解　説

2. CASE 環境の基礎技術†

松　本　吉　弘††

1. まえがき

契約に基づいて生成，運用，維持されるソフトウェアのライフサイクル中に，このソフトウェアに関して形成される一連の技術活動をソフトウェア・エンジニアリングとよぶ．CASE 環境 (computer-aided software engineering environments) とは，このような技術活動を計算機によって支援するシステムのことである．単にプログラミングとその周辺だけを支援するのではなく，プログラミング支援を含むさらに大きな範囲での支援を行う．

たとえば，ある国際的な会議[1]で，CASE 環境に対するベンチマーク問題が議論された．表-1 はその一部を示している．第1は工程管理，第2はプログラムモジュールの再利用，第3はデータや手続きの及ぼす副作用，第4はモジュールの設計，第5はデータベース設計，にそれぞれ関係した問題である．これで分かるように，CASE 環境には，広い範囲にわたる機能が期待されている．

一般に，実生産に役立っている CASE 環境には，対象を絞った特定 CASE 環境（たとえば，オフィスオートメーション用，工業プロセス制御用など）や，複数の異目的特定 CASE 環境（たとえば，プログラミング支援環境，工程管理支援環境，原価管理支援環境など）の連合体 (federation)（たとえば，プログラミング支援環境/工程管理支援環境/原価管理支援環境を連合させたものなど）が多い．特定 CASE 環境とは，つぎのような性質が特定されている環境のことである．

1) 対象とするソフトウェアの性質
2) 対象とするソフトウェア・エンジニアリングの範囲
3) 開発パラダイム，設計方法論
4) 目的ソフトウェアのプログラミング言語
5) 解放性，メタ性の程度

汎用の CASE 環境も種々試みられているが，真に生産に組み入れられるところには至っていない．問題は，「汎用」の捉え方である．特定の方法論や特定応用向けパラダイムを自由にカスタマイズできるような志向がとられるならば，汎用 CASE 環境の前途は明るいといえよう．

CASE 環境を一般的に論じることは，それが上に述べた各種の性質，CASE 環境を利用する個人や組織の人間的要素，組織風土などにきわめて強く依存するため，難しいとされている．しかし，見方を変えて，CASE 環境をそれを構成する要素技術という視点から観察すると，共通性がかなりあることが分かっている．すなわち，どの CASE 環境も，ほぼ共通の要素技術を使って構築できると考えられている．汎用が難しいという点については，特定化（カスタマイゼーション）を行う機能を用いることによって解決できる．

以上のような考えに従い，本解説では，CASE 環

表-1 CASE 環境に対するベンチマーク問題の例

Query 1:	5日以上の遅れを生じ，それがクリティカルパスとなっているタスクとその担当，管理責任者をすべてあげよ．
Query 2:	ホスト計算機と，ターゲット計算機を指定して，それぞれに対して装備した経験のあるモジュールを，あるモジュール集合のなかから選択し，列挙せよ．
Query 3:	再帰テストを実行する過程において，その各実行結果で行われる変更に際し，副作用を逆に辿り，変更すべき全項目をその都度報告せよ．
Query 4:	起こり得る例外を総覧し，そのなかで，例外処理を用意していないものを列挙せよ．
Query 5:	データベースのなかで，あるデータの機密レベルを指定変更した場合，データベース全体の機密度が受ける影響を示せ．

† Basic Engineering Technique for Implementing CASE Environments by Yoshihiro MATSUMOTO (Department of Information Science, Kyoto University).

†† 京都大学工学部情報工学教室

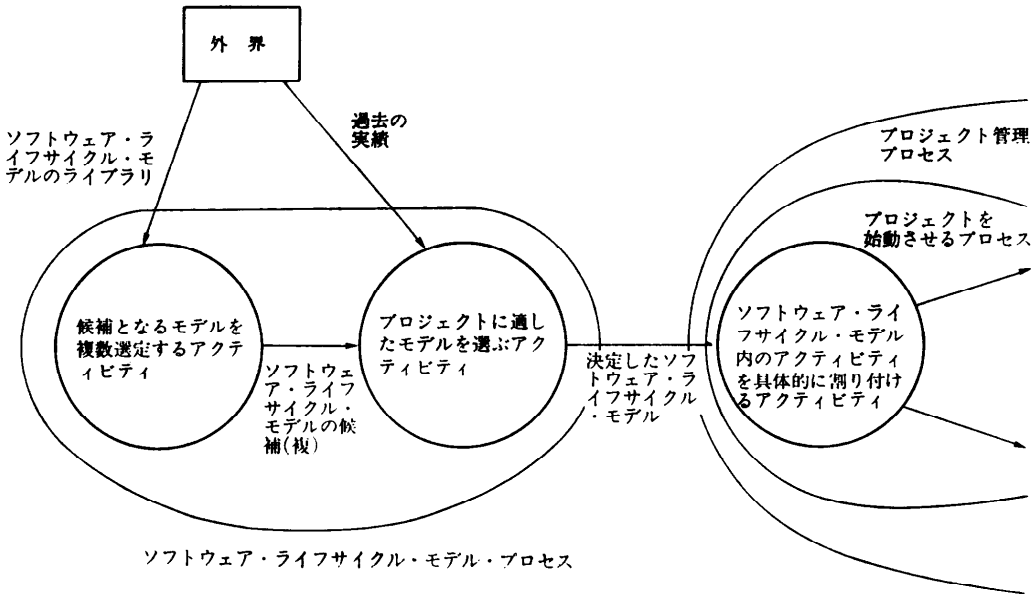

図-1 IEEE-P 1074/D 4 におけるプロセス，アクティビティと関係の一部

境に共通する要素に焦点を当て，基礎的な技術について述べる．

2. ソフトウェア・エンジニアリング・モデル

ソフトウェア，エンジニアリングに携わる技術者が共通して参照できるようなリファレンス・モデルを作ろうとする努力は，プロセス・プログラミング・ワークショップ[2]，IEEE-P 1074[3]，ISO/SC 7/WG 3/SG 3，同 WG 5 などで進められている．IEEE のモデルとはつぎのようなものである．

ソフトウェア・エンジニアリングを分割・詳細化していくと，プロセスという単位が識別され，さらにひとつのプロセスはアクティビティという単位で構成されることが分かる．プロセスの入力，出力は実体である．ここでいう実体（4. でさらに述べる）とは，可視化された“もの”のことであり，ソフトウェア製品，ソフトウェア中間成果物，ソフトウェア構成要素，指示書，報告，計画書，標準/規定などを意味している．**図-1** に IEEE モデルの例を示す．IEEE では，各プロセスごとに，そのプロセスはどのようなものか，それに対する入力および出力がどのような実体を指すかを定義する．標準化するのは，個々のアクティビティとその入出力実体，プロセス/アクティビティ階層だけである．ソフトウェア・エンジニアリング・モデルそのものは標準化しようとせず，個々のプロジェクトに対してモデルをカスタマイズ（プロジェクトに合わせて特定化すること）する方法だけを示す．

実際にプロジェクトを計画するときは，プロセスすべてを総覧したプロセス一覧表（各プロセスごとにアクティビティ，入力実体，出力実体が定義されている表）を参照し，そのなかから，自分のプロジェクトに必要なプロセスだけを選択させる．その入出力を互いに結合し，自分のプロジェクト用にカスタマイズされたプロセスモデル（網状のモデル）を作成させる．このようにして作られたプロセスモデルが，そのプロジェクトのソフトウェア・エンジニアリング・リファレンス・モデルとなる．

上に述べたプロセス（アクティビティを含む），実体，リファレンス・モデルのカスタマイゼーションという三つの概念は，以下，CASE 環境を論じる上で，基盤となる思想である．

3. CASE 環境の構成

CASE 環境の構成のあり方は，2. で述べたソフトウェア・エンジニアリング・モデルと密接に関連する．CASE 環境の構成は，ソフトウェア・エンジニアリング・モデルと素直に対応していることが望ましい．2. で述べたプロセス，実体，モデルカスタマイ

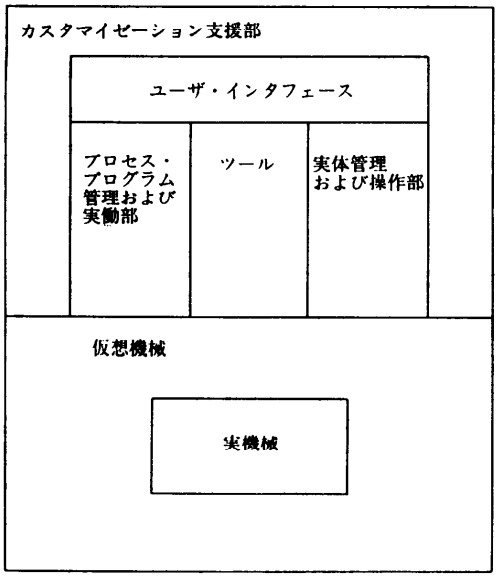

図-2 CASE 環境の構成

ゼーションの概念から，図-2 に示す構成（このような構成は，文献 7)～9) でも述べられている）と，下記の要素が導かれる．

a） プロセス・プログラム管理および実働部

プロセス/アクティビティを記述したプロセス・プログラム (7. で述べる) の管理および実働 (enaction) を行う部分である．この部分は，一般に，記述支援部，記述されたプログラム，またはインスタンスの管理部，実行/保存管理部の 3 階層から構成される．

b） 実体管理および操作部

実体を管理し，その内容を操作する部分である．この部分も，一般に上と同様の 3 階層から構成される．

c） カスタマイゼーション支援部

CASE 環境を構成する各要素を，特定のプロジェクトに向くように，記述，編集，組合せ，などによって特定化する作業を支援する．

環境として機能するためには，さらにつぎのような部分を加えることが必要である．

d） ユーザ・インタフェース

プロセス・プログラム管理および実働部，実体管理部，カスタマイゼーション支援部を，人が操作するための便宜を与える．また，8. で述べるグループ・コミュニケーションのための便宜も与える．

e） ツ ー ル

ツールは，ユーザ・インタフェース，またはプロセス・プログラムによって制御される．

f） 仮 想 機 械

実機械は，ハードウェア，固有のオペレーティングシステム (native operating system)，ユーティリティなどから構成される．実機械と上に述べた a）から e）までの部分とは直接には結び付かない．そこで，仲介部が必要になる．たとえば，実体を実機械のデータベース管理システムへ展開するための変換部，プロセス・プログラムの実働のための解釈部，ユーザ・インタフェースから入力される操作の解釈部，共通のルーチンやライブラリなどである．実機械にこれらの仲介部を重ねたものを仮想機械とよぶ．

ソフトウェア・エンジニアリング・モデルに直接対応する部分が，a）および b）であげたプロセス・プログラムおよび実体に関する部分である．この部分を CASE 環境で実現する際に，手続き指向か，またはオブジェクト指向が選択される．ソフトウェア・エンジニアリング・モデルをデータフロー・モデルとして捉える人々は，手続き型を志向することが多い．実用されている多くの環境はこの部類に属する．

これに対して，実体を中心に置いて考える人々は，オブジェクト指向を志向する傾向にある．すなわち，実体およびそれに対する操作をひとつのモジュールにカプセル化し，オブジェクトとする．この場合には，プロセスを記述したプロセス・プログラムも実体として扱われる．オブジェクト指向を選んだ場合には，上に述べた a）および b）の部分は，オブジェクト指向環境によって実現される．オブジェクト・モデルの特徴は，実世界との対応の自然さ，階層性/属性継承を用いることによる体系化/再利用の容易性，動的束縛を用いることによる拡縮の容易さなどにある．以下の説明では，オブジェクト指向を中心に扱う．

4. 実体の管理と操作

ここでいう実体とは，プロジェクトのなかで作られ，最終のソフトウェア製品の生成に貢献するすべての記述，表現を意味している．実体には，(1)仕様書のような自然言語で書かれた記述，(2)データフロー図 (DFD と略称する)，状態推移図，コントロールフロー図 (CFD と略称する)，ペトリネット，エンティティリレーション図 (E-R と略称する)，逐次制御論理図のような図，(3)デシジョンテーブル (DT と総称する) のような表，(4)構文木のような木構造，(5)その時々の意志伝達などに向くように作られる特

定の様式の記述，などがある．

CASE 環境がこれら実体の内部に立ち至って細かく管理，操作するためには，その実体に含まれる単語や単位記号のひとつひとつをデータとし，それらの間の関係を構造に写像したデータ集合として実体を表す必要がある．実体に対応して作られたこのデータ集合がモデルとよばれるものである．

実体を印刷，または表示する際には，モデルをいったん，ユーザの要求する表示様式，印刷様式に合う構造や属性をもったデータ集合へ変換する必要がある．変換されたこのデータ集合のことをビューと称する．

同じ実体でも，いくつかの異なる様式で見たいという希望がある．たとえば，プログラムはフローチャート様式で見たり，構造的に美形化されたテキスト様式で見たりしたい．このような場合に，実体のモデルには手を加えず，ビューを複数用意する．実体の表現様式を変えることをビューポイントを変えるという．ひとつのモデルによって表されている実体を複数のビューポイントで見たいときには，ビューポイントに対応するビューをもつ，モデルとビューを分離する考え[4),10)]は，ひとつの実体の内部に関する操作（生成，追加，変更など）と，異なる実体相互間に関する操作とを互いに独立させるために必須の概念である．この場合，モデルとビューの間には常に一貫性が保たれるような管理が行われることが前提となる．

オブジェクト指向型実体管理を行っている CASE 環境においては（たとえば，文献 6) を参照），図-3 に示すように，モデルとビューを同じオブジェクトのなかに入れるように構成し，相互の一貫性を管理する．図では，MODEL というクラス（オブジェクト），および VIEW というクラス（オブジェクト）を，インスタンス変数としてもつような ARTIFACT というクラス（複合オブジェクト）が示されている．この場合，MODEL のもつインスタンス変数（データ集合）がモデルであり，VIEW のもつインスタンス変数（データ集合）がビューである．このクラス ARTIFACT は，モデルとビューの一貫性を常に維持するようなメソドをもっている．ARTIFACT は，それぞれの実体に関してインスタンスを生成する．すなわち，ARTIFACT のインスタンス名がひとつの実体名，たとえば，要求仕様書 “REQSP-X 1” というよう名を表す．

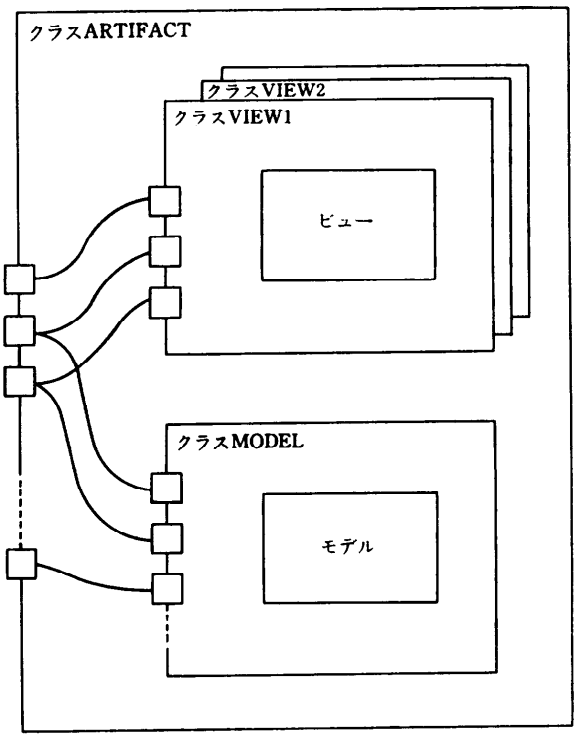

図-3 オブジェクトの基本的構成

ここで，ユーザ・インタフェース（8. で説明）を通して実体を作っていく過程を概説しよう．“REQSP-X 1” をなんらかの仕様記述言語に従って表示面上に記述していくとする．ユーザは，コマンドによって，キーインされた記述を，この記述言語を対象にした直構文型エディタ（syntax-directed editor）に渡す．構文木，属性文法の意味関数に基づいた計算が行われ，記述フォールトが正された後，記述に対応した構文，および意味を表すデータ構造が形成される．このデータ構造が，モデルであり，インスタンス “REQSP-X 1” がもつ MODEL のインスタンス変数となる．このインスタンスの各メソドは，このデータ構造を新たに作ったり，修正したり，削除したりする作用を果たす．今，ユーザ・インタフェースからユーザ・コマンドが入力されると，まず ARTIFACT インスタンス “REQSP-X 1” に作用し，つぎに MODEL インスタンス “REQSP-X 1” のメソドに作用してモデルを生成，変更，削除する．一方，ビューについては，つぎのとおりである．上で入力した仕様記述言語による “REQSP-X 1” の記述が，DFD，および CFD へ変換

できると考えられているとする．そうすると，DFDのためのビューと，CFDのためのビューが必要となる．すなわち，ARTIFACTインスタンス“REQSP-X 1”のなかには，ふたつのVIEWオブジェクトが作られることになる．ユーザ・インタフェースの表示面には，ユーザの選択するいずれかのビューが表示される．MODELインスタンスのなかに作られた前記データ集合（モデル）と，これらVIEWオブジェクトのなかに作られるデータ集合（ビュー）はまったく独立である．しかし，これら3者のもつ値や構造に関する一貫性は，ARTIFACTインスタンス“REQSP-X 1”のもつメソドによって常に維持される．したがって，仕様記述言語による新しい入力は，速やかに現在表示されているビューへ反映されることになる．また，たとえばDFD，CFDを重ねて見たいという希望がある場合には，ふたつのビューを指示された重ね方に従って重複表示するツールを通して表示する．

5. 実体と実体との間の関係

5.1 関係とは

モデルとビューを分離し，それぞれを独立にする理由は，すでに述べたように，ひとつの実体内でその実体に関する複数のビュー間の一貫性維持のための操作と，異なる実体相互間の一貫性を維持するための操作を分離したいということにあった．前者については，4. で述べた．ここでは，モデルに基づいて行われる異なる実体相互間の一貫性維持操作について述べる．ソフトウェア開発パラダイム，設計方法論などの議論は，すべてこの関係定義の問題に帰着される．

ソフトウェア・エンジニアリング・モデルの上流にあるある実体集合 A が，それより下流にある実体集合 B に変換されるときに，A，B それぞれに含まれるデータ相互の間に関係（垂直方向の関係という）が生まれる．また，同じ分岐点・合流点をもつ並行した流れで作られる実体集合それぞれに含まれるデータ間には，水平方向の関係が生まれる．

実体間の関係は，A 内の各データに対してなんらかの変換規則をある思考過程に沿って繰り返し加えた結果，最終的に得られたものであるが，この関係は式（等式，不等式，論理式など），述語形式，図式，集合論における写像と関係など，応用数学的方法ではなかなか表し切れない．プログラム等価変換論で提唱されたような変換規則の繰り返し適用としても表すことが難しい．この変換は，技術的の思考過程に従って行われたものであることから，思考プロセスを記述するプログラムと，そのなかに組み込まれた推論機構との実働によって，表す方向が志向されている．技術者の思考過程が，模範過程や方法論に裏付けられた手続きと，その手続き過程上の特定の箇所で行われる判断や意思決定のための推論規則および，推論機構によって導かれるようにする．模範過程は，各職場で，熟練した技術者が選ぶ思考過程を追跡，計量することによって得られるものであり，推論規則，推論機構は，その職場で特定された変換規則集合（普通は知識ベースとして実現される）および推論方法を計量することによって得られる．

5.2 関係の定義

関係を定義する目的は，次の三つに分類される．

a） ソフトウェアを新しく作っていく場合には，品質の作り込みが目的になる．たとえば，下流で作られる実体が上流で作られる実体を意味的に満たしている度合（正当性），また，垂直/水平方向での脱落/冗長のない度合（完全性），矛盾のない度合（一貫性）などを検証できるようにする．この観点では，思考プロセスを品質作り込みプロセスと捉える．

b） いったん作られたソフトウェアの維持という立場からは，生産に関係しなかった第三者による改変，進化を容易にすることを目的とする．この場合には，なぜそうしたか，というような意思決定プロセスを重視する．

c） 再利用という立場からは，熟練者の知識，思考過程を可視化し，伝承，初心者の教育，改善を容易にする．

a）でいう品質作り込みを目的とする関係定義においては，思考プロセスが実行した操作（変換など）の履歴を関係とし，関係の記述を実働させ，下流の実体記述に適用することによって，上質の実体記述の正しいことを証明できるようにする．これに対して，保守，再利用，伝承，教育を目的とする関係の記述は，それをなんらかの方法で正確に，分かりやすく表示，あるいは演出できるようにする．

具体的には，1組の関係記述に対してひとつの関係インスタンスを割り当てる．この関係インスタンスのインスタンス変数には，たとえば，その関係を形成する過程で行われた思考プロセスを記述したプロセス・プログラム，このプログラムを実行することによって実施された変換の系列（この系列を用いてプロセス・プログラムを追試できる），推論に使われた知

識データベース，使われた変換規則系列（この系列を用いて推論の追試ができる），などが格納される．この関係インスタンスのもつメソドは，インスタンス変数の管理手段，アクセス手段，利用手段（目的に合わせて，インスタンス変数を実働させたり，演出させたりする）を提供する．

5.3 関係オブジェクトの配置

つぎに，このようにして作られた関係オブジェクトをどこへ配置すればよいかの問題がある．これは，その関係に関わる実体がもつ階層的性質と深く関連する．一般に，オブジェクト指向が提供する is-a（汎化），is-part-of（集約）の機構のなかで，関係オブジェクトを配置する場合について考える．

要求仕様書と設計仕様書を例にとる．両仕様書は，それぞれ独自の属性をもつとともに，互いに強い関係をもっている（前述の垂直関係）．一方，各仕様書はいくつかの副仕様書を組み合わせて構成され，各副仕様書はそれぞれ独自の属性をもつとともに，共通の属性をもち，さらに互いに関係をもっている（前述の水平関係）．版（version）管理のために必要な版間関係もこの関係に属する．

垂直関係では，要求仕様書/設計仕様書の上に仕様書という is-part-of スーパクラスを置き，関係オブジェクトをこのスーパクラスのインスタンス変数とする．要求仕様書（設計仕様書でも同じ）はいくつかの副仕様書から構成される．副仕様書間には水平関係がある．このような場合，要求仕様書を is-a クラス，副仕様書をそのサブクラスとし，副仕様書間関係オブジェクトは，要求仕様書クラスのインスタンス変数とする．

垂直関係についてはつぎのような問題がある．一本のライフサイクルに沿ってのみ連接して実体が作られる場合，すなわち一つながりの垂直関係だけが存在する場合には，上に述べた配置で問題ないが，ソフトウェア・エンジニアリング・モデルは，複雑な網構造をもち，それぞれの矢に沿って関係が各所に形成される．したがって，上記の配置は困難である．

そこで，垂直関係を完全に独立したひとつのクラス体系とすることが考えられる．この場合，問題となるのは，このインスタンス AB へのアクセス手段をどのように提供し，管理するかである．たとえば，つぎのような方法がある．

まず，A および B を表す実体インスタンスのインスタンス変数として，それぞれ同じように関係インスタンス AB を宣言しておく，A および B には，AB の存在位置（ポインタ）を，あらかじめ知らせておき，A, B が関係インスタンス AB を参照したいときには，AB 全体を A, B がそれぞれ，自分のインスタンス変数へコピーして利用する．関係オブジェクト AB には，A, B が共用するデータやプログラムも入れておくことができる．このようにしたときに問題となるのが，A, B から並行に行われる AB に対する書き換え，読み取りをともなうトランザクションの制御である．CASE 環境の場合，この制御は，グループ・コミュニケーション（8. で述べる）によって実現することができる．

6. 実体データの保存

CASE 環境においては，前日作った実体は，翌日続いて改変，進化されるのであるから，前日のトランザクションが翌日へ継続されなければならない．すなわち，ユーザ・トランザクションは，ユーザのログイン/ログオフ，装置の電源入/切などに関係なく，プロジェクトの存続期間中にわたって，切れ目なく存続する．プロジェクトの進行とともに変形する実体に関する最新のモデル，およびプログラムまたはオブジェクトの中間状態は，プロジェクトの存続期間にわたって，必要な時点ごとに保存できるようにしなければならない．

普通のプログラミング言語で記述されたプログラムでは，手続き指向であれ，オブジェクト指向であれ，プログラムが参照する変数は，それが意識的に永久記憶装置へ保存されないかぎり，プログラムの実行終了とともに消滅する．CASE 環境では，意識的に永久記憶装置へのデータ掃き出しをいちいちしなくとも，プログラム実行終了時に必要なデータが，必要期間にわたって保存されるような仕組みを考える必要がある．

データベースプログラミング言語やパーシステントオブジェクト指向言語が実用に供されるようになってきた[5]．これら言語は，CASE 環境の構築にも役立つと考えられる．パーシステント（persistent）という語は，永続性と訳されることがあるが，CASE 環境の場合は，その保存期間を指定できる必要があり[6]，永続ではないので，以下では保存性という語を用いる．保存性をどのように抽象化し，どのように定義させるかは，まだ確立されていない．保存性を記述できる言語はつぎのような性質をもつ．

イ） プログラム（またはオブジェクト）のなかで，

上位クラスとの関係において，プログラムの実行終了時に自動的に保存したいデータに保存性（persistency）宣言を追記しておくことができる．

ロ） 保存性宣言の際に，保存したいデータに保存名（persistent name）をつける．保存名は，そのデータの存在するプログラム（またはオブジェクト）の名とは異なる視点（直交する視点が望ましい）からつける．

ハ） 保存性を与えられたデータは，同じトランザクションのなかで再びプログラムが実行されるとき，自動的にそのプログラム領域内に呼び込まれる．

ニ） 保存性をもったデータは，その保存名によって，前記のトランザクションとは関係なく，検索し，利用できなければならない．ただし，書き換えは前記トランザクションとの関係で定まる特定の方法でしか許されない．

保存性をもったデータを実際に保存する入れ物として，既成のデータベース管理システム，データディクショナリをそのまま利用することが多い．このような場合には，データ保存時に，保存名の形式や型についての整合チェックを行う必要がある．

7. プロセス・プログラムの管理と実働

ソフトウェア・エンジニアリング・モデルで示されたプロセス/アクティビティ/手順要素を表すプログラムのことをプロセス・プログラムとよぶ．

プロセス・プログラムには，文献 2) の国際会議などの議論から，階層的分類を行う必要があることが知られている．図-4 に，その階層の1例を示す．一番低層には実体への操作を目的とするプロセス・プログラムがある．ひとつの実体だけの操作を目的とした手順要素に関するプロセス・プログラムは，実体オブジェクト内のメソドとすることができる．その上には実体を加工するためのツールを操作するプロセス・プログラムが存在し，さらにその上にはプロセスに共通したプログラムのライブラリが存在する．その上には，ひとつのプロセス内を記述した応用プログラムがあり，さらにその上には異なるプロセス間を接続する応用プロセス・プログラムがくる．

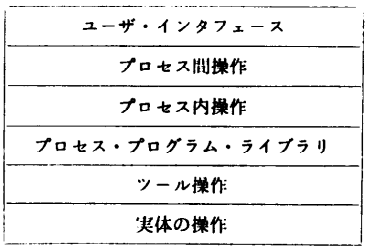

図-4 プロセス・プログラムの階層

プロセス・プログラム自身をデータと考えることもできる．したがって，プロセス・プログラムを抽象化したオブジェクトを作り，それをインスタンス化したり，継承したりすることができる．プロセスとプロセスとの間に跨るプロセスを記述するプロセス・プログラムは，プロジェクト全体の工程計画，要求定義/設計/試験/保守の方法論，グループコミュニケーションのプロトコル，などを記述する．

一般に，プロセス・プログラムを記述する言語にはつぎのような性質が必要であるとされている．

＊強力な型づけ（typing）機構をもつこと．実体オブジェクト型，プロセスオブジェクト型を管理，検査し，一貫性維持に貢献するためである．

＊並行処理，例外処理，複合ループ，後戻りを前提とした試行，リフレクション（メタレベルで実行を監視，制御すること）などに対する制御機能をもつこと．

8. ユーザ・インタフェース

すぐれたユーザ・インタフェースに期待される基本的な性質として，一様性（uniformity），直接操作性（direct manipulation），寛容性（permissiveness）があげられている．一様性とは，細かい規則をいちいち覚えたり，操作指示書を参照しなくても，既成の概念からの類推で新しい機能が自然に使えるようになっている性質である．直接操作とは，操作する対象を表示し，それにユーザが直接操作を加えることができ，その操作結果を併せて表示することによってユーザが操作の結果を遅滞なく確認できるようにすることである．寛容性には，種々の性質が含まれる．不正なキーイン入力の自動修正もそのひとつである．最も注目されている性質は，ひとつの操作を行っているプロセスを中断し，別のプロセスを生成し，その操作へ移ることができるとともに，元のプロセスを含めて複数の操作を並行して持続させ，自由にどのプロセスにも平等に入ることができ，かつ存在するプロセスを任意の順序で終了させることができる性質である．このような性質は，ユーザ・インタフェース・プロセスと連動して実働するプロセス・プログラムについても必要とされる．

ユーザ・インタフェースは，特定のユーザの視覚お

よび手操作，ツール，実体管理/操作部，およびプロセス・プログラム管理/実働部などと交信するが，プロジェクトが複数の CASE 環境に跨る場合，他の CASE 環境とも交信する必要がある．この交信は，グループ・コミュニケーションの一方式として捉えられる．近接位置に在席する個人の小集団内の話し合い，遠隔位置に存在する複数の小集団間の話し合いを基に，大集団が抱える問題の解決を行う方式をグループ・コミュニケーションとよぶ．CASE 環境の場合は，地域的に分散して進められる分散型ソフトウェア・プロジェクトを想定する必要がある．

グループ・コミュニケーションのための理論的基礎は，心理学，社会科学，教育学，言語学，認知科学，などに求められる．十分な基礎的な検討を経て作られるコミュニケーション・モデルの策定が必要で，このモデルに基づいて，ユーザ・インタフェースはコミュニケーション支援を行う．一般的には，CCITT/X.400/1985, ISO/DIS/10021 で規格化されている MHS (message handling system) の上に，支援環境を構築するのが適当とされている．

たとえば，CASE 支援環境では，つぎのようなコミュニケーションを支援する．メンバ A が自分の担当している仕様書を作成する過程で，メンバ B が担当する仕様書の一部を変更して欲しくなり，この変更がメンバ C および D に影響を及ぼす場合を想定する(影響の及ぶ範囲は，前記の関係オブジェクトへのアクセスによってユーザ・インタフェースが自動的に検出する)，A は変更依頼を B, C, D へ発送し，承認をもとめる．関係メンバは，必要な場合，プロジェクト・マネージャも加えて電子会議の場を設定し，解決を図る．すべてのメンバが，担当する文書間の一貫性を保つべき修正を行った上で，全員の文書ファイルおよびそれを表すモデル (4. で述べたデータ構造) を書き換える．コミュニケーション・プロセスの進行においては，プロセス上の必要な状態値系列の保存が行われ，事後に行われる工程分析や品質分析の際に利用される．

9. むすび

CASE 環境に関する基礎技術を，図-2 に示した構成と，要素に従って説明した．ソフトウェア・エンジニアリング・モデルがプロジェクトに合わせてカスタマイズされるように，CASE 環境もカスタマイズが可能であることが望ましい．そのためには，各要素は，それぞれカスタマイズのための機能をもち，カスタマイゼーション支援部にその機能を提供する必要がある．ユーザ・インタフェースが自らのインタフェースを記述するための言語をユーザに解放する機能，実体オブジェクト，関係オブジェクト，プロセス・プログラム，コミュニケーションモデルを記述する言語をそれぞれの部分がユーザに解放する機能などがこれに当たる．

管理志向重点であった我が国のソフトウェア生産技術が，今後，新しい時代に向けて脱皮を遂げるためには，CASE 環境への傾注がとくに重要と考える．外国に劣らない成果を期待したい．

参考文献

1) Rowe, L. A.: Report on the 1989 SOFTWARE CAD DATABASE WORKSHOP, Information Processing 89, Ritter, G. X. (ed.), Elsevier, pp. 719-725 (1989).
2) Perry, D. E. (ed.): Proceedings of 5th International Software Process Workshop, IEEE-TCSE/ACM SigSoft (1989).
3) IEEE-TCSE (sponsered): Standard for Software Life Cycle Processes, IEEE-P 1074/D 4, August 8 (1989).
4) Meyers, B. A.: Incense, A System for Displaying Data Structures, Comput. Graphics, Vol. 17, pp. 115-125 (July 1983).
5) Zdonik, S. B. et al. (ed.): Reading in Object-Oriented Datatbase Systems, Morgan Kaufman (1990).
6) Taylor, R. N. et al.: Foundation for the Arcadia Environment Architecture, ACM 0-89791-290-X/88/0011/0001 (1988).
7) Oberndorf, P. A.: The Common Ada Programming Support Environment (APSE) Interface Set (CAIS), IEEE Trans. on Software Engineering, Vol. 14, No. 6, pp. 742-748 (1988).
8) Thomas, I.: Software Environments: PCTE and Related Projects, Information Processing 89, Ritter, G. X. (ed.), Elsevier, pp. 875-878 (1989).
9) Penedo, M. H. and Riddle, W. H.: Guest Editor's Introduction Software Engineering Environment Architectures, IEEE Trans, on Software Engineering, Vol. 14, No. 6, pp. 689-695 (1988).
10) 暦本他：エディタを部品としたユーザインタフェース構築基盤：鼎，情報処理，Vol. 31, No. 5, pp. 602-611 (1990).

(平成2年2月28日受付)