TECHNOLOGY REVIEW

UNISYS 技報 通巻 38

1993年8月発刊 Vol. 13 No. 2

特集：開発支援環境

日本ユニシス

ISSN 0914-9996

TIPPLER* は，高性能ワークステーションをビジネス分野でも有効に活用できること，オブジェクト指向技術の適用によりビジネスアプリケーションの生産性・拡張性の飛躍的向上を図ること，GUI を言語モデルの中に取り込むことによって開発を容易にすることを目標に開発された，知的活動領域をカバーしながら，従来の業務系・情報系システムを包含するワークステーション上の統合開発支援ツールである．保科剛は**統合開発支援ツール TIPPLER** の中で，現在までの TIPPLER の設計と実装，今後の開発目標について述べている．

ダウンサイジングを始めとした分散システム環境指向の流れが定着しつつあり，企業の情報利用形態や情報システムの構築方法に変化が現れ始めた．このような分散環境では，情報の統合管理や各種ネットワーク管理等のためにリポジトリの役割が重要となる．原田公敬・城代優二の**リポジトリ・システムのあり方とその実現に向けて**は，リポジトリ・システムの概略，リポジトリ・システム実現のための手順について述べ，さらに具体的要件を仮定して，手順の実践を試みている．

EUC という情報技術に取り組むには，EUC の必要性と利益について正しく認識し，適用が効果的な情報システム分野を明確にする必要がある．松木規子は**既存システムをベースとする EUC 環境の実現**の中で，情報システムの対象領域を，トランザクション処理の分野とマネージメント・サポート・システム（MSS）の分野に分け，MSS を EUC 適用分野と位置付けている．さらに，MSS 分野の実現形態として EUC によるデータ利用環境モデルを提案するとともに，モデル実現のための手順や考え方を述べている．

統合 CASE LINC は，ユニシスが提供する情報システム構築環境の体系である ASDF の中で，迅速なソリューション構築を実現する LINC による統合 CASE 環境として位置づけられている．佐伯克己は**統合 CASE LINC と LINC オブジェクト・モデリング**の中で，構成要素である LINC システムズ・アプローチ，その中核技法である LINC オブジェクト・モデリング，ワークステーション上の GUI 環境の中で支援する CASE ツール LINC-DA の解説を行っている．

モダナイゼーションとは「既存アプリケーションを大幅に変更することなく活性化し，時代の要請にあったシステムに近代化する」ことであり，具体的には，GUI 化が第一歩である．DW（デザイナ・ワークベンチ）は，4 GL の MAPPER, LINC で構築したホスト・アプリケーションのモダナイゼーションを支援するワークステーション上のソフトウェアである．河合昭男の**DW とモダナイゼーション**は，前半で DW の 4 GL 対応の機能について，後半で開発中の 3 GL 対応の機能について述べている．

IDES は COBOL 言語によるシステム開発を支援する下流 CASE ツールとして開発された．IDES によるプログラム開発では連接，反復，条件の制御構造を木構造図に作図し，処理部をその図の中の木の葉に記述して仕様書を作成する．リバース機能はこの逆の手順を辿る．儀間哲雄は**IDES のリバース機能**の中で，IDES のフォワード・エンジニアリングのプロセスと現在開発中のリバース・エンジニアリングのプロセスを記述している．

UNIX** をベースとした環境で稼働する業務アプリケーション・システムの開発が増加している．福地修一・吉野良成は**UNIX 環境における業務アプリケーション構築**の中で，筆者等のこれまでの開発経験をもとに，業務アプリケーションと UNIX のプロセスとの関係や，業務アプリケーションで提供すべきいくつかの機能についての技術的自由度を考察し，本格的にアプリケーションを作り込む場合の筆者等の開発スタイルや工夫点，留意事項等を述べている．

---

* TIPPLER は(株)野村総合研究所と日本ユニシスにより共同開発されたソフトウェアである．

** UNIX オペレーティングシステムは，UNIX System laboratories, Inc.が開発し，ライセンスしている．

特集「開発支援環境」の発刊によせて

本池　洵

この数年，多くの分野で“変化”が言われている．もともと何がどう変わったかと言う文脈で評価すべきことだと思うが，現在は，変化自身がさまざまな事象に共通の価値として認識されているようである．たしかに，長い間あることをしていて，その間見るべき進歩がなく，マンネリズムに陥っているとしたら展望を拓くため何かを変えて見る必要がある．これは日頃よく言われることでもある．

さてシステム開発分野はどうであろうか．かなり以前からハードウェア技術の進歩に対し，ソフトウェア技術の進歩が遅いと言われてきた．ここで言われる技術進歩は前者がCPU速度とメモリ容量であり，後者が開発生産性である．この言はソフトウェア開発の生産性の低さに対する苛立ちを強調し，象徴的に言う時の常套句であり，ここから何かが生まれるものでもないであろう．この対比はともかくとし，現在さまざまな困難があるにしろ商用でも500万行に及ぶ巨大なソフトウェアが開発されている．これは，1970年頃盛んに言われた“ソフトウェア危機”の視点からすれば想像しにくい．生産性が実感としてそれほど向上したと思えない状況で，これほど巨大なソフトウェアが開発できることは規模により生産性がそれほど低下していないともいえる．これを可能としている一つは，ハードウェアの進歩に負っているであろう．CPU速度の向上とメモリ容量の拡大は，ソフトウェアが持つ重要な特性である効率面の制約を大幅に緩和し，平易なソフトウェア作りができるようになったことでもある．もう一つは，過去，何度かの開発経験を経ての開発プロジェクトの成熟にあると考えられる．

今でも，生産性要因のトップに開発に携わる個人の技術的資質があげられる．しかし，大きなプロジェクトの場合，全員が高いスキルを持つこと等，現実として困難であり，技術水準は平均化するが，プロジェクト全体としての学習効果が知識の再生産を図ってきているのであろう．

しかし，ソフトウェア開発の前線にいる者として，当初言われていた，生産性は規模により指数関数的に悪化するというほどではないにしろ，技術面，管理面の工夫・努力によってぎりぎりの線でリスクを回避し何とか切り抜けていると言うのが実感である．規模的にはもはや限界に近いのではあるまいか．このような状況打破のため，これまで開発方法，言語，支援ツール，開発環境，プロジェクト管理等開発に関わるすべてにわたりさまざまな工夫がされてきたし，今も続けられている．その中でもソフトウェアを部品化し再利用することができれば，生産性と品質は飛躍的に向上するであろうということから本命視され，これまで多くの提案と試みがなされてきた．しかし，現場の実務レベルで成功したと言う事例は皆無に近く，一時は幻に過ぎないのではないかとまで言われた．もちろん，連結編集対象としてのライブラリは古く

から存在するが，少し変形させて再利用しようということになるとソースコードレベルでのライブラリが必要になる．しかし，一般的な手続き型言語でこれを実現するには無理があった．たしかに，声高に再利用をうたわないが，アプリケーション開発では類似のシステムを参考にして新しいシステムを作ることが行われてもいるし，特定のソースコードが姿を変えながら広く使われているケースもある．汎用的にはむずかしいが，場面を限定すれば効果を上げることができるということであろう．だが，ここ2，3年オブジェクト指向技術の一般化に伴い，再び汎用レベルでの部品化・再利用が現実味を増してきている．オブジェクト指向技術自体スケールアップし，実行環境を整備するためには要素技術レベルで解決しなければならない課題がまだ多い．今後の発展に期待したい．

本号では「開発支援環境」をテーマに話題性の高いリポジトリ，オブジェクト指向，EUC，CASE，モデリング，リバースエンジニアリング，GUI，UNIX，エキスパートシステムを取り上げた．皆様の参考になれば幸いである．

（システム技術本部　本部長）

統合開発支援ツール TIPPLER

TIPPLER —— an Integrated Development Support Tool

保　科　　剛

要　約　コンピュータシステムの発展に伴い，従来の業務系システムや，情報を戦略的に利用する情報系システムSIS(戦略的情報システム)に加え，人間の知的活動領域にまでアプリケーションを展開することが求められている．TIPPLERは，このような知的活動領域をカバーしながら従来の業務系・情報系システムを包含するワークステーション上での統合開発支援ツールである．

TIPPLERの当初の開発目標は，すでにエンジニアリング分野では定着しつつあった高性能ワークステーションをビジネス分野でも有効に活用できるようにすること，オブジェクト指向技術を適用することによりビジネスアプリケーションの生産性・拡張性の飛躍的な向上を図ること，コンピュータに不慣れなエンドユーザにとって使い勝手のよいインタフェースではあっても，開発者には大きな負担を生じるために十分に活用するに至っていなかったグラフィカルユーザインタフェースを，単にライブラリとして提供するのではなく，言語モデルの中に取り込むことによって開発を容易にすることであった．本稿では，現在までのTIPPLERの設計と実装，今後の開発目標について述べる．

Abstract　Computer systems have been growing to the extent that it is demanded that applications be expanded to the areas of human-like intelligence, surpassing traditional business data processing systems and strategic information systems (SIS) created to process information strategically. Designed to cover such areas of intelligence in addition to conventional business applications-/information precessing-specific systems, TIPPLER serves as an integrated development support tool operable on workstations —— as a software platform for brand-new navigation systems.

What initially motivated the development of TIPPLER included (1) the practical, effective diversion of sophisticated engineering workstations then about to be widely accepted for business applications, (2) the assuring of remarkably higher productivity and expandability in business applications through the use of the object oriented technique, and (3) the taking of graphical user interfaces into modeling for easier applications development, instead of offering them simply as items listed in the library, with the fact taken into account that GUIs were not being made full use of because of heavy workloads always involved on the side of applications developers though they turned out beneficial for end-users, who were not yet accustomed to computer usage.

This paper describes how TIPPLER has been desinged and implemented up to the presnet as well as what to be done for further enhancements.

1. はじめに

ハードウェアプラットフォームの急速な技術革新は，ダウンサイジングの潮流を生み出した．しかし，ソフトウェア技術の追従が遅れ，ハードウェアプラットフォーム

本稿に記載の会社名，商品名等は，一般に各社の商標または登録商標である．
TIPPLERは，㈱野村総合研究所と日本ユニシス㈱により共同開発されたソフトウェアである．

の能力を十に生かしたコンピュータシステムの構築が進んでいない，あるいは必要以上にコストをかけているのが現状である．ハードウェアプラットフォームの技術革新によって可能となってきた今後の新しいシステムの形態としてのナビゲート型システムの位置付けと，ハードウェアプラットフォームの能力を引き出しアプリケーション開発のダウンサイジングを図るソフトウェアプラットフォームの必要性について述べる．

1.1 問題の背景

コンピュータシステムの利用形態は，ますます多様化・高度化が進み，経営・マーケティング・営業力の強化に向けたシステム構築が必要とされている．効率的なデータ処理をめざしたプロセス型システム・業務系システムから，蓄積されたデータを情報へと集約し，情報による創造的思考，最適な判断をめざしたインフォメート型システム・情報系システムの構築が進められている．しかし，データの集約化の一方では，必要なデータが簡単に取り出せない，大量複雑となり一般ユーザ，現場の担当者では使い切れないといった問題も出ている．

そこで，集中・集約する一方のシステム構築だけではなく，これはこれでさらに高度化する必要があるが，集めた情報を現場に分散・還元し，日々の業務に生かすシステムが望まれている．情報による思考や行動の誘導をもたらすナビゲート型システムである（図1）．

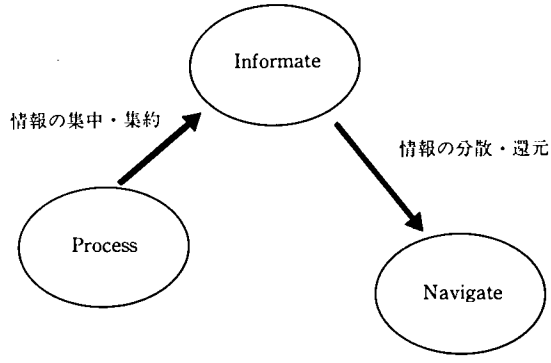


図1 システムの発展

Fig. 1 Change in computer systems

1.2 問題の整理

これからの新しいシステムの方向の一つと考えられるナビゲート型システムの実験システムとして，知的営業支援システム，知的経営支援システムを試作試行した過程で発見された機能要件や課題を，利用者の立場からナビゲート型システムがアプリケーションとしてどのような機能や性質を要求されているか，開発者の立場からナビゲート型システムを開発するためにはどのような問題が残されているか，企画者の立場から開発したナビゲート型システムを運用していくためにはどのような考慮をしておかなければならないか，という観点で整理してみる．

1.2.1 システム利用者の要求

現場のすみずみにまで情報を展開し，日常業務で利用していくことを想定すると，

これまでコンピュータにあまり触れていなかった人にもコンピュータを利用してもらうことが必要になる．たとえば，知的営業支援システムでは，セールスマンが顧客情報や在庫情報等を操作しながら電話をする，知的経営支援システムでは，さまざまな関連情報をみたり，シミュレーションをしながら経営会議を行う，等が考えられる．これまでコンピュータをあまり使っていなかった人がコンピュータを使い，電話をかけながら，議事進行の中で思考を妨げることなく，その活動を支援/誘導することが要求される．これらを実現するための要件として，以下の五点をとらえた．

① システムの素人でも直観的に使用できるインタフェース
② 思考を妨げないレスポンス
③ 集中分散共存型の目的別データベース
④ 関連した情報を自由に素早く取り出せる
⑤ 情報が不十分でも利用者を支援できる

1.2.2 システム開発者の要求

試作に使用したハイパーテキストソフトウェア，表計算ソフトウェアは，ナビゲーション型システムのイメージアップには効果があったが，詳細な仕様を実現することができなかったり，効率に問題が生じたり，他のシステムとの連係が取りにくいことがあり，実用システムの開発ツールとしては問題が残された．そもそも，これらのツールは，パーソナルな非定型業務に利用すると効果があるもので，作り込みが必要となる定型的な組織業務を意識して作られていないようである．

一方，ワークステーションの高性能化，GUI の普及によって，先に示した利用者の要求を満たすアプリケーションをワークステーション上で開発することはできるが，C 言語と X ライブラリおよび GUI ツールキットでは開発できたとしても，習得性が低く，誰にでもできるものではなく，また工数がかかる．

利用層が拡大し，さまざまな局面に展開されていくことから，いかに短期間にシステムを作り上げられるかを課題とし，以下の五点を開発者の観点からの要件としてとらえた．

① ソフトウェアプラットフォームの整備と活用
② GUI プログラミングの生産性向上
③ アプリケーションの部品化と再利用
④ 各種パッケージ・ソフトウェアの組込み，他のシステムとの連動
⑤ プロトタイピング・アプローチ

1.2.3 システム企画者の要求

一般的にソフトウェアプラットフォームの課題は，いくつかのアプリケーションが期待通りに効率良く開発できたとしてもソフトウェアプラットフォーム自身が陳腐化しないか，ベンダーの支援がきれて保守が困難にならないか，といった長期的な運用がネックとなることである．アプリケーション資産の将来が，ソフトウェアプラットフォームに依存することを避けるために，生産性や保守性をないがしろにしてでも，低水準な言語やツールを選択することがしばしば行われている．中長期のシステム運用や企画に携わっている人たちは，一見うまくいったシステム開発がかえってコスト高につくことがあることを経験しているからであろう．これらに対応するために以下

の三点を要件とした.

① 業界標準の採用

② 最新技術のキャッチアップ

③ オープンシステム

1.3 解決のための方針

問題は，個別のナビゲート型システムをどうつくるかではなく，どうすれば，期待通りのナビゲート型システムを効率良く開発/運用できるか，そのためには，どんなツールがあればよいかということである.

ハードウェア/基本ソフトウェアの性能面から，少なくとも開発に着手した1990年の時点では，ワークステーション/UNIX* が適当と考え，これを採用することにした.ワークステーションのアプリケーションは,C言語と第1~2層に用意されたライブラリ群を使用してスクラッチから作り上げなければならなかったが,C言語,ライブラリのAPIとも，アプリケーション開発のツールとしては抽象度が低く，生産性が上がらない，保守性が低い等の課題を抱えていた.UNIXはその生い立ちから，アプリケーション（とくにビジネス分野向け）のプラットフォームとしては歴史が浅く，その高いポテンシャルを十分に引き出しきれていなかった.

そこで，アプローチとしては，アプリケーションを記述するために適切な高水準言語と，その言語でアプリケーションを開発するための環境と合わせて，第3層にソフトウェアプラットフォームとして位置付ける.高水準言語で記述されたプログラムは，ソフトウェアプラットフォームの一機能であるコンパイラによって，第1層に用意されたC言語に変換する.新たに設計する高水準言語は，アプリケーションの特性を考慮して，GUI記述を強化したものとすること，また第2層のライブラリ群を部品として抽象度の高いインタフェースで使用でき,かつアプリケーションの部品化/再利用のためオブジェクト指向型言語とするのである.新しく定義した言語でプログラムを作成するための支援ツールは，新たに定義した言語を用いて開発することで開発者が自分の好きなようにカスタマイズできるようにし，部品群を第4層に位置付け，クラスライブラリ群として随時拡充していく（図2).

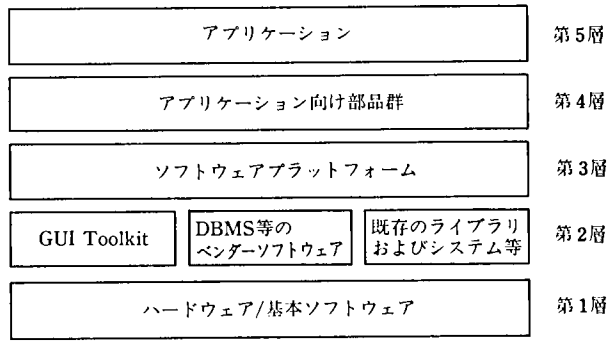


図2 アプリケーションの階層

Fig. 2 Application systems hierarchy

* UNIXは，UNIX System Laboratories, Inc.が開発し，ライセンスしている.

COBOL や FORTRAN によって，アセンブラでは不可欠であった細かい計算機の計算方法を意識せずに，事務計算，科学技術計算等のアプリケーションの記述に専念できるようになったことと同じように，それがどんなに小回りは効いても GUI アプリケーションを記述するという点ではアセンブラ的である C 言語ではなく，GUI を記述するために GUI をモデルとしてとらえた高水準な言語が必要である．将来的には，ネットワークアプリケーションもモデルとしてとらえた言語へ拡張していく必要があると考えている．アプリケーションの記述をライブラリ呼出しの列として実現するのではなく，モデル化し言語仕様に取り込んでいかなければ抜本的な生産性の改善ができないと考えた．

2. 統合開発支援ツール TIPPLER

統合開発支援ツール TIPPLER は，オブジェクト指向プログラミング言語を中核に，システムライフサイクル全体で発生する作業を統合的，包括的に支援するソフトウェアプラットフォームを目標としている．

システムライフサイクルを分析・設計・開発・保守ととらえた時に，GUI アプリケーションの最大の課題は開発段階の生産性にあった．オブジェクト指向という観点からは，プログラミング言語に関連する技術はほぼ確立されていたが，分析設計の手法について課題が残されている状況だった．将来的にはオブジェクト指向技術の普及が予想されたので，GUI アプリケーションの課題を，オブジェクト指向プログラミング言語と開発支援ツールで解決を図りつつ，合わせて試行，実験を繰り返しながら，上流工程にも適切な支援ツールを提供できるような拡張性を意識した．言語を中核においたのは，将来機能を拡張する際に，上流工程の出力としての外部形式を用意しておいた方が実験，拡張が容易であろうという狙いもあった．

新たにプログラミング言語を設計する必要を感じたのは，GUI アプリケーションのプログラミングの実態は，使用する言語の特徴に関わらず関数呼出しの列で実現されていることからである．X ライブラリを使用したプログラムは単調で量が多く意味が取りにくい．かつてアセンブラではレジスタへのロード/ストア/演算の羅列であったものを，実際の計算機の動きから離れて抽象化し，FORTRAN 等により算術演算として記述できるようになったことで，プログラムの生産性は大きく向上した．手続きの単調な羅列は，抽象度を上げ，モデル化し言語仕様に取り込むことが，生産性に最も寄与するのではないかと考えた．GUI 以外にも，ネットワーク/データベース/帳票/データエントリ等についても検討することも考えたが，モデル化し言語仕様に採り入れるということでは GUI に注力した．他の機能については，クラスライブラリとして実現する方法をとることにした．図 3 に現状のソフトウェア構成を示す．

2.1 オブジェクト指向プログラミング言語 UNISCRIPT

バッチ，キャラクタ端末を想定した情報システムの開発言語は COBOL や FORTRAN が中心で，不足する部分をアセンブラで補っていた．COBOL は大量データの事務処理，FORTRAN はバッチ型の科学技術計算を記述することを意識して設計されている．ナビゲート型システムを記述するのに適した言語仕様はどんなものか．われわれは，従来の第三世代言語の位置付けに「GUI のモデル化＋手続き言語＋オブジ

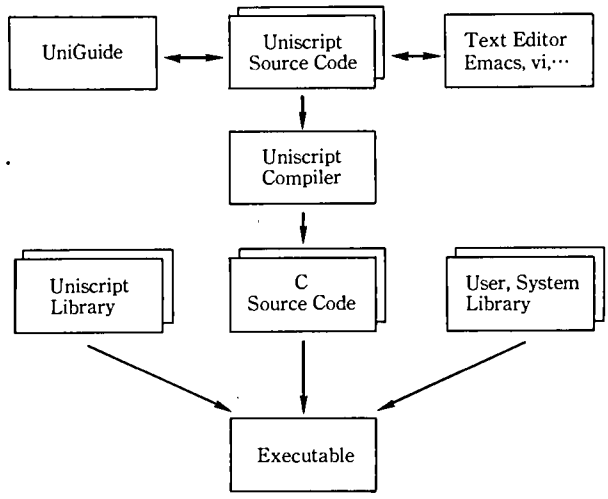


図 3 ソフトウェア構成

Fig. 3 TIPPLER software components

ェクト指向型言語」というイメージの言語をおき，C 言語をアセンブラにみたてて UNISCRIPT 言語を設計することにした．アプリケーションをなるべく UNISCRIPT で統一して記述できるようにすることを目標とし，汎用的な高水準言語をめざした．これまで構築されてきた数百本のアプリケーション（大きいもので COBOL 換算 100 万ステップ程度）は，ほぼ UNISCRIPT だけで記述されている．C 言語で補った部分は，ベンダーソフトウェア，ペリフェラルとのインタフェースをクラスライブラリ化する部分である．アプリケーション以外にも，実験として，ハイパーカードソフトウェア，表計算ソフトウェア等を UNISCRIPT でプロトタイプを作成し記述性を確認した．

手続きの記述は，PASCAL 風に基本操作を組み合わせてプログラムするため，初心者でも習得が早く，日本語識別子により文書性も高い．

GUI の記述は宣言的でプログラムから表示をイメージしやすいように設計した．

処理系については，インタープリタ，コンパイラの選択が考えられるが，インクリメンタルコンパイルの技術により，インタープリタの代表格である LISP を使った開発においてもコンパイラを使用する利用者が増えている．ある程度，実行までの時間を短縮することができれば，静的なチェックにより全体でのプログラミングの生産性を向上できる．思考を妨げない応答を満たすために，実行効率を重視してコンパイルすることを前提に言語仕様/処理系を設計した．

1) 基本データ型/制御構造……オブジェクト指向型に徹してみることが主題ではないので，手続き型が馴染みやすそうな部分は積極的に手続き型言語の特徴を採用することにした．if-then-else，while，for 等の制御構造，整数，実数，文字，配列等の基本データ型は手続き型言語を踏襲している．基本データ型を積極的に認めることで，これらの型については高速に演算できるよう生成コードを考慮した．

また，リスト，連想リスト等を基本データ型に追加して，プロトタイプ開発の生産性の向上を図った．実行時に長さ（大きさ）が決まる可変長データが扱えるようにし，プログラミングの段階であらかじめ最大長の見積りを行う必要をなくした．使用されなくなったデータ領域を自動的に回収し再利用するメモリ管理機能と合わせて，システム開発者の負担を軽減している．

2) オブジェクト指向プログラミング……「The Object-Oriented Database System Manifestro」に示されたオブジェクト指向システムの必須条件および付加条件を考慮し，オブジェクト指導プログラミング言語として必要な機能を採り入れた．

3) プレゼンテーションモデル……GUI を持つアプリケーションは，通常 C 言語と X ライブラリを用いて作成する．小さな機能の関数コールを組み合わせて GUI を開発しなければならないため，各関数の詳細についての知識，それらを組み合わせてまとまった機能を実現するための知識が必要となり，また生産性が低い．簡単なウィンドウを表示するためのプログラムもステップ数としては大きくなり，可読性が低く変更に弱い．表示の整合性を維持するための処理，イベントを受けた時の処理を記述するのが大変である．XVIEW* や MOTIF** 等のウィジェットを用いることもあるが本質的な差異はない．

そこで，一般的には，手続き中に記述した関数コールで実現していた GUI を，オブジェクト指向モデルを拡張することにより宣言的に記述することを狙って，プレゼンテーションモデルを設計した．XVIEW や MOTIF 等のツールキットで提供されるボタン，フィールド等のウィジェットに加えて，ビジネスアプリケーションを記述することはできない．スプレッドシート・ビジネスグラフ・自由図形等を追加して，アプリケーションの記述性，生産性を高めることにした．この詳細については別途報告したいと考えている。

4) テキスト形式……筆者も本当のところはよくわからないのだが，プログラマはテキスト形式の言語を好むようである．一つには，ハイパーカードソフトウェアや表計算ソフトウェア等に見られる編集機能は，局所的にプログラムの断片を見せてくれるだけで全体像がつかみにくい上に，プログラマは，文書作成，メイル，ニュース等の日常的な作業にテキストエディタを使用していることが多く，一つのインタフェースでどんな作業でもこなせる方が便利なのではないか．過去，さまざまな構造化エディタがありながら，結局テキストエディタが広く使われている要因なのかもしれない．

またプログラマにとっては，プログラムそのものがデータということもある．たとえば，プログラムを自動生成したり，フィルタによって変形したりする．実際，UNISCRIPT 処理系のテストのために，テストプログラムを自動生成している．TIPPLER の開発支援ツールは，UNISCRIPT プログラムを編集するエディタであり，実験中の分析設計支援ツールは，UNISCRIPT プログラムを自動生成する．その他にも，UNISCRIPT プログラムを出力するクラスライブラリ，フィ

* XVIEW は Sun Microsystems, Inc. の登録商標である．
** MOTIF は Open Software Foundation の登録商標である．

ルタを開発している．

昨今，開発支援ツールとしてビジュアルプログラミングの機能の提供は当然のことだが，熟練したプログラマのためにテキスト形式によるプログラミングの余地を残すことにした．UNISCRIPT 言語を中心に，プログラマが好きなエディタを使えるように配慮している．プロトタイプから大規模アプリケーションまで一貫して UNISCRIPT で開発されているのは，テキスト形式の汎用言語であることが一因になっていると考えている．

5) 他システムとの連係……COBOL や FORTRAN でアプリケーションを記述しきれずに，アセンブラが必要になるように，UNISCRIPT を汎用的な言語としても，C 言語で書かなければならない部分が残ると考えた．たとえば，C プログラムとの API を提供しているベンダーソフトウェアとインタフェースを採りたい時に，UNISCRIPT プログラムから C 言語で作成した関数を呼び出せれば，簡単にインタフェースを作成し，あとは UNISCRIPT で記述することが可能である．

また，UNISCRIPT プログラム中から，既存の実行形式プログラムやシェルコマンドを起動できるようにした．パイプラインを使って，コマンドの標準入力・出力をファイルとして利用することもできる．さらに，コマンドの標準出力を非同期に入力することができ，リアルタイムデータ等を非同期入力することも可能である．

2.2 コンパイラ/ランタイム

コンパイラ/ランタイムの課題は実装であり，以下の三点についてとくに注意した．

① 動的束縛の効率

② 表示の整合性を維持するための再計算/再表示の効率

③ ハードウェア/基本ソフトウェアの非互換の UNISCRIPT ソースレベルでの吸収

オブジェクト指向プログラミング言語の実装においては，実行時のインスタンスによって，実際に呼び出すメソッドを切替えなければならないために動的束縛が必要になるわけであるが，頻繁に発生する動的束縛のアルゴリズムの善し悪しが，実行効率に大きく影響する．UNISCRIPT 言語の実装においても，これまでに三度にわたって，この部分の全面的な書換えを行い実行効率の改善を進めてきた．この詳細については別途報告したいと考えている．

プレゼンテーションモデルの機能の一部として，表示データの整合性を維持する機能がある．通常，データの更新の都度，闇雲に再表示をしていてはレスポンスが悪くなるので，データ変更の影響範囲を特定して，再表示するための処理を記述しなければならない．UNISCRIPT でプログラムを作成する人は，プレゼンテーションモデルの機能によりこのための処理を記述する必要がない．したがって，明示的には記述されていない整合性維持のための再表示を，データ間の依存関係によりどのデータの再表示が必要か，そのデータがウィンドウのどこに表示されているか計算し，なるべく少ない再表示で表示の整合性を維持しなければならない．この詳細についても別途報告したいと考えている．

UNISCRIPT コンパイラは UNISCRIPT プログラムを C プログラムに変換し，通

常のC言語処理系を通して実行形式を得る．オープンシステムを目指すTIPPLERは，対象となるハードウェア/基本ソフトウェアの非互換を吸収するために，UNISCRIPTコンパイラでCコンパイラの非互換を吸収し，UNISCRIPTランタイムでウィンドウやシステムコールの非互換を吸収し，全体としてUNISCRIPTソースレベルで互換性を実現することにした．

2.3 開発支援ツール UNIGUIDE

プロトタイプアプローチによる開発を効率良く行うためには，インタプリタとデバッガ等からなる開発支援ツールが有効である．複数の開発者が並行してアプリケーションを開発ができるように，UNISCRIPTのプログラムの格納と排他制御機能を持つリポジトリサーバ，クライアントにブラウザ，インタプリタ，デバッガを用意することにした．UNIGUIDEはその総称である．

UNIGUIDEは，リポジトリサーバとのインタフェースとなるクラスライブラリと他システムとの連係機能を使ったUNISCRIPTのアプリケーションプログラムになっている．このため，アプリケーション開発者の好み，開発するアプリケーションによってUNIGUIDEのインタフェースをカスタマイズすることが可能である．

1) ブラウザ……リポジトリに格納されている定義を視覚的に検索，編集することができる．クラスの階層構造を直観的に理解できるように表示し，スロット，メソッド定義の編集，新しいクラスの定義を会話的に行うことができ，既存の部品の再利用，カスタマイズが容易となる．プレゼンテーションの定義を，マウスを使った直接操作によって，作成・編集することができる．これによって，表示画面を実際に見ながら画面のレイアウトを編集することができ，見栄えの良い画面を容易に作成できる．

2) インタプリタ……クラス定義，メソッド定義，フレーム定義をロードした後に，UNISCRIPT実行文を逐次実行することができる．プログラムを変更することなく，メソッドの呼び出しをトレースしたり，呼び出し時に割り込みをかけてデバッガに入ることができる．同様に，実行中にエラーが発生するとデバッガに入る．

3) デバッガ……割り込みが発生してインタプリタからデバッガに制御が移ると，その時点での変数の値，インスタンスの属性値をデスクトップ環境で検索・参照・更新することができる．また，インタプリタで割り込みが発生した時点に復帰し，実行を再開することができる．

2.4 クラスライブラリ

データベースインタフェース，帳票，データエントリ等は，ビジネスアプリケーションに欠かすことのできない機能ではあったが，従来のアプリケーション開発において手法(SQL等)が定着していること，実際に困っているGUIのモデル化の方が効果が大きそうなことから，モデル化はGUIに注力し，それ以外はクラスライブラリとして用意することにした．データベースインタフェースには，動的SQL文，標準的なエラー処理機能等，帳票には，デスクトップ環境でフォームオーバーレイを作成する機能，帳票を編集/検索/管理する機能，非定型帳票を作成する機能等，データエントリには，キャラクタ端末イメージでの大量データ入力機能等がある．

マルチメディアについては，それを実際の業務に適用した時の業務の流れ/進め方の

変化，利用者からの実需がはっきり見えなかったので，ペリフェラル関連の入出力および制御のためのクラスライブラリを用意するにとどめた．イメージスキャナ，FAXモデム，ビデオカメラ，スティルカメラ，レーザーディスクプレーヤ等を利用したアプリケーションを作成することができる．

3. おわりに

GUI記述を言語モデルに採り入れたオブジェクト指向プログラミング言語と諸ツールを提供することを目標に，言語処理系と開発支援ツールを中心に開発を進めてきた．オブジェクト指向技術は，注目を集め普及の方向にあるが，技術として定着してきているのはプログラミング技術であり，分析設計技術・データベース技術・ネットワーク技術は十分に確立されている状況にない．今後の開発においては，新技術への対応という観点からも，これまで並行して実験を進めてきた技術について製品への実装を始める．具体的には，オブジェクト指向分析設計を含めたシステムライフサイクル全般にわたる方法論と支援ツール，オブジェクト指向データベース/ネットワークをモデル化した分散オブジェクトを実現することである．GUIのモデル化/言語仕様への組み込みと同様にモデル化を通して，わかりやすいプログラムが記述できるようにしたい．

また，1993年3月，UNIXの有力ベンダー6社からCOSEが発表され，UNIXの一本化がようやく現実のものとなった．UIとOSFの2分化はユーザに対して決してよい影響をもたらさなかったが，COSEは別の2分化，UNIXとWindows*の始まりと考えている人も少なくない．TIPPLERをソフトウェアプラットフォームとして利用することにより，アプリケーションをこれらのハードウェアプラットフォーム/基本ソフトウェアと独立したものとし，アプリケーションを変更することなくさまざまな環境で利用することができるようになる．業界標準への対応，オープンシステムという観点から，TIPPLERのソフトウェアプラットフォームとしての稼働環境を広げることを目標に，Motif版の出荷およびWindowsへの対応を開始した．

参考文献
[1] G. L. Steel Jr., Common Lisp : The Language, Second Edition, Digital Press, 1990.
[2] B. W. Kernighan and D. M. Ritchie, The C Programming Language, Second Edition, Prentice Hall, 1988.
[3] N. Wirth, 系統的プログラミング入門（第二版補訂），近代科学社，1986.
[4] M. Atkinson, The Object-Oriented Database System Manifestro, Proc. of the 1st International Conference on DOOD, 1989.
[5] J. Rumbaugh, Object-Oriented Modeling and Design, Prentice Hall, 1991.
[6] Sun Microsystems, Inc., OPEN LOOK Graphical User Interface Application Style Guidelines, Addison Wesley, 1989.
[7] Sun Microsystems, Inc., OPENLOOK Graphical User Interface Function Specification, Addison Wesley, 1989.
[8] W. R. Stevens, UNIX Network Programming, Prentice Hall, 1990.

* Windowsは米国Microsoft社の商標である．

執筆者紹介　保　科　　剛（Tsuyoshi Hoshina）

昭和56年 東京理科大学理学部 応用数学科卒業．同年日本ユニシス(株)入社．数理計画，人工知能を応用したアプリケーション開発，プロダクト開発に従事．現在，システム技術本部 知識システム部に所属．日本OR学会会員．

リポジトリ・システムのあり方とその実現に向けて

An Implementation Approach for a Standard Repository System

原 田 公 敬, 城 代 優 二

要 約 コンピュータ・ネットワーク技術や, UNIX* を始めとするマイクロ系プロダクト技術の進歩に伴い, ダウンサイジングを始めとした分散（オープン）システム環境指向の流れが定着しつつあり, 企業の情報利用形態や情報システムの構築方法に変化が現れ始めた. このような分散環境においては, 情報の統合管理や各種ネットワーク管理等のために, リポジトリの役割が重要となる. 現時点でのリポジトリの実装レベルでは, システムの開発・保守に関する情報をリポジトリに格納し, それを中核として開発・保守作業を支援するCASEツールの分野が, 最も先行していると言える.

本稿では, まずリポジトリ・システムの概略について述べ, その後リポジトリ・システムを実現するために必要な手順について述べる. そして具体的要件を仮定して, この手順の実践を試みる. また, 実際にリポジトリ・システム構築への試みを実施している現場の事例についても紹介する.

Abstract With remarkable technological progress as seen these days in computer networking and micro-products including UNIX-related systems, computer user trends are gaining ground toward systems downsizing and distributed (open) systems environments, thus beginning to have influences on the processing of corporate information and the way information systems are built. In such open systems computing environments, the role of a repository is considered highly important for the management of integrated information and varieties of networking.

In view of the current level of repository implementation, CASE tools are regarded as having the most advanced technology, which are designed to support systems development and maintenance with the help of the repository where all such related information is stored. This technical report first briefs on a scope of repository systems, and then describes the process steps required for their creation, in addition to indicating how the approach proposed by the authors can be applied for assumed specific requirements. Also shown is a sample shop-floor effort to create a repository system at a certain customer.

1. はじめに

分散システム環境の発展に伴い, ソフトウェアの開発/実行も分散化されつつあり, それらの環境を管理するリポジトリ・システムについて, ANSI, ISO等の標準化団体で標準化が進められている. これに準拠する形で, 関連各社は分散リポジトリの枠組み/プロダクトの提供を開始している. 弊社も, UA/ASDFの中でリポジトリについての計画を発表している. また, 今後のシステム開発に重要な役割を持つと考えられるCASEツールも, 開発に関する各種情報をリポジトリ上で管理することから, リポジトリ・システムとは不可分なものとなっている. このように, 今後のソフトウェア環

本稿に記載の会社名, 商品名等は, 一般に各社の商標または登録商標である.

* UNIXオペレーティングシステムはUNIX System Laboratories, Inc.が開発し, ライセンスしている.

境はリポジトリ無しには語れなくなると言われるほど，リポジトリ・システムの重要性は増してきている．すでに，リポジトリを中核としたシステム構築の検討を開始したユーザもあり，リポジトリ・システム実現に向けての具体的な活動が，現実に始められている．

このような状況の中で，筆者らもこの分野に関する技術研究/調査の場を与えられ，まずリポジトリ・システムについての一般的な標準化および技術動向を調査し，現状を認識することから開始した．この調査を基に，リポジトリ・システムを実現するために必要な手順を検討し，次に与えられたユーザ要件に対してこの手順を机上で実施し，実際にその要件を満たすものを作ることができるかの検証を行った．幸いなことに，その後実際にN社のリポジトリ・システム開発に参画することができ，現在この手順の正当性の更なる検証を継続中であるが，5章ではその途中経過を紹介する．

本稿ではあるユーザ要件を想定し，机上でリポジトリ実現のステップを実践するにとどまっているが，ここで紹介する手順により，トップダウン/ボトムアップの両面から段階的に実施していくことが，リポジトリ・システム実現の現実的な方策であると考えている．

2. リポジトリの現状認識

2.1 リポジトリの起源

リポジトリという概念が世の中に登場したのは，1989年に統合CASEのためのデータベース共有の概念として考案されたIBM社の「AD/Cycle」からである．これに類する概念は，それ以前から以下の二つのコンピュータ・エンジニアリング手法の中に存在しており，リポジトリはこれらの手法の流れが合流した形で誕生した[2]．

一つは，DBMSに始まるIRM（情報資源管理）手法としての「ディクショナリ」である．データ情報の統合管理を目的とし，データそのものの管理に始まり，データの意味情報であるメタ情報の管理も行うことにより，IRMの実現を図っている．

もう一つの流れは，CASEツールにおけるシステム開発・ライフサイクル（SDLC: System Development Life Cycle）管理手法としての「エンサイクロペディア」である．システム開発の各工程で発生する開発データの集中保管，および次工程への矛盾の無い引渡しによる開発/保守生産性の向上を目的とするもので，SDLCの各工程を個別に支援するC-CASE（Component CASE）に始まり，最近ではSDLC全体を統合して支援するI-CASE（Integrated CASE）が出現してきている．

この二つの流れが合流するきっかけとなったのは，1988年のANSI（American National Standard Institute）による標準ディクショナリであるIRDS（Information Resource Dictionary System）の制定と，AD/cycleの出現である．AD/Cycleのリポジトリは，IRDSには準拠していなかったが，その後DEC社の統合CASE概念であるATIS（A Tool Integration Standard）がIRDS準拠を打ち出し，これによりIRMとCASEの流れが合流することとなった．その後，ISO，ECMA等の団体による様々な標準化が行われ，CASEツールではADW，IEF* 等，新たなプロダクトが出現している．CASEツールの多くは標準化されたデータモデルを用意し，そのユーザ・イン

* ADW, IEF: Ernst & Young CASE Technology社，Texas Instruments社の統合CASEツールである．

タフェースとして幾つかのダイアグラムを用いてリポジトリに展開するという形態をとっており，ダイアグラムの意味表現を格納する機能が，リポジトリ・システムの大きな特徴となっている．近年では，オブジェクト指向技術（OOA：Object Oriented Approach)の普及により，データモデルの設定にオブジェクト指向の考え方が取り入れられるようになり，リポジトリ上でのデータ格納形態と，ダイアグラムによる入力インタフェースとが，より簡易な形で実現できるようになりつつある[3]．

2.2 リポジトリ・システムの必要性

リポジトリに格納する情報は，それを利用するユーザの要件により決まるものであるが，一例としては次のようなものがある．

業務情報：組織情報，業務規約，業務用語，業務活動に必要な各種情報

開発情報：システム/プログラム等の仕様情報および仕様書，プログラム，ファイル/レコード/画面等の構造形式，データモデル/プロセスモデル等のダイアグラム

リポジトリとしての特徴でありまた重要な役割を果たす機能は，これらの情報そのものの管理だけではなく，その意味や関係も含めた，いわゆるメタ情報も体系的に意味付けて格納する機能である．

近年，以下のような観点からリポジトリ・システムの必要性が，クローズアップされつつある．

1) 企業経営の観点から見たリポジトリ・システム……経営資源としての情報の有効利用や，社内情報の社外への情報提供サービス等の業容拡張に伴う，企業のメタ情報を含む社内情報統合管理の必要性[4]．

2) システム開発/保守の観点から見たリポジトリ・システム……開発の上流工程から下流工程までの情報の一貫性の保証と，開発の生産性および品質の向上を目的とした，メタ情報を含むシステム開発情報の体系的な統合管理の必要性．

3) システム運用/保守の観点から見たリポジトリ・システム……オープン環境における，市販プロダクトを始めとする多様なソフトウェア，ハードウエア，ネットワーク等の，各種構成/配置状況の統合管理，およびそれらの連動を実現するためのインタフェース統合の必要性．

4) 情報管理場所および保全の観点から見たリポジトリ……情報の集中配置（大規模ホストマシン上）から分散配置（マイクロ系マシン上）へのシフトに伴う情報配置場所等の統合管理，および情報保全性の管理（セキュリティ保証，分散管理されている情報の整合性維持等）の必要性．

2.3 リポジトリ・システムの効果

リポジトリ・システムによる特徴的な効果は，データ中心アプローチによる情報の統合管理が可能となることである．データ中心アプローチは，現実の世界の中でプロセス（処理手順）よりは比較的安定性が高いと言われるデータに着目し，それを体系立てて管理した上で，それを使用するビジネス上のユーザ要件を実現させる問題解決手法である．このように，管理されたデータに着目してプロセスを組み立てることにより，プロセス間のデータ共有が可能となり，データ処理の一貫性，整合性の保証も実現できることになる．

リポジトリを先行的に取り入れているシステム開発・保守の領域においては，システムのライフサイクル全般（分析から運用まで）で発生する各種情報をリポジトリで統合管理し，その上で稼働するCASEツールを提供することにより，各工程間のシステム要件や設計情報等の一貫性・整合性を保証して，生産性/品質の向上を実現している．またソフトウェア変更時の影響範囲の把握が可能となることにより，保守性も向上することになる．

3. 開発支援環境におけるリポジトリの役割

2章でリポジトリ・システムの現状について述べたが，統合CASEツールに見られるように，リポジトリの活用という点で，先行しているのがシステム開発・保守の領域である．

システム開発の生産性，品質の向上策として今日まで叫び続けられてきたのは，分析から運用までのシステム・ライフサイクル全般にわたっての作業の標準化である．これは個人差から生じる不確かさを最小に抑えることに他ならず，この標準化はガイドと呼ばれるドキュメントで，作業手順と作業に伴う成果物を規定するものである．多くの場合，このような規定は作業者に対しその作業手順の把握/理解や，実際の開発作業とは別の然るべきアウトプット（主として報告書等のドキュメント類）の作成を要求するものである．このような方法では作業者の負担が増すだけで，不満とやる気のなさを生み出し逆効果になることが多かった．このような状況を解決するポイントは，作業の中に個人差の入り込む余地を無くすことと，ツールを提供して作業者の負担を軽減し，やる気を出させることであり，CASEツールは正にこの実現を目指して出現してきたものである．CASEツールに共通していることは，人手に頼っていた作業部分をできる限り取り除き，機械に任すことである．すなわち人が行う作業を最小にし，自動化の部分を最大にすることである[5]．

このCASEツールを支えているのが，個々に搭載しているリポジトリ・システムである．多くの場合，CASEツールは一つの商品としての閉じられた世界であるため，その仕様に合った独自のリポジトリ構造を持ち，機能を提供している．そして，機能追加等によりリポジトリの構造を変更する時でも，ベンダーがすべてを処置し新しいバージョンとして利用者に提供していることが多い．

現実を見ると，一つのCASEツールだけでシステムが開発できる訳ではなく，他のソフトウェアを導入・併用していることが多い．このような環境では，閉ざされた独自の世界から開かれた世界への脱皮が必要となってくる．すなわち，さまざまな環境の変化に耐えられるリポジトリのデータモデル（抽象化したシステムのライフ・サイクルを支援する世界）と，API等のリポジトリ・インタフェースを設定し，リポジトリが押さえる共通的な情報や，リポジトリへの共通インタフェースを定め，ツール開発の基本となる開発用ガイドを作成することが必要となる．この具体的アプローチは，前記のAD/Cycleに見ることができる．

このようにリポジトリ・システムで大事なことは，リポジトリに格納された情報を利用者に直接利用させるのではなく，ツールを通して利用させ，リポジトリ情報の精度を高めると共に，利用者の負担を軽減するツールを始めとする周辺ツールに，情報

利用機能（リポジトリ情報の格納・参照）を装備し，利用者にリポジトリ・システムを意識させずに恩恵を与えることである．

以上を整理して，開発環境においてリポジトリ・システムを適用するために必要な要素の骨子を列挙すると，次のようになる．

① 作業の標準化（方法論と技法の確立）
② 作業の自動化（方法論と技法を支えるツールの提供）
③ 作業に関わる情報の共用（リポジトリの標準データモデルとインタフェースの設定）
④ 情報の隠蔽（ツールによるリポジトリへの情報格納/利用）

4. リポジトリ・システム実現に向けての考え方

4.1 実現のためのステップ

前章で述べたことは，とくにシステム開発に限ったものではなく，システム運用や企業情報の統合管理等，あらゆる対象領域において前記の①から④のすべてが満たされなければならないことを示唆している．ここまでの事柄を踏まえて，リポジトリ・システムを実現するために必要となる活動ステップを，次のように設定する．

① リポジトリ全体像の設定
② リポジトリ・システム概要の設定
③ 標準リポジトリ・インタフェースの設定
④ 対象領域ごとに以下の作業
- ユーザ要件の洗い出し・整理
- 標準データモデルの設定
- ツール開発ガイドの設定
- ツールの開発

対象領域を明確にするために，まずリポジトリの全体像を設定し，リポジトリ・システム概要とAPI等の標準リポジトリ・インタフェースを設定する．次に，リポジトリ・システムの個々の対象領域ごとにユーザ要件を洗い出し，それらを満たす標準データモデルを全体像の中に設定する．その上で要求機能を実現するツールをガイドに準拠して開発する．そして，これらの部分々々を統合・整理し徐々に広げてゆき，リポジトリ・システムを完成させる．

標準リポジトリ・インタフェースに関しては，さまざまな標準化の動きがあるのでそれに準じることが望ましい．それ以外の項目について，以降で若干の説明を加える．このステップの中でとくに強調したいのは，ガイドの基本となる④の2番目“標準データモデルの設定”の必要性である．

4.2 リポジトリの全体像

2.2節で述べたリポジトリ・システムの必要性の観点から，リポジトリの全体像として，次に説明する五つの辞書とその間に必要と考えられる関連を設定した（図1）．

① 業務辞書：業務に関する情報を管理する（たとえば，業務モデル，業務規約，業務用語等）．
② ドキュメント辞書：規約，基準，事例，等の標準規則および参考情報を管理

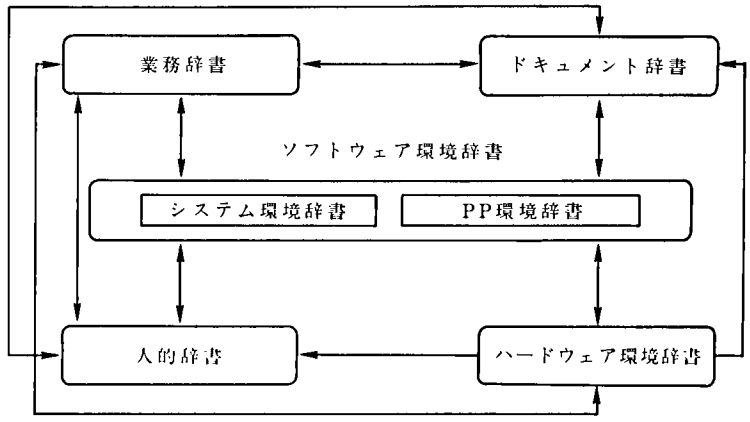


図 1　リポジトリの全体像
Fig. 1　General scope of repository

し公開する．

③　ソフトウェア環境辞書：ユーザ AP (アプリケーション) とベンダー提供ソフトウェアを管理するために，それぞれシステム環境辞書と PP(プロダクトプログラム) 環境辞書の二つを設定する．

- システム環境辞書：ユーザ AP の構成 (AP システム，ジョブ，プログラム，モジュール等) を管理する．
- PP 環境辞書：OS，DB，DC，言語，開発支援系，等のベンダー提供のソフトウェアを管理する．

今回の検討における具体的要件としては，この PP 環境辞書の管理要件を取り上げている．

④　人的辞書：エンドユーザ，SE，オペレータ，管理者(データ，データベース，開発，運用) 等の人的情報を管理する．

⑤　ハードウェア環境辞書：ホスト，サーバ，ワークステーション，ネットワーク，周辺機器，等のハードウェア・システムの構成を管理する．

4.3　リポジトリ・システムの概要

リポジトリ・システムは図 2 に示すようにリポジトリ，リポジトリ・マネージャ，ツール群から構成される．分散環境ではこのリポジトリ・システム構成要素が次のような形態で分散配置されることになる．この分散環境でのリポジトリの統合管理機能の検討も重要である．

①　プラットフォーム別（ハードウェア構成での分散：水平/垂直等）

②　用途別（たとえば，開発系と実行系との分散）

③　地域別（たとえば，東京本社と大阪支社との分散）

リポジトリに関して責任を持つのがリポジトリ・マネージャであり，リポジトリの維持管理を行う．リポジトリ・マネージャは次の機能を提供する．

1)　リポジトリの標準データモデルを設定し，インスタンスの作成（登録）だけで必要な情報をリポジトリに格納できる標準環境を提供する．

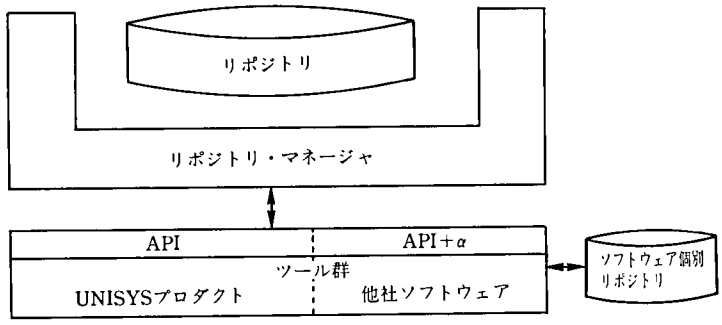


図 2　リポジトリ・システム構成

Fig. 2　Component of repository system

・標準データモデルは共用できるものである．たとえばシステム環境・構成情報の管理であれば，新しいソフトウェアあるいはハードウェアを導入する場合には，標準データモデルに対して個々のインスタンスを作成すると考えればよい．このように，標準的なデータモデルをあらかじめ用意しておけば，リポジトリ利用者のデータモデル作成の負荷を軽減することができる．

2)　API を介して利用者（ツール開発者）に次の機能を提供する．

・オブジェクトに対する登録，更新，削除，検索，版管理等の API

・標準データモデルをカスタマイズするためのクラス，サブクラス，属性，関連，操作等を定義する API

・分散環境でのリポジトリ間の情報転送のための API

・ソフトウェア個別リポジトリとの情報受け渡しを行う API

3)　データモデルを表示/出力（全体，部分詳細）する機能を提供する．

ツールは UNISYS プロダクトと他社ソフトウェアとに区分し，リポジトリとの入出力形態を次の通りとする．

・　UNISYS のプロダクトは上記の API を使用して，リポジトリの情報から他ソフトウェアとの関連を知り，自身の公開すべき情報はリポジトリに格納し，システム全体での整合性を保つ形で開発する．

・）　他社のソフトウェアを導入する場合は，直接ソフトウェアに API を埋め込めないため，何らかの策（α）を講じリポジトリと連携をとる．リポジトリに対する情報利用（参照・格納）の目的は UNISYS プロダクトと同じである．

リポジトリと各ソフトウェアが持つ個別リポジトリの位置付けとしては，次の二つ形態が考えられる．

①　リポジトリでは UNISYS プロダクトと他社ソフトウェアを区別せず，各ソフトウェア間で共通の管理オブジェクトと管理項目を保存し，ソフトウェア固有の管理情報はソフトウェア個別リポジトリで管理する．

②　UNISYS プロダクトに関する情報は，すべてリポジトリに保存し管理する．

管理対象とするソフトウェアの追加や，ソフトウェア固有の情報の頻繁な変化に耐え得るリポジトリ構造を定義するためには，①のリポジトリ構造の方が望ましい．

4.4 標準データモデル

リポジトリにさまざまな情報をモデル化し，かつ将来の変化に耐え得る仕様を作り上げるには抽象化が必要である．

データモデル化としてはERモデルも考えられるが，汎化—特化関係（is—a関係）と全体—部分関係（part—of関係）というように，より抽象化が表現できるオブジェクト・モデルの方が妥当と判断する．リポジトリのデータベース・エンジンとしてOODBを採用するならば，データモデル表現がそのままリポジトリ・システムの開発で利用でき望ましいことである．本稿では，オブジェクト・モデル化の一技法であるOMT（Object Modeling Technique）による表記を試みている．OMTモデルは，オブジェクトモデル，動的モデル，機能モデルからなるが，今回はオブジェクトモデルを使用して，標準データモデルの表記を行った[3]．

4.5 具体的要件に基づくアプローチ

本節では，リポジトリ・システムに対する具体的な要件として，図1のPP環境辞書内のDB，言語，開発支援系のソフトウェアに対して以下のものを仮定し，DB中心のデータ定義とデータ操作の範囲で，前記した手順の一部である標準データモデルの設定を試みる．ここではシステム環境辞書，ハードウェア環境辞書等との関係情報（データ定義変更時にどのシステム構成要素を処置すれば良いか等）には言及していない．

1) 仮定要件……DMS 1100，RDMS 1100，MAPPER，FACILE，ORACLE*，informix**，IDES，LINC，RDIP，ALLY等のソフトウェア群が，それぞれ運用上の辞書を必要とし，現在個別の辞書を持っている．利用者はその用途に応じて複数のソフトウェアを導入・併用することになるが，全体での情報の一元管理がなされていないため以下の問題が指摘されている．

① 個々の辞書設計が異なっており，各辞書間の連携がとられていない．辞書設計についての標準的なルール，ガイドがない．

② データ定義・操作に関して，ソフトウェアごとに言語が異なり，それらを習得しなければならず煩雑である．

③ 利用者はデータの特性等により使用するソフトウェアを選択するが，その選択基準が示されていない．一旦選択し使用開始したソフトウェアから別のソフトウェア使用に変更する場合には，そのシステムやツール等，初めから作り直しとなる．

④ 同一データが複数のソフトウェアで管理されることがあり，そのようなデータ更新整合性の対策を人手（経験と保有知識）に任している．

⑤ EIS（エグゼクティブ・インフォメーション・システム）等で必要性の高まっている，緊急なデータ検索要求“何時でも何処でもほしいデータを即時に見たい”が，データがどのソフトウェアで管理され，どのサーバに配置されているか判らないため実現できない．

⑥ 辞書を持つことで高度の制御が可能となったが，辞書の登録・維持を行う運用管理部門の作業量は格段に増えている．複数のソフトウェアを採用してシス

* ORACLEは米国ORACLE社の登録商標である．
** informixは米国informix社の登録商標である．

テムを構築することになると，ソフトウェア間の関連等，さらに運用管理が複雑になる．

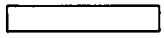
⑦ ソフトウェア同士が連携することにより，システム環境変更のタイミングが限定されるようになってきた．たとえば，ソフトウェアによってはその入れ替えのために，システムをある一定時間停止しなければならない．

⑧ 開発支援系ソフトウェアの開発については，データベース等のデータも含めシステムのすべてを開発するのであれば，そのソフトウェアの閉ざされた世界で作業が進められるため，ソフトウェア個別のリポジトリで対応しても矛盾は生じない．しかし，使用するデータベースや他のソフトウェアのリポジトリがすでに存在する場合，情報の二重管理作業を避けることができない．既存の開発支援系ソフトウェアであるIDES, LINC, RDIP等は，DMS, RDMSの定義を連携のないまま，閉ざされた個別のリポジトリで重複管理しているのが現状である．

これらの問題が標準データモデルを設定し，管理情報をリポジトリに格納して利用することにより，解決されるか否かを検証する．

2) データモデル……図3はOMTにより表記しており，その表記法の意味は次の通りである．

① クラス，サブクラス



ボックスで表す．PP・定義体がクラス，OS系・DB系がサブクラスである．

② 継承

クラスとサブクラスをつなぐ三角形で表す．サブクラスOS系・DB系がクラスPPの性質を継承している．クラスとサブクラスの間で汎化—特化関係を表す．

③ 結合（association）

クラス，サブクラス間の線で表し，線の両端の印で基数を表す．全体—部分関係および関連を表す．

無印：1　　○：0か1

●：0かそれ以上　　◎：1かそれ以上

図3はPPを定義体との全体—部分関係でとらえ，PPを汎化—特化関係で表している．さらに定義体以下をユーザデータの観点に絞って表現したものである．全体—部分関係については，構成という結合名で表している．

【図3の解説】

- クラス：PPはサブクラス：OS系，DB系，DC系，言語系，開発支援系，オンライン系ソフトウェア等を持ち，性質(名称，レベル，メモリサイズ等の属性)を継承させている．
- PPと定義体の関係は，PPが何らかの定義体を持つことを示す．
- ファイル定義はさらにクラス階層（汎化—特化関係と全体—部分関係の組み合わせ）で表されている．
- 定義体をユーザデータに限定すると，さらにレコードとレコード構成要素とい

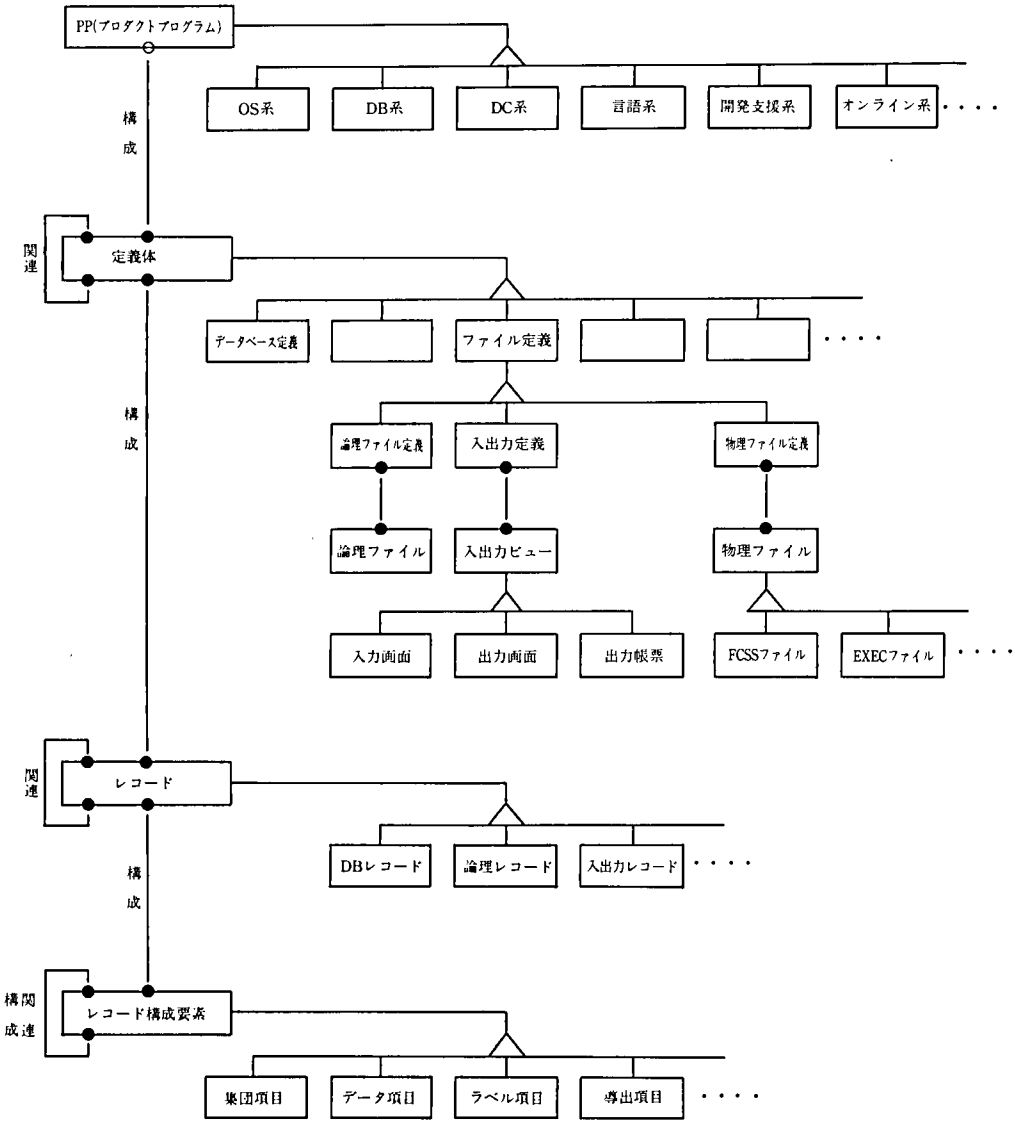


図 3　PP 環境辞書のデータモデル

Fig. 3　Data model of PP environment dictionary

う，それぞれサブクラスを持つクラスからなるクラス階層を形成する．

このように定義すると，一番抽象化の高い表現は PP—定義体—レコード—レコード構成要素のクラスからなる，全体—部分関係のクラス階層である．新しくユーザデータを扱うソフトウェアが登場した場合，このデータモデルの下でリポジトリに情報を登録することになるが，そのソフトウェア固有の定義（特化の末端定義）をどのように扱うかが問題である．ソフトウェア個別のリポジトリを持つ考えであれば，そのソフトウェアのインスタンスを作成（登録）するのみで，ソフトウェア固有の定義は個別に措置すればよい．固有定義もこのデータモデルの下で持つ考えの場合は，ソフトウェア固有の定義のサブクラスを新たに追加定義することで対処することになる．

このデータモデルを持つリポジトリにより，前記の問題のうち②以外のものを解決するための情報は，ここから得ることができる．したがって，この情報を使用して，実際に問題解決を行うためのツールを提供すれば良いことになる．

図3のPPのサブクラス：開発支援系をさらに詳細化したものが図4である．その他のサブクラスについても同様に詳細化を行ったが，紙面の都合上本稿では開発支援系についてのみ記述する．

【図4の解説】

- ユーザデータの観点で開発支援系ソフトウェアを表したのが図4である．開発支援系ソフトウェアはユーザデータ全般に亘る範囲の定義（物理ファイル，論理ファイル，データベース，入出力定義）を支援する想定で表現している．
- 開発支援系ソフトウェアは，データベースの定義/情報と整合を取って，ソフト

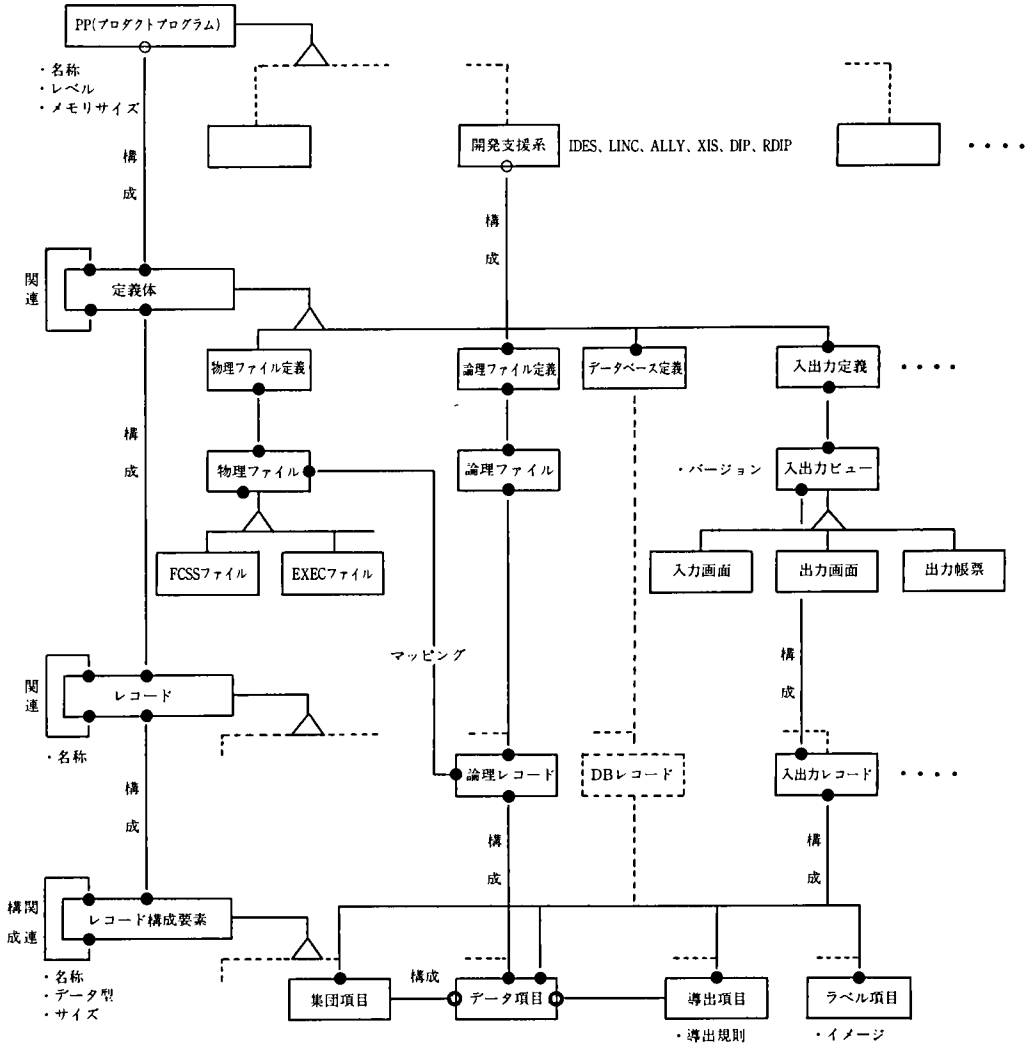


図4　PP環境辞書・開発支援系ソフトウェア・データモデル

Fig. 4　Data model of development support software

ウェアの開発を支援しなければならない．したがって，ユーザデータの定義情報を一つのリポジトリで管理することにより，開発支援系ソフトウェアの間で，API を通した相互参照が可能となる．また，一元管理により重複定義の防止，開発後の DB 系ソフトウェア側での DB 定義変更操作との連携等が可能となり，整合性のあるシステムが構築できる

- IDES を例にして論理レコードを定義すると，サブクラスとして IDES でいう物理レコード，論理レコード，エリア・レコード，コード・レコードを，このデータモデルに追加すれば良いことになる[6]．

3) ツール機能……ここでは，データ定義とデータ操作の範囲で必要となる機能について説明する．これらの機能を提供するツールと，リポジトリおよび各ソフトウェアの個別リポジトリとの関連を示すのが図 5 である．

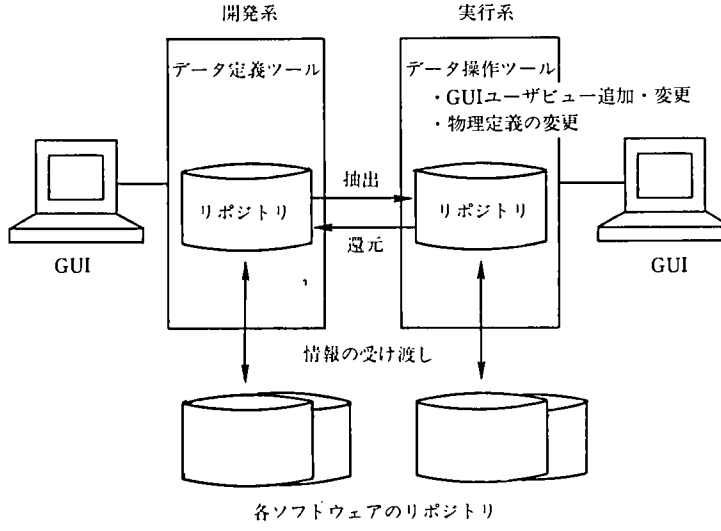


図 5　ユーザデータ一元管理機能

Fig. 5　Outline of user data management function

- 対象ソフトウェアは DB 系および開発支援系ソフトウェアを仮定しており，UDS, ORACLE, MAPPER, IDES, LINC, ALLY, 他社ソフトウェアを指す．
- 開発環境において，以下のデータ定義ツールを提供する．
  - ユーザデータを定義しリポジトリに登録する．この時のデータは各ソフトウェア間で一元管理すべき共通なものである．
  - 共通インタフェースとして，データ定義に関する言語，GUI, API を提供し，各ソフトウェアのデータ定義を一元管理する．
  - これらの共通インタフェースは，新しいソフトウェアの定義や，各ソフトウェア固有のデータ定義も行えるように，拡張可能な仕様でなければならない．
  - 各ソフトウェアの個別リポジトリとの間で，情報受け渡しができるインタフェースを提供する．
- 実行環境においてデータ操作ツールを提供する．

・開発系のリポジトリから，実行時に必要となる情報を実行系のリポジトリに抽出する．実行時に動的に追加・変更定義したGUIユーザビューおよび物理定義を，実行系のリポジトリに反映し管理する．これらの追加・変更のうち必要なものは開発系のリポジトリに還元し，開発・実行系全体の整合性を保証する．

・ここでは開発系と実行系でそれぞれリポジトリを分散した形で示しているが，場合によっては一つであってもよい．

・共通インタフェースとして，データ定義・操作に関する言語，GUI，APIを提供する．これは開発系のデータ定義に比べ，エンドユーザ層が利用するものであるため，次のような利用者要件を十分検討して仕様を決める必要がある．

① 簡易な入力でデータ操作が行えるGUI

② プロトタイプのGUIと，エンドユーザがそれを基にカスタマイズできるプレゼンテーション機能

③ ソフトウェア，データの所在場所を意識させずに行えるデータ操作(作成，追加，変更，削除，検索)

④ ユーザビューに対するバージョン管理や，更新されたデータのウィンドウへの即時反映等の，属性定義状況と操作との関係の管理

・これらの共通インタフェースは開発系のデータ定義ツールと同様に，新しいソフトウェアの定義や，各ソフトウェア固有のデータ定義も行えるように，拡張可能な仕様でなければならない．

・開発系のデータ定義ツールと同様に，各ソフトウェアの個別リポジトリとの間で，情報受け渡しができるインタフェースを提供する．

5. N社におけるリポジトリ・システム構想

1章で述べた通り，一部のユーザではリポジトリ・システムに着目し，実際にプロトタイプ構築による試行が開始されている．本章では，システム・ライフサイクルの全工程を標準化するために，各工程に最も適した市販のCASEツールを採用し，それらのリポジトリによる連動/結合を計画しているN社の事例について紹介する．

N社では，各部門ごとのアプリケーション・プログラム（AP）開発を，市販および自社開発のCASEツールを使ってオープン環境の下で行っている．開発工程は全社レベルで標準が決められており，各工程ごとに作成するドキュメントが義務付けられている．この出力情報（ドキュメント）を基に品質管理，生産性管理，プロジェクト管理を行う開発手順（開発方法論）が確立されている．

N社にとって，開発方法論に合致したI-CASEツールが市販されていて，それを導入できることが理想ではあるが，現時点では条件を満たすものがないため，各工程で最適のC-CASEツールを導入しているのが現状である．この開発環境を改善する手段としてリポジトリ・システムの構想が浮上し，平成4年4月より3期に亘る開発計画が開始された．

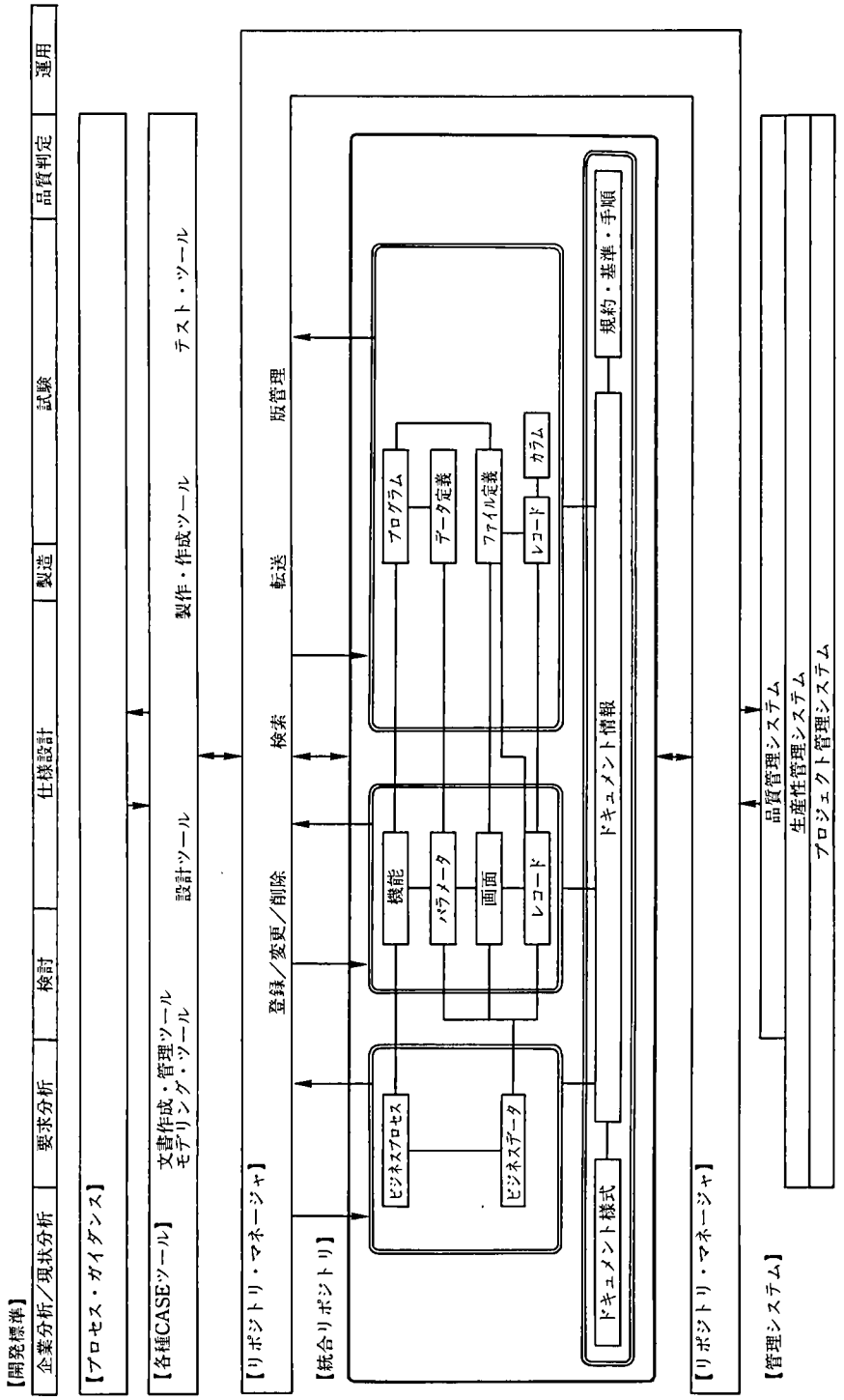


図 6　リポジトリ・システム概要図

Fig. 6　Component of repository system

5.1 リポジトリ・システム概要

各工程におけるC-CASEツールは市販のもので，その時点で最良のものを採用する．これらのCASEツールは，上流工程から下流工程へと，標準開発工程と親和性の取れた形で無理なく連動しなければならない．このとき標準開発方法論のために必要な情報は，この開発工程を通して自動的にどこかに蓄えられなければならない．連動情報と開発方法論のために必要となる情報を蓄える場所がリポジトリであり，その情報を蓄え有効利用するのがリポジトリ・システムである．リポジトリ・システムの要件を要約すると次の二点である．

1) C-CASEツールの統合化
2) 開発方法論の支援

リポジトリ・システムの概要を図6に示す．

【図6の説明】

1) 開発に携わる作業者は，WS (Workstation) からプロセス・ガイダンスを開き，自分の作業工程をメニューから選ぶ．
2) プロセス・ガイダンスは，そのメニュー選択に応じて，対応するCASEツールを起動する．
3) 作業者は，そのCASEツールの下で作業を行う．作業と同期して作成すべきドキュメントは，プロセス・ガイダンスの下でCASEツール（文書作成・管理ツール）が自動的に起動され，所定の形式画面に従って作成することになる．
4) プロセス・ガイダンスを経由することによって，各種CASEツールの連動情報と，開発方法論実施の管理システムが必要とする情報は，自動的にリポジトリに蓄えられる．
5) 作業者はプロセス・ガイダンスの操作法を覚えるだけで，CASEツール，リポジトリ・システム，管理システムを意識することなく自分の作業に没頭できる．
6) 管理者は，リポジトリに蓄えられた情報を使って，品質管理等の目的を達成できる．また，スケジュール管理等の目的でリポジトリに作業指示情報を格納し，それとプロセス・ガイダンスを連結することで，作業者への指示も可能となる．

5.2 開発計画と開発方針

開発は平成4年から8年までを期間とし，その中を3期に分けて段階的に進める．

開発ステップ	開発項目
I期	リポジトリ基本機能 ①プロセス・ガイダンス試作 ②リポジトリ・マネージャ基本機能(登録・検索・変更・削除等) ③市販の上流CASEツールAと自社開発の下流CASEツールBとの連動 ④LAN内アクセス
II期	リポジトリ拡張機能 ①プロセス・ガイダンス ②リポジトリ・マネージャ拡張機能(セキュリティ機能等) ③連動対象CASEツールの拡大およびインタフェース開示 ④WAN内アクセス(分散処理マネージャ)
III期	CASEツールと品質管理，生産性管理，プロジェクト管理システムとの連動 (情報の自動収集)

開発方針は以下の通りである．

1)　リポジトリについて国際・業界の標準化動向を意識し，標準準拠のシステム開発を行うことでオープン化を目指す．

2)　市販ソフトの有効利用を図る．

3)　マルチベンダ開発環境を意識し，システムの可搬性に留意する．

4)　ネットワークを介した相互接続性を確保する．

5.3 開 発 経 緯

今日現在，I期開発が終了した時点である．以下にI期開発の概要と今後の展開を記述する．開発は図7に示すシステム構成を基に進められている．リポジトリのエンジンにどのようなDBMSを採用すべきか，各構成要素間のインタフェースをどのように定めるか，どのような機能を装備すべきかが検討の重要な観点であった．

I期ではリポジトリ内へのデータ・アクセスI/F（登録・検索・変更・削除）機能を開発し，II期以降でリポジトリ・システムの運用回りの機能を開発する．そして，最終段階でシステム全体で整合のとれた，体系立てた操作性を提供する，という計画で開発プログラムが推移している．

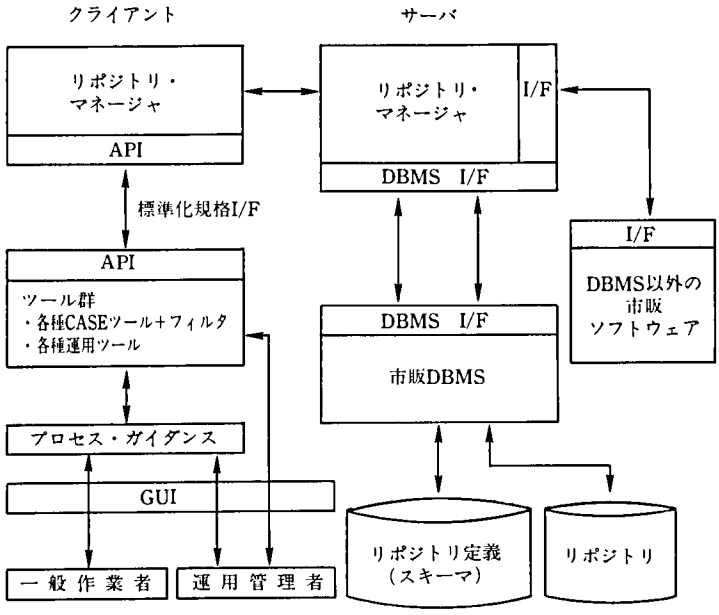


図 7　リポジトリ・システム構成図

Fig. 7　Structure of repository system

【図7の説明】

1)　本システムは，オープン環境を前提とするクライアント・サーバ方式である．

2)　本システムは，開発工程全域を支援する統合CASEシステムであり，利用者（一般作業者と運用管理者）とツールに対するユーザ・インタフェース（API，GUI，コマンド）を体系立てて提供する．

3) ユーザ・インタフェースは標準規格に準拠する.

4) 構成要素として市販ソフトウェアの依存度を高め，開発部分を極力少なくしたシステムである．I期の範囲であれば，APIは標準（ISO IRDS I/F）に準拠し，機能自体はリポジトリ・エンジンである市販DBMSの標準機能で対処する．リポジトリ・マネージャは標準ユーザ・インタフェースをDBMSの標準機能操作に変換する役割を担う．またII期以降で開発する項目が，DBMS以外の市販ソフトウェアを併用することで開発部分が軽くなるのであれば，その方法を採用する.

【I期の実績】

1) リポジトリのデータモデル（概念スキーマ）の設計……OMT表記法を用いてオブジェクト指向でイメージ案作りを行った.

2) 標準APIの採用……ISO IRDS, ANSI/IRDS II, ECMA PCTE, OMG CORBAの中から国際標準，JIS標準ということでISO IRDSに準拠することに決定した.

3) リポジトリ・エンジンの採用……開発の安全性および安定性を重視してRDBMSにするか，将来性でOODBMSにするかで慎重に検討を重ねた．その結果，データ・モデルとの親和性，関連（クラス間のリレーション）操作の優位性（ポインタによる高効率，RDBにおけるn：m表現での関連表管理の煩わしさの排除）からOODBMSでチャレンジすることに決定した.

4) リポジトリ・マネージャの開発……リポジトリ・マネージャは，ISO IRDS I/F（利用者要求）をOODBMS操作に変換し，制御をOODBMSに渡す．そして，OODBMSから渡された結果を利用者（プログラム）に返す役割を果たす．リポジトリ・マネージャは，リポジトリ定義（スキーマ）とは独立した形で開発されている．すなわちオブジェクトとなるクラス，関連の定義情報はすべて外からの入力であり，リポジトリ・マネージャはそれらを変数として処理する形態となっている.

5) プロセス・ガイダンスのプロトタイプ開発……ある部分の標準開発工程と開発方法論に準拠した形で，文書作成・管理ツールと自社開発の下流CASEツールB（デバッグ・ツール）を起動するプロトタイプを開発した.

6) 市販の上流CASEツールA，自社開発の下流CASEツールBに対するフィルタの開発……ツールAで設計したRDBのDB定義（スキーマ情報）をリポジトリに格納し，それをツールBにつなげるためにそれぞれのフィルタを開発した．Aのフィルタはリポジトリへの情報格納であり，Bのフィルタはリポジトリからの情報入手をAPIで行うものである.

7) 運用ツールの開発……I期の範囲での運用に必要な最低限度のツールとして，リポジトリ定義・管理ツールとEXPORT/IMPORTツールを開発した.

【II期，III期の展開】

以下に示す開発項目が候補として挙げられている.

主候補：① プロセス・ガイダンスの本格化

② CASEツール連結のためのメッセージ通信機能

③ 分散リポジトリ機能

④ セキュリティ機能

⑤ 運用ツールの充実（GUI 装備，新ツール開発，一貫性の保証，体系化）

準候補：① バージョン管理機能

② 参照制約機能

③ EXPORT 機能の拡張（CDIF 等の標準出力形式への準拠，および他の CASE ツールへの情報の一括受け渡し）

④ 効率改善

6. おわりに

今回，分散協調処理体系（ACCF : Advanced Cooperative Computing Framework）の開発に関連して，リポジトリ・システムについて検討する機会を得た．リポジトリについては，CASE ツールに密着したものとしては実用段階に入っているが，リポジトリ独自のシステムはまだ試行中という状況にある．

このような状況の中で，リポジトリ・システムにはどのような機能や，どのような情報が格納されるべきなのか，リポジトリ・システム構築にはどのような活動ステップが必要なのか，について一考察を提示したのが本稿である．なかでも重要と考えられるのは，なるべく多くの要件を充足することが可能な標準データモデルの設定と，そのリポジトリを前提に置いて，ユーザ要件を実現するツールの標準的な開発ガイドの作成である．

ガイド内容については，今回述べることができなかったが，標準データモデルについては，一部の要件に限ってではあるが，今後の検討のたたき台とできるものを設定できたものと考える．紹介した事例にもあるように，先進的ユーザではリポジトリ・システムを中核とした，情報システム構築の試行を開始しており，今後この分野についての標準化や，技術的発展に加速がつくことは必至であり，われわれもさらに研究を重ね，積極的に情報システム構築への適用を図ることが必要であると感じている．

参考文献 [1] 国友義久，“リポジトリの役割と仕組み”，インタフェース，1991, 8.
[2] 佐藤正美，“リポジトリ入門解説～技術体系と活用ガイド～”，ソフト・リサーチ・センター，1991.
[3] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, “Object-Oriented Modeling and Design”, Prentice Hall, 1991.
[4] 山田　進，“情報資源管理概論”，オーム社，1987.
[5] (社)情報サービス産業協会，“ソフトウェア技術の標準化に関する調査研究—CASE ツール評価の標準化—”，1991. 3.
[6] 板倉　教，“システム開発環境 IDES のリポジトリ”，UNISYS 技報，Vol. 12 No. 1, 1992. 5.
[7] 原　潔，“次世代データベースの展望”，UNISYS 技報，Vol. 12 No. 1, 1992. 5.
[8] 酒井博敬，堀内　一，“オブジェクト指向入門”，オーム社，1989.
[9] 堀内　一，“データ中心システム設計”，オーム社，1988.
[10] J. Martin, “Managing the Data-Base Environment”（坂本　弘，塗師省三訳，データベース環境の実現と管理—企業経営を成功に導く情報システムとは），日経マグロウヒル社，1983.
[11] W. Durell, “データ資源管理”（味村重臣監修，IRM 研究会訳），日経 BP 社，1987.

執筆者紹介 原 田 公 敬 (Kimihiro Harada)

1970年学習院大学法学部卒業．同年日本ユニシス(株)入社．シリーズ2200/1100 OSの保守業務を担当後，同じくシリーズ2200/1100の開発支援ツールの開発・保守業務に従事．現在，I&Cシステム本部MISAプロジェクト所属．

城 代 優 二 (Yuji Jodai)

1971年東京農工大学工学部卒業．同年日本ユニシス(株)入社．シリーズ2200/1100のデータベース・プロダクトの開発・保守業務に従事．現在，I&Cシステム本部MISAプロジェクト所属．同時に，オープンシステムサービス本部リポジトリ開発プロジェクトに参画し，リポジトリ・システム開発業務に従事．

既存システムをベースとするEUC環境の実現

Creating an EUC Environment Based on Existing Information Systems

松 木 規 子

要 約 EUC(End User Computing)という情報技術に取り組むためには，EUCの必要性とEUCがもたらす利益について正しく認識し，EUC適用が効果的な情報システム分野を明確にすることが必要である．

本稿では情報システムの対象領域を，事実としてのデータを取り扱うトランザクション処理の分野と，そこで発生したデータを管理・活用するマネジメント・サポート・システム(MSS)の分野に分類し，MSS分野をEUC適用領域と位置づけた．さらに，MSS分野の実現形態としてEUCによるデータ利用環境モデルを提案している．

モデルの実現については，ユーザの既存資産を有効活用するために上記の二つの要素が混在する従来の情報システムから，MSS分野を分離しモデルに即したデータ利用環境を構築することとし，そのための手順や考え方を，主にMSS分離の範囲，データ環境，ユーザ教育の観点から述べている．

Abstract In order to deal with the information technology called end user computing (EUC), it is necessary to clarify what areas of information systems benefit from EUC in addition to a need for a correct understanding of why EUC must be installed and its resultant advantages.

Through an attempt to classify the areas covered by information systems into two categories: transactions processing where data are handled as facts; and management support systems (MSS) which serve to monitor and manipulate the data provided in transactions processing, this paper has positioned MSS as the area where EUC can be applied, and also proposes an MSS environment model which allows EUC to use the data as it implemented form.

Based on the conception that, to make the best use of existing user computing resources and to build a favorable data utilization environment which conforms to such a model, the creation of the model requires the separation of the MSS segment from conventional systems where the two foregone elements are intermixed, this paper also discusses conceptual basics as well as the process flow in the light of the separable MSS range and data environments as well as end-user education.

1. はじめに

社会情勢の激変，世界経済の低迷の中，情報システムへの投資の肥大に対する反省から，その効果に対して以前にも増して厳しい目が注がれるようになってきている．

このような中で，複雑化するユーザ・ニーズを的確に実現するための手段として，エンドユーザ・コンピューティング（以下EUCと省略）に注目が集まっているが，適用方針の曖昧さや技術的環境の未整備，適用分野の不明確さから，十分な効果を上げるに至っていない例も多い．また，従来稼働している情報システムとEUCの関係についても明快な説明が不足しているように思われる．

本稿では，EUCの必要性とEUCがもたらす利益について確認するとともに，資産

としての既存情報システムを最大限有効に活用しながら，新しいユーザ・ニーズに対応可能な情報システムの形態へ暫時移行していくための考え方と手順を提案する．

2. EUCの定義

EUCという言葉は，1980年代半ばから頻繁に使用されるようになったが，一般に明確な定義は存在せず，人によってさまざまな言い方がなされてきた[1][2]．

この後，本稿を進めるに当たって，「EUCとは何か」ということについて認識を統一しておくことは重要なことである．したがって，本稿におけるEUCを定義し，その目的を明らかにするところから始めたいと思う．

EUCとは，“エンドユーザ自身が自発的・積極的にコンピュータを利用し，各種のデータから自分の仕事にとって必要な情報を創り出し，業務に役立てるための情報技術”である．

EUCの目的は，単にユーザ自身がコンピュータによって問題解決を行うことで情報システム部門の抱えるバックログの削減に貢献するといったことよりは，ユーザによる積極的な業務改善や創意工夫を引き出すことで企業の競争力を強化することにある．

3. EUCの必要性

では，なぜ今EUCが関心を集めるのであろうか．企業の情報処理の現状を見ると，効率向上，省力化のためのライン業務の機械化は一段落したといえる．また，社会的には，大量生産，大量消費の時代は終わり，多品種少量生産，多様化の時代に移行した．企業は多様化した社会に対応し，市場に提供する“もの”の付加価値を高めることで他社と差別化をはからなくてはいけなくなってきている．

そのためには，市場のニーズの動向や変化をいち早くつかみ，さらに的確に予測していかなければならない．このような状況下では，従来のように特定の人の経験と勘に頼るのではなく，データに基づき仕事の精度をあげることが必要である．それには，以下の事柄が重要なポイントとなる．

- 個人の創意工夫，独創性，創造性を発揮できるようにする．
- 個人の経験やノウハウを組織のものとして蓄積・利用するようにする．

このことは，中央集権的に業務の流れを標準化し利用者を縛りつける従来型の情報システムや，単に個人の仕事の効率を向上させるOAでなく，個人を支援し，個人個人がグループの一員として協調しあって効果的に仕事の精度をあげるEUCが必要とされていることに他ならない．

4. EUC環境実現の考え方

4.1 EUC適用分野の明確化

EUCとは，前述のようにエンドユーザのための情報技術の一つである．したがって，この技術が効果的に適用され得る情報処理環境をEUC環境と呼ぶことにする．

EUC環境実現のためには，まず情報システム全体の中におけるEUCの適用分野を明確にする必要がある．

ASDF*ではソリューションの領域を次の二つに分類して考える．

① トランザクション処理

現実の『人・もの・金』等の状態や変化を客観的に再現するもの

② マネジメント・サポート・システム（以下MSSと省略）

現実の『人・もの・金』等の状態や変化を管理し，その中から業務に役立つ情報を主観を通じて読みとるもの

MSSの分野では，データの見方・必要とされる情報は，担当している仕事の種類，担当者，時と場合等によって頻繁に変化する．このような処理をあらかじめアプリケーションとして作り込むには莫大な量のプログラムを開発しなくてはならないし，頻繁に改修しなくてはすぐ使い物にならなくなってしまう．また，MSS分野の処理は，ユーザの操作によって発生した情報が新たな行動や新たな情報に対する要求を誘発することが多い．この新たな処理はあらかじめ定義することすら不可能である．

このような問題は，ユーザ自身が，その都度，その場で必要な情報を創り出していくことで解決することができる．したがって，エンドユーザ自身のコンピュータ活用による業務改革を目的とするEUCの適用分野は，MSSの分野であるということができる．

情報システム全体をトランザクション処理とMSSの二つのシステム分野として捉えるという考え方は，従来の情報システムの構造を根底から覆そうというものではない．従来の情報システムもこれら二つの要素を含んで成り立っており，単に明確に分離して認識されていなかっただけである．また，MSS分野の処理の中には，従来情報システム化されなかったものや，パソコン等で実施されていてホスト中心の従来の情報システムとは分離されていた処理等もある．

これらを整理することで，現状の情報システムを土台に，よりユーザの要求に柔軟に対応可能な包括的な情報システムへと暫時移行してくことが可能である．

4.2 EUC環境モデルとツール

では，具体的にどのようなシステム形態に移行していけば良いのであろうか．

EUC環境をモデル化したものが図1である．

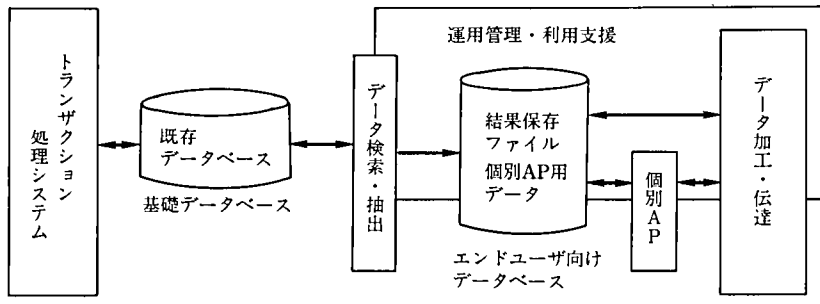


図1 EUC環境モデル

Fig. 1 EUC environment model

* Advanced Solution Development Framework の略．UA（Unisys Architecture）に基づき，統合CASEとエンドユーザ・コンピューティングを基軸としたソリューション構築のためのツール，サービスおよび方法論を統合した体系．

「トランザクション処理システム」とは，既存の情報システムからMSS分野の処理を除いたものであり，その対象とするデータベース（既存データベース）はMSSの分野で仕事をするユーザから見ると，EUCの基盤となるデータベースである．

ここでは，EUCの基礎データベースと呼ぶ．基礎データベースはあくまでも業務処理のためのものであり，EUCサイドからこれを直接更新することは望ましくないので，データはここから検索・抽出され，利用される．いったん基礎データベースから抽出されたデータは，そのまま使い捨てられることもあるし，必要に応じて保存・蓄積されることもある．したがって，ユーザが自由にデータを保存したり，変更したりすることができる保存領域が必要になる．ここでは，これをエンドユーザ向けデータベースと呼ぶ．

この環境を構築するために必要となる機能要件には以下のものがある．

① EUC環境下では，ユーザの要求に応じて，必要なデータを随時検索・参照できなければならない．

② 検索・参照されたデータは，ユーザの要求に応じて柔軟に加工（集計，編集等），効果的に伝達（グラフ化，電子メール等）できなければならない．

③ 検索・参照されたデータは，必要に応じて保存され，適切に管理されなければならない．

④ EUCは単に個人の作業を支援するだけでなく，個人個人がグループの一員として協調して仕事を進めることを支援する．したがって，複数の人が同時に同一データを利用して仕事ができなければならない．

⑤ MSS分野のソリューション・システムのユーザには，各業務担当者やその管理者，上級管理職から経営者層まで含まれる．これらの人々が行うMSS分野の仕事には，定型的な処理も存在するし，経営者層を支援する特別なアプリケーションが必要になる場合もある．したがって，検索したデータを定型的あるいは高度に加工するアプリケーションを簡易に開発できなければならない．

⑥ 従来の情報システムが稼働しているハードウェア・プラットフォームと，MSS分野の処理を実行するハードウェア・プラットフォームが同一とは限らない．各システムが最適のプラットフォームで稼働し，かつ有機的に結合していなければならない．

弊社では，これらの機能要件を実現するベースとなるツールとしてMAPPERを提供している．MAPPERとは，キャビネット構造のデータベースと豊富なレポート操作命令を最大の特徴とする，データベース・マネジメントからユーザ・インタフェースまで一元的に提供しているエンドユーザ向けの4GLである．とくに，エンドユーザ向けデータベースとしてのMAPPERデータベースは，その可視性の高さ，直感的理解の容易さ，運用・管理の簡便さにおいて優れたものである．さらに，近年EUCの要請に応えるべく，ネットワーク機能やデータベース・インタフェース機能，ユーザ・インタフェースを強化している．上記の機能要件に対し，MAPPERがどのように対応するかについては表1を参照されたい．

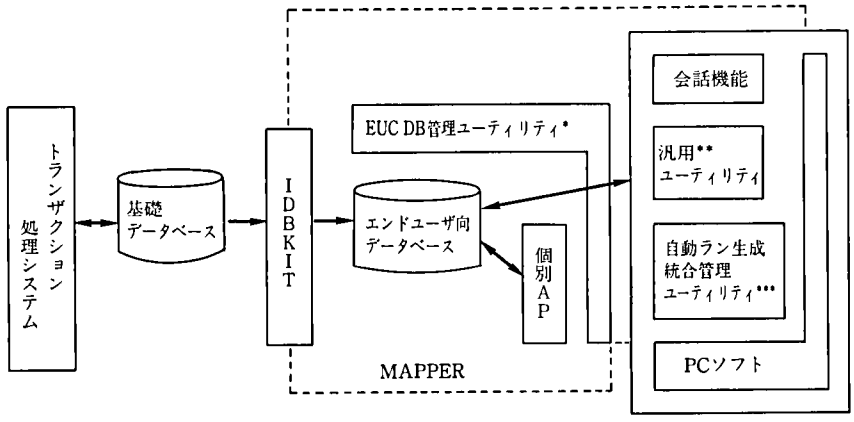
データ検索・抽出の機能には，対象となるデータベースの種類によって，またそれぞれの機能が配置されるプラットフォームに応じて，ツールの選択に幅があるが，

表1　EUC環境の機能要件とMAPPER

Table 1　Functional requirements and MAPPER for an EUC environment

EUC環境の機能要件	MAPPERの機能
基礎データベースからのデータ検索・抽出	リレーショナル・データベースからはMRI，その他のファイルからは外部ファイル・インタフェース機能を使って仕組みを作ることができる．
データの集計・編集・伝達	計算(TOT，CAL)，分類(SORT)，照合(MA)等，豊富な会話命令を持つ． 効果的伝達のためにはグラフ機能，イメージ機能，電子メール機能等がある． また，PCソフトで加工するためにデータを引き渡す機能もある．
データの保存・管理	MAPPERの基本的な命令でも対応可能だが，より簡便に行うためのユーティリティがある．
同一データに対する複数端末からの同時操作	レポート・ロック機能，排他制御機能がある．
簡易なアプリケーション開発	会話操作をカタログ化する自動ラン生成機能，条件判定や複雑なデータ操作を可能にするラン機能，データ辞書を中心とした開発支援ツールがある．
最適プラットフォームでの稼働・各プラットフォーム間の有機的結合	インフォメーションハブ，サーバ，ワークステーションでMAPPERが稼働し，それぞれがネットワーク機能を持つ．

IDBKIT*やRDIP**等も有効なツールの一つである．　また，データの加工機能としては，MAPPERだけでなく，ユーザが日頃使い慣れているパソコン上のソフトウェア



＊エンドユーザ向けデータベースに格納するレポートに名称や使用期限を設定し管理するツール．
＊＊MAPPERの会話機能だけでは実現が困難だったり，操作が煩わしかったりする処理を可能にするラン
（例：縦横変換，二次元編集，簡易フォーマット変換，重複項目表示制御等）．
＊＊＊ユーザの会話操作を自動的にカタログしランを作成したり，そのランを再実行したり，レポートと同様に名称や使用期限を設定し管理するツール．

図2　EUC環境とツール

Fig. 2　EUC environment and tools

＊　従来の情報システムで使用されているデータベースやファイルから，項目，条件等を指定することで，必要なデータをMAPPER内に取り込みユーザに提供するためのツール．

＊＊　情報系システムのフレームワーク(しくみ)と情報系システムを効率的に構築/維持/運営支援するためのソフトウェア群．

(たとえばLotus1-2-3*やExcel**等) も対象ツールと考える.

EUC環境モデルに, 現在弊社として提供可能なツールを当てはめた例を図2に示す.

5. EUC環境実現のポイント

本章では, ユーザの既存システムを土台とするEUC環境を実現するには, どのような手順で進め, 何に着目すれば良いか考察する.

5.1 EUC環境実現の手順

図3にEUC環境実現の手順を示す. ここでは, 各手順の中で実施される個々の作業までは定義していない.

1) 対象業務の絞り込み……EUCを全社一斉に平均的に展開するのは無理であり, 無意味である. EUCの必要性の高い部署, 効果の大きい部署に対して実施し, 社内の成功事例を増やしながら拡大させることを考える.

2) 従来の情報システムからのMSS分離……EUCの対象となった業務に関連する情報システムを持たない部署というのはほとんどないと思われる. 既存のシステムを最大限有効に活用するために, 対応する情報システムの分析を行う.

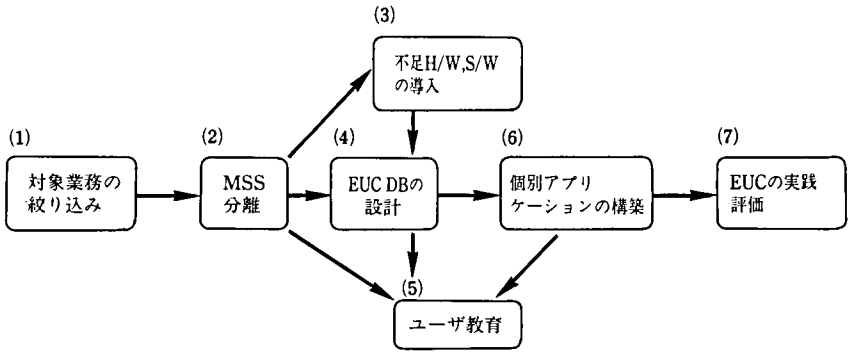


図3 EUC環境実現の手順

Fig. 3 A process flow for creation of an EUC environment

3) EUC環境構築のためのハードウェア, ソフトウェアの導入……EUC環境として必要なものの中から, 不足しているハード, ソフトを洗い出し準備する. ここでも, 既存の資産を有効に活用することを考える.

4) エンドユーザ向けデータベースの設計・構築……既存システムが出力している情報を元に, エンドユーザにとって, EUC活用のきっかけとなるようなデータベースを構築する.

5) ユーザ教育……EUC環境下で, エンドユーザができるだけ独立して活動できるような教育を実施する.

6) 個別アプリケーションの構築……MSS分野の定型処理を担うアプリケーショ

* Lotus1-2-3は米国ロータスディベロップメント社の登録商標である.

** Excelは, 米国マイクロソフト社の商標である.

ンや，経営者層を支援する特別なアプリケーションを構築する．開発者は必ずしもエンドユーザとは限らないが，エンドユーザが中心になって行うことが必須である．

7)　EUCの実践・評価……実際の業務へのEUC適用の推進とその効果の把握，評価を行う．

5.2　従来の情報システムからのMSS分離

既存の情報システムからMSS分野を探し出し，EUC環境へ移行する例を挙げる．

例1は，社員の担当業務の各作業の工数と進捗を管理するためのもの（図4）でまず，個人ごとの作業が定義され，予定工数が投入される．社員はこの情報をもとに，作業予定を立て，実作業にとりかかる．その後，作業にかかった工数と進捗状況を投入し，これをもって作業実績を管理する．

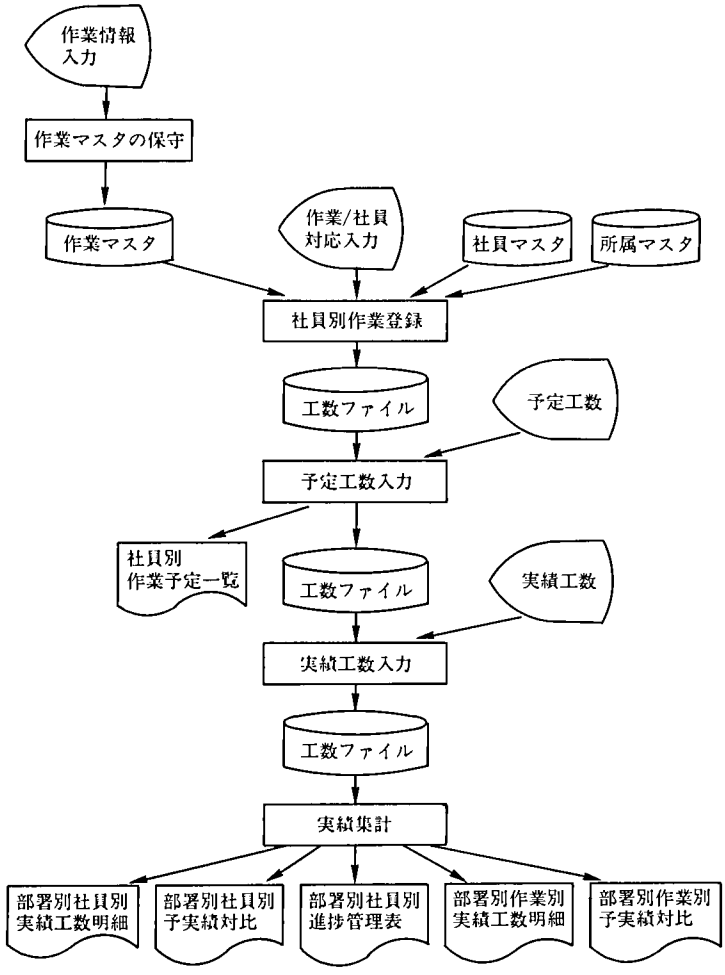


図4　例1　実績工数管理システム

Fig. 4　Sample 1—Performance management system

従来の情報システムでは，利用者（作業担当者やその上司）に対して帳票の形で情報が提供される．もし，部門をまたがるプロジェクト・チームの新設等により管理の

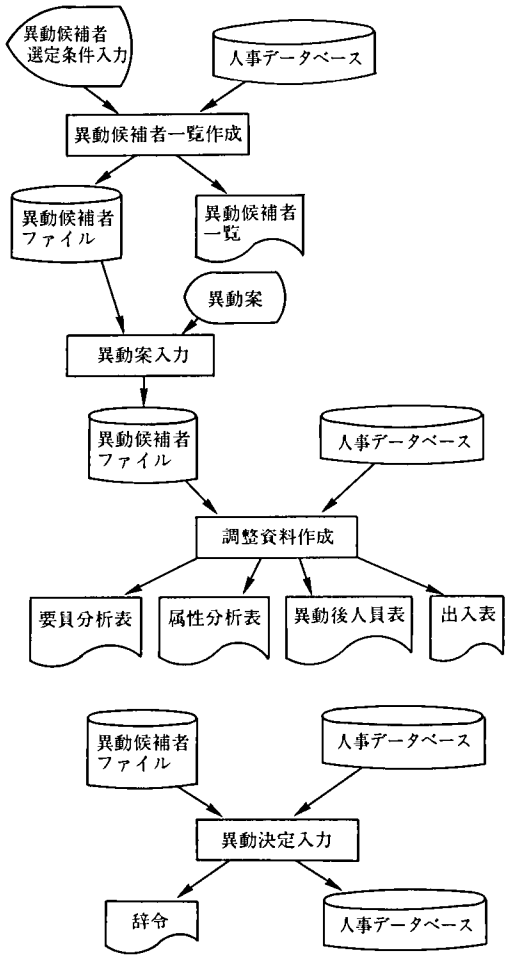


図 5　例 2　人事異動サブシステム

Fig. 5　Sample 2—Personnel change record sub-system

対象範囲が従来より広がったり，集約のキーが異なったりすると，出力情報はそのままでは役に立たなくなってしまう．この帳票出力の部分を切り離して EUC 環境下に埋め込むことで，ユーザの要求に柔軟に対応できるようになる．

例 2 は，人事異動サブシステムである(図 5)．異動システムも，例 1 と同様にたくさんの帳票を出力している．これらの帳票は，異動決定のための各種の調整作業で参照されるもので，異動の要件や，候補者の属性によって参照の仕方は変化する．また，候補者を選定する時の条件も固定ではありえない．このように，その時の要求によって内容や手順が変化する部分は，切り離すべき部分である．

同じ出力帳票であっても，辞令等は単に異動結果という事実を通知するものなので，MSS には含まない．

従来の情報システムから，どの部分を MSS として切り離せば良いかを判断する上で，着目すべきポイントとしては，以下のことがあげられる．

① 何らかのキーで集約した情報を出力している．

② ユーザが出力情報を入力して，さらに加工している．

③ 過去に出力の形式や内容について変更要求が出されたことがある．

5.3 エンドユーザ向けデータベースの考え方

EUC環境モデル（図1）には，基礎データベースとエンドユーザ向けデータベースの2種がある．基礎データベース内のデータは，業務活動によって発生する事実を表したものであり，エンドユーザ向けデータベース内のデータは，事実としてのデータに対し，いったんユーザの視点を通すことでデータの性格が変化したものである．

基礎データベースは，データベース・システムのアーキテクチャとアプリケーション・システムの機能によってその構造が決まっており，必ずしもエンドユーザにとって理解しやすい構造とは限らない．したがって，ここからデータを抽出してくる際には，ユーザにはできる限りその構造を意識させないようにする必要がある．

EUC環境下では，ユーザは何らかのデータが必要になった時点で，データ検索・抽出機能を利用してデータを入手し，必要に応じてそれを加工・保存する．このことから，EUC環境構築時には，エンドユーザ向けデータベースは空の状態で良いということになる．しかし，多くのユーザは，実際に何らかの“もの”を目にすることで，新たな要求を生み出したり，すでに持っていた要求をより洗練させることが可能になる．そこで，エンドユーザ向けデータベースには何らかの初期データが設定されることが望ましいといえる．これに対して，5.2節で述べた従来の情報システムから切り離した帳票類を構成するデータが，ユーザにとってトリガーとなる．

したがって，既存システムの帳票から，エンドユーザ向けデータベースの初期設計を行う．このデータベース設計は，あくまでもユーザの要求を顕在化させるための元となる情報を提供することが目的であるので，ユーザのすべてのニーズを洗い出したり，それを実現できるデータベースを考えたりする必要はない．

データベース設計は以下の要領で行う（具体例は図6参照）．

① 一つの帳票を構成する項目を洗い出す．

② 各項目をその元となるデータと対応付ける．

③ データから帳票項目への加工方法を定義する．

④ すべての帳票に対して上記の作業を行う．

⑤ 帳票をグループ化する．グループ化する際には，同一データ項目を利用しているということだけでなく，帳票利用者の範囲やデータの有効期限等にも注意する．

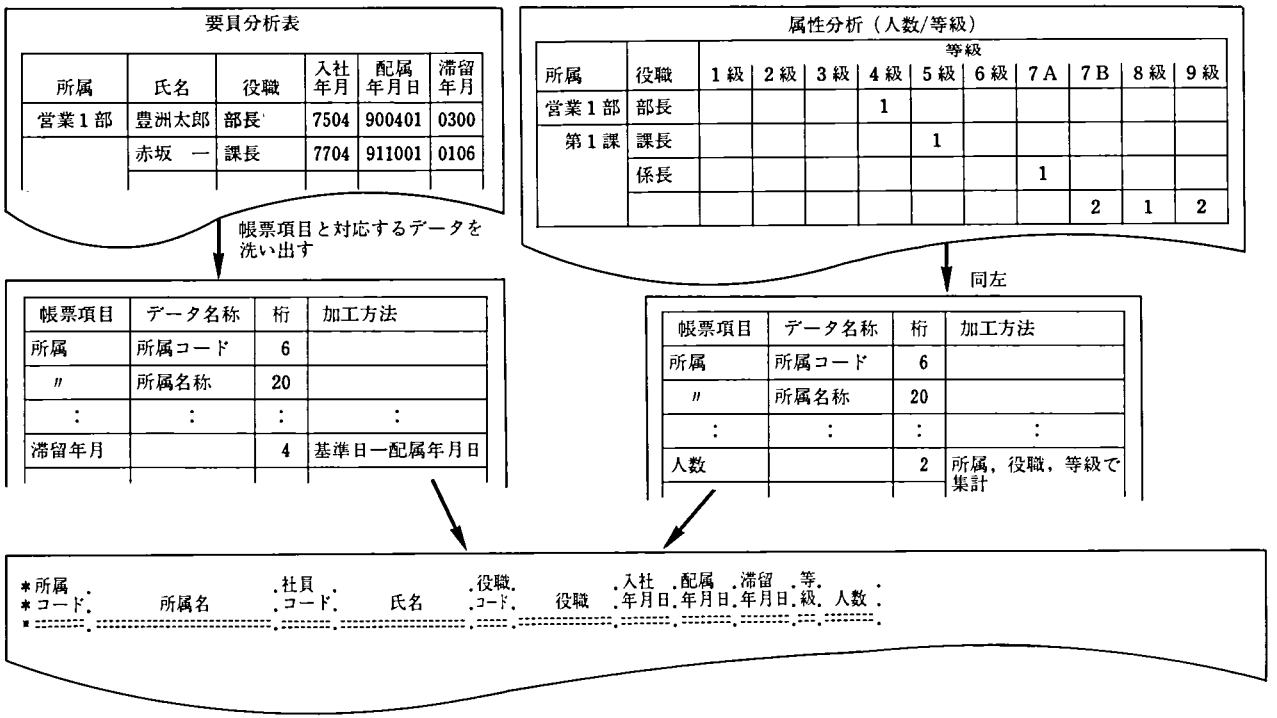
⑥ グループ化した帳票に対して，できるだけ共通に利用できるようにレポートを設計する．ただし，あまり操作が複雑になるような場合は，共通利用にこだわらなくても良い．

⑦ レポートごとの操作手続きを設計することで，データベースの設計を検証する．

5.4 ユーザ教育とデータ活用

EUCの適用というのは，環境が整えばそれで良いということではない．データがユーザの業務に活用されてはじめて意味があるのである．

ユーザによるデータ活用を可能にするためにはユーザ教育が重要である．これを次



レポート操作手順

①滞留年月を計算(CAL)
②所属コード，役職コード，入社年月で分類(SORT)
③表示項目の絞り込み(VIEW)

①所属コードで抽出(S)
②人類欄に1を充填(TOT)
③所属コード，役職コード，等級で分類(SORT)
④所属コード，役職コード，等級をキーに人数集計(TOT)
⑤所属，役職を縦軸，等級を横軸に二次元編集(XYRPT)

図 6　エンドユーザ向けデータベース設計例

Fig. 6　A sample database design for end-users

の点から考える．

・教育の対象（EUCにとってのエンドユーザとは誰なのか）

・教育の種類（エンドユーザのレベルや必要度に合わせた品ぞろえ）

5.4.1 教育の対象

現在の企業で，情報を利用せずに仕事をしている人は皆無といって良いであろう．したがって，企業で働くものは皆エンドユーザであるといえる．しかし，EUCの適用効果はすべての人にとって一律ではありえない．また，社員全員に一斉に教育を実施して，全員が等しくその効果をあげることも考えにくいことである．

したがって，教育の範囲は，当初必要性が高く効果の大きいと思われる部署に対して行い，徐々に拡大していくことを考える．また，一つの部署に関しても同様に，その部署の構成員全員に対して一律に行うのでなく，部署内で中核となって推進していける人材をキーパーソンに据え，重点的に教育し，その後はキーパーソンを中心に利用者を広げていくことを考える．

5.4.2 教育の種類

エンドユーザの数が多ければ，それだけ情報の利用方法も異なり，さまざまなレベルの技術が要求されることになる．ユーザの要請に応えるためにどのような教育体系を持てば良いかは企業ごとに異なるが，一般的には以下のものが考えられる．

・与えられたツールの機能を理解し，操作できるようになるための教育（操作教育）

・どのようなデータが利用できるのか理解するための教育（データ教育）

・EUC環境下で実際にどのような問題に対応できるのか理解するための教育（応用操作教育）

・EUC環境下で新たな問題を発見し，業務を改善できるようになるための教育（業務改善教育）

1) 操作教育のポイント

・操作方法そのものよりも，何ができるのかという機能や何のためのものかという役割を理解できるような教育を実施する．

・ユーザが楽しく操作を覚えられるように，実務に即したデータや遊び心のあるデータを準備する．

・いつでも一人でも学習できるような環境やツールを準備する．

2) データ教育のポイント

・操作教育と並行して行う．

・基礎データベースの項目について，データの意味・内容・更新のタイミング等をユーザにわかりやすく説明する．

・エンドユーザ向けデータベースのレポートについて，その位置づけや役割，基礎データベースとの関係等について説明する．

3) 応用操作数育のポイント

・ユーザが興味をもって実施できるよう，また職場ですぐ利用できるよう，実情に即した課題を用意する．

・課題は，簡単なものから徐々に複雑なものになるように数多く設定しておく．

・一つの技術を反復利用し十分習得できるような課題を設定する.
・ユーザが飽きたりだれたりしないように，一つの技術は連続しないように課題を設定する.

4) 業務改善教育のポイント
・業務改善を模擬体験するロールプレーイング中心の演習を行う.
・業務改善の結果,EUC 環境下で独自のアプリケーションが必要になったとき，それを構築できるような教育も場合によっては実施する.

5.5 個別アプリケーションの構築

MSS 分野の個別アプリケーション構築は必ずエンドユーザ中心で行うが，開発作業そのものはエンドユーザ自身が実施するとは限らない．エンドユーザが開発まで行う場合には，業務改善教育の一環としてシステム設計，プログラミングの教育を行う必要がある.

開発作業の担当が誰であっても，以下の点に留意すべきである.

① 処理要求に変更が生じた時には，いったん開発したプログラムを保守するのではなく，変更になった部分だけ切り捨て新しいものを作成する.

② そのためにも，個々のプログラムはできる限り単機能で，切り捨てても惜しくないくらい小さくシンプルなものにする.

6. まとめと課題

これまで EUC 適用領域を明確化し，それが実現される環境を見てきた.

MSS の分野では,EUC という技術を適用することによって,従来の情報システムで考えられていたより広い範囲にコンピュータが適用でき,それが仕事の質を高めたり，仕事そのものやその流れを変えていくことになる.

これに伴い，ユーザが必要とするデータそのものの種類や量が変化してくる．EUC 環境下のユーザに対して，必要とするデータを生み出し提供するのは，トランザクション処理の分野を担当する情報システムである．データが変化するということは，この情報システムにも何らかの影響を及ぼすということである.

したがって,EUC が効果的に適用されるということは,情報システム全体に対して，それが真に業務に役立つものとして機能するような再構築を推進する原動力となるものであるといえる.

最後に，この EUC 適用に関して，本稿で述べることができなかったことを挙げておきたい．今後，実際に企業における EUC 環境構築を実施する際には，これらのことが重要なテーマになると思われる.

・EUC のユーザ適用を支援する方法論の提供
・各種プラットフォームへのデータベース分散，機能分散に対応する方法論およびツールの提供

また，本稿では既存システムの資産価値を損なわずに EUC 適用を推進することを述べたが,EUC 推進にあたって現状の情報システムにまったく問題がないというわけではない．たとえば，既存の情報システムでは取り扱わないデータ項目が MSS 分野で必要になったり，基礎データベース上のデータがすでに何らかの集約データになって

おり，MSS分野のユーザが1件1件の明細データを確認することができなかったりする．これらのことから，今後以下の事柄について，一般論でない個々の状況に即した検討や判断が必要になると思われる．

① トランザクション処理のソリューションにとっての業務データベースと，MSSのソリューションにとっての基礎データベースは同一のものか．そのデータベースにはどのような種類のデータが，どのような形で存在すれば良いのか．

② EUCの推進は，トランザクション処理とMSSの両分野のソリューションの変化を促進する．この変化を吸収することができる，変化に強い情報システムの全体像――つまり，EUCを含む今後の情報システム全体のあるべき姿とはいかなるものか．

日本では，EUCの真の必要性，有効性は理解され始めたばかりであり，エンドユーザにとっても，情報処理技術者にとっても，挑戦のしがいのある分野である．今後の発展に期待するとともに，微力ながら力を尽くしたいと考えている．

参考文献　［1］ 一瀬益夫，“エンドユーザ・コンピューティングの役割とその管理について”，経営情報学会誌，Vo 11.1990.

［2］ N. Meyer, M. Boone, “情報優位の企業戦略”，TBSブリタニカ．

執筆者紹介　松　木　規　子（Noriko Matsuki）

1956年生．80年早稲田大学第一文学部卒業．同年日本ユニシス(株)入社．81年10月よりMAPPERシステム適用サポートに従事．現在 システム技術本部CASE/4GL技術部 技術推進課に所属．

統合CASE LINCとLINCオブジェクト・モデリング

The LINC Environment (Integrated CASE LINC) and LINC Object Modeling

佐伯克己

要　約　統合CASE LINC（The LINC Environment）は，ユニシスが提供する情報システム構築環境の体系であるASDF（Advanced Solution Development Framework）の中で，迅速なソリューション構築を実現するLINCによる統合CASE環境として位置づけられている．従来のLINC IIは下流工程の支援ツールとして位置づけられ，さらに開発方法論であるLINCシステムズ・アプローチ，上流工程を支援するLINC-DA，テスト/デバッグ作業を支援するLINC-TEが新たに加わって総合的な体系になっている．

LINCシステムズ・アプローチはLINCの特徴を活かして情報システムの開発を行うためのガイドラインであり，その中核技法であるLINCオブジェクト・モデリングは，ビジネスの世界で行われている業務活動を調査・分析しながら情報システムのモデルを作り上げていく技法である．LINCオブジェクト・モデリングでは，個々の業務活動をイベント，コンポーネント等のLINCオブジェクトを用いてプロトタイプしながらモデル化する．モデルをさまざまな視点から分析し全体の整合性を検討しながらモデルを成長させていく作業手順となっている．この作業をワークステーション上のGUI環境の中で支援するCASEツールがLINC-DAである．LINC-DAは，LINCオブジェクト・モデリングを支援すると同時に，システム規模の見積り等のシステム開発の管理に関する情報も提供し，より効果的なシステム開発を支援する．

Abstract　In the Unisys Advanced Solution Development Framework (ASDF), the architecture supportive of an environment for building information systems, the LINC Environment (integrated CASE LINC) is positioned as providing an integrated LINC-based CASE environment which makes available rapid solution development tools and methodologies. The LINC Environment is a total combination of conventional LINC II (now placed as a lower CASE tool) and new additions such as LINC Systems Approach (which defines systems development methodologies), LINC-DA (as an upper CASE tool) and LINC-TE (which serves testing and debugging purposes).

LINC Systems Approach is the guidelines for building information systems through the full use of LINC features. LINC object modeling, the core technique advocated in the guidelines, is so designed as to enable users to create models for information systems while allowing them to monitor and analyze business activities being conducted on the operational front. In other words, LINC object modeling helps build models of individual business activities, using both a prototyping technique and LINC objects such as EVENT, COMPONENT, and so forth, so that those activities can be translated into the models for LINC information systems. All this process analizes models from a variety of viewpoints while watching a system's integrity till models grow up to be mature. It is LINC-DA that serves as a CASE tool which supports this task in a GUI-loaded workstation environment. Besides supporting LINC object modeling, LINC-DA also assists in building systems in a more effective way by providing information related to project management including the estimated size of a whole system to be developed.

1. はじめに

ビジネスの世界で伝票や元帳，台帳等を使って行われている作業をコンピュータのリソースを使って支援するのが情報システムである．情報システムの設計・開発とは，ビジネスの世界の振る舞いを分析しながらシステムの姿を，そのプラットフォーム・プロダクトや適用技術の特性を考慮しつつ実現する仕組みを作っていく作業である．

この作業にはコンピュータ処理に関する高度で専門的な知識が要求されるため，利用者はこの内容を理解することがむずかしく，開発がかなり進んだ後にレビュアとして参加することが多かった．しかし，利用者がこの段階で情報システムに対して不都合な点や新たな要求に気づいても，一般にシステム開発の後工程になるほど変更には多大なコストを要するため，その対応は困難なことが多い．また，情報システムの設計・開発の工程が複雑であるほど生産性や保守性が低下し，ビジネスの世界の環境の変化にも対応しにくいため，情報システムの陳腐化を招くという問題があった．

LINC はビジネスの世界で行われている業務活動を投影してモデル化を行い，そのまま情報システムとして実現するという自動化のメカニズムを持ったプロダクトである．LINC を用いることにより，利用者と共に業務活動を分析しながらそのモデルを作成し，そのまま情報システムとして構築していくことができる．このため，利用者の業務知識を活用しその要求を満たした情報システムを開発することができ，またビジネスの世界の環境の変化にも容易に対応できるはずである．

ところで，LINC は日本で出荷されて以来 11 年を経過し 800 システム以上の出荷実績を持つプロダクトであるが，その利用のされ方は必ずしも本来の考え方を活かしたものになっていないのが現状である．その主な原因は，開発者自身が 3 GL 等の従来使用してきたツールで行うことと同様のことを単にツールとして LINC を使って実現しようとすることにあると思われる．このため，LINC の持つシステム記述の管理機能や，一部の業務機能が自動化されることによる効果は得られるものの，本来の効果を十分に発揮できないでいる．

このような問題に対して包括的な解決策を提供するのが統合 CASE LINC である．

2. 統合 CASE LINC

2.1 統合 CASE LINC の構成

統合 CASE LINC は，各種のツール群と開発方法論により構成されている．図 1 は統合 CASE LINC の構成を示したものである．

2.2 統合 CASE LINC のツール群

統合 CASE LINC のツールは次の三つにより構成されている．

1) LINC-DA (LINC Design Assistant) ……上流 CASE ツールであり，情報システムの分析・設計段階であるシステム調査/システム定義のフェーズで使用する．さらに，システム評価フェーズの作業をデザイン監査機能により支援する．

2) LINC II (Logic & Information Network Compiler) ……下流 CASE ツールであり，システム開発のフェーズで使用する．LINC リポジトリにより開発情報を一元管理し，システム全体の整合性を保証しながら情報システムを生成するツールである．

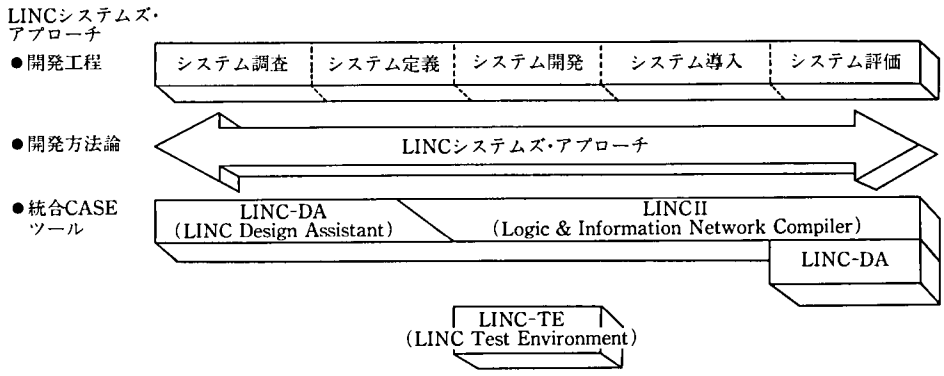


図 1　統合 CASE LINC の構成

Fig. 1　Composition of the LINC environment

3)　LINC-TE（LINC Test Environment）……システム開発のフェーズで行うテスト/デバッグ作業を支援するツールである．LINC リポジトリに登録されているシステム仕様をインタープリティブに実行することができ，オンラインデバッグ機能によりエラーの発見/修正を支援する．さらに，データベースの I/O 回数や CPU 使用率に関する統計情報を提供することにより，設計の最適化やコードレビュー作業を支援する．

2.3　開発方法論 LINC システムズ・アプローチ

LINC システムズ・アプローチは，統合 CASE LINC を効果的に用いるための開発方法論であり，次のような特徴を持つ．

1)　開発作業のガイドラインを提示し作業工程の標準化/品質の向上を支援

システム開発工程を五つのフェーズより構成し，各フェーズにおける「目標，作業手順，成果物」を規程する．LINC システムズ・アプローチの開発工程を次に示す．

フェーズ 1：システム調査

対象とするビジネス領域の業務目標と業務活動の調査/分析を行い，システム化の優先順位に従って情報システムの範囲と目標を決定する．

—主な成果物：業務活動分析表，情報システム概要書

フェーズ 2：システム定義

フェーズ 1 で明らかにされた業務活動を LINC オブジェクトを用いてモデル化し，LINC 情報システムのモデルを作成する．

—主な成果物：LINC オブジェクト・モデル

フェーズ 3：システム開発

LINC II を用いて情報システムの開発/構築を行う．

—主な成果物：導入用 LINC 情報システム

フェーズ 4：システム導入

LINC 情報システムを本番稼働環境へ導入する．

—主な成果物：本稼働用 LINC 情報システム

フェーズ 5：システム評価

稼働中の LINC 情報システムに対する改善点および機能強化点を明確化し，次回のライフサイクルを開始するための準備を行う．

—主な成果物：システム評価報告書

2) 高品質な情報システムの開発を支援

ビジネス指向の開発アプローチであり，ビジネスの世界における個々の業務活動をそのまま LINC の構成要素（LINC オブジェクト）を用いてモデル化する．作成されたモデルは業務活動と LINC 情報システムの両方を表現したものである．情報システムの分析/設計/開発を一貫して同一のモデルで行うことにより，システム開発の生産性および品質の向上を図ることができる．

3) 進化型アプローチによる開発リスクの削減

進化型プロトタイピングを取り入れたモデル化技法/開発工程により開発リスクを削減する．LINC オブジェクト・モデリングは，画面/帳票イメージ，モデル図等を用いて情報システムの姿をプロトタイプしつつ，モデルを作り上げていく技法である．これにより，システム開発の早い段階で情報システムの機能についての確認/検証を行うことができる．

4) 利用者の要求に合った情報システムの開発を支援

ビジネスの世界の業務活動をそのまま LINC の構成要素を用いてモデル化できるため，利用者にとってもわかりやすく，情報システム開発の全工程にわたって利用者の参加が可能である．これにより次のような効果を実現できる．

- 文書作成や開発各フェーズごとの終了手続きを簡素化できる．
- 機能性の要求に関するコミュニケーションを密にできる．
- 短期間にトレーニングを行える．
- 機能テストをより徹底して行える．
- 利用者側に自分達のシステムだという自覚が高まる．
- 利用者にとって親しみやすいシステムができる．

3. LINC の考え方

LINC ではビジネスの世界を「業務活動」と「業務活動を支援する情報」，および「情報を活用するための視点」という三つの要素で捉える．図 2 に LINC が捉えるビジネスの世界とそれを実現する LINC 上の要素の関係を示す．

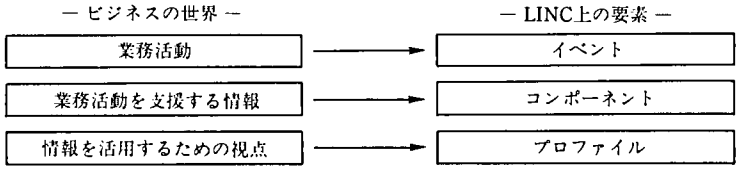


図 2　LINC の構成要素

Fig. 2　Constituent element of LINC

LINCのイベント，コンポーネント，プロファイルは次のような要素である．

1) イベント……「注文の受付」や「商品の販売」等の，ビジネスの世界の業務活動を扱う要素であり，

—活動を行うためのユーザインタフェース

・活動処理画面，キーボード操作機能

—活動を行うための業務手続き

・活動処理ロジックの定義

—活動の結果発生した情報

・活動データの定義

をカプセル化した構造を持っている．

2) コンポーネント……「顧客情報」や「商品情報」等のような資源情報（業務活動を支援する情報）を扱う要素であり，

—資源情報を維持管理するためのユーザインタフェース

・資源情報保守画面，キーボード操作機能

—資源情報を維持管理するための手続き

・資源情報保守ロジックの定義

—活動を支援する資源情報

・資源データの定義

をカプセル化した構造を持っている．

3) プロファイル……業務活動の結果，発生した「活動データ」および活動を支援する「資源データ」を利用者のさまざまな要求に応じて活用するための視点である．複数の活動データを一つの視点で見ることもできるようになっており，活動データの加工を容易に行うことのできる構造になっている．

4. LINCオブジェクト・モデリング技法

4.1 LINCオブジェクト・モデリング技法の概要

LINCオブジェクト・モデリング技法は，LINCシステムズ・アプローチの中核となる技法であり，主な特徴として次の二つが挙げられる(図3)．

1) 業務活動中心の分析・設計……ビジネスの世界の業務活動を投影したモデル化

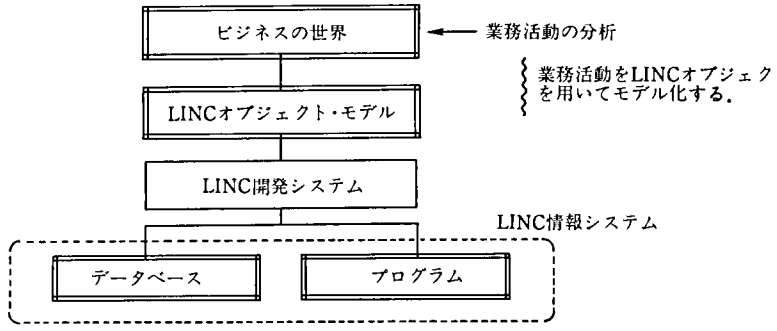


図3 LINCオブジェクト・モデリング

Fig. 3 LINC object modeling

を行い，そのモデルをそのまま情報システムとして実現することができる．このため，ビジネスを取り巻く環境の変化を容易にモデルに取り込むことができ，さらに現状の問題点への対応も容易に行うことができる．

2) データと処理（機能）をカプセル化したLINCオブジェクトによる分析・設計……LINCオブジェクト・モデリング技法では，開発者は対象とする業務活動をデータ，処理（機能）というように分割して分析するのではなく，その全体を“オブジェクト”として把握しつつシステムを作り上げていく．

4.2 LINCオブジェクト・モデリングの作業手順

LINCオブジェクト・モデリングの作業手順は，大きく三つのステップに分けられる．

・ステップ1：モデル，開発機能領域，アクティビティを定義

・ステップ2：個々のアクティビティをLINCオブジェクトによりモデル化

・ステップ3：モデル全体を機能，操作，効率，運用等の観点からレビュー

各ステップの概要を紹介する．

1) ステップ1：モデル，開発機能領域，アクティビティを定義

作成するモデルは図4で示すような構造を持つ．ステップ1では情報戦略計画および業務活動分析の成果に基づき，上位レベルの構造であるモデル，開発機能領域，アクティビティを定義する．

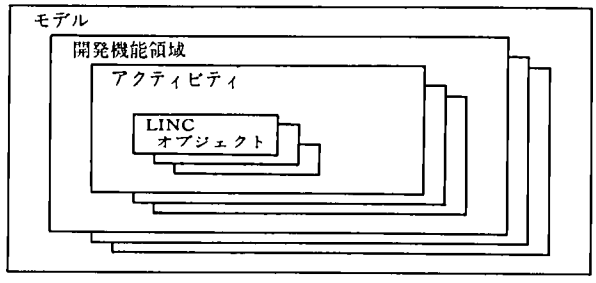


図4 LINCオブジェクト・モデルの構造

Fig. 4 Construction of LINC object model

① モデル

新規に開発する情報システムの対象となる組織を指す．

全社の統合情報システムを構築するのであれば会社全体が対象となり，会社の中の一部の組織を支援する情報システムを構築するのであれば，その組織を対象として定義する．

② 開発機能領域

個々の情報システムを構築する単位となる領域を指す．たとえば，販売管理システムや経理システム等の個々の情報システムを指す．

③ アクティビティ

モデル内で行われている個々の業務活動を指す．たとえば，商品の受注や顧客への販売等の活動を指す．

④ LINCオブジェクト

イベント，コンポーネント，プロファイル等のLINCの構成要素を指す．

2) ステップ2：個々のアクティビティをLINCオブジェクトによりモデル化

各アクティビティを詳細に分析しながら，LINCオブジェクトによりモデル化する．全アクティビティについて次の作業を終了するまで繰り返す．

① 各業務活動に関連した入力情報の分析

② 入力情報に対応するイベントの定義

③ イベントの属性の定義

・画面イメージ

・データ項目定義

・ビジネス規則

④ イベントの処理に必要な資源情報をコンポーネントとして定義

⑤ イベントの処理に必要なプロファイルの定義

⑥ LINCオブジェクト(イベント，コンポーネント，プロファイル等)間の関連の定義

3) ステップ3：モデル全体を機能，操作，効率，運用等の観点からレビュー

① システムの最適化の検討

・イベント，コンポーネント，プロファイルの合理化

・パフォーマンス

・機密保護機能

・運用負荷

② 情報システムの目的に対する機能の実現レベルのレビュー

4.3 LINCオブジェクト・モデル図の表記法

LINCオブジェクト・モデリングでは，LINCオブジェクト，およびLINCオブジェクト間の関連をモデル図として次のように表記する．

1) LINCオブジェクト……LINCオブジェクトは四角形の中にオブジェクトの型と使用属性，およびオブジェクト名を記述して表記する．図5はLINCオブジェクトの表記を示したものである．

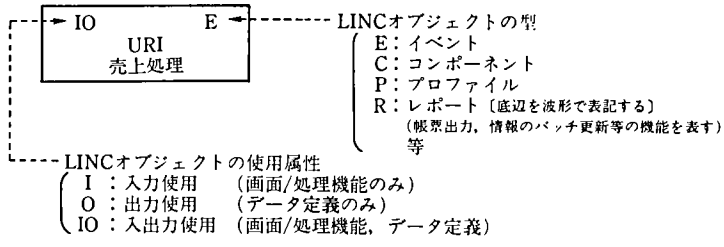


図 5 LINCオブジェクトの表記

Fig. 5 LINC object icon

2) LINCオブジェクト間の関連……LINCオブジェクト間の関係は，直線でLINCオブジェクト同士を結び付け，その関係を表す文字を直線上に記述する．

LINCオブジェクト間の関係には次のようなものがある．

① A (Auto)　：LINC オブジェクトにレコードを追加
② F (Flag)　：LINC オブジェクトのレコードを更新
③ D (Delete)：LINC オブジェクトのレコードを削除
④ R (参照)　：LINC オブジェクトのレコード，またはレコード・グループを参照
⑤ V (検証)　：別の LINC オブジェクトにレコードが存在していることの検証
⑥ G (Recall)：別の LINC オブジェクトの画面に遷移
など

図 6 に LINC オブジェクト・モデル図の例を示す．たとえば「売上処理」はイベントであり，画面/処理機能とデータ定義を持っている．「売上処理」は，「商品情報」と「得意先情報」のコンポーネントに対して検証と更新の関係を持っている．

また，「受払状況照会」は活動データを持たないイベントであり，「商品情報」および「売上処理」のデータを参照している．「売上処理」の活動データの参照は，プロファイル（NYUSYUPRO）を経由して行っている．

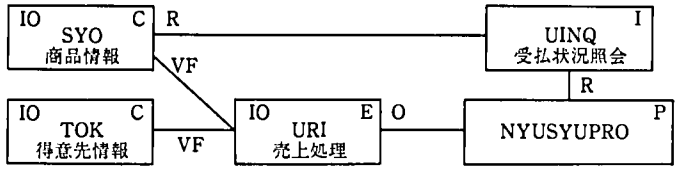


図 6　関連の表記の例
Fig. 6　Example of LINC object model diagram

5. 上流 CASE ツール LINC-DA

前章で述べた LINC オブジェクト・モデリングを支援する上流 CASE ツールが LINC-DA である．

LINC オブジェクト・モデリングは利用者と共に業務活動を分析し，情報システムのモデルを作成する作業であり，そのモデルが LINC 情報システムへと成長していく進化型アプローチの工程の一部になっている．LINC オブジェクト・モデリング作業は，従来より LINC II を用いて行っていた画面・帳票のイメージを提示しながら，開発中の情報システムを明確にするという形態のプロトタイピングを発展させたものと考えることができる．この場合，画面・帳票のイメージを提示するのみではなく，実際の業務の運用を説明した文書が必要であるが，従来はそれを主に人手で作成しなければならなかった．このため，プロトタイピング実施にかなりの工数を要し，頻繁に行うことは困難であった．

LINC-DA は，業務活動と LINC オブジェクトとの関係や，LINC オブジェクト間の関係等をワークステーション上でグラフィカルに表示したり，文書化することができる．LINC-DA を使用することにより，作業の品質を向上すると共に工数を削減でき，利用者の要求の変化にも容易に対応しながらモデル化の作業を遂行することがで

きる．

また，LINC-DA 上に定義された LINC オブジェクト・モデルは下流 CASE ツールである LINC II へアップロードでき，開発フェーズへスムーズにつなげることができる．

5.1 LINC-DA の主な特徴と機能

1) LINC オブジェクト・モデルの作成を支援……ワークステーション上の最新の GUI 環境の中で LINC オブジェクト・モデルの作成をグラフィカルに支援する（図 7）．

—ペインタ，辞書，対応表，グラフ，ビジネス規則等

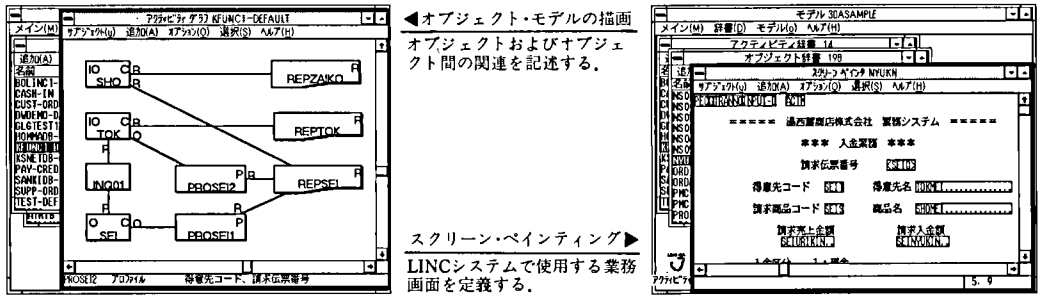


図 7 LINC オブジェクト・モデルの描画およびスクリーン・ペインティング

Fig. 7 LINC object model diagram and screen painting

2) フォワード/リバース・エンジニアリングによる LINC II との親和性の高い連携……LINC-DA 上の LINC オブジェクト・モデルはフォワード・エンジニアリング機能により LINC リポジトリ上へ移行できる．個々の LINC オブジェクトのビジネス規則に基づき LINC 定義言語（LDL 言語）でロジックを記述することによって，情報システムとして完成させることができる．

また，LINC II リポジトリに登録されている既存の LINC 情報システムの記述を，リバース・エンジニアリング機能により LINC-DA 上に移行しデザイン監査機能を用いて評価したり，文書機能を用いて最新の設計情報を得て，保守作業に役立てることができる（図 8）．

3) システム規模の見積りを支援……LINC オブジェクト・モデルから，ファンクション・ポイント法によりシステム規模を自動算出でき，システム開発の管理を支援する．

4) 設計の最適化を支援……LINC オブジェクト・モデルを情報システムの設計モデルとして評価するデザイン監査機能により，不完全な定義，冗長な定義の指摘，およびディスク容量の見積り等を行い，設計の最適化を支援する．

5) 文書出力機能……LINC オブジェクト・モデルに関する各種の分析・設計情報を文書出力機能により提供する．

—モデル図等の各種グラフ，画面/帳票イメージ，辞書，各種対応表，ビジネス規則等

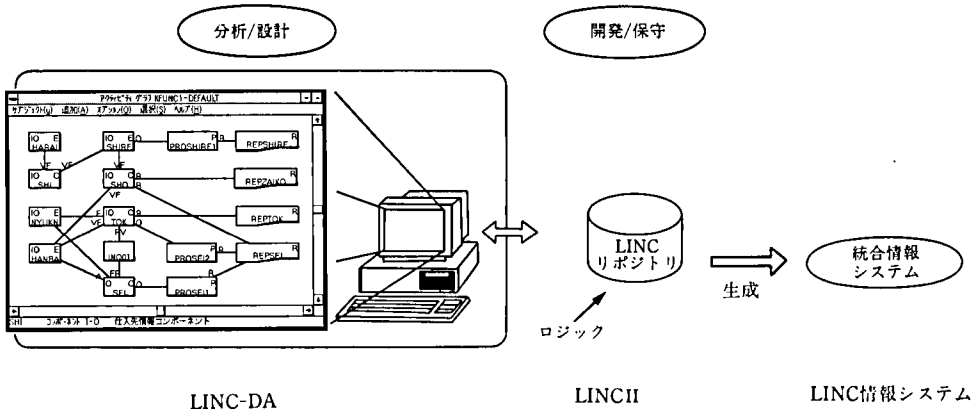


図 8　LINC-DA と LINC II の連携

Fig. 8　The interface between LINC design assistant and LINC II

6. 統合 CASE LINC によるシステム開発

6.1 情報システムの開発作業の進め方

従来型の開発では，情報システムの分析段階は利用者と共同で作業を進めることが多いが，データベースやプロセスの設計を行う段階は情報システム部門の専門家が行い，利用者は多くの場合レビュアとして参加する程度であった．

LINC オブジェクト・モデリングでは，ダイヤグラムを用いて概略から詳細へと利用者と共に業務活動を分析し，情報システムのモデルを作り上げていく．LINC オブジェクト・モデリングと LINC-DA を用いることにより，開発者の一員としての利用者の参加が可能となり，利用者自身の情報システムに対する理解が深まると共に，利用者の要求を反映したより高品質な情報システムを開発することができる．LINC オブジェクト・モデリングを効果的に行うためのポイントは，利用者部門からの代表者を開発プロジェクトのメンバとして参加させる体制を作ることである．

6.2 分析・設計の作業環境の変化

従来は分析・設計段階等で手作業による文書作成が必要であったが，LINC-DA は分析・設計段階の作業をワークステーション上でグラフィカルに行うことができる．定義文書の作成は自動的に行うことができ，さらに最新の GUI 環境の中でモデルをさまざまな視点で分析するのを支援する．また，設計の最適化や既存の LINC 情報システムの評価をデザイン監査機能によって分析したり，さらにシステムの規模をファンクション・ポイント法により算出して工数見積り等のシステム開発の管理に必要な情報を提供する．

これらの機能を利用することにより，属人的な個人技に依存してきた作業を，CASE ツールによる自動化された作業環境によって支援し，作業の品質や生産性の向上を図ることができる．

7. 今 後 の 課 題

統合 CASE LINC と LINC オブジェクト・モデリングについて述べてきたが，これらを導入するだけで直ちにシステム開発の生産性が飛躍的に向上するわけではない．

とくに，次のようなことに留意すべきである．

1) 利用者参画型の開発プロジェクトの編成……LINC オブジェクト・モデリングは，利用者が中心となってプロトタイピングに基づき業務活動を分析する技法である．従来のように情報システム部門が中心となって分析・設計や開発をそれぞれの専門家が行い，利用者はレビュアとしてのみ参加という開発組織・体制はあまり効果的ではない．利用者部門を開発プロジェクトの一員としたチームを編成し，情報システムの分析・設計，開発を一貫して運用する組織・体制を実現することが重要である．

2) 情報システムの目的，対象範囲の整理……LINC はビジネスの世界で行われている業務活動をそのまま情報システムとして実現するツールであるが，実現すべき業務活動が問題を抱えていた場合，そのまま情報システムとして実現しても意味はない．LINC オブジェクト・モデリングは業務活動の理解を促進し要求定義段階で見落とされていた問題点の発見を支援するが，事前に情報戦略計画において情報システムの目標が正しく策定され，それに基づいてシステム化の対象とすべき業務活動が整理されていることが重要であることに変わりはない．LINC オブジェクト・モデリングを効率的に行うために，事前作業である業務活動分析において情報システムに対する要求が正しく整理されていることが重要である．

8. おわりに

本稿では，統合 CASE　LINC と LINC 本来の考え方に基づいた開発技法である LINC オブジェクト・モデリングの概要を紹介した．

統合 CASE LINC 自身の多くの人々の経験を活かした開発方法論・技法に基づいたものであり，今後さらに進化を続けていくべきものである．統合 CASE LINC を有効に活用するためには，その基礎となっている開発方法論を理解した上で，自身の環境に応じた適用を策定しその文化を築き上げていくことが大切である．

参考文献
［1］ LINC Systems Approach, 米国ユニシス社, 1992.
［2］ LINC-DA 解説書, 日本ユニシス社, 1992.
［3］ LINC-DA 操作ガイド, 日本ユニシス社, 1992.

執筆者紹介　佐　伯　克　己　(Katsumi Saeki)
昭和 50 年日本ユニシス(株)入社．主に流通業関連の SE サービス業務に従事．60 年より LINC を中心としての適用サービスを担当し，現在，システム技術本部 LINC ソフトウェア部に所属．

DWとモダナイゼーション

DW (Designer's Workbench) as a Modernization Tool

河合昭男

要 約 モダナイゼーションとは,「既存アプリケーションを大幅に変更することなく活性化し,時代の要請にあったシステムに近代化すること」である.文字端末を前提に構築されてきた膨大なホスト・システムのソフトウェア資産を,最小限のコストでモダナイゼーションができるなら,新規に再構築する必要もなく,システムの延命化が可能となり,その価値は計り知れない.

DW(デザイナ・ワークベンチ)は,4GLのMAPPERまたはLINCで構築されたホスト・アプリケーションのモダナイゼーションを支援するために開発されたワークステーション上のソフトウェアで,Unisys Architecture (UA) のアプリケーションサービスおよびインフォメーション管理サービスに位置付けられるものである.またDWは,INFOConnectを介して1100/2200シリーズ,Aシリーズおよび U6000等のホストと接続でき,文字端末ベースのホスト・アプリケーションは,WindowsのGUIを利用するクライアント/サーバ型として運用可能となる.さらにDWは,COBOL等の3GLホスト・アプリケーションへの適用も計画されている.

Abstract "Modernization" is "to vitalize existing applications with minimum changes entailed so they can be made into renewed ones which meet current requirements." No one would disagree that the practicable modernization, at minimum costs, of huge amounsts of host systems software assets so far created for character-based terminals is surely of immeasurable value because of an eliminated need to rebuild host applications afresh and for the sake of an extended longevity of the systems once built.

DW (Designer's Workbench), a workstation software tool developed to assist in modernizing the fourth-generation language MAPPER- or LINC-based host systems applications, is positioned as one for application and information management services in the UA (Unisys Architecture).

DW, which is connectable to the 1100/2200 Series, the A Series and U6000s through INFOConnect, makes character-based host applications operable in a client/server environment where Windows GUIs are installed. DW is also planned to support 3GL applications as well including COBOL-based programs.

1. はじめに

近年,ワークステーションの操作環境はGUI (Graphical User Interface) 化によるオブジェクト指向操作の普及が急速に進展してきているが,一方ホスト系のアプリケーションの操作環境は依然としてCUI (Character User Interface) のまま取り残された感がある.

モダナイゼーションとは,「既存アプリケーションを大幅に変更することなく活性化し,時代の要請にあったシステムに近代化すること」である.具体的には,その第一歩はGUI化である.文字端末を前提に構築されてきた膨大なホスト・システムのソフトウェア資産を,最小限のコストでモダナイゼーションができるなら,新規に再構築

する必要もなくシステムの延命化が可能となり，その価値は計り知れない．

ホスト系のアプリケーションを，如何にして最小のコストでモダナイゼーションできるかについて一つの解答を示すことが本稿のテーマである．

DW（デザイナ・ワークベンチ）は，4GL の MAPPER または LINC で構築されたホスト・アプリケーションのモダナイゼーションを支援するために開発されたワークステーション上のソフトウェアである．

DW の UA（Unisys Architecture）における位置付けは，次のようになる．まず，ASDF（Advanced Solution Development Framework）とは，UA の「アプリケーションサービス」および「インフォメーション管理サービス」に位置付けられるソリューション構築のための，ツール，サービスおよび方法論を統合した体系である（図1）．DW は，この ASDF の「ツールとサービス」の中の「リエンジニアリングとモダナイゼーション」に位置付けられるソフトウェアである（図2）．

DW は，INFOConnect を介して 1100/2200 シリーズ，A シリーズおよび U6000 等のホストと接続でき，文字端末ベースのホスト・アプリケーションは，Windows* の GUI を利用するクライアント/サーバ型として運用可能となる．

DW は，COBOL 等の 3GL ホスト・アプリケーションへの適用も計画されている．本稿では，前半で DW の 4GL 対応の機能について，後半で開発中の 3GL 対応の機能について述べる．

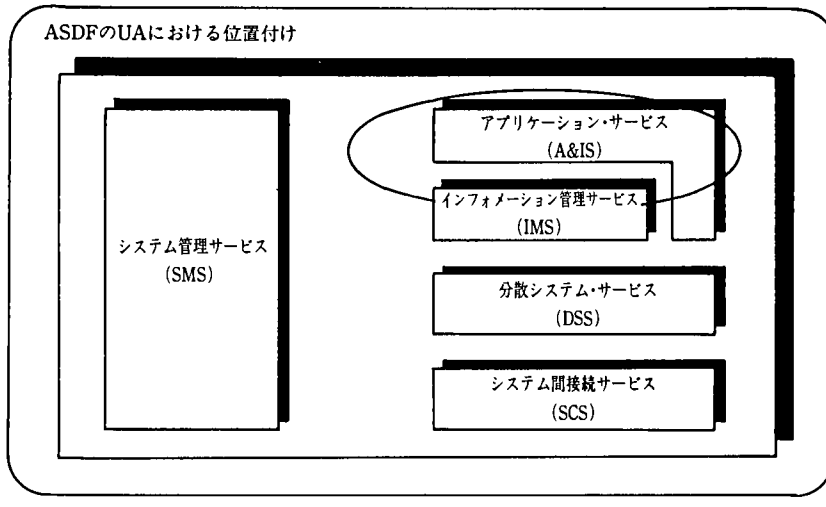


図 1　ASDF の UA における位置付け

Fig. 1　The relationship between ASDF and UA

2. D W と は

2.1 D W 概 要

最初に提供される DW レベル 2R1 は，4GL ソフトウェアである LINC および MAPPER を対象としている．DW により，これらの上で開発された文字端末ベースの

* Windows は Microsoft 社の登録商標である．

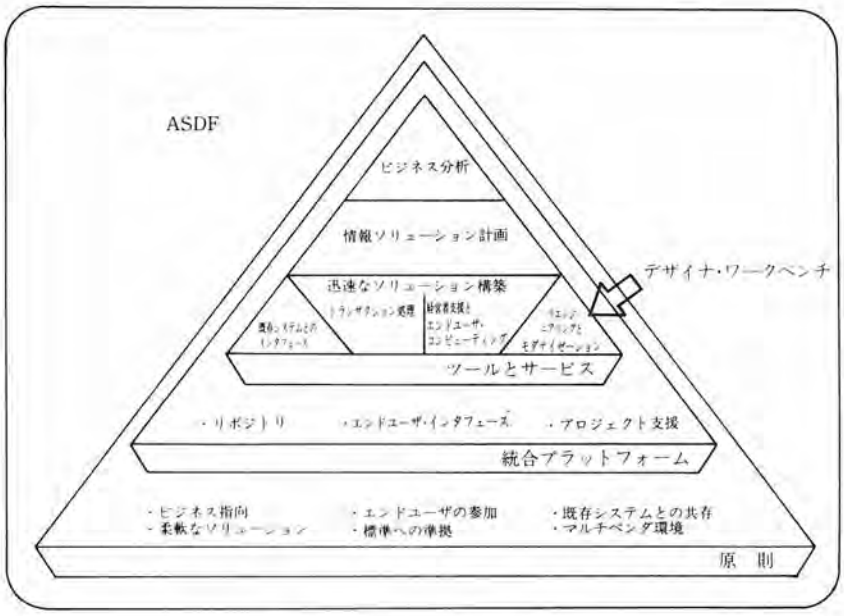


図 2　DW の ASDF における位置付け

Fig. 2　The relationship between DW and ASDF

図 3　DW-MAPPER の例

Fig. 3　An example of DW-MAPPER

アプリケーションは，Windows 上の GUI アプリケーションに生まれ変わることができる．

図 3 に，DW により GUI 化された MAPPER アプリケーションの例を示す．この例では，日本地図の「九州」のボタンをマウスで選択すると，地図左のリストボックスに九州の各県の名前が表示される．続いて，そこから「福岡県」をマウスで選択すると，下のリストボックスに福岡県の保養所の名前と住所が表示される．さらにそこから適当な施設をマウスで選択すると，その施設の内容紹介が文字で，写真と地図がイ

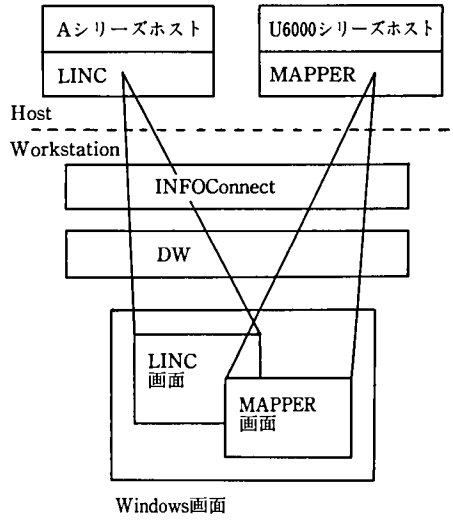


図 4　INFOConnect/DW による複数ホスト接続例

Fig. 4　An example of multi host support by INFOConnect/DW

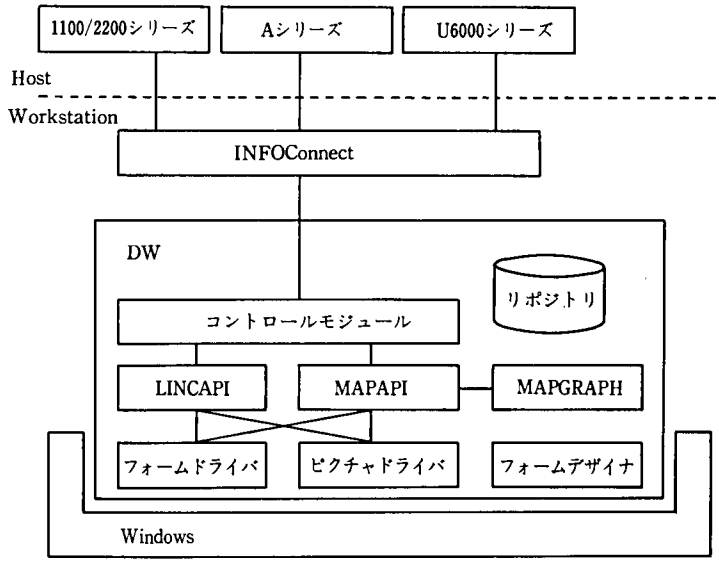


図 5　DW の構成

Fig. 5　DW components

メージで，それぞれが別々のウィンドウとして表示される．

DW は，通信ソフトウェア INFOConnect を介して 1100/2200 シリーズ，A シリーズおよび UNIX* 等のホストと接続できる．たとえば図 4 に示すように，1 台のワークステーションで，一つのウィンドウが A シリーズホストの LINC アプリケーション実行中の状態で，同時に別のウィンドウが U6000 シリーズの MAPPER アプリケーションを実行することもできる．

DW の構成は，以下のようになっている（図 5）．

* UNIX オペレーティングシステムは UNIX System Laboratories, Inc. が開発し，ライセンスしている．

・コントロールモジュール：INFOConnect を介してホストと通信を行う．
・MAPAPI：MAPPER コマンドを解釈して，Windows に表示できるように変換する．
・LINCAPI：LINC コマンドを解釈して，Windows に表示できるように変換する．
・フォームデザイナ：開発用ツール．マウス操作により，アプリケーションのための会話画面を作成する．(後述)
・フォームドライバ：フォームデザイナで作成したフォームを表示する．
・MAPGRAPH：MAPPER のグラフを表示する．
・ピクチャドライバ：ビットマップイメージを表示する．
・リポジトリ：フォーム，リストデータおよびイメージ等を管理するワークステーション側のデータベース．

2.2 フォームデザイナ

フォームデザイナは，GUI の会話画面を作成するためのアプリケーション開発ツールである．マウス操作によるオブジェクト指向操作で，実際の画面を見ながら容易に作成することができる．

フォームデザイナでは，次のようなオブジェクトを作成することができる．

・テキスト：固定文字
・フィールド：文字の入力および表示
・ボタン：コマンドボタン，チェックボタン，オプションボタン
・イメージ：イメージ自体は，事前に他のツールで作成してリポジトリにインポートしておくことが必要
・リストボックス：一覧表表示
・罫線：直線，矩形

これらのオブジェクトは，フォームデザイナ主ウィンドウ(図 6(a))左端に表示される適当なツールをマウスで選択して，ウィンドウの適当な位置にマウスで作成する(図 6(b))．次に，そのオブジェクトをマウスでダブルクリックすると，オブジェクト属性を定義するためのダイアログボックスが表示される (図 6(c))．属性はオブジェクトの種類により異なる．図のフィールドオブジェクトの例では，入力属性としてアルファベットや漢字等の指定ができ，表示属性として下線や枠等の指定ができる．

こうして作成されたフォームオブジェクト（会話画面）はリポジトリに保存して，実行時に MAPPER 等から呼び出すことができる．

3. DW によるモダナイゼーション

本章では，DW により，どのようにして今までの文字ベースの MAPPER アプリケーションが Windows により GUI 化できるかを示す．会話画面を作成するには，DW 画面作成ツールとして提供されるフォームデザイナにより一画面分をマウス等を用いて対話形式で一度に作成する方法（図 7）と，個々のオブジェクトを DW のための新しい MAPPER ラン命令により作成する方法（図 8）の 2 通りの方法がある．

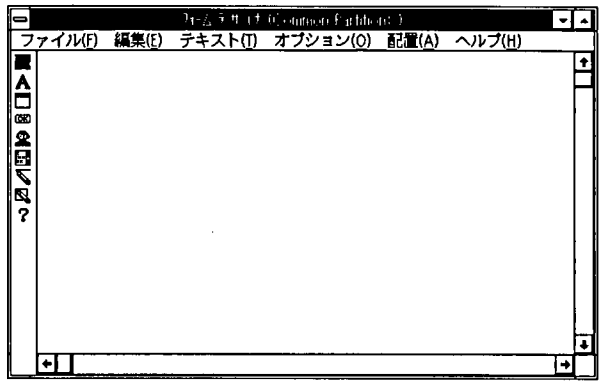


図 6(a) フォームデザイナ初期画面

Fig. 6(a) Forms designer main window

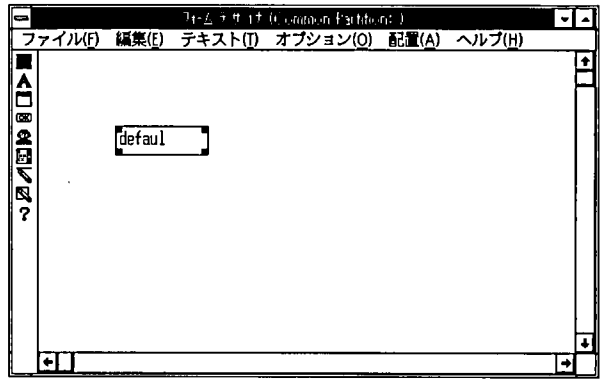


図 6(b) フィールド作成

Fig. 6(b) Create a field object

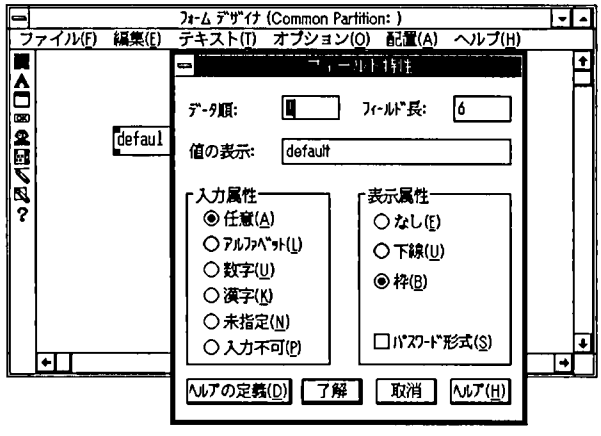


図 6(c) フィールド属性定義

Fig. 6(c) Define attributes for the field object

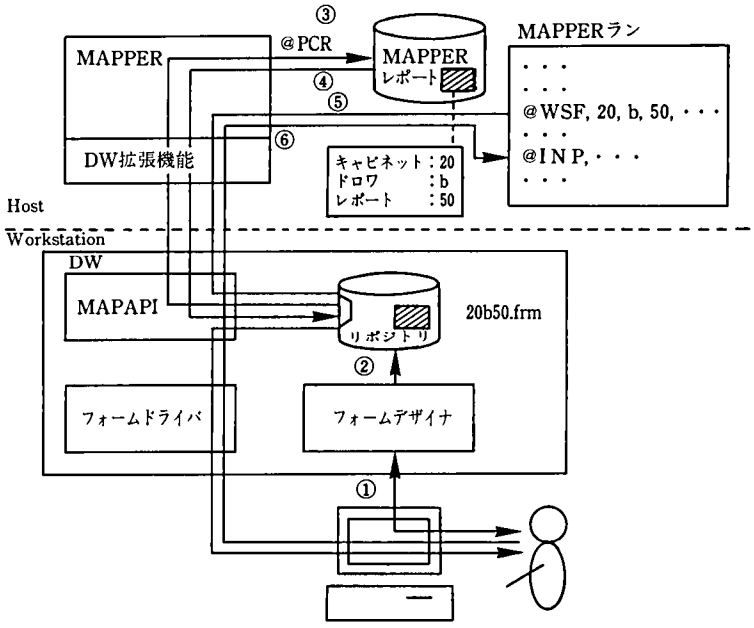


図 7　フォームデザイナによる会話画面作成例

Fig. 7　Create a dialogue by the Forms Designer

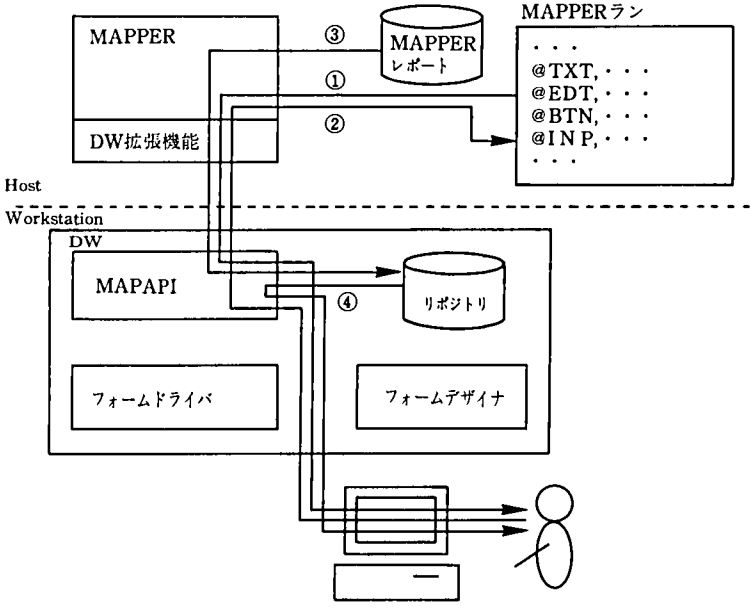


図 8　MAPPER ラン命令による会話画面作成例

Fig. 8　Create a dialogue by the MAPPER run

3.1　フォームデザイナにより会話画面を作成する方法

① フォームデザイナによりマウス操作で会話画面をあらかじめ作成する．

② 完成した画面を，リポジトリにフォームオブジェクトとして保存する．

③ 保存したオブジェクトを，MAPPER の新しいラン命令 PCR により MAP-

PER レポートに取り込む.

④ 以後実行時には，MAPPER の新しいラン命令 WSF によりこの画面を DW に表示させることができる. MAPPER は画面を表示する時，リポジトリに該当するオブジェクトが存在するかを確認して，存在すればそれを使用し，存在しない場合のみダウンロードを行う.

⑤ 操作員が入力したデータは，MAPPER の新しいラン命令 INP により受け取ることができる.

フォームデザイナによる方法の特徴は，次に述べる MAPPER ラン命令による方法と比べて，マウス操作で容易に画面作成できることである.

3.2 MAPPER ラン命令により会話画面を作成する方法

DW のために新しく MAPPER ラン命令が追加され，次のような Windows スタイルの会話画面が自由に作成できるようになった.

ウィンドウ
メニュー（メニューバーおよびプルダウンメニュー）
ボタン
エディットボックス（文字入力/表示フィールド）
リストボックス
イメージ
テキスト

① MAPPER ランより DW 命令が呼ばれると，DW のモジュール MAPAPI が Windows 命令に変換して画面表示を行う.

② 操作員が入力したデータは，MAPPER の新しいラン命令 INP により受け取ることができる.

③ リストデータやイメージは，本体は MAPPER レポートに保存するが，ワークステーション側のリポジトリにもコピーを保存することができる.

④ MAPPER ランからこれらリストデータやイメージのオブジェクトが呼び出されたとき，MAPAPI はワークステーション側のリポジトリにコピーが存在するか調べ，存在しなければホストより取り込み，存在すればそれを使用する.

MAPPER ラン命令による方法の特徴は，先に述べたフォームデザイナによる方法と比べてアプリケーションからきめ細かな制御ができることである.

4. DW の 3 GL 対応

4.1 概　　要

現在使用されている DW レベル 2 R 1 は，LINC および MAPPER の 4 GL ソフトウェアを対象としている. 現在計画中の DW の 3 GL 対応は，COBOL 等のホスト 3 GL アプリケーションの操作を現在の DW と同等の Windows 操作にするためのソフトウェアである.

DW の 3 GL 対応は，既存のホスト 3 GL アプリケーションを変更することなく Windows 化できることが特徴である. 画面を変換するためのツールしてモダナイザが提供される. モダナイザは文字ベースの画面を GUI 化するためのものである.

LINC および MAPPER の 4 GL ソフトウェアを，DW の 3 GL 対応機能によりモダナイゼーションすることも可能である．

DW の 3 GL 対応実行時の処理の流れは，以下のようになる（図 9）．

① ホストの従来の 3 GL アプリケーションより，端末に画面が送られる．この画面は端末のディスプレイには表示されない．

② 画面を識別し，対応するフォームオブジェクトをリポジトリより読み取って表示する．リポジトリにない場合は，ホストより自動的にダウンロードされる（後述）．

③ 利用者はマウスとキー操作でフォームとの会話を行う．

④ 端末は利用者の入力したデータを，従来通りのデータ形式でホストアプリケーションに送る．

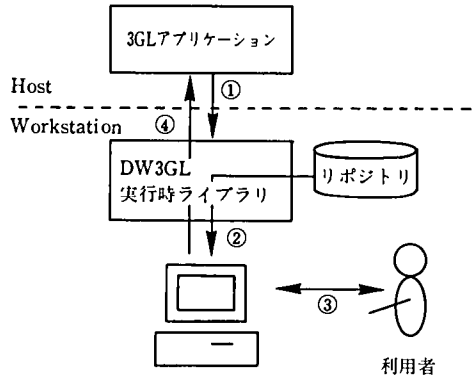


図 9　DW の 3 GL 対応の仕組み

Fig. 9　DW for 3 GL overview

DW の 3 GL 対応は，次のようなワークステーション上の開発ツールと実行時ライブラリおよびホスト上のプログラムから構成されている．

① 開発ツール

・3 GL モダナイザ

ホスト 3 GL アプリケーションのエミュレータ画面情報を，Windows 環境で使用できる形式に変換する．

・フォームデザイナ

3 GL モダナイザで取り込んだ画面に，ボタン，リストボックス，イメージやカラー等を付加して，Windows スタイルの画面に化粧直しをする．

② 実行時ライブラリ

ホスト 3 GL アプリケーションとデータを交換し，利用者のためにフォームを表示するためのワークステーション側ソフトウェア．

③ ホストプログラム

・ホストフォームサーバ（HFS）

フォームやリストデータ等のマスタを管理し，接続されているすべ

てのDWワークステーションに配布するためのホスト側ソフトウェア（後述）.

4.2 開発環境

DWの3GL対応でのモダナイゼーションとは，3GLで書かれたホストアプリケーションの画面情報とユーザインタフェースを定義する処理のことである．画面情報としては，各画面を識別するためのデータや各フィールドのタイプ，位置および長さ等がある．

この処理で取り込んだ情報は，DWの3GL対応の実行時に，利用者が3GLで書かれたホストアプリケーションと会話するために使用される．

画面の開発手順は次のようになる（図10）.

① 3GLホストアプリケーションの画面を受け取る.

② モダナイザにより画面情報を取り込む.

③ 画面の各フィールドの形式を，画面操作により次の四つから選択して定義する.

表示フィールド
画面識別フィールド
入力フィールド
固定フィールド

④ モダナイゼーションの作業が終了すると，フォームとそれに付随するフォーム識別情報およびバージョン制御ファイルをリポジトリに保存する.

⑤ フォームデザイナによりフォームオブジェクトの編集を行う.

⑥ フォームオブジェクトをHFSにアップロードする.

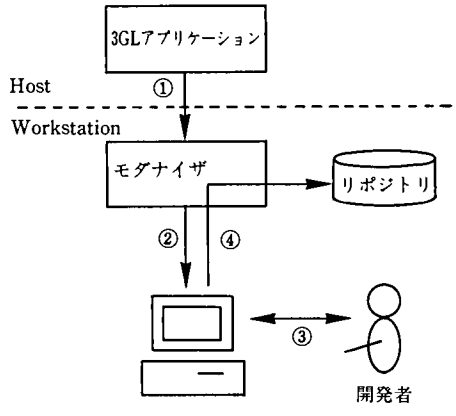


図 10 モダナイザ処理

Fig. 10 Modernizer process

4.3 ホストフォームサーバ

ホストフォームサーバ（HFS）は，DW3GL対応のデータを集中管理するためのホスト常駐ソフトウェア（図11）で，以下のデータを持つ：

フォーム
リストボックスのデータ

ビットマップイメージ
フォーム識別データ
3 GL アプリケーション画面形式

1) 開発環境……管理者用のワークステーションでモダナイゼーションまたはフォームデザイナによるフォーム編集処理が終了すると，これらのフォームを DW 3 GL 対応の全利用者に使用できるように HFS へのアップロード処理を最初に必ず実行しなければならない．

2) 実行環境……一度 HFS にアップロードされると，後は各 DW ワークステーションからリクエストされた時に自動的にダウンロードされる．

ホスト 3 GL アプリケーションより画面が送られてくると，DW 3 GL 実行ライブラリはその画面を認識し，対応するフォームをローカルにあるリポジトリから探す．見つからないとき HFS にダウンロードを要求する．一回目は必ずダウンロードすることになる．新しいバージョンが HFS にアップロードされたとき，HFS は各 DW ワークステーションにそのフォームが無効であり HFS に新しいバージョンがあることを通知する．

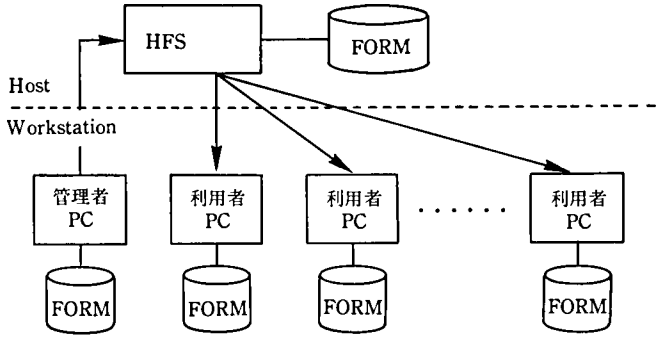


図 11　ホストフォームサーバ
Fig. 11　Host form server

5. 今後の展望

DW は，文字端末ベースのホストアプリケーションのモダナイゼーションを支援するソフトウェアとして開発された．その第一段階として，LINC および MAPPER の 4 GL アプリケーションのモダナイゼーションを DW レベル 2R1 で提供できるようになった．第二段階は，ホスト 3 GL アプリケーションをモダナイゼーションできるように現在計画中である．では，その第三段階としてどのような発展形態が考えられるであろうか．

(案 1)　DW クローズ型

DW にローカルロジックを定義できるように拡張する．

(案 2)　外付け型

DW に OLE (Object Linking and Embedding)，DDE (Dynamic Data Exchange)，DLL (Dynamic-Link Libraries) 等の Windows 標準機能を利

用できるように拡張する．

(案3)　NBDI型

"Name Brand Drop In"とは，市販されている著名ソフトウェアとDWを連携動作させることである．たとえば，現在エンドユーザ向けのWindowsアプリケーション開発用ツールとして，次のようなソフトウェアが市販されている．

Visual Basic*
Tool Book**
Power Builder***
Object Vision****

これらのツールに共通している特徴は，C言語等の開発者向け言語を知らなくても主として画面操作によりアプリケーションを作成できることである．たとえば，会話画面等はマウス操作で容易に実際の画面を見ながら作成することができる．ロジックの作成には，それぞれ固有のスクリプト言語等が提供されている．

画面とロジックを誰が定義するのかと言う観点からこれらの案を比較する（表1）．

表1　DW拡張案の比較

Table 1　Comparison of DW extension

比較項目 / 拡張案	画　面	ロジック
(案1)DWクローズ型	DW	DW
(案2)外付け型	DW，一部外付け	外付け
(案3)NBDI型	外付け	外付け

3案ともワークステーション側で定義する点では共通している．

3案の基本となるDWは，今までホスト側で定義していた画面をワークステーション側で定義可能にした．このDWに対して(案1)「DWクローズ型」は，ロジックもワークステーション側のDWでできるように拡張するものである．この方式は，DWの改造が大きいことが考えられる．(案2)「外付け型」は，DWにOLE，DDE，DLL等のWindows標準インタフェースをインプリメントするのみで，これらの機能の特徴を活かしてユーザアプリケーションのロジックを外付けで組み込めるようにするものである．ロジックのみならず，OLEが支援できれば画面も取り込み可能となる．この方式は（案1）よりDWの改造は少ないと思われ，またさらに利用者が外付けで機能拡張できる可能性が開ける．

（案1）（案2）はDWが主体であるのに対して，（案3）「NBDI型」はロジックおよび画面ともに市販ソフトウェアで開発し，DWはホストとの連携に利用しようという考えである．この方式の特徴は，前述のような優れたオブジェクト指向のWindowsアプリケーション開発ツールを利用できることである．

* Visual Basicは Microsoft 社の製品である．
** Tool Bookは Asynmetrix 社の製品である．
*** Power Builderは Powersoft 社の製品である．
**** Object Visionは Borland 社の製品である．

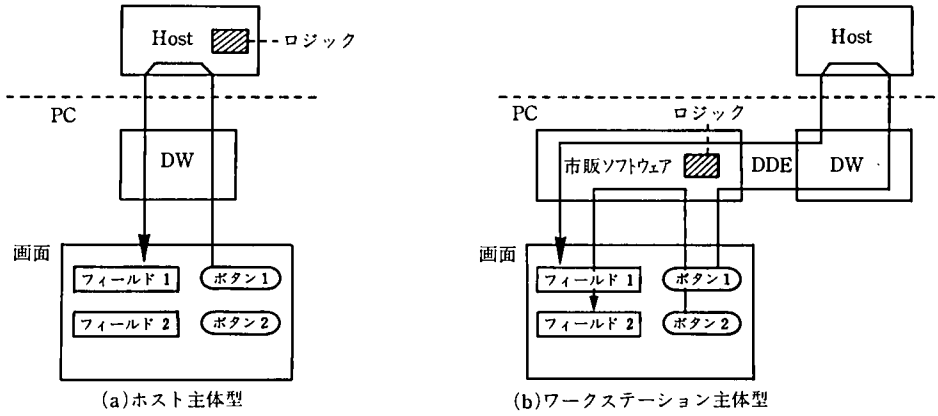


図 12　市販ソフトウェアとの共同作業

Fig. 12　Name brand drop in

クライアント/サーバ型アプリケーションとして見たとき，現状の 4 GL/3 GL 対応の DW はホスト主体型で，クライアント側（WS 側）は画面とユーザインタフェースの処理をすることが中心となる．

(案 3)が実現できると，ワークステーション主体型のクライアント/サーバ型アプリケーションを，C 言語等の助けを借りずに構築することができる．たとえば図 12 のように，あるボタンを選択するとホストからデータを取り込み，別のボタンを選択するとワークステーションからデータを取り込むことや，複数の画面間をワークステーション側の論理でナビゲーションするようなことも容易にできることになる．

6. おわりに

本稿のテーマは，「ホスト系のアプリケーションを，如何にして最小のコストでモダナイゼーションできるかについて一つの解答を示すこと」であるとはじめに述べた．DW2R1 により，LINC および MAPPER の 4 GL ソフトウェアのモダナイゼーションが可能となり，DW の 3 GL 対応により COBOL 等ホスト 3 GL アプリケーションのモダナイゼーションが可能となる．ここまでのレベルはモダナイゼーションの第一歩，すなわち GUI 化によるオブジェクト指向操作の実現である．

モダナイゼーションの次のステップとして，次のようなことが考えられる．

クライアント/サーバ型アプリケーション

オブジェクト指向の開発環境

エンドユーザ指向の開発環境

これらを実現するための一つの可能性は，5 章で述べたように市販の開発ツールを活用することである．これは，DW が Windows というオープンな環境で稼働するため実現可能なことである．これらの市販ツールは，オブジェクト指向かつエンドユーザ指向の開発環境として強力なものであるが，ホストとはそのままではインタフェースがとれない．これらとホストとの間でインタフェースをとることができれば，一歩進んだモダナイゼーションが実現できる．

執筆者紹介 河 合 昭 男 (Akio Kawai)

昭和22年生．45年大阪大学理学部数学科卒業．46年日本ユニシス(株)入社．EXEC保守，シリーズ1100性能評価，無人化システム開発を経て59年よりマイクロ部門に移りDS 7イメージ処理，EXOS開発，DW導入・開発を担当．情報処理学会会員．

IDESのリバース機能

Reverse Engineering for IDES

儀 間 哲 雄

要 約 IDESはCOBOL言語によるシステム開発を支援する下流CASEツールとして開発された．IDESによるプログラム開発では連接，反復，条件の制御構造を木構造図に作図し，処理部をその図の中の木の葉に記述して仕様書を作成する．

リバース機能はこの逆の手順をたどる．しかし，IDESには固有のルールがあり，多種多様なプログラムをIDESに受け入れるには困難がある．

本稿では，IDESのフォワード・エンジニアリングのプロセスと現在開発中であるリバース・エンジニアリングのプロセスを記述する．

Abstract IDES has been developed as a lower CASE tool which supports the development of COBOL-based systems. Program development by IDES requires a structural diagram to be drawn for controls including sequence, iteration and selection, and operations to be described in the diagram for the production of program specifications so COBOL programs can be generated from the specifications.

Reverse engineering provides the process of the opposite. However, IDES's own rules make it difficult for all varieties of programs to be accepted by IDES. This paper discusses the forward engineering process that IDES supports and the reverse engineering process now under development.

1. はじめに

IDESは設計から保守工程までを支援する下流CASEツールである．IDESはこれまでシステムの新規開発における開発環境を主に支援してきたが，リバース機能の充実により適用範囲を既存システムの保守にまで拡大しつつある．多くの使用者はすでに莫大なシステム資産を持っており，この資産を有効に利用するための開発環境を切望している．

IDESのリバース機能とは，既存のシステム資産をIDESの成果物として取り込むことだと考えられる．IDESの成果物にはリポジトリと仕様書がある．リポジトリは登録集へ展開し，仕様書はソースコードへ展開することによってプログラムという開発の目的とする成果物を作り出す．この流れを逆にたどるのがIDESのリバースであり，リポジトリは既存の登録集から，仕様書は既存のソースコードから作成する．

しかし，既存のシステムは多くの使用者の多種多様なルールに基づいて作成されており，一方IDESも固有のルールに基づいている．このことは，IDESへ取り込むことの困難性を示唆している．

本稿では，既存のCOBOLプログラムをIDESの仕様書に取り込む機能に的を絞って取り込みの規則や適用上の問題点等を明らかにし，その解決策を考えていきたい．

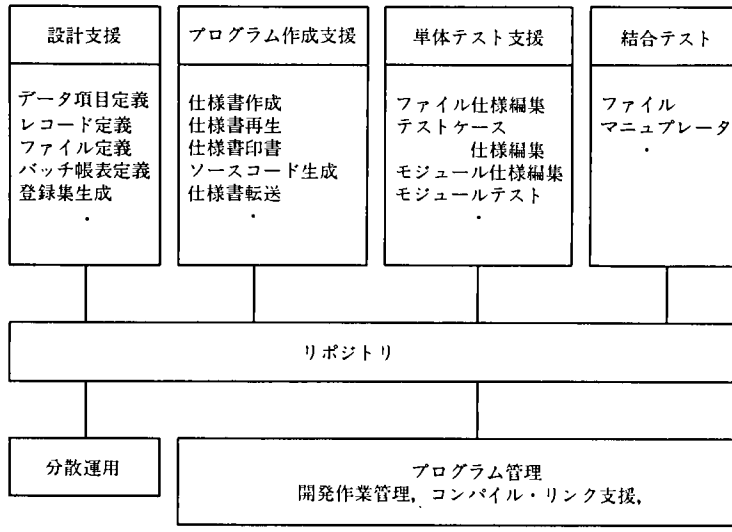


図 1　IDES のツール体系
Fig. 1　Functional diagram of IDES

2. IDES の概要

IDES はシステム開発の各工程を支援するツール群から構成される．使用者は，開発作業の各工程でその工程を支援するツールを使う(図 1)．

各ツールは情報をリポジトリに蓄え，またリポジトリを参照しながら処理を進める．リポジトリは，開発工程の進行につれて充実していく．

設計工程で確定したデータ項目やレコードの定義はリポジトリに蓄えられ，また登録集の形に変形されて保存される．プログラミングの工程ではリポジトリを参照しながら，仕様情報を蓄積し，仕様書が作り出される．仕様書からソースコードが生成され，また登録集を参照して構文検査が行われる．ここで生成されたソースコードは登録集とともに次工程の単体テストで使用される．単体テストの不具合は，仕様書の修正によって解決される．ソースコードをホストで修正した場合は，ソースコードから仕様書が再生される．

3. IDES によるプログラム開発

既存 COBOL プログラムのソースコード（以降，ソースコードと呼ぶ）を IDES のモジュール仕様書（コンパイル単位に作成する仕様書）に取り込む規則はモジュール仕様書からソースコードを生成する規則を裏返ししたものとなる．また，IDES に取り込むことは IDES で作成するモジュール仕様書の形に変形されることを意味する．このため，IDES のリバース機能を理解するには，IDES のフォワード・エンジニアリングによるプログラム開発の流れや規則を知ることが糸口となる．

3.1 モジュール仕様書の作成

IDES ではコンパイル単位の仕様書をモジュール仕様書と呼び，モジュール仕様書の定型書式を画面に表示して，書式ごとにデータを登録し仕様書を完成させる．類似

表1　モジュール仕様書の書式と定義する内容
Table 1　Form and definition of module specification

書式名(画面書式)	定義する内容
表紙	システム名，サブシステム名，言語等のモジュールの属性や仕様書およびコード作成者等の情報
処理概要	前提条件，処理内容要旨，処理内容(自由形式で記述)
呼び出し	当モジュールの呼び出し系列の説明(引き数名，型，桁数等)や使用する外部サブルーチンの名前
メッセージ	実行時に表示するメッセージとその説明(番号単位に自由形式で記述)
構造図	モジュールの構造(作図)，条件や処理の詳細(自由形式で記述)
エリア	使用するデータエリア等の登録集原文名または COBOL 言語での定義(後者は自由定義と呼び，特定の節や段落に自由形式で記述)
ファイル定義設定	使用するファイルのレコード名や SELECT 句，FD 句の登録集原文名
テストケース説明	テストケース(番号単位に自由形式で記述)

の仕様書を画面に表示して，修正する方法（ひな型流用）もある．モジュール仕様書の各書式で定義する内容は，表1のとおりである．

モジュール仕様書に記述するものには，文書としての意味を持つものとソースコードに展開されるものがある．メッセージやテストケース説明は文書としての意味を持ち，組み込み式のテキストエディタを使い，日本語で自由に記述することができる．

構造図はモジュールの構造を木構造エディタで作図し，選択や繰り返しの条件および処理の詳細を COBOL 言語で記述していく．これらのデータはソースコードに展開される．モジュールで取り扱うファイルやレコードの指定，報告書節で使用する登録集の指定等は登録集の複写 (copy 命令) としてソースコードに展開される．ここで指定する登録集は前工程で使われる設計支援ツールで作成される．

この他，WORKING-STORAGE　SECTION で使用するワークエリアや COMMON-STORAGE や LINKAGE 等の節，SPECIAL NAMES や I-O-CONTROL 等の段落に定義する情報は，組み込み式のテキストエディタを使い COBOL 言語で自由に記述することができる．これらの記述もソースコードに展開される．

なお，モジュール仕様書はワークステーションで作成または修正し，サーバに保存する．サーバではこの仕様書をソースコードに展開する．この過程でマクロの展開や構文検査を行う．構文検査でエラーのなくなったソースコードをホストへ転送し，コンパイル/リンクして単体テストを行う．

完成した仕様書はホストへ転送し，ホストで管理する．ホストで管理している仕様書の修正は，仕様書をホストからサーバに転送後行う．

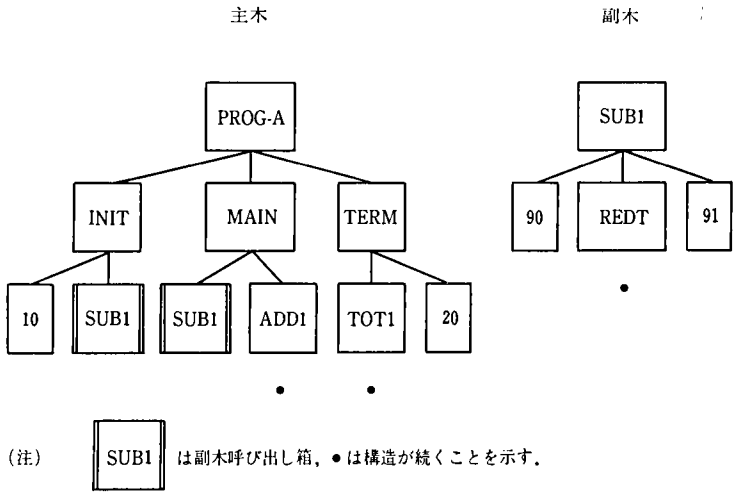
3.2　モジュール構造と処理の記述

3.2.1　木構造図の作図

IDES ではモジュールの構造を木構造図として作図する．木構造図では連接，選択，繰り返しの三つの基本構造を組み合わせてモジュールの構造を表現する．

モジュールの全体を制御する構造を持った木を主木と呼ぶ．この主木の複数箇所から呼び出されるものを副木と呼び，別の木として作る．副木から副木を呼び出すこともできる．副木を呼び出す箱を副木呼び出し箱と呼ぶ．一つのモジュールは一つの主

木と0個以上の副木から構成される．



3.2.2 条件および処理の詳細の記述

選択や繰り返し構造の条件には条件箱が与えられる．この箱は組み込み式のテキストエディタで開かれ，その中に選択や繰り返しの条件をCOBOL言語で記述する．

なお，条件箱は条件番号で管理されるため，同じ条件のものには同じ条件番号を与えればよい．

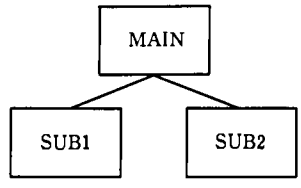
木のルートの箱や副木呼び出し箱を除く箱に数字の名前を与えると木の葉（オペレーション箱）として定義される．オペレーション箱は最下位の箱でこれに従属する箱はない．この箱は組み込み式のテキストエディタで開かれるので，その中に処理の詳細（オペレーション）をCOBOL言語で記述する．オペレーション箱はオペレーション番号で管理される．

3.3 モジュール仕様書からソースコードへの展開

3.3.1 木構造図のソースコードへの展開

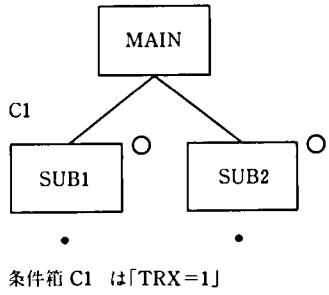
モジュール仕様書はソースコードに展開される．IDESでは，この機能をソース生成と呼ぶ．ソース生成では一定の規則に従って，モジュール仕様書をCOBOLのソースコードに展開する．このとき，ソースコードからモジュール仕様書を再生するための印を付ける．木構造図からソースコードへ展開する規則は次のとおりである．

1) 連接の構造……上位の箱は節または段落とし，下位の箱はPERFORM命令の手続き名とする．




```
MAIN.
        PERFORM SUB1
        PERFORM SUB2.
SUB1 .
                •
SUB2 .
                •
```

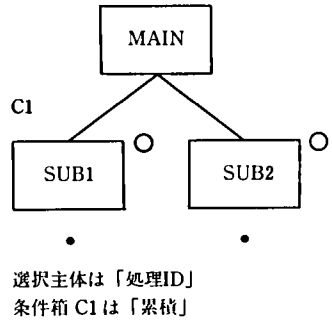
2)　選択の構造……上位の箱は節または段落とし，その下に選択を表す構文（IF, IF～ELSE～）を生成する．下位の箱は選択を表す構文の中のPERFORM命令の手続き名とする．




```
MAIN.
C1      IF   TRX = 1
             PERFORM SUB1
        ELSE
             PERFORM SUB2.
SUB1.
          •
SUB2.
          •
```

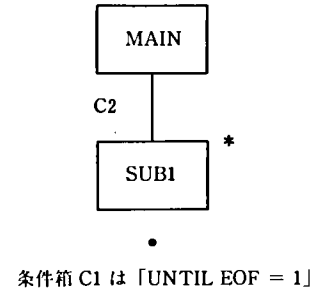
なお，選択を表す構文をEVALUATE（選択の箱の属性で指定する）とした場合，木構造を作図する時にCOBOL言語で選択主体を入力する．

ソース生成では選択主体をEVALUATEとWHENの間に展開し，WHENの次に条件箱の内容を展開する．下位の箱はその後に続くPERFORM命令の手続き名とする．



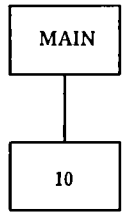

```
MAIN.
        EVALUATE 処理ID
C1        WHEN 累積
             PERFORM SUB1
          WHEN OTHER
             PERFORM SUB2
        END-EVALUATE.
SUB1.
          •
SUB2.
          •
```

3)　繰り返しの構造……上位の箱は節または段落とし，下位の箱はPERFORM命令の手続き名とする．手続き名の後に条件箱の内容を展開する．




```
MAIN.
        PERFORM SUB1
C2              UNTIL EOF = 1.
SUB1.
          •
```

4)　木の葉（オペレーション箱）……上位の箱は節または段落とし，下位の箱はそのままソースコードに展開する．



オペレーション箱 10 は
「MOVE TREC TO MREC
MOVE TCOD TO MCOD」

```
MAIN.
10          MOVE TREC TO MREC
10          MOVE TCOD TO MCOD.
```

3.3.2 ソースコードに付加される IDES の印

IDES で作成したソースコードには仕様書再生で必要な情報が付加される．このうち，ソースコードの一連番号領域（1～6 カラム）に付加されるものを IDES マークと呼ぶ．使用者はこの領域を別の目的で使用してはいけない．

IDES マークには以下のものがある．

① Cnnn ：条件番号と呼び，木構造図で定義した条件箱が展開される行の 1 カラム目から左詰めで条件箱の番号を付加する．nnn は 1～999．

（例） C1 IF TRX=1

② nnn ：オペレーション番号と呼び，木構造図で定義した木の葉の箱が展開される行の 1 カラム目から左詰めで木の葉の箱の番号を付加する．nnn は 1～999．なお，マクロ展開部分はソース生成の過程で展開されるものであり木の葉の中には記述されていない（木の葉の中にはマクロ呼び出しが記述されている）ため，オペレーション番号は付加しない．

（例） 10 MOVE TREC TO MREC

③ J ：自由定義が展開されたことを表す印で，データ部や手続き部の特定の節や段落に自由定義として記述した情報が展開される行の 6 カラム目に「J」を付加する．

```
（例）      WORKING-STORAGE    SECTION.
          J 01 WORK-DATA    PIC X(10).
```

④ M ：マクロ呼び出しやその展開部分であることを示し，その行の 6 カラム目に「M」を付加する．

なお，ソース生成ではマクロ呼び出し部分はコメント行にする．

```
（例） 20  M*   &ADD-MACRO A B C ←マクロ呼び出し
           M      ADD  A, B  TO  C ←マクロの展開部分
```

また，木構造図の情報のうち，COBOL 言語で記述できない情報はコメント行として生成する．IDES が生成するコメント行には以下のものがある．

① ＊SW ：直後に続く選択の記述（IF～ELSE～等）が多分岐の選択構造であることを示す．

② ＊PROC ：直後の節または段落が，1 箇所からしか呼び出されていない副木であることを示す．

③　＊LABEL：空白をおいて「＊LABEL」の右に書かれた文字列が以降の構文をくくった箱の名前であることを示す．

これらのコメント行は，ソースコードから仕様書を再生する時に必要な情報であり，使用者はその目的以外で変更してはいけない．

3.4 モジュール仕様書の修正

IDESではモジュール仕様書からCOBOLのソースコードを自動生成するが，この時点のソースコードは構文上のエラーや処理上のエラーを含んでいる．IDESはソース生成の過程で構文検査を行う．このときCOPY命令で指定された登録集は，前工程の設計支援ツールで作成された登録集ライブラリから読み込んできて検査の対象に含める．使用者はモジュール仕様書を修正して，構文エラーを解決する．モジュール仕様書をワークステーションに読み込み，エラーの箇所を表示して修正し，サーバに保存する．保存後，再度ソース生成を行い，エラーがあればこの手順を繰り返す．エラーがなくなるとホストへ転送し，単体テストを行う．

単体テストで発見されたエラーを解決する場合，ワークステーションで仕様書を修正しソースコードに展開してホストへ転送する．この方法では，仕様書とソースコードは常に同期の取れたものとなる．

4. モジュール仕様書の再生

単体テストで発見した不具合を解決する方法として，ホストでソースコードを修正する方法もある．IDESではこのような場合でも仕様書とソースコードの同期が取れるように，ソースコードからモジュール仕様書を修正する機能を支援している．この機能を仕様書再生と呼ぶ．

仕様書再生では，IPF等のエディタを使ってホストでソースコードを修正し，単体テスト結果の不具合がなくなった時点でホストのソースコードをサーバに転送し，サーバのモジュール仕様書にソースコードの修正部分をマージする．

ただし意図したとおりに再生するためには，IDESのソース生成の規則に準拠してソースコードを修正する必要がある．ソースコードを修正する場合に遵守すべき規則には，以下のものがある．

1) 条件番号が付いている条件の行が追加になる場合，新しい行にも同じ条件番号を付ける．
2) オペレーション番号が付いている部分で行が追加になる場合，新しい行に同じオペレーション番号を付ける．
3) オペレーション番号や条件番号は別の構造で同じものを使用できる．このため，オペレーションや条件に変更がある場合，他の部分では使われていないことを確認する必要がある．もし，同じ番号を複数箇所で使用し他の箇所で同じ修正が有効でない場合は，オペレーション群全体に新しいオペレーション番号を与える．
4) 新たにオペレーション番号や条件番号を使う場合，この番号は他では使っていない番号を使用する．同じ番号を使用した場合の不具合は3)と同じである．

仕様書再生機能は平成5年7月より機能拡張され，これまで対象外であった木構造図の構造部分や登録集指定の情報が再生されるようになる．これに伴い，いくつかの

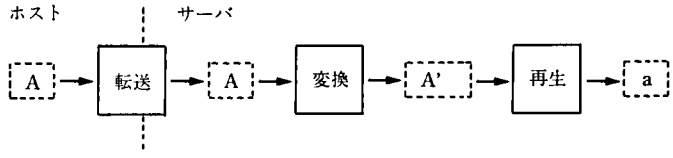
規則が追加される．

なお，構造部分は本来設計者がモジュールに要求された機能を正しく実現するための構造として定義したものであり，その修正はモジュール仕様書の修正という形で実現することを推奨する．ホストでソースコードを修正する方法は手軽であるが，以下の注意点があることを忘れてはいけない．

1) 使用者がIDESのソース生成の規則を正しく理解し，再生を考慮してソースコードの修正ができること．
2) 再生前後でのソースコードが論理的に同じであることを確認する作業を作業の手順に組み込むこと．
3) すでに修正中の仕様書を間違って使用しないよう，ソースコードと仕様書の同期を確認する作業を作業の手順に組み込むこと．

5. 既存COBOLプログラムのIDESへの取り込み

ここまでIDESによるモジュール仕様書の作成手順，仕様書からソースコードへの展開規則，仕様書の修正手順および仕様書の再生機能について触れてきた．本章ではIDES以外で作成されたソースコードをIDESのモジュール仕様書に取り込む機能について説明する．IDES以外で作成されたソースコードをモジュール仕様書に取り込む手順は，ホストにあるソースコードをサーバに転送し，サーバでIDESの印を付けるための変換処理を行いIDESで作成したソースコードと同じ形式にしたあと，モジュール仕様書の再生機能を使ってIDESに取り込む（図2）．



（注）A は使用者のCOBOLプログラム
A' はAにIDESの印が付加されたもの
a はA' より再生されたモジュール仕様書

図 2 既存COBOLプログラムを取り込む処理の流れ
Fig. 2 Process to talk in the COBOL program

このようにしてソースコードから作成された仕様書は，IDESのフォワード・エンジニアリングにそってテストを行い，正当性を確認しなければならない．

IDESで作成したソースコードからの再生はモジュール仕様書の存在が前提であり，これにソースコードをマージしていたが，IDES以外のソースコードを取り込む場合は，仕様書を新規作成する．なお，ソースコードから再生されない部分は，仕様書編集処理（更新読み込み→修正→保存）で補う必要がある．

なお，既存COBOLプログラムをIDESに取り込む機能はACOBプログラムを当面のターゲットにしており，以降の記述は，ACOBプログラムに限定される．

5.1 手続き部の取り込み

COBOLプログラムのソースコードをIDESのモジュール仕様書に取り込む規則は仕様書の各書式によって異なる．ここでは，ソースコードの手続き部からモジュール仕様書の木構造図に取り込まれる時の規則を説明する．

5.1.1 節や段落の取り込み

最初の節または段落は主木の箱になる．それ以外の節や段落も構造の箱になる．ただし，その箱が木の枝として作成されるか副木のルートとして作成されるかは，以下の規則による．

1) 木の枝になる場合……PERFORM命令で一度しか呼び出されていない節や段落は呼び出し箇所で木の枝になる．
2) 副木のルートになる場合……その節や段落が複数のPERFORM命令で呼び出されている場合，またはGO TO命令やSORT命令の手続き名として指定されている場合は，主木から独立した木（副木）のルートになる．このとき，呼び出し箇所（PERFORM命令の所）には副木呼び出し箱が生成される．

なお，ソースコードのどこからも参照されない節や段落は，その直前の木の葉の中にコメントとして取り込まれる．

5.1.2 構造記述の取り込み

手続き部の記述のうち，連接，選択，繰り返しの構造を表す記述は，木構造図の構造の定義として取り込まれ，他の構文は木の葉の箱に取り込まれる．その規則は次のとおりである．

1) IF ～ または IF ～ ELSE ～ は選択の構造を作る．
2) PERFORM 手続き名 の後に条件の構文がある場合は繰り返しの構造を作る．
3) 2)以外の PERFORM 手続き名 は連接の構造を作る．
4) IFの後の条件文や PERFORM 手続き名 の後の条件文は条件箱に取り込まれる．条件箱に付ける番号（条件番号）は変換処理の中で自動採番される．
5) 1)，2)，3)に該当しない構文は木の葉の箱に取り込まれる．木の葉の箱はソースコードの内容（複数行も可）をそのまま持つが，この部分の文字数が多い場合は複数の箱に分割する．木の葉に付ける番号（オペレーション番号）は変換処理の中で自動採番される．

なお，木構造の階層はPERFORM命令によって深くなる．IDESではこの深さのことを深度と呼ぶ．深度は浅すぎると構造を表すには不足し，深すぎるとかえって構造を把握しにくくする．このため，深度は仕様書の単位またはIDESシステムの単位に使用者が指定できるものとする．使用者の最適な深度は試行結果により求められる．

また，IDESでは木の葉の中に節や段落を記述することはできない．このため，深度を越える部分にある節や段落は副木として取り込まれる．

5.1.3 コメントの取り込み

手続き部のコメントは木の葉に含める．通常ソースコードに記述するコメントは直後の節や段落の処理を説明するものが多いが，IDESではコメントは木の葉の箱に記述し，木の葉の箱は節や段落の後にソース生成される．このため，節や段落の直前のコメントは，直後の節や段落に含める機能を持つ．この機能を利用する場合は，取り

込み前後のソースコードでコメントの出現場所が変わることになる．

```
（取り込み前のコメント）
*****************************
* (1.1) TRX , MASTER OPEN  *
*****************************
 FILE-OPEN.
     OPEN FILE-1
          FILE-2.
```

```
（取り込み後のコメント）
 FILE-OPEN.
*****************************
* (1.1) TRX , MASTER OPEN  *
*****************************
     OPEN FILE-1
          FILE-2.
```

5.1.4 ジェネレート・ラベル

IDES のソース生成では選択構造の IF や ELSE の後には 1 個の PERFORM 命令を生成する．このため，IF や ELSE の後に PERFORM 以外の命令があったり，複数個の PERFORM 命令がある場合，取り込み処理では選択構造の下位の箱を自動的に 1 個作り，その箱の下に，本来の命令群の入った箱を作る．このため，取り込み後の仕様書からソースを生成すると自動的に作成した箱の名前が段落名または節名として残る．これを，ジェネレートラベルと呼ぶ（6.3 節 3）参照）

5.1.5 そ　の　他

COPY 命令は木の葉の中に取り込まれる．取り込み処理では COBOL 登録集の解析は行わない．

6. 既存 COBOL プログラムを IDES へ取り込む場合の問題点

これまでの説明の中でも，IDES に取り込む場合の問題点がいくつかあげられてきた．ここでは，それらも含め予想される問題を列挙し，検討していきたい．

6.1 IDES に取り込めないプログラム

1) 構造の大きなプログラム……IDES のプログラム作成支援では，開発しやすい単位に分割されたモジュールの開発を支援するものであり，モジュール構造の大きさやオペレーションの記述に対し，数の制限を持つ．主な制限には以下のものがある．

① 木の節の数は 416 個以内
　木の節とは，構造や木の葉の箱の数に主木と副木の数を加えたもの．

② 副木の数は 50 個以内
　副木とは主木から独立した木をさす．

③ 木の葉と条件箱の数の総和は 256 個以内

このような制限を越えるものは IDES には取り込めないが，通常のモジュールはこの制限内であり，特殊な大きいものだけが取り込めないと思われる．これらのモジュールは，今後の保守も考えれば IDES に取り込まないとしても保守しやすい大きさに分割するのが妥当だと思われる．

なお，制限を越えるものについては，変換処理の中でその数が明示される．

2) 手続き名—1　THRU　手続き名—2　の記述があるプログラム……PERFORM

A THRU Cの構文はAからCの節または段落がソースコードの記述順に実行されることを要求している．一方，IDESのソース生成も節や段落を実行順にソースコードに展開するが，上記PERFORM命令より以前にPERFORM Cが実行される場合は，ソースコード上はCが先に展開される．このため，変換処理では，AからCまでの節や段落が他の構文で参照されている場合はエラーとする．節や段落は一つのまとまりのある処理の単位であり，木構造図でモジュールの開発や保守を行う場合，その呼び出し箇所も含め木構造に明示されるべきである．上記PERFORM命令の場合，たとえば複数のPERFORM命令に分解する修正を行えば，IDESに取り込むことができる．

（修正前：取り込めない）

```
    PERFORM C.
      •
    PERFORM A THRU C.
      •
A.
    ADD X1 X2 TO XX.
B.
    ADD Y1 Y2 TO YY.
C.
    ADD XX TO YY GIVING ZZ.
```

（修正後：取り込める）

```
    PERFORM C.
      •
    PERFORM A.
    PERFORM B.
    PERFORM C.
      •
A.
    ADD X1 X2 TO XX.
B.
    ADD Y1 Y2  TO YY.
C.
    ADD XX TO YY GIVING ZZ.
```

6.2 構造の把握しにくいプログラム

1) 構造化されていないプログラム……木構造図は構造と木の葉から構成される．既存COBOLプログラムを取り込む場合，構造を表現する手がかり語(IF, ELSE, EVALUATE, PERFORM等)から構造が作成され，その他の構文は木の葉に埋め込まれる．この方法により，多くのCOBOLプログラムをモジュール仕様書として取り込むことができる．しかし，もともと構造化されていないプログラムは木構造図で表現するには無理があり，横いっぱいに広がった木の枝に構造的な部分が散見される形で取り込まれる．また，構造化されていないプログラムはGO TO命令を多用していると思われるが，GO TO命令は木の葉に隠れるため，木構造図から処理を見ることは困難である．

2) 構造の大きなプログラム……6.1節の1)で述べた構造の大きなプログラムでも，木の深度を浅くすることによって箱の数を減らせば，IDESに取り込める可能性は多い．この場合，所定の深度を越えた構造部分は木の葉となり，構造として把握できなくなるが，それぞれのモジュールや木にとってどこまでが構造かということをソースコードから判断することは困難である．

IDESに取り込んだ場合，構造として把握する局面は画面や印書された木構造図を見る時であり，最適な深度が不明だとすれば，木構造図として見やすい程度の深度にするのがむしろ良いと思われる．IDESの1画面で表示できる深度は約7深度であるが，この程度の深度が一番見やすい．

6.3 ソースコードの記述が変わるプログラム

IDES への取り込みは IDES のルールに押し込めるという側面を持つ．このため，取り込みによってソースコードが変形される（ソースコードの記述が変わる）場合があり，取り込み前後のソースコードの比較だけでは取り込み結果の正当性は確認できない．IDES へ取り込んだ場合，新しい仕様書を基にソースコードを生成しテストで正当性を確認する必要がある．このテストは今後の保守の基盤を確認するものであり，この手間を省いてはならない．

ソースの記述が変わるケースのうち，いくつかのものを以下に列挙する．

1) 節の出入り口の段落名

ソース生成では節の出入り口の段落名を節名より自動的に生成する．

入り口の段落は節名-ST，出口の段落名は節名-EX となる．この形式はソース上で節の開始，終了を見やすいものとしているが，使用者が作成した節の出入り口の段落名とは異なると思われる．

また，節の出口の段落名の次の行には EXIT 命令だけを生成する．このため，

```
（取り込み前のソースコード）
MAIN.
      IF    TODAY = 1ST-DAY
            PERFORM SUB1
            PERFORM SUB2
      ELSE
            PERFORM SUB3
                UNTIL EOF = 1
            MOVE TREC TO MREC.
条件箱 C1 は「TODAY=1ST-DAY」
条件箱 C2 は「UNTIL EOF=1」
オペ箱 10 は「MOVE TREC TO MREC.」
```

```
（取り込み後のソースコード）
MAIN.
      IF TODAY = 1ST-DAY
         PERFORM G-L1
      ELSE
         PERFORM G-L2.
G-L1.
      PERFORM SUB1
      PERFORM SUB2.
G-L2.
      PERFORM SUB3
          UNTIL EOF = 1
      MOVE TREC TO MREC.
```

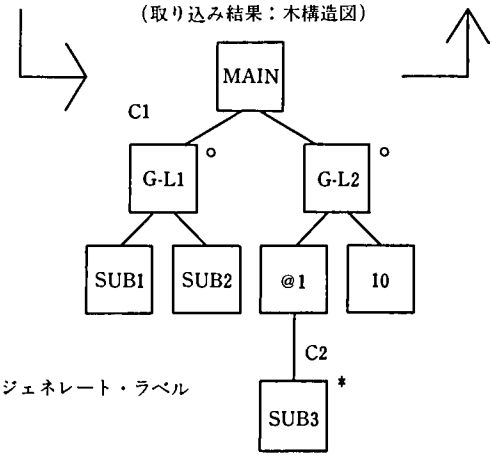


図 3 ジェネレート・ラベルの例

Fig. 3 Example of generate label

節の出口の段落にEXIT PROGRAM命令やSTOP RUN命令を記述しているソースコードは，その位置を出口の段落名の前に変更する等の修正をあらかじめ行う必要がある．

2) 誰からも参照されない段落名

木構造では木の葉の中に段落名を記述することはできない．このため，ソースコードにある段落で構造の箱にならないもの（誰からも参照されない段落）はコメントとして取り込まれる．もともと誰からも参照されない段落はソースの解読性を高める意味を持つものと思われるが，IDESでは同じ位置にコメントとして生成する．

3) 選択構造で選択された一方の直下には一つの箱しか作成できない．このため，そこに複数の構造を持つ場合，木構造図に取り込む時に内部的に選択の箱を1個作成し，その下に複数の構造を作成する．このため，ソースコードにPERFORM命令と手続き名の行が追加される．生成された手続き名をジェネレート・ラベルと呼ぶ(図3)．

7. おわりに

以上述べてきたように，IDESのモジュール仕様書の取り込みにはいくつかの問題がある．構造化によりプログラムの開発や保守を支援するという意味でいえば，既存のCOBOLプログラムを解析し，適切な構造化プログラムに変形することが最適であるが，実現には非常なスキルと時間を必要とする．むしろ，今回開発された機能を早く試行し，その評価に基づき逐次改善していく方が妥当である．

既存COBOLプログラムをIDESに取り込む目的には，ソースコードとドキュメントの同期を取ることや既存システムの保守にIDESのフォワード・エンジニアリング機能を使用すること等が考えられるが，いずれにしてもIDESへ取り込んだことの成果が現れるものでなければならない．

これまでIDESで開発されたシステムより，IDES外で開発されたシステムの方がはるかに多い資産をかかえていることを考えれば，IDESへ取り込む機能を充実し，IDESに取り込むことによる効果を多くの使用者が享受できるようにしなければならない．このことは，単にリバース機能の充実を図るものではなく，同時にIDESそのものの習熟度を高めることが必要となる．

執筆者紹介　儀間哲雄（Tetsuo Gima）
昭和46年大阪大学文学部卒業．同48年日本ユニシス(株)入社．金融ユーザのSEサービスを担当．63年以来生産技術一部所属．IDESの開発，保守に従事．

UNIX環境における業務アプリケーション構築

The Creation of Applications in a UNIX-based Environment

福地修一, 吉野良成

要 約 UNIX*をベースとした環境で稼働する業務アプリケーション・システムの開発が増加してきている. 従来の, 汎用計算機を利用したアプリケーション・システムに比べ, エンジニアリング・ワークステーション (EWS) を採用することによる利点は多いが, 計算機システムの安全性や, 大量導入時の運用管理・セキュリティ管理等の問題点も指摘されている.

筆者等は, 平成元年より, EWS/UNIX 環境下での科学技術系/分析・計画系業務アプリケーションの適用評価, 調査研究, および受託開発業務をいくつか経験してきた. これらの経験を通して認識した, UNIX 環境での業務アプリケーション開発における最大の特徴は, その技術的な自由度の高さであった.

本稿では, これまでの開発経験をもとに, 業務アプリケーションと UNIX のプロセスとの関係や, 業務アプリケーションで提供すべきいくつかの機能についての技術的自由度を考察し, 本格的にアプリケーションを作り込む場合の筆者等の開発スタイルや工夫点, 留意事項等を述べる.

Abstract Application systems which run in a UNIX-based environment have been developed in growing numbers. Compared with the development of application systems on conventional mainframe systems, more advantages are found in using engineering workstations (EWS), but problems are also emerging such as the reliability of computer systems, operational management and security in the case of volume installation.

Since 1989, the authors have gone through several projects including the evaluation, research and studies, and contracted development of scientific engineering applications in addition to ones for analytical and planning purposes. The authors' recognition resulting from those experiences is that the most significant characteristic involved in developing UNIX-based applications is a high degree of technical freedom.

Besides considering, on the basis of the development efforts experienced so far, the relationships of an application system with the UNIX process and the extent of technical freedom relative to some functions to be supplied in applications, this paper discusses development methods, ideas and items deserving of attention which the writers hope are helpful for any intensive project for the creation of new applications.

1. はじめに

エンジニアリング・ワークステーション (以下 EWS と略記) の高性能・低価格化の進展につれ, EWS/UNIX 環境で稼働する業務アプリケーション・システムの開発要求が増加している.

従来の汎用計算機とキャラクタ端末装置を前提とした場合と比べ, EWS/UNIX を

本稿に記載の会社名, 商品名等は, 一般に各社の商標または登録商標である.

* UNIX オペレーティングシステムは, UNIX System Laboratories, Inc. が開発し, ライセンスしている.

採用した場合の業務アプリケーションの特徴として，一般に次のような点が挙げられる．

1) 強力な CPU パワーと大容量メモリの占有により，アプリケーションのレスポンスが向上する．
2) 高解像度ディスプレイと X Window System*をベースとしたグラフィカル・ユーザ・インタフェースの提供により，大容量の情報を表示でき，操作性を向上させることができる．
3) 一つの login セッション内で，同時に複数のアプリケーションを実行させることができる．
4) ネットワークを前提とした標準的な各種サービスや，ネットワーク透過なプロセス間通信メカニズムを利用することにより，負荷の分散や資源の共有を図ることができる．
5) 業務アプリケーションの一部やその開発支援のために，各種の UNIX 標準コマンドや多くの流通ソフトウェアを利用できる．

一方，計算機システムの安全性や信頼性，大量導入時の運用管理・セキュリティ管理等，現時点では，EWS を採用した場合に問題となる点も多い．

筆者等は，平成元年より，US ファミリ**をハードウェア・プラットフォームとする科学技術系/分析・計画系業務アプリケーションの適用評価，調査研究，および受託開発業務をいくつか経験してきた．

汎用計算機と比べた場合の，UNIX での業務アプリケーション設計/開発の最大の特徴は，その技術的な自由度の高さにある．たとえば，ネットワーク上でのデータ共有方式やユーザ・インタフェースの実現方法，印書出力方法等，アプリケーション・システムが提供すべき機能のほとんどに関して，多様な実現方法が考えられる．各々の機能について，UNIX の標準機能やパッケージ・ソフトウェアを利用して比較的容易に実現できる場合もあれば，本格的に作り込まなければならない場合もある．

しかし，逆に自由度が高い分，ハードウェア構成やネットワーク構成，ソフトウェア構成，環境設計，運用方式等，業務アプリケーションの設計時に考慮しなければならない事項が増大している．

本稿では，まず業務アプリケーションと UNIX のプロセスとの関係について概観し，次に業務アプリケーションで提供すべきいくつかの機能に着目して，これまでの経験を紹介しながらその技術的自由度を考察する．最後に，UNIX/C を前提として本格的にアプリケーションを作り込む場合の，筆者等の開発スタイルや工夫点，留意事項を述べる．

2. 業務アプリケーションと UNIX のプロセス

本章では，業務アプリケーションと UNIX のプロセスについて考察する．

2.1 サーバ・クライアントモデル

サーバ・クライアントモデルに基づいた分散処理が脚光を浴びているが，一口に「サ

* X Window System は米国 MIT の登録商標である．
** 日本サン・マイクロシステムズ(株)の SPARCstation, 同 server の OEM 機である．

ーバ・クライアント型」と言っても，その意味するところは様々である．UNIX ワークステーションによるローカルエリア・ネットワーク環境に限定しても，次のような意味で「サーバ」「クライアント」という用語が使われている．

1) X Window 環境で，アプリケーションの表示部（フロントエンド）として X サーバ（X 端末を含む）を使用する．
2) アプリケーションのデータベース部（バックエンド）としてサーバ・クライアント構造で稼働する市販 DBMS を使用する．
3) アプリケーション自体のインストール場所や共有データファイルを格納するディレクトリを NFS マウントする．
4) 特定の周辺機器に関するサービスを提供するワークステーション（プリンタサーバ，ディスクサーバ等）．
5) 各種のソフトウェア的なサービスを提供するためのプロセス，またはそのプロセスが稼働しているワークステーション（メールサーバ，NIS サーバ，仮名漢字変換サーバ，ネームサーバ等）．
6) 各業務アプリケーションで，独自にソケットや RPC(Remote Procedure Call) を用いて実現したサーバプロセスとそのクライアント．

UNIX をベースとしたネットワーク環境では，上記のような各種の形態の役割分担によって，負荷の分散やハードウェア資源の共有を図ることができる．

一般に，サーバ・クライアントモデルでは，あるサービスを提供するプロセス（＝サーバ）と，そのサービスを利用するプロセス（＝クライアント）によって，各種の処理が実現される．サーバ・クライアント方式によりハードディスクやプリンタ等のハードウェア資源を共有する場合もあるが，あくまでもソフトウェア的にそのサービスが実現されていると捉えるべきである．したがって，あるワークステーション上で稼働しているプロセス群を見ると，あるサービスについてはサーバであり，他のサービスについてはクライアントであるような場合もある．

アプリケーション・システムの構築においては，どのような意味で「サーバ」「クライアント」という用語が使用されているのかに注意を払う必要があろう．

2.2 プロセス構成とコマンド

UNIX 上で稼働する業務アプリケーションでは，まずそのアプリケーションを構成するトップレベルのプロセスが何等かの形で起動され，その後ユーザの操作・指示に従って，ユーザ要求を受け付けたプロセス自身が該当する処理を実行したり，すでに起動されている他のプロセスや新たに起動した子プロセスに処理を依頼したりする．

ある場合は，ユーザ要求の受付，アプリケーション内部での処理，および処理結果の表示/出力がすべて同一のプロセス内で行われ，またある場合には，その各々が独立したプロセスとして実現される．

データベースとこれを中心とする中・小規模のプロセス（＝データベースクライアント）群から構成されるようなアプリケーションもあれば，一つのプロセスが大量のデータを一括して主記憶に展開し，そのプロセスがほとんどの処理を行うようなアプリケーションもある．

どのようなプロセス構成を採用するかは，アプリケーションに要求される対話性の

高低や，処理内容，使用するミドルソフトウェア等により異なる．

プロセスとは，プログラムのインスタンスのことであり，そのプログラムは，「コマンド」として用意される．したがって，UNIX オペレーティング・システムから見れば，業務アプリケーションは，コマンドの集合として実現される．

コマンドは，その実装形態により，主に C, C＋＋等のコンパイラ言語により記述し，コンパイル・リンクされたバイナリ形式のコマンドと，シェルや awk 等のインタプリタへの入力となるテキストファイルの形で用意されるスクリプト系のコマンドとに分類できる．

また，一つの業務アプリケーションの中では，その業務アプリケーションに固有のものとして開発されたコマンドの他に，UNIX で標準的に提供されるコマンドや，ミドルソフトウェアやパッケージ・ソフトウェアに含まれるコマンドが利用できる．

2.3 プロセス間の連係方法

汎用計算機では，一つのジョブの中ではタスクがシリアルに実行され，タスク間のデータの授受には中間ファイルを使用する場合がほとんどであった．

これに対し，UNIX では，一つの login セッション内で，複数のプロセスをパラレルに起動することが容易であり，あるプロセスから別プロセスを動的に起動したり，複数のプロセス間で連係を取ったりしながら全体の処理が行われる形態が中心である．

また，業務アプリケーションの実装レベルで利用可能な，プロセス間のデータ連係や同期のための方法も多様である[1][2]．その代表的なものを次に示す．

① コマンドライン引数	⑧ 共有メモリ
② 環境変数	⑨ セマフォ
③ 通常のファイル	⑩ 仮想端末
④ シグナル	⑪ ソケット
⑤ パイプ	⑫ TLI(Transport Level Interface)
⑥ 名前付きパイプ(FIFO)	⑬ RPC
⑦ メッセージ	⑭ X サーバを介したクライアント間通信

このようなメカニズムにより，汎用計算機では業務アプリケーションレベルでの実現が技術的に困難だったことが，UNIX 環境では比較的容易に実現できる．これにより，アプリケーションに対する高度な要求に対しても，多様な実装手段で対応することができる．

3. アプリケーションの機能面から見た技術的自由度

本章では，業務アプリケーションが提供すべき各種機能の面から，EWS/UNIX 環境下での技術的自由度を概観しつつ，筆者等の経験を紹介する．

3.1 データ保存/共有形式

ネットワーク上の複数のワークステーションから，複数のユーザによって同時に使用されるような業務アプリケーションにおいて，ネットワーク内でのデータ共有が必要な場合，これを実現するための方式として，次のようなものが考えられる．

1) 市販のリレーショナル・データベース管理システム (RDBMS) を導入し，これをサーバ・クライアント型で使用する．

2) データ管理に UNIX ファイルシステムをそのまま使用する．

この場合，データ更新時の排他制御が別途必要になるが，データ共有に関するメカニズムとしては，次のようなバリエーションが考えられる．

① NFS サーバ上にデータ管理用ディレクトリを用意し，各 NFS クライアントがこれを NFS マウントする．

② データ管理用のホストとローカルホスト間で，rcp や ftp によるファイル転送を行う．

③ 参照が中心になる少容量のデータで，更新がまれであれば，非標準の NIS マップとして管理する．

④ X Window 環境であれば，rlogin/xon/xrsh 等を用いて，ディスプレイサーバをローカルホストにしたまま，データ管理用（＝リモート）ホスト側でアプリケーションを実行する．

⑤ アプリケーション層のプロトコルを独自に設計し，データ管理用のサーバプロセスとクライアントライブラリを開発する．サーバ・クライアント間の通信には，ソケットや RPC を利用する．

単に，「データの共有」を図るのが目的であれば，ネットワーク対応の RDBMS を採用するのが簡便である．しかし，とくに科学技術系の業務アプリケーションにおいては，必ずしもリレーショナルモデルが適切とは言えないケースも少なくない[3]．このような場合，オブジェクト指向データベースを採用する方法や，データ実体は UNIX ファイルとするが，そのインデックス情報の管理には RDBMS を利用する方法等が考えられる．そのアプリケーション自身の性質や稼働環境に応じた選択が必要になる．

ソケットや RPC を利用して，独自にサーバプロセスを構築するよりは，サーバ・クライアント間のプロセス間通信の詳細を隠蔽してくれるミドルソフトウェアとして RDBMS を採用した方が，開発効率は高い．しかし，近年，いわゆる「グループウェア」的なアプリケーションへの要求も出始めている．この手のアプリケーションでは，たとえば，あるデータベース・サーバプロセスに接続されている一つのクライアントからの要求によってデータベースが更新された場合に，サーバに接続されているすべてのクライアントに対する更新内容の同報通知が必要になる．このような機能の実現は，現在市販されている RDBMS 等のパッケージ・ソフトウェアでは困難である．

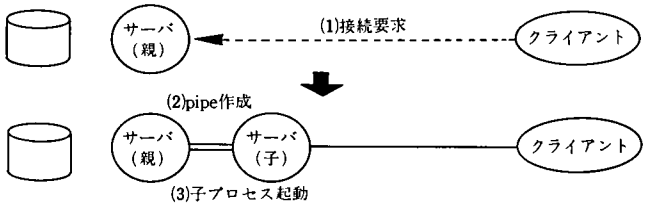
筆者等は，実際に生じたこのような要求に対して，データ管理用のサーバプロセスと，アプリケーション・プロトコルを，ソケットをベースとして独自に開発することで対応した．参考までにそのメカニズムの概要を図 1 に示す．

ただし，その後 Sun Microsystems 社からは，ToolTalk（アプリケーション間でのメッセージ交換のためのメカニズム）が提供されるようになり[4]，また，GUI ツールキットを使用した X クライアントであり，かつ RPC のサーバであるようなプロセスの実現も可能になっている[5]．これらを用いることで，今日では別の解決方法もあり得ることを付記しておく．

3.2 ユーザ・インタフェース

ユーザ・インタフェースについても，実現方法は多様である．最近はグラフィカル・ユーザ・インタフェース（以下 GUI と略記）ばかりが脚光を浴びているが，GUI だけがユーザ・インタフェースではない．

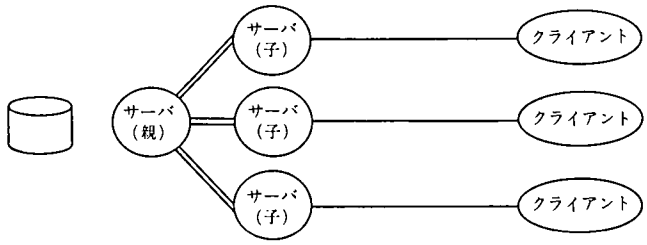
(A) クライアントからの接続時



クライアントがサーバ（親）に対し接続要求を出す．
サーバ（親）は，この接続要求に対してサーバ（子）プロセスを起動する．

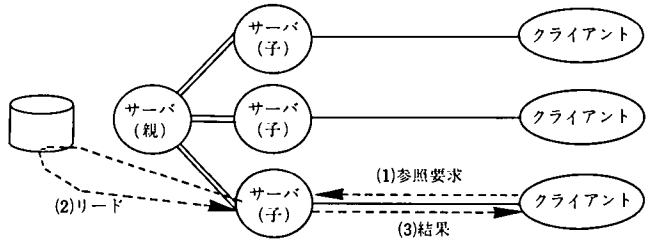
サーバ（親）－サーバ（子）間はパイプで，
サーバ（子）－クライアント間はソケットで，
それぞれ接続される．

(B) 複数のクライアントが接続されている状態



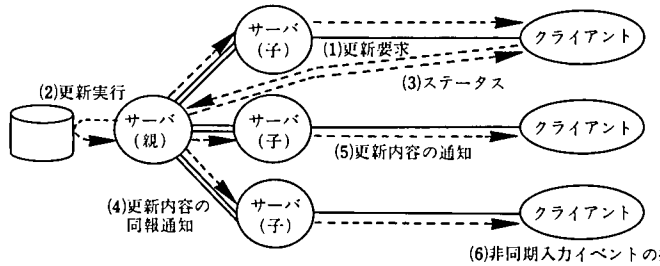
クライアントとサーバ（子）は1：1に対応する．

(C) クライアントからデータ参照要求時



クライアントからのデータ参照要求は，対応するサーバ（子）に伝えられる．
サーバ（子）は，ソケットを介して結果をクライアントに伝える．
この間の通信は同期通信である．

(D) クライアントからのデータ更新要求時



クライアントからの更新要求が発生した時は，その時点で新たにサーバ（親）とのソケットを張り，更新要求を伝える．
サーバ（親）は，実際に更新を実行し，ステータスのみをクライアントに返す．
クライアント側では，このステータスを待ち，これを受信後，ソケットを閉じる．
ここまでは同期通信である．

更新に成功した場合，サーバ（親）はすべてのサーバ（子）に対して、パイプ経由で更新内容を通知する．
サーバ（子）は，クライアントからの要求とサーバ（親）からの同報通知メッセージを監視しており，親からの入力をソケットを経由して各クライアントに伝達する．
各クライアントは，サーバ（子）からの通知を非同期入力イベントとして捕捉し，必要な表示の更新を行う．

図 1　同報通知のためのサーバクライアント構造

Fig. 1　A server-client structure for message broadcasting

EWSでの業務アプリケーションでは，ユーザ・インタフェースの実現のために，次のような方法が採用可能である．

1) コマンドインタプリタ方式のインタフェース……シェルを会話型で使用する場合のように，ユーザが逐一キーボードからコマンドを入力する方法である．エンドユーザ向きではないが，パワーユーザやシステム管理者向けのものであればこれでも良い．

2) 会語型のスクリプト……端末上にスクロールしてくるメッセージを見ながら，ユーザが処理番号等をキー入力していく方式である．パッケージ・ソフトウェアのインストールや環境設定等によく使用される．

3) キャラクタ端末ベースのフルスクリーン方式……汎用機端末のフルスクリーン画面とほぼ同様のユーザ・インタフェースである．内部的にはcurses（端末画面制御ライブラリ）を使用しているため，端末エミュレータさえ動けば端末装置の種類を選ばないのが利点である．

4) RDBMS系の画面定義ツール……RDBMSベンダーの提供する画面定義ツールを使用して，ユーザ・インタフェースを作成する方法である．データベース指向の強い業務アプリケーションには向いているが，キャラクタベースのものが中心であり，高度なユーザ・インタフェースや図形の表示等には機能不足である場合が多い．

5) GUIツールキットやXlibの使用……X Window環境下で稼働する業務アプリケーションを，GUIツールキット・ライブラリやXlibを使用して，C言語でXクライアントとして作り込む方法である．

6) パッケージ・ソフトウェアやミドルソフトウェアの利用……たとえば，非定型のデータ分析や探索的データ解析を目的とした業務アプリケーションであれば，各種の表計算ソフトウェアや統計解析ソフトウェアが利用できる．これらをEWSで使用することにより，汎用計算機に比べて優れたユーザ・インタフェースが提供できるし，扱えるデータ量や処理速度の面で，パーソナル・コンピュータ上の同種のものに比べて有利である．

また，よりマクロなレベルでGUI対応のアプリケーションを記述することを目的としたミドルソフトウェアを利用することも考えられる．

一つの業務アプリケーションの中でも，エンドユーザが日常的に使用する部分にはGUIを提供し，まれにしか使用しないような管理者向けの機能はコマンドベースのインタフェースとする等，目的や使用頻度，使用する端末，ユーザ，開発工数/予算等に応じて，これらの実現方法を柔軟に使い分けるべきである．

X Window環境下でGUIを本格的に作り込む場合，筆者等は主にGUIツールキットを用いて開発を行ってきた．GUIツールキットを使用することで，Xlibのみでの開発に比べて生産性が向上し，見た目や操作感覚の統一が図れることは事実である．しかし，GUIツールキットも機能的に十分とは言い難く，図形等の表示や操作のためにはXlibレベルでのコーティングが不可欠なのが現状である．本格的に作り込む場合の考慮点については後述する．

3.3 帳票・図面の出力

帳票や図面を出力するためのハードウェアとしては，各種のページプリンタやプロッタ等，その内容に応じて多様な出力デバイスが選択できる．最近は，ネットワークに直接接続できるタイプのものも増えてきている．

一方，ソフトウェア的には，テキストファイルをそのまま出力するのであればUNIX 標準のコマンドが使用できるが，罫線や複数種類/サイズのフォントを必要とするような帳票や，各種の図形要素を含んだ図面を出力する場合には，そのための方式を検討し，必要なソフトウェアを用意しなければならない．

多くの市販 RDBMS では，レポートライタの類が用意されているが，罫線やフォント等について制約が大きい．また，表計算ソフトウェアの中には，高度な印書機能を備えているものもある．しかし，エンドユーザ自らが必要に応じて出力をデザインするような非定型帳票の印書には向いているが，定型的/バッチ的な帳票の出力には向いているとは言い難く，きめ細かいレイアウト制御にも限界がある．

そこで筆者等は，これまでの開発で，次に示すような 2 種類の出力方式を設計し適用してきた．

その第 1 は，最大 A 3 サイズ程度の図面を，X Window とページプリンタの両方に出力するためのものである．筆者等は，

・独自の図形データ定義言語
　直線，多角形，円や文字列等の描画プリミティブや，線種，色，中塗りパターン等の描画属性に加え，名前付きの矩形領域を定義できる中間言語
・オーバレイ指示のためのファイルフォーマット

を定義し，これらを中心として，

① グラフィカルエディタ
② データアクセスおよびウィンドウ表示用ライブラリ
③ 名前付き矩形領域にアプリケーションからのデータを埋め込むためのフィルタ
④ 中間言語からプリンタ依存形式への変換フィルタ

を開発した（図 2）．

この方式は，出力デバイスに依存したフィルタ部分を書き換えるだけで，多彩なデバイスに対応できるようになっているため，たとえばカラー対応の PostScript* プリンタにも，比較的容易に対応することができる．

第 2 は，DTP (Desk Top Publishing) ソフトウェアを利用して，一般的な帳票を印書するためのものである．DTP ソフトウェアを用いて帳票の様式定義ファイルを作成し，これとアプリケーション・プログラムからのデータをマージするプログラムによって，DTP ソフトウェアの内部形式で帳票ファイルを出力する方式を実現した（図 3）．様式定義ファイルとアプリケーション・データのマージは，アプリケーション側がページの物理的レイアウトを意識しなくてもいいように，「名前付きフィールドへの文字列の流し込み」を基本としている．

この方式は，内部形式が公開された DTP ソフトウェアであれば広く応用可能であ

* PostScript は米国 Adobi Systems 社の登録商標である．

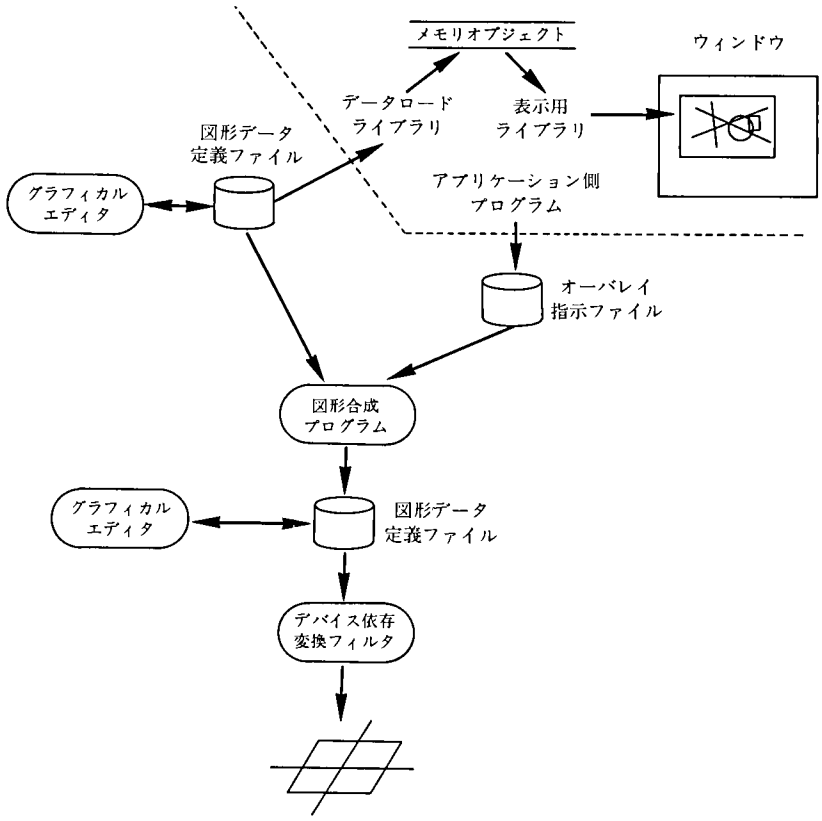


図 2　ウィンドウとプリンタへの図面出力メカニズム

Fig. 2　A figure output mechanism for X-windows and printers

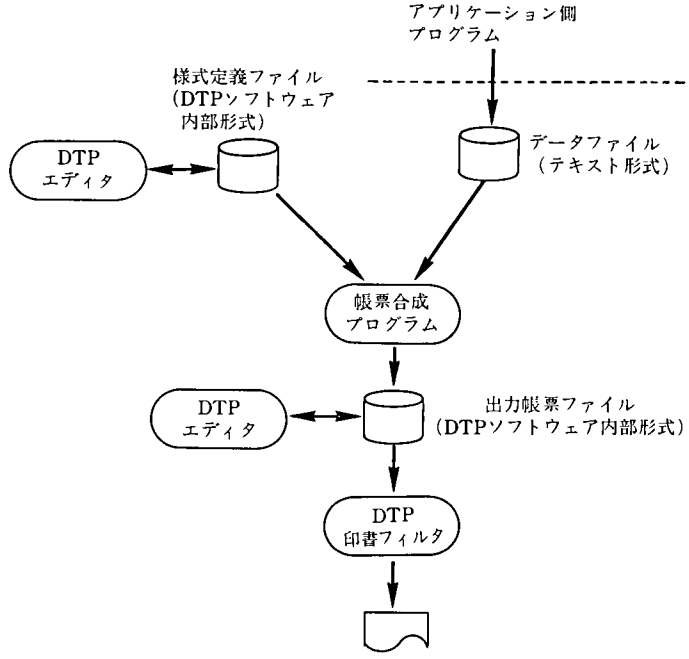


図 3　DTP ソフトウェアを利用した帳票印書のメカニズム

Fig. 3　An overlay printing mechanism with DTP software

る．また必要であれば実際の出力に先立ち，DTP ソフトウェア上で出力内容を確認し，コメント/備考等を追加することができる．

なお，最近，様々なデータフォーマットからのデータコンバージョン用フィルタを提供する DTP ソフトウェアも増えてきている．これらを利用することで，各種の業務アプリケーションからの出力を自由に組み合わせて文書を作成したり，これを電子的に保存し再利用するような，統合的な文書処理支援環境を実現できる可能性も見えてきている．

4. 本格的に作り込む場合のアプローチ

前章で述べたように，UNIX 上での業務アプリケーションでは，一つの機能を実現するための方法が何通りも存在する．また，実現可能なユーザ・インタフェースや業務アプリケーションの拡張性，パフォーマンス等に関して割り切ってさえしまえば，ある範囲の業務アプリケーションは，パッケージ・ソフトウェアを組み合わせて比較的少ない開発工数で実現することができる．

しかし，このような方式も万能ではないことに注意する必要がある．実際，パッケージ・ソフトウェアの組み合わせでは実現しにくい業務アプリケーションも少なくないし，業務アプリケーション自体に求められる機能仕様（とくにユーザ・インタフェースや出力帳票に関する要求仕様）を満たせないケースも多い．その業務アプリケーションの将来的な機能拡張まで含めて，各種のパッケージ・ソフトウェアを採用することによる制約が受け入れられるものであれば良いが，そうでない場合には，業務アプリケーションを本格的に作り込むことが必要になる．

本章と次章では，これまでの経験をもとに，UNIX/C をベースとして業務アプリケーションを作り込む場合の，筆者等の開発スタイルや注意点について述べる．

4.1 テキストファイルの活用

アプリケーション・システムで管理すべきファイルを設計する場合は，テキストファイルを活用するのがよい．UNIX の場合，通常のレコードであれば 1 行 1 レコードとし，適当なフィールドセパレータを設定する．こうすることの利点は，次のとおりである．

① C での入出力が容易
② エディタ（vi, emacs）上やダンプ出力上でのデータの確認が容易
③ 簡単な処理（形式変換・データ抽出・整形・ソート等）は，UNIX の各種標準コマンドや awk スクリプト等の組み合わせで実現可能
④ 表計算ソフトウェアや RDBMS 等へのデータ取り込みが容易

また，とくに小容量の定義ファイル類は，コメントも記述できる仕様にしておくのがよい．データの可読性が高まり，更新履歴をファイル中に残すことができるからである．

場合によっては，レコード形式ではなく，適当なデータ記述言語を設計することも検討に値する．言語方式を採用すると，構文解析が必要になる．筆者等は，構文規則が単純である場合にはトップダウンパージングを採用し，構文規則が複雑である場合には yacc を使用するようにしている．

4.2 シェルの活用

シェルは，それ自体が高度なプログラミング機能を持ったコマンドインタプリタである．処理速度に問題が残る場合もあるが，各種の標準コマンドを組み合わせて，かなり高度なデータ処理をシェルスクリプトとして実現することができる．

また，C言語等で開発した単体プログラム(コマンド)を，シェルスクリプト内で他のコマンドと組み合わせて使用することも容易であるため，アプリケーションのタイプによっては，一つ一つのコマンドをかなり細分化することができる．たとえば，ユーザからの入力が小容量であるようなバッチ処理は，パラメータ入力用の簡易なユーザ・インタフェースを提供する，汎用的で小さなGUIプログラムを用意すれば，全体をシェルスクリプトとして実現することも可能である．

一般に，シェルスクリプトで実現できることはCで実現することも可能である．通常，Cで実現した方が実行効率が良い場合が多いが，開発工数や保守性の面ではシェルスクリプトの方が有利である．データ量や処理速度に関する制約，使用頻度，開発工数等を考慮してどちらを採用するかを決めるべきである．

シェルスクリプトは，簡単な開発ツールの作成にも適しており，カスタマイズや機能追加も容易である．また，最終的なプログラムがCで実現されるような場合でも，その起動環境の設定やチェックのためのwrapperをシェルスクリプトレベルで用意することによって，環境の相違等に柔軟に対応できる．

4.3 GUIの設計とプロトタイピング

GUIを本格的に作り込む場合には，基本設計において，まずアプリケーションが提供すべき機能をすべて洗い出し，次にウィンドウの構成と，各機能に関する具体的操作方法を定義していくことになる．このとき，業務分析結果に基づいた，実業務の流れにそった設計を行うことが大切である．また，メタファ，直接操作感覚，エンドユーザの入りやすさに対する考慮も必要である．

従来のキャラクタ端末をベースとしたユーザ・インタフェース設計は，80カラム(漢字ならば40文字)×25行という，限定された画面の中での設計であり，一つの画面に取り入れる項目が決まれば，誰が設計しても大差ない画面ができあがる．しかしGUIの場合は，複数ウィンドウへの分割の方法やウィンドウ間の制御，一つのウィンドウのサイズやそのレイアウト，使用する部品の種類等，設計段階での自由度が非常に大きい．

洗練されたGUIを設計するには，従来のキャラクタ端末ベースのユーザ・インタフェースに捕らわれない発想が必要である．Macintosh*上の各種ソフトウェア等，先進的なGUIを持つさまざまなソフトウェアに触れることでセンスを磨くのがよい．

また，GUIツールキットで標準的に提供される部品の機能仕様に反するような設計を行うと，製造段階で多少の無理をすれば対処できる場合もあるが，製造工数には大きな差が出てくる．使用するGUIツールキットの思想や標準的な部品を良く理解した上で設計を行うことが望ましい．

外部仕様を詰める段階で，エンドユーザに具体的な操作イメージを理解してもらい，エンドユーザの潜在的な要求仕様を引き出すためには，GUI部分のプロトタイピング

---

* Macintoshは米国Apple Computer社の商標である．

が有効である．このためには，GUI を WYSIWYG にデザインし，GUI 部分のソースコード・スケルトンを生成することを目的とした，各種の GUI 構築ツールが利用できる[6]．

GUI に限らず，一般にプロトタイピングによって次のような効果が狙える．

① ユーザ・インタフェースの外観や操作性についてユーザ確認をとることによる開発手戻りの縮小化と，ヒアリングの効率化

② 技術的評価によるフィージビリティの確認（効率評価，技術的な実現性や機能範囲/限界等の評価）

③ ハードウェアやソフトウェアの選択および組み合わせの検討による開発工数の削減

④ 開発規模，難易度，開発方法の見通しを立てることによる開発管理の確実化

⑤ 開発要員のスキルアップ

4.4 プロセス分割

基本設計が固まった後，使用するミドルソフトウェアの評価/選択と並行して，プロセス分割を行う．ここでは，機能仕様をもとに，業務アプリケーション全体をどのようなプロセス群から構成するかを検討していく．

通常，筆者等は，次のような手順で設計を進めている．

1) まず，主要なプロセスや論理データベース，中間ファイル等を列挙し，これらの間の関係（read/write/起動/通知等）を「プロセス関連図」として 1 枚の図にまとめる．

2) プロセス関連図をもとにすべてのプロセスを列挙し，各プロセスについて，その外部仕様や実装方法等を「プロセス概要定義書」として整理する．

3) プロセス関連図やプロセス概要定義書は，設計の初期においてはプロセス分割の原案/基本方針に過ぎず，各プロセスの詳細設計が進むにつれて，何度か見直しが行われる．

4) 各プロセスは，実際にはシェルスクリプトとして実現されるか，または主に C 等のコンパイラ言語で作成することになる．C で作成する部分については，さらにプロセス内部設計へと進む．

4.5 プロセス内部設計

C で作成するプロセスには，ユーザ・インタフェースを持つものと，そうでないバッチ型のものとがあるが，ここでは X Window を前提とし，GUI ツールキット・ライブラリを使用する場合を中心に議論を進める．

多くの場合，このタイプのプロセスの論理的な内部構造は図 4 のようになる．

この図からわかるように，業務アプリケーションのプロセスは，X サーバに対するクライアントであると同時に，データベースに対するクライアントになる．

GUI ツールキットを使用し，C 言語で作成するプログラムの特徴として，次の点が挙げられる．

1) イベントドリブン型のプログラミングスタイル……アプリケーション固有の初期処理や，X サーバとの接続，ウィンドウ・インスタンスの生成と各種コールバック関数の登録を行った後，プロセスは，GUI ツールキットで用意されたメイン

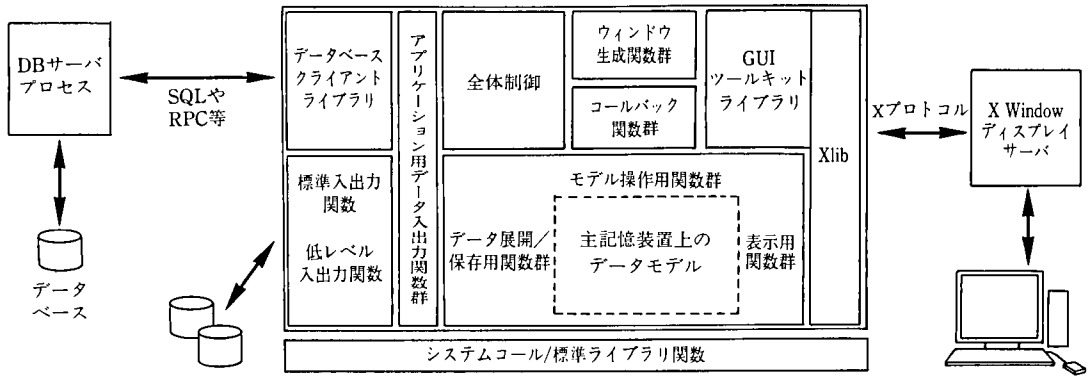


図 4　典型的なプロセスの内部論理構造

Fig. 4　A typical internal structure of a process

ループ関数をコールする．その後，ユーザの操作に応じて，メインループ関数からアプリケーション側の対応するコールバック関数が呼び出され，その中で，内部的なデータ処理やユーザに対する表示のフィードバックが行われる．コールバック関数が終了すると，再びメインループ関数に制御が戻り，次のイベントを待つ．

2)　複数ウィンドウや内部状態の制御……一つのプロセスは，一般に複数のウィンドウを使用する．ある時点でポップアップされているウィンドウの種類や，特定のウィンドウ上でのユーザの操作状態によって，ボタンやメニュ項目のアクティブ/インアクティブを制御する必要が出てくる．

また，複雑な入力制御やウィンドウ制御のために，状態遷移図に基づいたプログラミングやテーブルドリブン方式がしばしば採用される．

3)　1プロセスの開発規模……一つのプロセスのソースコードの規模は，そのプロセスが生成するウィンドウの数や各ウィンドウの構成，イベントハンドリングや内部処理の複雑度によってまちまちではあるが，数千～数万 step の規模となる．

4)　動的データ構造の使用……とくに科学技術系のプログラムでは，大容量の情報表示やデータ処理のため，大量のデータを，リスト構造や木構造，ハッシュ表等の動的データ構造を用いて主記憶上に抱え込むケースが多い．

C言語で作成するプログラムは，最終的には関数の集合として実現される．上述のような特徴を持ったプログラムを見通し良く設計/実装していくために，筆者等は，「パッケージ」という概念を導入している．

その基本的なアイデアは，簡単に言えば，プログラムをモジュールに分割することであり，ここではそのモジュールを「パッケージ」*と呼ぶ．プログラムのモジュール化は，これまでも様々な形で推奨されてきたが，われわれは，オブジェクト指向プログラミングの概念を意識したモジュール化を行っている．

プログラムをパッケージに分割する際の着眼点は，

---

*　ここでいう「パッケージ」という用語は，「パッケージ・ソフトウェア」という場合での意味ではなく，Cの標準入出力ライブラリを時に「stdio パッケージ」と呼んだり，tty 端末画面制御ライブラリを「curses パッケージ」と呼ぶ場合の意味に近い．

1)　個々のパッケージは,「それがプロセス全体の中で果たす機能や管理すべきデータは何か」という見地から分割し，設計する．

2)　「あるパッケージと，他のパッケージとの間で，どのようなコミュニケーション（インタフェース）が必要か」という点を重視する．

ことである．これは，オブジェクト指向的な見方をすれば，各パッケージを「オブジェクト」，パッケージ間のインタフェースを「メッセージ」と捉えることができる．

一つのプログラムは複数のパッケージから構成され，また，あるパッケージは複数のプログラムから（ライブラリ的に）使用され得る．パッケージとは，従来のモジュールやライブラリの概念を包含し，情報隠蔽，抽象データ型，オブジェクト指向プログラミングのアイデアを意識したカプセル化の概念である．

実開発における設計の手順は，概略次のようになる．

1)　まず，そのプロセス全体の機能をもとに，必要と思われるパッケージを列挙しながら，各パッケージが担うべき機能や管理すべきデータを徐々に定義し，パッケージ間のメッセージの流れを図にしていく．プロセスの初期化やユーザからのイベント発生をトリガーとしたメッセージの流れを丁寧に整理するのである．どのようなパッケージに分割すべきか，また各パッケージの機能やメッセージの流れが自然なものであるか，何度かレビュを行い，洗練していく．

2)　次に各パッケージについて，そのパッケージの機能や他のパッケージから受け取るメッセージをもとに，そのパッケージの公開情報（構造体や外部関数の外部仕様）を定義していく．これは，まさにヘッダファイルを記述する感覚である．実際のコーディングレベルでは，メッセージは関数呼び出しとその引数として実現される．メッセージの受け手になるパッケージが，そのメッセージを受け付けるための外部関数を用意するのである．

3)　基本的には，最終的に一つのパッケージを，一つのヘッダファイル（＊.h）と一つのソースファイル（＊.c）によって実現する．ヘッダファイルには，構造体や型，マクロ等の定義と，関数の外部（ANSI C ではプロトタイプ）宣言等，そのパッケージが他のパッケージに公開する情報が記述される．ソースファイルには，他のパッケージから利用される外部参照可能な関数群や，そのパッケージ内部でのみ利用される静的な関数群が定義される．

このようなパッケージの概念を設計に導入することの利点は，次のとおりである．

1)　あるデータ・オブジェクトは，特定のパッケージによって一元的に管理されるため，大域変数を排除することができ，保守性が向上する．

2)　製造やテストは，パッケージを単位としてほぼ独立に進めることができる．

3)　一つのプログラムは，理論的にはパッケージの集合として実現されるため，既存機能の拡張や新機能の追加が容易である．

4)　独立した機能単位として実装されたパッケージは，複数のプログラムで（ライブラリ的に）共用することが容易であり，実装済みのパッケージの組み合わせによってプログラムを仕上げるような，ビルディングブロック方式の開発スタイルも可能になる．

各パッケージを設計する場合に重要になるのは，そこで用いるデータ構造の選択で

ある．前述のように，GUI を持つプログラムでは，大量のデータを動的データ構造を用いて主記憶装置上に抱える場合が多い．パッケージ間の関連を元に，そのパッケージに対するリクエストを整理して，そのパッケージで扱うべき集合の表現に適切なデータ構造を選択する必要がある．

また，主記憶装置上のデータ構造の設計を，現実世界のエンティティとリレーションシップにできる限り忠実に行うことで，機能拡張への対応が比較的容易となる．

5. 開発の周辺

5.1 開発支援ツール

多くの文献で紹介されているため，ここでは詳しくは言及しないが，UNIX 上には，標準で各種の開発支援ツールが提供されている[7]．

また，市販の流通ソフトウェアでも，GUI 構築ツールやデバッギングツール，ソースコード解析ツール等，生産性の向上に役立つ開発支援ツールが多く存在するので，これらを目的に合わせて選択し，使用することができる．

さらに，UNIX 上での開発では，汎用計算機に比べ，既存のコマンドを自在に組み合わせた各種のツールや，一つ一つは小さいが，汎用的に使用できるコマンド等を独自に作成することが容易である．

たとえば，筆者等は，これまでの開発のうちいくつかにおいて，データベース管理ソフトウェアとして ORACLE*を使用したが，独自にテーブル定義ファイルを設計し，いくつかのツールを自作して，このファイルからのソースコード生成を試みた．自動生成の対象にしたものは，次のとおりである．

① テーブル定義/削除用の DDL SQL 文
② ORACLE データ・ローダ用の制御ファイル
③ テーブル入出力用の構造体と入出力用関数群の定義
④ テーブル一覧とテーブル記述ドキュメント

これらを生成するためのツール類は，すべて標準的な UNIX コマンド（とくに awk）の組み合わせからなるシェルスクリプトとして実現している．これらは，単にプログラミングに関する生産性の向上だけでなく，テーブル定義変更時の保守性の向上やテスト環境設定，客先インストール作業の効率化にも寄与している．

また，シェルスクリプトの中から GUI を利用するための次のようなコマンド群を用意し，子プロセスの起動やバッチ型プログラムのためのフロントエンドとして活用している．

① 警告メッセージを表示し，ユーザの選択結果を終了ステータスとして返すコマンド．
② スクリプトファイルに基づいて，ウィンドウの生成/表示/制御，子プロセスの起動およびその出力の表示を行う簡易言語インタプリタ．プロセス間のドラグ&ドロップ機能を提供している．
③ メニュ定義ファイルに従ってメニュウィンドウを生成し，ユーザからの指示によって子プロセスを起動するツール．

---

* ORACLE は米国 ORACLE 社の登録商標である．

④ 任意のタイミングで発生するアプリケーションからのメッセージ文字列を受けとり，これをウィンドウに表示するためのコマンド．

5.2 開発要員の育成

UNIX/C での大規模な開発は，ある意味では「各種ライブラリとの戦い」である．標準ライブラリ関数やある範囲のシステムコールはもちろん必須である．また，たとえば，GUI ツールキットとして XView*を使用する場合は XView と Xlib を，また，Motif**を使用する場合は Motif ウィジェットと Xt イントリンシックスおよび Xlib をそれぞれ押えておく必要がある．さらに，RDBMS を使用する場合は，採用する RDBMS の SQL インタフェースやクライアントライブラリに関する知識も必須となる．

また，シェルスクリプトについても，実行効率や保守性の高いものを実現するためには，各種標準コマンドに関する幅広い知識と，シェル自身の持つ機能に関する深い理解等，それなりの経験と技術が必要である．

このような状況下で，業務アプリケーション開発を遂行するためには，知識/情報/ノウハウの入手とその共有・継承が不可欠である．そのためには，

① サンプルソースコードの収集と解析
② ライブラリ関数の振舞いを確認するためのショートプログラムによる実験
③ 各種ツールの使用方法に関する教育
④ 開発した C ソースコードやシェルスクリプトのコードレビュ

といった地道な努力を続けることが大切である．

このように UNIX/C を前提として，業務アプリケーションを設計・開発する場合に必要となる計算機技術や知識は広範囲に及び，これを担う要員の育成は決して容易ではない．

UNIX/C をベースとしたアプローチに替わるもう一つの方法は，業務アプリケーションの要求を満たすことができる，より生産性の高いミドルソフトウェアを採用することである．たとえば，当社の提供する知的活動支援プラットフォーム TIPPLER は，GUI の記述を言語仕様に採り入れたオブジェクト指向プログラミング言語 UNISCRIPT と，UNISCRIPT を用いてアプリケーション・プログラムを構築するための諸ツール群によって，GUI 対応のビジネス系アプリケーションを，GUI ツールキットや Xlib といったレベルの知識なしで実現するための開発環境を提供している[8]．

6. おわりに

EWS/UNIX 環境における業務アプリケーションの開発について，その技術的な自由度を概観しながら，筆者等の経験や工夫，開発スタイルを紹介した．本稿の背景となった筆者等のこれまでの開発業務は，ユーザ数やネットワーク規模がある程度限定された，科学技術系/分析・計画系の業務アプリケーションが中心であり，大規模ネットワークや大量のトランザクション処理を前提とするような OLTP 系の業務アプリケーションは含まれていない．

---

* XView は SunMicrosystems, Inc. の登録商標である．
** Motif は Open Software Foundation の登録商標である．

今日，UNIXをベースとした業務アプリケーションに対するニーズは，これまで以上に高度化し，OLTP業務やマルチメディアへの対応，グループウェア等，多様なタイプのアプリケーションが要求されてきている．

また，ハードウェアの高性能・低価格化の勢いは著しく，既存ソフトウェアは頻繁にバージョンアップを繰り返し，新しいソフトウェアが次々と発表されている．プラットフォームとしても，UNIXだけでなく，NetWare*，DOS/V**，Windows***等，選択範囲が広がっている．

一方，EWS/UNIX環境に限らず，いわゆる「オープンシステム」系プラットフォームにおける業務アプリケーション開発では，「定番」と呼べる開発方法論や，どのようなタイプの業務アプリケーションにも適した「万能」のミドルソフトウェアは存在しない．

今後も，多様な実装スタイルの共存が続くと思われるが，対象となる業務アプリケーションの性質や目的に応じて，各種の方法や流通ソフトウェアを使い分け，組み合わせていく必要があろう．また，ミドルソフトウェアを採用する場合には，それらがどのような形でアプリケーションに組み込まれ，最終的にはどのようなプロセス構成になるのかを十分に見極めることも重要である．

さらに，今後のソフトウェア技術動向としては，マルチメディア技術やオブジェクト指向技術をベースとした製品やその標準化，各種のCASEツール等にも注意を払う必要がある．

本稿が，EWS/UNIXベースの業務アプリケーション構築の参考になれば幸いである．

参考文献 [1] W. R. Stevens, UNIX NETWORK PROGRAMMING, Prentice-Hall, Inc, 1990. (邦訳「UNIXネットワークプログラミング」, 篠田陽一訳, トッパン, 1992).
[2] J. Bloomer, Power Programming with RPC, O'Reilly & Associates, Inc, 1991.
[3] 増永良文, "次世代データベースシステムとしてのオブジェクト指向データベースシステム", 情報処理 Vol. 32 No. 5, 1991.
[4] Sun Microsystems Inc., ToolTalk 1.0.2 プログラマーズガイド, Sun Microsystems Inc., 1993. 2.
[5] J. Bloomer, "RPC Programming in X Applications", THE X RESOURCE ISSUE 4, O'Reilly & Associates, Inc, 1992.
[6] J. C. Armstrong Jr., "Six GUI builders face off", Sun World, Integrated Media, Inc., Vol. 5, No.12, 1992.
[7] W. B. Frakes, C. J. Fox and B. A. Nejmeh, Software Engineering in the UNIX/C Environment, Prentice-Hall, Inc, 1990. (邦訳「UNIX/Cソフトウェアエンジニアリング」, 小川晃夫訳, トッパン, 1991).
[8] 保科剛, "統合開発支援ツール TIPPLER", 技報, 日本ユニシス(株), Vol. 38, 1993.

---

* NetWareは米国NOVELL社の登録商標である．
** DOS/Vは米国International Business Machines社の商標である．
*** Windowsは米国Microsoft社の商標である．

執筆者紹介　福　地　修　一（Shuichi Fukuchi）

昭和55年 早稲田大学理工学部 応用物理学科卒業．同年日本ユニシス(株)入社．以来，汎用計算機ならびにEWSでの原子力関連アプリケーションシステム開発に従事．現在社会公共システム第一本部 公共システム部開発四課課長．

吉　野　良　成（Yoshinari Yoshino）

昭和59年 早稲田大学政経学部 政治学科卒業．同年日本ユニシス(株)入社．システム関連の社内教育に従事後，平成元年より主にEWS環境下での計画系業務アプリケーション開発に従事．現在システム技術本部 研究開発部に所属．情報処理学会会員．

C++を利用したCAIシステムの開発

The Development of a CAI System Using C++

橘田　明, 山田繁夫, 醍醐裕之

要　約　筆者らは，UNIX*のもとで利用できるCAIシステム『LearningNavigator』を開発した．あわせて利用可能な教材（コースウェア）も4種類作成した．システムの実装では品質の向上，拡張のし易さ，保守のし易さ等を目的に，また当該分野のノウハウ蓄積も兼ねてオブジェクト指向プログラミング言語C++を利用した．C++は効率，Cのスーパーセットといった長所も多いが，オブジェクトを永続的なものとして扱うためには，アプリケーション・プログラムでオブジェクトに対するメモリの割当，解放等を考慮する必要があり，プログラム作成負荷が大きくなる．そこでオブジェクトの生成，削除，2次記憶との入出力，またガーベージ・コレクション等の機能を持つオブジェクト管理ルーチン『Object Manager』を開発し，アプリケーション・プログラム作成の負荷軽減を図った．またオブジェクト識別子の形式を工夫することによって，ネットワーク環境下でも統一的にオブジェクトが扱え，クライアント/サーバ形態の利用もできるようにした．

本稿では，LearningNavigatorの概要と，Object Managerの実装方法を紹介する．

Abstract　The authors have jointly developed a CAI system named "LearningNavigator", which runs under the UNIX operating system, and newly made available four different types of courseware (teaching materials). For its implementation, the object-oriented C++ programming language was used so as to enable the "LearningNavigator" to provide higher quality, maintainability and expandability, and also make it possible for the authors to accumulate knowhow and expertise in this field of technology. Although the advantages of C++ includes its high performance and usability as the superset of the C language, adherence to object persistency enlarges loads of programming work because of the need to consider memory allotment and release for object-oriented application programs. That is why the "Object Manager" has also been developed as an object management routine which provides a generic framework for object generation, deletion, naming, primary storage management, garbage collection and so forth, so loads involved in making application programs can be lessened. In addition, a new design in the format of object identifiers helps users run object-oriented applications in a network and client/server environment.

This paper describes and overview of the "LearningNavigator" and some "Object Manager" implementation requirements.

1. はじめに

UNIX関連セミナーの受講者数およびコース数の推移を図1に示す．図からわかるように，ここ数年この分野の教育ニーズが急増している．このような状況に対応するために，平成3年9月からUNIX関連教育のためのCAIシステムおよびコースウェア（Courseware）の開発を始めた．サーバとしては弊社のUNIX機であるU6000シリーズを，学習端末にはX端末の使用を前提とした．開発ではシステムの拡張性，保

* UNIXオペレーティングシステムは，UNIX System Laboratories, Incが開発し，ライセンスしている．

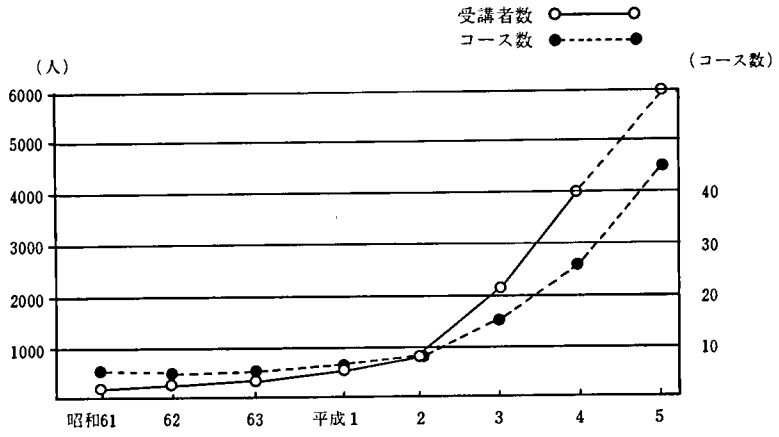


図 1　UNIX 関連セミナ受講者数およびセミナ種類の推移
Fig. 1　The change of number of students and cources on UNIX

守性等を目的にオブジェクト指向プログラミング言語C++，また一部にNIH (National Institute of Health) が提供しているクラスライブラリ[1]を利用した．平成4年5月にフェーズ1およびコースウェアとして『UNIXコマンドの使い方』，『viエディタの使い方』の開発を終了した．その後，フェーズ1システムの試行結果による機能改良，USファミリへの移植，負荷分散を目的としたクライアント/サーバ形態への対応，パフォーマンス向上，またコースウェアの追加(『シェル・プログラミング』，『C言語プログラミング』)を行い，平成5年3月フェーズ2の開発を終了した．現在このシステムを『LearningNavigator』と名付け，ユーザ教育等に利用している．

本稿では，まずLearningNavigatorの概要を紹介する．つぎにオブジェクト指向プログラミング言語C++を利用した背景，C++の特徴および課題を述べる．最後に当システムの核となるオブジェクト管理ルーチンObject Managerの実装について考察する．なおオブジェクト指向関連用語として，たとえば抽象データ型とクラス，インスタンスとオブジェクトといった同義語がいくつかあるが，本稿では対献[2]の用法に従った．

2. LearningNavigatorの概要

2.1 CAIシステムの構成

CAIシステムは一般に図2に示すような要素から構成される．ここで学習者インタフェースは，CAIシステムによって管理されている教材（解説やテスト問題等）と学習者間の表現の変換を行う．テキストばかりではなく音声，画像等も利用される．教授方略モジュールは学習者の学習状況（理解状況）に応じて，次に提示する解説やテスト問題を決定する機能を持つ．復習や質問といった学習者からの要求に対応する機能や，学習者の反応に応じた適切な診断情報（KR情報；Knowledge of Results）を生成・提示する機能も必要とされる．学習者モデル（Student Model）は学習者の理解状況を管理したもの（理解状況を示す仮説）で，オーバーレイ・モデル（Overlay Model），バギー・モデル（Buggy Model）等，多くの実現方法[3]が提案されている．

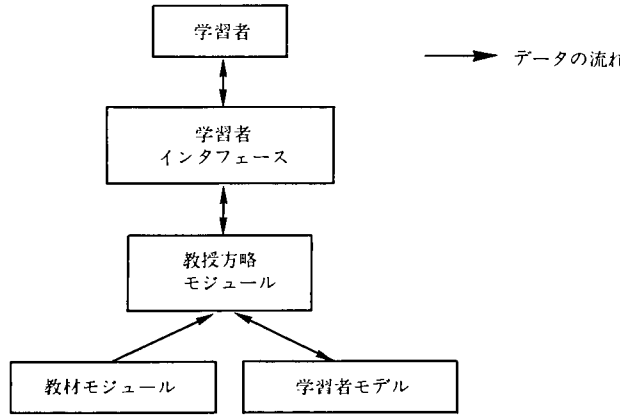


図 2　CAI システムの構成

Fig. 2　Structure of CAI system

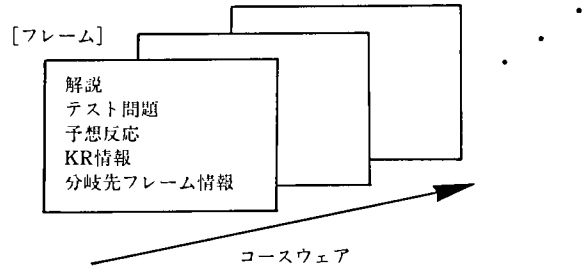


図 3　フレームの構成

Fig. 3　Structure of frame

教材モジュールは解説やテスト問題を管理する．教材として任意に検索可能な辞書，シミュレーションのためのプログラム（実習環境）等が利用される場合もある．

CAI システムはその実現様式によって，伝統的 CAI（Traditional CAI）システムと知的 CAI（Intelligent CAI）システムの二つに分類することができる[4]．伝統的 CAI システムを AFO 型（Add-hoc Frame Oriented）と呼ぶ場合もある．知的 CAI システムは，人工知能研究の成果として得られた技術を利用して，より人間教師に近い振る舞いができる CAI システムの開発を目標としている．そのための要素技術もいくつか提案されているが，まだ研究・開発段階のものがほとんどで，情報処理教育分野で実用化されているものはない．現在利用されているものはほとんど AFO 型の CAI システムである．

AFO 型 CAI システムは図 3 に示すように，解説やテスト問題，その予想反応，反応に応じた KR 情報や分岐先（フレーム）情報等を一つのデータとして管理する．このデータをフレーム（Frame）と呼ぶ．フレームの列をコースウェアと呼ぶ．学習者の反応に応じた簡単な分岐制御を行うこともできるが，通常フレームを順番に提示することによって学習を進める．したがって，教授方略はあらかじめフレームの列として固定されているとみなしてよい．また学習者モデルは学習済みフレーム群と未学習フレーム群によって表現される．簡単な仕組みではあるが，使いやすい教材作成支援シ

ステムが用意されている場合には比較的容易にコースウェアの開発ができるという長所を持つ．しかし，すべての学習者に対して画一的な学習となること，また同一フレーム内にテスト問題および反応に応じた分岐先フレーム情報を含むことによって，たとえばテスト問題を変更することによって分岐先フレームも変更しなければならないといった保守性の低下等が課題となる．

2.2 LearningNavigatorの構成と機能

2.2.1 教材モジュール

教材モジュールは図4に示すように，学習目標ネットワーク，解説，テスト問題，例題，および用語辞書から構成される．学習目標は学習内容とその達成レベル（行動目標）によって規定される．一般に学習目標間には前提と目標の観点から関連が定義できる．学習目標ネットワークでは，学習目標の記述および目標間の関連が定義されている．解説，コマンドやCの関数等の使用例，また到達度を診断するためのテスト問題等が各学習目標に対応して管理されている．重要な用語に対する辞書も各コースウェアごとに用意し，任意の時点で参照することができるようにした．さらに，解説や例題で学習した内容をその場で確認できるよう実習環境も用意した．必要に応じて実習用のウィンドウを開き，UNIXやviエディタのコマンドが入力できる．誤ってシステムのファイル等を削除してしまわないよう，シェルのシミュレータを開発しそれを利用している．また，Cのプログラミング学習を支援するためのCインタプリタも開発した．学習者は作成したプログラムをコンパイル，リンクといった操作を経ずにそのまま実行できる．このインタプリタは任意の行でプログラムの実行を中断し，変数値の表示や変更を行い，プログラムを再実行することもできる．また，インタプリタが出力するエラー・メッセージも工夫した．

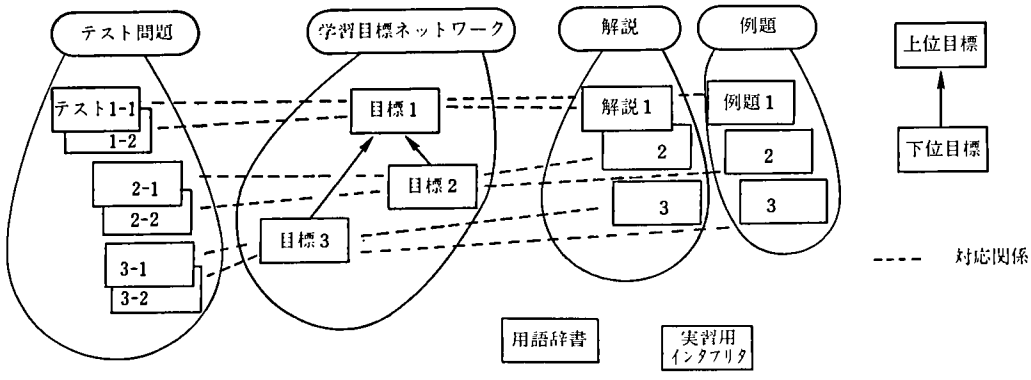


図4　教材モジュールの構成

Fig. 4　Structure of learning materials

2.2.2 教授方略モジュール

CAI本来の目的である学習者の理解状況に応じた個別学習環境を実現するためには，学習者の理解状況に応じて，動的に次の学習目標を決定する機構が必要となる．またある程度の前提知識を持った学習者に対しては，前提の程度に応じた学習目標の選択機能も必要となる．また意欲のある学習者の場合には，システムではなく，学習

者自身が学習目標を決定しながら学習を進めた方が効果がある場合も考えられる．

LearningNavigatorでは，システム主導および学習者主導型の学習が可能な教授方略を採用した．システム主導型学習では，次に学習すべき学習目標をシステムが決定し，対応する教材を提示する．学習者は必要に応じて，用語辞書や例題を参照したり，コマンドあるいはC言語インタプリタ用ウィンドウを利用して，学習内容を実習しその確認を行う．該当する学習目標に対する学習が終了した時点で，テスト問題が提示される．その反応に応じた到達度診断・評価を行い次の学習目標が決定される．学習順序は学習目標ネットワークを利用して，教材の内容とは独立した学習目標決定モジュールが行う．目標決定アルゴリズムは，沼野[5]の「基礎性/応用性」を基本とした．またプリテストによって，学習者ごとのレディネスを診断し，学習者の前提知識に応じた学習もできるようにした．

学習者主導型学習では，ウィンドウに表示された学習目標のネットワーク図を使い，任意の学習目標をマウスによって選択し，対応する教材やテスト，例題等を参照しながら学習を進める．システム主導と学習者主導は，任意の時点で切り替えることができる．

2.2.3 学習者モデル

学習者モデルは，各学習目標ごとの達成/未達成情報によって理解状況を表現するオーバーレイ・モデルを採用した．KR情報はテスト問題への反応，テスト問題の選択肢に与えられたウェイト，実施回数等を利用して自動生成するようにした．

2.2.4 学習者インタフェース

学習者インタフェースはMotif*によるGUIとした．解説や例題，用語辞書の参照，また実習による確認等が同一の端末で行うことができ，効果も期待できる．マウスを用いた操作が可能であり操作性も高い．図5に学習者インタフェースの例を示す．

2.3 C++採用の背景

本システムの開発は二つのフェーズに分けて実施した．フェーズ1は社内研修での利用を目的に，フェーズ2ではその結果に基づいた機能改良や機能追加を行うためである．そのため実装に当たっては，システムは拡張性や保守性を主要な目標とした．システムの仕様について一部未定の部分があり，パフォーマンス等についても机上では判断できない部分があった．開発メンバは新人が多く，OJTを兼ねた開発プロジェクトといった側面があった．このような背景からモジュラ・プログラミングやプロトタイピングといった開発を目的にオブジェクト指向プログラミング言語C++を使用した．また当該分野のノウハウ蓄積も目的の一つであった．

オブジェクト指向プログラミングは，データと手続きをカプセル化することによってモジュールの独立性を高め，しかもそのモジュールを汎用的かつ高品質な部品ととらえ，部品を組み合わせることによって信頼性の高いソフトウェアを作成することを目的とする．また機能改良や機能追加は，部品の改良や部品の追加で対応し，それに伴う影響を局所化しようという技術である．オブジェクト指向プログラミング言語としては，弊社のTIPPLER/UNISCRIPTやゼロックス社によって開発されたsmal-

* Motifは，Open Software Foundationの登録商標である．

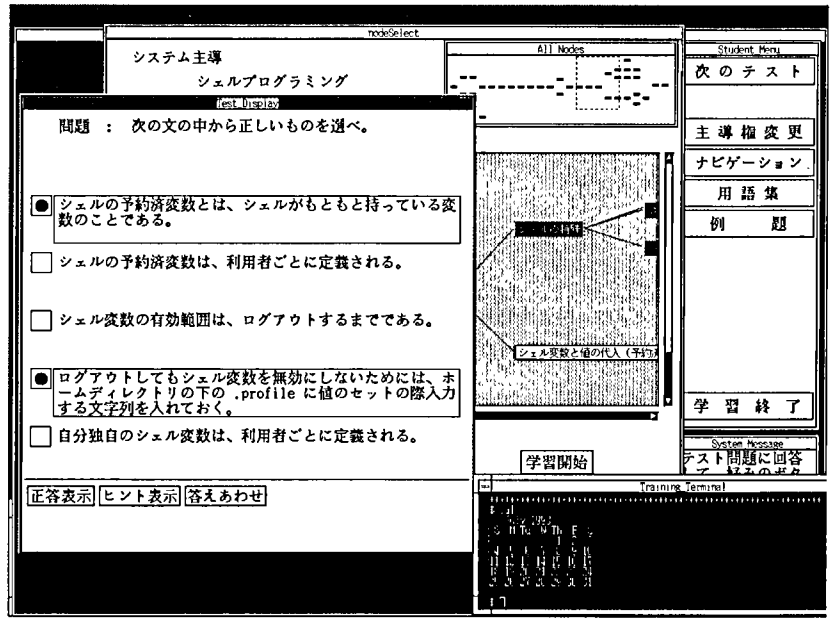


図 5　学習者インタフェースの例

Fig. 5　An example of student interface

ltalk 等がよく知られているが，U 6000 シリーズで使用できることが必須条件であったため C++を採用した．

図 6 に LearningNavigator で定義したクラスおよびその階層を示す．ここで，各クラスに共通のデータメンバおよびメンバ関数は基底クラス Generic を設定し，そこに定義した．Generic に定義されたデータ・メンバにはオブジェクトに対して一意に割り当てられる識別子(Object Identifier；以降 OID と略記する)，またオブジェクトの生成，削除等を行うためのメンバ関数等が定義されている．

学習者クラスには学習者の属性(氏名，コンピュータ経験有無，……)，コースごとの学習状況(学習済み，学習中，未学習)，また学習中のコースについては現在到達している学習目標オブジェクトの OID といったデータ・メンバが定義されている．コース・クラスにはコース名，コースの標準学習時間，関連する用語オブジェクトの OID 等が定義されている．用語辞書クラスでは用語名，用語の説明等のデータ・メンバま

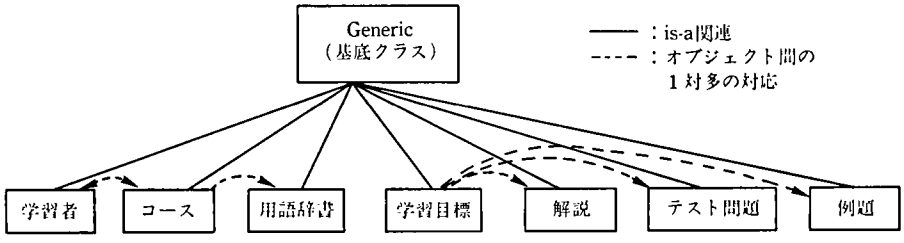


図 6　クラスとその階層

Fig. 6　Hierarchy of classes

た用語の説明を表示するためのメンバ関数等が，学習目標クラスでは目標の記述，前提となる学習目標オブジェクトのOID，対応する解説，例題，テスト問題オブジェクトのそれぞれOID，また学習目標決定手続き等のメンバ関数，解説および例題クラスには解説や例題の記述，教材や例題の表示を行うためのメンバ関数等が定義されている．テスト問題クラスでは問題の記述，選択子，正答，学習者の反応といったデータ・メンバや診断，KR情報生成，テスト問題表示，反応入力といったメンバ関数が定義されている．

2.4 C++とオブジェクト指向プログラミング

C++は1980年，AT&TのB. Stroustrupによって開発された．当初クラス定義が可能なC (C with classes) として位置付けられていたが，1983年C++と命名され，1985年AT&Tから正式にリリースされた．

C++は，Cで提供される機能のほかに，クラス定義，派生クラスの利用，継承機能，データと手続きのカプセル化，メッセージ・パッシングといったオブジェクト指向プログラミング言語としての機能を持つ．実行効率もよい．

しかし，課題もある．たとえばC++のみでソフトウェアの開発を行おうとすると，リストや配列といった基本的なデータ型に対応するクラスを使用者が定義しなければならず負荷も大きい．オブジェクト指向プログラミングではクラスを部品ととらえる．基本的な部品群があらかじめ用意されていれば，ソフトウェア開発の負荷を軽減することやソフトウェアの品質向上が期待できる（一般にこのような部品群をクラスライブラリと呼ぶ）．今回の開発ではC++で記述されているNIHクラスライブラリを利用した．NIHのクラスライブラリ（以下NIHCLと記す）は米国National Institute of Healthによって開発され，パブリック・ドメインとして提供されているライブラリで，C++でオブジェクト指向プログラミングを行うためのクラス群を定義している．たとえば文字列(String)，日付(Date)，列(OrderedCltn)，集合(Set)，リスト(List)等の基本的なデータ型に対応するクラスが，またプロセス（Process)，スケジューラ(Scheduler)，キュー（Queue）といったプロセスの制御に利用できるクラスも定義されている．

3. LearningNavigatorの実装

LearningNavigatorのモジュール構成を図7に，各モジュールの機能を以下に示す．

1) View Manager……View Manager（以降VMと略記する）は学習者とのインタフェースを支援するモジュールで，Motifのツールキットを用いて作成されている．各種ウィンドウおよびボタンを学習端末に表示し，学習者からのボタン操作に対応するイベント・ハンドラによってオブジェクト管理ルーチンであるObject Managerにメッセージを送信する．またObject Managerの要求に応じてデータをウィンドウに表示する．VMは学習者ごとに起動されるプロセスで，Object Managerとはプロセス間通信によって交信される．

2) Object Manager……Object Manager（以降OMと略記する）は以下に示す機能を持つ．

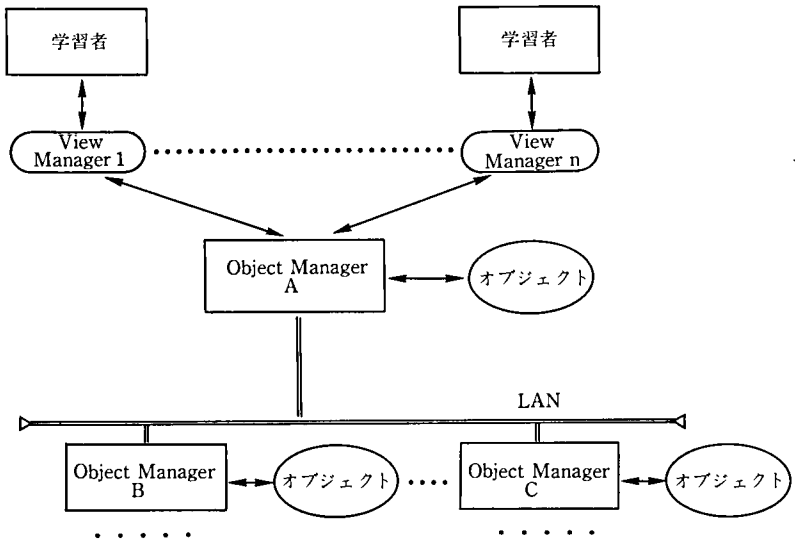


図 7　LearningNavigator のモジュール構成

Fig. 7　Modules of LearningNavigator

① オブジェクトの生成，初期化および識別子（OID）の割当
② オブジェクトの2次記憶への入出力
③ オブジェクトの削除
④ オブジェクト間のメッセージ・パッシング
⑤ VM との交信
⑥ 他サーバ機上の OM との交信
⑦ 複数の学習者からオブジェクト共同利用を可能とするためのスレッド制御

OM はサーバごとに起動されるプロセスで，学習を始める前にあらかじめ起動しておく．OM が起動されると，すべてのオブジェクトが2次記憶からメモリ上（OM のプロセス空間内）にロードされる．ただし，このときロードされるオブジェクトには学習者に依存するもの（学習目標ごとの到達状況やテスト問題に対する反応等）は含まれない．学習者に依存するデータを管理したオブジェクトは，学習を開始したとき学習者ごとにロードされる．また OID は，クライアント/サーバの環境下でもその一意性を保障するために図8に示す形式とした．

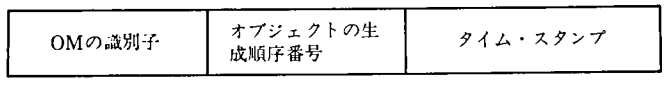


図 8　OID の形式

Fig. 8　Format of object identifier

4. OM の実装について

OM は C++では提供されていない機能を組み込んだモジュールで，データベース管理システムに相当する．また OM と VM，OM と OM 間の相互作用をプロセス間通

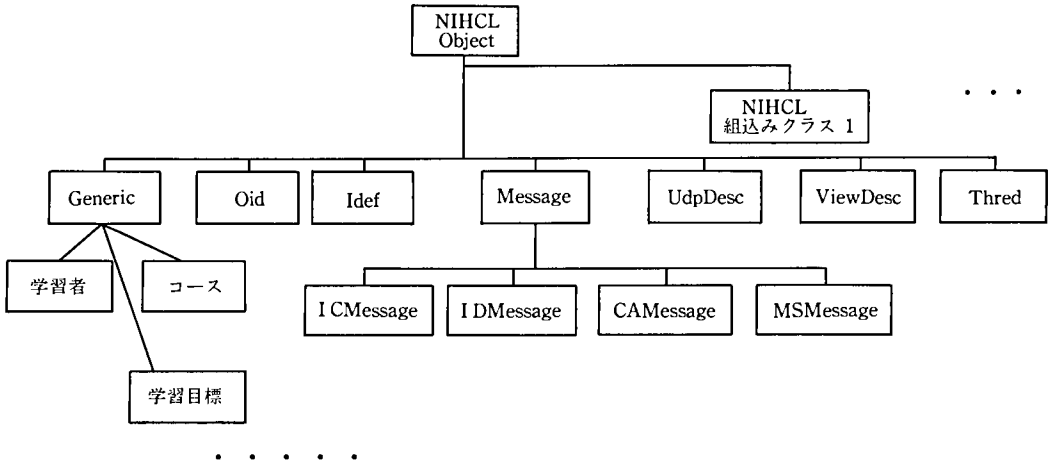


図 9　OM で定義されているクラス

Fig. 9　Classes using in Object Manager

信によって実現し，たとえばVMとOMをネットワークを介した別の計算機上で利用するといった，クライアント/サーバ型の運用も可能としている．

このような機能を持ったOMの実装方法をCAI関連クラス(学習者，コース，学習目標，……）とのインタフェース，プロセス間通信，マルチスレッド制御の観点から説明する．

4.1　CAI関連クラスとのインタフェース

OMの実装にあたり図9に示すようなクラス群を定義した．OMの実装に必要なクラスであり，CAI関連クラスと区別するためこれ以降OMクラスと呼ぶ．CAI関連クラスとのインタフェースに利用しているOMクラスには以下のものがある．

4.1.1　Genericクラス

NIHCLのObjectクラスからの派生クラスで，CAI関連クラスの基底クラスとなっている．CAI関連のクラスは，必ずこのクラスの派生クラスとして定義しなければならない．GenericにはOMとのインタフェース用メンバ関数が定義されている．CAI関連のオブジェクトは，このメンバ関数を介してOMの機能を使用する．OMはオブジェクトへの参照をOIDで行う．他のオブジェクトのメンバ関数を呼び出す場合には，そのオブジェクトのOIDを引き数として，OMの規定する手続きを呼び出す．Genericクラスには，以下のメンバ関数が定義されている．

1)　iCreate……オブジェクトの生成を要求する関数で，OMは該当するCAI関連クラスのオブジェクトを生成し，OIDを割り振る．オブジェクトは次に述べるIdefクラスのオブジェクトとして管理され，以後OIDでアクセスされる．
2)　iDelete……OMに対してオブジェクトの削除を要求するメンバ関数
3)　Call……オブジェクトのメンバ関数を呼び出す関数で，返り値を要求するとき，あるいは同期的な処理を必要とする時に使用する．OMは関数呼び出しが終了するのを待ち，その返り値を呼出し側のオブジェクトに返す．
4)　Msg……オブジェクトのメンバ関数を呼び出す手続きで，返り値を要求しない

とき，あるいは非同期な処理を必要とする時に使用する．OMは関数の実行が終了するのを待たずに，制御を呼出し側に戻す．

5) attach……VMとのコネクション確立を要求するメンバ関数

6) detach……VM間のコネクションを解除するためのメンバ関数

4.1.2 Oidクラス

OIDはオブジェクトに与えられる識別子で，システム内（ネットワーク内）で一意である．クラスOidにはOIDの値を保持するためのデータ・メンバ，またOID決定のためのメンバ関数が定義されている．CAI関連オブジェクトが生成されると，そのOIDを管理するオブジェクトが生成される．

4.1.3 Idefクラス

クラスIdefには，オブジェクトのOID，参照時刻，アドレス，2次記憶上のファイル名，ファイルのオーナおよびグループID等のデータ・メンバ，また2次記憶に対する入出力のためのメンバ関数が定義されている．オブジェクトが生成されるとクラスIdefのオブジェクトも生成され，オブジェクトのアドレスをIdefのオブジェクトに登録する．検索ではOIDにより該当するIdefのオブジェクトからCAI関連オブジェクトのアドレスを得る．もし，オブジェクトがメモリ上にない場合には2次記憶から読み込まれ，そのアドレスが返される．Idefクラスには，一定時間以上参照されないオブジェクトはメモリから削除し，更新されたオブジェクトについては2次記憶に書き戻すためのメンバ関数が定義されている．

4.1.4 ViewDescクラス

VMを操作するためのクラスで，ウィンドウの識別子を用いてattach関数を呼び出すとViewDescクラスのオブジェクトが生成される．その中で定義されている関数を用いてVMに対する操作が可能となる．detach関数によってViewDescクラスのオブジェクトは削除される．

4.2 プロセス間通信

関数CallおよびMsgを使用する場合に，プロセス空間内に該当するOIDを持つオブジェクトが存在しない場合，OMはネットワーク全体にそのOIDを持つオブジェクトが存在するかどうか問い合わせる．これにはUDPのブロードキャスト・メッセージが用いられる．OMはオブジェクトの検索要求およびOIDをUDPのパケットを用いてネットワークに送り出す．検索要求を受けた他サーバ上のOMは自分のプロセス空間をまず検索する．該当するOIDのオブジェクトが存在すれば，その返答を要求を出したOMに返す．OMは返答のあった他サーバ上のOMにメッセージを送り，自分の処理を継続する．メッセージが送られたOMはスレッドを生成し，該当するオブジェクトに制御を渡す．一度呼び出された他サーバ上のオブジェクトの存在場所はキャッシュされ，それ以降の参照ではブロードキャスト・メッセージは出されない．このプロセス間通信のために以下のようなクラスを定義した．

4.2.1 Messageクラス

Messageクラスにはデータ・メンバとしてクライアントのIPアドレスを，また仮想関数（Virtual Function）Do_Messageを定義している．このクラスから子クラスを導出し，必要なデータ・メンバおよび関数を実装することによって，OMとOM，OM

とVMといった異なるプロセス間の通信を行うことができる．この目的で以下の四つの派生クラスを定義した．

1) ICMessage……CAI関連オブジェクトを生成する時に使用する．生成するオブジェクトのクラス名とそのデータ・メンバを引き数とし，OIDを返す．
2) IDMessage……オブジェクトを削除する時に使用する．削除するオブジェクトのOIDを引き数とし帰り値は無い．
3) CAMessage……CAI関連オブジェクトのメンバ関数を実行する．オブジェクトのOIDとメンバ関数名を引き数とする．関数の帰り値を得ることができ，処理は同期的となる．
4) MSMessage……オブジェクトのメンバ関数を呼び出す場合に使用する．オブジェクトのOIDと関数名を引き数とする．関数実行の帰り値は得られないが，非同期の処理が可能となる．

4.2.2 UdpDescクラス

OMとOM，OMとVM間の通信を行うためのUDPポートとパケットおよびサービスを定義している．サービス名をパラメータとしてUdpDescのオブジェクトを生成すると，サービス用のパケットが生成され通信が可能となる．

4.3 マルチスレッド制御

OMはイベント駆動型のプロセスであり，外部および内部からの要求に対しスレッドを生成し，そのスレッドを順次実行する．OMは一つのUDPポートと一つのTCPポートを用意し，OMのメインループはselect( )システムコールでイベントの到着を待っている．UDPまたはTCPのメッセージが届くことによりOMの処理は開始され，スレッドが生成され，オブジェクトのメンバ関数が呼ばれる．スレッドはその処理が終わると消滅するが，オブジェクトのメンバ関数の持つ副作用でいろいろな処理を行うことができる．たとえば，iCreateを呼び出せば，新たにオブジェクトが生成され，またattachを呼び出すことによって端末上にウィンドウを開くことができる．他のオブジェクトのメンバ関数を呼び出す場合にはCallやMsgを用いる．マルチウィンドウ制御のために以下のクラスを定義した．

・Threadクラス

OMは常に一つのUDPポートと一つのTCPポートに対して要求を待っている．ポートに処理要求が届くと，Threadクラスのオブジェクトが生成される．このオブジェクトが一つのスレッドとして動作する．要求が発生するとOMはThreadクラスのオブジェクト（スレッド）を生成し，OMのスケジューラに登録する．スケジューラは順番にスレッドに制御を渡す．スレッドはライトウェイト・プロセス（一つのプロセス空間内で非同期に動作するプロセス）として動作する．したがってスレッドごとに非同期の処理が行われ，複数の要求に対応することができる．処理が終わるとThreadクラスのオブジェクトは削除される．Threadクラスの実装には，NIHCLで提供されているクラスScheduler, Process, SharedQueue等を利用した．

5. おわりに

OMを作成することによって,CAI関連クラスの定義が簡単になった.またNIHCLを利用することによって,高品質の部品(クラス)が利用でき,ソフトウェアの品質,生産性も向上できたと思われる.システムの拡張性,保守性向上という所期の目的も達成できたと考えている.本稿では述べていないが教材作成者のためのツールとして図形エディタ,またコースウェア「C言語プログラミング」の実習用にC言語インタプリタも作成した.いずれもC++で実装した.これらの実装方法についてもまた別の機会に報告させていただきたい.

C++は処理効率,C言語のスーパセットといった長所を持つプログラミング言語である.今回の実装で使用したAT&TのC++の他にもGNU, TURBO C++等多くの処理系も提供されており,使用するための環境は整いつつある.しかしながら今回の開発で痛感したことであるが,NIHCLのようなクラスライブラリを利用せずに,部品としてのクラスをすべて自作するのはかなりの労力を必要とすると思われる.充実したクラスライブラリの利用をおすすめしたい.

最後に,開発方法について述べておきたい.ノウハウの蓄積という目的もあって,C++, MotifまたはNIHCLといった未経験の技術を使用した.スケジュールとの関係から,これらの技術のスタディと開発を並行して行わざるをえなかった.結果的に試行錯誤をしながらの開発となった.機能を確認するためにプロトタイプを作成し,評価し,プロトタイプの修正,機能拡張を繰り返すという手順である.たとえば当初,オブジェクトはコースごとに一つのファイルに保存した.しかし十分なパフォーマンスが得られないことがわかり,結果的に1オブジェクト/1ファイルとした.設計段階でこれを予測するのは困難であった.また予期せぬ問題も多発した.たとえば,Motifツールキットと NIHCL での名前の衝突(object, this 等),またC++トランスレータによるシンボル・テーブルのオーバーフロー等,このような観点からも経験のない技術の利用にはプロトタイピング的なアプローチが重要と考える.

参考文献 [1] Keith E. Gorlen, Sanford M. Orlow, Perry S. Plexico, DATA ABSTRACTION AND OBJECT-ORIENTED PROGRAMMING IN C++, John Wiley & Sons, 1990.
[2] U 6000 シリーズC++解説書, 日本ユニシス, 1991.
[3] 溝口理一郎, 角所　収, 知的CAIにおける学習者モデル, 情報処理, Vol. 29, No. 11, pp. 1275～1282, 1988.
[4] 大槻説乎, 山本米雄, 知的CAIのパラダムと実現環境, 情報処理, Vol. 29, No. 11, pp. 1255～1265, 1988.
[5] 沼野一男, 授業の設計入門, 国土社, 1987.

執筆者紹介 橘　田　　明 (Akira Kitta)

昭和 49 年横浜国立大学工学部生産工学科卒業. 同年日本ユニシス(株)入社. 教育部の AI 教育担当, UNIX 教育担当を経て, 現在, 教育開発部所属.

山　田　繁　夫 (Shigeo Yamada)

昭和 58 年日本ユニシス(株)入社. エキスパートシステム, 知的 CAI 等の開発を経て, 現在 UNIX をプラットホームとする CAI の開発に従事. IEEE 準会員.

醍　醐　裕　之 (Hiroyuki Daigo)

平成 3 年日本大学文理学部社会学科卒業. 同年日本ユニシス(株)入社. UNIX 上の CAI の開発に従事, 現在, オープン・システム・サービス本部 教育システム部に所属.

オブジェクト指向によるGUIクラスライブラリの開発

The Development of GUI Class Libraries Using Object-oriented Technology

伊藤直行, 熊本厚志

要　約　われわれのグループではUNIX*のXウィンドウ**上で動作するグラフィカル・ユーザ・インタフェース（GUI）のための汎用的なライブラリ（Xmpp, Mui）をオブジェクト指向に基づいて開発した．両ライブラリは，われわれが現在開発しているアプリケーションプログラムのために開発したもので，GUIで必要とされる基本的な機能を持つ一連のクラスを提供している．アプリケーション開発者は，これらの基本クラスを組み合わせることによってGUIを構築することができる．

また，ライブラリの提供している基本クラスだけでは機能的に不足がある場合は，基本クラスの下に子クラスを作成して独自の機能部分だけを追加することにより，基本クラスの機能と独自の機能を併せ持つ新たなクラスを作成することが可能である．

GUIに使われるウィンドウはボタン，ラベル，アイコン等の多くの部品から構成されている．Xmppライブラリは，これらの部品をクラスとして定義したものである．一方，Muiライブラリは任意のデータ構造をオブジェクトとして表現し，取り扱う機能を持ったライブラリである．本稿ではMuiの特徴，機能を中心に両ライブラリの開発について報告する．

Abstract　With the use of object-oriented technology, the authors have developed generalized software libraries (called Xmpp and Mui) designed to be used for creating graphical user interface (GUI) programs that run on the UNIX X window system. Both libraries, which have been developed for the benefit of the application programs now under development by the authors, provide a set of classes equipped with basic GUF functionalities, thus enabling applications creators and programmers to develop GUI programs through any combination of those basic classes. If they discover a functional deficiency in the classes provided only by the libraries, then they are allowed to create a new subclass under a basic class, to which required functions of their own can be added, so a new class can be made where both basic and customized operations are jointly present.

Windows used in GUI programs consist of several items such as button, label, icon, and so on. Xmpp is a library that defines these items as classes, while Mui is a library capable of representing and handling all data structures as objects. This paper is intended to report the development of the two libraries with a special focus placed on the main features and functions that Mui has made available.

1. はじめに

われわれのグループでは，分散システムを運用管理するためのシステムの開発をUNIX上で行っている（以下，運用管理システムと呼ぶ）．この運用管理システムは，分散システムを視覚的に表現した図1のようなGUI（グラフィカル・ユーザ・インタ

* UNIXオペレーティングシステムは，UNIX System Laboratories, Inc.が開発し，ライセンスしている．

** Xウィンドウ（X Window System）は米国MITの登録商標である．

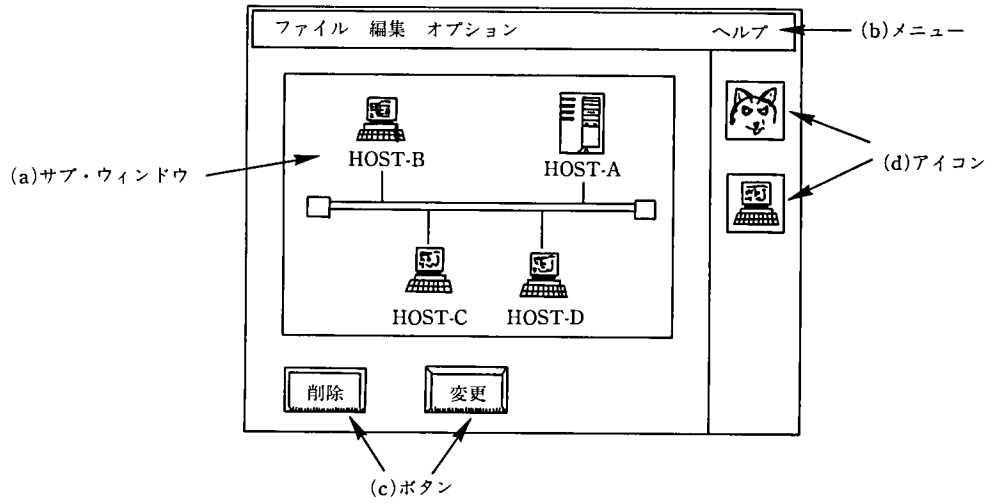


図 1 ウィンドウの例
Fig. 1 Sample window

フェース）の操作画面（ウィンドウ）をシステム管理者に提供する．システム管理者はこのような操作画面を通して，ネットワークの障害状況を把握したり，ネットワークの構成を変更する（ネットワークにホストマシンを追加したりする）ことが可能になっている．

この運用管理システムは最終的には数多くの操作画面を持つことになり，それらの開発に要する工数が非常に多くなってしまうという問題がある．また，大人数による開発のため，同じような操作画面を開発する場合でも，担当者ごとにソースコードの内容がかなり異なったものになってしまい，ソースコードの再利用がむずかしいという問題も考えられる．

そこで，われわれは運用管理システムの開発に先立ち，標準的な操作画面，機能を提供する GUI ライブラリの開発を行うことにした．ライブラリの設計，開発はオブジェクト指向に基づいて行った．なぜなら，オブジェクト指向の三つの特徴，データ抽象，ポリモーフィズム，継承を活かすことによって，ライブラリの利用者（開発者）にとって，①ライブラリの提供するクラスの下にサブクラスを作成し，独自機能を実装することにより，機能の拡張が容易に行える(継承)，②ライブラリに関して，わずかな知識を持っているだけでライブラリの提供する機能を利用できる（データ抽象，ポリモーフィズム)，③一部のオブジェクトの内容に変更が発生しても，他のオブジェクトに影響を与えない(データ抽象)，等の利点があると考えたからである．また，業界の標準動向として，分散システムの運用そのものがオブジェクト指向に基づいているため，運用管理システムに馴染みやすいという利点もある．

ライブラリとして，Xmpp ライブラリ，Mui ライブラリの二つのクラスライブラリを開発した．Xmpp ライブラリは操作画面（ウィンドウ）の構成部品（ボタン，メニュー等）をオブジェクトとして定義したものである．一方，Mui ライブラリは任意のデータ構造をオブジェクトの集合として捉え，ウィンドウ上に視覚的に表現し，操作

する機能を持ったライブラリである．標準的なウィンドウを作成する場合であれば，開発者はライブラリの提供するクラスからインスタンスを生成して，組み合わせるだけですむ．

本稿ではMuiライブラリの特徴，機能を中心に両ライブラリの開発について報告する．

2. Xmppライブラリの概要

運用管理システムのGUIはOSF/Motif*（以下Motif）をベースに開発を行うことにしている．事実上のUNIXの標準ウィンドウシステムであるXウィンドウシステムの環境の統一を目的としたライブラリの一つがMotifである．このライブラリを使用することによって，標準化されたウィンドウの外観，操作を持つGUIを開発することができる．

運用管理システムが表示するウィンドウは前出の図1のようなものであるが，このようなウィンドウは実際には一つ以上の部品から構成されている．図1のウィンドウは(a)サブ・ウィンドウ，(b)メニュー，(c)ボタン，(d)アイコンから構成されている．Motifは概念的にはオブジェクト指向に基づいて作られており，これらウィンドウの構成部品をオブジェクトとして捉え，それらの特徴をクラスとして定義している．したがって，開発者はそれらのクラスのインスタンスを生成し組み合わせることによってウィンドウを作成することが可能になっている．

しかし，MotifはC言語で実装されているため，前出のオブジェクト指向の三つの特徴のうちの継承しか実現されておらず，それも不完全なものである．そのため，インタフェースがオブジェクト指向的でない，常に変数のデータ型を意識してコーディングをしなければならない，機能を拡張したサブクラスを作成することが容易でない，等の問題がある．

Xmppライブラリは，これらの問題を解決するためにオブジェクト指向言語C++を使用してMotifを再構築したものである．図2にXmppのクラス階層図を示す．これは基本的にはMotifのクラス階層に従ったものであるが，Motifでは概念的にしか実現されていなかったものを実際のクラスオブジェクトとして実現しており，各関数もクラスのメソッドとして実装している．このことにより，オブジェクト指向の三つの特徴を実現し，前出のようなMotifの問題点を解決している．

また，XmppはMotifの提供していなかった多くの機能を提供している．たとえば，Motifではアイコンは矩形のものしか実現できないが，Xmppでは図3のような非矩形アイコン，この例ではワークステーション型，を作ることも可能にしている．また，Motifではウィンドウの背景は常に白紙であるが，Xmppでは背景に画像イメージを貼り付けることも可能にしている（図4）．また，アイコンの反転（前景色と背景色の入れ替え）等の頻繁に使用される機能も一つのメソッドで簡単に実現できるようにした．

以上のようにXmppはMotifの問題点を解消すると同時に，多くの便利な機能を簡単に利用できるように実装し，開発者の負担を軽減している．

---

* Motifは Open Software Foundationの登録商標である．

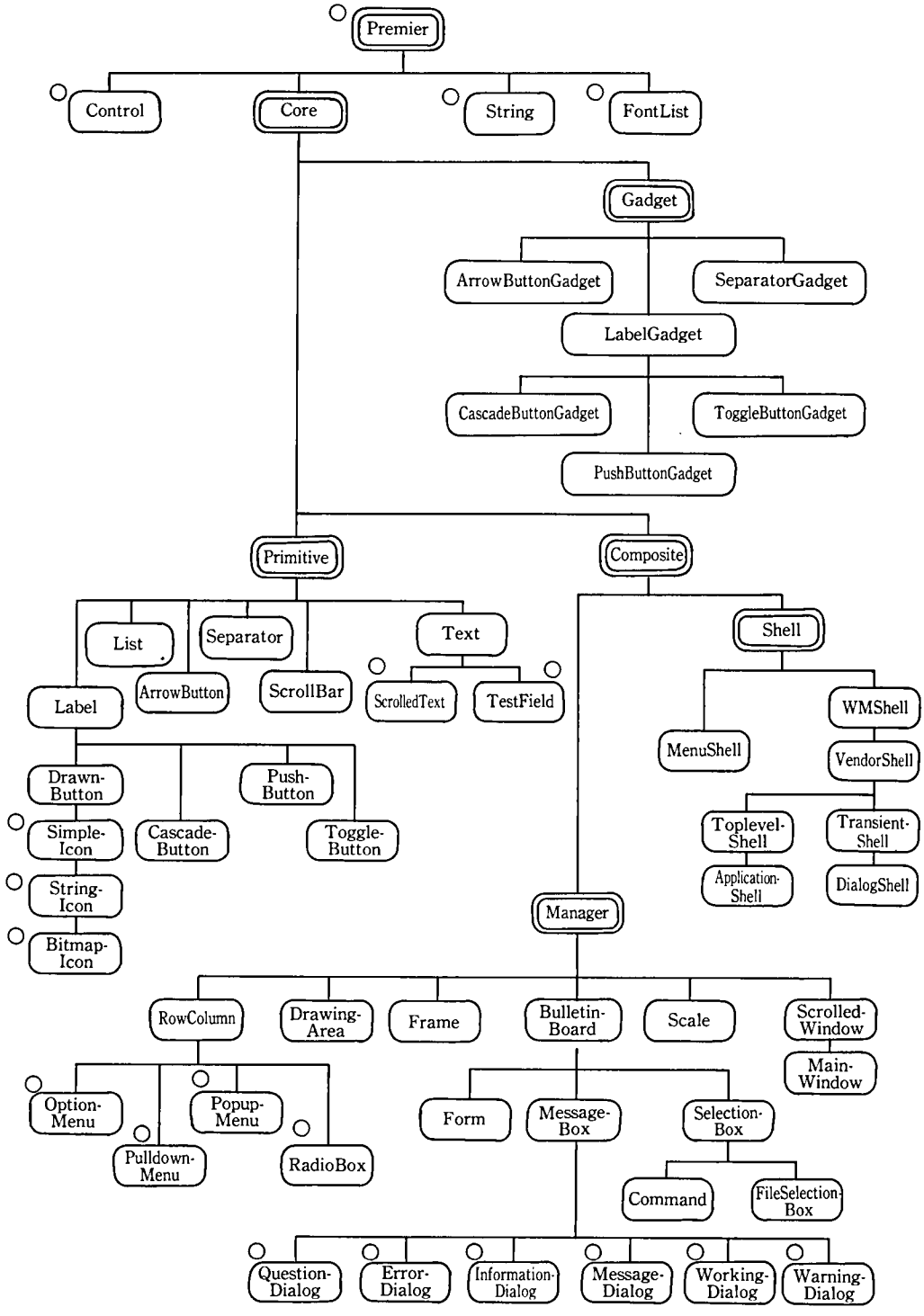


図 2　Xmpp クラス構造図

Fig. 2　The structure of Xmpp class

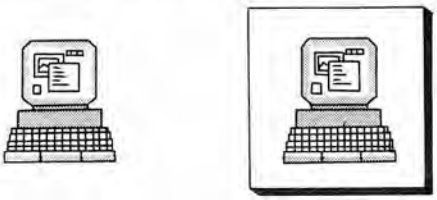


図 3　非矩形アイコン（左）と矩形アイコン（右）
Fig. 3　Non-rectangle icon (left) and rectangle icon (right)

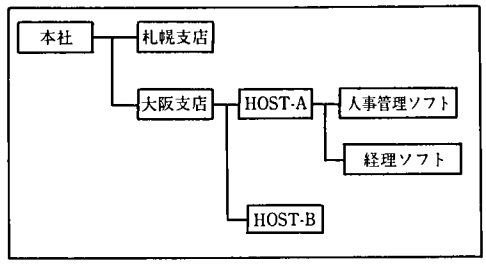
図 4　画像イメージを貼り付けたウィンドウ
Fig. 4　Sample window with a picture image

3. Mui ライブラリの概要

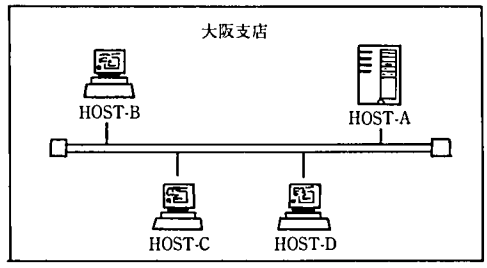
3.1 Mui の概要

Mui ライブラリは運用管理システムが管理する管理対象オブジェクト（以下，管理対象）の集合をウィンドウ上に視覚的に表現し，操作するためのライブラリである．管理対象とは具体的には HOST マシン（ワークステーション，汎用機等），ユーザ，組織（支社，営業所等），ソフトウェア等を指す．

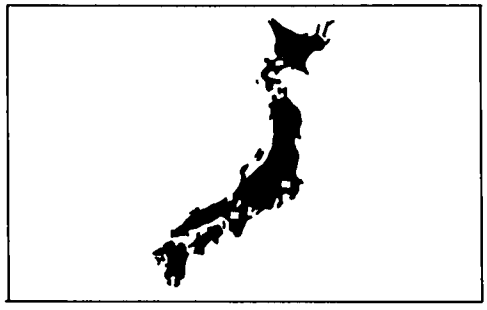
Mui ライブラリは，これらの管理対象の集合を表現するために図5のような3種類の構成図（ツリー図，ネットワーク図，分布地図）を提供する．それぞれの構成図上では管理対象は，図6に示すような3種類のアイコン（印（□）のみ，名称のみ，絵



(a)ツリー図



(b)ネットワーク図



(c)分布地図

図 5　Mui が提供するグラフィックの例

Fig. 5　Sample of GUI provided by Mui

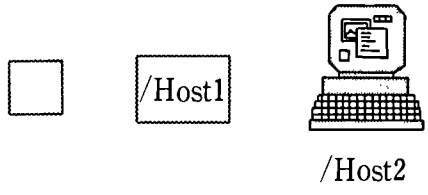


図 6　アイコンの種類

Fig. 6　Type of icon

(ビットマップ) つき) によって表現される．図5(a)は運用管理システムが管理する会社の組織構成をツリー図で表現したもの，図5(b)は大阪支店のネットワーク構成を表したもの，図5(c)は東京本社およびその直下の組織（大阪支店，札幌支店）を地図上に配置したものである．システム管理者はこのような図によってネットワークの構成を変更したり，ネットワークの稼働状況を監視したりするわけである．

これら管理対象に関するデータ（名称，ネットワークアドレス，稼働状態等）は運用管理データベース(以下，管理DB)に収められている．したがって，構成図とは管理DB上の管理対象のデータを視覚的に表現し，変更するためのGUI，と言い替えることができる．

ところで，この三つの構成図は一見するとまったく異なる図のようであるが，図の上に管理対象を表すアイコンを配置しただけという点では同じものである．異なるのは，アイコンの配置（ツリー図では木の節の位置になければならないのに対し，分布地図では地図上の任意の位置に配置できなければならない，等)，アイコン間に引かれる線の形式，の二点である（アイコンの外観は図5の例ではそれぞれ異なったものになっているが，場合によっては，ツリー図上で図5(b)のような絵付きアイコンで表されることもあり，さまざまなケースがあるので共通点と考えてよい)．

そこで，Muiでは三つの構成図を基本的に同じものとして捉え，共通の属性，機能を一つのスーパークラスとして定義し，相違点を各サブクラス（ツリー図，ネットワーク図，分布地図）に定義している．これにより，①同じ機能を重複して開発するという無駄が省けるので開発工数を削減できる，②ライブラリの利用者は，基本的にはスーパークラスの提供する機能（メソッド）を覚えるだけで3種類の構成図を利用できる，という利点が得られると考えたからである．

Muiが標準で提供するのは標準的なパターンの構成図だけであり，ユーザの要求によってはその内容を変更しなければならないケースもありうる．しかし，Muiでは構成図を，複数種類の独自の役割を持つオブジェクトの組み合わせによって構築しており，それらオブジェクト間でメッセージをやり取りすることによって構成図全体が機能するようになっている．つまり，構成図としての機能を各オブジェクトに分散し，それぞれ独立して機能するように設計しているわけである．これにより，機能変更が発生しても，一部のオブジェクトの修正のみを行えば，他のオブジェクトには影響を与えずに対応できるようになっている．

3.2 Muiのオブジェクト構成

構成図を構成している基本的なオブジェクトはノードオブジェクト，構成図オブジェクト，管理対象インタフェースオブジェクト，コントローラオブジェクトの四つである．前出の分布地図をオブジェクトの関係によって書き換えたものが図7である．

ノードオブジェクトは構成図上で管理対象を表現するものである．その外観は前出の3種類のアイコンによって表現される．構成図オブジェクトは構成図全体を管理するものであり，配下の全ノードの管理も行う．管理対象インタフェースオブジェクトはノードオブジェクトと1対で存在し，管理DB内の管理対象(のデータ)とのインタフェースを受け持つオブジェクトである．コントローラオブジェクトは構成図オブジェクトと1対で存在するもので，ユーザプログラムとのインタフェースを担当する．

前述したように，管理対象の名称等の属性データは実際には管理DBに収められており，ノードオブジェクトは管理対象インタフェースオブジェクトを通してその属性データを取得し，構成図上に表現する．つまり，ノードオブジェクトとは単なる空箱，あるいは構成図上の単なる“点”であって，その外観は運用管理DB上の属性データによって決定されるわけである．

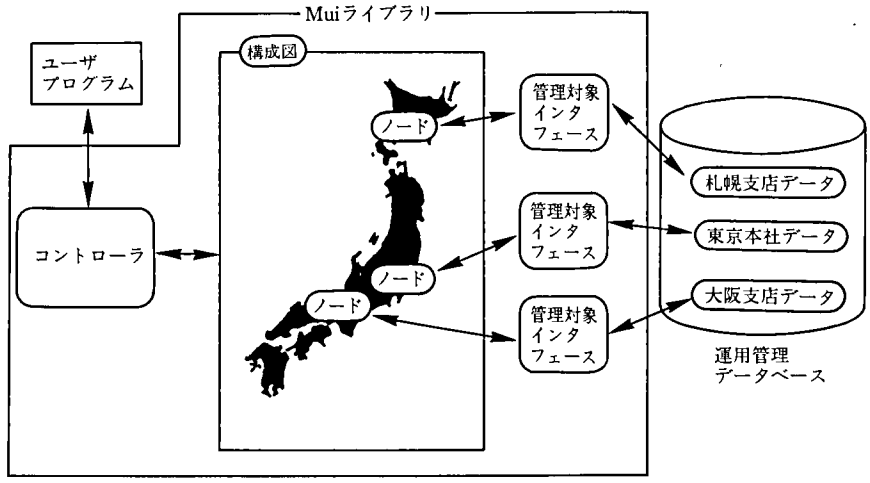


図 7 Mui の構成

Fig. 7 The structure of Mui

3.3 Mui の位置付け

図 8 に Mui および Xmpp を使用する場合のアプリケーションプログラムの構成を示す．Mui ライブラリが提供する構成図はウィンドウ上では一つのサブウィンドウに相当する（図 1）．したがって，ウィンドウのその他の部分，メニュー，ボタン等，は Xmpp ライブラリ等を使用して開発者が独自にコーディングする必要がある．また，Mui ライブラリ内部でもアイコン等の表示機能は Xmpp ライブラリを使用している．

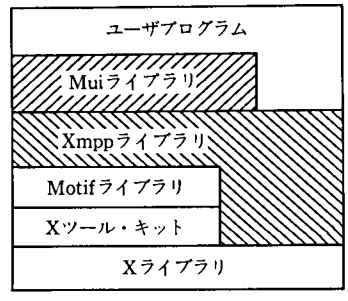


図 8 プログラム構成

Fig. 8 The structure of application program

4. Mui クラスライブラリ

4.1 クラス構成

Mui では，3.2 節で説明したオブジェクトの特徴を，それぞれコントローラクラス，構成図クラス，ノードクラス，レスポンダクラス（ノードオブジェクトは実際にはノードクラスとレスポンダクラスに分かれている．それぞれの役割は 4.4 節，4.5 節参照），管理対象インタフェースクラスに定義している．また，管理対象インタフェースクラスを除く各クラスの下には 3 種類の構成図に対応するサブクラスがそれぞれ存在し，ここにそれぞれの相違点を定義している．以下では，各クラスの概要を説明する．

図9にMuiライブラリのクラス構成を示す（図9）.

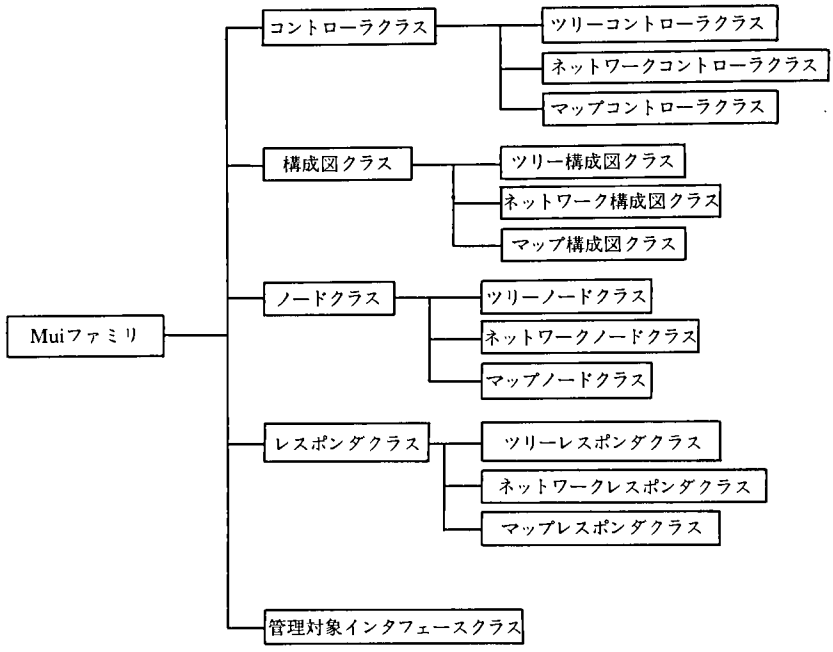


図 9　Mui クラス構成図

Fig. 9　The structure of Mui class

4.2　コントローラクラス

コントローラクラスはユーザプログラムとのインタフェースを担うもので，開発者がMuiの内部構造を意識せずにMuiのコントロールができるように用意されたクラスである(図10).実際,Muiを使用する開発者はこのクラスのインスタンスを生成し，そのインスタンスに対して簡単なメッセージを送るだけでMuiの機能が使えるようになっている．したがって，開発者はコントローラクラスの提供するメソッドを覚えるだけで構成図が表示できる．

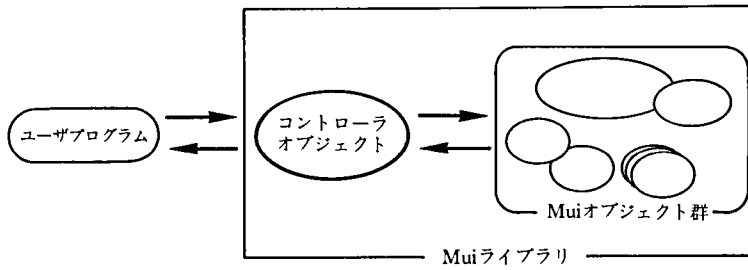


図 10　コントローラクラスの位置付け

Fig. 10　The role of controller class

サブクラスとしてツリーコントローラクラス，ネットワークコントローラクラス，マップコントローラクラスの三つがある．各クラスはツリー図，ネットワーク図，分布地図にそれぞれ対応している．ツリーコントローラクラスを例にすると，簡易なツ

リー表示だけならコントローラクラスに関するコーディングがユーザプログラム中に占めるのは，たった数行程度ですむ．以下にコントローラクラスに関する具体的な処理の流れを示す．

① (MuiCtrl *)newMuiTreeCtrl(root, scroll, "back", rspType, multiFlag);

② ctrl → setBackImage ("japan. h", "SaddleBrown", "CornflowerBlue");

③ ctrl → initShow( ) ;

①ユーザはツリーコントローラクラスのインスタンスを生成し，②表示画面の土台となる背景画面に対する各種設定を行うためのメッセージを送る．③最後に初期画面の表示メッセージを送るだけでデフォルトで設定されている画面が表示される．

4.3 構成図クラス

構成図クラスは，表示画面全体の管理・描画を行う．すなわち表示画面の大きさ，背景に貼り付ける画像イメージファイル名等の画面情報を管理し，それらの情報を基に表示画面の描画を行う（図11）．

サブクラスとして，ツリー図，ネットワーク図，分布地図それぞれに対応する，ツリー構成図クラス，ネットワーク構成図クラス，マップ構成図クラスの三つがある．

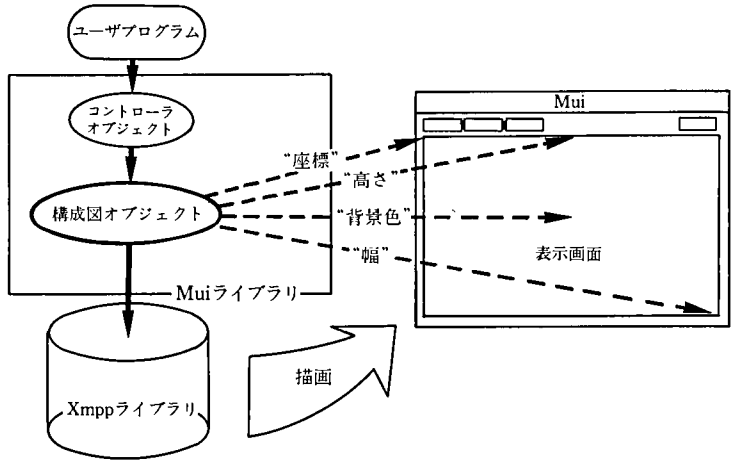


図 11 構成図クラスの位置付け

Fig. 11 The role of configuration class

4.4 ノードクラス

表示画面上における抽象的な管理対象を表すアイコンは，ノードクラスとレスポンダクラス（後述）という二つのクラスによって管理され，これらは互いに1対1で対応している．ノードクラスはアイコンの表示データの管理を行い，画面表示の内部的な処理を担当するクラスである．サブクラスとしてツリーノード，ネットワークノード，マップノードの三つがある．ノードクラスが管理するデータとしては，アイコンの表示位置，高さ，幅，背景色，前景色，アイコンの種類（前出の3種類）等がある．また，各ノードと管理対象は1対1の対応を持っており，管理対象とノードクラスとの橋渡しは管理対象インタフェースクラス（後述）が行う．

また，Muiライブラリでは，ノード表示のために各ノード間に以下の四つの親子関

係を持たせている（図12）.

- 親 …… 一つ上位のノードを指す.
- 子 …… 一番目の下位ノードを指す.
- 兄 …… 同位の前のノードを指す.
- 弟 …… 同位の次ノードを指す.

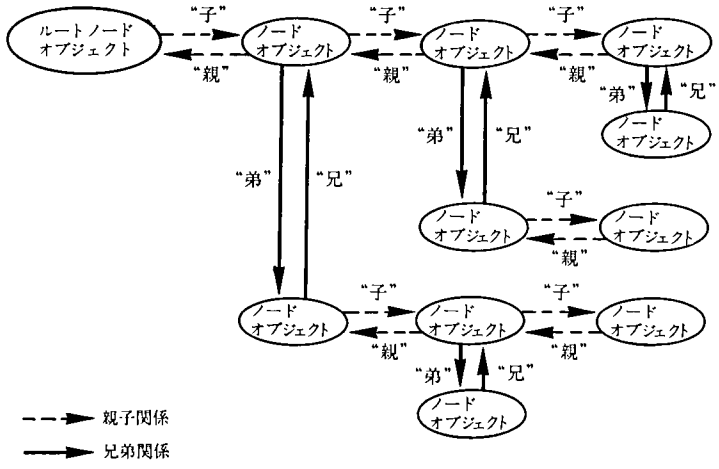


図 12　各ノードオブジェクト間の親子関係

Fig. 12　The parent-child relationship between each node objects

以下にノード表示の際の親子関係の使われ方を示す．図12中のルートノードとは，コントローラオブジェクトから，構成図オブジェクトを経て送られてきたノード表示のメッセージを，一番最初に受け取るノード表示の要のノードである．そして，そのメッセージはルートノードの子ノードに，次にその子ノードに，もし子ノードがいなければその兄弟ノードに…という具合にノード間の親子・兄弟関係を巡りながらノード表示は行われる（図13）.

4.5　レスポンダクラス

ノードクラスがアイコン表示の内部的処理を行うのに対して，レスポンダクラスはアイコンに対する画面操作イベントの受け取り，アイコンの描画といった外部的処理を行うクラスである(図14)．サブクラスとしてツリーレスポンダクラス，ネットワークレスポンダクラス，マップレスポンダクラスの三つがある．三つのクラスは各々，前出の3種類のアイコン表示（図6参照）を持ち，これらは表示画面のプルダウンメニューから選択することにより，動的に表示の変更ができる．また，その他にアイコンカラーの動的変更，移動可能状態への変更も可能である．こうしたアイコン上，または背景画面上で起こったイベント（マウスクリック等のアクション）に対して各ノードはアイコンの反転，子ノード群の表示・消去といった視覚上の反応を見せる.

4.6　管理対象インタフェースクラス

ノードオブジェクトと管理DB上の管理対象とのインタフェースを行うクラスで，DBアクセスのためのメソッドを実装している(図15)．具体的なメソッドとしては以下のものが挙げられる.

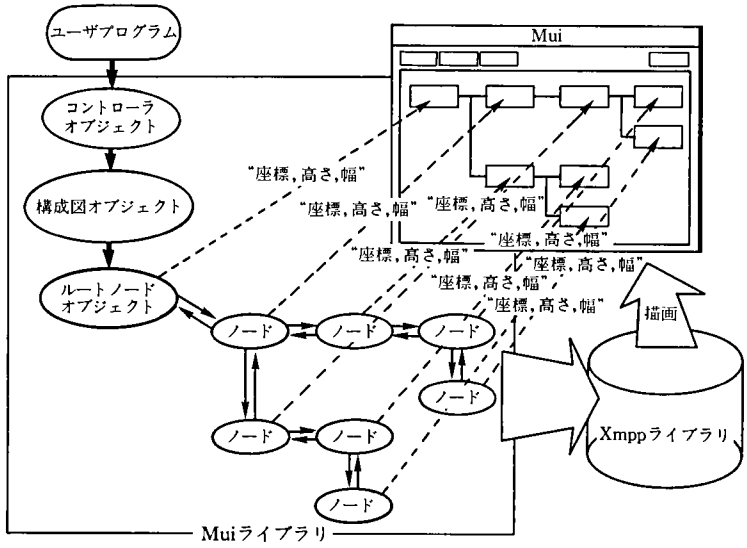


図 13　ノードクラスの位置付け

Fig. 13　The role of node class

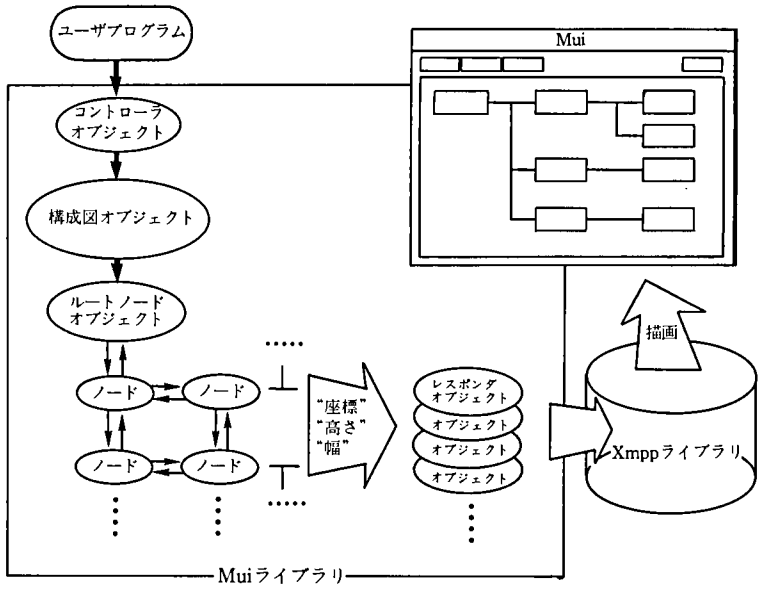


図 14　レスポンダクラスの位置付け

Fig. 14　The role of responder class

・データ名の取得メソッド：管理対象から管理対象名を取得する．

・下位の管理対象取得メソッド：管理対象が自分の直下に管理対象を保持する場合(日本ユニシスは，東京本社，関西支社，中部支社を持つ，等)，それに 1 対 1 で対応する管理対象インタフェースオブジェクトを生成する．

・管理対象取得確認メソッド（DB の検索)：管理対象の存在確認を行う．

・イメージファイル名取得メソッド：アイコンの表示に必要なイメージデータフ

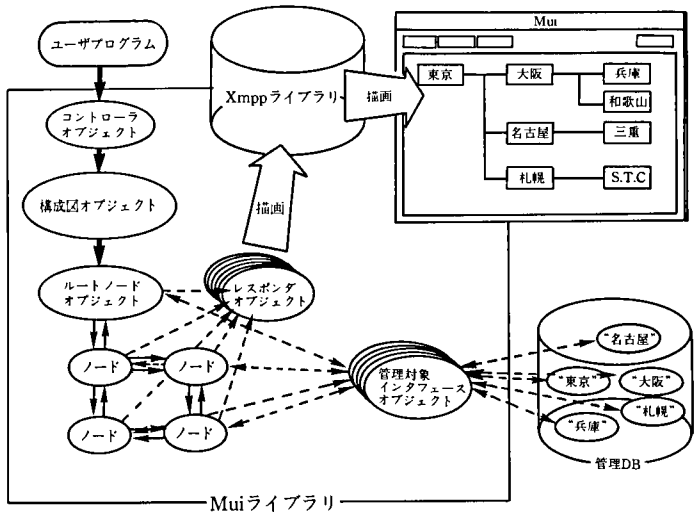


図 15　管理対象インタフェースクラスの位置付け

Fig. 15　The role of MuiMgdObj class

ァイル名を返す．

画面上のノード表示（アイコンの現われ方）は，管理DBが保持するデータによって決定される．管理対象の保持するデータモデルが不変ならば，たとえば管理対象側のDBシステムが，RDB（リレーショナルデータベース）からOODB（オブジェクト指向データベース）へ変更されたとしても，当クラスのメソッド内のDBインタフェース部分を新しいものに変更するだけで，他にはまったく影響を与えずに以前と変わらないノード表示を可能にしている．

4章の総括として，Muiの内部構造と視覚上の外部構造との相関関係を図16に示す．Muiにおいては，すべての処理がオブジェクト間の相互作用によって引き起こされることを示している．

5. Muiの機能

Muiライブラリの提供する三つの構成図の機能を簡単に説明する．

5.1 ツリー図の機能

ツリー図は構成木を表現したものである．構成木は運用管理システムの管理対象となる各部署の構成，および部署に接続されている機器の関係を表現するための木である．各ノードは各々一つか複数のネットワークを保有している構成単位（組織単位）や，その組織内のネットワークに接続されているワークステーション，プリンタ等の機器を表現する．

初期画面には画面左上に，ツリーのトップとなるルートノードが表示される．ルートノードをマウスでシングルクリックすればルートノードの直下の子ノード群がルートノードから線を引かれた状態で，内部的に自動計算された位置に表示される．仮りにそのノード群の子ノード（ルートノードからすれば孫ノード）を表示したければそのノード群をクリックすればよい．各アイコンは表示画面のプルダウンメニューからの選択により自由な配色，アイコン表示の変更ができる（図17）．

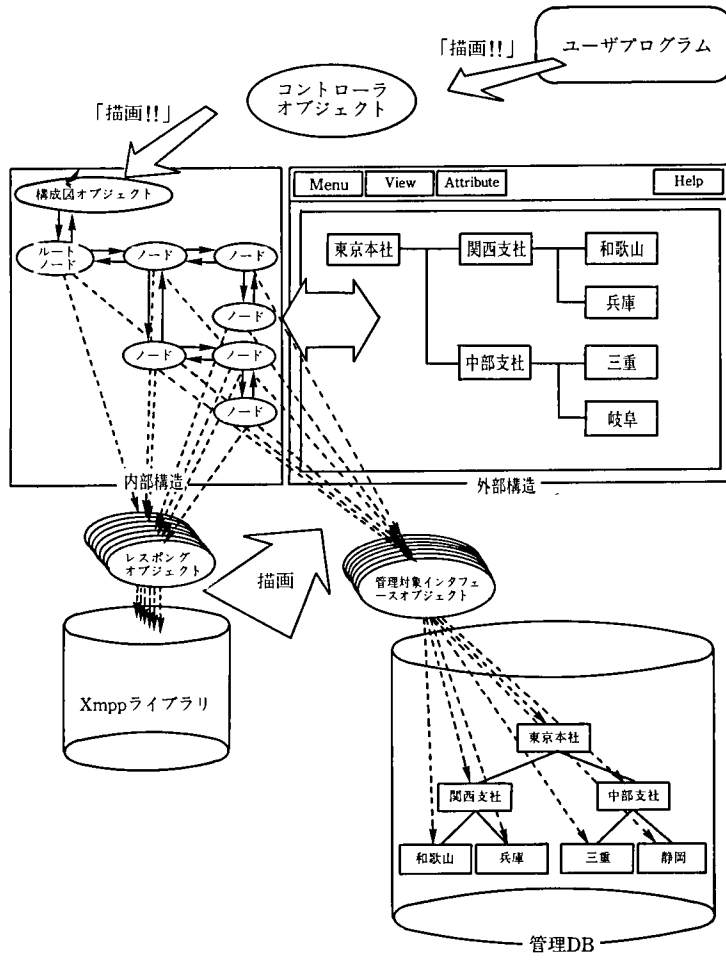


図 16　Mui における内部構造と外部構造の相関関係

Fig. 16　Corelationship between interior structure and exterior structure in Mui

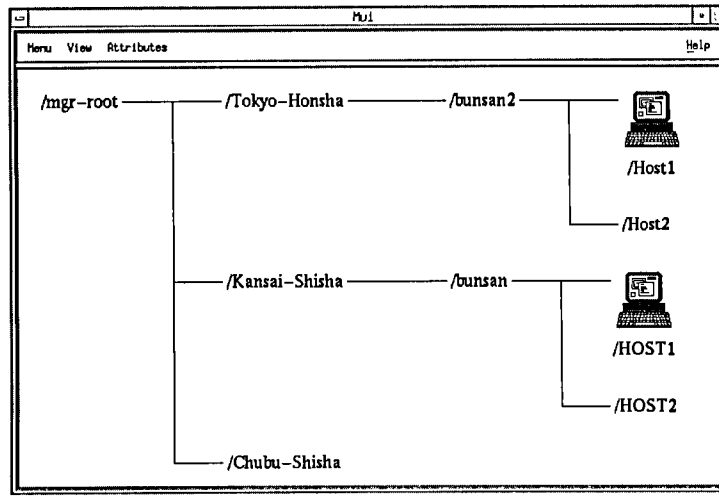


図 17　ツリー図の例

Fig. 17　The sample of tree window

5.2 ネットワーク図の機能

ツリー図における一つのノードがある組織の部署を表現しているものとすれば，ネットワーク図が表現するのはその部署内のネットワーク構成ということになる．

ネットワーク図で表現するのはホストマシンとLANケーブル等である．ホストマシンノードはユーザによって自由な配色・配置，アイコン表示の変更が可能である．また，各ホストマシンノードを接続しているケーブルノードも自由な配色・配置や，実線・点線といった表示が可能である．

初期画面では，ある部署におけるネットワークが，すべてのホストマシンノードがケーブルノードと線でつながった状態で表示される．親ウィンドウのプルダウンメニューから，アイコンの移動可能ボタンを選択すれば，自由なマシン配置が可能になり，ユーザによる思いのままのネットワーク構成が可能になる（図18）．

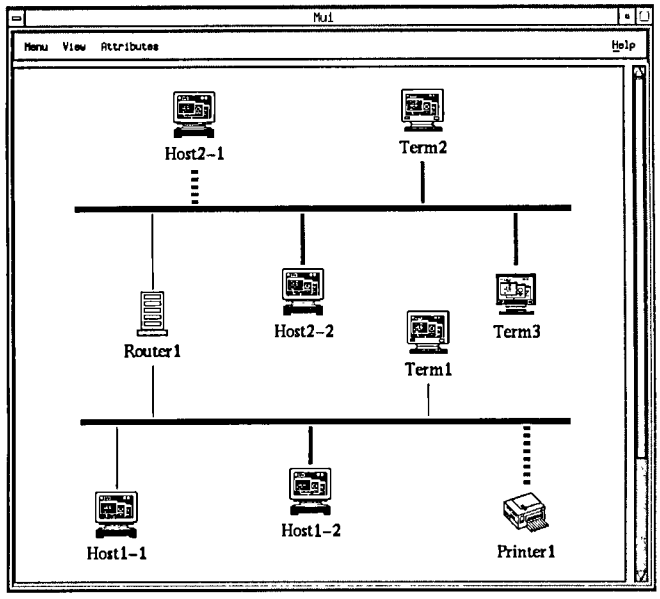


図 18　ネットワーク図の例
Fig. 18　The sample of network window

5.3 分布地図の機能

ツリー図における各ノードがある組織単位を表現しているものとすれば，分布地図上のノードはツリーノードの所在を表していることになる．分布地図上のノードも，ネットワーク図上のホストマシンノードと同様，自由な配色・配置・アイコン表示の変更が可能である．

たとえば，ある会社組織（日本ユニシス）の日本国内の分布地図であれば，地図上のある部署ノード（関西支社）をマウスでクリックすれば，日本地図からその支社の管轄地域（近畿地方）の地図に表示画面が切り替わり，その地域の支店，営業所等のノードが表示される（図19）．

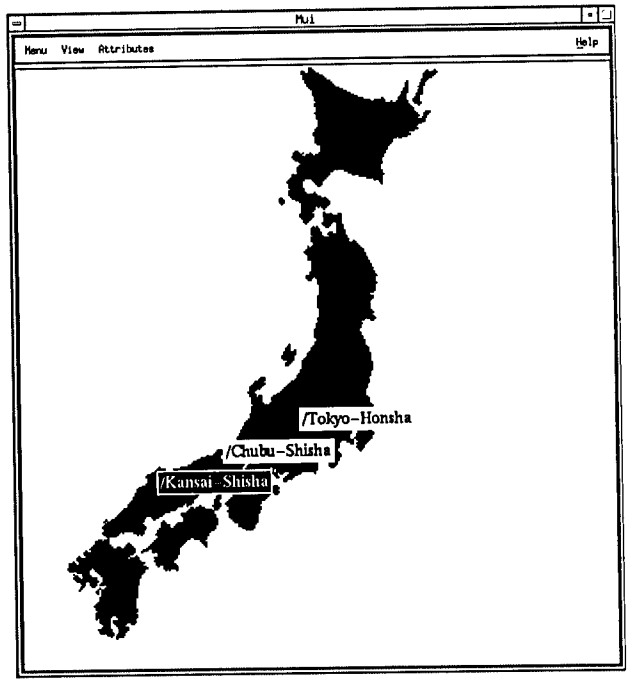


図 19 分布地図の例
Fig. 19 The sample of map window

6. 開発工数，総ステップ数等

Mui ライブラリの開発にかかった工数，および総ステップ数を以下に示す．

開発工数

設計 ：3 人月

プログラミング，テスト：7 人月

総ステップ数 16721 step

7. 今後の課題

現在，運用管理システムは基本設計中であり，Xmpp, Mui ライブラリも第 1 版が完成したところである．今後，システムの設計が進み，必要となる操作画面が確定するのに合わせて，両ライブラリも機能拡張を行う予定である．

必要となるであろうウィンドウは本稿で見てきたようなものの他にも数多く存在する．しかし，これらの多くは Xmpp の提供する構成部品を組み合わせるだけで構築できるものが多く，また何通りかのパターンに分類することができる．そこで，Xmpp の機能拡張として，頻繁に使用されると思われるパターンのウィンドウをクラスとして提供し，開発担当者の負担を軽減することを考えている．

また，Mui の機能拡張としては，とくにネットワーク図は現在バス型の接続形態の表示機能のみを提供しているが，スター型等，他の接続形態の表示機能を提供し，それらを組み合わせた複雑な接続形態のネットワークを表現できるように改良する予定である．

最後に，本ライブラリの設計，開発にわたり多大な御協力を頂いたソフトウェア興業株式会社大阪事業所システム部の皆様に深謝の意を表する．

参考文献

[1] Ellis & Stroustrup, The Annotated C++ Reference Manual, 1986 by Addison Wesley.
[2] R. S. ウイナー, L. J. ピンソン, C++：オブジェクト指向プログラミング, Addison Wesley, 1988.
[3] 兜木昭男, 木下凌一, 栄谷政己, 林　秀幸, 安川悦子, X-Window OSF/Motif プログラミング, 日刊工業新聞社, 1990.
[4] Douglas A. Young, The X Window System : Programming and Applications with Xt, OSF/Motif Edition, 1990 by Prentice-Hall, Inc.
[5] Douglas A. Young, Object-Oriented Programming With C++ And OSF/Motif, 1992 by Prentice-Hall, Inc.
[6] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, William Lorensen, Object-Oriented Modeling And Design, 1991 by Prentice-Hall, Inc.
[7] 小嶋隆一, Smalltalk プログラミングマニュアル, JICC 出版局, 1989.
[8] 小林史典, オブジェクト指向と Smalltalk, CQ 出版株式会社, 1989.

執筆者紹介

伊　藤　直　行 (Naoyuki Ito)
1965年生. 1991年筑波大学第一学群自然学類卒業. 同年日本ユニシス(株)入社. オブジェクト指向による分散システム運用ソフトウェアの開発に従事. 現在 関西支社システム技術部 分散システム課に所属.

熊　本　厚　志 (Atsushi Kumamoto)
1961年生. 1985年岡山大学経済学部経済学科卒業. 同年日本ユニシス(株)入社. 金融機関向けシステムの開発, 保守を担当. 1990年大阪大学基礎工学部情報工学科受託研究員. 現在 関西支社システム技術部 分散システム課所属. 情報処理学会会員.

エキスパートシステムの開発方法とツール

A Method of Developing Expert Systems and Development Support Tools

東野康臣

要 約 近年，エキスパートシステム（ES：Expert System）は，問題解決手法の一つとして従来技術に自然に溶け込んだ形で実用化が進展してきている．また，ESのプロトタイピング開発手法は，新しいソフトウェア開発技術の一環として捉えられ広く活用されてきている．

本稿では，実用的なESの開発支援環境として，従来システムの一部にESを統合する際の開発方法，開発ツールと活用事例，今後のES開発・実行支援環境についての展望を述べる．

ESをはじめとする各種AI（Artificial Intelligence：人工知能）技術は，今やさまざまなアプリケーションの知的化技術として必須のものである．本稿が一助となり，ESがさらに広範なアプリケーションで実用に供されることを期待したい．

Abstract In recent years, expert systems (ES) have been getting into a higher level of applicability as one of the problem-solving methods in the form of their natural integration into traditional techniques. In addition, the ES prototyping approach, now accepted as benefiting new applications development, has also come into more frequent use.

This paper is intended to discuss (1) a method by which to integrate an expert system into part of the conventional information processing systems so as to provide an environment which facilitates efficient ES development efforts, and (2) existing development support tools and instances of their uses, as well as (3) perspectives for future ES development/implementation support environments.

Varieties of AI (artificial intelligence) technologies now available, including expert systems, have proved to be instrumental in producing intelligence-oriented applications. The author's wish is that this paper can be of any assistance in encouraging the more practical adoption of expert systems in a wider range of applications.

1. はじめに

近年，エキスパートシステム（ES：Expert System）は，問題解決手法の一つとして従来技術に自然に溶け込んだ形で実用化が進展してきている．また，ESのプロトタイピング開発手法は，新しいソフトウェア開発技術の一環として捉えられ広く活用されてきている[1]．

本稿では，従来システムの一部にESを統合する際の開発方法，開発ツールと活用事例，今後のES開発・実行支援環境についての展望を述べる．

ESの開発方法では，実用的なESを効果的に開発する方法として，従来システムとの結合を重視した組込み型ESの標準的な開発方法を述べる．従来システムの開発工程の中にES開発をどう組み込むか，ESの開発工程はどうあるべきか，各工程の主要作業と成果物は何か，等を示す．

ESの開発ツールでは，ここ数年間で利用が急増したUNIX*ワークステーション上

* UNIXオペレーティングシステムは，UNIX System Laboratories. Inc. が開発し，ライセンスしている．

での開発支援環境として，エキスパートシステム構築支援ツールGNOSIS-II*と知的活動支援プラットフォームソフトウェアTIPPLER**の概要を紹介する．さらに，これらツールの活用事例として機械割当てスケジューリングを紹介する．

今後のES開発・実行環境では，これまで主として専門家の業務を支援・代行するシステムとして実用化が進展してきたESの今後の開発・実行支援環境を展望する．

2. エキスパートシステムの開発方法

ESは，問題解決手法の一つとして従来システムに組み込まれる形で実用化が進展してきている．実用的なESを効果的に開発するには，従来システムの開発工程にES開発をどう組み組むか，ESの開発工程はどうあるべきか，各工程の主要作業と成果物は何か，等を明確にすることが必要である．

ここでは，従来システムとの結合を重視した組込み型ESの開発方法について述べる．まず，ESの定義と構成の概要を示す．次に，実用的なESを開発する際の基本的な考え方を，ES開発のモデルとサイクルで説明する．最後に，この考え方に基づくES開発標準工程と各工程での主要作業および成果物を示す．

ここで述べる内容は，これまでわれわれが携わった数多くのES開発経験とそこで培ったノウハウを整備・体系化したものである．

2.1 エキスパートシステムの定義と構成

ESとは，『業務専門家の知識やノウハウを知識ベース化し，それを用いて専門家の思考過程を模倣することにより，専門家並みの問題解決を行うことを目的としたシステム』である．

従来システムとの結合を重視した組込み型ESでは，システム全体は，知識処理プログラム，知識ベース，推論エンジンから構成されるES部分と関連システムから成る．各構成要素の内容を示す．

1) 知識処理プログラム……問題解決の手順を示すプログラム．推論の初期状態生成，推論の起動，推論結果の検索，関連システムとのインタフェース等を含む．
2) 知識ベース……推論で活用する知識を一定の形式で保持したもの．プロダクションルール，フレームと関係，メソッド/ルール関数/付加手続き（デモン）等から構成される．知識ベースは，ES構築支援ツールを用いて定義する．
3) 推論エンジン……知識ベース内の知識を使って推論を実行する制御機構．推論エンジンは，ES構築支援ツールが装備しているものを利用する．
4) 関連システム……ES部分と関連して業務システム全体を構成するサブシステム群．利用者インタフェース，データの入出力と管理，帳票作成，関連業務システム，他システム・インタフェース等のサブシステムから構成される．

典型的な組込み型ESの構成を図1に示す．図中の＊印で示した部分が，ES開発の対象物（推論エンジンはES構築支援ツールを利用）である．また，図中のES部分をES開発モデルと呼ぶ．

---

* GNOSIS-IIは，日本ユニシス（株）が開発し販売しているソフトウェアである．

** TIPPLERは，（株）野村総合研究所と日本ユニシス（株）により共同開発されたソフトウェアである．

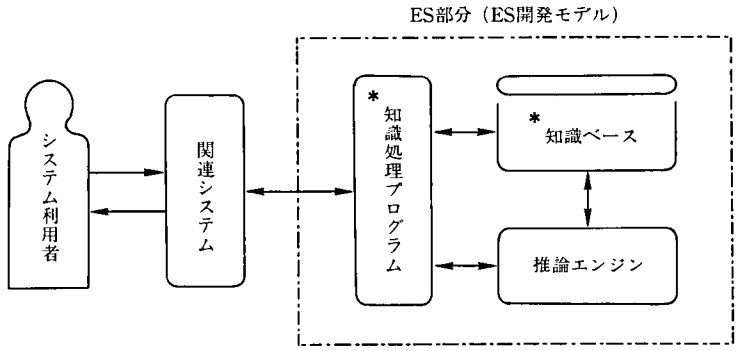


図 1　組込み型 ES の構成

Fig. 1　Construction of the embeded expert system

2.2　エキスパートシステム開発のモデルとサイクル

ES の開発では，初期の段階で問題解決に必要な知識を十分に獲得できないことが多い．このため，比較的に短い周期（1か月～半年程度）で段階的に開発を進めながら，順次システムの仕様を確定していく必要がある．このことから，ES 開発には同じような過程を繰り返し（スパイラル）ながら連続的にシステムを成長させていく進化型プロトタイピング手法の採用が有効となる．

進化型プロトタイピング手法を採用した実用的な ES 開発では，システムの機能や性能を限定したモデルを作成し，そのモデルの検証と評価を通して段階的にシステムの機能拡張や性能向上（解の質や効率等）を図っていく．この段階的な繰り返し過程をサイクルと呼ぶ．各サイクルでは，それぞれ動作可能なモデルを作成し，そのモデルの検証と評価を実施する．これらのモデルは，標準的には「初期モデル」，「部分モデル」，「実用モデル」の三段階で成長する．各モデルの位置付けを示す．

1)　初期モデル……初期に獲得した部分的（基本的）な知識をもとに，機能の一部分を実現したモデル．作成するシステムの基本的な仕様の確認，知識ベース構造や推論方式の検証を目的とする．

2)　部分モデル……定められた範囲内で問題を解くことができるモデル．システムの成長過程に位置するモデルで，機能範囲の拡大や性能の向上等を目的とする．部分モデルの開発は，システムの規模に応じて複数のサイクルを繰り返すことがある．

3)　実用モデル……最終的な機能範囲と実用に耐え得る性能レベルを実現したモデル．

機能範囲と性能レベルから見た ES 開発モデルの成長過程を図 2 に示す．部分モデルは初期モデルを含み，実用モデルは部分モデルを含んでいる．

2.3　ES 開発標準工程

進化型プロトタイピング手法を用いた実用的な ES の開発標準工程を図 3 に示す．この工程が対象とする範囲は，システム化する業務範囲が明確になった時点から始まる ES 開発の全般である．したがって，企業の取るべき戦略や課題を明らかにし，何をなすべきか，何をシステム化すべきかを決めるシステム化活動の上流段階は対象とし

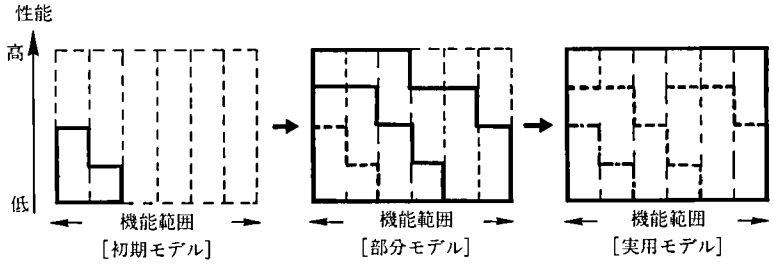


図 2　ES 開発モデルの成長過程

Fig. 2　Evolution of the expert system development model

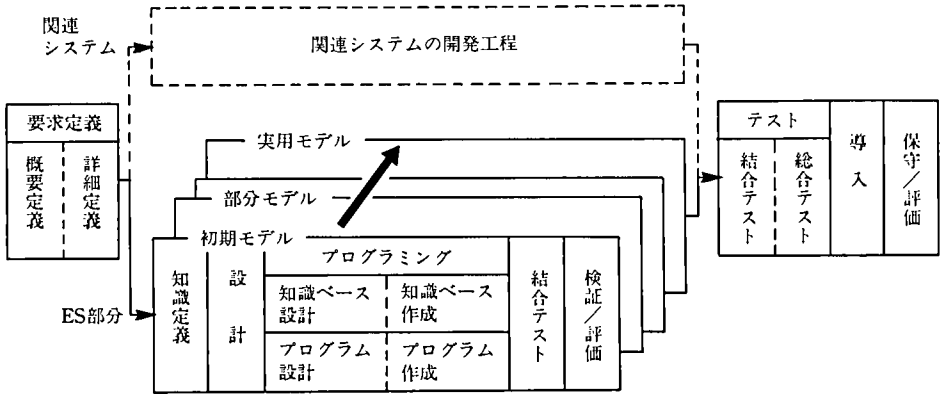


図 3　ES 開発標準工程

Fig. 3　Standard process of the expert system development

ていない．

ES 開発標準工程は，大きく次の三段階から構成される．第一段階は，システム全体の要求仕様を確定し，ES 部分と関連システムとを切り分ける部分である．この段階の工程は，大工程の要求定義と，その中工程である概要定義と詳細定義から構成される．

第二段階は，切り分けられた ES 部分と関連システムとを並行に開発する部分である．この中の ES 部分の開発には，初期モデルから部分モデル，実用モデルへと三段階で成長するプロトタイピング手法を用いた ES 開発モデルを採用する．これらのモデルは，すべて基本的には同一工程をとる．開発の各サイクルは，次の工程から構成する．知識定義，設計，プログラミング（知識ベース設計と知識ベース作成，プログラム設計とプログラム作成），結合テスト，検証/評価．知識定義では，業務の専門家から獲得した知識を知識定義書として整理・体系化する．設計では，知識定義書をもとに知識ベースと知識処理プログラムとを切り分け，それぞれの仕様を詳細化する．プログラミングでは，知識ベースと知識処理プログラムを並行して作成する．結合テストでは，これら両者を結合し整合性を確認する．検証/評価では，作成したモデルを知識定義書に基づいて検証し，ソフトウェア要求仕様書に基づいて評価する．さらに，この結果をもとに次のサイクルの開発計画を修正・立案し確定する．関連システムの開発工程は，基本的には通常のシステム開発と同様である．従来システムの開発では，

一般的にこの部分の工程は，設計とプログラミング（プログラム設計とプログラム作成）から構成される．グラフィカル・ユーザ・インタフェース（GUI）を用いたシステム等では，ES 部分の開発と同様，プロトタイピング手法を採用することが多い．

第三段階は，並行して開発した ES 部分と関連システムとを結合し，システム全体の整合性の確認，新システムの稼働準備，稼働したシステムの評価と維持を行う部分である．この段階の工程は，基本的には従来システムの開発工程と同様であり，テスト(結合テストと総合テスト)，導入，保守/評価から構成される．ここで ES として考慮すべき点は，知識ベースの保守に関する問題である．知識ベースを常に最新の状態に維持するために，知識ベースの保守体制を明確にしておくことが重要となる．

ES 開発標準工程の各工程での主要作業と作業内容，成果物を表 1 に示す．

3. エキスパートシステムの開発ツール

実用的な ES の開発には，業務専門家の知識やノウハウを用いて問題解決を行う知識処理と ES の結果を人が判断し補正する知的活動支援とが協調したシステムで，ES と既存業務システムやデータベースとが連係したシステムが必要となる．当社では，これらの要請に応えるため，知識処理の既存システムへの組み込みと連係を重視した ES 構築支援ツール GNOSIS-II と人の判断や思考・創造性を援助するシステムを効率良く開発できる知的活動支援プラットフォームソフトウェア TIPPLER を開発し提供してきている．ここでは，これらの開発支援ツールの概要を紹介する．

3.1 エキスパートシステム構築支援ツール GNOSIS-II [2]

GNOSIS-II は，既存システムへの知識ベースの組み込みを重視した ES 構築支援ツールである．

3.1.1 GNOSIS-II の機能と特徴

GNOSIS-II の主な機能と特徴には，知識表現の容易さ，強力な前向き推論エンジン，メタ知識の実現，既存システムとの結合の容易性等がある．

1) 知識表現……知識表現には，フレームと関係(リレーション)，プロダクションルール，手続きプログラム（メソッド，ルール関数，付加手続き）を備えている．

フレームは，知識の中にある物や事象の概念を構造的に表現するための枠組みであり，フレームが値を持つことにより一つの事実が表現できる．フレーム間には階層関係が定義でき，上位フレームの属性は下位フレームに多重・多段で継承できる．任意の二つのフレーム間には，階層関係や従属関係，相互関係等を表現する関係（リレーション）が定義できる．

プロダクションルールは，観測できる事実から結論を導き出すための形式的な表現であり，『IF “条件” THEN “行動”』あるいは『IF “前提” THEN “結論”』形式で表す．“条件” が成立すれば “行動” を実行する，あるいは “前提” が成立すれば “結論” を生起することを示す．ルールモジュールは，特定の目的を持ったプロダクションルール群を構造化するためのもので，知識の階層構造的な表現を可能とする．

手続きプログラムは，問題解決の過程で要求される手続き的な知識の表現であり，特定のフレームに任意の操作を組み込むメソッド，算術計算等のプロダクシ

表1　ES開発標準工程の主要作業と成果物(続く)

Table 1　Major works and outcoming things of the standard process(cont.)

大工程 / 中工程		作業	主成果物	参加者
要求定義				システム開発者 業務専門家 システム利用者
概要定義	主要作業	対象業務の現状の問題点や課題を抽出し，その解決方法とシステム化の範囲を明確にする．そして，システムに要求する機能，入出力を定義する．	システム概要定義書	
	作業内容	・業務の現状調査と分析 ・業務の学習 ・開発条件の記述 ・処理環境定義 (使用ハードウェア，ソフトウェア構成) ・システム化の範囲，目的，機能の定義 ・入出力の定義 (種類/媒体/データ量/タイミング) ・品質要求の定義 ・知識源の決定		
詳細定義	主要作業	確定したシステム化の範囲について詳細化を進め，かつ各種制約条件下での実現性を確認する．そのために入出力/業務機能の詳細とデータ処理の概要を確定し，かつ基本的なシステムの仕組みを考案する． ES部分と関連システム部分を切り分け，これ以降の工程を分ける． ESについては，問題解決に必要なデータや知識の種類を明らかにする．また，プロトタイピングの計画(サイクル回数と各モデルの目的，内容等)を決定する．	ソフトウェア 要求仕様書 システム開発計画書	
	作業内容	・対象領域と問題解決の枠組みの明確化 ・各入出力の詳細(項目，サイズ)の確定 ・システム機能の定義 (機能の構成，流れ等) ・処理論理(機能)の記述 ・インタフェース(マンマシン，ハードウェア，ソフトウェア，端末/ホスト，他システム)の概要定義 ・ファイル体系，データベース論理構造の記述 ・開発計画の立案 ・開発規模の見積り		
知識定義	主要作業	知識ベース作成のために必要となる知識を抽出し，業務専門家の思考過程や知識内容を体系化する． 問題解決のための知識の詳細を定義する．	知識定義計画書(☆) 知識定義書	システム開発者 業務専門家
	作業内容	・知識の収集と分析(専門家，事例，文献等) ・用語の定義 ・知識の整理，体系化 ・知識内容の記述		

(注)　成果物の☆印はその工程以前に作成される．

大工程 / 中工程		作業	主成果物	参加者
設計	主要作業	システム仕様の設計を行い，知識ベース部分と知識処理プログラム部分を切り分ける．知識の内容について知識ベース仕様が作成できるように詳細化する． 知識処理プログラムの機能，入出力等についてプログラム仕様が作成できるように詳細化する．	設計計画書(☆) 設計基準(☆) (設計の最初に作成) システム設計書	システム開発者
	作業内容	知識ベースの詳細設計 ・知識ベース構造の設計 (フレーム/ルールモジュール) ・推論方式の設計 ・ルールの設計 知識処理プログラムの詳細設計 ・データ構造の設計 ・制御構造の設計 ・プログラムの共通処理内容 ・モジュールごとの詳細処理内容，方法の設計		
プログラミング				システム開発者
知識ベース設計	主要作業	フレーム，ルールモジュールおよび外部手続きの仕様を作成する．	知識ベース作成計画書(☆) 知識ベース仕様書 テスト基準(☆) テスト仕様書	
	作業内容	・知識ベース仕様書の作成 (フレーム，ルールモジュール，外部手続き)		
知識ベース作成	主要作業	フレーム，ルールおよび外部手続きのコーディングと単体テストを実施する．	知識ベースリスト 外部手続きリスト 単体テスト結果	
	作業内容	・フレームとルールのコーディング ・外部手続きのコーディング ・外部手続きの単体テスト ・ルールモジュールのテスト		
プログラム設計	主要作業	プログラム単位の仕様書を作成する．	プログラム作成計画書(☆) プログラム仕様書 テスト基準(☆) テスト仕様書	
	作業内容	・プログラム仕様書の作成		
プログラム作成	主要作業	コーディングと単体テストを実施する．	プログラムリスト 単体テスト結果	
	作業内容	・コーディング ・プログラム単体テスト		

(注) 成果物の☆印はその工程以前に作成される．

大工程 / 中工程	作業		主成果物	参加者
結合テスト	主要作業	システム設計書をもとに，知識ベースと知識処理プログラム間および知識ベース全体の整合性を確認する．	結合テスト計画書(☆) 結合テスト結果	システム開発者
	作業内容	・知識ベースと知識処理プログラム間のインタフェース確認		
検証/評価	主要作業	知識定義書に基づいてモデルを検証する．ソフトウェア要求仕様書をもとにモデルの評価を行い，その結果を次のサイクルに反映させる．次サイクルの計画を立てる．	検証評価基準(☆) 検証評価計画書(☆) 検証評価報告書	システム開発者 業務専門家
	作業内容	・モデル全体の機能確認テスト ・問題解決能力の評価 ・拡張性，保守性，操作性，効率等の評価 ・次サイクルの計画立案		
テスト				システム開発者 業務専門家 システム利用者
結合テスト	主要作業	ESと関連システムとのインタフェースの整合性を確認する．	結合テスト計画書(☆) 結合テスト結果	
	作業内容	・利用者インタフェースとの整合性確認 ・他システムとの結合確認		
総合テスト	主要作業	開発システムが本番構成(同様構成)で稼働できる状態になっていることを確認する．	総合テスト計画書(☆) 総合テスト結果 操作マニュアル(☆)	
	作業内容	本番稼働確認のためのテスト ・システム全体の運用確認 ・システム性能の確認 ・対外接続システムとの確認		
導　入	主要作業	新システムの稼働準備を行う． 知識ベースの保守体制を確認する．	導入計画書(☆) 教育計画書(☆)	システム開発者 システム利用者
	作業内容	開発システムの本番稼働の準備と移行作業 ・利用者の教育 ・データベース，プログラム等の移行 ・新システムの作動確認		
保守/評価	主要作業	稼働したシステムの評価と維持． システム維持と知識ベース保守のための必要な手続きの設定と見直しを行う． 知識や環境の変化に対応し，知識ベースを更新する．	システム開発完了報告書 知識ベース保守マニュアル	システム開発者 業務専門家 システム利用者
	作業内容	・開発したシステムの効率，運用状況等の評価 ・稼働後のシステム維持(システム変更，バグ対応)と手順の見直し ・知識ベースの保守		

(注)成果物の☆印はその工程以前に作成される．

ョンルールから利用する一定の処理手順を定義するルール関数，フレームの状態が変化したとき自動的に実行する付加手続き（デモン）等から構成される．手続きプログラムは，C言語，FORTRAN，COBOLで記述する．

2) 推論機構……推論機構には，推論状態の監視にインシデンステーブルと呼ぶ独自の手法を採用した高速の前向き推論エンジンを備えている．ルールモジュールを用いた推論では，メッセージボードと呼ぶ制御フレームを用いて，連鎖の発生する範囲を限定した推論が実行できる．これによりプロダクションルールの表現が簡略化でき，推論の効率化が図れる．ルールモジュールやメッセージボードを用いることにより，知識を活用するための知識であるメタ知識を用いた推論が実現できる．また，図4に示すような階層的分散協調型システムが実現できる．

一連の推論処理における推論状態の保存/回復や推論の初期状態設定等を効率化する機能として，ワーキングメモリの保存/回復機能がある．

3) システムインタフェース……GNOSIS-IIは組込み型のES構築支援ツールであり，GNOSIS-IIが組み込まれる知識処理プログラムやGNOSIS-II内の手続きプログラムから，推論の起動やワーキングメモリの設定・更新・検索等を行うためのライブラリ群を備えている．これらのライブラリは，C言語，FORTRAN，COBOLから使用することができる．

4) 稼働機種……GNOSIS-IIは，シリーズ2200/1100，UNIXワークステーション(USファミリ，SunSPARC*，U 6000シリーズ，HP 9000シリーズ**)で稼働する．

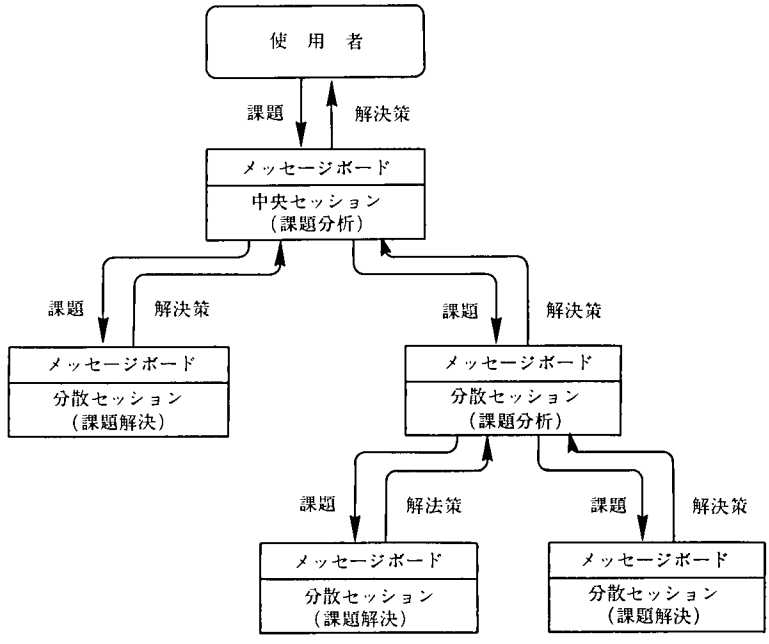


図 4 GNOSIS-IIを用いた階層的分散協調型システム

Fig. 4 Hierarchical distributed co-operate system with GNOSIS-II

* SunSPARCは，日本サン・マイクロシステム(株)の製品である．

** HP 9000シリーズは，横河・ヒューレット・パッカード(株)の製品である．

GNOSIS-IIの稼働には，特別なハードウェア環境やソフトウェア環境を必要としない．GNOSIS-IIの提供方法には，開発・実行環境（フル版）と実行環境（ランタイム版）とがある．

3.1.2 GNOSIS-IIのソフトウェア構成

GNOSIS-IIは，知識ベースコンパイラと対話型デバッガから構成される開発環境，推論エンジンとアプリケーション・インタフェースから構成される実行環境，推論の実行に必要な知識と推論の状態を格納する知識ベースから構成されている．GNOSIS-IIのソフトウェア構成を図5に，知識ベース構造を図6に示す．

知識ベースコンパイラは，GNOSIS-IIの構文規則に従って定義された知識ベース

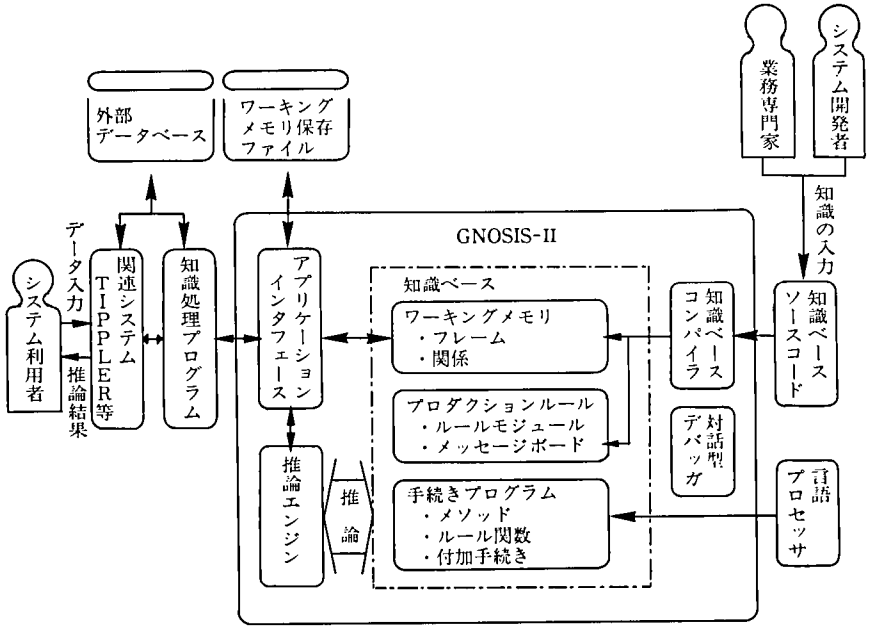


図 5　GNOSIS-IIのソフトウェア構成

Fig. 5　Software structure of GNOSIS-II

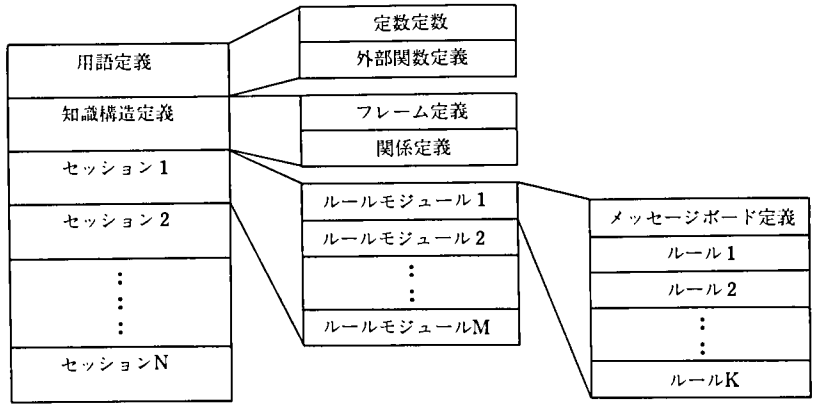


図 6　GNOSIS-IIの知識ベース構造

Fig. 6　Knowledge-base structure of GNOSIS-II

ソースコード（フレーム，関係，プロダクションルール等）を推論機構が処理できる形に翻訳し，知識ベースに格納するプログラムである．対話型デバッガは，知識ベースのデバッグを支援する機能群で，トレース，トラップ，ステップモード，フレームダンプ等を備えている．推論エンジンは，GNOSIS-IIの推論全体を制御する機構である．アプリケーション・インタフェースは，GNOSIS-IIと他のプログラムとのインタフェースをとるライブラリ群である．

3.1.3 GNOSIS-IIの適用分野

GNOSIS-IIは，これまで企業の専門家にしか行えなかった業務の新規システム化や，既存システムの中で専門家のノウハウを必要としていた分野のシステム化に有効なツールである．GNOSIS-IIは組込み型のES構築支援ツールであり，既存のシステム資産を損なうことなく，システムの知的なレベルアップができる．GNOSIS-IIの適用分野は，特定の業種や業務に限定されず，広く活用できる．とくに，問題領域としては強力な前向き推論エンジンを備えていることから，計画・設計型の問題解決に威力を発揮する．

GNOSIS-IIの主な適用業務には，生産計画，日程計画，宿日直要員計画，プラント操業計画，船積み計画，工法選定，部品設計，設備配置設計，資金運用相談，年金相談，運行ダイヤ編成，時間割編成，広告自動配列，テレックス電文解読等がある．

3.2 知的活動支援プラットフォームソフトウェアTIPPLER [3][4]

知的活動支援プラットフォームソフトウェアTIPPLERは，UNIXワークステーション上で知的活動支援システムを構築するための統合開発支援ツールである．知的活動支援システムとは，経験やデータを知識化した組織としての活用を支援するシステムのことで，利用者の創造的思考や最適行動を援助し誘導するシステムである．

3.2.1 TIPPLERの機能と特徴

TIPPLERは，UNIXワークステーション上でGUIを使ったビジネスアプリケーションを構築する際に有効なツールである．TIPPLERは，次の機能と特徴を備えている．

1) オブジェクト指向プログラミング言語UNISCRIPT……従来のプログラミング言語が持つ基本的な型と制御構造を踏襲し，さらにクラス/インスタンス，属性，メッセージ，継承等のオブジェクト指向技術を取り入れたプログラミング言語である．UNISCRIPTは，スロット（データ定義），メソッド（手続き定義），プレゼンテーション（表現定義）から構成されている．プロトタイピング手法による高開発生産性の実現が可能なプログラミング言語である．

2) 開発環境UNIGUIDE……効率的で快適な開発環境を提供するため，アイコン等の画面操作でUNISCRIPT言語が生成できるビジュアル・エディタ，UNISCRIPT言語の作成と更新に関する多彩な機能を備えたブラウザ，プロトタイピング開発に有効なインタプリタ，UNISCRIPT言語をC言語に変換するコンパイラ，プログラム部品や開発ドキュメントを格納するリポジトリを備えている．ボタンやメニュー等マウス操作を中心とした優れたLook & Feelを実現したアプリケーションが効率良く開発できる．

3) 豊富なプレゼンテーション機能……一つのクラスには，グラフやスプレッドシ

ート等，複数のプレゼンテーションを定義できる．このため，表示データ間の連動を意識することなく，同一情報をさまざまな視点から表示できる．ボタンやフィールド，ボックス，スクローリングリスト等，GUI 画面の作成に必要な各種の組込みライブラリとビジネスアプリケーションの作成に欠かせない XY グラフ（棒，折線，積上棒，山積み，高低）や円グラフ，相関グラフ，ガントチャート，レーダチャート等の多彩なビジネスグラフを備えている．音声や映像等，マルチメディアデータも取り扱える．

4) 他システムとの連動……データベースソフトウェアやエキスパートソフトウェア，統計解析ソフトウェア，通信ソフトウェア等とのインタフェースを備えている．知的活動支援システムと既存システムや知識処理システムが連動したアプリケーションを容易に構築できる．

3.2.2 TIPPLER のソフトウェア構成

TIPPLER は，UNIX オペレーティングシステム上のウィンドウ環境をベースに，UNISCRIPT 言語等の開発環境，GUI 部品の構築とビジネスグラフやスプレッドシート等の作成を支援するライブラリ類，他システムとの連係を支援するハイブリッド型インタフェース等から構成されている．TIPPLER のソフトウェア構成を図 7 に示す．

3.3 GNOSIS-II と TIPPLER の活用

GNOSIS-II のフレームによる知識表現と TIPPLER のクラスによるデータ表現は，オブジェクト指向の観点から親近性が強い．また，GNOSIS-II が対象とする知識処理と TIPPLER が対象とする知的活動支援は，知識情報処理システムの構築において相互補完と協調が必要な分野である．さらに，開発方法には共にプロトタイピング

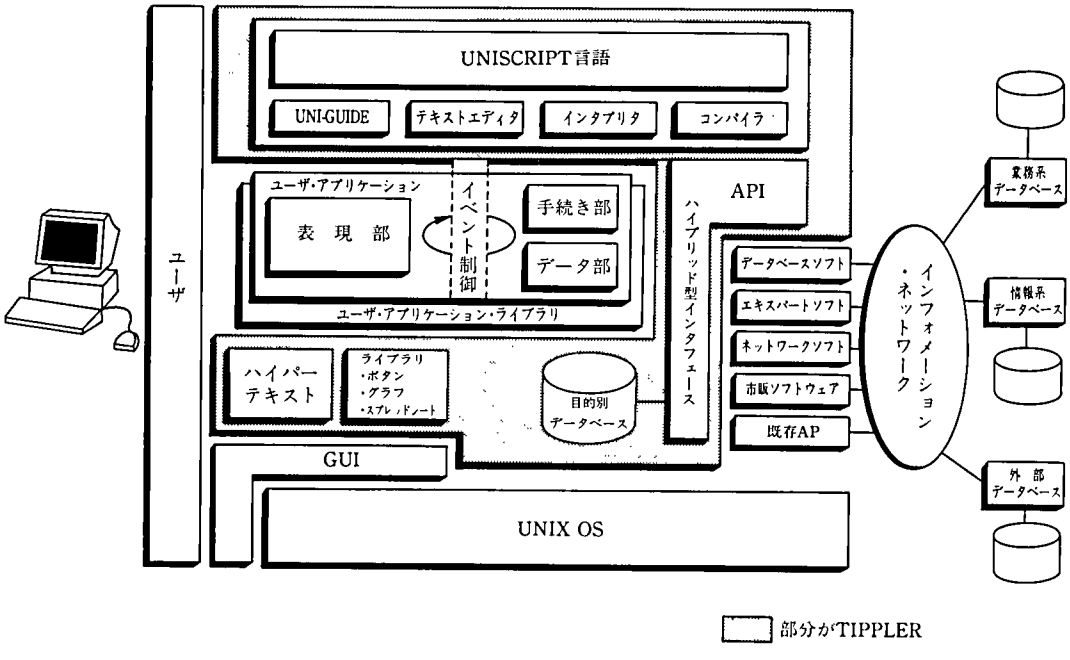


図 7　TIPPLER のソフトウェア構成

Fig. 7　Software structure of TIPPLER

手法の採用が適している．つまり，GNOSIS-IIとTIPPLERを活用すれば，人にやさしい知的なシステムが効率良く開発できる．ここでは，GNOSIS-IIとTIPPLERの活用事例として，機械割当てスケジューリングの概要を紹介する．

3.3.1 機械割当てスケジューリング

機械割当てスケジューリングとは，部門レベルの日別/ワークセンタ別の機械割当て/投入順序計画のことで，何時どの機械にどの作業を投入するかを決めることである．機械割当てスケジューリングのシステム要件とシステム構成を示す．

1) システム要件……機械割当てスケジューリングでは，多種多様な条件を考慮した最適な計画作成が必要となる．たとえば，機械に関しては，①能力，②性能，③稼働時刻等の条件，作業に関しては，①段取時間，②加工時間，③加工サイズ，④納期優先度等の条件，スケジューリングに関しては，①機械能力と制約の遵守，②機械性能の考慮，③機械負荷の平準化，④総段取り時間の最小化，⑤作業投入順序制約の遵守，⑥納期優先度の考慮等の条件を考慮する必要がある．とくに，スケジューリング条件では業務専門家の知識やノウハウの活用が重要な要素となる．また，計画調整では特急オーダや作業実績への対応等が重要が機能となる．つまり，知識処理と知的活動支援とを協調したシステム構築が必要となる．

2) システム構成……機械割当てスケジューリングは，工場レベルの生産日程計画システムからデータを入力し修正する機能，計画の自動作成機能，計画の調整機能，現場レベルの作業指示/実績収集システムとの接続機能から構成されている．機械割当てスケジューリングの機能構成を図8に示す．

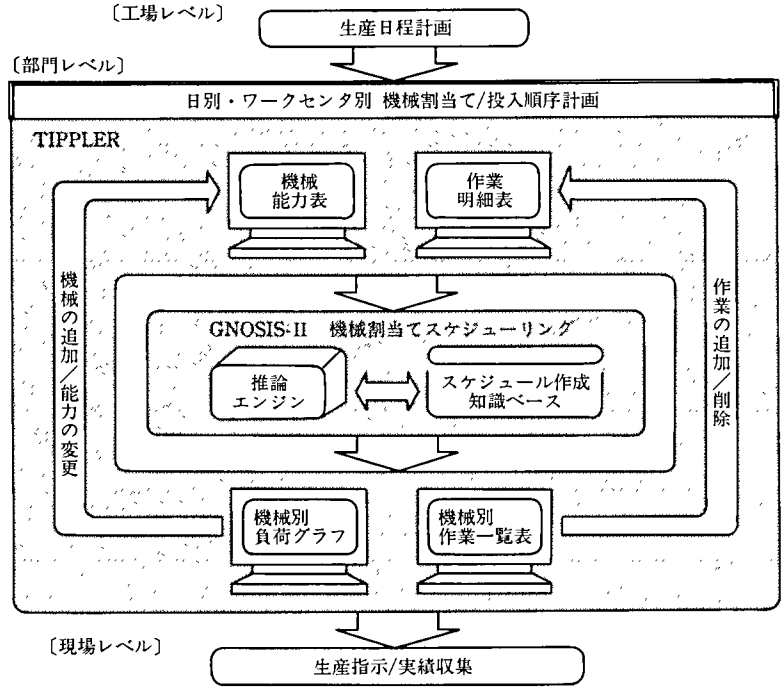


図 8 機械割当てスケジューリングの機能構成

Fig. 8 Functional structure of machine assignment system

計画の自動作成機能は，業務専門家の知識とノウハウを知識ベース化したスケジューリングESであり，GNOSIS-IIを用いて構築している．知識ベースの構造を図9に示す．図の（a）フレーム/関係図では，フレーム型を四角の枠で，枠内にフレーム型名を，フレーム間の関係型を矢印付の実線で，実線上に関係型名を表記している．また，矢印付の破線は，フレーム間の関係をフレームの属性（スロット）で表現していることを示す．図の（b）ルールモジュール構造図では，ルールモジュールを四角の枠で，枠内にルールモジュール名を，ルールモジュール間の関係を実線で表記している．

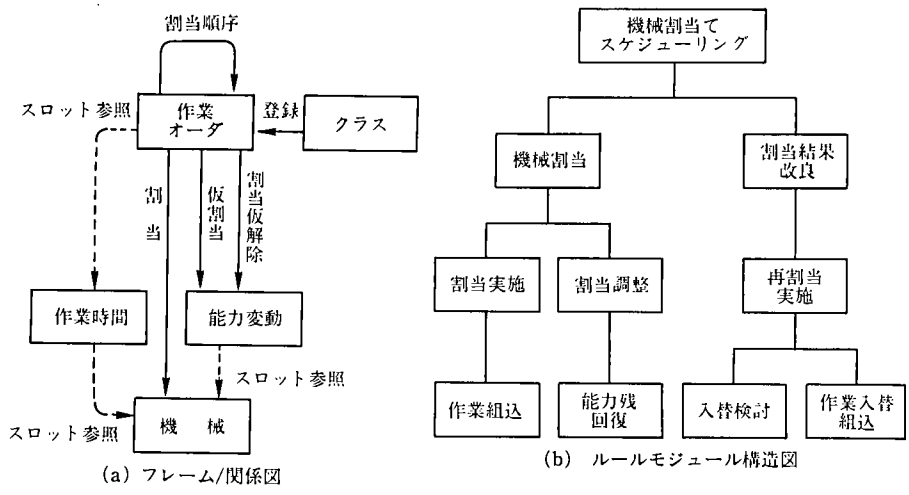


図 9　機械割当てスケジューリングの知識ベース構造

Fig. 9　Knowledge-base structure of machine assignment system

データの入力と修正，計画の調整，作業指示/実績収集システムとの接続の各機能は，TIPPLERを用いて構築している．とくに，計画調整ではESが作成した結果を利用者が判断し，最適な実行計画に修正する機能が重要となる．この部分には，TIPPLERのプレゼンテーション機能を全面的に活用している．計画調整機能の一例として，機械別負荷グラフの例を図10に示す．この画面は，ESが作成したスケジュール結果の表示画面であると同時に，利用者が計画を調整する際の操作画面でもある．作業の移動や入れ替え等の計画調整は，ガントチャート上の作業をマウスでドラッグすることによって行える．

3.3.2 適 用 事 例

GNOSIS-IIとTIPPLERを活用したES構築事例には，この他，信号連動図表作成システム，案内広告自動配列システム，シールド施工計画策定システム，バスダイヤ自動編成システム，学校時間割編成システム等がある．

4. 今後のエキスパートシステム開発・実行環境

近年，ESは専門家の知識やノウハウを知識ベース化し問題解決に活用するシステムとして実用化が進展してきた．今後のESは，これまでの主として個人業務を支援・代行するシステムから，企業レベルで知識ベースを活用し組織業務を支援・代行する

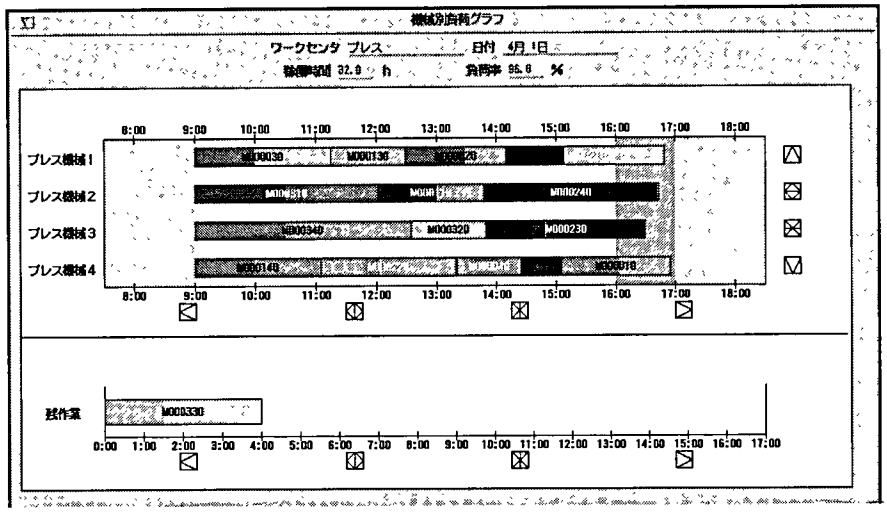


図 10　機械別負荷グラフの例

Fig. 10　Example of machine loading chart

システムに発展すると考えられる．このようなシステムを，従来の ES と区別する意味で，知識ベースシステム（KBS：Knowledge Base System）と呼ぶ[5]．

高度な問題解決能力が要求される KBS の実現には，事例ベース推論や高次推論，協調分散型推論等の新しい推論技術の採用，知識の部品化/再利用，知識ベースの共有化，オブジェクト指向技術の活用，データベース技術との連携，CASE との連携，KBS 開発方法の整備/体系化，クライアント-サーバ・アーキテクチャの採用等が必要となる．

このような ES が発展した形として構築される KBS の開発・実行環境には，次の機能が必要となる．共通基盤には，組織業務を代行・支援するための共有知識を格納する知識データベース，個々の KBS と知識ベースリポジトリや知識データベース，既存データベース，既存システムとの間を接続するネットワーク環境等が必要である．開発環境には，知識の部品化/再利用や知識ベースの開発管理等を支援する知識ベースリポジトリ，業務知識の獲得や知識ベースの分析・設計・作成等を支援する知識ベース開発支援ツール等が必要である．実行環境には，事例ベース推論や高次推論等を備えた高度な推論機能，個々の知識ベースと共有の知識データベースを相互活用して推論を行う協調分散型の推論機能，個々の KBS と知識データベースや既存データベース，既存システムとの連携を支援するインタフェース機能等が必要である．KBS の開発・実行支援環境を図 11 に示す．

このような KBS の構築に必要な開発・実行支援環境は，現在の GNOSIS-II や TIPPLER，オブジェクト指向データベース等を技術基盤として構築することができる．

5. おわりに

ES をはじめとする各種 AI（Artificial Intelligence：人工知能）技術は，もはや特別視する存在ではなく，今日では問題解決手法の一つとして，あるいはシステムを知

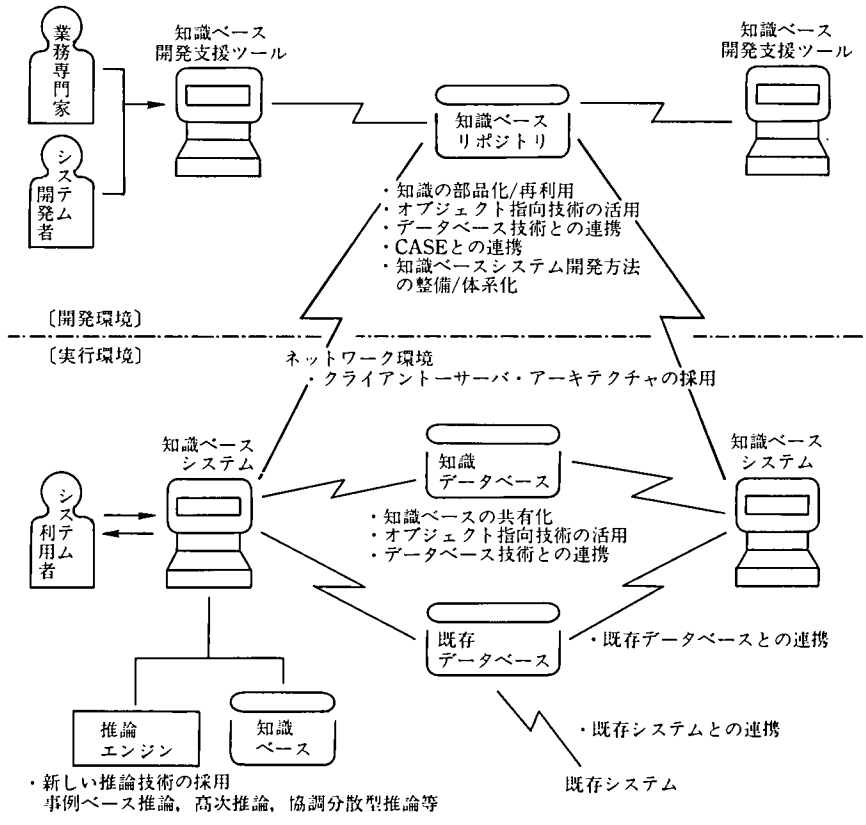


図 11　知識ベースシステムの開発・実行支援環境

Fig. 11　Development and run-time supporting environments of knowledge base system

的にレベルアップする技術として必須のものである．

当社では，これまでESの実用化を目指して，エキスパートシステム構築支援ツールGNOSIS-IIや知的活動支援プラットフォームソフトウェアTIPPLERの開発・改良に取り組んできた．また，ES等の知識情報処理技術を活用したシステム構築サービスの提供等を行ってきている．今後も，さらに知的なシステムを効率良く開発・実行できる環境を目指して，各種ツールの開発やサービスの提供等に積極的に取り組んでいくつもりである．

本稿が一助となり，ES等の知識情報処理技術が今後さらに広範なアプリケーションで実用に供されることを期待したい．

参考文献　［1］ ICOT-JIPDEC AIセンター編，“AI白書1992—人工知能の技術と利用—”，(財) 日本情報処理開発協会，1992.
［2］「エキスパートシステム構築支援ツールGNOSIS-II解説書」，ユニシス・マニュアル，481745901.
［3］「統合開発支援ツールTIPPLER概説書」，ユニシス・マニュアル，481743901.
［4］「統合開発支援ツールTIPPLERプログラミング解説書」，ユニシス・マニュアル，

481745902.
［5］ 日経 BP 社，“特集：進化する知識ベース・システム”，日経インテリジェントシステム別冊 1992 秋号，1992.

執筆者紹介 東 野 康 臣 (Koshin Higashino)
昭和 24 年生．46 年芝浦工業大学工業経営学科卒業．同年日本ユニシス(株)入社．主に OR 関連のシステム開発に従事．現在，システム技術本部 知識システム部 企画課課長．人工知能学会会員．

仕様言語Zとモデル化の考え方

Specification Language Z and a New Style of Describing Models in Z

染谷　誠

要　約　仕様言語Zを紹介し，その上で，Zによる仕様記述のスタイルについて，一提案を試みる．

Zが支援するのは仕様記述だけではない．ソフトウェア開発の全般にわたる．この意味で，Zは言わば開発用言語なのだが，本稿では仕様言語の面だけを取り上げる．仕様記述に際して，Zが想定している対象世界の把握の仕方と，Z言語で中心的役割を演ずるスキーマ記法が，仕様の「部品化」と「再利用」をいかに自然に成し遂げているか，話題をこの点に絞って紹介する．

Zは汎用性の高い言語なので，Zが仕様を書く場合，さまざまなスタイルが可能である．にもかかわらず，すでに出版されているZの教科書に記載されている仕様の記述例を見ると，どういう訳か皆よく似ている．しかも困ったことに，どれも感心しない．そこで，そのどういうところが感心しないのか，具体例について具体的に批判する．その上で，ではどう書けば良いのか，われわれの記述スタイルを提案する．

Abstract　The author first introduces specification language Z, and then proposes a different style of describing models using the language.

What Z supports is not limited only to the description of software specifications but covers all aspects of software development effort's. In this context, Z can be said to be what is called a language for software design, but this paper specifically deals with the aspect of Z being a specification langugage. The major focus is placed on what the point of view is, which is assumed in Z for comprehending of object worlds, and on how naturally the schema notation available in Z as its key role can provide a technique for modularizing and re-using common parts.

Z, which provides a high level of general-purpose use, allows varieties of model description styles to exist. The author's pending question is that, despite this characteristic of Z's, sample model description styles as seen in several Z textbooks already published are quite alike for unknown reasons, and what is worse, are far from satisfactory.

Then this paper points out where the author feels uncomfortable about those samples, and suggests a better way to describe models through showing the author's description of the library problem.

1. はじめに

本稿の目的は二つある．

一つは，Zがソフトウェアの開発中に出会う諸問題の記述のための枠組みとして，極めて優れた特性を備えていることを紹介し，さらにそれを具体例を通して実地に味わってみることである．ここで，優れた特性とは，

- 表現力が豊かで，広範囲の課題が記述できる，
- 大規模な課題の仕様でも，書きやすくかつ読みやすくするため，仕様を構造化する仕組みを持っている，

・正確で曖昧さのない記述ができる，
・記述された仕様について，その諸性質が議論できるような論理計算が用意されている，

ことを言う．Zの用途は決して仕様記述にとどまるものではない．仕様から対象システムの諸性質を導くための議論の記述や，仕様からプログラムコードを導出する過程の記述も，Zの守備範囲に入る．この意味で，Zはソフトウェアの開発用言語なのである．だが本稿で取り上げるのは，仕様記述の側面だけである．仕様書の品質向上こそ急務だと思うからである．仕様から対象システムの性質を導くための議論や，コードの導出については次会を期したい．

もう一つの目的は仕様記述のスタイルについてである．Zについては，すでに教科書が何冊も公にされている．Zの基本を理解するためには，いずれも優れた教科書だが，そこに載っている仕様記述例はどれも気に入らない．稚拙というのでは勿論ない．それどころか，むしろ洗練された書きっぷりといえる．だが，何かしら違和感を感ずる．この違和感の淵源は，どうやらモデルについての考え方の違いにあるようだ．そこで，そのような仕様の具体例として，文献[5]から図書館問題の仕様を取り上げ，その仕様のどんなところが不満なのか具体的に批判した上で，その改善案として一つの記述スタイルを提示したい．これがもう一つの目的である．

2. Z とは

2.1 Z の基本

Zとは何か．Zは何よりもまず集合論である．ただし，Zは標準的な集合論よりも弱い集合論である．

こう言うと，きっと疑問に思われるだろう．なぜ集合論なのか．なぜ集合論がことさら開発用言語にふさわしいと言えるのか，と．

集合論というのは，実は理論構築のための普遍言語なのである．その意味は議論の対象は何であれ，その議論が客観的な議論である限りすべて集合論で記述できる，ということである．これについては，ここで詳しく述べるわけにはいかないが決して根拠薄弱な勝手な主張ではないこと，この点は強調しておかなければならない．これは，いくつかの分野の理論を実際に記述することを通して得られた経験的事実を根拠として立てられた確かな見通しであり，定説として広く受け入れられている．ソフトウェアの開発も，その対象はありとあらゆる分野に及ぶ．したがってその開発用言語には普遍言語，つまり集合論が望ましい．

2.2 Z の特徴

集合論が望ましいことは一応認めてもらうとしても，なぜZなのか．依然この問題が残る．集合論といってもいろいろあるだろう．その中で，なぜZがことさらソフトウェアの開発用言語にふさわしいと言えるのか．

その理由はZの提供する記述言語にある．数ある集合論の枠組み中で，Zがとくにソフトウェア開発のための記述言語に向いていると言えるのは，その記述言語上の工夫である．Z言語は，一階の述語論理の上に組み立てられた集合論であるが，これに，ソフトウェアの開発用言語として使いやすくなるよう，次のような記述上の工夫を凝

らしている．

・強い型付け：変数は必ずその型を宣言してから使う

・スキーマ記法：文書の構造化（モジュール化）のための記法

・スキーマ計算：スキーマを対象とする論理計算

強い型付けは，集合論を構築する上で理論上の都合もあるが，むしろ実用の見地から大きな意味を持つ．型検査という仕様記述上の誤りを検出するための，簡単で有効な手段を提供してくれるからである．

スキーマ記法．これが，Z言語に特有の，理論の構造化のための，記述の仕組みである．スキーマ記法には，何種類かのスキーマの書き方のルールと，既存のスキーマから新しいスキーマを作り出すためのルールがある．この記法を使って，Z仕様はスキーマを構成要素とする階層構造を持った構築物として記述する．この記法があること，これこそZを他の集合論から区別し，ソフトウェアの開発用言語たらしめている理由である．

この構造化記法の利点については，いろいろな言い方ができるが，ここではこれがあれば巨大なソフトウェアの仕様も読み書きが可能となること，この点を強調しておきたい．形式仕様は，小さな課題なら書けるが，巨大ソフトウェアの仕様はとても書けるものではない．そういう意見をしばしば耳にするからである．

また，スキーマ記法は構造化のための記法だから，仕様の「部品化」と「再利用」を支援するのは当然だが，その徹底ぶりも注目に値する．併せて強調しておきたい．

スキーマ記法は，文書の構造化のための便宜にとどまらない．論理計算をも構造化する．そのため，大きな仕様についても，その性質が論理計算によって議論できる．

スキーマ記法こそZ言語のキーコンセプトである．

2.3 スキーマの基本型

では仕様の構成要素となるスキーマとは何か，それはどのように書き，どう使うのか．

その記述形式はまことに単純である．スキーマには3種類あるが，その基本型は

スキーマ名	
特性部	(変数の型宣言)
叙述部	(制約条件)

である．まったく同じことだが，これは次のように書いてもよい．

スキーマ名 ＝ ［特性部 ｜ 叙述部］

スキーマ名に当のスキーマの名前を指定する．スキーマは一度定義すると，スキーマ名を引用することによって，何回でも再利用できる．

特性部(signature) では変数（識別子）とその型を宣言する．ここでは記述対象を特徴付ける基本属性（変数）を提示したり，叙述部で使う用語（変数）を準備する．

叙述部には，特性部で提示した変数を含む命題(式)を列挙する．列挙は，暗に個々の命題（式）の論理積（AND結合）を意味する．ここで，特性部で提示した用語（変数）を使って，記述対象について叙述する．特性部の変数が充たすべき制約条件を列挙するわけである．何も言う必要がなければ，ここは省いてよい．

これがスキーマの記述形式である．特性部で宣言された個々の変数（識別子）をそのスキーマの**成分**という．

2.4 スキーマの記述例

スキーマ理解の第一の鍵は，この形式の適用範囲がいかに広いか，この点を認識することである．

X.1 平面幾何 *POINT* と *LINE* を所与の集合とする．その表明は次のように書く．

$$[POINT,\ LINE]$$

所与の集合とは基本となるデータ型のことで，無定義概念に相当する．このとき平面幾何は次のスキーマで表現できる．

PlainGeometry

$$\_isOn\_ : \mathbb{P}(POINT \times LINE)$$
$$\cdots$$

$$\forall p_1, p_2 : POINT \mid p_1 \neq p_2 \bullet (\exists_1 l : LINE \bullet p_1\ isOn\ l \wedge p_2\ isOn\ l)$$
$$\cdots$$

POINT × *LINE* は *POINT* と *LINE* の直積．$\mathbb{P}\ X$ は X のベキ集合．*isOn* は $\mathbb{P}(POINT \times LINE)$ の要素．したがって，

$$isOn \subseteq POINT \times LINE$$

要するに *isOn* は *POINT* と *LINE* との関係である．特性部に*_isOn_*とあるのは，*isOn* が中置の関係であること，つまり引き数が *isOn* の前後にくることの表明．つまり，任意の点 p と任意の直線 l について，

$$p\ \ isOn\ \ l \Leftrightarrow (p,\ l) \in isOn$$

となる．

叙述部が語っていることは，「二点を通る直線が一意的に存在すること」である．

X.2 格子点 任意の格子点は次のスキーマで表現する．

Point

$$x : \mathbb{Z}$$
$$y : \mathbb{Z}$$

第一象限に限ると

Point

$$x : \mathbb{Z}$$
$$y : \mathbb{Z}$$

$$0 \leq x$$
$$0 \leq y$$

となる．

X.3 社員レコード *NAME, ADDRESS, DATE* を所与の集合，すなわち基本データ型とする．

$$[NAME,\ ADDRESS,\ DATE]$$

このとき，社員レコードは次のようなスキーマになる．

社員レコード

$name : NAME$
$addr : ADDRESS$
$birthday : DATE$
$employedday : DATE$
$\ldots$

X.4　長さが *maxsize* 以下の待ち行列　待ち行列の構成要素は所与の集合 *ITEM* の要素とする．

$[ITEM]$

maxsize が正の整数であることの表明の仕方は後述する．

Queue

$queue : \mathrm{seq}\ ITEM$

$\#queue \leq maxsize$

1) 成分は *queue* だけ．seq は集合生成子．seq *ITEM* は *ITEM* の要素の有限列全体の集合．したがって，*queue* は *ITEM* の要素の有限列を値に持つ．
2) # は有限集合の要素数を返す演算子．だから# *queue* は「*queue* の長さ」と同じ．叙述部は「待ち行列の長さは *maxsize* 以下である」の意味．この待ち行列が常に満たすべき制約条件を記述している．

X.5　待ち行列へ項目を追加する操作

EnQueue

$queue, queue' : \mathrm{seq}\ ITEM$
$i? : ITEM$

$\#queue \leq maxsize$
$queue' = queue^\frown \langle i? \rangle$
$\#queue' \leq maxsize$

1) $i?$は待ち行列に追加すべき入力データ．その型は *ITEM*．入力データ名には?を付けるのがZの約束である．
2) *queue* がこの操作を施す前の値，すなわち**事前値**で，*queue'* がこの操作を施した後の値，すなわち**事後値**を表す．プライムのある，なしで，事後値と事前値を区別する．
3) 叙述部に制約条件が三つ並んでいるのは，三条件の論理積(AND)を意味する．
4) 第一の条件は「この操作を行う前の待ち行列（*queue*）の長さが最大長 *maxsize* 以下である」ことの表明．
5) 第二の条件の右辺にある$^\frown$は二つの列の連接を作る操作．だからこの制約条件は，「*queue'*（事後値）は *queue*（事前値）と$\langle i? \rangle$との連接である」との主張．$\langle i? \rangle$は入力 $i?$の単一列．要するに，事後値 *queue'* が事前値 *queue* の最後に入力項目 $i?$を追加した列であることを要請している．
6) 第三の条件は「この操作の実行後，つまり項目の追加後も，待ち行列の長さが

最大長 *maxsize* 以下である」ことを要請している.

7) ところで,第一の条件はこのままでよいのだろうか.仮りに#*queue* = *maxsize* でも,この条件を満たしてしまうが,これでは項目を追加しようがない.だから #*queue* < *maxsize* と改めるべきではないか.

これで一向に差し支えない.第二,第三の制約条件を勘案すると,その条件 #*queue* < *maxsize* が導かれる.

$$\begin{aligned} &\#queue' \\ &\quad = \#(queue \frown < i? >) \\ &\quad = \#queue + 1 \\ &\quad \leq maxsize \end{aligned}$$

したがって

$$\#queue < maxsize$$

このように,理屈の上では#*queue* ≤ *maxsize* で差し支えないが,実用の仕様では,この例なら,もちろん#*queue* < *maxsize* と書くべきだ.

X.6 待ち行列の先頭項目を取り出す操作

DeQueue

$queue, queue' : \text{seq}\, ITEM$
$i! : ITEM$

$\#queue > 0$
$\#queue \leq maxsize$
$queue = < i! > \frown queue'$
$\#queue' \leq maxsize$

1) *queue* と *queue'* は待ち行列の状態の事前値と事後値.

2) *i*! は出力データ.出力データ名には必ず! を付けるのが Z の約束. *i*! がどのような出力になるのか,この段階ではまだわからない.ここでは,ただ出力 *i*! (∈ *ITEM*) があることを予告するだけ.

3) さて,先頭項目を取り出すためには待ち行列が空っぽでは困るが,このスキーマの叙述部は,この点をどう語っているか.事前値 *queue* に関する制約条件を見よう.

$$\#queue > 0$$

$$\#queue \leq maxsize$$

この第一の条件で「待ち行列が空っぽではない」ことを要請している.

4) 次は,この操作を施した結果である.果たして,予告された出力 *i*! は本当に待ち行列の先頭項目なのか.さらに,待ち行列の事後値が,事前値から先頭項目を除いた残りの列になるのか.叙述部はこの点をどのように記述しているだろうか.出力 *i*! と事後値 *queue'* に関する制約条件を見よう.

$$queue = < i! > \frown queue'$$

$$\#queue' \leq maxsize$$

第一の条件の右辺にある⌢は二つの列の連接を作る操作. <*i*!> は出力 *i*! からなる単一列.とすれば,この一個の条件から,「*i*! が待ち行列の事前値の先頭項目

である」こと，および「待ち行列の事前値 *queue* から先頭項目を除いた残りが事後値 *queue'* になっている」ことが見て取れる．

以上のように，Zではあらゆるものごとをスキーマで表現する．では，スキーマを使って仕様書がどのように構成されるのか，次は仕様書の全体的な構成を見ておきたい．

3. Zによる仕様記述

3.1 モデルの構成要素

仕様書には何を書くのか．対象世界のモデル記述をもってその仕様とする．これがZ仕様記述の基本的な考え方である．

では，対象世界のモデルとは何か．何を書けば対象世界のモデルを記述したと言えるのか．Z仕様で想定するモデルとは**状態推移モデル**である．対象世界を**状態空間**とその**初期条件**，および**状態推移**を引き起こす**事象**を記述することによってモデル化しようというのである．

状態とは対象世界を構成する基本データ名の値の組，取り得る限りの値の組の全体を**状態空間**という．その状態の成分となる個々の基本データの名前は**状態**変数等ともいう．Zでは，個々の基本データ名あるいは状態変数は必ずデータ型を持つ．

用語法について一言．以下，状態と状態空間とは混同して使う．状態空間というべきところをしばしば状態で済ます．

一般に，対象世界は恒常的ではない．流動的である．対象世界の状態は変化する，ということである．対象世界にはさまざまな出来事が起きる．この出来事の生起によって，対象世界の状態が変化する．対象世界の状態を変化させる出来事を**事象**という．状態を変化させる出来事は，ある時は対象世界の存在者の**行動**であったり，ある場合には**操作**であったり，またある場合には**現象**といった方がふさわしかったり，具体的な現れは多様であるが，それらを抽象化して事象ということにする．Zでは，事象を構成する要件として，状態推移の外に，**入力**と**出力**を考えている．事象の生起時に**観測**される**入力**と**出力**と**状態**の**推移**を記述することによって，事象をモデル化しようというのである．

では，状態，初期条件，および事象はどう記述するのか．いずれもスキーマによって記述する．上記の **X3** は社員の状態空間と考えられる．社員の状態を構成する基本データ名を成分とするスキーマで社員の状態を表している．個々の成分あるいは，基本データ名は *NAME*，*ADDRESS*，*DATE* 等の型を持つ．

X4 は待ち行列の状態空間である．その状態を構成する成分は *queue* ただ一つ．その型は *ITEM* の有限列．叙述部では状態が恒常的に満たすべき制約条件を記述している．この意味で，叙述部に記載された制約条件を状態の**不変条件**等と呼ぶこともある．

X5，**X6** は待ち行列の操作（事象）の例である．**X5** には状態推移に加えて入力がある．**X6** では状態推移に加えて出力がある．入力や出力はどのようなものか．状態はどのように推移するのか．これらを規定するのが叙述部である．叙述部には，入力や出力や状態変数の事前値と事後値が満たすべき制約条件を列挙する．

3.2 仕様の構成

上述したように，Z仕様は対象世界の状態推移モデルを記述する．その主要な構成要

素は状態スキーマ，初期条件スキーマ，それに事象スキーマである．

ではZ仕様，つまりモデル記述は全体としてどのような構成になるか．この点を具体例によって示す．待ち行列の仕様の全体を書いてみよう．

3.2.1 待ち行列の仕様

記述すべきことがらは，

- 所与の集合（待ち行列の構成要素）
- 待ち行列の最大長
- 待ち行列の状態
- 待ち行列の初期条件
- 待ち行列への項目の追加の操作
- 待ち行列から先頭項目の取得の操作

である．これを書き下してみよう．

$[ITEM]$

$$maxsize : \mathbb{N}$$
$$maxsize > 0$$

Queue

$$queue : \mathrm{seq}\, ITEM$$
$$\#queue \leq maxsize$$

$Queue_{Init}$

$$Queue$$
$$queue = \langle\rangle$$

EnQueue

$$\Delta Queue$$
$$i? : ITEM$$
$$queue' = queue^\frown \langle i? \rangle$$

DeQueue

$$\Delta Queue$$
$$i! : ITEM$$
$$queue = \langle i! \rangle ^\frown queue'$$

3.2.2 補 足 説 明

1) はじめに記述すべきことは所与の集合である．ここでは，行列を構成する要素の集合 *ITEM* を所与の集合として記述してみる．その意味は，集合 *ITEM* は任意に外から与えられるもの．*ITEM* の要素が何であるのか，*ITEM* がどのように与えられるのかは一切問わない，と言う表明である．*ITEM* が所与の集合であることは次のように書く．

$[ITEM]$

2) 次は，待ち行列の最大長 *maxsize* の導入．

$$maxsize : \mathbb{N}$$
$$maxsize > 0$$

これもスキーマの一種で，**公理スキーマ**という．公理スキーマで宣言された変数は，仕様中，それ以降のどこからでも参照できる．このスキーマは，*maxsize* が正の自然数であることを表明している．が，これが待ち行列の最大長であるかどうかは，まだわからない．

3) 次は，待ち行列の状態スキーマの提示．

Queue
$$queue : \mathrm{seq}\, ITEM$$
$$\#queue \leq maxsize$$

この状態スキーマの成分はただ一つ *queue* だけ．その型は *ITEM* の有限列．seq は集合生成子，与えられた集合(この場合は *ITEM*)の要素の有限列全体を作り出す演算子である．叙述部の条件は，待ち行列に蓄える *ITEM* の要素の個数が *maxsize* 以下であることを要請している．これで，さきに導入した *maxsize* に「待ち行列の最大長」との意味付けがなされた．# ついては **X.4** を参照．

4) 次は初期条件スキーマ

$Queue_{Init}$
Queue
$$queue = <>$$

① これは特別の意味を持つスキーマ，状態スキーマ *Queue* の初期値の満たすべき条件を規定するスキーマである．< >は空列の意．だから，このスキーマは「状態変数 *queue* の初期値は空列」だと規定している．

② 特性部にスキーマ名を書くことを「スキーマ包含」という．「スキーマ包含」については後述するが，これは，その名指されたスキーマ，この場合は *Queue* をそっくり取り込むことの表明である．だから，こう書くだけで，*Queue* の特性部をこのスキーマの特性部に，*Queue* の叙述部をこのスキーマの叙述部に，そっくり複写したことになる．

③ スキーマの叙述部に現れる変数は必ず特性部で宣言する．これがスキーマ記述の鉄則だが，見たところ *queue* が叙述部にあるのに，特性部に見あたらない．これでよいのだろうか．

もちろんこれでよい．スキーマ *Queue* を包含しているからである．これで立派に変数 *queue* を宣言したことになる．状態スキーマ *Queue* を再利用した，記述の経済の例である．

5) 次は項目の追加と先頭項目取得の操作（事象）スキーマ.

EnQueue

$\Delta Queue$
$i? : ITEM$

$queue' = queue^\frown < i? >$

DeQueue

$\Delta Queue$
$i! : ITEM$

$queue = < i! > ^\frown queue'$

① ?の付いた変数は入力変数，!の付いた変数は出力変数であることはすでに述べた.

② ここで，前節の記述例 **X.5**, **X.6** と見比べてほしい．記述量がだいぶ違うことがわかるだろう．記述の経済!.

③ この二つの事象スキーマの特性部は，どちらも $\Delta Queue$ を包含している．詳細は後述するが，これを特性部に書くことで，状態スキーマ *Queue* とそれを「修飾」したスキーマ *Queue'* の両者を包含することになる．後述するように，$\Delta Queue$ があるだけで，*Queue* を更新することが見て取れる.

④ スキーマの叙述部に現れる変数は必ず特性部で宣言するのがスキーマ記述の鉄則だが，見たところ *queue* も *queue'* も叙述部にあるのに，特性部には見あたらない．これはルール違反ではないのか.

ルール違反ではない．これできちんとスキーマ記述の鉄則は満たしている．これが $\Delta Queue$ を包含していることの効用である．詳細は後述するが，*Queue* を包含すると，その特性部をそっくり取り込むから，「*queue* : seq *ITEM*」を宣言したことになる．同様に，*Queue'* を包含することで「*queue'* : seq *ITEM*」を宣言したことになる.

⑤ それともう一つ．この二つの事象スキーマの叙述部には「待ち行列の長さが *maxsize* 以下」という制約条件が見あたらないが，これでは待ち行列がその最大長 *maxsize* を越えるのを許してしまうのではないか.

許さない．これで待ち行列がその最大長 *maxsize* を越えないようしっかり要請している．これも $\Delta Queue$ を包含していることの効用．詳細は後述するが，スキーマ包含は特性部だけでなく叙述部も取り込む．だから，*Queue* を包含することで制約条件「$\#queue \leq maxsize$」を取り込み，*Queue'* を包含することで制約条件「$\#queue' \leq maxsize$」を取り込んでいる.

⑥ この二つの事象スキーマは $\Delta Queue$ を包含しているから，どちらも，*Queue* と *Queue'* の両者を包含している．ところが，初期条件スキーマは *Queue* しか包含していない．なぜか.

・初期状態はどのような状態か，これを規定するのが初期条件スキーマである．その規定には，状態変数が満すべき制約条件を列挙すればよいので，

状態スキーマ *Queue* を包含するだけでよい．

- これに対して，項目追加と先頭項目取得は状態推移を伴う操作（事象）である．したがってそれを記述するスキーマでは，状態がどう推移するのかを規定しなければならない．状態推移を記述するためには，状態変数の事前値と事後値を区別しなければならない．Zでは，事前値を表すには状態変数そのものを使用し，事後値を表すにはプライム'で修飾した状態変数を使用する．そこで，裸の状態変数を取り込むために *Queue* を包含し，プライム'修飾した状態変数を取り込むために *Queue'* を包含する．これを一挙に行うには Δ*Queue* を包含すればよい．

6) 初期化，項目追加，先頭項目取得．この三つの事象スキーマの記述例は，いずれも状態スキーマ *Queue* を再利用することで，特性部の記述についても，叙述部の記述についても，記述の経済を計っている．前節の記述例 **X.5**, **X.6** と見比べつつ，この点を味わってほしい．

3.3 状態の記述

Z仕様は，対象世界を**状態推移モデル**として記述する．その**状態**はもちろんスキーマで表現する．

状態とは対象世界を構成する基本データ名の値の組，取り得る限りの値の組の全体を状態空間といった．個々の基本データ名には，必ず取り得る値の範囲，つまりデータ型が決まっているものと考える．この基本データ名とその型を列挙したものが状態スキーマの特性部に外ならない．状態スキーマの**成分**を状態変数ともいう．なお，今まで通り，状態空間というべきところを状態で済ます．

一般には，状態変数はその型の勝手な値が許されるとは限らない．状態変数の値は恒常的に何らかの制約条件を満たすことが要請される．この恒常的な制約条件の記述が状態スキーマの叙述部を構成する．また，この制約条件を不変条件等ということについてはすでに述べた．

以上の点を，待ち行列の状態スキーマについて見てみよう．

Queue

$queue : \mathrm{seq}\ ITEM$

$\#queue \leq maxsize$

状態変数が *queue*，その型が seq *ITEM*，制約条件が

$$\#queue \leq maxsize$$

である．

以上で，状態スキーマの特性部の叙述部の意味合いは明かだろう．

3.4 事象の記述 = 状態の変化の記述

対象世界は流動的である．対象世界には諸々の出来事が起こり，対象世界の状態が変化する．対象世界の**状態推移**を引き起こす出来事を**事象**という．

さて，この事象はどうモデル化するか．既述したように，Zでは，事象の生起時に観測される**入力**と**出力**と**状態推移**を記述することによって事象をモデル化する．

ところで，**状態推移**を観測するとはどういうことか．少なくとも，そのためには，

個々の状態変数について，事象が起こる前の値と事象が起きた後の値とが区別して観測できなければならない．事象が起こる前の値をその状態変数の**事前値**，変化後の値を**事後値**と呼ぶ．状態変数の事前値と事後値が区別できれば状態推移は記述できる．

以上から，事象の生起を特徴づけるデータは

- 入力
- 出力
- 状態変数の事前値
- 状態変数の事後値

である．

3.4.1 観測データの名付ルール

事象の生起を特徴づけるデータは何か．Zでは，上のような種類のデータを想定し，それぞれの種類ごとにデータ名の付け方のルールを定めている．

- **入力データ**には，最後に「?」の付いた識別子を使う．
- **出力データ**には，最後に「!」の付いた識別子を使う．
- **事前値**は状態変数そのもので表す．
- **事後値**は状態変数にプライム「'」を付けて表す．

状態変数の事前値と事後値を区別すること，これがポイントである．これによって，状態変数の値の変化が簡単明瞭に記述できる．

Z仕様記述では事象もスキーマで表現する．事象スキーマの特性部でどのようなデータを提示し，叙述部にはどのようなことがらを記述するかはすでに明かだろう．

3.4.2 事象（操作）スキーマの特性部

事象（操作）スキーマの特性部では，

- 状態変数（事前値）
- プライム修飾した状態変数（事後値）
- 入力データ名
- 出力データ名

を宣言する．

具体例について見てみよう．待ち行列の操作「項目の追加」と「先頭項目の取り出し」スキーマの特性部は次のようになる．

EnQueue

$queue, queue' : \mathrm{seq}\ ITEM$

$i? : ITEM$

$\ldots$

DeQueue

$queue, queue' : \mathrm{seq}\ ITEM$

$i! : ITEM$

$\ldots$

どちらのスキーマも

$queue,\ queue'$: seq $ITEM$

を含んでいる．どちらの操作も状態推移を伴うからである．加えて，*EnQueue* には入力 $i?$が宣言されており，*DeQueue* には出力 $i!$が宣言されている．

スキーマ特性部では，あくまで事象すなわち状態変化を記述するために必要な関連データを列挙するだけである．どのように変化するかを記述するのは叙述部である．

3.4.3 事象（操作）スキーマの叙述部

スキーマの叙述部では事象の生起によって状態がどのように変化するか，またどのようなデータが入力され，どのようなデータが出力されるかを記述する．そのためには，状態成分の事前値と事後値，および入力と出力が満たすべき制約条件を列挙すればよい．

記述にさいしては，制約条件を二つのグループに分けて考えるとわかりやすい．事前条件と事後条件である．

はじめは事前条件について考える．事前条件とは，具体的には状態変数の事前値と入力データに関する制約条件で，その中にはプライムの付いた変数や！の付いた出力データは含まない．その意味合いは，一言でいうとその事象が起こる前の状態の記述だが，要するにその事象の生起以前の時点で状態成分の事前値と入力が満たすべき必要条件である．

その後で事後条件について考える．事後条件は，具体的にはプライムの付いた変数や！の付いた出力データを少なくとも一つ含む制約条件である．状態成分の事後値や出力が事前値や入力とどういう関係にあるのかを規定する制約条件である．その意味合いは，その事象の生起の結果の観測記録である．

ここでも具体例に則って考えてみよう．待ち行列の操作「項目の追加」と「先頭項目の取り出し」スキーマの叙述部を書いてみる．

EnQueue

$queue, queue'$: seq $ITEM$
$i?$: $ITEM$

$queue \leq maxsize$
$queue' = queue^{\frown} < i? >$
$queue' \leq maxsize$

DeQueue

$queue, queue'$: seq $ITEM$
$i!$: $ITEM$

$queue \leq maxsize$
$queue > 0$
$queue =< i! > {}^{\frown} queue'$
$queue' \leq maxsize$

両スキーマの叙述部の意味合いに付いてはすでに**スキーマの記述例**の **X.5**，**X.6** のところで詳述した．

ことに，それぞれの操作を行うための条件は，事前条件だけからでは必ずしも読み取れないこと，同様に操作の結果も事後条件だけからでは必ずしも読み取れないこと，どちらも叙述部全体を見ないとわからないことがあること等，を想起して欲しい．

事象スキーマの記述に際しての考え方は以上である．次は，事象スキーマを状態スキーマを再利用しながら，簡潔に記述するための記法である．

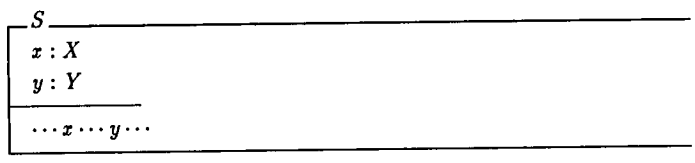
3.5 仕様の構造化 = 部品化と再利用

繰り返し述べたように，Z は既存のスキーマを再利用して新しいスキーマを組み立てるための構造化の仕組みを，豊富に用意している．そのいくつかを紹介しよう．

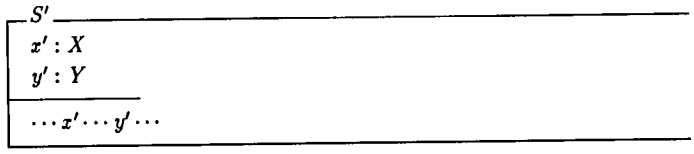
3.5.1 スキーマ修飾と成分名の変更

スキーマ修飾　与えられたスキーマ名に**修飾子**を付けると，もとのスキーマのすべての成分名にその修飾子を付けたスキーマを表す．修飾子にはプライム ′，疑問符 ?，感嘆符 !，下付き添え字$_{1,2,}$ …等がある．

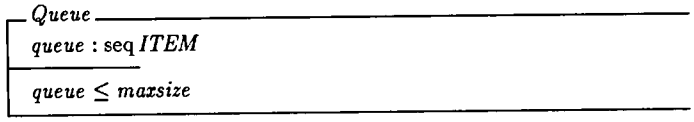
スキーマ S を



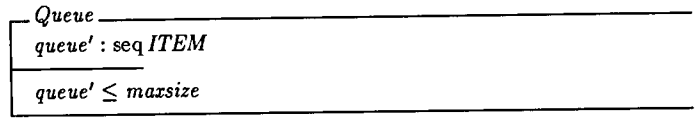
とする．このとき S' は



となる．



のとき *Queue′* は



となる．

成分名の変更　S がスキーマのとき

$$S\,[y_1/x_1,\ \cdots,\ y_n/x_n]$$

は，S の各成分 x_i をこれに対応する名前 y_i で置き換えて得られるスキーマを表す．たとえば $S\,[a/x,\ b/y]$ は次のようになる．

$S[a/x, b/y]$

$a : X$
$b : Y$

$\cdots a \cdots b \cdots$

成分名を変更する時には注意を要する．

- x_iは互いに異なる識別子でなければならない．
- x_iはすべてスキーマSの成分でなければならない．
- Sの二つの成分u, vが変更結果，同じ成分名に変わるなら，u, vは同じ型でなければならない．

3.5.2 スキーマ包含

既存のスキーマ名を特性部に書くと，そのスキーマをそっくり取り込むことを意味する．既存のスキーマの特性部を今書いているスキーマの特性部に，既存のスキーマの叙述部を今書いているスキーマの叙述部に取り込む．

たとえば

EnQueue

$Queue$
$Queue'$
$i? : ITEM$

$queue' = queue^\frown < i? >$

は二つのスキーマ *Queue*, *Queue'* を包含している．これは次のスキーマと同じ．だが，記述量が違う．また上のように書く方がスキーマ *Queue* との関連が明白．記述の経済と知識の構造化の一例．

EnQueue

$queue : \mathrm{seq}\ ITEM$
$queue' : \mathrm{seq}\ ITEM$
$i? : ITEM$

$queue \leq maxsize$
$queue' \leq maxsize$
$queue' = queue^\frown < i? >$

ここでも注意が必要．いくつかのスキーマを包含するとき，それらの成分に共通の識別子がなければ問題はないが，それがある時は，その識別子の型は一致していなければならない．

3.5.3 Δ, Ξスキーマ生成規約

再利用する頻度の高いスキーマ構成子を紹介しよう．

Δ規約　Sがスキーマのとき，ΔSはSとS'の両者を包含するスキーマを表す．

ΔS

S
S'

上の *EnQueue* のように，状態変更を引き起こす操作(事象)スキーマは，必ず状態スキーマと状態スキーマをプライム'で修飾したスキーマを包含する．事前値と事後値の両者を含むからである．Δ構成子を使うと *EnQueue* は次のように書ける．

EnQueue

$\Delta Queue$
$i? : ITEM$

$queue' = queue^\frown < i? >$

というわけで，$\Delta Queue$ を見れば，一目で状態 *Queue* を更新することが見て取れる．

ここで注意して欲しいのは，記述の経済だけではない．決まりきった事柄はすべて $\Delta Queue$ を包含することで済ませられることは，言い替えると，*EnQueue* には *EnQueue* に固有の事柄だけを明示すれば済む，ということでもある．だから，*EnQueue* を記述する時には，*EnQueue* に固有の事柄だけ考えればよい．記述の経済は同時に思考の経済でもある．かくして，構造化記法が巨大ソフトウェアの仕様の読み書きを可能とする．

Ξ規約 ΞS は ΔS に加えて，その叙述部に，状態変数の事前値と事後値が等しいとの制約条件を含むスキーマを表す．

ΞS

S
S'

$\theta S = \theta S'$

詳しい説明は省略するが(追記参照)，$\theta S = \theta S'$ で S と S' の全成分が等しいことを言い表している．

この使い途だが，状態は更新せず，単に状態を検索してレポートを出力する類のコマンドのスキーマを書くとき，これを包含するとかなり記述の経済になる．

3.5.4 スキーマ結合

スキーマを論理結合子を使って結合する．いくつかのスキーマを結合するとき，それらの成分に共通の識別子がなければ問題はないが，それがある時は，その識別子の型は一致していなければならない．

否定¬ 特性部は S の特性部と同じ．叙述部は S の叙述部の否定．

$\neg S$

Sの特性部

$\neg$ (Sの叙述部)

その他の二項論理結合子 $S\ op\ T$ 二項論理結合子∧, ∨, ⇒, ⇔による結合は次の通り．

S op T

*S*の特性部
*T*の特性部

(*S*の叙述部) *op* (*T*の叙述部)

3.5.5 構造化記法の応用例

これまでの，待ち行列への項目追加操作の仕様では，正常な場合しか考えていない．ここでは異常処理を含む仕様を考えよう．

1) 待ち行列が満杯の時は，異常報告「待ち行列満杯」を出力し，
2) 項目追加ができた時は，正常報告「正常終了」を出力する，

ことにしよう．それぞれの報告出力処理のスキーマは次のようになる．

[*REPORT*]

AbnormalReport

$\Xi Queue$
$rep! : REPORT$

$\#queue = maxsize$
$rep! =$「待ち行列満杯」

SuccessReport

$rep! : REPORT$

$rep! =$「正常終了」

正常処理，異常処理の双方を含んだ仕様 *RealEnQueue* は次のようになる．

$$RealEnQueue \triangleq (EnQueue \wedge SuccessReport) \vee AbnormalReport$$

正常処理のスキーマの中に異常処理の仕様を追加していくと，たちまち大きくなってその構造が掴み難くなる．その点，Zの構造化記法を使うと，上に示したように個々の処理は小さなスキーマで表現し，最終的な仕様はそれらを部品として組み立てればよい．

その利点は，まず第一に仕様が書きやすいこと．第二は，上のように構造が明示されるので仕様が読みやすいことである．それだけではない．コードを導出する段階でも，その構造が活きる．また，スキーマ部品を差し替えれば，別の仕様を組み立てることもできる．部品化・再利用とはこのようなことを言うのではないだろうか．

4. 図書館問題の記述例 — 批判と改善

図書館問題の仕様について検討する．それを通して，Z仕様記述について一つのスタイルを提示したい．

Z仕様記述についてはすでに多くの教科書が公にされているが，そこに記載されている記述例に，いつも不満を抱き続けてきた．不満を感ずる理由は，どうやら彼我のモデル観の違いにありそうだ．以下，不満を感ずる仕様の具体例について，そのどういうところが不満なのかを具体的に指摘し，われわれなりの改善案を示したい．改善

案は，Z 仕様言語を用いて仕様を書く時の一つの記述スタイルの提案でもある．批判の対象として文献[5] から図書館問題を取り上げた．

図書館問題について議論しながら，Z 言語の使い勝手を味わうこと，これが本章のもう一つの目的である．

4.1 広義の対象主導接近法

モデル記述に対するわれわれの考え方は，広い意味での対象主導といってよさそうだ．その要点は，対象世界を抽象データ型によって再構成しようというものである．つまり，対象世界にある「もの（種類）」を「抽象データ型」で表現し，対象世界にある「もの（個体）」を「抽象データ型の具体例」(instance) として表現する．そして，それら具体例を材料にして対象世界を再構築しようというのである．

それにしても，対象主導は大いに繁盛しているようだ．対象主導をテーマとする研究論文の数の多いことは，それこそ汗牛充棟ただならぬところではない．他方，対象主導に寄せる産業界の期待の過熱ぶりもただごとではない．これさえあれば，ソフトウェア業界の悩みは万事解決すると思いこんでいる傾きすらある．度はずれた期待は論外だが，ソフトウェア開発に対象主導接近法が極めて有効であることには十分理由がある．

仕様言語 Z についても，対象主導の考え方に則ったモデルを，素直に，自然に，活きいきと表現できるよう，Z を拡張する試みが盛んである．そのような試みの中では，今のところ Object-Z [6] [7] が最も進んでいるようだ．今後に期待したい．しかし，今の Z の記述レポートの範囲でも工夫すれば何とか対象主導風の書き方ができる．そのような記述スタイルを示したい．対象主導という観点からすれば，不満は残るが，Z の教科書や，その外われわれが管見した限りでの記述スタイルより，こちらを採りたい．

4.2 第一の批判

さて，文献[5] の図書館問題の仕様は，いきなり所与の集合と図書館の抽象状態を仕様化するところから始まる．

$$[Copy, Book, Reader]$$

$$maxloans : \mathbb{N}$$

Library

$$stock : Copy \rightarrowtail\!\!\!\!\!\!+\!\!\!\!\!\!\rightarrow Book$$
$$issued : Copy \rightarrowtail\!\!\!\!\!\!+\!\!\!\!\!\!\rightarrow Reader$$
$$shelved : \mathbb{F}\, Copy$$
$$readers : \mathbb{F}\, Reader$$

$$shelved \cup \mathrm{dom}\, issued = \mathrm{dom}\, stock$$
$$shelved \cap \mathrm{dom}\, issued = \emptyset$$
$$\mathrm{ran}\, issued \subseteq readers$$
$$\forall r : readers \bullet \#(issued \rhd r) \leq maxloans$$

そして，この大きな抽象状態をもとに，ただちに貸し出し *Issue* の仕様化に取り掛かっている．不満の第一はこの点である．

4.3 対象主導接近法モデル化の考え方

図書館のような大きな対象を一挙に仕様化するのは無理というものである．一度に大きなことは考えられないからである．細かく分けて考えたい．大きな課題なら小さな課題に，複雑な課題なら単純な課題に分割したい．

もちろん，これは対象主導接近法に特有の考え方ではない．課題を分割し，各個撃破するのはいわば常識だろう．問題はどういう単位に分割するのか，その分割の単位はどのように特徴づけられるのか，である．

まず，対象世界を，それを構成する「もの」に解体する．「もの」は対象世界の中でさまざまな行動を起こす．あるいは，さまざまな出来事に参画する．この行動者たる「もの」を「抽象データ型」でモデル化する．そして，その具体例（instance）を構成要員として対象世界の再構成へと進むのが「われわれの対象主導接近法」である．

図書館問題でいえば，いきなり図書館を考えるのではなく，その前に，この対象世界を構成している最も基本的な「もの」を選び出し，それらを「抽象データ型」でモデル化する．その後で図書館世界の構築へと進む．

では，図書館世界を構成する基本的な「もの」は何か．その選定の手がかりを与えてくれるのは文献[5] 6.2節の冒頭にある図書館システムの日常語による記述である．それを見ると

・図書館は本の在庫と，登録された利用者からなる．

とある．とすれば，この対象世界に登場する「もの」は当然

・本
・利用者

である．

この場合は単純なので問題にならないが，一般に対象世界を構成する基本的な「もの」はどう選び出せばよいだろうか．対象世界を記述した日常語の文書から「もの」を選び出すとよい．そうすれば「わかりやすく，仕様変更に善く耐える」モデルが構築できる．これがJSD[8]の基本戦略のようである．日常言語の安定性を活用しようということらしい．まことにもっともなので，それに従った．

では，本，利用者はどのようにモデル化できるだろうか．文献[5]の第6章の図書館仕様の記述例にはまだ不満はあるが，それを批判する前に，この二つの「もの」「本」と「利用者」をモデル化しておこう．対象主導接近法でいう「もの」としてモデル化するためには，それぞれの状態と行動を把握しなければならない．

・「本」のモデル＝「本」の状態＋「本」の行動
・「利用者」のモデル＝「利用者」＋「利用者」の行動

「利用者」には「行動」という言い方はごく自然だが，「本」の「行動」となると奇異な感じがする．「行動」というと，何らかの出来事あるいは事象を「主体的に起こす」といった意味合いを伴うせいかも知れない．要は，状態推移を伴う出来事のことである．出来事に関与することで，その「もの」の状態が変化することが肝要なので，出来事に主体として参画するか，あるいは客体として参画するかはまったく区別しない．ただ何らかの**事象に参画する**という点だけに着目する．それで，以下では，「本はこれこれの行動を起こす」とか「本はこれこれの行動を起こされる」といった表現が無理

な時は「本はこれこれの事象に参画する」ということにする．

では図書館の「本」や「利用者」はどのような事象に参画するのか．それを選び出す手がかりを与えてくれるのは，再び文献[5] 6.2節の冒頭にある図書館システムの日常語による記述である．選び出した結果は次のように整理するとよい．

本が参画する事象

事象　登録
説明　図書館の所蔵本として新規登録する．
関連データ　Copyid，タイトル，著者名，出版者名，発行日，登録日

事象　貸出
説明　利用者に本を貸し出す．
関連データ　利用者id，貸出日付

事象　返却
説明　利用者が借り出した本を返す．
関連データ　利用者id，返却日付

事象　廃棄
説明　ぼろぼろになった本は捨てる．
関連データ　Copyid

利用者が参画する事象

事象　登録
説明　図書館の利用者として新規登録する．
関連データ　利用者id，名前，住所，登録日

事象　貸出
説明　本を貸し出してもらう．
関連データ　Copyid，貸出日付

事象　返却
説明　借りた本を返す．
関連データ　Copyid，返却日付

事象　登録削除
説明　利用者登録を取り消す．
関連データ　利用者id

それぞれの事象の説明を読むと，貸出と返却の二つの事象には，その生起に「本」と「利用者」が共に参画することがわかる．

文献[5]の6.2節冒頭の記述には，このほか，「利用者」に借りている本を問い合わせたり，「本」の貸出先を問い合わせたり，等の事象もあるが，このような問い合わせは，「もの」状態が変化するわけではないから，当面のモデル化からは外す．課題の整理の仕方，あるいは設計法についてはMickel Jackson [8]が優れている．以上の書き方は文献[8]に従った．

ここでとくに注意して欲しいのは，このように課題を整理すると，本が参画する事象を選び出すだけでなく，本の状態を構成するデータも同時に選び出されている点である．

4.4 モデル記述

課題をこのように整理しておけば，そのモデルは直ちに記述できる．所与の集合は関連データをみながら列挙すればよい．

[*COPYID, TITLE, NAME, DATE, ADDRESS, USERID*]

4.4.1 本のモデル化

まず「本」の状態だが，二つに分けて考えよう．登録の時点で確定し，その後の事象の生起によって変動しない部分と，変動する部分の二つである．まず変らない部分から．これは文献[5]でいえば，6.3節の記述例の *Book* に相当する．

静的状態

BookStatic

title : *TITLE*
author : *NAME*
...

動的状態　事象の生起によって変動する状態は実にさまざまなモデル化ができる．ここでは「本」について生起した最後の事象の記録をもって，その動的状態としよう．

先の課題整理から明らかなように，事象には，本の登録と廃棄，利用者の登録の削除，それに貸出と返却がある．貸出と返却は本と利用者に共通している．

それぞれの事象に名前を付けよう．

$$EVENTID ::= bookreg \mid bookdel \mid userreg \mid userdel \mid issue \mid entrance$$

事象の記録内容は先の課題整理の関連データの項を見れば明かである．

登録記録

BookRegRec

regdate : *DATE*

貸出記録

BookIssRec

issuser : *USERID*
issdate : *DATE*

返却記録

BookEntRec

entuser : *USERID*
entdate : *DATE*

事象の記録

$$BookDynamic == BookRegRec \cup BookIssRec \cup BookEntRec$$

本の廃棄については，当の本を抹殺することとし，記録は採らないことにする．本の状態は静的状態と動的状態の両者を含む．

本の状態

BookState

$$
\begin{array}{l}
sta : BookStatic \\
evid : EVENTID \\
dyn : BookDynamic \\
\hline
evid \in \{bookreg, issue, entrance\} \\
evid = bookreg \Leftrightarrow dyn \in BookRegRec \\
evid = issue \Leftrightarrow dyn \in BookIssRec \\
evid = entrance \Leftrightarrow dyn \in BookEntRec
\end{array}
$$

本が参画する事象　今度は本が関与する事象のモデル化である．最初は本の属性を獲得するところから始まる．属性値の初期値登録だが，ここでは事前値がなく，事後値だけの事象スキーマで表現した．

属性登録

BookAttrib

$$
\begin{array}{l}
BookState' \\
title? : TITLE \\
author? : NAME \\
\ldots \\
date? : DATE \\
\hline
sta'.title = title? \\
sta'.author = author? \\
\ldots \\
evid' = bookreg \\
dyn'.regdate = date?
\end{array}
$$

貸　出

BookIssue

$$
\begin{array}{l}
\Delta BookState \\
uid? : USERID \\
date? : DATE \\
\hline
evid = bookreg \lor evid = entrance \\
evid' = issue \\
dyn'.issuser = uid? \land dyn'.issdate = date? \\
sta' = sta
\end{array}
$$

返　却

BookEntrance

$$
\begin{array}{l}
\Delta BookState \\
date? : DATE \\
\hline
evid = issue \\
evid' = entrance \\
dyn'.entuser = dyn.entuser \land dyn'.entdate = date? \\
sta' = sta
\end{array}
$$

以上で本のモデル化は終わる．次はたくさんの本のモデル化である．

4.4.2 本の集団のモデル

言うまでもなく，図書館にはたくさんの本がある．それらをどう識別するのか．たくさんあると言っても，初めからたくさんあったわけではない．初めは少なかったかも知れない．否，初めは一冊も無かったのかも知れない．それが徐々に増えてきたのだろう．一度，図書館に登録された本は永久に存続するわけではない．ぼろぼろになれば廃棄される．このような本の集団はどうモデル化すればよいか．

本の集団の状態　個々の本は識別子によって区別する．

BooksState

$$books : COPYID \pfun BookState$$

BooksState$_{Init}$

BooksState

$$books = \varnothing$$

本の集団の事象　本の集団の事象は以下の通り．

登　録

BooksReg

$$\Delta BooksState$$
$$c? : COPYID$$
$$t? : TITLE$$
$$a? : NAME$$
$$\ldots$$
$$d? : DATE$$

$$c? \notin \mathrm{dom}\, books$$
$$(\mathbf{let}\ bk = (\mu\, BookAttrib \mid title? = t? \land author? = a? \land \ldots \land date? = d? \bullet \theta BookState') \bullet books' = books \cup \{c? \mapsto bk\})$$

「θ *BookState*」については追記参照．

貸　出

BooksIssue

$$\Delta BooksState$$
$$c? : COPYID$$
$$u? : USERID$$
$$d? : DATE$$

$$c? \in \mathrm{dom}\, books$$
$$(\mathbf{let}\ bk = (\mu\, BookIssue \mid \theta BookState = books(c?) \land uid? = u? \land date? = d? \bullet \theta BookState') \bullet books' = books \oplus \{c? \mapsto bk\})$$

返　却

BooksEntrance

$\Delta BooksState$
$c? : COPYID$
$d? : DATE$

$c? \in \mathrm{dom}\, books$
$(\mathbf{let}\ bk = (\mu\, BookEntrance \mid \theta BookState = books(c?) \land$
$\quad date? = d?$
$\quad \bullet\ \theta BookState')$
$\quad \bullet\ books' = books \oplus \{c? \mapsto bk\})$

廃　棄

BooksDelete

$\Delta BooksState$
$c? : COPYID$

$c? \in \mathrm{dom}\, books$
$(books(c?).dyn).evid \in \{bookreg, entrance\}$
$books' = \{c?\} \lhd books$

4.4.3　利用者のモデル

次は利用者のモデル化である．

利用者の状態　まずは利用者の状態から．

UserStatic

$name : NAME$
$addr : ADDRESS$
$\ldots$
$regdate : DATE$

UserState

$sta : UserStatic$
$bookl : COPYID \twoheadrightarrow DATE$

bookl はその利用者が借り出し中の本の *Copyid* と借り出した日付のリストである．この中でこれだけが唯一の動的状態である．

利用者の事象　利用者が関与する事象は以下の通りである．

属性登録

UserReg

$UserState'$
$name? : NAME$
$addr? : ADDRESS$
$date? : DATE$

$sta'.name = name?$
$sta'.addr = addr?$
$\ldots$
$sta'.regdate = date?$
$bookl' = \emptyset$

貸　出

UserIssue

$\Delta UserState$
$cid? : COPYID$
$date? : DATE$

$cid? \notin \mathrm{dom}\ bookl$
$bookl' = bookl \cup \{cid? \mapsto date?\}$
$sta' = sta$

返　却

UserEntrance

$\Delta UserState$
$cid? : COPYID$

$cid? \in \mathrm{dom}\ bookl$
$bookl' = \{cid?\} \lhd bookl$
$sta' = sta$

4.4.4　利用者集団のモデル

本と同様，利用者の集団の振る舞いをモデル化する．

利用者の集団の状態　大勢の利用者を識別子によって区別する．

UsersState

$users : USERID \pfun UserState$

$UsersState_{Init}$

$UsersState$

$users = \emptyset$

利用者集団の事象　事象スキーマは以下の通り．

登　録

UsersReg

$\Delta UsersState$
$u? : USERID$
$n? : NAME$
$a? : ADDRESS$
$\ldots$
$d? : DATE$

$u? \notin \mathrm{dom}\ users$
$(\mathbf{let}\ ur = (\mu\ UserReg \mid$
$\quad name? = n? \land addr? = a? \land \ldots \land regdate = d?$
$\quad \bullet\ \theta UserState')$
$\quad \bullet\ users' = users \cup \{u? \mapsto ur\})$

貸 出

UsersIssue

$\Delta UsersState$
$u? : USERID$
$c? : COPYID$
$d? : DATE$

$u? \in \mathrm{dom}\ users \land$
$(\mathbf{let}\ ur = (\mu\ UserIssue \mid \theta UserState = users(u?) \land$
$\quad cid? = c? \land date? = d?$
$\quad \bullet\ \theta UserState')$
$\quad \bullet\ users' = users \oplus \{u? \mapsto ur\})$

返 却

UsersEntrance

$\Delta UsersState$
$u? : USERID$
$c? : COPYID$

$u? \in \mathrm{dom}\ users$
$(\mathbf{let}\ ur = (\mu\ UserEntrance \mid \theta UserState = users(u?) \land$
$\quad cid? = c?$
$\quad \bullet\ \theta UserState')$
$\quad \bullet\ users' = users \oplus \{u? \mapsto ur\})$

4.5 批判その2

文献[6]の図書館仕様に対する第二の不満は「もの」の状態についてである．文献[6]の仕様では，本の静的な状態は *Book* を見ればわかるようになっているが，本の動的な状態は *issued* や *shelved* に分散されている．気に入らないのはこの点である．われわれの仕様では，本に関する情報はすべて本の状態を見ればわかるようになっている．「本のことは本に聞け」の精神である．

4.6 モデル記述その2

次は本と利用者を材料にして図書館世界のモデルを構成する．

4.6.1 図書館の状態

図書館の状態は本の集団と利用者の集団である．

LibState

$stock : BooksState$
$members : UsersState$

$\forall c : COPYID; u : USERID; d : DATE \bullet$
$\quad c \in \mathrm{dom}\ stock.books \land$
$\quad stock.books(c).evid = issue \land$
$\quad u = stock.books(c).dyn.issuser \land$
$\quad d = stock.books(c).dyn.issdate$
$\Leftrightarrow$
$\quad u \in \mathrm{dom}\ members.users \land$
$\quad c \in \mathrm{dom}\ members.users(u).bookl \land$
$\quad d = members.users(u).bookl(c)$

$$
\begin{array}{|l}
\hline
LibState_{Init} \\
\hline
LibState \\
\hline
\exists\, BooksState_{Init} \bullet \theta BooksState = stock \\
\exists\, UsersState_{Init} \bullet \theta UsersState = members \\
\hline
\end{array}
$$

Libstate の制約条件の意味は，ある本のコピーの状態が貸出中であれば，その貸出先 *issuser* の利用者の状態の方でも，その借り出し本のリストに当のコピーが含まれていること，およびその逆，つまり，ある利用者の状態の借り出し本のリストに含まれている本は，その本の状態の方でも，当の利用者に貸出中になっており，*stock* と *members* との間に矛盾がない，と言うことである．

stock は文献[5] 6.3 節の *Library* の *stock* とよく似ている．違うのは *Book* と *BookState* の内容である．*Book* には静的な情報しかないが，*BookState* には静的な情報に加えて動的な情報も入っている．この違いが，*Library* には必要だった *issued*，*shelved* を，*LibState* では要らなくしている理由である．

では，図書館で起こる事象のモデル化へ進もう．

4.6.2 図書館の事象

本の登録

$$
\begin{array}{|l}
\hline
LibBookReg \\
\hline
\Delta LibState \\
c1? : COPYID \\
t1? : TITLE \\
a1? : NAME \\
\ldots \\
d1? : DATE \\
\hline
stock' = (\mu\, BooksReg \mid \theta BooksState = stock \wedge \\
\quad c? = c1? \wedge t? = t1? \wedge a? = a1? \wedge \ldots \wedge d? = d1? \\
\quad \bullet\, \theta BooksState') \\
members' = member \\
\hline
\end{array}
$$

利用者登録

$$
\begin{array}{|l}
\hline
LibUserReg \\
\hline
\Delta LibState \\
u1? : USERID \\
n1? : NAME \\
a1? : ADDRESS \\
\ldots \\
d1? : DATE \\
\hline
members' = (\mu\, UsersReg \mid \theta UsersState = members \wedge \\
\quad u? = u1? \wedge n? = n1? \wedge a? = a1? \wedge d? = d1? \\
\quad \bullet\, \theta UsersState') \\
stock' = stock \\
\hline
\end{array}
$$

貸　出

LibIssue

$\Delta LibState$
$u1? : USERID$
$c1? : COPYID$
$d1? : DATE$

$members' = (\mu\ UsersIssue \mid \theta UsersState = members \land$
　　$u? = u1? \land c? = c1? \land d? = d1?$
　　$\bullet\ \theta UsersState')$
$stock' = (\mu\ BooksIssue \mid \theta BooksState = stock \land$
　　$c? = c1? \land u? = u1? \land d? = d1?$
　　$\bullet\ \theta BooksState')$

返　却

LibEntrance

$\Delta LibState$
$c1? : COPYID$
$d1? : DATE$

$stock' = (\mu\ BooksEntrance \mid \theta BooksState = stock \land$
　　$c? = c1? \land d? = d1?$
　　$\bullet\ \theta BooksState')$
$(\textbf{let}\ u1 = stock.books(c1?).issuser \bullet$
　　$members' = (\mu\ UsersEntrance \mid \theta UsersState = members \land$
　　$u? = u1 \land c? = c1?$
　　$\bullet\ \theta UserState'))$

本の廃棄

LibBookDelete

$\Delta LibState$
$c1? : COPYID$

$stock' = (\mu\ BooksDel \mid \theta BooksState = stock \land$
　　$c? = c1?$
　　$\bullet\ \theta BooksState)$
$members' = members$

追　　記

0.1　スキーマとデータ集合

Z 仕様では，勝手なスキーマ S を

$$a : S$$

のように，データ集合として使うことができる．そしてその x_1 成分，x_2 成分，... を $a.x_1$，$.x_2$，... 等と表す．

それでは，スキーマ S はどのようなデータ集合を表すかだが，第一近似としては，x_1，x_2，... をフィールド名とするレコード型を考えればよい．各フィールド x_i の値は勿論 T_i でなければならない．S は，そのようなレコードのうち，そのフィールドの値

が S の叙述部に記された制約条件を満たすレコードの集合と解釈すればよい．

文献[2]では，レコードという概念は仮定せず，代わりに**束縛**(*binding*)という対象を導入し，それを使って説明している．

0.1.1 束縛とスキーマ型

$id_1, \ldots, id_n$ を相異なる識別子，$ob_1, \ldots, ob_n$ をそれぞれ型 t_1，$ldots$，t_n の勝手な対象とする．このとき，**束縛**と呼ばれるある対象

$$z = < id_1 \Rightarrow ob_1, \ldots, id_n \Rightarrow ob_n >$$

が存在し，その成分については $z.id_i = ob_i (1 \leq i \leq n)$ が成り立つと考える．さらに，この束縛 z は**スキーマ型**

$$< id_1 : t_1 ; \ldots ; id_n : t_n >$$

を持つ，と考える．

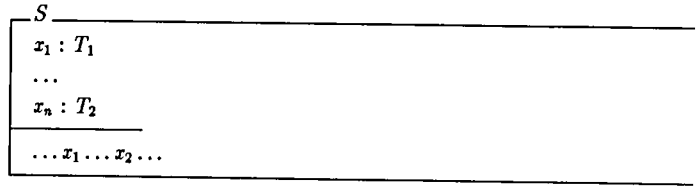
要するに，束縛とは，識別子がどのような値を持つのかを指定するものである．

同じスキーマ型を持つ束縛 z，w が等しくなるのは，$z.id_i = w.4id_i (1 \leq i \leq n)$ のとき，しかもそのときに限る．

二つのスキーマ型は，その一方のスキーマ型の成分を適当に並べ替えると，もう一つのスキーマ型になるようなら，両者は同一スキーマ型と見なす．同じことは，束縛についても言える．二つの束縛は，その成分の並び順が違うだけで，両者に現れる成分が同じなら，同一の束縛と見なす．

0.1.2 データ集合としてのスキーマ

スキーマ S



は

$$a : S$$

のように，データ集合として使うことができる．

データ集合として使う場合，スキーマ S は束縛の集合と解釈する．ただし，S の要素となる束縛は，S の特性部の規定するスキーマ型 $< x_1 : T_1 ; \ldots ; x_n : T_n >$ の束縛でなければならない．

$$S \subseteq < x_1 : T_1 ; \ldots ; x_n : T_n >$$

そのような束縛のうち，その成分の値が S の叙述部の制約条件を満たす束縛の集合をもって，データ集合 S の解釈とする．

0.1.3 束縛生成子 θ

勝手なスキーマ型 S やそれをプライム修飾した S' から，束縛を作り出す生成子．

書き方：　θS　あるいは　$\theta S'$

型の規約：両項 θS，$\theta S'$ とも束縛を表す．S の成分を $x_1, \ldots, x_n$，その型を $T_1, \ldots, T_n$ とする．このとき，両項の表す束縛は同一のスキーマ型 $< x_1 :$

$T_1; \ldots; x_n : T_n >$ を持つ．

ここで，$\theta S'$ の表す束縛の型の成分の識別子にはプライムが付かないことに注意．これは，束縛をレコードと考えれば，θS，$\theta S'$ が同じレコード型を持つことに相当する．

説明：θS，$\theta S'$ の表す束縛は，型は同じだが，値は違う．両項を評価する環境を束縛 u とする．このとき θS の値は束縛

$$< x_1 \Rightarrow u.\, x_1; \ldots; x_n \Rightarrow u.\, x_n >$$

となり，$\theta S'$ の値は束縛

$$< x_1 \Rightarrow u.\, x_1'; \ldots; n_n \Rightarrow u.\, x_n' >$$

となる

補足事項：　1．データ集合 S は次のように表せる．

$$S = \{S \bullet \theta S\}$$

2．θS，$\theta S'$ について以下が成立．

$$(\theta S').\, x_i = x_i'$$

$$\theta S = \theta S' \Longleftrightarrow x_1' = x_1 \wedge \ldots \wedge x_n' = x_n$$

参考文献

［1］ J. M. Spivey, *UNDERSTANDING Z*, Cambridge University Press, 1988.

［2］ J. M. Spivey, *The Z Notation, A reference Manual SECOND EDITION*, Prentice-Hall, 1992.

［3］ D. C. Ince *An Introduction to Discrete Mathematics and Formal System Specification*, Clarendon Press, Oxford, 1988.

［4］ Antoni Diller **Z** *An Introduction to Formal Methods*, John Wiley & Sons, 1990.

［5］ Ben Potter, Jane Sinclair and David Till, *An Introduction to Formal Specification and Z* Prentice Hall, 1991.

［6］ R. Duke, P. King, G. Rose, and G. Smith. *The Object-Z Specification Language : Version 1*. Technical Report 91-1, Software Verification Research Centre, Dept. of Computer Science, University of Queensland, Australia, May 1991.

［7］ D. Duke and R. Duke, *Towards a semantics for Object-Z*, In D. Bjorner, C. A. R. Hoare, and H. Langmaack, editors, *VDM '90 : VDM and Z !*, volume 428 of Lect. Notes in Comput. Sci., page 242～262. Springer-Verlag, 1990.

［8］ M. Jackson, *Jackson System Development*, Prentice Hall, 1980.

執筆者紹介　染　谷　　誠（Makoto Someya）

昭和17年生．昭和44年 立教大学大学院理学研究科 修士課程退学，同年(株)日本ユニバック総合研究所入社．事務処理システムの開発，形状処理システムの開発，仕様記述法やシステム開発方法の調査研究に従事．現在 日本ユニシス(株)システム技術本部 生産技術一部 技術開発課所属．ソフトウェア科学会会員．情報処理学会，情報処理企画調査会，FDT-SWG委員会，委員．

LINC IIテスト・デバッグ支援ツール LINC-TE

1. はじめに

1992年5月に発表されたASDF (Advanced Solution Development Framework)において,迅速なソリューション構築のための統合CASEツールの中の,下流CASEツールとして位置づけられるものにLINCがある.LINCによるトランザクション処理システムの構築は,「システム開発の多くの過程を自動化し,進化型アプローチを可能にする」という大きな特徴を持っているが,大規模なシステムになると,システムの生成に多くの時間を要し,そのためプロトタイピングのサイクルに時間がかかるという問題点があった.

LINC-TEはこの問題を解消し,さらにその他の点でもLINCによるシステム構築をより迅速に実現するためのテスト・デバッグツールとして提供されるものである.

ここではLINC-TEの特徴,機能,利用効果,およびLINCを使ったシステム開発がLINC-TEの使用によってどのように変わっていくかについて紹介する.

2. 特　徴

LINC-TEは,すでにLINCを使用しているユーザが容易に使用できるようになっている.

1) 生成されたLINCシステムの拡張部分として使用できる.LINC-TEの実行は,通常のLINCシステムにLINC-TE用の更新プログラムが追加された形となる.このためLINCを知っている開発者は,通常の更新プログラムと同じ感覚で使用することができる.

2) デバッグモードで使う指令がLINCのLDL (Linc Description Language) と良く似た形になっている.

3) 特別なツールやハードウェアを必要としない.LINC-TEは現在の環境にLINC-TEを導入するだけで使用できる.

4) 既存のLINC開発環境を変えずに使用できる.すでに導入し使用しているLINCの開発環境になんら変更を加えることなく,LINC-TEを使用することができる.

5) 稼働中の業務システムを変更することなく,修正したロジック等の実行ができる.

3. 機能概要

LINC-TEの機能には大きく分けて次の三つがある.

- インタプリティブ機能
- オンラインデバッグ機能
- 統計機能

これらについて概要を紹介する.

1) インタプリティブ機能……LINC対話型データベース内の定義を直接読み取って,システム/レポートを実行する.そのため,システムを変更したとき,再生成を行わずに即座に変更したシステムを動かすことができる.システムの場合は最低一回は生成されていることが必要だが,レポートの場合は一度も生成されていなくても実行が可能である.

システムの実行は〈システム名〉/UPDATE 20,レポートの実行は〈システム名〉/LINCLITE/REPORTSというLINC-TE用のプログラムがそれぞれ行い,データベースのアクセスにはDMINTERPRETERを使用する (図1).

2) オンラインデバッグ機能……ISPECまたはレポートのロジック実行時に,対話形式で,ロジック,データ項目の値,データベースを更新する値のトレースを行う.また,ブレイクポイントを設定し,任意にロジックの実行を止めデータの参照・変更を行うことができる.これにより,データベース読み出しの内容の確認や境界値のテストができる.

トレースの型として次のようなものがある.

- 全面画トレースモード
  1ページ分のロジックを表示し,現在実行中のロジック行を順次強調表示する (図2).

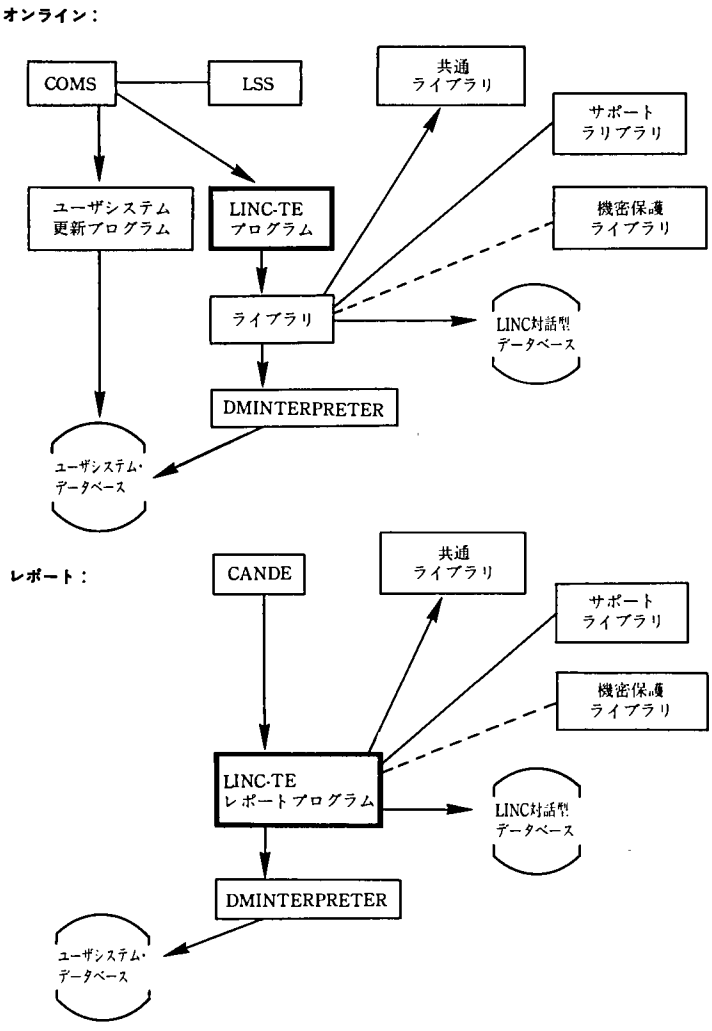


図 1 LINC-TE 実行構造

```
CONTINUE
ISPEC: URI          エラー コード       GLB.ERROR:      GLB.STATUS:
000100 SETUP.DATA; WORK1 (0) ED + LE 11
000200 SETUP.DATA; WORK2 (0) ED + LE 11
000300 SETUP.DATA; WORK3 (0) ED + LE 11
000400 DO.WHEN; KAKU NOT = (OK)
000500          MOVE; TOK.TOKMEI        TOKMEI
000600          MOVE; SYO.SYOMEI        SYOMEI
000700          MULTIPLY;  SURYO        TANKA   GIVING   KINGAKU
000800          RECALL;  (URI)
000900 END.EXIT;
001000:::::::::::::::::::::
001100:INS;     GL-MV
001200:::::::::::::::::::::
001300 DO.WHEN; KAKU = (OK) AND
001400 DO.WHEN; SURYO = GLB.ZEROS
001500          MESSAGE; ERROR ( 数量入力のこと )
001600 END.EXIT;
001700 MULTIPLY; SURYO     TANKA           GIVING  KINGAKU
001800 ADD;      KINGAKU   TOK.TOKURI      GIVING  WORK1
001900 ADD;      KINGAKU   TOK.TOKZAN      GIVING  WORK2
002000 SUBTRACT; SURYO     SYO.SYOZAN      GIVING  WORK3
002100 FLAG;     WORK1     TOK.TOKURI
002200 FLAG;     WORK2     TOK.TOKZAN
                                                          URD
 統行中
```

図 2 全画面トレースモード

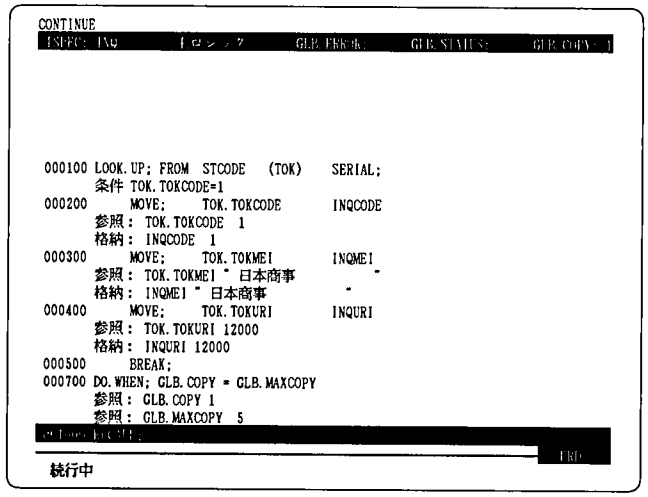


図 3　データ値トレースモード

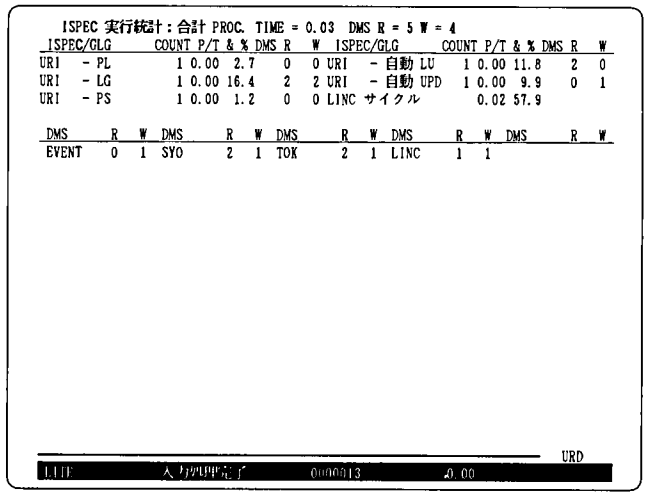


図 4　統計出力

- 指令行トレースモード
  実行中のロジックを画面の下からスクロールアップしながら強調表示する．
- データ値トレースモード
  実行中のロジックと，参照，移送，比較等のロジックを実行した時のデータの値を自動的に表示する（図 3）．
- データベース・アクティビティモード
  データベース・アクセスを発生させるロジックのみを強調表示する．
- ブレイクポイントモード
  ブレイクポイントを設定して，ある条件に合致した時にそこでロジックの実行をいったん止め，そこから全画面デバッグモードに入る．ブレイクポイントが発生するまでは，通常のようにトレースをとらずに実行されるので，時間の節約になる．

いずれの場合も，トレースの結果を画面上にリアルタイムで表示させることもディスク上にファイルとして残すことも可能である．

3)　統計機能……各ロジックの型ごとに次のような統計結果を取ることができる．

- ロジックが実行された回数
- ロジックの実行にかかった CPU 時間と全体の CPU 時間に対する割合

• そのストラクチャでのDMSのREAD, WRITEの回数

レポートの場合は，さらにDELETEの回数も含む．

統計結果は，関連しているデータベース・ストラクチャの確認や，ISPEC・レポートのパフォーマンス，処理効率のチェックに使用でき，システムの最適化がはかれる（図4）．

採取した統計結果は，画面に出力することもディスク上にファイルとして残すこともできる．

4. 利用効果

LINC-TEの利用により，次のような利用効果が期待できる．

1) インタプリティブ機能により，システム生成の回数を大幅に減少させ，開発工数を削減することができる．

2) オンラインデバッグ機能によりテスト・デバッグ作業の効率が向上し，これらに要していた時間の短縮がはかれ，全体的な開発期間の短縮へとつながる．

3) 1)および2)により，LINCの特徴であるプロトタイピングのサイクルを短くすることができる．

4) 本番システムのデータを用いたテストが可能なため，システムの生成を行わずにデータに原因のある問題を診断することができ，システム保守の効率を改善することができる．

5. LINC-TEを使用したLINCシステムの開発方法

今までのLINCシステムの開発では，小さい変更でもシステム全体の生成が必要になり，その間プログラム開発ができなかった．そのため生成をかける前には，必要なテスト項目をすべて盛り込んでおくことが必要であった．しかしLINC-TEを使用した場合，生成の必要がなくなるため，各テスト項目の結果を随時確認することができるようになる（図5）．

これにより，LINC-TEでは今までよりも短いサイクルのプロトタイピングが可能になり，それに合わせて，プロトタイピングの目標設定もより細かいステップに分けて立てることが可能となる．

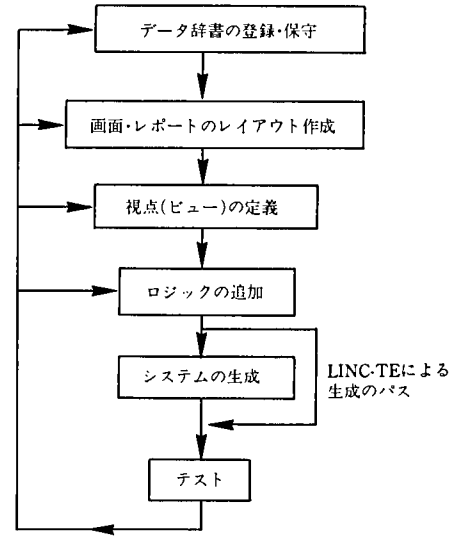


図5 LINCによる開発手順

6. おわりに

以上紹介してきたように，LINC-TEは今までのLINCシステム開発をさらに強力にする支援ツールである．とくに大規模なシステムを開発しており，開発全体における生成時間の比重の多かったサイトにおいては，LINC-TEから受ける恩恵は多大なものになると考えられる．

LINCシステムの開発におけるテスト・デバッグツールとしてのLINC-TEは，開発方法論・技法としてのLINCシステムズ・アプローチ，上流CASEツールとしてのLINC-DAとともに，より統合化された開発環境の実現を可能とするものである．

8003 グラフィック・サブシステム

1. はじめに

ここ数年，CAD/CAM/CAE/CG分野でもダウンサイジングに対する要求が高まり，メインフレーム上で稼働していたアプリケーション・プログラム（AP）をワークステーションへ移植するユーザが増えている．しかし，現状のワークステーシ

ョンはそのグラフィックス性能が低く，これらユーザのAPを実行させるには能力が不足している．そこで，グラフィックス・アクセラレータ(GA)をワークステーションに搭載し，そのグラフィックス性能を向上させ，グラフィック・ワークステーション(GWS)とする方法が普及しつつある．

一方，ユーザは，市場環境の厳しさへの対応として開発・製造期間の短縮やコストの低減による競争力の向上を図っており，そのためにユーザは彼らのAPの高速化/高性能化を図ると共に，今まで蓄積してきたソフトウェア資産を有効利用できる環境，すなわちオープン・アーキテクチャの提供を強く要求している．

つまり高性能で高機能，かつオープン・アーキテクチャを採用したGMSの実現が要求されているのが現状である．

それに対して，ハードウェアの世界ではマイクロプロセッサの高性能化が進み，ソフトウェアの世界ではウィンドウ・システムを始め各種標準化が進み，ユーザのこうした要求に対応することが可能となっている．

当社は，こうした状況に対応しユーザの要求に応えるため，以下のGWS関連商品を発表し販売を開始することにした．なお8003グラフィック・サブシステムに続き，廉価機および高級機の販売も開始する予定である．

- ハードウェア
  8003グラフィック・サブシステム
- ソフトウェア
  8003 GA BASIC SOFTWARE
  8003 GSM

写真1　筐　体

2. 概　要

1) 8003グラフィック・サブシステム……8003グラフィック・サブシステムは，当社が日本サン・マイクロシステムズ社から，OEM供給を受けている高性能エンジニアリング・ワークステーション「USファミリモデル110シリーズ」にS-Bus経由で接続され，USファミリ上で高性能/高機能グラフィックス環境を実現するためのハードウェアでグラフィックス・アクセラレータボードと19インチ・カラーモニタ等から構成されている．

2) 8003 GA BASIC SOFTWARE……8003グラフィック・サブシステムを使用するための必須ソフトウェアであり業界標準である，X/PEXを採用しクライアント/サーバ環境での3次元グラフィックス環境を実現するためのソフトウェアである．

3) 8003 GSM……OSF/Motif*を移植したもので8003グラフィック・サブシステム上で高レベルGUIを提供するソフトウェアである．

3. 特　徴

●高速，高品質描画を実現

以下の機能をハードウェア化することにより，高速，高品質描画を実現した．

- ベクトルやポリゴン・エッジの斜線を滑らかに表示し，高品質な画像を得るためのアンチエリアシング機能
- 図形の各面に模様を貼り付けたり，材質感を与えたりして現実感のある画像を得るためのテクスチャ・マッピング機能
- 3次元図形や画像を，立体的に，より現実的に表現するための，デプス・キューイング機能やシェーディング機能

●レンダリング・エンジンに新規開発ASICを採用

レンダリング・エンジン部に，新たに開発したASIC 4種（Setup, Slicer/Dicer Blender, Mixer）を採用し，上記高速，高品質描画を実現している．

● CSSメモリの搭載

システム全体の高速性を実現するため，図形データを保持するCSS（Central Structure Store）メモリ（従来のディスプ

* Motifは Open Software Foundationの登録商標である．

レイ・リストに相当）をグラフィックス・アクセラレータ上に搭載し，ワークステーションと8003グラフィック・サブシステム間のデータ転送量を軽減させている．

●パラレル処理の採用

8003グラフィック・サブシステムは，ジオメトリ・エンジンに8個のi860 XP（40 MHz）を2段4列のパラレル処理で使用し，高速図形演算処理を実現している．

● X/PEXの採用

UNIXネットワーク環境下での業界標準ウィンドウ・システムと，その3次元拡張機能であるPEX（PHIGS Extension to X）を採用し，クライアント/サーバ環境での3次元グラフィック機能の使用を可能にしている．

● CAD/CAM向け機能拡張

CAD/CAMでの操作性を向上させるため，PEXの基本機能に加え次のような機能拡張を行っている．

・日本語入出力機能
・ピックエコー機能
・ローカル・ビューイング，ライティング機能
・各種ユーザ定義機能：カーソル，マーカ，外字，線種，パターン

4. 仕　様

8003グラフィック・サブシステムの仕様を以下に示す．

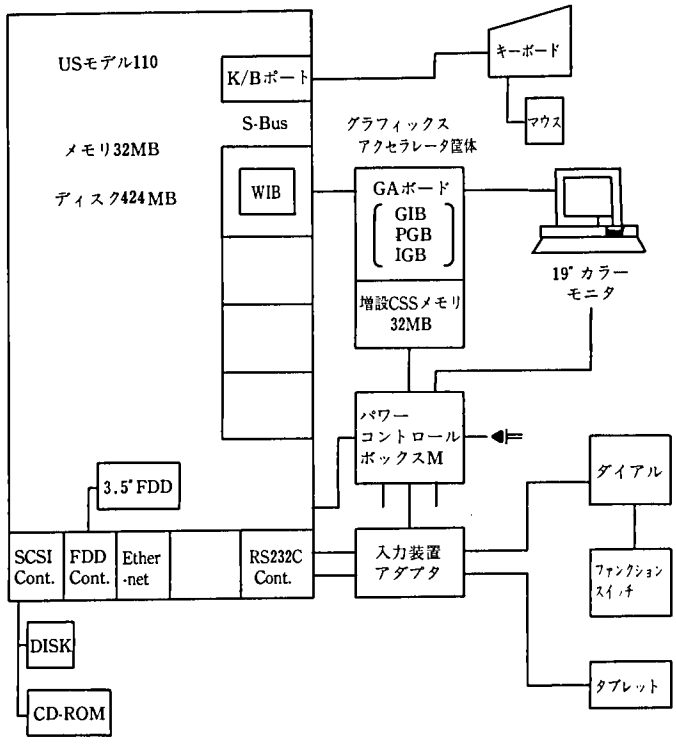
5. 構　成

1) 8003グラフィック・サブシステム

次に，8003グラフィック・サブシステムの構成を示す．

● WIB（Workstation Interface Board）

項　目		8003グラフィック・サブシステム
使用プロセッサ		i860 XP（40 MHz）×9 ASIC（5種類）×24
CSS（図形データ保持用）メモリ		32 MB（64 MBへのオプションあり）
フレーム・バッファ	カラープレーン	24ビット×2（ダブルバッファ）
	Zバッファ	24ビット
	αプレーン	7ビット
	コントロールプレーン	8ビット
モニタ		19インチ・カラーモニタ （リフレッシュレート74.11 Hz）
表示色数		1,670万色
表示ピクセル数		1,280×1,024
表示性能	3Dベクトル性能	125万ベクトル/秒（アンチエリアシング付き）
	3Dポリゴン性能	60万ポリゴン/秒（グーロー・シェーディング）
ワークステーション		USファミリ
外形寸法（単位mm）	幅	240
	奥行	560
	高さ	520
重量（単位Kg）		36.0
電源仕様	電圧（単位V）	90～132
	周波数（単位Hz）	47～63
	電源プラグ	NEMA 5-15 P
最大消費電力（単位W）		760
グラフィックス・ライブラリ		X/PEX（X 11 R 5.0対応）



USファミリモデル110のS-Busに挿入されるインタフェース・ボードでGIBを介してPGBとデータやコマンドの授受を行う．(S-Bus 1スロット使用)

● GIB (Graphic Interface Board)

グラフィックス・アクセラレータの筐体に内蔵されるグラフィックス・アクセラレータ側のインタフェース・ボードでWIBとPGBの間でのデータやコマンドの授受を制御する．

● PGB (PEX Geometry Board)

主としてCSSメモリに格納されている図形データ（ディスプレイリスト）を読み出し，解析し（トラバース処理）座標変換を行うボードで，グラフィックス・アクセラレータの筐体に内蔵される．トラバース処理を行うトラバース・エンジンはi860 XP*（40 mHz）1個で構成され，次段の座標変換やライティング計算（ジオメトリ処理）を行うジオメトリ・エンジンはi860 XP（40 MHz）8個を使用し，4列のパイプラインを構成している．

● IGB (Image and Graphics Board)

PGBから受け取ったデータをもとに，4種20個のASICを使用し，高速にピクセル情報を生成するレンダリング処理を行うボードでグラフィックス・アクセラレータの筐体に内蔵される．

2) ソフトウェアの構成

8003 GA BASIC SOFTWAREと8003 GSMの構成を次に示す．

—API (Application Program Interface)

APIとして，ISO PHIGSおよびPHIGS PLUSに準拠したインタフェースを提供している．

● Xクライアント・ライブラリ

Xウィンドウ・システムを使用するためのクライアント側C言語ライブラリで，2次元のグラフィック機能をサポートする．

● PEXクライアント・ライブラリ

Xウィンドウ・システム環境下で，3次元

* i860XPは米国インテル社の商標である．

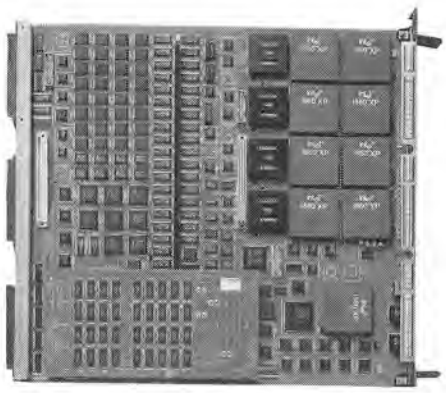


写真 2 PGB

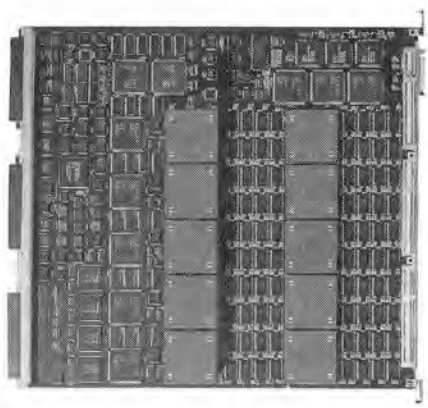
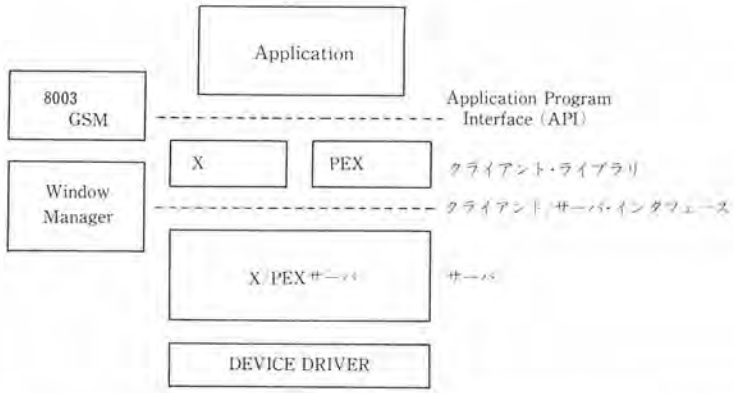


写真 3 IGB



グラフィック機能を実現するために用意された，国際標準の PHIGS とほぼ同じ，クライアント側 C 言語ライブラリである．

● X/PEX サーバ

X/PEX ライブラリからの要求を受け，実際のグラフィック処理を行う．ワークステーション上で稼働するソフトウェアの部分と高速処理を実現するためグラフィックス・アクセラレータ上で稼働するファームウェアの部分からなる．

● ウィンドウ・マネージャ

カラーモニタ上に表示されている X ウィンドウの枠編集や表示位置，サイズの変更およびアイコン化等をオペレータが会話形式で行うために用意された管理プログラムである．

掲載論文梗概

筆者等はUNIX*のもとで利用できるCAIシステム『LearningNavigator』を開発した．橘田明・山田繁夫・醍醐裕之は**C++を利用したCAIシステムの開発**の中で，『LearningNavigator』の概要，C++を利用した背景，C++の特徴・課題を述べるとともに，システムの核となるオブジェクト管理ルーチン『Object Manager』の実装について考察している．

筆者等はUNIX*のXウィンドウ**上で動作するGUIのための汎用的なライブラリ（Xmpp，Mui）を開発した．Xmppライブラリは操作画面（ウィンドウ）の構成部品（ボタン・メニュ等）をオブジェクトとして定義，Muiライブラリは任意のデータ構造をオブジェクトの集合として捉え，ウィンドウ上に視覚的に表現し，操作する機能を持ったライブラリである．伊藤直行・熊本厚志は**オブジェクト指向によるGUIクラスライブラリの開発**の中で，Muiの特徴，機能を中心に両ライブラリの開発について報告している．

近年，エキスパートシステム（ES）は，問題解決手法の一つとして実用化が進展してきた．また，ESのプロトタイピング開発手法は，新しいソフトウェア開発技術の一環として捉えられ広く活用されてきている．東野康臣は**エキスパートシステムの開発方法とツール**の中で，実用的なESの開発支援環境として，従来システムの一部にESを統合する際の開発方法，開発ツールと活用事例，今後のES開発実行支援環境についての展望を述べている．

染谷誠の**仕様言語Zとモデル化の考え方**は，まずZがソフトウェアの開発中に出会う諸問題の記述のための枠組みとして極めて優れた特性を備えていることを紹介し，それを具体例を通して検分している．次に，Zで仕様を書く場合，さまざまなスタイルが可能であるが，筆者としての一つの記述スタイルを提案している．

* UNIXオペレーティングシステムは，UNIX System laboratories, Inc.が開発し，ライセンスしている．

** Xウィンドウは MITの登録商標である．

掲載論文梗概

ISSN 0914-9996

技　報
UNISYS TECHNOLOGY REVIEW
Vol. 13　No. 2　(No. 38)

発　行　日　平成5年8月31日
編集発行人　柳　生　孝　昭
発　行　所　日本ユニシス株式会社
東京都江東区豊洲1-1-1 〒135
TEL(03)5546-4111（大代表）
印　刷　所　三美印刷株式会社