はプロトタイピングへの参加により、従業員の通常の作業がかなりの間中断されることを認めなければならない。

この落し穴は、当事者を正しく教育し訓練することによって避けることができる。したがって、プロトタイピングを考える場合、要求定義に対するこのアプローチについてできるだけ早い時期に知らせることは賢明なことで、これによって関係者にはすべて長所短所が十分伝わることになる。

（4）不十分な要求分析

プロトタイピングを不注意に使うと、本当の情報問題を明らかにできず、単に症状を取り去るだけのシステムを開発することになる。この問題の原因についていくつかの重要な点を以下に述べる。

まず急いで解決しようとする

まず情報問題を迅速に解決することをプロトタイピングでは強調しすぎるために、プロトタイプ作成者の誤解を招き、問題を十分理解する前にプロトタイプ作成に熱中してしまうことになる。ユーザーが自分の要求しているものを既に熟知していれば、このようなプロトタイプはすぐに捨て去り、すばやく最終的なプロトタイプに集中できることから、必ずしもこれは困難ではない。しかし、ユーザーが問題の性格について、またその解決策がどのような形になるかについて確信がなく、さらにシステム開発者の影響を受ける（多分無意識に）ような場合は、また別の問題である。問題解決の作業中、人は得てして受けた提案の解決策に安住しようとする。この事象は、定着化（anchoring）といわれることがある（Davis、1982）。当事者が、自分が望んでいるものに対し明確な考えをもっていないならば、根本的に異なる解決策が提案されることもない。

最初のプロトタイプが本当の問題を解決するには不十分であるならば、最終的に完成したシステムは結局間違った問題を解くことになるか、あるいは（最良の場合でも）満足できる解決に達するため何度も必要以上に繰返し作業を行なうことになるか、そのいずれかとなる。このことを図5．1－3

に示した。

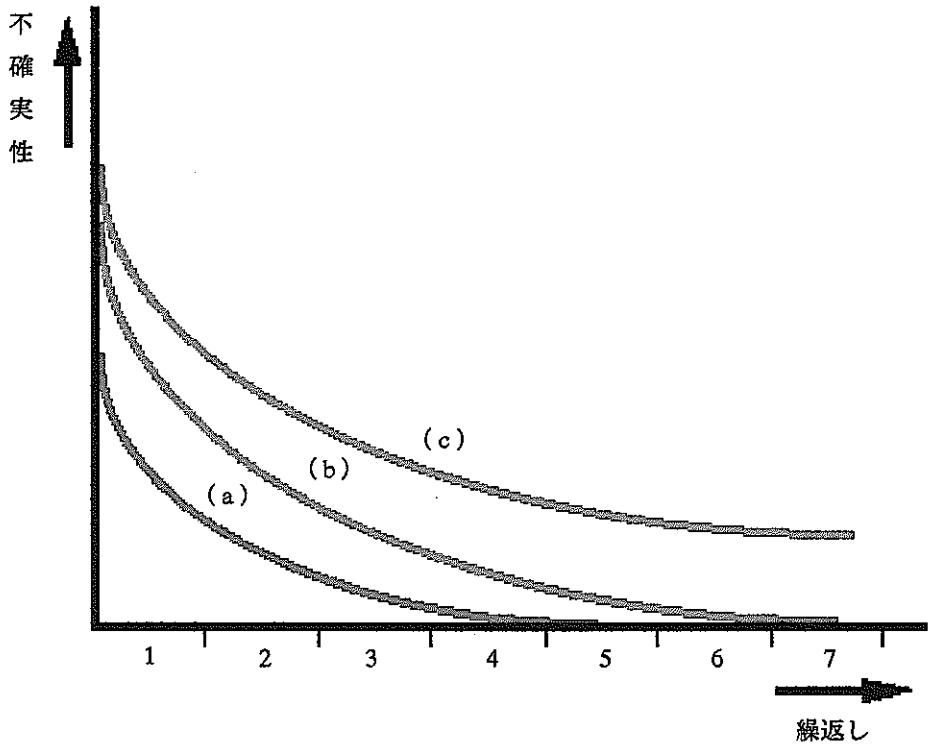

図５．１－３　不十分な問題分析のリスク

(a) の場合、十分な問題分析が最初のプロトタイプを作る前になされており、最初の繰返しの作業中に問題の不確実性は急激に低減している。プロトタイピングの作業がさらに進むと、繰返しの度に行なう変更の数は少なくなる。プロトタイピングの作業は、プロトタイプにもはや著しい変化がなくなった時点で終りとなるが、それにはユーザーの本当の要求がその時点で明確になったとの前提が必要になる。この前提は (b) の場合にも当てはまる。しかし、問題の分析が不十分なため、(b) に現われる不確実性はプロトタイプ開発時に相当あり、その結果繰返し作業の回数は必要以上に多くなっている。(c) の場合、本当の問題が現実にまだ明らかになっていない時点で、もうプロトタイプは根本的な変化を受け付けなくなっている。この場合、プロトタイピングは正しい完全な要求定義に結び付くことにはならない。

細部に深入りするプロトタイピングは、開発中のシステムの細部にまで注意を払うことができるため、システム全体の見通しを見失うことがある。ユーザーは、その時点で理解できるプロトタイプの部分についてあら捜しをしがちであり、その実大規模システムに関する限り、ユーザーはシステム内のそのような部分については全体として批評することはほとんどない。

このことは、プロトタイプを評価期間に完全に評価するにはあまりにもプロトタイプの範囲が広すぎたり、またプロトタイプ作成者が明確な構造（たとえば、階層的に構成されたメニューを使って）を利用しなかった場合に当てはまる。この問題は“おまけ”をシステムに欠くことのできないものと

見てしまうユーザーの性癖によるものであり、その結果、繰返しの工程は長くなり、本番システムは不必要に高価になり、主要な問題から注意が逸れてしまう危険性が出てくる。

この原因は、プロトタイプ作成者がすばやく、また安易に新規の変更要求・希望をシステムに取り入れてしまうことにある。このような状況を排除するには、開発担当者が適宜な平衡感覚をもつ必要がある。新たな要求や希望を見境いなく受け入れてはいけない。あらゆる折にふれ、ユーザーと開発担当者はともに新たな機能が有用か、またそれらの機能を実現するため必要となる作業を正当化できるかどうかを更に検討を加えなければならない。細部にわたる配慮と全体に対する配慮との間に均衡のとれた解決策を見い出すのはプロトタイプ作成者の責務である。

インターフェイスへの配慮が不十分である

開発中のシステムと他のシステムとのインターフェイスを最初概念的に分析する間に十分な注意を払わないと、統合フェイズで回避できない問題に当面する。Boehm他による比較調査（1984）によると、プロトタイピングで開発したシステムは、従来の方法に従って開発したシステムよりも統合するのがより困難だったようである。この根本的な原因は、インターフェイス仕様の欠如にあった。この種の問題は、分析フェイズの間で開発するシステムと顧客あるいは情報提供者との間の情報の流れを有効な情報分析手法の助けを借りて（たとえば、Chen（1983）のエンティティ・リレーションシップ・ダイアグラム法）調査することで回避できる。大規模なあるいは複雑なシステムの場合、従来のトップダウンの作業、すなわち工程分析手法（たとえば、構造化分析と構造化設計）を使って、かなり自主独立した管理可能なサブシステムにすることができる。その後再び、様々なサブシステム間を流れる情報を分析する必要がある。

プロトタイピングの目標は、システム（の側面）に関するユーザーの要求と要望について洞察を深めることにある。もし入力が現在の情報システムに向けられたものであるなら、あるいは出力がそこから来るものであるなら、どんな情報が作られるべきか、または受け入れられるべきかは既にわかっていることである。プロトタイピングの役割はそこではない。つまり、インターフェイスの分析によってのみ、統合の問題は回避できる。プロトタイピングは、情報システムとユーザーとのインターフェイスに対してのみ適用でき、他のシステムとのインターフェイスには適用できない。

問題分析と問題解決が混同されている

従来の情報システム開発方法は、問題指向活動と解決指向活動とを厳密に区別することに断固たる力点を置いている。問題指向の活動は、最終的にモデルの中で見つけ出した問題を解決するために情報システムは何をし、何を備えていなければならないかを示すことになる。つまり、ユーザーの問題とニーズに焦点が集められる。解決指向の活動は、ユーザーの様々な要求や要望を満たすシステム（自動化されていようがいまいが）をどのように実現するかその方法を確立することにある。解決策を実行するには、様々なソフトウェア・ツールの可能性と制約は重要になってくる。問題分析と問題解決とを厳密に区別する理由をいくつか次に紹介する。

a．要求定義は、決して現在の技術的環境（ハードウェア、オペレーティング・ソフトウェア、開発ツール、その他同種のもの）の制約によって影響を受けるべきでない。要求定義は、情報シ

ステムが問題解決のため満たさねばならない要求を厳密に表現すべきである。機能的要求のほんのいくつかだけがシステム設計に移されることになれば、問題は完全に解決されないことは論理的な帰結である。この線に沿って考えれば、ハードウェアの構成は定義された要求から直接派生すべきもので、要求定義フェイズでは重要ではない。

b．要求定義は、システムが実現される技術的環境に無関係であり、たとえその環境が変わったとしても相変わらず有効でなくてはならない。要求定義は、論理的にはどのハードウェアでもシステムの設計と実現化の確たる出発点となる。つまり、要求定義はいわば移植可能なのである。

c．問題の分析とその解決策を厳密に分ければ両方の面が十分考慮される保証となり、また問題を定義する前に解決方法を考えるといったことはなくなる。情報分析者は、要求上の問題だけに力点を置けることになる。そのことにより、結果の要求定義は設計者にとっての確固たる出発点となる。要求定義による枠組みの中で、設計者は様々な解決方法を自由に考え、最高のものを選ぶことができる。

d．システムの利用者は、技術的な事柄や技術的な細部について一般に興味を示さない。実際、システムがどのように作られようが彼らにはまったく興味がないのである。彼らは設計と実現化の工程をブラックボックスとして見たいのであり、入力は要求定義であり、出力はこれらの要求を満たす本番システムなのである。

プロトタイピングは、一見したところ問題分析と問題解決とを厳密に分けるのをより難しくしている。結局、プロトタイピングは要求定義のフェイズに実用モデル、つまり問題に対する解決策を持ち出すことになる。このことは、プロトタイピングが定義と設計の厳密な分別による利点をすべて無にしてしまうのではないかと多くの人を恐れさせることになる。

この方法をより深く調べると、それは必ずしも事実とはいえないことが明らかになる。プロトタイピングの間、問題指向と解決指向の活動はかなり厳密に分けられる。もちろん、プロトタイプを実現するのに用いるツールの制限や特性が要求定義に影響を与えることは事実である。

しかし、ユーザーがプロトタイプ（とそれに関する要求定義）を受け入れることについて最終決定権をもっていることから、それは問題ではない。結局、もしユーザーが完全に、そして十分な理解の基にシステム要求定義を受け入れるならば、それがどのように達成されようが大した問題ではない。

定義と設計を厳密に分けることは理想的ではあるが、この考えをあまりに強調するのは実際非現実的であることに気をつけるべきである。組織は、一般に持定のハードウェアとシステム・ソフトウェア戦略に委ねられている。この戦略に対するどんな変更も歓迎されないし、また高価につく。あらゆる機能的要求や好みには定価札が張り付けられているのである。要求定義段階に対する現実的なアプローチは、初期の段階で技術的な可能性とコストの両方を考慮に入れることである。コストと利益の比較により、要求が経済的に見合うものか、そしてその要求をシステム要求定義に含めるべきかどうかを明らかにしてくれる。このように、ある程度分析と設計の工程を混ぜ合わせることは好都合なことすらありうる。プロトタイピングにより、開発者は機能を実現するために必要となる作業を現実的に眺めることができ、また初期の段階で機能を実現できることを確かめることができる。

5．1．7　プロトタイピング手法の導入

従来の要求定義方法に代わるものとしてプロトタイピングを受け入れるかどうかは、プロトタイピングを導入するその方法と非常に関係がある。プロトタイピングが確実に失敗する１つの方法は、十分な準備もしないで偶然うまく行きそうな何処かのプロジェクトにそれを使い始めることである。このような環境で不運な経験をした人は、このアプローチに敵意をもつことは確かである。組織にプロトタイピングを導入するには準備や計画に十分気をつけ、そしてうまく管理を行なわねばならない。

大々的に実施する前に、懐疑論者にはその有用性とプロトタイピングの価値を認めてもらわねばならない。理論上の学術論文だけでは不十分で、プロトタイピングの価値は実際のテストで成功した後、初めて受け入れられるのである。このためパイロット・プロジェクトは必須であり、開始する前に正しい初期条件を設定しておかねばならない。

（1）プロジェクトの準備

順調に、そして成功するプロトタイピングを導入できるか否かは次の要因に左右される。

- **要求定義用のプロトタイピングに対する管理者の態度**
- **導入に伴う教育と活性化活動**
- **適切なツールが使えること**

パイロット・プロジェクトでこのアプローチを実際に始める前に、これらの重要な成功要因それぞれに注意を払わねばならない。

管理者による支持

どんな改革でも組織で成功を収めるには前提条件が必要である。それは非常に高い意識をもって現状を変革し推進せねばならないという管理者の積極的な態度である。もし管理者層からの支持、公約、献身がないとなれば、プロトタイピング導入のどんな試みも暗い運命に遭遇することになる。

優秀な情報システム部門が問題に直面すると、その管理者は喜んで問題を解決しようとするのが一般的である。保守作業が大変で、またアプリケーションのバックログがたまることは、現在の作業方法が適切でないことを十分に示している。何かを変えねばならないことは管理者にとって明らかである。しかしながら、この問題に取り組む最良の方法、最も有効な手段は明らかでないことが多い。この点については、４つの面を、つまりエンドユーザー・コンピューティング、アプリケーション・パッケージ、CASEツール、プロトタイピングをどう組み合わせて適材すればよいかを述べておいた。責任ある管理者の支持と約束が得られるのは、彼らがプロトタイピングについて十分精通したときだけ可能となる。このアプローチについての誤解はありうる。十分丁寧にその概念と背景を説明できれば、最初の疑惑を感激へと変えることができる。さらにこのように十分説明できれば、プロトタイピングを実際に評価するための、また必然的に発生する支出についての正式な承認を得ることができる。少なくとも以下の点に対処しなければならない。

- **アプリケーション・バックログの主な原因、構造化手法と技法の長所と短所、さらにプロトタイピングでこれらを補う方法**

・プロトタイピング・アプローチの広範囲な説明書
・プロトタイピングの費用・便益性
・プロトタイピングの前提条件
・提示された評価方法とその費用

関係する人すべての教育

少なくとも管理者の支持を得ると同様に大事なことは、パイロット・プロジェクトに参加しているユーザーと開発担当者を活性化し教育することである。プロトタイピングの概念について知識が不十分であると細部に深入りすることになる。細部に深入りすると開発担当者の役割への反抗とユーザーの役割への過大な期待になる。したがって、プロトタイピングの目標とこのアプローチの実施によって開発工程の各人の役割が受ける影響を認識すべきである。開発担当者やユーザーの立場にとって組織の土壌はプロトタイピングに好意的でなくてはならない。ユーザーはプロトタイピングによって求められる努力を惜しまないように意識付けされねばならない。開発担当者はこのアプローチに積極的な態度を取り、それが結局彼らの作業方法に大きな成果をもたらすのである。教育と訓練はプロトタイピングに対してやる気を起こさせる土壌を育てるのに重要な役割を果たす。

プロトタイプ作成者になろうとする情報分析者は、CASEツールの使用にかかわる実際的な技術を習得しなければならない。DBMSとデータ・ディクショナリ・システム（DDS）、その環境で使われる非手続き言語、それに画面定義とレポート生成機能を使って迅速に実用モデルを構築できなければならない。さらに（もし使用できれば）分析者用ワークベンチも容易に使えなくてはならない。情報システム開発に対するアプローチがプロトタイピングと従来の方法とでは違いがあることから、設定された標準、基準、手順がプロトタイピング環境では制約になってしまう。たとえば、１つの潜在的な対立の原因は開発工程におけるデータベース管理者の役割である。従来、データベース管理者は関係するデータベースを構築するための実際の行動を起こす前に、データベースの設計内容を承認していた。プロトタイピングでは、まずすばやくデータベースを構築し、その後引き続き迅速に最初のデータベースを変更することが必要であるが、この場合時間を要する形式的な手続きが入り込む隙はない。

プロトタイピングでのデータベース管理者は、本番システムで実現するデータベースの設計内容を承認することだけが求められる。つまり、データベース管理者は違った方法で開発工程に携わることから、現在の手続きの改訂が必要となる。標準と手続きについて責任のある人は重大な混乱を予想しておかねばならず、またプロトタイピング・アプローチの影響を認識しておかねばならない。プロトタイピングの導入により標準の変更が必然的に起こることはかなり明らかではあるが、このことに実際十分気づいている人はいないようである（Alavi、1984 b）。

（２）プロトタイピング・ツール

プロトタイピングで必要なCASEツール（なるべくならRDBMS、非手続き言語、優れたスクリーン・ペインタ／レポート・ジェネレータを含む）を使えない場合は失敗することになる。このようなツールを使って仕事をしていない組織にプロトタイピングを導入することは無意味である。DBMSと統合されたDDSがあれば、優れた分析者用ワークベンチのようにプロトタイプの作成は簡単になる。

このようなツールを組織に導入することと、プロトタイピングを導入することは、一般的には無関

係である。これらのツールは開発工程の効率向上を目指したものであり、従来の直線的な開発アプローチを含めいくつかの開発方法に使用されている。“ソフトウェア危機”から逃れる１つの可能性の高い方法は前述したように、より効率的な開発をすることである。しかし、CASEツールの使用によってもたらされる最大の利益は、プロトタイピングと組み合わせたときに得られる。プロトタイプ作成者がハードウェアの制約に束縛されないことは重要である。各プロトタイプ作成者には（インテリジェントな）ワークステーションが与えられ、十分な内部・外部記憶装置が用意されるべきである。応答時間は十分速くなくてはならない。オフィス環境もまた重要で、進行を邪魔されずにエンドユーザーとコミュニケーションできるように、開発担当者は十分なミーティング用の場所をもつべきであり、可能ならば大きなスクリーン・プロジェクタのような発表用設備を備えた所がよい。

（３）パイロット・プロジェクトの実施

実質的な評価のためのスタートをうまく切れると、次の段階は適当な１つのプロジェクト（あるいは多くのプロジェクト）を選択することである。この選択を行なう場合、細心の注意を払わねばならない。つまり、選んだシステムがプロトタイピングに役立たなければ、その結果は単に失敗や失望だけでなく、多分プロトタイピングは不当に拒まれることになる。プロトタイピングの選択基準として以下の項目を挙げる。

システムのタイプ

プロトタイピングが、ユーザー・インターフェイスを重視しなければならない情報システムの開発に特に適しているのは、現在のCASEツールの性格による。データの処理に重点を置いた場合、プロトタイプを迅速に作り上げるのに十分適した手段は現在のところない。

システムの規模

一般に、大規模なアプリケーションも小規模なアプリケーションもプロトタイプは可能である。しかし、開発するシステムの規模はパイロット・プロジェクトを選択するときの要因となる。アプリケーションは、このアプローチを大規模システム開発にも使えることを示すために適当な大きさでなくてはならないが、経験を積みつつあるプロトタイプ作成者が、そのプロジェクトの大きさのために問題に衝突するほどの大きさであってもいけない。

プロジェクトの様子が組織内の皆によくわかるようにするには（興味をもった外観者が多数いるだろう）、あまり大きなプロジェクトを選ばないようにすることである。そうすれば結果をすばやく示せるというのがもう１つの理由である。

パイロット・プロジェクトを始める前に、このプロジェクトを評価するための基準を作ることが重要である。パイロット・プロジェクトが成功に終われば、プロトタイプによって完全な要求定義も、あるいはもしプロトタイプの性能を改良できれば、本番システムも作れることを示すことができる。パイロット・プロジェクトを通して、懐疑者はこのアプローチが可能であり、実際的であり、また成果をもたらすことを納得することができる。しかしながら、客観的な基準がないため、この成果の質を測定することができない。保守費用を基にすれば、そのシステムの有効性についてある見識を得る

ことができるはずである。しかし、パイロットという状態であることから、長期間保守費用を測定するためにシステムを使用することは実際的にほとんど不可能である。したがって、従来のアプローチと比較して経済的な利益が生まれるかどうかについて客観的に示すことは難しい。さらに、従来のアプローチによるそのプロジェクトへの影響は（第4世代言語と組み合わされているかどうかにかかわらず）わかっていない。

したがって、パイロット・プロジェクトを評価するために使える基準は大部分主観的である。たとえば、その工程と成果物の両方に対するユーザーの満足度、要求定義にこのアプローチを用いたことに対する開発担当者の評価、プロジェクトがその過程で行なったプロジェクト管理とその結果についての印象などがある。参加する人は、すでに従来のアプローチに経験があるほうが望ましい。なぜならば、適切な比較ができるからである。

十分踏み固められた道からそれたプロジェクトは、型にはまったプロジェクトより失敗の危険度が高いということを認識すべきである。もしパイロット・プロジェクトがいくつかの点で不満足な結果になろうとも、これは必ずしも使用した開発アプローチの問題ではない。

その原因は新しい作業方法に十分な経験がない、あるいはツールの使用に熟練していないことにある。パイロット・プロジェクトを開始する前に、開発者が小さなテスト問題をユーザーとのやりとりなしに全工程を通して試みることができれば失敗の危険度は低くなる。パイロット・プロジェクトでは、開発工程の他の改革に気をとられずプロトタイピングの評価にのみ集中することが重要である。新たな構造化手法、新たなツール、それに新たな開発アプローチを同時に導入し、そしてさらに未知のアプリケーション分野に乗り出そうとすれば、失敗を避けることはほとんどできない。

（4）プロトタイピング・アプローチ

プロトタイピングは、従来の要求をまとめる方法に代わる完全に独立した方法として受け入れるべきであるとの勧告でパイロット・プロジェクトが終了し、管理者もその勧告に同意するならば、組織内ではプロトタイピングの基盤の拡大方法を検討しなければならない。

プロトタイピングの提案の前に、アプリケーションが成功するための十分な条件を設定しておかねばならず、また組織の用途に合わせた実用的なマニュアルを作成しなければならない。標準のプロジェクト・モデルは、要求定義フェイズではプロトタイピングを、また本番システムのベースとしてはプロトタイプを使えるように変更が可能でなくてはならない。プロジェクトの管理者と開発担当者のための訓練コースの開設が必須である。最初にプロトタイピングで作業をする組織・部門は、このアプローチの経験をコンサルタントに相談できなくてはならない。

小さな組織では、パイロット・プロジェクトとその成果を誰でも認識できる。もしパイロット・プロジェクトにかかわる人がプロトタイピングに熱中すると、他の人にプロトタイピングを使うように勧める必要はほとんどない。大きな組織のパイロット・プロジェクトは、開発手法・技法およびツールの選択と標準化に責任をもった企業内組織の部門によって進められることが多い。組織の大きさや手法を定める部門の権限にもよるが、プロトタイピングが広く用いられるようになる前に、相当渉外活動を行なわねばならない。すなわち、プロトタイピングの正しい提案をしなければならない。そうすることで、本当の組織に受け入れられるための不可欠な広い支持が得られることになる。

プロトタイピングが、情報システム開発の分野における重要な進展であることは疑いない。他のエンジニアリングの分野では、すでに一般に通用している開発システムの実用モデルを導入することを通して、プロトタイピングは開発工程をさらに専門化することになろう。

プロトタイピングを用いると、自動化はユーザーの目に“より親しみのある顔”となり、そこでの要求定義戦略はユーザーと情報システム専門家との大きな隔りを埋めるのに役立つ。

開発費用に対するプロトタイピングの効果

プロトタイピングにより開発費用はかさみ、プロジェクトの進行が遅れることが予想される。この仮定は、主としてプロトタイピングを従来の手法・技法の代替としてではなく、それを補うものとして見なければならない事実を基にしている。プロトタイプが本番システムのベースとして維持されるとしても、工程は繰り返し行なわれることから、要求定義にプロトタイピングを使うプロジェクトは多分従来の開発より費用を要す。もしこの仮定が確たる数字で裏付けされれば、それは役に立つことになろう。この分野で測定を行なうのは難しい。最大の悩みの種は、ソフトウェア開発費用が非常に広い様々な要因に依存していることである。開発工程に及ぼすプロトタイピングの影響について断言するためには、非常にしっかりと管理された“実験室”の条件のもとで調査を行なうか、あるいは膨大な観察を行なうかのいずれかとなる。

保守費用に対するプロトタイピングの効果

プロトタイピングが、従来のアプローチより質の優れた要求定義に結び付くと期待できるかどうかはいまだにはっきりとしない。これも確たる数字に欠けるのである。残念ながら、要求定義の質を計るための優れた基準はない。しかしながら、保守費用の大きさとシステム要求定義の品質との間には逆の関係があることから、ある洞察を行なうことは可能である。プロトタイピングは主に保守フェイズで顕著な効果を上げることができる。

しかしながら、保守費用に対するプロトタイピングの効果を測定する場合、非常に多くの費用決定要因があるという問題に再び直面する。プロトタイピングの効果を取り出すには、数多くの観察が必要となる。この種の調査は費用が掛かり非常に長期間を要する。事実、測定はシステムが“期限切れで捨てられる”まで続けなければならない。したがって、プロトタイピングが保守費用にどんな影響を及ぼすかについて正確に測定しようとすれば、異常なほどの忍耐が要求される。実験室での調査は、かなり長期間にわたって実際にシステムを使うことが興味の対象となることから、観察に代わるものとはほとんど考えられない。十分な期間にわたって興味ある要因をすべて固定しておくことはほとんど不可能である。

開発工程と完成したシステムの両方に対するユーザーの態度にプロトタイピングが与える影響

これは数少ない調査の１つで、ここではプロトタイピングの期待効果についてある経験的基礎資料を得るための努力がなされている。この調査によれば、プロトタイピングでより一層ユーザーと親和性のあるシステムには必ずしもならないようである。このような驚くべき結論には将来の研究が求められる。

上に述べた点について、より優れた経験的基礎数字を得ることは学術的な世界にとっても、現場の社会にとっても1つの挑戦である。実際、プロトタイピングの費用対便益効果を経験的に調査するには、費用を要し時間的な浪費となってしまう。様々な研究機関と産業界とで情報を交換し協力し合うことは、この問題についての内容を理解するのに不可欠である。

関連する調査活動

本項では、プロトタイピングと接点をもつ多くのことについていままで簡単に述べてきた。これらのうちいくつかについて、現在の文献は十分な注意を払っていない。短期的に行なわれるべき調査活動は以下の通りである。

・プロジェクト管理と開発作業の両方のため、開発戦略、手法、技法、自動化ツールの選択を行なうための暫定的アプローチの作成と適用。

プロトタイピングと従来のアプローチとを選択する基準に注目してきたが、これはより広い観点から見なければならない。つまり、プロジェクト・リーダは他の選択もできるのである。実際、以下の選択ができるよう広範囲な枠組みが緊急に求められている。

a. エンドユーザー・コンピューティングによる開発か、情報システム専門家による開発か
b. 標準パッケージを購入するか、システムを社内で開発するか
c. 従うべき開発アプローチ
（参加型、進化型、プロトタイピング型、直線型、プロセス指向型、データ指向型など）
d.それぞれのアプローチで使うべき構造化手法・技法、CASEツール

・戦略、手法、技法、ソフトウェア・ツールの生産性に対する効果の評価。

現在新しい開発ツールを組織に導入する場合の企業の判断には、ほとんどビジネス上の適切な理解に欠けている。プロトタイピングは単なる要求定義戦略ではない。また十分な定量的データがそろっているわけでもない。現場では、費用対便益の観点から妥当であるか否か前もって分析せずに、その時点での流行に従うこともありえる。文献はほとんど手掛りを与えてくれない。生産性についての記述は一般に定性的であり、定量的なことが述べられる場合は、ほとんど実践的な証拠がそろっていることはない。様々な開発戦略、手法・技法、（第4世代あるいはその他の）ツールの生産性に対する効果についての調査が緊急に求められている。

・新たな戦略、手法・技法、およびツール導入に関連する問題点の評価。

実際（たとえば）構造化手法あるいは第4世代ツールの導入は少しも簡単ではないことがよくあるようである。マネージメント科学の分野において、最近の数十年間多くの注意が組織変更のプロセス管理に払われてきた。この知識をシステム開発戦略その他に導入可能性の調査は、まだ組織的にはなされていないようである。新しいCASEツールの出版物や提供業者は、この問題にもっと注目すべきである。

最後の注意事項

情報システム工学の教義はまだ若く、精力的に発展している。いままでのところ、しっかりと用語が確立されたとの徴候はない。現場では、どんな“確立された真実”もほとんどなく、実際すべて議論の対象となっている。ハードウェアの急激な進展は、開発方法論の発展にとって刺激となっており、次はテクノロジー分野のニーズを明らかにすることになってきている。プロトタイピングは“テクノロジー主導”開発のよい例である。要求定義のこの戦略の発端を直接刺激すれば、第4世代ツールの時代の到来となり普及することとなる。

このダイナミックな開発分野では、時々後戻りすることは有益であるし、必要でもある。したがって、再考すべき多くの点について述べた。新しいものに対して批判的な態度を取ることは賢明である。あまりに早急にあらゆる種類の目新しいものを導入することにより、それらを吸収する組織の能力を越えてしまうことがあり、問題を引き起こすことにもなる。一方、あまりに導入が遅れると、企業の競争上の地位を最終的に損なうことになる。

5．2　思考の道具としての情報システム

コンピュータ技術の現状を概観したところで、次に研究者の立場からどのようにコンピュータ活用していったらよいかを考えてみる。そもそも、コンピュータを活用しようとすれば、コンピュータには何ができるか、自分（達）の仕事に、コンピュータをどう効果的に役立てるかという2点をよく理解していなければならない。コンピュータに関係した書物や雑誌は巷に氾濫しているし、今日の新聞や一般雑誌、テレビの広告にも頻繁に登場するようになった、しかし、この種の情報によってコンピュータに何ができるかを理解することは、実際には不可能に近い。

レベルは多少異なるとはいえ、一般の研究者にとって、コンピュータについての最新情報や流行語が自分達の仕事とどう関係しているかを理解することは困難である。ワープロやレーザープリンターならばまだしも、マルチウインドウ、マウス、Ethernet（イーサーネット）、EWS（エンジニアリングワークステーション）、スーパーコンピュータ、LISP、PROLOG、リレーショナル・データベース、知識シェル、人工知能、ニューロコンピュータ、ファジィコンピュータなどが、どういう機能をもつ道具で、地層処分研究にどのようなインパクトを与えることになるか、適確に把握し、見通している人は少ないであろう。逆に、そういった新しい道具の作成者や、（広い意味での）販売提案者にしても、彼らが触れまわっている道具が研究にどう役立つかをおそらくは正しく理解していないであろう。このギャップは、どう埋められるであろうか。この課題の最良の解決策は、コンピュータの側からでなく、むしろそれが利用される仕事を分析することによって得られる。地層処分研究は、科学研究の一分野であり、科学研究は人間の創造的な思考活動の一種である。この関係を逆に遡れば、上記の問題解決の糸口がつかめる。すなわち、コンピュータを創造的な思考活動にどう役立てるか、その中でも科学研究にどう役立てるか、その中でも地層処分研究にどう役立てるか、さらにそれぞれの課題ごとにどう役立てるかを考察して行けばよいのである。もちろん、その前提として情報処理機械や制御機械、ないし万能機械としてのコンピュータの基本技術と基本ソフトウェアを理解しておく必要ある。このように“思考の道具”としての体系を追って行けば、我々は急速に進歩するコンピュータの眼先の変化に幻惑されずに、自分の頭で考え、必要な情報を集め、自分に適したコンピュータ“環境”を構築して行くことができる。この意味で、コンピュータに関連した技術を思考の道具という立場に立

って、概念的に整理することがシステム開発にあたってまず必要である。

５．２．１　研究者の環境

地層処分は実験を主とする科学の一分野であると考えられる。それは高度に創造的な人間の活動であり、その最終産物は“知識”である。この活動の最初の起動力は、好奇心やアイデアなどである。実験を行う目的はデータを得ることである。したがって、一般に科学の研究者は絶えざる情報の流れの中に身を置きながら、情報（データ）を得、情報を操作して、新しい価値ある情報（知識）を生み出そうとしているのである。

この情報という観点からみた場合、研究者は三つの異なる世界と関わっている。すなわち、外の世界、思考を専門にする研究室（オフィス）、実験室である。この三つの世界において、コンピュータの利用形態は著しく異なっている。まず外の世界には、誰れでも料金さえ支払えばアクセスできる公開情報やコンピュータリソースが存在する。大学や国公立の研究者の場合は、共同利用型のコンピュータ施設や、データベースにアクセスすることもできる。また国内だけでなく、海外の情報源も利用できる。

これらの情報は、学術雑誌や特許公報の形で刊行された書誌が源であり、それらを選択的にコンピュータ可読な型に編集して作成される。一般にこの世界のコンピュータは、大規模、広域的なネットワークシステムである。国際間を結ぶのは、KDDの国際コンピュータ通信網である。大学であれば、大学間を結ぶ共同利用型大型コンピュータネットワークである。また企業の場合は、研究所間や、研究所と本社、各事業所間を結ぶ全社的なネットワークであろう。そのような環境にふさわしいコンピュータシステムを、どうデザインしたらよいかは、おそらく一般の地層処分研究者の関心外のことであろう。

必要なことは、どのようなコンピュータリソースがあり、それをどのように利用できるかについての知識であろう。実験室には、コンピュータ利用からみて、性格を異にするいくつかの機器が存在する。一つは、試験設備である。それらはコンピュータで制御されてはいるが、それ自身の内部にすでにプログラムが存在し、この指令で仕事が完結する種類の装置である。このような装置の場合、利用者がコンピュータとして手を加える余地もなく、プログラムを書くこともない。

もう一つは、質量スペクトル分析器のような計測・解析を目的とする機器である。これらの装置はすでにほとんどコンピュータを内蔵している。このような装置では、装置から出てくるデータをどう処理するかが大きな問題である。すなわち、離れた場所の（汎用）コンピュータで行うか、あるいは計測・解析装置に接続されているその場（オン・サイト）の専用コンピュータに任せるかは、議論の分かれるところである。

そこには、データの蓄積、データの転送（移動）、データ解析、データの総合的な利用という課題がある。データの転送・移動には、フロッピーディスクなどによる物理的に媒体を運ぶ方法と、オンラインでのデータ転送方式がある。後者はLAN（Local Area Network）の思想である。この種の機器の中には、コンピュータで制御された顕微鏡のように、利用者がコンピュータと対話しながら、計測を行う装置が含まれる。また実験室あるいは実験装置内の湿度、炭酸ガス濃度、溶液量などのモニタリングと制御を行う環境制御システムも含まれる。

ラボラトリーオートメーションは、このような計測と制御をコンピュータの（ローカルな）ネット

ワークによって効率的に行い、生産性の向上を計ろうとするものである。ただ、研究は生産工場などと違うため、このようなネットワークが効率を発揮できるのは、かなりルーチン化されたスクリーニング試験を指令とする施設に限られる。もちろん、コンピュータによる実験室の高度な省力化、自動化、高精度化の理念は、基礎研究を指向する研究部門でも大切なことではある。

いずれにしても、実験室ではやはり機器のシステム化が常に問題となる。これに対して、思考の世界である研究室の主役は人間である。普通、研究者は外の研究情報を仕入れて蓄積しておく一方で、実験室やフィールドからデータを集め、それを自分の経験知識や直観に照らして分析し、そこから新しい知識を生成しようとする。その結果は、外の世界（組織内も含めて）への報告書や論文となり、また新たな研究計画や実験へとフィードバックされて行く。これが、研究者の創造的な思考活動である。

この過程のコンピュータ化が最も遅れている分野であり、まだ概念的にも十分整理されていない。それというのも、この過程の核心である創造思考のメカニズムが解明されていないからである。しかし、コンピュータ化は、創造思考に関連した周辺の仕事の支援から外堀を埋めるように、ゆっくり着実に進んでいる。ただ、核心に迫れば迫るほど、一般的な方法論が通用しなくなり、その分野特有の具体性を帯びた問題が立ちふさがる。

例えば、外から収集したデータや、実験データをデータベース化するような仕事には、かなりの共通性がある。また、研究成果としての知識をドキュメントやプレゼンテーションの材料とするツールも共通性が高い。またデータの統計的な解析などは、分野を問わず、共通の手法がパッケージ化されている。しかし、実際問題の解決には、もっと具体的なデータや機能特性、機能に関係した構造部位や、分析に関する知識などが必要となる。これらの知識とその構造、数学的なモデル、経験的な技法は、分野や課題ごとに大きく異なる。したがって、普遍性と特殊性が共存するこれらの情報処理の問題をどのように分割して、解決すべきかが問題となる。この他に、研究室の仕事には研究予算や、備品、スケジュールなどの管理的な仕事がある。この種の仕事は、研究室で行われていても、本質的に事務部門の仕事と同じ性格をもつ。したがって、そこでのコンピュータ利用は、いわゆるOA（オフィスオートメーション）そのものである。

研究成果の文書化も単に論文を書くという仕事であれば、事務的な文書の作成と同じ道具が使える。しかし、コンピュータの中にあるデータベースの一部や、構造式のような図形情報、分子の立体表示などを編集して組み入れようとすると、より高度な科学技術報告書の作成システムが必要となる。いずれにしても、このように研究者の情報環境から分析して行けば、これまで開発されてきた多様なコンピュータシステムをうまく分類することができる。

5．2．2　思考支援システムという考え方

研究者の思考を支援する中核システムをどうデザインするかという問題は、あまり明確に論じられることがない。一般に、図5．2－1のようなデザイン理念が考えられる。それは、要約すれば以下のとおりである。

- **思考支援システムは、普遍的思考支援システム、分野の基盤システム、課題別のシステムという階層構造をとる。**
- **これらのシステムには、知識とデータベースが接続されるということになる。**

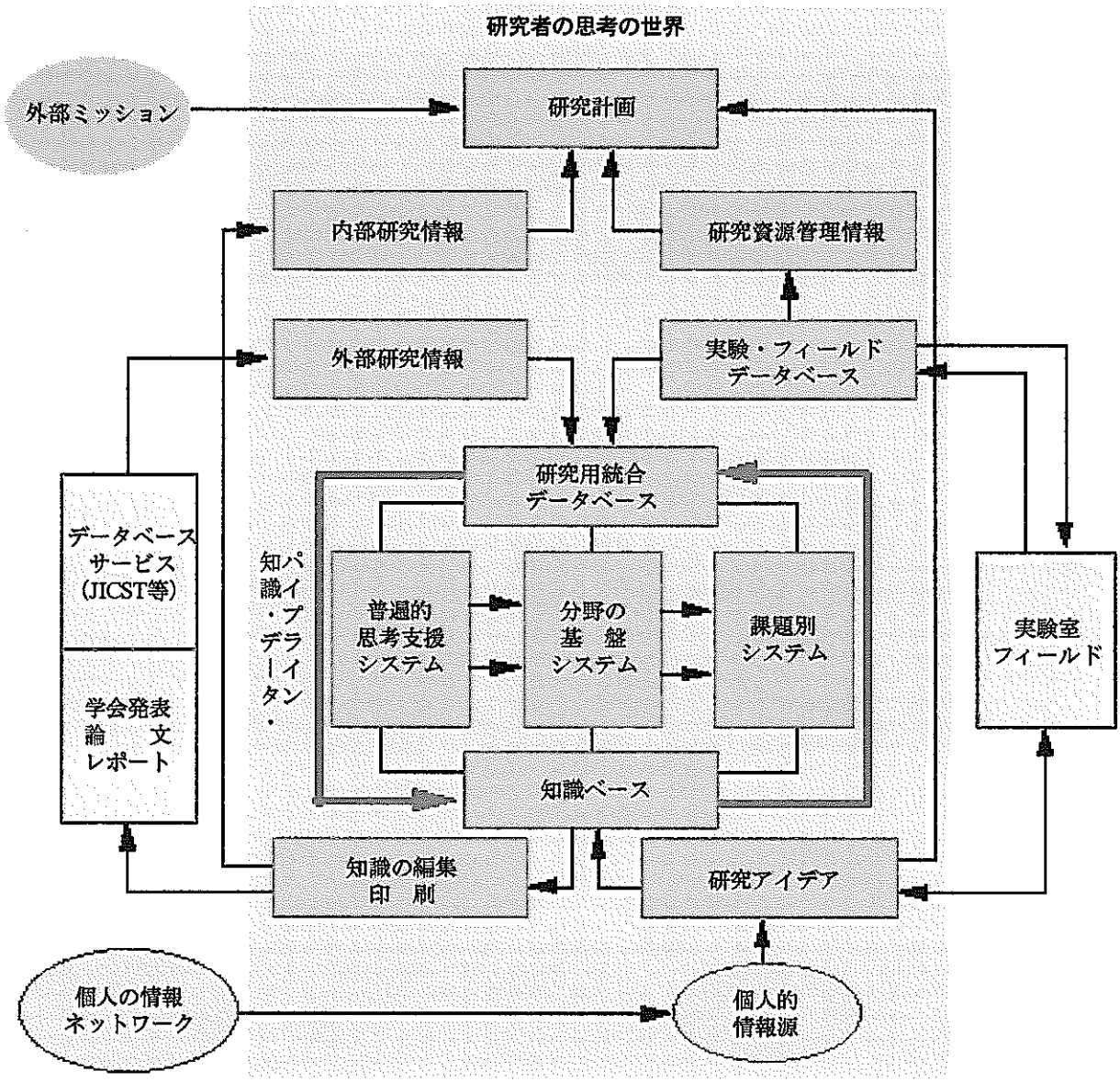

図５．２－１　研究者の思考体系

普遍的な思考支援システムとは、帰納、演繹、発想、計画というような、分野に依存しない人間の思惟の基本的なパターンに対応した支援システムである。帰納とは、集めたデータから一般的な知識や規則を導く過程である。データとしては、数値、記号、記述の他、信号（波形）や映像のようなアナログデータなどがある。これらのデータを扱うコンピュータシステムとしては、（数値）データ解析、統計処理、記号処理、パターン認識、時系列（信号）解析、画像解析などがある。

これらのシステムは、人間の代りに帰納を自動的に行うものではないが、人間の帰納的な思考を助ける思考のツールとみなすことができる。演繹を支援するシステムは、しばしば人工知能と同一視されるエキスパートシステムである。この種のシステムは最初、化学、地質学、医学など特定の問題領

域ごとに開発された。

これらのシステムは、必ずしもその目標を達成しなかった。しかし、方法論としては一般の関心を呼んだ。この結果、エキスパートシステムを構築するソフトウェアや、専用のハードウェアが数多く開発され、商品化されるまでになった。発想に関しては、まだこれを支援するコンピュータシステムのイメージが固まっていない。そもそも、発想は創造の源ではあるが、そのメカニズムは不明で、コンピュータに任せることはできない。したがって、発想自体をアルゴリズム（計算の手順）として表現するよりは、コンピュータに人間の発想の手伝いをさせるアプローチの方がより現実的である。

例えば、コンピュータ・グラフィックスを利用して、現象やデータを視覚化することにより、研究者の発想を刺激することが考えられる。

それは、従来の紙の上の２次元思考を３次元、あるいはさらに多次元化し、研究者の発想を豊かにすることを目的とする。すでに３次元グラフィックスは、地層処分研究者の必須の道具となっている。

計画技法としては、すでに数理計画法や最適制御理論のような数学的な方法論が、様々なプログラミング・パッケージにまとめられている。この他に人工知能としては、学習という課題があるが、これは上記の基本パターンの組合せとして扱うことができる。

ある分野の基盤システムとは何か。それは、思考の対象となるその分野ないし研究領域において基盤となっているシステムのことである。例えば、地層処分は化学と地質学等に基盤をおいている。

５．２．３　想像を活発化するシステムと言う考え方

情報処理の立場からみると、この過程は、外界部からデータを取り込んで、何らかの知識、ないしは知識の断片を出力することである。そこで研究者の頭の中にすでに蓄積されている情報や知識以外に、科学的な判断や類推能力や経験的な直観的なヒラメキなどがフルに動員される。

このような思考過去をコンピュータによって自動化することはできるのであろうか。また、完全な自動化はできないにしても、部分的にもコンピュータを導入して知識生成の効率を上げたり、コンピュータを使うことによって、研究者の思考能力の限界を越えた、より創造的な仕事をする可能性はないのであろうか。このようなことが本当に可能であるかどうかは別として、仮りにそのようなことが可能であるとすれば、その前に少なくとも次の２つの技術的課題が達成されていなければならない。

第１は、人間が判断の材料とするデータや知識をすべてコンピュータで扱えるようにすることであり、第２は、人の思考・判断のステップをコンピュータが実行できるように、あらわな形で記述することである。

実はこの両者は互いに関係し合ってもいる。なぜかというと、人間は受動的に動作するコンピュータと違って、自らの意識で必要な情報を集める。ところが人間が思考のためにどれだけの外部情報と、自分の中の情報を動員しているかは、本人でもよくわかっていない。しかも、現実の判断行為は、理想とは程遠い限定された情報収集に基づいて行われる。つまり、何を入力情報とするかは、情報をどう活用するかと表裏の関係にある。

したがって、第１の課題は、第２の課題が解決されてからでないと明確になってこない。ところが、もちろん第２の課題は、すべて解決されていないことから、第１の技術目標も完全には明らかにはなされていない。このことを説明するために、まず人間の判断に必要なデータや知識を、すべてコンピュータに入れることを考えてみよう。そのような情報は、記号や数値のように、デジタル化しやすい

情報と、時系列信号や映像のようなアナログ情報に分けられる。

すでにみてきたように、後者のようなアナログ情報も、今日では比較的容易にデジタル量に変換できる。すなわち、論文の型で発表された過去の知識や、実験室から得られたデジタルおよびアナログ情報をコンピュータに蓄える技術はほぼ確立されているのである。

しかし、コンピュータ内部に蓄えられた情報は、研究者には利用できても、コンピュータにも同じように利用できるとは限らない、デジタルオーディオ機器は音楽をデジタルに畜え、再生することができても、音楽を観賞することはできない。同じように、イメージスキャナーは、文字や写真をデジタル量に変換してコンピュータに蓄えるが、それらの文字を解読したり、写真を観賞することはできない。

これと同じで、研究者の知識やデータをコンピュータに記録して再生できることは、その高次な利用に向けての第1歩であるが、それだけでは本当に活用することには直結しない。しかし、アナログ量を、とにかくデジタル化してコンピュータに蓄えるという技術が実用化されたという意義は大きい。

なぜならば、これによってコンピュータに記録されたあらゆる形態のデータや知識を、どう活用したらよいかを誰れにでも研究できるようになったからである。内部に取り込んだ情報をどう活用するかという問題は、生物の感覚系から中枢神経系に到る情報処理と共通したところがある。とくに、時系列信号データの処理や画像データの処理は、聴覚や視覚の神経系における処理と類似したところが多い。

したがって、コンピュータ工学者も生物の情報処理機構を研究し、そこからいろいろとヒントを得ようと努力している。しかし、音を聞きわけたり、餌をみわけたり、巣の方向を探したりというような、アナログ的なパターン認識から先に進んだ統合的な判断や、創造、発見のような思考のメカニズムとなると、参考となるような生物学的知見は、ほとんど得られていない。

もう一つ見落しやすい点がある。それは、研究者はコンピュータにデジタル形式で取り込めるような情報だけに基づいて思考してはいないという事実である。実験の最終結果だけならば、例えアナログ情報であっても、確かにコンピュータに入力できる。しかし、研究者の勘や発見的ヒラメキは、それ以外の実験の全過程や、極言すれば、彼がそれまで外界から入力したすべての情報に誘発されて生れてくるのである。

しかし、それらの無制限な情報をあらかじめコンピュータに蓄積しておくのは到底不可能なのである。したがって、このような高次の思考過程を完全に自動化することは、現在の技術では不可能であるし、将来も難しいことであろう。しかし、帰納、演繹、ある目的に到達する計画など人間の理性的な思考の基本パターンの各々に関しては、人間の思考を助けるような計算手法がある程度まで開発されている。そしてこれらの手法は場合によっては、人間だけでは不可能な大量のデータや複雑な知識を整理して活用する強力な道具となりうるのである。

（1）数値計算から得られたデータをどう解釈するか

これは科学研究一般の大きな課題である。このとき研究者は漠然とではあっても、まず何らかの数学的なモデルをあてはめて考えている。とくに物理学のように数学と表裏をなすようにして発達してきた分野では、理論モデルは不可欠の思考の道具である。

そこで、物理学のような分野では、高度に抽象化された数学体系に基づくモデル、高次の方程式の

記述と解法、複雑な数値計算手法などが発達してきた。このような仕事をコンピュータを使って効率的に進めるために開発されたのが、各種の数学的なライブラリである。これらは大別して、数値計算と数式処理に分けることができる。

このうち、最初に実用化されたのは数値計算である。コンピュータは元来、数値計算を高速で機械的に実行することを目的として開発されたので、このことは当然であろう。数値計算のプログラムを集めたライブラリは、数値計算ライブラリ、科学（技術）計算パッケージなどと呼ばれ、大手のコンピュータメーカーから提供されている。

このように、大型機のために開発されてきた各種の科学技術計算プログラムも、最近では個人利用のパソコンに移植されるようになった。これは、一昔前の大型機に負けない演算能力と記憶容量を備えたパソコンが登場し、さらにその上でFORTRAN77やCが走るようになったからである。現在標準的なパソコンの環境としては、MSDOS、1GBのディスク、そして数値演算コプロセッサが揃っていることである。

（2）数式処理

数式処理というのは、コンピュータに因数分解とか、不定積分のような式の計算を行わせようという試みである。普通、数値計算の場合は、人間が式や数値を代入して値を求めればよいところまで変形しておいて、あとの機械的計算の部分をコンピュータが実行するという形を取る。一般に数式を変形して行くには、発見的な思考が必要である。もちろん、非常に簡単な因数分解や不定積分なら、ある種のルールを機械的にあてはめるだけでよい。そういう操作がコンピュータに実行できるようにプログラム可能ならばコンピュータに数式処理を任せることができる。

しかし、一般にこのような変形は、純粋の機械的操作だけではうまくいかない。コンピュータ工学の立場からすれば、これは一種の記号処理（シンボリックプロセッシング）の問題であり、あとで述べる人工知能の研究の一分野とも考えられている。ここで人工知能という意味は、人間の経験に裏づけられた勘、ないし直観のような推理能力をコンピュータに代行させるという意味である。このような発見的なアプローチをヒューリスティックスと呼ぶ。そこで人工知能の一つの課題は、人間のようなヒューリスティックな推論能力をどうコンピュータに備えさせるかということである。このような問題を一般的な形で解決することには、まだ成功していない。

しかし、この分野の研究者は、それぞれの問題ごとにコンピュータにも実行可能な何らかの代替案を粘り強く探し求めてきた。数式処理について、その努力の過程をみるならば、“人工知能”というような恰好のよい言葉のイメージからは、相当隔ったもので、人工的な人工知能研究者からはしばしば“そういう試みは人工知能ではない”と冷たく批判されるほどであった。

これまで数値計算と数式処理のプログラム・パッケージを別々に紹介したが、この双方に共通した将来課題が、この双方のプログラムの結合である。すなわち、方程式あるいは数学的なモデルを記号的に書き下してから、数式処理システムでこれを数値計算可能な形に変形して、数値計算のプログラムを呼んで、実際の数値解を得るという手順の自動化である。しかし現在の数式処理の出力は、一般的なケースでは、そのまま数値計算パッケージに渡すことはできない。このような融合ソフトは、多分限定されたモデル（方程式）ごとに作成されていくものと思われる。

（3）データ解析・統計解析

数値計算や数式処理が、数学や物理学の領域で研究され、これらの分野で実際のソフトウェア・パッケージが開発されてきたのに対し、統計処理のようないわゆるデータ解析のためのプログラムは、どちらかといえば医学や生物学分野の利用者を想定して開発されてきた。

後者の研究領域では、抽象的な数学概念を現象のモデルにあてはめることがないかわりに、得られたデータを研究者の経験知識を多大に加えて解釈することが多い。また、臨床医学データのように、第1の目的が科学研究でなく、他の目的で記録されたデータがある程度蓄積されてから、研究目的で吟味し直してみるというケースもある。このような場合、解析は二の次で、まずデータを後日利用可能なようにコンピュータに蓄えておく機能が必要となる。

後者の機能は、ある手法で解析した結果を再び他の手法の入力データとするというような場合にも必要となる。また、そもそもデータを眺めて、その中に何かの規則を発見したり、データを目的とする解析手法に適用できるように編集するためにも必要となる。

これには2次元の表として整理されたデータに、新しい項目（行や列）を追加したり、いくつかの項目を統合して新しい項目を生成したりする作業が含まれる、このようなデータの編集や作表を自由に行えるようにするには、リレーショナルデータベースの機能が有効である。また結果を確認しつつ、新たな処理を考えるためには、表を自在に展開したり、グラフを表示したりする作図・作表機能が必要となる。これらは結局、対話型のグラフィック機能と呼ばれるものである。

このように、データ解析を支援するシステムは、統計学的手法のような各種の解析手法が集められており、リレーショナルデータベースのような、データを蓄積し、編集し、柔軟に検索できる機能があり、結果をディスプレイに視覚的に表示し、必要ならば、それらのグラフやチャートをハードコピーとして保存することのできる対話型グラフィック機能が備わっている。いろいろな操作を、できるだけメニューの選択かコマンドを打ち込むだけで実行できる、いわゆるユーザーフレンドリーな環境で提供することが望まれる。

（4）パターン認識

データ解析の基盤となる統計は、国の統計事業や、実験計画、生物学的検定などとして昔から時間をかけて体系化が進んできた。これに対してパターン認識は、コンピュータの誕生以後、いわゆる人工知能的な興味から研究され、文字認識のような実用的なターゲットに向けてシステムの開発が行われてきた。それだけに、まだ学問としての体系化もなされておらず、どこからどこまでの手法をパターン認識と呼ぶかは専門家の間でも意見が一致していない。

とくに統計的な手法は広く使われているので、統計学との線引きが難しい。そこで、ソフトウェアとしてもパターン認識としてまとまったパッケージというのはまだ存在しない。パターン認識は、数値化されていない波形信号や絵や図形のようなアナログ対象をも研究対象とする。もちろん、数量化されていない対象は何らかの手段で数値の組に変換しなければコンピュータで扱えない。この過程を前処理と呼ぶ。この前処理が実は重要であり、かつ難しいことが多い。

この点が、すでに数値化されている対象を相手とする統計学やデータ解析とは大きく違う。前処理には、A／D変換のような機械的処理だけでなく、情報の選択的抽出、情報の要約、データ（形式）の変換、構造表示など、多様な手法が組み合される。これらの手法は、扱う対象と目的によって様々

であり、それぞれかなり独立な研究分野を形成している。

しかし、ひとたび数値の組（ベクトル）で表現された対象と考えれば、これを識別する（クラスに分ける）問題には、かなり共通したアプローチが存在する。それは、パターン認識における特徴抽出（featureextraction）と識別（classincation）理論、人工知能の判断ルール、ニューロネットワークとして表現された適応学習システム、ファジイ論理、統計的判別関数理論などである。

これらは、コンピュータ自体で対象を分類するクラスを発見するか、あるいは予め与えられたクラスに、学習を繰り返しながら対象を分類していく手法の基礎となる理論である。

5．3　プロトタイピング手法

自動化の分野において、プロトタイピングに関する論文を掲載したことがない専門雑誌はほとんどない。このように多くの注目を浴びているにもかかわらず、自動化分野の発展は依然としてどちらかといえば遅々としている。この理由の１つは、プロトタイピングとは何か、あるいはどうあるべきかという問題についての合意が得られていないことによる。それぞれの著者がそれぞれの定義を行なっている。このテーマについて研究者や著述家は概念についてまったく明確に定義しないことが多い。また類義語が使われることから、プロトタイピングについての論文は正確に同じように理解がなされない。ここで、繰り返すようだがいま一度プロトタイピングについて記述する。

本項ではプロトタイプを次のように定義する。つまり、プロトタイプは

情報システム（の一部）の実用モデルであり、そのシステムの特定の面を強調するものである。

この定義の中心をなすのは“モデル”の概念である。プロトタイプが完全さを求めることに疑問の余地はない。この機能は情報システムの特定の面に光を投げかけ、特に最も不確実な面を強調するものである。すでに様々なモデル化手法・技法が情報システムの開発に使用されるようになってきている。プロトタイプと、このような手法・技法を用いて作り出されるモデルとの間の基本的な相違は、“実用（workig）”という用語の定義に現われる。

一般に、プロトタイプはモデル化しようとするシステムのように行動し、また外観を示す。プロトタイプとモデル化しようとする情報システムとの間での目に見える相違は、まったく限られたものとなろう。プロトタイプは単に紙の上にシステムを表現するだけでなく、コンピュータ上で実現される本当に使用可能なシステムのモデルである。プロトタイプは、エンドユーザーがそのシステムで実際的な経験を得ることができるかどうかによって評価される。もちろん、これは伝統的な類いのモデルでは不可能で、それは相当抽象化された図形、あるいは形式言語で情報システムの機能的特性を表現している。プロトタイプはどんな大きさ、形でもよい。プロトタイプには、システムの外観（画面、レポート、ダイアログ（会話））のみをモデル化し、ユーザーが入力したデータについては何もしないものもある。この種のプロトタイプはモックアップ（mock-up、実物大の模型）と呼ばれている。そのほか、プロトタイプによってはデータを貯えたり、これらのデータ上での操作を行なったりすることができるものもある。

このようなプロトタイプは、一般に（実際にもそうだが）機能型プロトタイプ（functional prototype、ファンクショナル・プロトタイプ）と呼ばれている。プロトタイプの中には非常に包括的でまた非常

に複雑なシステムをモデル化できるものがある。一方、比較的単純で小さなシステムをモデル化するものもある。プロトタイプは開発すべきシステムの一部のみでも、システム全体でもモデル化することができる。

プロトタイプと本番システム（producdon system）との間には明確な区別が必要である。本番のシステムは実際の業務処理のために使用されるものであり、このため品質に対しては一般に厳しい要求がなされる（信頼性、頑強性、保守性、効率性など）。逆にプロトタイプは不確実さが残るシステムの側面を目に見えるようにすることであり、本番システムとは区別される。プロトタイプはこれらの側面についての仮定が正確かどうか検証するのに使用される。本番システムに反して、プロトタイプは一般に不完全であり、誤りなく機能することを目指してはいない。

工学系の多くの分野で、プロトタイプの利用は一般的な行為である。自動車や航空機産業でプロトタイプを構築するのは、新しい製品を開発する場合に当り前のことである。これらの環境で、様々な種類のプロトタイプは開発中の製品のいろいろな面を視覚化するのに利用されている。しかしながら、製造環境では１種類のプロトタイプが作られることはよくあるが、情報システムの場合はそうではない。つまり製品開発サイクルの後工程では、製品の完成版かその構成要素の１つ、つまり“現実に使える”プロトタイプが作られることがある。この種のプロトタイプと最終製品との唯一の違いは、それがどのように作られるか、その製造方法にある。

自動車のような製品の場合、製造工程それ自体にコストがかかり、製品は大量に販売され、このため設計上のどんな誤りでも最終的に結果は一層悪くなる。したがって、現実に使えるプロトタイプに高いコストを要しても正当化される。情報システムにおいて“現実に使える”プロトタイプと呼ばれるものは実際に完成したシステムである。このようなプロトタイプに修正を加えることは、本番のシステムに修正を加えるのと同じくらいコストを要し得られるものはほとんどない。情報システム工学分野で、プロトタイプを最終的に完成されたシステムという場合、そのプロジェクトが本当にその目的を達成しなかったことを包み隠すための方便となることがよくある。この用語を誤って使うことによって、開発者たちは前もって将来の問題の言訳をしているのである。

アプリケーション・パッケージの場合は大量に生産されるが、実験的にテストするため、その実用システムを完全に作ることは経済的に非常に意味がある。

５．３．１　概念の定義

広く使われ、また正しく理解されている“プロトタイプ”という用語と異なり、“プロトタイピング”という用語は情報システム開発の分野に携わる人達だけのものとなろう。この用語は自動化の分野では、他と異なる特別の意味をもつものとして使われてきた。本項ではプロトタイピングを次のように定義する。つまり、プロトタイピングは、システム要求を定義するためのアプローチであり、何度も繰返しを行なうこと、ユーザーが開発工程に深くかかわること、広〈プロトタイプを利用すること、によって特徴づけられる。

プロトタイピング・アプローチは、様々な状況でいかにうまく１つのシステム要求を定義するかについて新たな見解を示している。プロトタイピングが重要なのは、プロトタイプが紙の上でのモデルより優れたコミュニケーション方法を確立でき、また繰返し作業は不可欠な学習過程を正しい方向へ導くために必要だということである。

要求定義へのアプローチ

“プロトタイピング”という言葉は要求定義段階の小さな部分（いわゆるプロトタイプが実現され、繰返しのサイクルで修正される部分）のみを指すのに使われることがよくある。その他の場合、この言葉は従来の情報システムの開発方法にまったく置き換わる開発アプローチとみなされる。しかし、さらにもう１つ共通の解釈があり、プロトタイピングを伝統的な要求定義方法に代わるものとみなす。このようにプロトタイピングは要求定義のアプローチであり、開発アプローチではない。プロトタイプを構築できるという事実は、これらの手法・技法を適用できる方法があるということである。本項では、手法・技法の適用に変更を加えることを、プロトタイピング工程それ自身の一部とみなしている。このように“プロトタイピング”という言葉を使用する場合、プロトタイプと従来の手法・技法を組み合わせて要求定義の全工程を行なうことを指し、要求定義段階でプロトタイプが実際に作られ評価される部分のみを指すことではない。

繰返し

プロトタイピングの基本的な前提は、直接経験したことのないシステムに対し、ユーザーが要求や要望を示すことはできないということである。プロトタイプを使用するかどうかは、ユーザーおよび情報分析者の問題に対する見識に依存し、またプロトタイプを使えばユーザーが経験しなければならない学習過程を支援することになる。この過程で、プロトタイプには繰り返し変更が加えられ、モデル化された側面／サブシステムに対してユーザーの要求や要望が明確になるまで続けられる。つぎつぎと変更されるプロトタイプによって、“本当”の要求により近いものとなってくる。プロトタイプを基に、新たな要求や要望が明らかになり、ユーザーと開発者が互いに正しく理解し合ったか検証できるようになる。

ユーザーの参加

プロトタイピングの工程に参加するには、ユーザーは時間を相当必要とする。この問題についての経験的な資料はほとんどないが、プロトタイピング工程への参加は従来の要求定義段階への参加よりはるかに多くの時間を費やすと想像される。プロトタイピングにおいてユーザーは積極的な参加者であり、プロトタイプを評価し、改善点を提案し、また同時に情報システムへの自分自身の要求と要望について見識を深め続けるのである。従来のシステム開発アプローチでのユーザーの役割は比較的受身であり、主導権は情報分析者の側にある。さらにユーザーのシステム要求定義記述言語に対する理解は、常にほんの限られたものであり、このためユーザーが初期の定義段階とその結論に積極的にかかわるようになるのは稀である。

５．３．２　プロトタイピングの手順

プロトタイピングの方法を取る要求定義段階は、５つのサブフェイズに分けることができる（図５．３－１）。

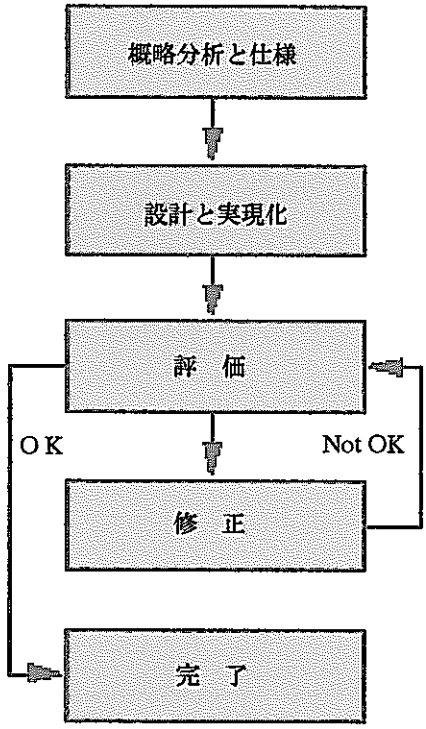

図５．３－１　一般的プロトタイプ手順

最初のサブフェイズ（“概略分析と仕様”）では、包括的な用語を用いて問題を分析し、以降のサブフェイズの強固な基盤を作り上げる。２番目のサブフェイズ（“設計と実現”）では、先進的な第４世代言語を用いてプロトタイプを構築する。第３番のサブフェイズ（“評価”）で、このプロトタイプはエンドユーザーに渡され評価される。もし仕様に何らかの調整が必要となれば、この必要な修正は第４番目のサブフェイズ（“修正”）でプロトタイプに対して行なわれる。この後、工程は前のサブフェイズに戻る。評価と修正のこのサイクルは、プロトタイプによってモデル化した側面の不確実さが減り、ユーザーに受け入れられるまで続けられる。最終サブフェイズ（“完了”）で、システム要求定義は性能と品質への要求を満たしたものとなる。

プロトタイピングのサブフェイズは、多くの製造分野の伝統的なライン工程のそれと似ている。しかしながら、重要な違いがある。つまり、プロトタイピングの工程は全開発工程の要求定義段階で行なわれ、個々のプロトタイピングのサブフェイズは非常に迅速に作成される。プロトタイピングは要求定義段階でミニチユア版のライフサイクルを作り上げる（Boar、1984；図５．３－２）。しかしながら、プロトタイピング作業結果としてのシステムは単にプロトタイプにすぎず、実用の情報システムとみなすことはできない。というのも、プロトタイプは一般に完全品から程遠く、わずかな側面のみを特別に重要視しているからである。プロトタイプを本番システムの基礎として残して置くことがしばしばある。そのほかの場合、プロトタイプは捨てられ、以降の本番システムの設計・開発作業は従来の方法で実行されるが、信頼性のある確実なシステム要求定義からそのシステムは生まれることになる。

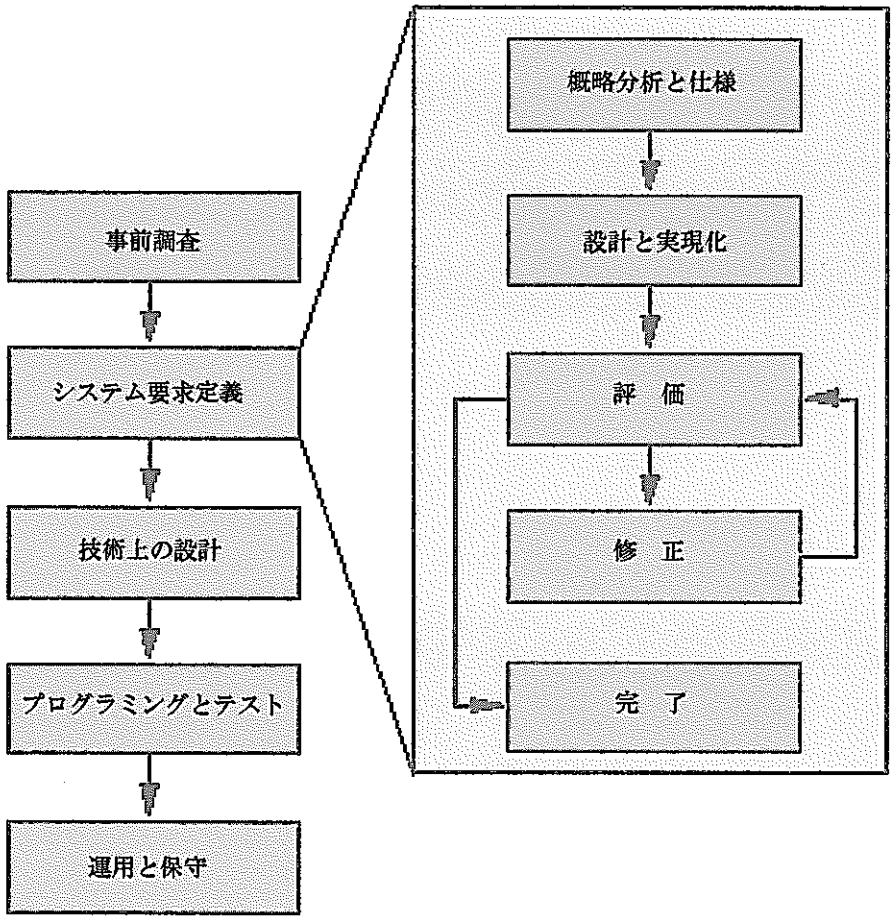

図５．３－２　開発工程とプロトタイピング

プロトタイプ・システムを実現し、それを何度も繰り返して精緻化していくには、強力な言語とツールなしでは不可能である。先進的なツールがない場合、プロトタイプの構築はあまりにも費用も時間もかかりすぎ、このため費用に対して効果は上がらないことになる。しかし、時に超高級言語と呼ばれる言語を使用すれば必然的にプロトタイピングと同じになると思われているが、それは誤りである。というのは、これらの言語は迅速なシステム開発に役立つことから、そう思われているのである。

５．３．３　設計段階の技法

プロトタイピングは、単に技術的設計段階にだけ役に立つとよくいわれる。そこで情報システムがシステム要求定義を満たすシステムにするには、様々な技術的設計を比較しなければならない。いったんシステム要求定義がなされると、最終目的のシステムに向かって一直線に作業することは可能であるが、そこでは経営上・管理上の性格をもつシステムがかかわってくるのは確かである。技術的設計段階における一般的な理解は引き続き高まっており、指針・規則・技法はこの段階で使用できるようになってきている（たとえば、ソフトウェア設計における“結合／束縛（coupling／bounding）”の原則やデータベースのアクセス経路分析など）。情報システムを設計することは芸術ではなく科学に

なりつつある。つまり、設計者がプロトタイプを構築する必要性はなくなりつつあるのである。

しかしながら、システムは非常に複雑になってきており、現在使える設計手法や技法では不十分なことがある。このような場合、設計者は純粋に分析的な方法で適切な技術的設計にたどり着くことはもはやできなくなりつつあり、試行錯誤のアプローチが不可避となっている。われわれの観点からすると、これはプロトタイピングと呼ぶべきではない。このようなアプローチは繰返しとプロトタイプの利用によって特徴づけられるが、プロトタイピング定義の第3番の重要な要素であるユーザーの参加が忘れられている。繰返しは、多くの開発作業における特質であるが、これだけの理由によってプロトタイピングと呼ぶべきではない。同様に、プロトタイプを採用しているからといって、あらゆる開発戦略をプロトタイピングと呼ぶべきではない。

非常に複雑なシステムの技術的な設計を行なう場合、システムの数学的モデルが重要な役割を果たすことがよくある。たとえば、シミュレーション技術を使えば、設計者は目的のシステムができたときにどれだけのレスポンス・タイムを期待できるか判断することができる。これはプロトタイピングの形で見受けられることがある。しかしながら、ここでは“シミュレーション”という言葉のほうがもっと適切である。

5．3．4　プロトタイピング技法の分類

様々なプロトタイプがあるように、プロトタイピングにも様々な種類がある。文献を見ると、実に様々な分類がなされている。まず、プロトタイプでモデル化される特定の側面を基に分類がなされる。システムのユーザー・インターフェイスのモデル化に重点を置いたプロトタイピングは、一般にユーザー・インターフェイス・プロトタイピングと呼ばれている。もし実際の機能をプロトタイプに付け加えると、機能型プロトタイピング（functional prototyping）と呼ばれることが多い。

2番目のアプローチは、開発工程のタイミングによってプロトタイピングを分類する方法である。これをFloydは発見型・経験型プロトタイピングと呼んでいる。発見型プロトタイピング（exploratory prototyping）は、ユーザー要求を見つけ出すのを目的としており、本項で述べるところのプロトタイピングである。

経験型プロトタイピング（experimental prototyping）は技術的設計で用いられ、その目的とするところは提案された解決法が適切であるかどうかを判定することにある。関心の対象となるのは、予想される処理負荷をシステムが処理できるか（性能型プロトタイピング、performanceprototyping）、あるいは適切なハードウェアを選択できるか（ハードウェア・プロトタイピング、hardware prototyping）である。

本項で述べるプロトタイピングは、常に要求定義段階で行なうことから、開発工程のタイミングを基にした分類は意味がなく、したがって“発見型”、“経験型”という言葉は用いない。

3番目のFloydや、その他の多くの人々が挙げるプロトタイピングは進化型プロトタイピング（evolutionary prototyping）である。この開発アプローチによれば、プロトタイプは本番システムへと発展して行く。したがって、本番システムの各バージョンは次のシステムのプロトタイプとして使用される。しかしながら、この種のプロトタイピングは、実際には発見型・経験型プロトタイピングとひとまとめに扱われるべきではない。これは、その後に続くプロトタイピング・アプローチの段階を指すのではなく、プロトタイピング作業が終了した後にプロトタイプを利用することを指しているのである。

この基準は、文献に見られるもう１つ別のプロトタイピングに対するアプローチの分類の基になる。もしプロトタイプを本番システムのベースとして用いるならば、保持型（keep-it）プロトタイピング、あるいは上で述べたように進化型プロトタイピングと呼ばれることになろう。またプロトタイプを捨て去るのであれば、廃棄型プロトタイピング（throw-away prototyping）という言葉が一般に使用されることになる。しかしながら、最後の２つの用語はあまり好ましくない。プロトタイピング作業の終了後プロトタイプに起こりうることは、プロトタイピング作業それ自体とはほとんど関係ない。さらに“進化型プロトタイピング”という言葉は要求を獲得するためではなく、プロトタイピングをシステム開発の戦略としてみなすことを意味している。

５．３．５　プロトタイピング・ツール等

本項では、要求定義を作成するための“アプローチ”としてプロトタイピングを定義した。プロトタイピングは漠然とであるが“手法”あるいは“技法”を指すことがある。これは不幸なことである。というのも、これらの用語はそれぞれ情報システム工学の分野で異なる意味をもっており、そして両方ともプロトタイピングに正しく適用できないからである。“手法”と“技法”という両方の用語は簡単に置き換えできるのではなく、また“手法”や“ツール”と置き換えできるものでもない。ここでのプロトタイプの定義を具体化するため、“アプローチ”という言葉とこの分野ではライバルと思われている手法、技法、方法論、ツールについて注意深く調べる必要がある。

アプローチ

アプローチ（戦略）は、開発工程の広い範囲の特徴、たとえば全員が従うべき指導要綱や主要な中間目標といったものを定義する一方、詳細な活動方針はそれぞれ異なるレベルや時点で決定するよう残しておく。それぞれの情報システム開発の状況において、プロジェクト・レベルの戦略決定は次の事柄が関心の的となる。

（１）社内開発かパッケージの購入か

開発はエンドユーザーによって行なうか、情報システム部門で行なうか

・開発工程におけるユーザーの参加度合い

データ中心かプロセス中心のアプローチかの選択要求定義段階でプロトタイピングを選択することは、より詳細な決定は後にされるものとして、ユーザーの参加や繰返しといった原則と中間目標（図５．３－２で示したサブ・フェイズ）を決定することになる。

・手法

情報システム開発において、“手法”という用語は特に規定された一連の行動と開発工程の特定の側面や部分を取り扱う標準を意味するのに用いられるのが一般的である。各段階で作り出される文書（“成果物”）についても定義するのが手法の代表例である。戦略が工程の一般的な特徴のみを指示するのと対照的に、手法は与えられた目的を達成するために実行すべき作業を詳細に指示する。

戦略と手法を区別する重要なもう１つの特徴はしばしばそうであるが、手法が独占的でかつ専有権

をもっていることである。よく知れ渡っている手法としては著作権があり、別の製品として、あるいは“方法論”の一部として商標名のもとに販売あるいは市場に出されている。

手法は大きく２つに分けることができる。つまり、情報システム開発プロジェクトを管理するためのプロジェクト管理方法（たとえばArthur AndersenのMethod／1やDutchfirm Pandata（Hice他、1987）のSDM）、それにこのプロジェクトを実行するための開発手法（たとえばYourdonの構造化設計）である。本項では特に断らない限り“手法”の用語は開発手段のみを指すことにする。

・技法

手法というのは一般的に、１つか２つの技法を定め、特定の仕事を遂行する場合に用いられる。手法はシステム開発の“何をすべきか”という側面を取り扱うが、技法は“いかに行なうか”という面を扱う。手法と同じように技法はプロジェクトの管理と統制（たとえばPERT）、あるいはプロジェクトの実行（たとえばデータフロー・ダイアグラム）のいずれかに適用できる。手法とは逆に、技法は非専有であり、学術機関のルートを通して入手できるようになってきている。一般に手法は多数の技法を合成したものであり、特定の技法は様々な広範囲の競合する専有の手法に利用されている。

・方法論

手法は上記で定義したように、開発工程のある特別の段階あるいは側面を取り扱うのが一般的である。実際、範囲を完全にカバーするにはいくつかの手法を組み合わせる必要がある。手法を有効に効果的に結合するには、重複があってはいけないし、また完全に定義されたインターフェイスがなくてはならない。

いくつかのこのような広範囲な一連の手法が専有の製品として市場に出されている。James MartinのInformation Engineeringがその例である。このような製品に対する一般的な用語が“方法論”である。

様々な手法を統合化し、“方法論”の名称を得るには、情報システム開発にまつわる問題を解決する確たる理論あるいは哲学を具現化するか、少なくとも取り込まねばならない。実際問題として、現実は方法論の背後にある哲学が常に明示的であるとは限らず、特定できるものでもない。そしてわずかな例しかないが、構成要素を結び付けるロジックは市場が求める“ロジック”以上ではありえないように見える。しかしながら、適切な言葉がないため、複数の専有の手法を組み合わせて開発工程全体をカバーする（あるいはカバーしようとする）ことを表わす一般用語を“方法論”とし、それを今後も使っていくことにする。プロトタイピングは特定の専有的な手法と結び付いてはいず、また要求定義段階以外をも狙っているわけでもないので、プロトタイピングを方法論と誤まることもないであろう。

・ツール

システム開発面において、また本項において、“ツール”という用語は様々な仕事の助けとなるコンピュータ・ソフトウェアに限って使用する。ツールは、たとえば作図プログラムといった範囲の限られたものから、分析者用ワークベンチ、あるいは第４世代開発環境といった非常に広範囲なソフトウェア群まで多種多様である。特定の技法を支援する所有権付きのツールと、特定の手法を適用するのに助けとなるツールがある。多くの専有権をもつ手法は、ひと揃いのツールとともに販売されてい

る。ツールの中には、様々な手法あるいは技法を結び付けるのに役に立つのもある。この意味でツールとプロトタイピングを混同してしまう危険はほとんどないが、手法と技法、およびそれらを支援するツールとの間の違いを明確に区別しておくことは非常に重要である。

5．4　プロトタイピング対構造化手法

プロトタイピングは手法でも技法でもないが、決して構造化手法や技法と矛盾するものではない。これからの議論で明らかになってくる。

プロトタイプはシステムの別の面をモデル化する

システムの異なる側面は、それをモデル化するために、それぞれ別の“言語”を必要とする。たとえば、データフロー・ダイアグラムといった、トップダウンの図形を用いるモデル化技法は、主にシステムが満たされねばならない機能をモデル化するのに適している。

情報の構造をモデル化するためには、たとえばChenやMeriseのエンティティ・リレーションシップ・ダイアグラムが使われることになろう。しかしながら、情報の構造を表現するのに適用される規則は複雑であることが多いが、それを記録するには図形による記法は適切でない。このためには（半-）形式言語のほうがより適している。

プロトタイプは、ユーザー・インターフェイスをモデル化するのにまずは最も有効な方法である。このことは、システムの外観をすべてプロトタイプから学びうるといっているわけではない。プロトタイプの開発を通して、解決しようとするシステムの問題や全体的な要求をより深く理解することが一般的に可能となってくるのである。しかしながら、たとえばシステムやデータベース構造といった問題の領域をモデル化し文書化するためには、従来の作図技法のほうがプロトタイプより優れている。さらに従来の手法のほうが分析工程のより大きな助けとなる。多くのプロセス指向の開発手法の特徴であるトップダウン・アプローチにより、複雑な問題は処理可能な部分に分割することができる。データ構造のモデル化のためのプロトタイプはほとんど役に立たない。従来の手法・技法を引用して十分に問題を調査し、その問題を解決するための情報システムの広範囲な概要を固めた後でのみ、プロトタイプを開発するのは意味がある。

プロトタイプは、従来の手法や技法より、主としてシステムの様々な面をモデル化することになる。したがって、プロトタイプは要求定義段階の情報分析者が利用できる救援の兵器庫を増やすことになるが、決してそれらの代用とみなすことはできない。

プロトタイピング・アプローチが取られる要求定義段階で、従来の分析や文書化の方法や技法の機能は、システムに求められているデータ構造や機能をモデル化するのに役に立つ。ユーザー・インターフェイスをモデル化するため、また開発者と将来のシステム利用者とのコミュニケーションの手段としてプロトタイプは位置づけられる。現在の紙の上に作成するモデルよりはるかに勝るコミュニケーション媒体として、プロトタイプがいかに優れているかは前章で述べた通りである。

プロトタイプもまたシステムである

プロトタイプは１つのシステムの側面、あるいは部分のみをモデル化するのが一般的であるが、依然プロトタイプはシステムなのである。つまり分析し、設計し、実際に開発しなければならない。さ

らにプロトタイピング工程での繰返しの観点からすると、プロトタイプは“現実”のシステムと同じように柔軟で理解しやすいものでなくてはならず、多分それ以上かもしれない。本番システムのベースとしてプロトタイプの一部を再利用し、プロトタイピングの作業を活かそうとするなら、再利用する部分は高品質でなくてはならない。このためには統制のとれた構造化されたプロトタイプの開発が必要となる（Riddle、1984）。

したがって、プロトタイピングは構造化手法や技法が出現する前に一般に取り入れられていた構造化されていない作業方法と同一視してはならないことは明らかである。

残念ながら、そのように解釈されることが多いのも事実である。プロトタイピング・アプローチは何も“目新しい”ことではなく、いままで常に行われてきた方法だといわれることもある。プロトタイピングはシステム関係者の間での“行為者”にとっての“アリバイ”、つまり言訳になりつつある。これらシステム開発者は（多少安心して）かつての“古き良き時代”と同じように構造化分析を無視し、すぐさまプログラミングに取りかかっているのである。このような状況下では、プロトタイピングが構造化されていない、テスト不可能な、そして保守ができない情報システムになってしまうとの懸念は最もなことになる。プロトタイピングは過程もその結果も管理できるように構造化された方法で行なわれなければならない。

５．４．１　関連概念

“プロトタイピング”という用語は、多かれ少なかれ関連する概念で混乱を招くこともあり、プロトタイピング・アプローチを正しく評価するのに大きな妨げとなっている。というのも、この長所・短所についての無意味な議論は、このような誤った組合せを基にしているからである。

進化型開発

プロトタイピングに関連はしているが、明らかに別な概念である良い例として、進化型開発（evolutionary development）がある。このアプローチでは、本番システムが柔軟で拡張可能であることを重要視していることから（Land、1982）、要求が変わるとシステムの修正版がすぐさま実現される。同じような状況は、新しいプロトタイプがユーザーの要求変更に合わせて作られる場合に見受けられることがある。進化型開発とプロトタイピングの間の基本的な相異は、進化型開発の目的が本番システムそれ自身の後続版を迅速に実現するのに対し、システムの（一部の）プロトタイプはプロトタイプ作成作業、つまりプロトタイピングに含まれることにある。この相違は明らかに学術的興味を越えるもので、結局プロトタイプと本番システムとはまったく異なる要求を満たさねばならぬということである（たとえば信頼性、性能、あるいは完全性の観点から）。この２つの戦略（アプローチ）は異なる目的をもっており、開発期間で用いられるいろいろな方法に反映されている。したがって、しばしば文献で示唆されているように、プロトタイピングと進化型開発を組み合わせてうまく使うことは決してできない。それぞれの戦略はまったく異なる問題状況を目指しており、このため２つの戦略を組み合わせねばならぬ理由は何もない。プロトタイピングは“現実”の要求が存在していることを前提にしている。ユーザーが実際に必要とするものだけを構築するには何度も繰返しを行なう必要があるが、最終的に要求定義は固まることになる。一方、進化型開発にとっても基本的な前提は要求が絶えず変化しがちであるということである。このような状況では、繰返しの工程に決して終りがないので、プ

ロトタイピングでは何の意味もない。

プロトタイピングと進化型開発との違いについて線引きを誤ると、間違った結論に到達することになってしまう。激しい変化に対応しなくてはならない環境では、プロトタイピング・アプローチが求められるとよくいわれる（たとえばKauber、1985参照）。この結論が不適切であることは明らかである。つまり、このような環境は最終の結果が柔軟であること、言い換えれば進化型開発アプローチを求めているのである。

増加型開発

進化型開発アプローチに近い関係をもつ概念で、よく間違えられるのが増加型開発（incremental development）である。増加型開発アプローチでのシステム開発は全部ではなく、指定された要求の一部分を満たすシステムの最初の版を出そうとするところから始まる。残りの要求は後の版で満たすことになる。このアプローチによって不完全ではあるが、少なくとも使用に耐え、重要な情報ニーズの一部を満たすシステムを早く使用できることから、ユーザーは利益を得ることができることになる。システム要求定義の完了から要求したシステムの稼動までに時間がかかりすぎるという不満がしばしば聞かれるが、増加型開発はこの問題に対する有効な解答である。

進化型および増加型開発いずれの方法で開発されようとも、次の版の性格は明らかに異なる。進化型開発では全システムの新しい版がそのつど開発される。増加型開発の場合、全体としてのシステムは一歩ずつ作り上げられ、つぎつぎ出てくる後続の版は変更を加える前の版に数多くの新たな機能を加えたものである。明らかにこれは理想的である。実際この２つのアプローチは厳密に区別できるものではない。たとえば、進化型アプローチでのシステムの新版には新機能が含まれることもある。とはいえ、進化型と増加型開発の違いを保っておくことは、これら２つの開発戦略が異なる目的をもち、したがってプロジェクトと完成したシステムとは様々な組合せを必要とすることから意味がある。もちろん、システムの機能を増しながら同時にシステムの特定の構成要素を開発するため、進化的なアプローチをとることは可能である。同様に、増加型開発とプロトタイピングとの組合せもうまくいく。問題の性格に照らし合わせ、開発すべきシステムの各構成要素に対し何が最高の開発アプローチかということに考慮を払うべきである。

ユーザー参加型開発

一様に、プロトタイピングと混同してしまうもう１つの開発アプローチは、ユーザー参加型開発（participative development）である（たとえばHirschheim、1983を参照）。このアプローチでは、開発工程にユーザーが深く絡むことになる。プロトタイピングの工程がユーザーの積極的な役割によって特徴付けられるからといって、ユーザーの参加に力を入れているすべての開発アプローチがプロトタイピングと同じであるとは言っていないのである。ユーザーが高度の繰返し作業に参加し、プロトタイプが広く利用される場合のみプロトタイピングの言葉は正当化される。

エンドユーザー・コンピューティング

エンドユーザーによるアプリケーション開発（エンドユーザー・コンピューティング）とプロトタイピングは同じようなものだと誤解されることがある。再度言うが、この２つのアプローチは関係な

い。つまり、一方は開発作業を実行する人ないしはグループについて述べており、他方は開発する方法について述べているのである。もちろん、ふさわしい経験をもつユーザーは、プロトタイピングに似たアプローチを用いてアプリケーションを実現することができるが、普通のエンドユーザーはプロトタイピングに必要なスキル（熟練度）を持ち合わせていない。というのも、本項で述べる構造化アプローチは特に情報分析者を意図しているからである。用語の混乱を避けるため、エンドユーザー自身による（小規模で簡単な）アプリケーション開発を指す場合は“プロトタイピング”ではなく“エンドユーザー・コンピューティング”という用語だけを使うほうがよい。

同義語

“プロトタイピング”のいくつかの同義語（synonym）に関する注釈が適切であろう。同義語あるいは、それに近い用語にヒューリスティック開発（Berrisford and Wetherbe、1979）と適応性のある設計（Alavi、1984 b）がある。これらの用語は一般的に明確には定義されていない。このような用語を用いて自分が述べるアプローチと、自分のプロトタイピングの解釈との間には小さな相違があることを示そうとしている例がいくつかある。しかしながら、様々な開発アプローチの本質にはまったく違いはない。

5．4．2　その他のアプローチ

情報システム開発者は、情報システムに対するISOの3レベル・アプローチによって、情報システムを概念化する一般的なモデルを得ることができる。情報システムの側面あるいは“レベル”は、このアプローチによって3つに分けられる。つまり、情報レベル、データレベル、内部レベルである。情報システムは互いに独立に開発されるが、開発者はその情報システムのあらゆる側面をこのアプローチによって調べることができ、また開発作業が複雑にならないようにすることができる。

3レベルの原則と、特にこの原則が持ち込んだ概念と外部レベル間を厳格に区別することは、システム要求定義の工程に大きな影響を与える。つまりプロトタイピングの工程に、そしてプロトタイピングにおける概略分析と仕様の作成のサブフェイズに具体的な影響を及ぼすことになる。ここで分析作業のフレームワークとプロトタイピングの関係についていくつかの注意事項を述べることにする。

この関係を理解するには、まず“（システム）要求定義”が意味するところを正確に定義しなければならない。要求定義は、いまから開発するシステムに対するユーザーの要求をすべて組み入れたものでなくてはならない。要求定義は常にシステムの概念を完全に記述したものであり、そこには保持すべき情報のリストがあり、その情報のもとに動作する機能のリストがある。会話型のシステムが出現して以来、ユーザーもまた一般にシステムの外部形式に対する要求を定義するようになってきた。この場合、要求定義にはユーザー・インターフェイス（画面およびレポート（帳表）のレイアウト、それにダイアログ（会話）構造）を記述しなければならない。システム要求定義にはシステム内部の実現方法（たとえば、コンピュータの種類あるいはプログラム規模の最大値）についても記述することになるかもしれない。最後に、性能に対する要求もまたシステム要求定義の一部となる。これらの要求には応答時間、セキュリティ、信頼性の観点からのユーザー・ニーズが含まれる。

プロトタイピングが第1に目指すところは、システムの外部レベル、つまりユーザー・インターフェイスに対するユーザーの要求をより鮮明に理解することである。プロトタイプは、情報システムの

この側面をモデル化する最高の手段である。外部、概念、内部レベルは互いに直接的な関係をもっていることから、プロトタイプもまた概念と内部の側面を深く理解する１つの方法として利用できる。プロトタイピングを利用することによって、ユーザーと開発担当者は両方とも理解できる具体的なモデルを基にコミュニケーションを行うことができる。ユーザー・インターフェイスを議論することによって、開発担当者はユーザーからの世界の見方について、またその見方がどのように広がっていくかについて非常に多くのことを学べるのである。１つ例を挙げると、あるデータ項目が画面の特定の場所に特定のフォーマットで表示されねばならぬとユーザーが指定したとすると、開発担当者はユーザー・インターフェイスについてだけでなく、データ項目に対するユーザーのニーズについても何らかを学ぶことになる。

２つ目の例は次の通りである。データ値の特別な組合せを認められないとユーザーが伝えたとすると、開発担当者は概念スキーマに制約（あるいは規則）を付け加えることができる。いずれの場合も、プロトタイプは概念スキーマ（たとえば、エンティティ-リレーションシップで表現される通り）が完全でしかも正確であることを検証する手段として機能するのである。プロトタイプを基に何らかの性能要求について学ぶことすらできるのである。

ユーザーは、いったんプロトタイプを使い始めると、次には信頼性、応答時間、セキュリティなどについての要求を挙げることになる。しかし、これはプロトタイピング工程の副産物であり、性能要求も同じように認めることがプロトタイピングの主な目的ではない。一般にプロトタイピングの初期の工程では繰返し作業を行なう。この間、開発担当者はユーザーが概念的な側面に集中し（プロトタイプの完全性と正確性）、しばらくはシステムがどのような形でその役目を果たすか、つまりどのように実現されるかについて気を取られないように求める。たとえば、ユーザーには特定のこと（たとえば注文）について知りたいことがすべて画面に表示されているかどうかといったことが尋ねられる。この段階で、すべての誤りや欠落が明らかになる。システムの形（会話、画面、レポート（帳表）のレイアウト、データ・フォーマット）は後の繰返し作業で徹底的に検討される。そこでのシステムに対する変更は比較的小さくてすむ。プロトタイプではユーザー・インターフェイスをモデル化することから、プロトタイピングを用いるプロジェクトが概念的側面を無視する危険性がある。この場合、開発担当者は自分はいま外部側面ばかりに集中していることに気づき、他の側面についても注意を払うべきであることを自覚しておかねばならない。プロトタイピングが従来の手法を不要としているのではないことはすでに強調してきた通りである。

５．５　試作システムの構成

地層処分ヴァーチャル・エンジニアリング・システム開発用のハードウェアは、大別して下記の５種類に機能分割した。本項では、試作したシステムの仕様を一覧する。

機能分野１：シミュレーション用システム（Visual Environment Simulation System）
機能分野２：操作用システム（Operator Console System）
機能分野３：データベース用システム（Data Base System）
機能分野４：パラメータ・ディスプレイ用システム（Status Display System）
機能分野５：オーディオ・ビデオ用システム（Audio Video System）

５．５．１　全体システム概念構成

図５．５－１に試作した地層処分ヴァーチャル・エンジニアリング・プロトタイプ・システムの概念構成を示す。

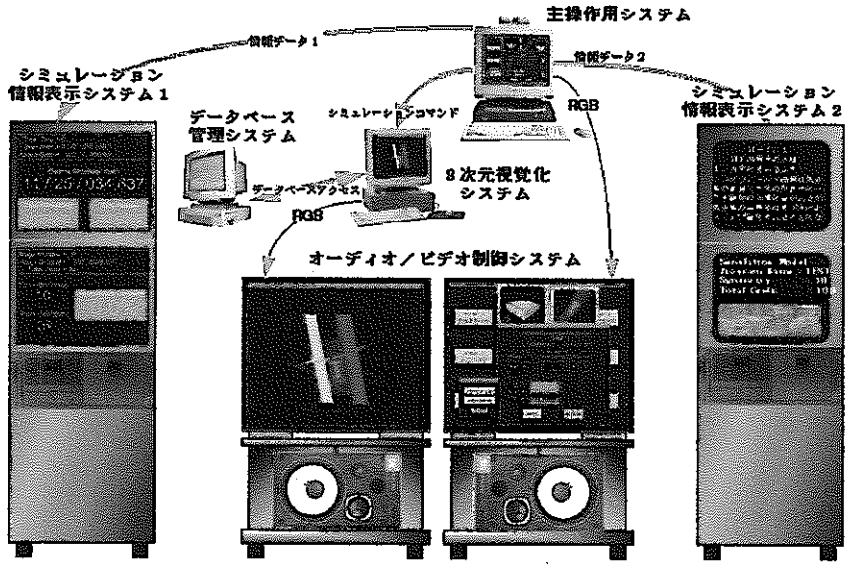

図５．５－１　システム概念構成図

５．５．２　多重並列分散動作機能概念

図５．５－２に試作したネットワーク並列処理機能の概念構成を示す。

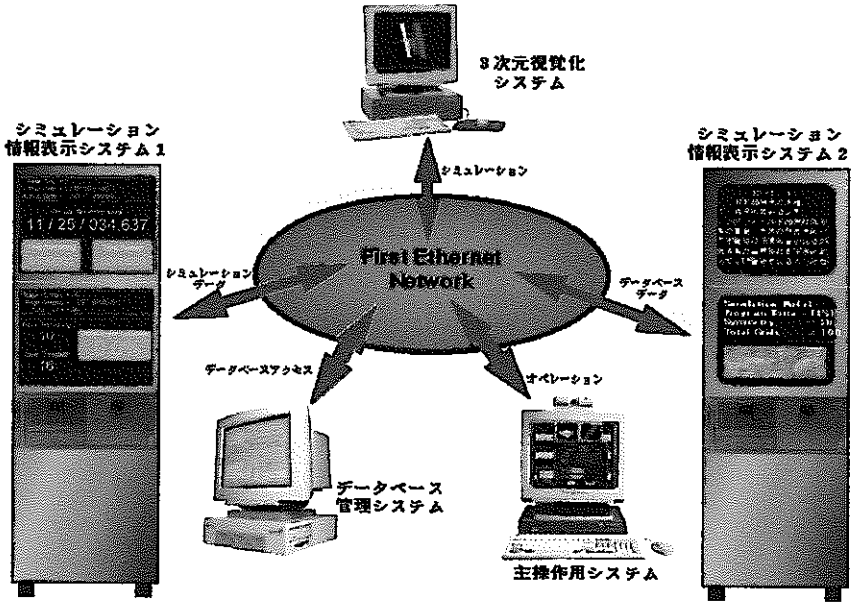

図５．５－２　ネットワーク並列処理機能概念

５．５．３　ビデオ信号制御仕様

図５．５－３に試作したビデオ信号処理系を示す。

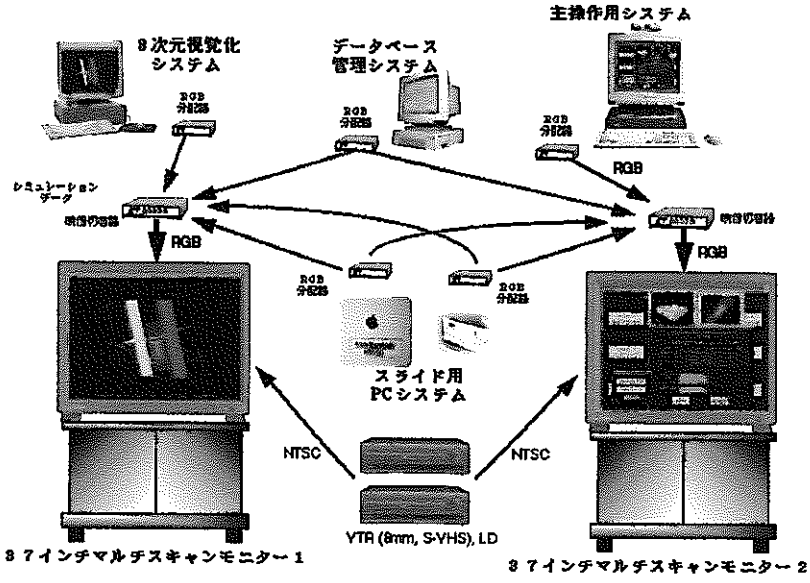

図５．５－３　ビデオ信号処理系

５．５．４　オーディオ信号制御仕様

図５．５－４に試作したオーディオ信号処理系を示す。

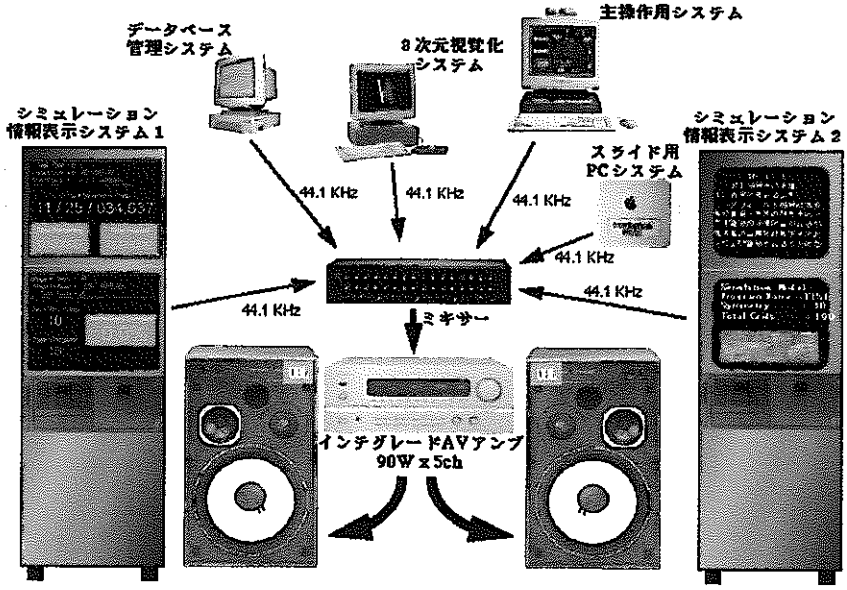

図５．５－４　オーディオ信号処理系

５．６　シミュレーション用システム・ハードウェア

シミュレーション用システムは、地層処分性能評価にかかわる数値計算を行うとともに、３次元グラフィックス機能を利用し、ビジュアルなシミュレーション結果の視覚再現を行うシステムとして構成した。システム・ハードウェアはシリコン・グラフィックス社製（以後、ＳＧＩとする）Indigo2を利用した。

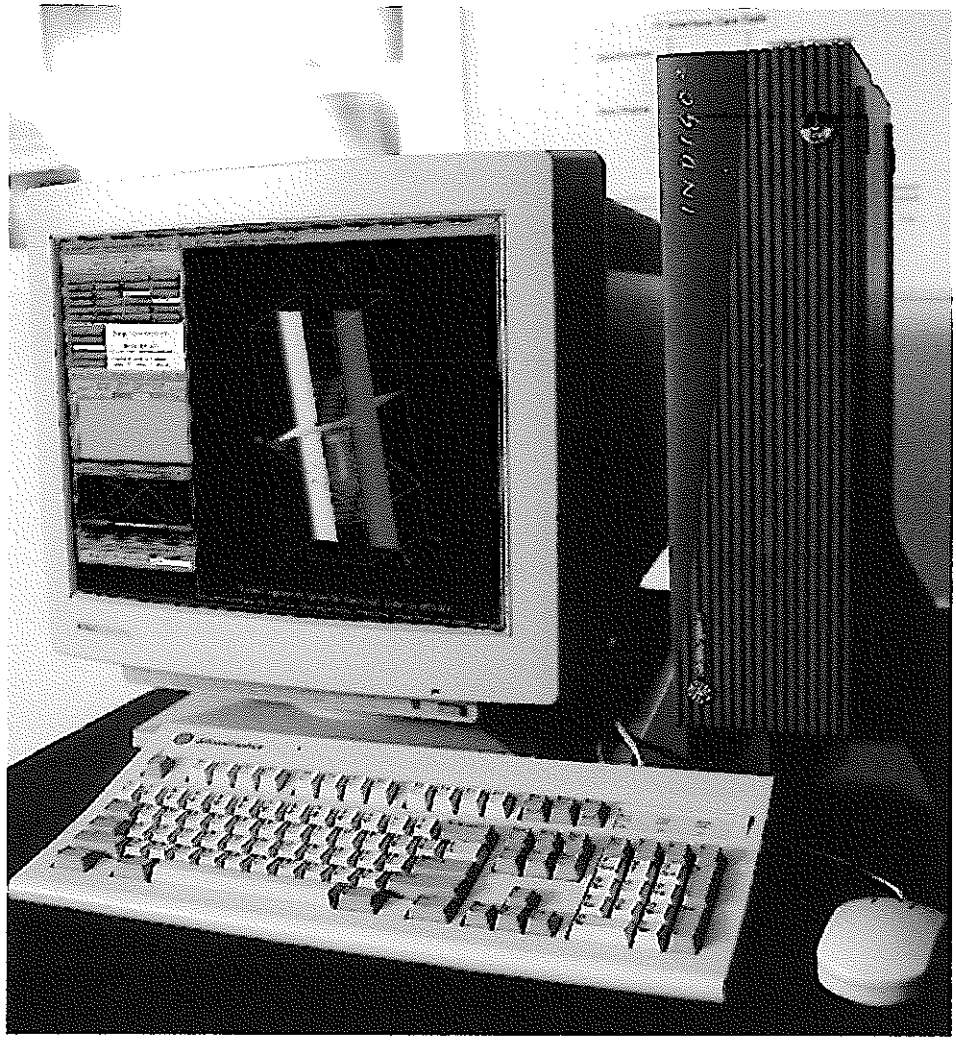

図５．６－１　システムの外観

（１）処理装置仕様（Processor Specification）

用途　　：解析コードの実行等を処理する為に用いる

選定論拠：３Ｄグラフィクス機能を内蔵していること

型式　　：SGI Indigo2 Extreme

仕様　　：MIPS R4400/150MHz

数量　　：１

（２）主記憶装置仕様（Main Memory Specification）

用途　　：シミュレーションの実行等をメモリー上に常駐させる為に用いる

選定論拠：９６ＭＢ以上のメモリを内蔵していること

型式　　：メモリ（Memory）

仕様　　：96MB

数量　　：１　空スロット無し

（3）磁気ディスク装置仕様（Hardware Disk Specification）

用途　　：システムＯＳを格納する為に用いる

選定論拠：２ＧＢ以上の容量があること

型式　　：System Disk (From Above)

仕様　　：2BG Disk (1.03GB + 1.03GB Internal Disk)

数量　　：2

（4）追加ディスク装置仕様（Additional Hardware Disk Specification）

用途　　：シミュレーション・ソフトウェアを格納する為に用いる

選定論拠：８ＧＢ以上の容量がある事

型式　　：Work Disk

仕様　　：8.6 GB External Disk (4.3GB + 4.3GB)

数量　　：2

（5）モニター仕様（Monitor Specification）

用途　　：シミュレーション結果を確認する為に用いる

選定論拠：1280 x 1024の解像度をサポートする事

型式　　：Stand Alone 20 inch Monitor (From Above)

仕様　　：20 inch

数量　　：1

（6）基本ソフトウェア仕様（System Software Specification）

型式　　：IRIX Operating System (Include Current System)

仕様　　：IRIX6.2

数量　　：1

用途　　：ＯＳとして使用する

選定論拠：AT & T UNIX System V ＋ Berkeley BSD準拠で、64Bit, 32bitをサポートする事

型式　　：Fortran Compiler (Include Current System)

仕様　　：Fortran 77 Compiler for IRIX6.2

数量　　：1

用途　　：Fortranプログラムの開発、保守に用いる。

選定論拠：解析コードとの整合性をとる事

型式　　：Open Inventor (Include Current System)

仕様　　：Open Inventor 3D Toolkit

数量　　：1

用途　　：3Dグラフィクスオブジェクトの開発に用いる

選定論拠：SGIマシン上で3Dグラフィクスオブジェクトを扱える事

型式　　：ImageVision Library (Include Current System)
仕様　　：ImageVision Library kit for IRIX6.2
数量　　：1
用途　　：画像処理アプリケーションに用いる
選定論拠：SGIマシン上でグラフィクスデータを扱える事

型式　　：ONC/3NFS (Include Current System)
仕様　　：Network File System for IRIX 6.2
数量　　：1
用途　　：ネットワーク間リソース共有に用いる
選定論拠：SGIマシンをネットワークを扱える事

5．7　操作用システム・ハードウェア

操作用システムは、シミュレーションにシナリオの組立、シミュレーションの実行、パラメータ・ディスプレイの設定等、本システムの全ての操作を行う対話システムとする。システム・ハードウェアはシリコン・グラフィックス社製Indyを利用した。

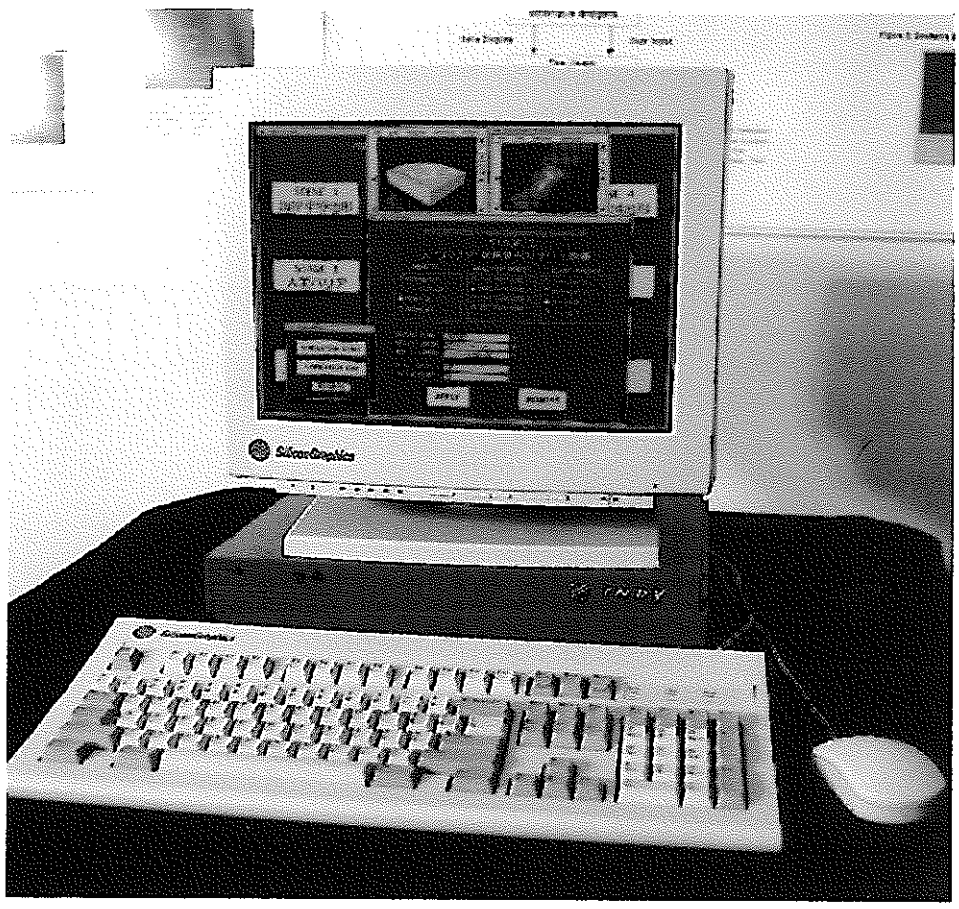

図5．7－1　システムの外観

（1）処理装置仕様（Processor Specification）

用途　　：シミュレーション等の操作を行う為に用いる
選定論拠：シミュレーション用システムとの整合性を計る事
型式　　：SGI indy
仕様　　：MIPS R4000/100MHz
数量　　：1

（2）主記憶装置仕様（Main Memory Specification）

用途　　：操作画面の実行等をメモリー上に常駐させる為に用いる
選定論拠：６４ＭＢ以上のメモリを内蔵していること
型式　　：メモリ（Memory）
仕様　　：64MB
数量　　：1

（3）磁気ディスク装置仕様（Hardware Disk Specification）

用途　　：システムＯＳを格納する為に用いる
選定論拠：２ＧＢ以上の容量がある事
型式　　：System Disk (From Above)
仕様　　：2BG Disk (1.03GB + 1.03GB Internal Disk)
数量　　：2

（4）モニター仕様（Monitor Specification）

用途　　：操作、または結果を確認する為に用いる
選定論拠：1280 x 1024の解像度をサポートする事
型式　　：Stand Alone 17 inch Monitor (From Above)
仕様　　：17 inch
数量　　：1

（5）基本ソフトウェア仕様（System Software Specification）

型式　　：IRIX Operating System
仕様　　：IRIX6.2
数量　　：1
用途　　：ＯＳとして使用する
選定論拠：AT & T UNIX System Ⅴ + Berkeley BSD準拠で、64Bit, 32bitをサポートする事

型式　　：NFS (Include Current System)
仕様　　：Network File System for IRIX 6.2
数量　　：1
用途　　：ネットワーク間リソース共有に用いる

選定論拠：SGIマシンをネットワークを扱える事

型式　　：Builder Xcessory
仕様　　：Builder Xcessory for IRIX
数量　　：1
用途　　：操作ＧＵＩの開発、保守に用いる
選定論拠：AT＆T X11 R4/R5をサポートする事

５．８　データベース用システム・ハードウェア

データベース用システムは、地層処分性能評価シミュレーションに必要な、評価シナリオ、廃棄物の工学情報、解析モデル・データ等、地層処分ヴァーチャル・エンジニアリング・システムの稼働に必要な情報を管理するシステムとした。

システム・ハードウェアは、サン・マイクロ社製SUN SPARC 10により構成した。

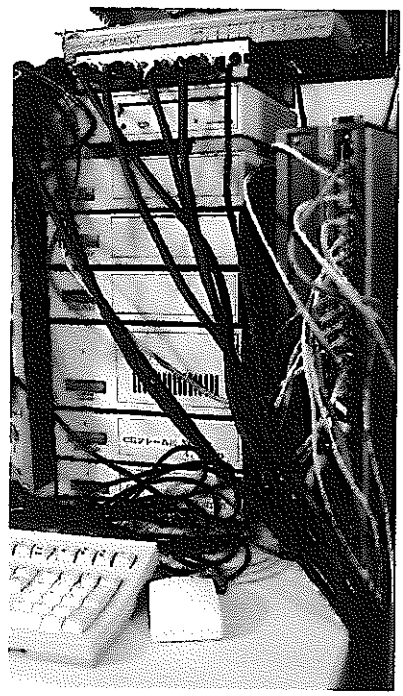

図５．８－１　システムの外観

（１）処理装置仕様（Processor Specification）

用途　　：データベースツール実行等を処理する為に用いる
選定論拠：AT＆T UNIX System Ｖ ＋Berkeley BSDをサポートしている事
型式　　：SUN SPARC 10
仕様　　：Model 10
数量　　：1

（2）主記憶装置仕様（Main Memory Specification）

用途　　：データベースツールの実行等をメモリー上に常駐させる為に用いる

選定論拠：３２ＭＢ以上のメモリを内蔵していること

型式　　：メモリ（Memory）

仕様　　：32MB

数量　　：１

（3）磁気ディスク装置仕様（Hardware Disk Specification）

用途　　：基本OSとデータベースツールで使用するデータを格納する為に用いる

選定論拠：１０ＧＢ以上の容量がある事

型式　　：System Disk + External Disk

仕様　　：2.1GB(From Above) + 9.0GB

数量　　：１

（4）モニター仕様（Monitor Specification）

用途　　：データベースツールを構築操作する為に用いる

選定論拠：1280 x 1024の解像度をサポートする事

型式　　：Stand Alone 20 inch Monitor (From Above)

仕様　　：20 inch Color Monitor

数量　　：１

（5）基本ソフトウェア仕様（System Software Specification）

型式　　：SUN-OS Version Solaris 2.1

仕様　　：Solaris 2.1

数量　　：１

用途　　：ＯＳとして使用する

選定論拠：AT＆T UNIX System Ⅴ + Berkeley BSDをサポートする事

型式　　：C Compiler

仕様　　：GNU (Free Soft-ware)

数量　　：１

用途　　：Ｃ言語プログラムの開発、保守に用いる

選定論拠：データベースツールを構築出来る事

型式　　：C++ Compiler

仕様　　：GNU (Free Soft-ware)

数量　　：１

用途　　：Ｃ++言語プログラムの開発、保守に用いる

選定論拠：データベースツールを構築出来る事

型式　　：Data-base Application
仕様　　：Postgres 95 (Free Soft-ware)
数量　　：1
用途　　：データベースツールとして用いる
選定論拠：配列型のリレーショナルデータベースとする事

5．9　パラメータ・ディスプレイ用システム・ハードウェア

パラメータ・ディスプレイ用システムは、地層処分ヴァーチャル・エンジニアリング・システムのシミュレーションにかかわるパラメータを表示するシステムとした。パラメータとは例えば、現在シミュレーション中の時刻、選択されたシナリオ、選択された廃棄体工学情報等とした。パラメータ・ディスプレイ用システムは、シミュレーション・システム等ネットワークで接続された、他のシステムと並列にメッセージ通信を行いながら多重処理動作を行う機能が要求されたため、市販の製品としては適当なものがなかったことから専用のハードウェアとして基盤等を調達して製作した。

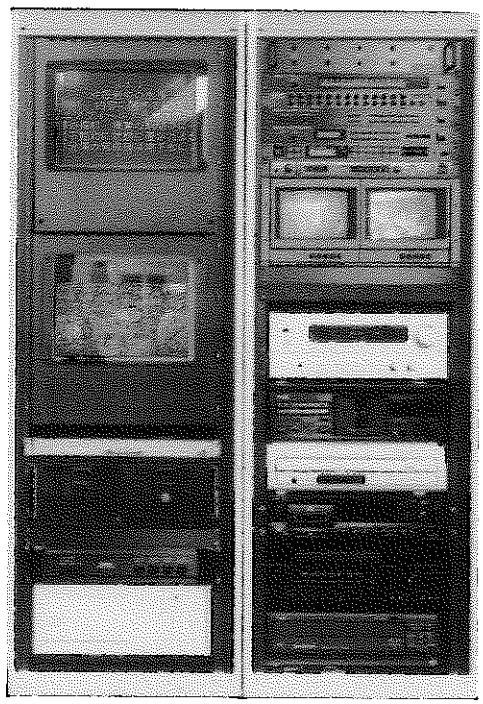

図5．9－1　システムの外観（右）

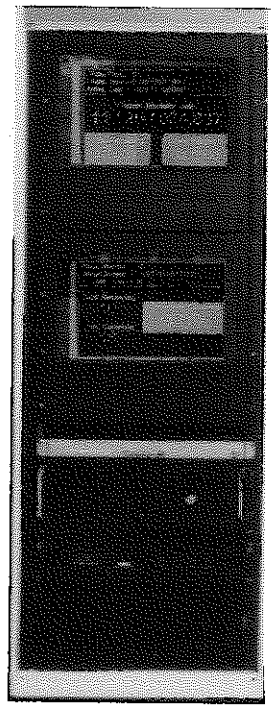

図５．９－２　システムの外観（左）

（１）処理装置仕様（Processor Specification）

用途　　：シミュレーション中の状態情報を実行する為に用いる

選定論拠：シミュレーション用、操作用システムと整合性がとれる事

型式（Type　）：GTA PCZ3

仕様（Spec,　）：Pentium 166MHz/32MB Memory

数量（Quantity）：２

（２）主記憶装置仕様（Main Memory Specification）

用途　　：状態情報の実行等をメモリー上に常駐する為に用いる

選定論拠：３２ＭＢ以上のメモリを内蔵していること

型式（Type　）：Memory

仕様（Spec,　）：32MB

数量（Quantity）：２

（３）磁気ディスク装置仕様（Hardware Disk Specification）

用途　　：基本システムと状態情報アプリケーションを格納する為に用いる

選定論拠：２ＧＢ以上の容量がある事

型式（Type　）：２GB Disk

仕様（Spec,　）：２GB Disk

数量（Quantity）：2

（4）モニター仕様（Monitor Specification）

用途　　：状態情報を表示する為に用いる

選定論拠：１９インチラックに取付けられ、且つ1280x1024の解像度をサポートする事

型式（Type　）：Rack Mount 17inch Color Display

仕様（Spec,　）：17inch

数量（Quantity）：4

（5）CD-ROM仕様（CD-ROM Specification）

用途　　：システム保守を行う為に用いる

選定論拠：ISO9660準拠とし８倍速以上の読み込み速度をサポートする事

型式（Type　）：8 Times CD-ROM Drive

仕様（Spec,　）：4xCD-ROM Drive

数量（Quantity）：2

（6）イーサーネット仕様（Ether-Net Specification）

用途　　：シミュレーション用、操作用システムとのメッセージ交換を行う為に用いる

選定論拠：シミュレーション用、操作用システムとの整合性がとれる事

型式（Type　）：Ether-Net Card

仕様（Spec,　）：First EtherLink PCI 10MB

数量（Quantity）：2

（7）グラフィック仕様（Graphics Specification）

用途　　：状態情報を１枚のグラフィックカードにより２画面に表示する為に用いる

選定論拠：２画面以上の表示を可能とし且つ2560x2048以上の解像度をサポートする事

型式　　：Graphics Card

仕様　　：Graphics Card

数量　　：2

（8）サウンド仕様（Sound Specification）

用途　　：状態情報の変動、変化を音によって表現する為に用いる

選定論拠：44.1KHz以下のサンプリング・レートをサポートする事

型式　　：Sound Card

仕様　　：Sound Card

数量　　：2

（9）基本ソフトウェア仕様（System Software Specification）

型式　　：Windows 95 Japanese Version

仕様　　：Windows 95 Japanese Version
数量　　：2
用途　　：ＯＳとして使用する
選定論拠：UNIXSystemとの整合性がある事

型式　　：Microsoft Visual C++
仕様　　：Microsoft Visual C++ Japanese Version
数量　　：1
用途　　：C++言語プログラムの開発、保守に用いる
選定論拠：状態情報アプリケーションを開発できる事

５．１０　オーディオ・ビデオ用システム・ハードウェア

オーディオ・ビデオ用システムは、ネットワークにて接続統合された各個別システムの出力するオーディオ・ビデオ信号を一元化する装置として構成した。これにより、個別のコンピュータシステムより出されるモニター信号、ビープ信号等を一元化して表示管理することができた。オーディオ・ビデオ用システムは、シミュレーション用システム、操作用システム、データベース用システム等のモニター信号を表示すると共にナレーション、効果音等を再生することを可能とした。

（1）大型カラー・ディスプレイ・モニター

図５．１０－１　モニターの外観

用途　　：シミュレーション操作、結果を表示する為に用いた
選定論拠：37インチ以上のブラウン管で1280 x 1024の解像度をサポートする事
型式　　：37 inch Multiscan Monitor
仕様　　：MITSUBISHI 37 inch Multiscan Monitor (XC-3730C)
数量　　：2

（2）音響再生装置

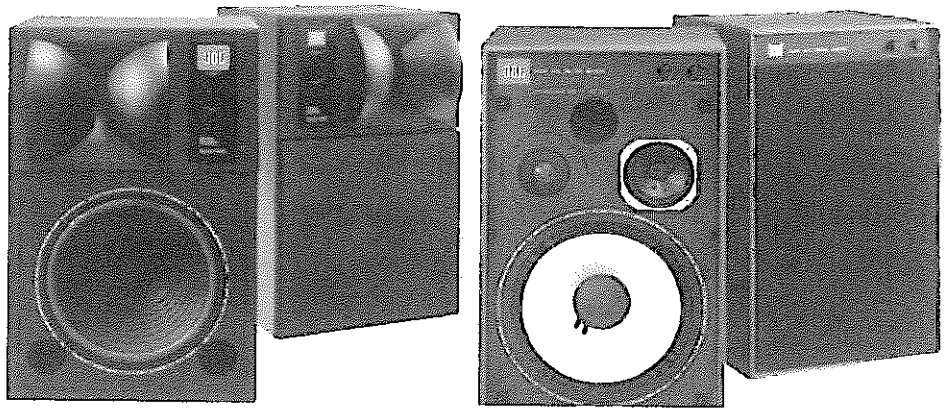

図５．１０－２　装置の外観

用途　　：シミュレーション状態を音にて表現する為に用いる
選定論拠：パラメータシステムSoundCardからの信号を音として表現できる事
型式　　：Speaker Set
仕様　　：Speaker Set
数量　　：１

（3）音響・ビデオ映像制御装置

図５．１０－３　装置の外観

型式　　：LD-player
仕様　　：LD-player
数量　　：２
用途　　：パラメータシステム用の音源等として用いる
選定論拠：映像、音響データを再生できる事

型式　　：9 inch Monitor
仕様　　：SONY 9 inch Color Monitor
数量　　：2
用途　　：ビデオ映像制御装置からの信号を確認する為に用いる
選定論拠：NTSC信号をサポートする事

（4）その他（The other requirement)

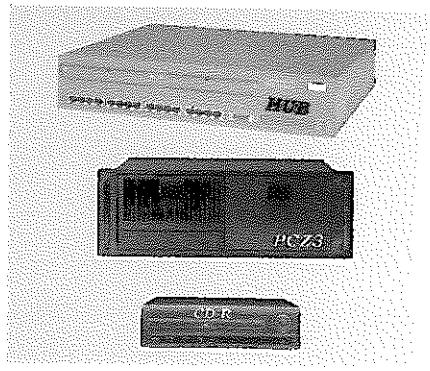

図５．１０－４　装置の外観

型式　　：Swiching Hab
仕様　　：Swiching Hab
数量　　：1
用途　　：各システム間とのデータ交換をスムーズに行う為に用いる
選定論拠：スイッチドネットワークをサポートしている事

型式　　：19 inch Rack form GTA PCZ3
仕様　　：19 inch Rack
数量　　：2
用途　　：パラメータシステムを19インチラック内へ収納する為に用いる
選定論拠：EIA規格19インチラックとする事

型式　　：CD-ROM Writer
仕様　　：PIONEER (DW-S114X-J + CDM-V74)
数量　　：1
用途　　：成果物をCD-ROMにて配付する為に用いる
選定論拠：ISO9660をサポートし書き込み、読み込みができる事

５．１１　システムの全景

図５．１１－１　システム全景１

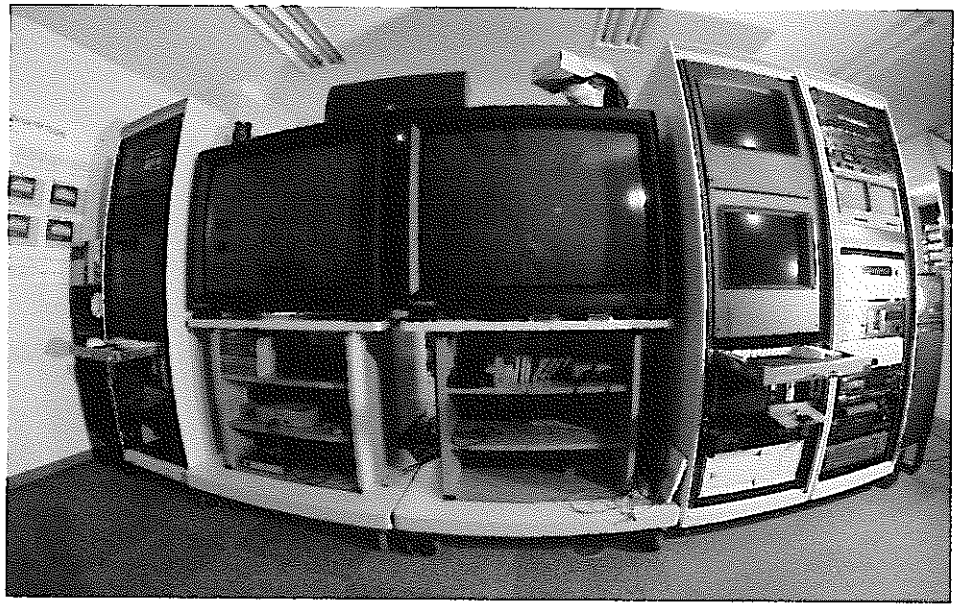

図５．１１－２　システム全景２

図５．１１－３

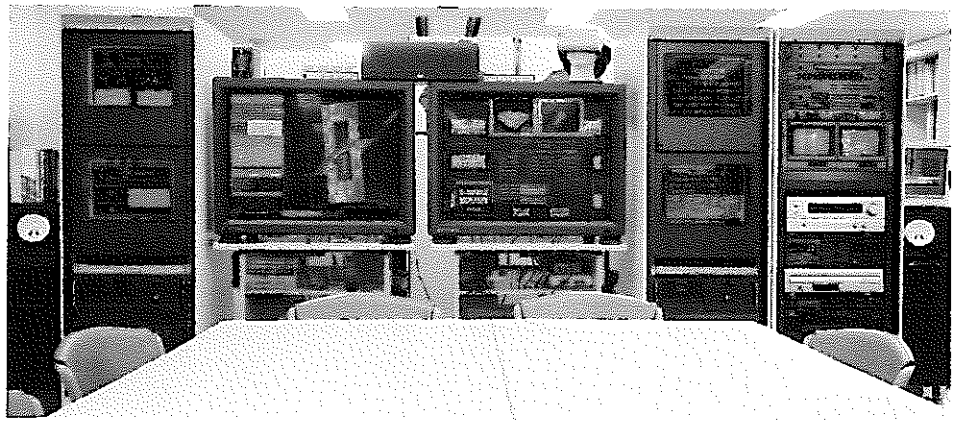

図５．１１－３　システム全景３

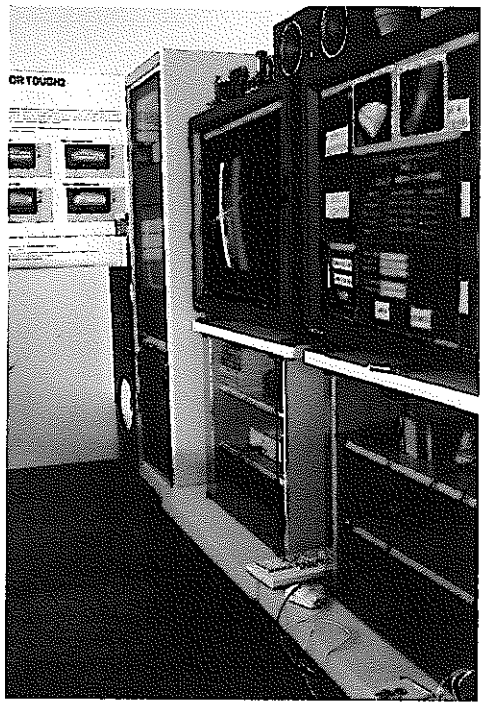

図５．１１－４　システム全景４