TECHNOLOGY REVIEW

UNISYS 技報

通巻 47

1995年11月発刊 Vol. 15 No. 3

特集：TIPPLER

日本ユニシス

ISSN 0914-9996

保科剛の**統合開発支援ツールTIPPLERの開発**は，変化の激しいオープンシンシステム環境におけるソフトウェア開発について，まずTIPPLERの概要について述べ，TIPPLERの開発を例に，各開発フェーズの概要・工夫・評価および課題について述べている．

TIPPLERはオブジェクト指向のGUIアプリケーション開発のためのソフトウェアであり，それを用いるためには，Uniscriptと呼ばれるTIPPLER固有の言語によりプログラミングする．杉野順清は**Uniscriptコンパイラ**の中で，Uniscriptの特徴の内，コンパイラに関係する事項の説明をし，さらにコンパイルする時の各ステップについて説明している．

オブジェクト指向プログラミング言語UniscriptはTIPPLERにおけるアプリケーション開発言語である．川辺治之は**TIPPLERの言語モデル**の中で，Uniscriptの言語設計および処理系開発を通じて明らかになったオブジェクト指向言語およびGUI定義言語に要求される機能および問題点を中心にUniscriptの言語モデルについて紹介している．

TIPPLERのプレゼンテーションモデルとは，ディスプレイ上のウィンドウや帳票はオブジェクトを表示するための枠組に過ぎず，クラスとしては定義しないとする，GUIを取り込むために拡張されたオブジェクトモデルである．プレゼンテーションモデルは，宣言的記述と表示の自動更新機能という二つの大きな特徴により，TIPPLERプログラミングの生産性を高めている．伊勢本勝一は**TIPPLERのプレゼンテーションモデル**の中で，TIPPLERにおけるプレゼンテーションモデルについて，MVCモデルとの比較を中心に説明している．

TIPPLERの使用により，リレーショナルデータベースを基にしたC/SSアプリケーションを容易に開発する事ができる．また新しいニーズであるマルチメディアデータを扱うアプリケーションも開発可能である．大石光太郎の**TIPPLERとオブジェクト指向データベース**は，TIPPLERの使用者にとり，より使い易いデータベースアプリケーションの開発環境を目指した時，どのようなデータベース機能が要請されるかを考察し，それを実現するするためにオブジェクト指向データベースとの枠組を提案している．

TIPPLER/V Factoryは，TIPPLER/VのUniscript言語に対応したCASEツールである．平井誠人の**TIPPLER/VのCASE機能－Factory**は，本機能開発の大きな課題であった「ソフトウェアの巨大さと複雑さ」を，機能を複数のプログラムの集合体として実現する事で見かけ上のソフトウェアの大きさを，データ構造とアルゴリズムを可能な限り単純化する事でソフトウェア全体の複雑さを，押さえる事ができたことを報告している．

三池良洋の**TIPPLERアプリケーションとInterleaf 5の連動**は，ソフトウェア開発に伴うドキュメンテーションに焦点を当て，TIPPLERのCASEツールと連動した仕様書の自動作成の仕組みを，統合文書管理システムInterleaf 5を活用して実現することを目的としたプロトタイプ・モデルの開発を行った際の報告であり，本システムの機能概要，開発の背景およびインプリメンテーションを述べている．

オブジェクト指向方法論にいまひとつ実績の裏付けがない現時点では，システム分析設計者が，ソフトウェアプロダクトライフサイクルの観点から「開発」の位置付けを見直し，オブジェクト指向の方法論的研究と同技術の実践による方法論の検証を行う必要がある．それには，システム分析設計者自らのオブジェクト指向技術を適用したプログラム開発，方法論の検証の積み重ねが重要である．野村潤一郎の**TIPPLERを使ったオブジェクト指向技術に基づくアプリケーションシステムの開発**では，TIPPLERによるオブジェクト指向開発の現状と問題点を指摘し対策について述べている．

特集「TIPPLER」の発刊によせて

原　　潔

本号はオブジェクト指向の統合開発環境ソフトウェアであるTIPPLERに関する特集号である．

90年代に入ってデータを単に処理することから，業務にとって価値のある情報として時宜を得て操作・加工することが強く要求され始めた．この状況を知的活動支援システムへの要求として捉え，使いやすいユーザインタフェースと思考を妨げない応答処理を実現することがその基本だと考えた．当時としてはPCはまだウィンドウシステムも処理時間も要求に対しては不十分なものであった．そこでエンジニアリング分野で操作性や処理時間で十分な実績を上げていたUNIX* ワークステーションをプラットフォームとして採用することにした．しかし，当時UNIXワークステーションには適切な開発環境はほとんどなく，知的活動支援アプリケーションを開発するプラットフォームをまず準備する必要があった．そのようにして生まれたのがTIPPLERである．このコンセプトは市場の要求に十分に適合し，UNIX版TIPPLERは大変評価され，多くのアプリケーションがTIPPLERで開発され利用されている．

このTIPPLERは95年3月末に，財団法人ソフトウェア情報センタ（略称：SOFTIC）からソフトウェア・プロダクト・オブ・ザ・イヤー'94（第6回）を受賞した．オブジェクト指向技術を適用した初めての統合開発環境であり，機能的にも十分な内容を持っているというのが受賞理由である．大企業の全社的なシステムの構築にも有効で，これまでの第四世代言語やCASEツールの限界を超えることが期待されると言及された．TIPPLERの開発コンセプトが第三者からも評価されたものと思う．

技術の発展は早く，PCの性能の向上は目を見張るものがある．そしてWindows** システムの出現はユーザの理解しやすいアプリケーションを容易に作成することを可能にした．TIPPLERの開発当初目的とした環境がPC上でも可能になり，いち早く93年には開発を開始し94年7月にWindows版のTIPPLERを提供した．しかし，このときUNIX版との互換を開発方針にしたため，せっかくの良さもMS-DOS*** の制約に突き当たり必ずしもその良さを十分に発揮することができなかった．しかし，その後WindowsNT** の出現によりTIPPLERを使って開発する高度で複雑なビジネスアプリケーションを実現する環境が提供されることになり，大規模なクライアント/サーバアプリケーション開発のための開発・実行環境としてWindowsNT版TIPPLERを95年10月に提供開始した．

---

* UNIXは，X/Openカンパニーリミテッドがライセンスしている米国ならびに他の国における登録商標である．

** Windows，WindowsNTは米国Microsoft社の商標である．

*** MS-DOSは米国Microsoft社の登録商標である．

TIPPLER 開発の歴史は，ユーザに知的な活動を支援するシステムを構築するためのプラットフォームを提供するという目標のもとに，ダウンサイジングするコンピュータ環境にいち早く適合し，オープン化の流れに従いながら，ソフトウェア生産の生産性と品質の向上に大きく貢献すると期待されるオブジェクト指向技術を具現化するものである．TIPPLER そのものを開発するに当たってあるいは TIPPLER を使ってユーザのアプリケーションを開発するに当たって遭遇した様々な課題は前例のないものが多く，それらを解決していった経験はそれだけで価値のあるものである．

本特集では，TIPPLER の設計方針と開発の技術について報告すると共に，TIPPLER を使って開発された先進的なアプリケーションの事例の紹介，そして変化の激しいオープン化，ダウンサイジングの動向の中でユーザの望むプラットフォームソフトウェアをいかに企画しマーケティングし開発していったかをまとめた．

大規模なネットワーク環境の上に知的な活動を支援する効率の良いシステムを生産性良く開発することの必要性はますます高まってきている．ここにまとめた技術報告が今後の情報システム構築に携わる技術者の参考になれば幸いである．

（システム技術本部知識システム部 部長）

統合開発支援ツールTIPPLERの開発

The Development of TIPPLER - an Integrated Systems Development Support Tool

保　科　　剛

要　約　TIPPLERの開発を例に，変化の激しいオープンシステム環境におけるソフトウェアの開発について考察する．半年ごとに，統合開発支援ツールとして大幅に機能強化し出荷する一方で，ANY UNIX*，ANY PCなどのマルチプラットフォーム対応を進めてきた開発の過程とそこで行った工夫および課題について述べる．

Abstract　Taking the development of TIPPLER as an example, this paper considers the production of the software that responds to the open systems environment which is undergoing rapid changes. TIPPLER, whose new versions have been shipped every six months with a wide range of functional enhancements implemented as an integrated systems development tool, has now so grown that it can meet "any UNIX/ any PC multi-platform" requirements. This paper is intended to describe how the development effort has been made, what ideas have been incorporated, and what needs to be further done to TIPPLER.

1. はじめに

ハードウェアの低価格化・高性能化，ネットワークの普及が進み，IT（情報技術：Information Technology）は大きく発展している．1990年代前半，ITのテーマの一つは，誰もが一人一台自分の前に計算機を置いて文房具のように計算機を利用できる

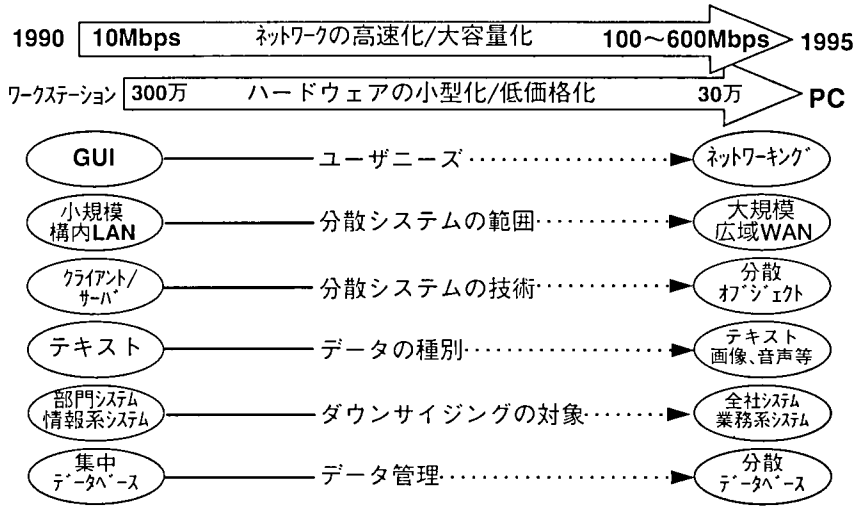


図1　情報技術の発展

* UNIXは，X/Openカンパニーリミテッドがライセンスしている米国ならびに他の国における登録商標である．

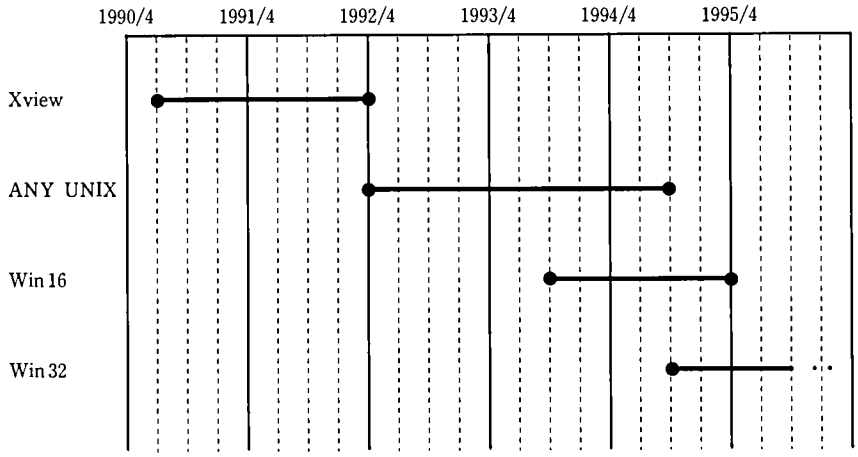


図 2 TIPPLER 開発の概略

ようにすることだった．1990 年代後半のテーマの一つは，そのようにして分散された計算機を繋ぎ合わせることによって，これまでにはなかった新しい計算機環境を構築し，企業で言えば，いわゆる業務系基幹システムをその上に実現することである．換言すれば，前半は集中から分散へ，後半は自律協調分散と捉えることができる(図 1)．

このような中で，1990 年 7 月，我々は統合開発支援ツール TIPPLER の開発に着手した．この時期の GUI の変化は著しく，Xview*，Motif*，Windows* が，その主役を担っていた．今後ともオープンシステムの世界では，同様のことが続くと考えられる．例えばオブジェクト指向技術をベースとして，通信分野では COM と CORBA，データベース分野では OQL と SQL 3，プログラミング言語では C++に加えて COBOL 97(オブジェクト指向コボル)が群雄割拠するであろう．競合があるからこそ成長するのもオープンシステムの特徴であり，我々は，このことを肯定的に捉え，うまく利用していくことが得策である．TIPPLER の開発の中で，GUI の変化にどのように対処してきたかを振り返ることで，オープンシステム環境でのソフトウェア開発時の工夫や留意点について考察する．2 章では，TIPPLER の概要について述べる．3 章から 6 章では，各開発フェーズの概要とそこで行った工夫とその評価および課題について述べる(図 2)．

2. TIPPLER とは

TIPPLER とは，『システムについて専門家でない人でも直観的に利用できる使い勝手のよいシステムを容易に開発・保守するためのソフトウェアプラットフォーム』である．1990 年代前半，ハードウェアの低価格化・高性能化に伴い，ダウンサイジングが急速に進んだ．今や我々の回りには PC が多数設置されている．それまで，集中・集約の一途を辿ってきたホスト集中システムに加え，情報を現場に分散・還元したり，多様な情報を発生源に置いておき日常業務の中で活用するシステムが必要とされた．TIPPLER は，このような要求に応えるプロダクトとして設計された．その要件を，シ

* Xview は OPEN-LOOK に準拠したウィンドウシステムの仕様であり，Sun Microsystems, Inc.の登録商標である．また Motif は Open Software Foundation の登録商標であり，Windows は Microsoft 社の商標である．

ステムの利用者・開発者の観点からまとめれると以下のように整理することができる．

利用者

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

開発者

- ソフトウェアプラットフォームの整備と活用
- GUIプログラミングの生産性向上
- アプリケーションの部品化と再利用
- 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
- プロトタイピング・アプローチ

これらを受け，TIPPLERの設計方針は，GUI記述を言語モデルに取り入れたオブジェクト指向プログラミング言語とその開発支援ツールおよび抽象度の高い外部インタフェースであるクラスライブラリを提供することとした．

3. オリジナル版の開発

開発開始後，6か月で開発ツールとしての機能をリリースすることが求められた．どれくらいの期間で作れるかではなく，6か月という期間を先に決めて作り方を考えることが課題だった．昨今では当たり前の話になりつつあるが，先にコストや期間を決め，その制約の中で物作りを考える方式である．オープンシステムのように変化が激しい環境においては，時間が経つと状況が変化してしまう．オリジナル版の開発におけるテーマは，短いサイクルで製品を開発するフレームワークを見定めることであった．6か月を一サイクルとする開発を度々繰り返している中で，様々な技術面，管理面でのノウハウを積み重ねていくことになった．

開発時に選択したプラットフォームは，Sunワークステーションである．32ビットアーキテクチャの高速なCPUとOS，高解像度ビットマップディスプレイとXウィンドウシステム，ネットワークおよびデータベースなどのソフトウェアが充実していたことが選択のポイントであった．機能・性能要求を満たしつつ，6か月毎のリリースを十分に可能にするプラットフォームであったが，さらに確実なものにするための我々の工夫は開発環境を整備することだった．

- 一人一台のワークステーション

  開発者自身が自らGUIを肌身で感じること，高速なCPUを活用して開発ドキュメントの作成・編集，プログラムの作成・編集，コンパイル・デバッグの効率を最大限にあげることを狙った．ハードウェアの低価格化・高性能化により，マシンの前で人が待たされることがコスト的に合わなくなっていた．

- インターネットに接続し外部から最新情報を得る

  開発を始めたときは，日本語OpenWindowsすらないありさまだった．海外のftpサイトからソースコードを入手したり，各種ドキュメントを収集した．ワ

ークステーションを活用するために各種フリーソフトウェアを入手し，要員に教育，生産性の向上を図った．ソースコードを持っていることで，不具合が発生したときの対応を早めることができ，また機能の確認をソースコードを通して行うことができた．生産性・保守性への寄与は大きかったと思う．

・電子メールの導入

メールの効用の一つとして非同期な同報がある．開発者は，自分のペースを崩さず開発に専念し，情報が必要なときにアクセスし，全員で共有する．もちろん，実際に人が集まって議論することもあるが，それは，そうすることがふさわしいときに限る．

・各種ツールの導入

UNIX には，長い間蓄積された便利なツールが数多くある．FSF(Free Software Foundation, Inc.) から提供される GNU（詳細は，ftp: //prep. ai. mit. edu/pub/gnu/GNUinfo/GNU)を始め，さまざまなツールをインストールして利用した．ソースコードがあることで，トラブル時の対応に役に立った．しばしば，フリーソフトウェアはベンダーの製品よりも性能，品質とも優れていたことを明記しておきたい．例えば，トラブルを追求していく中で，ベンダーのソフトウェアをフリーソフトウェアと入れ替えることで，回避できたことも少なくなかった．オープンシステム環境でのソフトウェア開発においては，フリーソフトウェアを適切に活用することもテーマになると考えられる．

3.1 評価と課題

開発は想像していたよりも順調であった．その要因として，少人数で開発を進められたこと，要員の技術水準・モラルが高かったことがあげられる．標準化された開発工程など皆無に等しかったが問題にはならなかった．開発要員の個々のスキルを高く維持することに加え，開発ラインの拡大に備えた工程の整備が課題に残された．

もう一つの要因は，設計コンセプトを早い時期に確立できたことである．中核部分の仕様は，設計開始後1か月で決定し，2か月後にはドキュメンテーションされ，いまも変わっていない．設計コンセプトを一貫させ，詳細な仕様を6か月毎に見直し，状況に応じて調整していったことがよかった．

設計コンセプトは IT を基軸に発想することが重要であるが，状況に応じた見直しは市場に目を向けることが肝要である．そのためには地道なマーケットリサーチが欠かせない．データによる裏付けのない決定は問題の要因となった．例えば，我々の誤算の一つは，グラフのレパートリを広げすぎたことである．何が本当に必要とされているのか，データによる裏付けを取らなかった．結果として使用頻度が著しく低いグラフアイテムがいくつか開発された．アプリケーション開発で言えば，開発したものの利用されていないモジュールが多々あるというケースと同じだ．後ほど，5章でも触れるが，コアを見誤ってはいけない．コアでないなら，外から持ってくることを，真剣に考えるべきである．オープンシステムでは，いろいろな製品があり，同種製品の和集合を取ると，とんでもなく巨大な仕様になる．そして，ついついその和集合に目がいってしまうのである．

4. ANY UNIX

1992 年 3 月までに当初計画した機能の開発を終え，1992 年 4 月より，開発のテーマはマルチプラットフォーム対応に移った．稼働プラットフォームの候補は，Sun，U 6000，HP，RS，NEWS，EWS 4800 であった．この時点での TIPPLER の稼働環境は，BSD 系のオペレーティングシステムである SunOS 4．x，ウィンドウシステムは Xview だった．マルチプラットフォームの対象は，SVR 4 系，Motif 系である．ANY UNIX でのテーマは，開発ラインの拡大にいかに対応するかであった．

・慣れたプラットフォームで開発する

Sun 以外のマシンに精通した要員がいなかったので，可能な限り Sun 上で移植することにした．まず，Sun の上で，Motif，SVR 4，ANSIC に対応したベースコードを作成し，各プラットフォームに展開することにした．展開の初期にアセンブラコードの可搬性で問題が発生したので，アセンブラ部分をすべて C で書き換えた．Sun 以外のマシンには，専任の担当者を小人数割り当て，個別ノウハウの集約を図った．

・ソースコードを一本化し世代管理する

マルチプラットフォーム対応後にソースコードを一本化しておくことが，将来に渡って保守性を確保するために不可欠と考えた．また，テストとデバッグは，機種ごとにチームを組んで進めるため，一つのソースコードに同時に複数人が修正を加えることが予想された．このため，複数人による並行開発をサポートするソースコード管理ツールを導入した．

・開発工程を整備する

対応機種の拡大に伴い，にわかに開発要員を増員したので，小人数で開発を進めてきたときのように意志の疎通が図れなくなった．このため，開発工程の整備が不可欠となった．自分達の作業のシステム化として，トラブル情報検索システム，出荷管理システム，勤務票システム，工数管理システムを TIPPLER を用いて開発した．また，部分的に手順書や書式を用意した．

4.1 評価と課題

プラットフォーム展開の初期に判明したので，大きな問題にならなかったが，アセンブラを残しておいたのは良くなかった．

開発工程の整備は，個別のツール，標準文書，書式を作成するに留まり，全体を見渡した工程設計にいたらなかった．そのため包括的な開発標準マニュアルのないまま開発ラインが広がり，品質の低下を招いた．この種の作業は，ボトムアップではなくトップダウンに進めるべきであった．

ANY UNIX 対応においても，マーケットリサーチが課題に残された．マルチプラットフォーム対応後，実際に利用されているものはごく一部である．

5. Windows 3.1 版

1993 年 10 月，Windows 3.1 版の開発に着手した．UNIX のマルチプラットフォーム対応と並行作業し，6 か月後に出荷することが前提である．Windows 3.1 版開発のテーマは，性質の異なるプラットフォームへの移植を技術面，管理面を含めて短いサ

イクルで実現することである.

・コアを定義する

時間的制約もあり，移植に先立ち，何を移植すればTIPPLERを移植したことになるのか検討した.UNIXでの開発をひと通り終えたTIPPLERを改めて見直すよい機会であった．このような見直し作業は，もっと短い間隔で定期的に行った方がよいと思う．移植の対象とする機能を絞りこむことを通して，何が本質なのかを見直せた．TIPPLERの特徴は，以下の3点に絞りこまれた.

—簡潔な言語仕様

—プレゼンテーションモデル

—日本語識別子

グラフを最小限に抑え，データベースインタフェース，帳票インタフェースを残した.

・調査レポートを作る

ネットワークボード，データベースミドル，何を取っても多くの製品があり選択に困惑した．そこで，テーマごとに，分担を決め調査レポートを作成し，それを元に選定した.数多く製品が溢れるオープンシステムでは，勘に頼らず，きちんとデータで語っていく必要がある．実際にテストしてみることができれば，なおよい.

・ツールを整備する

UNIXに慣れた開発者にとって，Windowsの開発環境は，極めて貧弱に感じられた.使い勝手の悪いエディタ，遅いコンパイラなどなど.そこで，UNIXマシンとPCをネットワーク接続し，UNIXで開発が進められるようにし，ソースコードの変換ツール,自動的なソースコードの転送ツールをなどを用意した.

・要員

TIPPLERの実装では，オブジェクト指向技術，コンパイラ技術，GUI技術が主要な要素技術である.また，性質は異なるが，Cを使ったプログラミング技術も必須である.UNIXとWindowsという,かなり異なる環境の上にソフトウェアを開発するに際し，Windows固有知識をもつ技術者を増強した.いろいろな知識，技術を持った人達が雑居していることが発想の幅を広げるために有効だと思う.

5.1 評価と課題

PC系のソフトウェアはソースコードが無く，また仕様が曖昧なので戸惑った.基本的に『あるがまま』であり，これを受け入れなければ前に進めない．以後の課題として，あらかじめ簡単なプロトタイプ作成を設計工程に組み込むことが残された.

WindowsとUNIXのギャップは大きく，互換性で吸収しようとすると二者択一とならざるを得ない部分が少なくない.我々は,UNIX互換をWindows上に実現する事を方針としたが,結果的にWindowsユーザから納得を得られなかった.重要なのは互換性ではなく相互運用性を実現することだったのだ.これは,OSやウィンドウシステム固有の問題ではなく一般的な課題である.マーケットリサーチも課題として残された.

6. Windows NT 版

1994 年 10 月，Windows NT への移植に着手した．Win 16 API から Win 32 API への移植，Windows らしいユーザインタフェースと開発環境の提供が目標となった．開発ラインを分離し，これに合わせて，TIPPLER/V と命名した．

・全開発工程の標準マニュアルを作成する

トップダウンアプローチで全工程を見直すことにし，オープンシステム環境用のソフトウェア開発工程定義書を作成した．各工程の成果物を明確にし生産性基準を設定，計数化の方法を定めた．

・開発環境を整備する

UNIX を利用することなく一通りの作業を WindowsNT で行えるように，ネットワークおよび開発ツールを整備した．UNIX に戻らないとできないことをなくすことが目標だった．

6.1 評価と課題

Win 16 開発時の課題を受けて講じた対策は，一定の効果を出しているようである．不便を感じることなく，UNIX 無しで作業を完結する事ができるようになったことは大きい．自ずと Windows をベースにして発想するようになった．やはり，まずは飛び込むこと，そして退路を絶つことなのだろうか．

今後の課題は，開発工程を継続的に改善していくことである．標準工程を文書化し，計数化の方法と各種基準値を設定した事で，進捗の把握，品質の把握，工程の改善のスピードは一気にあがった．

7. ま　と　め

TIPPLER の開発を通して，我々が体験した課題を，技術面，管理面で再度整理してみる．

技術面

先見性のあるコンセプトを早期に確立し，その中でコアを定義する．すでにあるもの，新しく出てくるものとの協調は，各要素の自律性を生かし，互換性よりも相互運用性を重視する．実社会において，コアコンピタンスやバーチャルコーポレーションがキーワードになっているが，ソフトウェアの構造においても同じことがいえると思う．オープンシステムでは，次々と新しく便利な製品や技術が登場し決して留まらない．しかも変化のスピードは早い．この状況に対応するためには，ソフトウェアの相互運用性が一つのキーワードになると考える．

・コンセプトの確立

・コアの定義

・相互運用性

管理面

道具の善し悪しは生産性に大きく影響を与える．客観的に計測し分析するためのフレームワークを持ち，標準化/明文化された工程は，ワークフローの絶間ない改善に不可欠である．あわせて，道具や工程の変化に追従し，それらの基

盤を最大限に活用し開発業務を遂行できる技術者を育成していかなければならない．変化を躊躇せず，とにかく飛び込んでいく柔軟さが望まれる．

・道具をそろえる
・工程を整備する
・要員を育成する

8. おわりに

短い開発サイクルを前提にすると，さまざまな発想がわいてくる．期間内に全部作れなければ，外部との提携・調達を考えざるを得なくなる．製品のコアを明確にし，ソフトウェア構造に反映する．ソフトウェアを部品化し，入れ替え可能な構造を作り込む．そして，アウトソーシングできるものは，徹底的にアウトソーシングを図る．そのためには，バーチャルコーポレーション/デパートメントを意識したオペレーションや，それを支えるマネージメント/リーダーシップが必要である．

短い期間で開発を終えても，そこから販売を考えるのでは出遅れる．また，高機能，高性能に越したことはないが，一般的にこれらは，価格や時期とトレードオフの関係にある．機能，性能，価格，出荷時期などあるゆる商品の特性を検討の対象としなければならない．そのためには，開発の初期から，企画部門，販売部門，製造部門が一体となり，あらゆる角度から，データを集め，数字よる客観的な検討をすること不可欠である．オープンシステム化を進めることは，ソフトウェア開発の BPR (Business Process Reengineering) を進めることに他ならないと思う．

参考文献 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.

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．

Uniscriptコンパイラ

Uniscript Compiler

杉野順清

要約 TIPPLERはオブジェクト指向のGUIアプリケーション開発のためのソフトウェアであり，それを用いるためにはUniscriptと呼ばれるTIPPLER固有の言語を用いてプログラミングする．

アプリケーションを開発するためにTIPPLERにはビジュアルエディタも用意されており，それを用いることでGUI部分のプログラミングはできるが，手続きなどいわゆるロジック部分は直接Uniscriptを記述する必要がある．また，ビジュアルエディタも最終的にはUniscriptを生成するため，内部的にはUniscriptが中心になっていることには変わりはない．

本稿ではこの「Uniscript」の特徴のうちコンパイラに関係するもの（オブジェクト指向言語，GUI表示モデルであるプレゼンテーションモデル，ガベージコレクション）を説明し，コンパイルをするときの各ステップ（字句解析，構文解析，意味解析，コード生成）について説明する．

Abstract TIPPLER is a software product which serves to develop object-oriented GUI applications, and programming requires its own language called Uniscript.

To help develop applications, Uniscript also provides a visual editor, which supports programming for a GUI, but logical parts such as method definitions need to be directly described in Uniscript. Since the visual editor also produces Uniscript programs as its end result, it can be said that Uniscript is one of the core modules of the TIPPLER system.

Out of the features that "Uniscript" offers, this paper exclusively discusses compiler-related ones (an object-oriented language, a presentation model positioned as a GUI model and garbage collection) in addition to all steps involved in compiling (lexical analysis, syntax checking, semantic analysis and code generation).

1. はじめに

TIPPLERはオブジェクト指向のGUIアプリケーション開発のためのソフトウェアであり，それを用いるためにはUniscriptと呼ばれるTIPPLER固有の言語を用いてプログラミングする．

アプリケーション（AP）のGUI部分の開発のためにTIPPLERにはビジュアルエディタが用意されており，それを用いることで簡便にGUIの開発ができる．また，そのビジュアルエディタもUniscriptを生成/編集するため，通常のテキストエディタで直接Uniscriptを編集してGUIの開発を行うこともできる．しかし，手続きなどいわゆるロジック部分は直接Uniscriptを記述する必要がある．

本稿に記載の会社名，商品名は，一般に各社の商標または登録商標である．

その Uniscript を解釈し実行コードに変換するために「Uniscript コンパイラ」というものが用意されている．それはCのコードを生成するトランスレータになっている．これは，他のライブラリ（C 言語標準ライブラリや Motif* や Windows** などのグラフィクスアクセスライブラリなど）を用いるためにも必要なことであり，このことは Uniscript の重要な特徴の一つで，これにより汎用性/移植性/拡張性などが高まっている．

本稿ではこの「Uniscript コンパイラ」の内部構造の概要を説明する．

2. Uniscript 言語

Uniscript 言語は次のような特徴を持つ．

- オブジェクト指向言語（OOPL）である．
- 表示モデルとしてプレゼンテーションモデルを採用している．
- ガベージコレクション（GC）を自動的に行う．

これらについて，ここでは本稿の目的に反するため詳細には述べないが，コンパイラを説明するために最低限必要なものの概略だけ述べておきたい．ここで GC は他の二つの項目に比べて言語仕様としてはそれほど大きな部分を占めないかも知れないが，実際にプログラミングをするにあたってこの機能による工数削減がかなり大きいため，あえて特徴の一つとしてあげた．

2.1 オブジェクト指向

Uniscript は OOPL としての基本的な性質を持っているが特筆すべきは次の項目である．また，同時にコンパイラとしてどのような作業が必要になるかを示す．

- 多重/多段継承が可能である

  継承という概念を持っていることは，OOPL になるための必要条件である．また，多重継承という概念を持っていることにより，「オブジェクトの部品化」において自由度が高くなっている．

  また，コンパイラが生成する C 言語は「継承」という概念を持たない．よって，コンパイラではこの継承を処理し適切に C 言語にマップした形でコード生成をしなければならない．

- 強い（宣言の必要な）データ型を持つ

  次に挙げるような型の処理が必要である．

  —関数の引数と仮引数や代入文の左辺と右辺などの型の整合性チェック

  —number 型と index 型のような相互変換可能な型の変換コードの生成

- 列や配列などを組み込みで持つ

  型の処理のみならずフレームアイテムの列/配列を参照するアイテムのためのコード生成をする必要がある．

2.2 プレゼンテーションモデル

プレゼンテーションモデルとは TIPPLER 開発初期時に名付けられたもので，フレ

---

* Motif は米国 Open Software Foundation 社の登録商標である．

** Windows, Windows NT は米国 Microsoft 社の商標である．

ームとクラスを別々に定義するというプログラミングスタイルを用いる．図1に示されるように，フレームはGUI Window表示方法の定義で，クラスは一種のデータ構造の定義を意味し，このモデルの特徴はそのフレームの定義とクラスの定義を明確に分離していることにある．

SmallTalkのMVC（Model View Control）モデルと対比すると，Modelの部分がクラスに対応し，ViewとControl部分がフレームに対応する．

図1にプレゼンテーションモデルの概略図を示す．

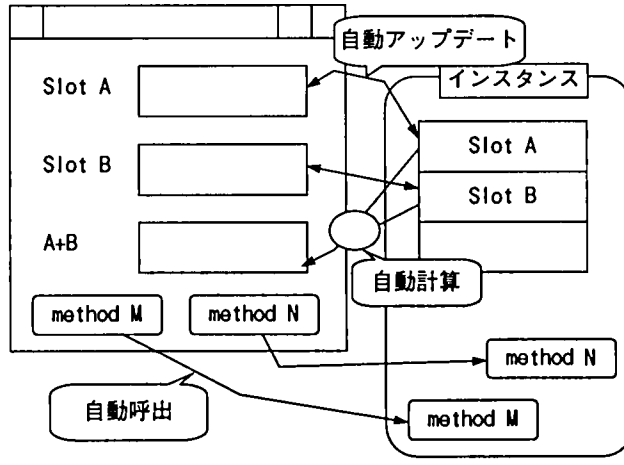


図1　プレゼンテーションモデル概略図

図1のプログラムは例：Uniscriptプログラムのようになる．

Uniscript プログラム

```
class C () -- クラスの定義
    A : number              -- Slot A の定義
    B : number              -- Slot B の定義
end
frame F C -- フレームの定義
    numericfield : A        -- Slot A の入出力のためのフィールド
    numericfield : B        -- Slot B の入出力のためのフィールド
    abbrevfield  : A + B    -- A もしくは B の変化により A + B は自動的に計算される
    methodbutton : M        -- method M の呼び出し(メソッドの内容は省略)
    methodbutton : N        -- method N の呼び出し(メソッドの内容は省略)
end
```

このモデルにより，ユーザは次のような利点を享受できる．

通常GUIプログラミングを行う時には次に示すいわゆるput/getメソッドを多量に書く必要がある．

[put/getメソッドとは]

putメソッドとはアイテム上に表示する文字列などを指定するための処理で，getメソッドとはアイテム上に入力された文字列などを実際のデータとして取り込む処理のことである．一般にこの処理は繁雑でかつ同様な処理の繰り返しである．これを示すために以下ではMotifを例にとってput/getメソッドを例示す

る．(Motif プログラムとしてはかなり省略してあり不完全であるが，繁雑さを示すのに十分なものにおさえた．逆にいうと Motif プログラミングはこれよりもっと繁雑であるということがいえる．また，Windows プログラミングであっても本質的に同じである．)

[例題]

インスタンスが一つあり，その中のスロットとして x, y がある．x, y は入出力が可能で各々数値型，文字列型とする．

```
struct Instance {
    int x;
    char* y;
};
Widget frame, labelx, itemx, labely, itemy;
void Expose( Instance* inst )
{
    frame = XtVaCreateManagedWidget( "bboard", xmBulletinBoardWidgetClass, .... );
    itemx = XtVaCreateManagedWidget ( "Text", xmTextFieldWidgetClass, .... );
    itemy = XtVaCreateManagedWidget ( "Text", xmTextFieldWidgetClass, .... );

    XtAddCallback (itemx,XmNvalueChangedCallback, (XtCallbackProc)value_change, NULL);
    XtVaSetValues( itemx, XmNuserData, inst, NULL );
    XtAddCallback (itemy,XmNvalueChangedCallback, (XtCallbackProc)value_change, NULL);
    XtVaSetValues( itemy, XmNuserData, inst, NULL );
}
set_X( Instance* inst, int data )
{
    char buf[256];
    inst->x = data;
    sprintf( buf, "%d", data );
    XmTextFieldSetString( itemx, buf );
}
set_Y( Instance* inst, char* data )
{
    if (inst->y) free(inst->y);
    inst->y = strdup(data);
    XmTextFieldSetString( itemy, data );
}
void    value_change (Widget w, caddr_t client_data, caddr_t call_data)
{
    char *jstr;
    Instance* inst;
    XtVaGetValues( w, XmNuserData, &inst, NULL ) ;
    if ( w == itemx ) {
        jstr = XmTextFieldGetString(w);
        inst->x = atoi(jstr);
    } else if ( w == itemy ) {
        jstr = XmTextFieldGetString(w);
        if ( inst->y ) free(inst->y);
        inst->y = strdup(jstr);
    }
}
```

このプログラムの Expose () を用いてウィンドウを表示することはできる．しかし，次のような繁雑さは残る．

- スロットへの代入結果が正常に表示されるように '=' を使ったプログラムではなく Set_X() などを呼び出さなくてはならない．
- スロットの値が他のプログラムやウィンドウに影響するならば，いちいち value_change() 関数をそれに合わせた形で書き直さなくてはならない．

さらに，このプログラムでは省略されているが次のような問題があり解決

をしなくてはならない.

- 一般に各フィールドにはラベルが必要でありその表示をする必要がある.
- itemx. itemy などがグローバル変数であるため,「インスタンスが複数個あり各々が別のウィンドウを持つ」場合などに何らかのデータをインスタンス自体に持つ必要がある.
- ユーザがマウスなどでこのウィンドウを閉じた場合, Set_X() などで使っている itemx は無効になるため, 閉じられたことをハンドルして Set_X() でエラーにならないような処理を記述する必要がある.

このように GUI プログラミングは非常に繁雑であり, ロジック部分より GUI 処理の部分の方が多くなってしまった AP も多数ある. また, よほど注意深くプログラミングしなければバグの混入の元にもなる. Uniscript コンパイラはこの put/get メソッドに対応するプログラムを自動生成する.

まとめると,「プレゼンテーションモデル」の導入により次の事項が大幅に改善されたといえる.

- 繁雑な put/get メソッドの記述の省略
- 上記省略によるバグの混入の低減
- オブジェクトモデルと独立に GUI を記述できるため, モデルの変化に伴う GUI 部分の変更を少なくできる.
- オブジェクトの内容の変化に伴う GUI 表示の変化を自動的に行える.

2.3 ガベージコレクション (GC)

GC の採用によるユーザはメモリに関する次のような作業から解放される.

- 取得したメモリが不必要になった時点を判断し開放するプログラムの記述
- 開放し忘れたときのメモリリークが発生していないかどうかのチェック
- 取得し忘れたときのメモリ破壊に伴うバグの修正

とくに最後の項目はデバッグの中でも最も難解な部類の一つである. それは, 取得し忘れによるメモリ破壊の影響がシステムストップ等の現象として現れるのは, 一般にずっと後になってからだからである.

このような, GC の採用によるプログラミング/デバッグの負担の減少は「プレゼンテーションモデル」の採用とともに Uniscript を用いるのに十分な理由になりうる.

さて, GC には色々なアルゴリズムがあるが, Uniscript で用いているのは「参照カウント法」というもっとも単純なものの一つである. また, メモリの取得/解放には C 言語標準ライブラリの malloc/free を直接使っており, コンパクションは行っていない. これはつまり次のことを示す.

- データ構造中に循環リストがある場合それをはずすのはユーザの責任であり, これを怠るとメモリリークが発生する.
- C 言語標準ライブラリ malloc/free はコンパクションを行わないため, メモリフラグメンテーションが発生する.

これはもちろん, 本格的な GC とはいえない. しかし, 循環リストに関してはプログラムでその循環を切るような記述を行うことにより対処することができる.

問題はフラグメンテーションであるが, 実用上はさほど問題になっていない. これ

コーディング例 1

```
variable x : C    -- C はクラス名
....
set x := new_C()
....
set x := new_C()  -- ここで古いインスタンスの開放が行われる.
```

コーディング例 2

```
variable s : sequence of C
set s := EMPTY
while ( 条件 )        -- このループでは malloc が大量に発生する.
    insert new_C() after x
    ....
end
set s := EMPTY        -- ここで一気に free が発生する.
```

は実際のプログラムはおおむね次のような性質を持っているからであると思われる.

- あるサイズのインスタンスを free した直後に同じクラスのインスタンスが必要になることが多い (コーディング例 1).
- malloc/free は連続して発生する (コーディング例 2).

もちろん, これらの問題点を解決している GC アルゴリズムを採用するのも選択肢の一つであるが, それのための計算コストは参照カウント法よりも大きくなってしまう. TIPPLER では上記理由により実用上問題ないと判断し, 完全さよりスピードを選択した.

3. コンパイラ概要

Uniscript コンパイラは C のコードを生成するため C 言語のトランスレータであるともいえるが, 通常のコンパイラと同様に大きく次の部分に分かれる.

- 字句解析
  入力文字列からトークンと呼ばれる構文解析上の最小単位を切り出す.
- 構文解析 (Syntax チェック)
  トークンから構文木を生成する.
- 意味解析 (Semantic チェック)
  型の整合性のチェックや識別子のスコープの処理などを行う.
- コード生成
  C のソースコードを生成する.

また, コンパイラは usc と mkmf に分類され, 各々次のような処理を行う.

- usc : コンパイルを行い file.c, file.i, file.f, file.v を生成する. (各々のファイルの説明は後の章で行う.)
- mkmf : file.i, file.f, file.v を読み込む usmain.c を生成する. (つまり, usmain.c から file.i, file.f, file.v を# include する.)

全体のコンパイルの流れを図 2 に示す.

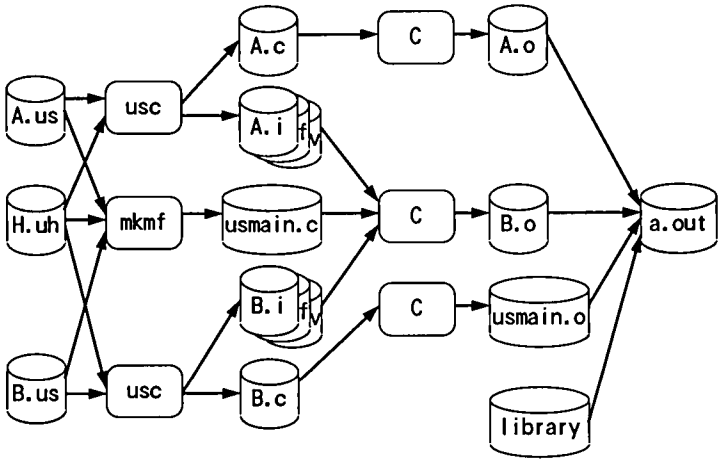


図 2　コンパイルフロー

4. 字 句 解 析

初期の段階では lex（もしくは flex）というテキストの簡易字句解析に用いるプログラムを用いていた．しかし現在は Uniscript の字句解析プログラム自体さほど複雑な処理が必要とされないため，スクラッチから記述されている．

字句解析により入力文字列は次のものに分類される（詳細は Uniscript のマニュアルを参照のこと）．

- 即値

  即値には，整数値，実数値，文字列，時刻がある．
- 識別子 abc や tbl など変数などの名前に用いられるものである．
- キーワード Uniscript ではかなり多くのキーワードがあるが，その多くは識別子としても使えるよう工夫がなされている．たとえば，NULL は変数名として使用できないが，block は変数名として使用できる．次にその一覧を示す．ここで，“*”が付いているものは識別子として使用可能なものを示す．

EMPTY,	NULL,	STDERR,	STDIN,	STDOUT,	abbrevcell*,		abbrevfield,		activate,
add,	alertbutton,		and,	array,	ascending,	axis*,	axisx*,	axisy*,	bar*,
belt*,	beltd*,	beltgraph,	bitmap,	block*,	call,	case,	checkbox,	choicebox,	class,
close,	comment*,		constant,	continue,	create,	daemon,	deactivate,		decxpose,
delete,	descending,		do*,	else,	elseif,	end,	enumeration,		exit,
expose,	external,	false,	filler,	for,	format,	frame,	from*,	function,	gantchart,
gauge,	hashtable,	hook,	if,	in*,	include,	insert,	label*,	line*,	lines*,
loop,	menu,	menubutton,		merge,	message,	method,	methodbutton,		move,
not,	numericcell*,		numericfield,		of*,	open,	or,	others*,	otherwise,
panel,	pass,	picture,	pie*,	piechart,	pied*,	pipe,	qsort,	quit,	
radarchart,		read,	reject,	repeat,	resource*,	result,	return,	scatter*,	
scattergraph,		scrollinglist,		search,	selectionfield,		send,	sequence,	set,
slider,	sort,	spreadsheet,		stackbar*,	start,	stereogram,		stereox*,	stereoy*,
stereoz*,	stock*,	stop,	switch,	system,	text,	textcell*,	textfield,	then*,	timecell*,
timefield,	true,	type*,	until,	var*,	variable,	virtual,	while,	words*,	write,
x*,	xaxis*,	xcolumn*,	xygraph,	y*,	yaxis*,	yrow*			

scan.c(字句きりだしプログラム)

```
#define IDENTIFIER INTERN_
....
struct keyword {
  char *string;
  int number;
};

static struct keyword keyword_table[] = {
  ....
  "block", Block,
  ....
  "var", Var,
  ....
  "textcell", Textcell,
  ....
};

int yylex() {
  ....
  for (;;) {
    ....
    switch (yytype) {
    ....
    case 0:
      if ('a' <= c && c <= 'z' || 'A' <= c && c <= 'Z') {
        yytype = IDENTIFIER;
        *p++ = c;
      } else if ( .... ) {
        ....
      }
    case IDENTIFIER:
      if ('a' <= c && c <= 'z' || 'A' <= c && c <= 'Z' ||
          '0' <= c && c <= '9' || c == '_') {
        *p++ = c;
      } else if ( .... ) {
        ....
      } else {
        ungetc(c, fd);
        *p++ = '\0';
        yytype = keyword_search(yytype, yytext);
        ...
        return yytype;
      }
    }
  }
  ....
}
```

tip.y(yacc ソースプログラム)

```
    %token      Var Block Textcell      /* scan.c でキーワードとして処理されたトークン */
....
 INTERN:
    INTERN_     { $$=$1;}               /* 識別子として処理されたトークン */
  ....
  | Var                 { .... }
  | Block               { .... }
  | Textcell            { .... }
  ....
  ;
 variable_state:
    Variable { .... } variable_list     { .... };
 variable_list:
    reference_declare                   { .... }
  | reference_declare variable_list     { .... };
 reference_declare:
    INTERN ':' type initvalue_opt       { ... };
```

キーワードの一部が識別子として使用できるように yacc のプログラムに次のような工夫がされている（scan.c プログラム参照）.

scan.c はだいぶ省略しているので若干解説する.

—case 0 の部分

最初の字句切り出しの時は yytype に 0 がセットされている.

—case IDENTIFIER

最初の文字が'[a-zA-Z]' の時の処理.

—keyword_search() 関数の内容

第二引数を用いて keyword_table の string から（二分探索法で）対応する文字列を見つける. もし, 見つければ keyword_table の対応する number を返す. もし見つからなければ, 第一引数の値をそのまま返す.

tip.y プログラムのような処理を行っているにもかかわらず yacc で shift/reduce は発生しない. これは, Var, Block, Textcell などのキーワードは Uniscript 上では特定の位置にしか現れないことを利用しているからである. 逆にいうとどちらとも判別がつかないようなキーワードは識別子として利用できない（「区別がつかない例」参照）.

(区別がつかない例)

```
variable x      : sequence of string
         EMPTY  : sequence of string
set x := EMPTY -- EMPTY が空の列を示すか,EMPTY という変数を示すのかわからない
```

・記号

記号は一種のキーワードともいえるがキーワードは見かけ上識別子として変わらないため, ここではあえて記号という別の分類に分けた. 記号には次のものが該当する.

```
"+"   "-"   ".."  "..."  ";"   "("   ")"   "{"   "}"   "["   "]"   ":="  "->"  "^"   ":"
">"   ">="  "=>"  "<"   "<="  "=<"  "="   "/="  "+"   "-"   "*"   "/"   "//"  "@"   "**"
```

5. 構 文 解 析

構文解析は yacc（もしくは bison）というツールを使っている. これは, UNIX* 上に標準で添付されてくるコンパイラコンパイラで, 文脈自由文法を簡易オートマトン用のテーブル群に変換する. そのアルゴリズムには LALR(1)構文解析アルゴリズムを使用している. 逆にいうと Uniscript 言語はそれで解析できるような文法構造になっている.

Pascal の文法は LALR(1)より弱い LL(1)アルゴリズムで解析できるが, 唯一コ

* UNIX は, X/Open カンパニーリミテッドがライセンスしている米国ならびに他の国における登録商標である.

ンフリクトが発生するパターンがあるのが知られている．しかし，Uniscript は全くコンフリクトが発生しない．これは Modula-2 を参考にし，最初からコンフリクトが発生しにくい文法を目指しているためでもある．文法的な特徴は次のようなものがあげられる．

- if-then-else-end や while-end など，制御構造は必ず end が終端になっている．(よって，Pascal のようなコンフリクトが発生しにくい．)
- Pascal や C のように文と文との区切り文字(もしくは，文の終り文字)';'を用いていない．
- たとえば，代入に set というキーワードを用いる．
- C 言語の# include に相当するものとして include 文を設けている．
- いわゆるマクロの記述は出来ない．
- 変数のスコープにはグローバル変数，スロット変数，ローカル変数がある．(C のようなファイル単位のスコープはない)

6. 意 味 解 析

意味解析は 2 パスで行われる．これは，Uniscript が前方参照を許しているからである．それぞれのパスでは次のような作業が行われる．

- 最初のパス

  メソッドやクラスなどのトップレベルの名前がそれを管理しているテーブルへ登録が行われる．また，その時 TokenXXX という名前の struct が生成される．以下にクラス定義の場合の struct を示す．

```
typedef struct tokenclass {
    int                 linenumber;             /* クラス定義が行われたファイルの行番号 */
    char*               filename;               /* クラス定義が行われたファイル名 */
    TokenCategory       category;               /* 構造体の名前 */
    int                 check;                  /* 既に意味解析が行われたかどうか */
    int                 error;                  /* error が発生したかどうか */
    Intern*             name;                   /* クラス名 */
    InternList*         superlist;              /* スーパークラスのリスト */
    TokenList*          superclasslist;         /* スーパークラスのリスト */
    TokenList*          allsuperclasslist;      /* スーパークラスのリスト（継承含む） */
    TokenList*          slotlist;               /* スロットのリスト */
    TokenList*          allslotlist;            /* スロットのリスト（継承含む） */
    struct tokenhashtable*      slottable;
    /* slottable 以外のハッシュテーブルはローカルである */
    struct tokenhashtable*      methodtable;
    struct tokenhashtable*      frametable;
    struct tokenhashtable*      menutable;
    struct tokenhashtable*      daemontable;
    char                *codename;              /* 実際にプログラム上で指定されたクラス名 */
    char                *encodename;            /* C のコード生成に使用される名前 */
    struct tokenselector        *selector;      /* コード生成時に用いられる．*/
    struct tokenexfun   *new_fun;               /* new 関数 */
    int                 inheritance;            /* check_inheritance で用いるフラグ既に継承した */
    int                 inheritance_check;
    /* check_inheritance で用いるフラグ既にあらわれたクラス (=error) */
} TokenClass;
```

- 二番目のパス

  メソッドの場合は文，クラスの場合はスロットなどその内容の解析が行われる．最初のパスで生成されたすべての struct TokenXXX に対し意味解析関数

が起動される.

意味解析で行われる内容は, たとえば次のとおりである.(全体では二百数十パターンのチェックが行われる.)

- 同じグローバル変数名が定義されていないかどうか.
- 使用されている関数の定義があるかどうか.
- クラス定義中でスーパークラスに指定されているクラスが定義されているかどうか.
- send 文で指定されているメソッド名が定義されているかどうか.
- format 文で指定されている書式と引き数の型があっているかどうか.
- 配列の添え字の型があっているか.
- 集合型のリテラルに指定しているものが, 列挙型として存在するかどうか.
- ' [] ' で指定されているものが列, 配列, ハッシュ型であるかどうか.
- 同じローカル変数が使われてないかどうか.
- //演算子の左辺と右辺が文字列型かどうか.

これらのチェック項目のうちの一部をコードを中心に解説する.

```
typedef struct statesend {
    int                 linenumber;
    char                *filename;
    StateCategory       category;       /* 構造体の名前 */
    int                 check;          /* makeup */
    int                 error;
    struct expmethod*   exp;            /* メソッド呼びだしの式 */
} StateSend;

static State *check_StateSend( State *this_ )
/* this_ には send 文に対応する構文木がわたってくる. */
{
    StateSend   *this = (StateSend*)this_;
    if ( this_->check || this->error ) return this_;
    this->check = 1;
/* check は一度のみ行う. これは, 構文木は時には共有される為で構文
   チェックの効率の向上と同じエラーを二度出さない為である. */

    this->exp = (ExpMethod*)check_ExpMethod((Exp*)this->exp);
/* check_ExpMethod 式で '<-' で呼び出されるメソッドと共通に用いられる. */

    if ( this->exp->error ) {
        this->error = 1;
        return (State*)this;
    }
    if ( !isVoidType(this->exp->type) ) {
        /*warning-19 "#xxx# 戻り値が無視されます.#" */
        usc_errmsg(this->filename,this->linenumber,ERR_WARN,19,
                   getname_Exp((Exp*)this->exp),NULL);
    }
    return (State*)this;
}

static Exp *check_ExpMethod( Exp *this_ )
{
    ExpMethod   *this = (ExpMethod*)this_;
    if ( this_->check || this_->error ) return this_;
    this->check = 1;
    {
        TokenMethod     *method;
        TokenClass      *class;
        this->instance = check_Exp(this->instance);
/* instance には send 文のインスタンス式に対応した構文木が入っている.
   ここでは, その式のチェックを行う */
        if ( this->instance->error ) {
            this->error = 1;
```

```
            return (Exp*)this;
        }
        class = isClassType(this->instance->type);
/* instance で指定された式の型がクラス型かどうかをチェックする. */
        if ( !class ) {
            /*error-23 "#xxx# はインスタンスとして定義されていません.#" */
            usc_errmsg(this->filename,this->linenumber,ERR_ERROR,23,
                       getname_Exp(this->instance),NULL);
            ErrorFlag = 1;
            this->error = 1;
            return (Exp*)this;
        }
        if ( class->error ) {
            this->error = 1;
            return (Exp*)this;
        }
        method = get_TokenMethod(class,this->methodname);
/* class に定義済みのメソッドの取り出しをする.*/
        if ( !method ) {
            /*error-18 "#xxx# は #xxx# のメソッドとして定義されていません.#" */
            usc_errmsg(this->filename,this->linenumber,ERR_ERROR,18,
                       this->methodname->name,class->name->name,NULL);
            ErrorFlag = 1;
            this->error = 1;
            return (Exp*)this;
        }
        if ( method->error ) {
            this->error = 1;
            return (Exp*)this;
        }
        method->selector->reference = 1;
/* 後で対応するメソッドが一度も使われてないかどうかをチェックする為に,
   既に一度以上使われていることをセットする */
        this->method = method;
        this->type = method->type;
        if ( !this->type || isErrorType(this->type) ) {
            this->error = 1;
            return (Exp*)this;
        }
        if ( this->arglist ) {
            this->arglist = check_ExpList(this->arglist);
/* メソッド呼び出しの引数リストのチェック */
            if ( this->arglist->error ) {
                this->error = 1;
                return (Exp*)this;
            }
        }
        if ( check_methodcall_arglist(this->method,this->method->name->name,this->arglist,
                                      this->filename,this->linenumber) ) {
/* メソッド定義で指定された仮引数の型と実引数の型があっているかチェッ
   クする. */
            this->error = 1;
            return (Exp*)this;
        }
    }
    return (Exp*)this;
}
```

このように，意味解析部分では構文木を元に意味的なチェックを行うことが中心になる．しかし，一部次のようなチェックだけに留まらない部分もある．以下はその例である．

- frame の frameinfo 属性に対するメソッドの作成

  frameinfo に指定されたメソッドはコンパイラで対応するメソッドを生成することになっている．ユーザはそのメソッドを自由に呼び出して，フレーム情報を格納した FrameInfo クラスのインスタンスを得ることができる．

- 比較演算子の三項的な取り扱いに関する処理

  $x <= y <= z$ といった式を $x <= y$ and $y <= z$ という式に組み替える．

とくに frameinfo に対するコード生成をコンパイラが行うという仕様は Uniscript

コンパイラの言語仕様的な問題点の一つである．つまり，ほかの属性はすべてアイテムの振る舞いや表示方法などを変えるために，式を指定したりユーザ定義のメソッド名を指定するものであるが，この属性のみがコード生成を必要とする．

7. コード生成

コード生成は，構文解析と意味解析において問題が発生しなかった場合に行われる．このコード生成は大きく次のようなステップで行われる．

- .c ファイルの生成
  - —ヘッダ部分の生成
    - # include＜runtime.h＞等，どの .c ファイルにも共通なヘッダ部分の生成
  - —セレクタの生成
    - セレクタとは SlotSelector や MethodSelector などスロットのアクセスやメソッド呼出時に用いられるデータ構造である．セレクタには次の種類がある．
      - クラスセレクタ，スロットセレクタ，メソッドセレクタ，フレームセレクタ，メニュセレクタ
  - —プログラム本体の生成
    - プログラム本体として，次のものが生成される．
      - 関数（function），メソッド（method），フレーム（frame），メニュー（menu），デーモン（daemon）
  - —定義テーブルの生成
    - セレクタを用いて間接的に使用されるテーブルを生成する．
- .i ファイルの生成
  - —enumeration のデフォルトのフォーマット文字列の生成
  - —enumeration format に対応する文字列の生成
  - —定数文字列の生成
  - —クラス定義に対応するテーブルの初期化
  - —メソッド/フレーム/メニュ/デーモン定義に対応するテーブルの初期化
- .v ファイルの生成
  - —グローバル変数の初期化
- .f ファイルの生成
  - —new_xxx() といった，インスタンス生成関数の生成
  - —enum_XXX(), index_XXX() といった enumeration 型と index 型の相互変換のための関数の生成．
  - —enumeration format 用配列の定義
  - —frame 構造の宣言

Uniscript 上のクラス定義は一般に .uh ファイルに定義され，複数の .us ファイルから include されて用いられる．ここで，そのクラス定義に対応するコード生成をする場合 .us に対する .c ファイルに生成してしまうと，複数回コード生成されてしまう可能性がある．そこで，.i ファイルに# ifdef 付きで生成し，usmain.c からその .i ファイル

を#includeして用いる（「Uniscriptクラス定義」,「対応するコード生成」参照）.

Uniscript クラス定義

```
class C ()
    x : number
    s : sequence of string
end
```

対応するコード生成

```
#ifndef _C_C
{
  class = add_class( "C", NULL );
  add_slot( class, "s", SISEQ );
  add_slot( class, "x", SINUM );
}
#define _C_C
#endif
```

これを見るとわかるように，add_classが何度#includeされようが一度しかadd_class() およびadd_slot() は呼び出されない.

8. おわりに

Uniscriptは「1. Uniscript言語」で述べたような幾つかの特徴を持ち，他のオブジェクト指向言語では得られない生産性がある．また，それを現実のものとするためのコンパイラ技術も重要である．しかし，最も重要な点はその言語仕様であることは言うまでもない．TIPPLERの評価が高い理由の一つはそのためだともいえる.

また，TIPPLERは当初US-family上で開発されたのだが，最近はWindows NT版TIPPLER/Vもリリースされ，従来のUNIX版のみならず，本格的なWin 32 APIを用いたプログラミング環境もラインナップに加わった．現在の潮流としてUNIX上で開発されてきた種類のAPの多くがWindows NT上で動作するように移植されてきている．TIPPLERは評価が定まった製品であり，ネイティブなWin 32 APIを使用できるNT版は非常に魅力的な開発ツールといえる.

最後に，私自身TIPPLERの開発に携わり様々な経験ができ，それは今後の業務を行うに非常に有益であると思う．ここに，そのような経験をさせてくれたことに感謝し終わりの言葉としたい.

執筆者紹介 杉野順清（Junsei Sugino）

昭和40年生．平成2年豊橋技術科学大学大学院修士過程情報工学専攻修了，同年日本ユニシス(株)入社．システム技術本部知識システム部にてTIPPLERの開発に従事．現在オープンソフトウェア事業部マーケティング企画部所属.

TIPPLERの言語モデル

The Language Model for TIPPLER

川 辺 治 之

要 約 オブジェクト指向プログラミング言語Uniscriptは統合開発支援ツールTIPPLERにおけるアプリケーション開発用言語である．Uniscriptでは，クラスの定義に対して，オブジェクトのデータ構造はスロットにより，オブジェクトの振舞はメソッドにより定義する．ユーザインタフェースは，一つのオブジェクトに対して複数の表示形態を定義できるマルチプルプレゼンテーションモデルに基づき，TIPPLERが提供する部品を用いて宣言的に定義することで，プログラムの可読性，保守性を向上させる．また，定義されたプレゼンテーションの部品化する機能や，実行時に表示が変わるような柔軟なプレゼンテーションを定義できる機能を提供する．

Abstract Uniscript, an object-oriented programming language, is intended as the applications development language for TIPPLER, an integrated development support tool. In the case of Uniscript defining classes, the data structures of objects are defined by slots, and the behaviors of objects by methods. Applications user interfaces can be defined in a declarative way through the use of segmented items provided by TIPPLER based on the multiple presentation model which assists in defining multiple display styles for each object, thus improving the level of program readability and maintainability. Uniscript also provides such features as make it possible to modularize defined presentations and to define presentations flexible enough to permit a dynamic change in the course of the running time.

1. はじめに

1970年代から，システムで扱う対象をオブジェクトとしてとらえるオブジェクト指向[1]の考え方が，分析・設計・プログラミングにおいて用いられるようになってきた．オブジェクトとは，識別性を持ち，それぞれが自分自身のデータと手続きを管理するものである．

オブジェクト指向は，システム化の対象をより直接的に表現できるため，今では標準的な技法となっている．また，オブジェクト指向はウィンドウやプロセスをはじめとする計算機資源を表現するためにも適しているため，開発環境やオペレーティングシステムの記述にも用いられるようになった．

このオブジェクト指向によるプログラミングをより直接的に表現できるよう，言語仕様にオブジェクト指向機能を取り入れたプログラミング言語が現われてきた．最初に商用化されたオブジェクト指向プログラミング言語としてsmalltalk[2][3]がある．smalltalkはすべてがオブジェクト指向に基づいて設計されており，繰り返しや条件分岐などのプログラムの制御構造さえもオブジェクトに対するメッセージ送信という形で記述される．

一方，Lisp系言語では，その記述性の高さのためにさまざまなオブジェクト指向機能が実装され出したが，逆に，このためにアプリケーションの移植性の低下を招いていた．このため，Lispのオブジェクト指向機能を標準化するという方向で，ANSI Common Lisp[4][5]の一部としてCLOS（Common Lisp Object System[6][7]）が定義された．CLOSはこの上に様々なLispのObject指向機能を実現できる枠組を提供することも目標としており，これにより，CLOS自身でCLOSを記述することさえできるようになった．

また，Cに対するオブジェクト指向機能の拡張という形でC++[8][9]が定義された．C++におけるクラスはCの構造体型の拡張となっている．

これらのオブジェクト指向プログラミング言語の特徴をいくつかあげ，それらに対して統合開発支援ツールTIPPLER[10]におけるアプリケーション開発言語であるUniscriptの設計指針を述べる．

1) 手続き型プログラミングとの親和性

smalltalkはオブジェクト指向に基づいて新たに設計された言語であり，すべてがオブジェクト指向で定義されているという意味で，非常に単純な設計となっているが，従来の手続き型言語になれたプログラマにとっては敷居が高い．一方，CLOSやC++は従来の手続き型言語に対するオブジェクト指向機能の拡張であり，制御構造などは，もとの言語の構文に従う．このため，手続き型言語から移行しやすい．

2) 弱い型づけと強い型づけ

smalltalkやCLOSは変数などの記憶領域自体の型は定義せず，任意のデータ型を保持することができる．これは，プロトタイプ開発においては開発生産性を向上させるという利点があるが，コンパイル時に発見できないエラーもあり，実行時に型を検査しなければならないために実行効率が低下するという欠点もある．一方，C++ではCと同様に各記憶領域自体に型が定義されており，コンパイル時に型の整合性が検査される．

3) 型とクラス

smalltalkにおいては型はクラスと同義である．整数や文字列に対する操作も，それらのクラスに対するメソッドとして定義される．また，CLOSにおいては，従来のCommon Lispが持つそれぞれの型に対応するクラスを導入し，従来の型階層をクラス階層の一部として定義している．これにより，整数や文字列などの基本データ型に対する操作も，それらのクラスに対するメソッドとして定義することができる．ただし，Common Lispがもともと持っている基本データ型に対する操作は，関数として定義されている．

一方，C++では，定義されたクラスは型としても使用できるが，Cで定義されている基本データ型はクラスには対応していない．このため，基本データ型に対する操作は関数として定義しなければならない．

4) メッセージ送信

smalltalkでは手続きの呼出しはすべてオブジェクトに対するメッセージ送信という枠組で記述される．メッセージ送信において実行すべきメソッド名が指定

され，メッセージが送信されたオブジェクトのクラスに定義された同名のメソッドが実行される．この枠組はC++においてもオブジェクトに対するメンバ関数呼出しという形で記述される．ただし，C++では同名のメンバ関数でも，引数の型の組合わせが異なると，異なるメンバ関数として扱われる．

CLOSでは従来のメッセージ送信を，メッセージが送信されるオブジェクトを実質的な最初の引数とみなし，そのクラスによって実行されるメソッドが決定されるととらえて，これをそれ以外の引数にも適用できるようにした．すなわち，手続き（Common Lispでは関数）が呼ばれたときに，その実引数のクラスの組合せによって，実行されるメソッドを決定することができる．これは，従来の関数の拡張にもなっており，総称関数（generic function）と呼ばれる．メッセージ送信では，概念的には，メソッド名とそれがメッセージ送信されたときに実行するメソッドの対応表を，各クラスが保持していると考えることができるが，総称関数では，引数の型の組合せと，実引数の型がその組合せであったときに実行するメソッドの対応表を，各総称関数がもっていると考えることができる．また，CLOSでは，すべてのデータ型はTと呼ばれるすべてのクラスの最上位に位置するクラスに属するため，従来の関数は，すべての引数の型がTであるメソッドととらえることができる．

5)　クラスに対する手続きの宣言

上記のどの言語も，クラス定義において，そのクラスのインスタンスが持つ記憶領域（smalltalk，CLOSではスロット，C++ではメンバ変数という）を定義しなければならないが，C++では，それに加えて，そのクラスに対して定義される操作（smalltalk，CLOSではメソッド，C++ではメンバ関数という）は，すべてそのクラス定義で引数や返り値の型が宣言されていなければならない．これは，この情報がコンパイル時の型検査のために必要となるからである．逆に，この仕様のために，すでに定義されているクラスに対して，新たに操作を追加できないことになる．CLOSでは，すでに述べたように，メソッドは，クラスに対してではなく，クラスの組合わせに対して定義されるため，クラス定義中でのメソッド定義という概念はなくなっている．

6)　情報隠蔽

C++では，クラスや個々のメンバに対して，そのメンバを参照できる手続きの範囲を規定できるようにした．これによって，意図しない手続きからのデータの参照を防ぎ，モジュール性を高めている．smalltalkでは，基本的にはすべてのオブジェクトのスロットが参照できる．また，CLOSにおいては，メソッドは，もはや一つのクラスに属するという概念はないので，smalltalkと同様，すべてのオブジェクトのスロットが参照できる．

7)　メタクラス

smalltalkはすべてがオブジェクト指向で記述されているため，また，CLOSはこの上に他のObject指向機能を実現できる枠組を提供するため，各クラス自身がオブジェクトとして定義されている．これらの言語処理系では一般的に開発環境が提供され，定義されたクラスがその環境中で永続的に存在し，またインクリメ

ンタルにクラスを再定義したり，メソッドを追加することができる．このような場合には，クラスやそれに付随するメソッドやスロット自体をオブジェクトとして取り扱えることによって，処理系のかなりの部分をこの言語自体で記述でき，処理系の移植性も向上する．しかし，C＋＋のように，定義されたクラスはコンパイルされた実行モジュール内でのみ有効となるような言語では，クラス自体をオブジェクトとして記述できる利点はあまりない．

これに対して Uniscript では，次のような設計指針をとった．

1) 手続き型プログラミングとの親和性

Uniscript は TIPPLER のアプリケーション開発言語として新たに設計された言語であるが，従来の手続き型言語になれ親しんだプログラマが容易に Uniscript に移行できるよう，プログラムの制御構造は手続きに記述できるようにした．

2) 弱い型づけと強い型づけ

実行時の型検査によるエラーは，プロトタイピングにおいては有用であるが，本番アプリケーションでこのようなエラーが発生することは，アプリケーションとして設計ミスと考えられる．このため，Uniscript ではコンパイル時に型検査を行い，可能な限り実行時エラーを発生させないようにした．

3) 型とクラス

Uniscript では，数値や文字列といった基本データ型はクラスにはならない．(逆にクラスは型として扱うことができる．) 強い型づけがされた言語では，基本データ型に対するメソッドが定義できたとしても，コンパイル時に実行すべきメソッドが決定されるために，メソッドとして定義する利点は少ない．また，これによって，実行時にデータ構造自体にそのデータ型に関する情報を保持する必要がなくなるため，実行効率を向上させることができる．

4) メッセージ送信

Uniscript は，smalltalk と同様，標準的なメッセージ送信の枠組をもつ．ただし，従来の手続き型言語との親和性を保つために，通常の関数呼出しの枠組も提供する．

5) クラスに対する手続きの宣言

Uniscript では，クラスに対する手続き (メソッド) の定義は，クラス定義を変更せずに追加変更することができる．これにより，クラスライブラリなどの既存のクラスに対して，新たにメソッドを追加することができ，プログラムの部品化が容易に行える．ただし，コンパイル時の型検査ができるように，C のプロトタイプ宣言に相当する構文を用意した．

6) 情報隠蔽

Uniscript では，smalltalk と同様，すべてのオブジェクトのスロットを参照することができる．

7) メタクラス

Uniscript では，C＋＋と同様，定義されたクラスはコンパイルされた実行モジュール内でのみ有効となるため，クラス自体をオブジェクトとして記述できる利

点はあまりない．このため，クラス自身はオブジェクトとして実現されていない．

また，Uniscriptはこれ以外に次にのべるような機能を提供する．

8) プレゼンテーション

プレゼンテーションとは，オブジェクトの見せ方（型画面への表示方法）である．smalltalkでは，プレゼンテーションは，表示すべきオブジェクトとは異なる別個のオブジェクトであり，この二つのオブジェクトを結びつけ，制御するために，コントローラという別のオブジェクトが存在する．

Uniscriptでは，オブジェクトのプレゼンテーションは，そのオブジェクトが属するクラスに対して定義し，一つのオブジェクトに対して複数の見せ方を定義して表示させることができる．これをマルチプルプレゼンテーションと呼ぶ．これは，一般に世の中のものは多面的であり，他のものとのインタラクションはその一面を介して行われることのモデルとしてよく適合している．

Uniscriptにおけるプレゼンテーションは，インスタンスのスロットの内容を表示する（出力）だけでなく，キーボードやポインティングデバイスを用いてその表示を直接操作して，スロット値の変更やメソッドの起動といったインスタンスに対する入力も行うことができる．Lisp系言語ではSymbolics社のGenera[11]がこれに近いアプローチをとっているが，Generaにおいても，プレゼンテーションは別のオブジェクトである．

また，Uniscript独自の特徴として，プレゼンテーション定義の宣言的な記述がある．X Window SystemやWindowsでは，画面の定義は，プログラム中で手続き的に記述されるが，入力が行われた場合の処理や，データが更新された場合の再描画処理を個別に記述しなければならず，表示しているデータとプレゼンテーションの対応づけが見づらい．Uniscriptでは，この対応づけを明確に記述することによりプレゼンテーションを定義し，プログラムの可読性を向上させている．

9) デーモン

デーモンとは，特定の記憶領域に対する参照が発生した場合に実行される手続きのことで，AIプログラミングでは一般的な機能である．デーモンがあれば，記憶領域の内容の変更に伴うアプリケーション固有の副作用を実現するためのアクセス関数を用意する必要はなく，通常の記憶領域と同様に参照することができる．Uniscriptでは，デーモンもクラスに対して定義することにより，指定されたスロットへの書き込みが発生した場合に実行する手続きを記述する．

2. Uniscriptの概要

Uniscriptプログラムは，以下の構成要素を定義することで構成される．

- クラス
- メソッド
- フレーム
- メニュ
- デーモン

- ●関数
- ●変数
- ●定数
- ●列挙型
- ●列挙型書式

これらのうち，メソッド，スロット，フレームおよびメニューはクラスを指定して定義する．これによって，クラスを継承することにより，メソッド，スロット，フレームおよびメニューを再利用することができる．すなわち，あるクラスに対して定義されたフレームは，そのクラスおよびそのクラスを継承するクラスのインスタンスを表示することができる．

3. Uniscript における型

Uniscript は C＋＋と同様に強い型を持つ言語である．このため，型の整合性検査はコンパイル時に行い，実行時にはそれぞれのデータ構造や記憶領域の型に関する情報は保持しない．

Uniscript は次のような基本型を組み込み型として持つ．

- ●数値型
- ●時刻型
- ●文字列型
- ●論理型
- ●ファイル型
- ●アドレス型

また，次のような三種類の複合型を持つ．

- ●列型
- ●配列型
- ●ハッシュ表型

それぞれの複合型は単一の型ではなく，要素の型として他の型を指定することにより定義される型の集合である．要素の型としては，基本型，他の複合型，および次に述べるユーザ定義型を指定することができる．列型は，同種の要素の並びに対して，動的に要素の挿入削除が可能なデータ構造を提供する．

ユーザ定義型は次の二種類である．

- ●クラス型
- ●列挙型

前者は class 文によって定義されるクラス名を型名とする型の集合であり，後者は enumeration type 文によって定義される列挙名を型名とする型の集合である．

4. オブジェクト

Uniscript ではクラス定義で定義されたクラスのインスタンスと，列型，配列型，ハッシュ表型データがオブジェクトとして表現され，その同一性が保証される．たとえば

```
variable x: array [1.. 2] of number
variable y: array [1.. 2] of number
set y:=x
set x [1]:=10
```

では，変数yへの代入によって，変数xと変数yは同じ配列オブジェクトをさすため，最後の変数xの1番目の要素への代入の結果，変数yの1番目の要素も10となる．

それ以外のデータは即値で表現される．

```
variable x: number
variable y: number
set x:=0
set y:=x
set x:=1
```

では，最後の代入の結果，変数xの値は1となるが，変数yの値は0のまま変わらない．文字列は即値ではないが，部分文字列の変更が発生すると，コピーを作成し，もとの文字列を破壊しないという意味で，即値と同様に扱うことができる．たとえば

```
variable x: string
variable y: string
set x:="abc"
set y:=x
set x [2: 2]:="xyz"
```

では，変数yへの代入では文字列のコピーは発生せず，変数xと記憶領域を共有するが，最後の変数xの部分文字列への代入により文字列のコピー後，もとの文字列の記憶領域を変更するため，最終的には変数xの値は"axyzc"となり，変数yの値は"abc"のままである．

また，次のものはオブジェクトではなく，データ型としても記述することはできない．

- ●関数
- ●クラス
- ●メソッド
- ●スロット

CLOS処理系のように，一般に開発環境が提供され，定義されたクラスはその環境中で永続的に存在し，またインクリメンタルなクラスの再定義やメソッドの定義が許されるような言語では，クラスやそれに付随するメソッドやスロット自体を（メタ）オブジェクトとして取り扱えることによって，処理系のかなりの部分をCLOS自身で記述でき，処理系の移植性も向上させられる．しかし，UniscriptやC++のように，定義されたクラスはコンパイルされた実行モジュール中で閉じている言語では，クラスやそれに付随するメソッドやスロット自体をオブジェクトとして記述できる利点はあまりない．（ただし，C++ではメンバへのポインタをデータ型として取り扱うことはできる．）

5. クラス

Uniscriptではクラスを中心に定義が行われる．クラスとは，共通のデータ構造と手続きを持つオブジェクト（そのクラスのインスタンスと呼ぶ）の集まりを表現するものである．クラスに依存しない関数，大域変数，列挙型などの定義もあるが，クラス定義なしには実質的なプログラムは作りえない．

クラスは次の構文で定義する．

class　*class* 名（*superclass* 名 *1*　*superclass* 名 *2*...）
　slot 定義 1
　slot 定義 2
　...
end

この構文で，*class* 名という名前のクラスを定義するとともに，そのクラスの持つスロットを定義する．クラス名はアプリケーション(Uniscriptがコンパイルリンクされて作成される実行モジュール）単位でユニークでなければならない．また，このクラス定義により *class* 名は型名として用いることができ，この型の記憶領域は，このクラスおよびこのサブクラスのインスタンスを保持することができる．

スロットとは，このクラスの個々のインスタンスに割り合てられた記憶領域のことである．その記憶領域の内容は，インスタンスとスロット名を指定して参照および変更することができる．*slot* 定義 *n* は以下の構文で行われる．

　slot 名：型

あるいは

　slot 名：型：＝初期値

前者はスロット名とその型を記述し，後者はさらにそのスロットの初期値として格納される値を記述する．

スロットの値の参照は *object. slot* 名で行い，スロットの値の変更は次の構文で行う．

　set　*object. slot* 名：＝*newvalue*

そのクラスで定義されるスロットの型と，スーパークラスで定義されたスロットの型は一致しなければならず，この場合，一つのインスタンスに対して両者は同じ記憶領域を意味する．C＋＋では，基底クラスと導出クラスで同名のメンバ変数が定義された場合，異なる記憶領域を意味するが，これは，一つのメンバ変数参照に対して，実際にどちらの記憶領域が参照されているかがわかりづらく，プログラムの可読性，保守性を低下させるため，Uniscriptではこのような制限を導入した．

superclass 名 *n* でこのクラスで継承する別のクラス名を指定することにより，*superclass* 名 *n* で定義されたスロット，メソッド，フレーム，メニュー，デーモンが *class* 名で定義されたものと同様に使用することができる．この継承されるクラス *superclass* 名 *n* を継承するクラス *class* 名のスーパークラスといい，*class* 名を *superclass* 名 *n* のサブクラスという．この継承機能により，プログラムを部品化し，再利用することができる．

また，あるクラス型であると定義された変数またはスロットの値として，そのクラ

スのインスタンスだけでなく，そのクラスのサブクラスのインスタンスを代入することができる．また，どのクラス型のスロット/変数の値としてNULLをとることができる．これはどのインスタンスも指さない状態であり，初期値を指定しないクラス型のスロットや変数はNULLに初期化される．

新しいインスタンスの生成はnew_*class*名（）によって行う．この関数new_*class*名はクラス定義によって自動的に定義される．

6. メソッド

メソッドとはクラスに付随した手続きである．メソッドは次の構文で定義する．

```
method　method名 class名（引数1　引数2...　　）
  メソッド body
  ...
end
```

この定義により，*method*名を*class*名のメソッドとして定義する．メソッド*body*では通常の手続き型言語と同等の繰り返しや条件分岐などの制御構文を用いてプログラムを記述できる．また，この*class*名またはそのスーパークラスで定義されたスロットは，局所変数と同様にスロット名だけで参照および変更することができる．

メソッドの呼び出しはsend文を用いて，メソッドを呼び出すインスタンス，メソッド名，およびメソッドに渡す引数を指定する．つまり，smalltalkやC++と同様に古典的なメッセージ送信の枠組を持つ．呼び出されたメソッド内では，このインスタンスをselfという特別な変数で参照することができる．

CLOSやsmalltalkでは，すべての組込み型について，対応するクラスがあり，これらのクラスについても，ユーザ定義クラスと同じようにメソッドが定義できるが，Uniscriptでは，このようなクラスは存在せず，したがってこれらにメソッドを定義することはできない．これは，すでに述べたようにUniscriptでは，実行時には各データ構造は自身のデータ型についての情報を持っていないため，どのメソッドを実行するかの情報を得ることができないからである．また，組込み型についてはクラスの継承関係に相当するものがないため，コンパイル時に実行すべきメソッドは決定され，メソッドとして定義する利点はあまりない．

あるクラスで，そのスーパークラスで定義されたメソッドと同じ名前のメソッドを定義することができる．これをメソッドのオーバーロードと言う．オーバーロードができるためには，そのクラスで定義されるメソッドの引数と，スーパークラスで定義されたメソッドの引数の型および順序が一致しなければならない．C++では，一つのクラス（あるいはその規定クラス）に同名のメンバ関数で引数の型の組合せが異なる複数のメンバ関数を定義することができるが，Uniscriptではそれぞれのクラスについて，そのクラスおよびそのスーパークラスで定義されるメソッドの集合中に存在する同名のメソッドは同じ引数の型の組合せをもたなければならない．この制限は，C++と異なり，Uniscirptではそれぞれのクラスに対して定義されるメソッドの集合が，そのクラス定義には規定されていないため，コンパイル中にメソッド呼出し構文を解析する際にいずれのメソッドを呼び出すのかが特定できないからである．また，

C++でさえ，この仕様のためにコンパイル時にどちらのメソッドを呼び出すのか特定できない場合が発生し，プログラムの可読性，保守性を低下させるために，Uniscriptではこのような制限を導入した．

たとえば，

```
class C ()
end
class D (C)
end
method M C ()
  write stdout "C"
end
method M D ()
  write stdout "D"
end
function F ()
  variable c : C
  set c :=new_C ()
  send c M ()
  set c :=new_D ()
  send c M ()
end
```

この例では，関数F中で変数cに対してメソッドMを呼び出しているが，最初の呼び出しでは変数cの値はC型のインスタンスであるので，Cのメソッドが呼び出され，2番目の呼び出しでは変数cの値はD型のインスタンスであるので，Dのメソッドが呼び出される．結果として，標準出力に“C”そして“D”が出力される．

また，単にスーパークラスのメソッドをオーバーロードするだけでなく，スーパークラスのメソッドと組み合わせてメソッドを定義することができる．たとえば，先の定義で，クラスDのメソッドMを次のように書き直すと，

```
method M D ()
  write stdout "D"
  pass ()
end
```

pass文の実行により，スーパークラスの同名のメソッドを実行する．この結果，標準出力には“C”，“D”そして“C”が出力される．

7. 継　　承

クラスのスーパークラス/サブクラス関係はクラスを頂点とする有向グラフとなる．(辺の向きはサブクラスからスーパークラスへ向かう向きにつける．) この有向グラフをもちいて，各クラスについて次のような手続きでメソッド探索リストを作成する．

1) 頂点Cよりはじめる．

2) if 今いる頂点がメソッド探索リストに含まれていない then
この頂点をメソッド探索リストの最後に追加する．
else if この頂点がC then
終了
else
この頂点にやってきた辺を逆向きにたどった先の頂点に戻る．
endif

3) if 今いる頂点から出て行く辺で，まだたどっていない辺がある then
その辺をたどり新しい頂点に行く．まだたどっていない辺が複数ある場合は，それらの辺に対応するスーパークラスのうち，今いる頂点に対応するクラスの定義でより左に記述されている方を選び，2)を実行する．
else
この頂点にやってきた辺を逆向きにたどった先の頂点に戻り，3)を実行する．
endif

クラスCのインスタンスにメソッドMが呼び出された時には，このメソッド探索リストを前から探索して，最初にメソッドMが定義されているクラスのメソッドを実行する．フレームやメニューについても，このメソッド探索リストを用いて表示するフレームやメニューを決定する．

8. デーモン

デーモンはインスタンスのスロットが変更された時に呼び出される手続きである．デーモンは次の構文で定義する．

daemon　*daemon* 名 *class* 名 *mode*（*slot* 記述1　*slot* 記述2...）
　デーモン *body*
　...
end

slot 記述 *n* で指定されたスロットが変更されると，デーモン *body* で指定した処理が実行される．*mode* は before か after のいずれかで，デーモンが呼び出されるタイミングがそれぞれスロット値の変更前か変更後かを指定する．*slot* 記述 *n* は次の構文で指定する．

変数名：*slot* 名

mode が before ならば，変数名で指定された変数は，スロット *slot* 名の変更後のスロット値に束縛されて，デーモン *body* が実行され，*mode* が after ならば，スロット *slot* 名の変更前のスロット値に束縛されて，デーモン *body* が実行される．

デーモンもまた，定義されたクラスのサブクラスに継承される．すなわち，サブクラスのインスタンスに対して（デーモンが指定された）スロットの変更を行った場合も，デーモン *body* が実行される．

9. プレゼンテーション

Uniscriptでは，一つのインスタンスに対して複数の見せ方を定義して表示させる

ことができる．これをマルチプルプレゼンテーションと呼ぶ．これは，一般に世の中のものは多面的であり，他のものとのインタラクションはその一面を介して行われることのモデルとしてよく適合している．

たとえば，人をオブジェクトとして考えた時に，人事システムの中では，人の持つ属性のうち，社員コードや給与に焦点があてられるが，顧客管理システムの中では，資産や趣味などに焦点があてられる．

Uniscriptにおけるプレゼンテーションは，インスタンスのスロットの内容を表示する（出力）だけでなく，キーボードやポインティングデバイスを用いて，その表示を直接操作して，スロット値の変更やメソッドの起動といったインスタンスに対する入力も行うことができる．

このプレゼンテーションは画面上で独立した矩形領域（ウィンドウ）を形成するフレームと，そのフレーム中に配置されたデータを表示する複数のアイテムから構成される．(このほかにメニューがあるが，メニューはボタンしかないフレームと同等と考えてよい.)フレームはクラスを指定して定義される．定義されたフレームは，このクラスのインスタンスおよびこのクラスを継承するクラス（サブクラス）のインスタンスを表示することができる．MVC (Model-View-Controller) モデルに基づいて設計されたsmalltalkでは，モデル(データ構造)，ビュー(プレゼンテーション)，コントローラ(制御)をそれぞれ別のクラスとして実現しているが，Uniscriptではモデルをクラスで表現し，ビューおよびコントロールをそれぞれフレームおよびメソッドとして，クラスに対して定義する．これによってすべてのビューおよびコントロールはどのモデルを表示し制御するものかを明確に記述することができる．

こうして定義されたフレームは，インスタンスとフレーム定義で指定されたフレーム名を指定して表示される．あるインスタンスをあるフレームで表示しているとき，このフレームはこのインスタンスに対応づけられているという．アイテムは，そのフレームが対応づけられたインスタンスのスロットの内容を表示し，また，ポインティングデバイスやキーボードによってインスタンスを操作するものである．

アイテムとして以下のものを提供する．

- textfield　文字列型データ表示/入力用
- numericfield　数値型データ表示/入力用
- timefield　時刻型データ表示/入力用
- selectionfield　列挙型データ表示/入力用
- abbrevfield　任意の式を表示する
- message　複数行にわたる文字列表示用
- bitmap　画像データ表示用
- filler　余白用
- scrollinglist　列型および配列型データ表示
- choicebox　列挙型および集合型表示/入力
- checkbox　列挙型および合型表示/入力
- methodbutton　メソッド起動用
- alertbutton　メソッド起動用（警告つき）

- menubutton　メニュー表示用
- menubar　メニューボタン表示用
- spreadsheet　表を表示する．
- picture　自由図形を表示する．
- xygraph　折れ線グラフ，棒グラフ，積み上げ棒グラフを表示する．
- ole OLE　オブジェクト表示用

フレーム定義において，各アイテムは表示すべきデータと対応付けられる．(正確には，各アイテムは，表示すべきデータが格納されている場所と対応付けられる．これによって，一度の記述でそのデータを表示し，また，アイテムからの入力によってその格納場所に納められたデータを更新することができる．)

フレームを含めて，各アイテムは，アイテムの表示方法や入力が行われた場合の振る舞いを詳細に規定するための属性と呼ばれる項目を複数個指定することができる．しかし，属性を指定しない場合でも，TIPPLERの実行時ライブラリはもっとも一般的と考えられる設定でアイテムを表示する．アプリケーションにとって，この設定が適切でない場合には，適当な属性を指定することにより，きめの細かい設定が可能となる．

一般に属性の指定は（属性）値を伴う．指定できる属性値の範囲は属性毎に規定されている．Uniscriptのデータ型を属性値として持つ属性については，その型の式を指定することができる．システムは実行時にこのフレームが表示される際に，この式を計算（評価）し，属性値として用いる．この式中では，メソッドと同様に，表示するフレームに対応付けられたインスタンスのスロットを参照することができる．これによって，一つのフレーム定義からも，様々なバリエーションを持ったフレームを表示することができる．

プレゼンテーション（フレームおよびアイテム）は宣言的に記述する．次に単純なクラスCの定義とそれに対するフレームFの定義を示す．

```
class C ()
  s: string
end
frame F C
  textfield: s
end
```

この記述により，クラスCの文字列型スロットsがフレームFのtextfieldアイテムで表示され，また，このtextfieldアイテムからの入力により，このフレームが対応付けられているインスタンスのスロットsを更新する（図1）．

このように，アイテムとそれが表示するデータの関連を直接的に記述でき，また，表示すべきデータが更新された場合にも，明示的にアイテムに対して再描画を指示する必要はない．TIPPLERの実行時ライブラリが自動的にアイテムの表示を更新する．

プレゼンテーション表示を手続き的に表現する言語では，入力が行われた場合の処理や，データが更新された場合の再描画処理を個別に記述しなければならない．この方法では，入力が行われた場合に，表示しているデータの格納場所と異なる場所に入

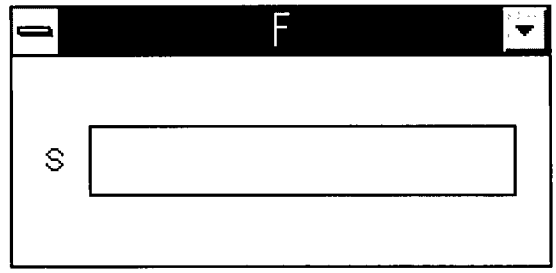


図 1　宣言的記述によるフレーム

力データを書き込むことが可能であるが，通常は，表示しているデータを更新するのが一般的であり，これに対する手続き的記述は冗長で，アイテムと表示されているデータの関連がわかりづらい．ただし，別の格納場所にデータを書き込む必要がある場合もあり，Uniscript では属性やデーモンを指定することにより実現可能である．

smalltalk や Lisp Machine のウィンドウシステムでは，ウィンドウは表示すべきデータとは独立したオブジェクトである．これらのシステムでは，ウィンドウを変更せずに，表示させるデータだけを変更することができるが，この方法では，データとプレゼンテーションの関連が希薄となり，利用者は，自分が実際にどのデータを操作しているのかわかりづらくなる．Uniscript では，このようなデータとプレゼンテーションを分離して扱えないようにして，データとプレゼンテーションの関連をより直接的に表現できるようにしている．

上記の frame 文で定義されたフレームは Uniscript の手続き（メソッドおよび関数）中で expose 文あるいは activate 文を呼ぶことにより，画面に表示することができる．

activate　*object*　フレーム名
expose　*object*　フレーム名

プレゼンテーションはオブジェクトではない．つまり，フレームやアイテムは固有の識別子を持たない．特定のフレームに対して操作（たとえば，そのフレームを消去）する場合には，上記のように，そのフレームが対応付けられているインスタンスとフレーム定義で定義されたフレーム名によってフレームを指定する．フレームは，フレームが対応付けられているインスタンスとフレーム名の組に対して一意に定まる．

同様にして，表示されたフレームの消去は deexpose 文あるいは deactivate 文によって行う．

activate　*object*　フレーム名
expose　*object*　フレーム名

10. 複合フレーム

フレーム定義についても，手続きと同じように，共通に使われるアイテムを別フレームとして定義しておき，これを別のフレームで取り込む複合フレーム機能を提供する．たとえば

```
class C ()
  s: text
end
frame F C
  methodbutton: method () quit end
end
frame G C
  textfield: s
  panel: F
end
```

このプログラムではフレーム G 中に textfield アイテムと別のフレーム F が表示される（図 2）.

フレーム F はボタンアイテムからなるフレームなので，上のプログラムのフレーム G は次のプログラムで表示されるフレーム G と等しい.

```
frame G C
  textfield: s
  methodbutton: method () quit end
end
```

複合フレーム機能を用いることにより，フレーム間で共通なアイテム定義を再利用可能にし，生産性/保守性を向上させることができる.

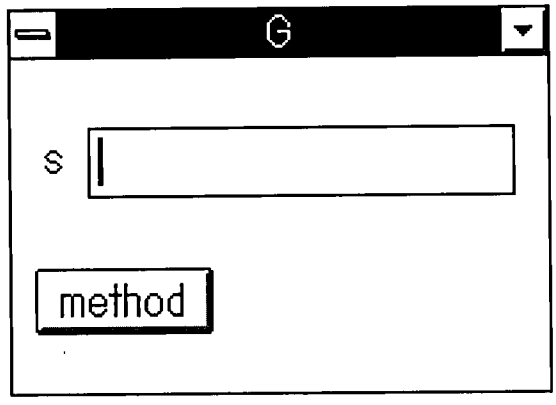


図 2　複合フレーム

11. 動的パネル参照

インスタンスをフレームで表示する場合，ある条件によって表示させるアイテムの種類や数を変更したいことがある．アイテムの位置や大きさについては属性で制御できるが，アイテムの種類や数はフレーム定義で宣言的に記述されているため，ここまでで述べた枠組では変更できない．これを実現するために，フレーム中に別のインスタンスのフレームを取り込む機能を提供する．たとえば

```
class C ()
  s: text
  d: D
end
class D ()
  t: number
end
frame F C
  textfield: s
  panel: d using G
end
frame G D
  numericfield: t
end
```

このプログラムでは，クラスCのフレームF中にスロットsを表示するtextfieldアイテムとスロットdを表示するフレームGが表示される（図3）．

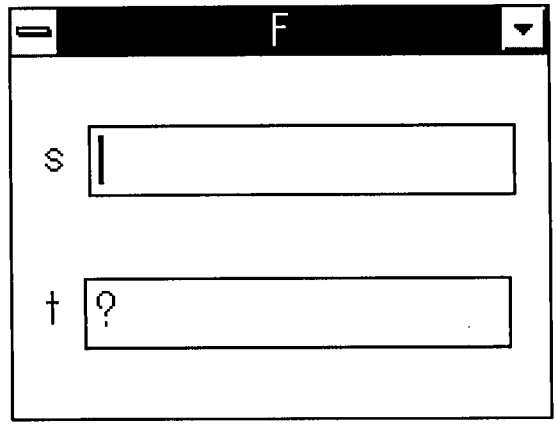


図 3　動的パネル参照

クラスDであるスロットdのフレームGとはスロットtを表示するnumericfieldアイテムからなるので，上のプログラムで表示されるフレームFは次のプログラムで定義されるフレームFFと本質的に同じである（図4）．

```
frame FF C
  textfield: s
  numericfield: d.t
end
```

ここでd.tはスロットdの値であるクラスDのインスタンスのスロットtを意味する．

この機能を用いると，次のようなプログラムによって，実行時にフレームに含まれ

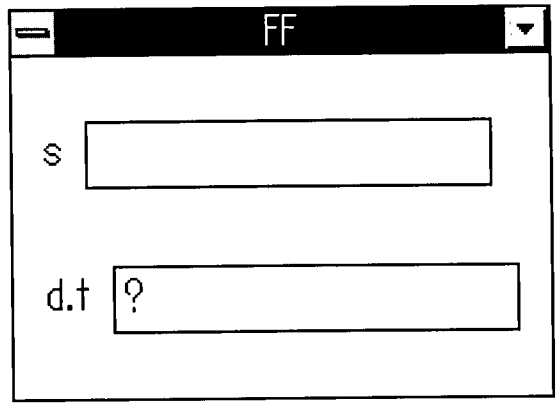


図 4　動的パネル参照と同等のフレーム

るアイテムの数を変更することができる.

```
class C
end
class D (C)
  s: string
  c: C:=new_C ()
end
frame F C
end
frame F D
  textfield: s
  panel: c using F
end
frame G D
  methodbutton: M
  panel: self using F
end
method M D ()
  variable d: D:=new_D ()
  set d.c:=c
  set c:=d
end
start D using G
```

フレームG(図5)のボタンMを押すことにより，スロットcの値がクラスCのインスタンスからクラスDのインスタンスに変更され,それにともなってフレームGにtextfieldが一つ追加される（図6).

このプログラムではボタンを押すたびにtextfieldが一つ増えていくことになる.も

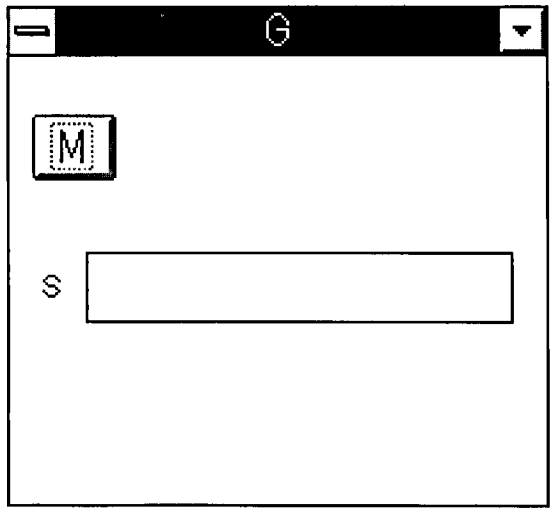


図 5　フレームの初期状態

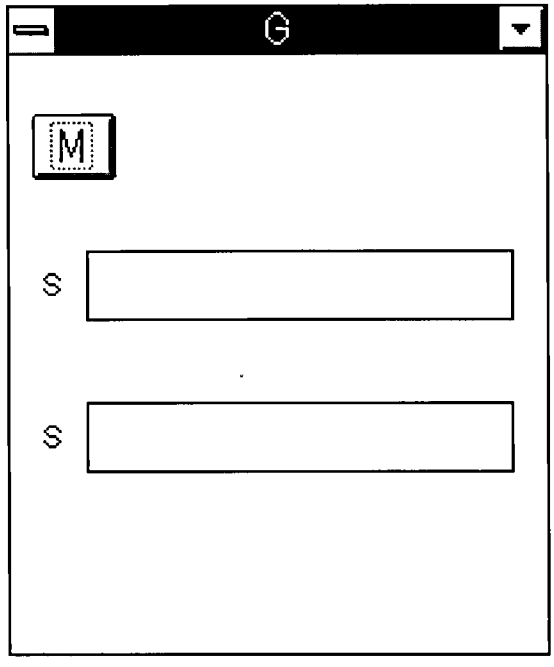


図 6　ボタン M を一度押した後のフレー G

し，新たに追加されるアイテムに属性が指定さえれているならば，その属性の評価は，アイテムが追加された時に行われる．

12. 今 後 の 課 題

現在の Uniscript の言語処理系はコンパイラ，インタプリタともに一括型であり，ア

プリケーション実行中にインクリメンタルに再定義されたコードを追加することはできない．しかし，TIPPLERという統合開発環境での位置付けとしては，より快適な開発環境を提供するために，このような機能も実現していく必要があるであろう．このためには，クラス自身をオブジェクトとして取り扱えることができるようにすることにより，Uniscript自身で開発環境の大部分を記述することができ，開発環境の移植性が向上するとともに，アプリケーション自身がプログラムの変更を記述できるようになる．

宣言的なフレーム定義およびフレーム定義のオーバーロードはTIPPLERの特徴であるが，さらに柔軟なフレーム定義を可能にするために，フレーム定義に対して次のような拡張が考えられる．

1) メソッドと同様に仮引数が記述でき，プレゼンテーション表示の際にデータを実引数としてわたす．

```
frame F C (x : number, y : number)
  textfield : s {xpositionpixels : x
                 ypositionpixels : y}
end
method M C()
  expose self using F (10, 20)
end
```

この例では，textfieldの表示される位置を仮引数で指定できるようにし，このフレームを表示されるexpose文でその位置を（10，20）に指定している．

2) フレーム定義中に条件分岐や同種のアイテムを複数表示するための反復の制御構文を記述できるようにする．

```
class C()
  a : array [1.. 10] of string
end
frame F C
  variable i : number
  for i from 1 to 10
    textfield : a[i]
  end
end
```

この例では，長さ10の配列の文字列型要素に対応するtextfieldを表示する．

ただし，これらの拡張は，一方では宣言的記述の利点を少くするため，導入には十分な検討が必要である．また，前者の拡張では，abbrevfieldのように任意の式が記述できるアイテムに対して

```
frame F C (x : number, y : number)
  abbrevfield : x+y
end
method M C()
```

```
    expose self using F (10, 20)
end
```

とフレームを定義した場合に，expose 文はフレームを表示すると制御をすぐ呼び出し側 M に戻すので，x や y のエクステントをどのように取り扱うかという問題が発生する．また後者の拡張でも，局所変数を導入すれば，仮引数の場合と同様に，局所変数のエクステントをうまく定義しなければならない．

13. おわりに

TIPPLER におけるアプリケーション開発言語 Uniscript の言語モデルについて紹介した．Uniscript の言語設計および処理系開発を通じて，オブジェクト指向言語および GUI 定義言語に要求される機能および問題点が明確できたと思われる．これを今後の計算機言語処理系設計においても，活かすことができると考えている．

参考文献

[1] 所真理雄，松岡聡，垂水浩幸編，“オブジェクト指向コンピューティング”，岩波書店，1993.
[2] A. Goldberg, D. Robson, “Smalltalk-80 : The Language and its Implementation”, Addison Wesley, 1983.
[3] 梅村恭司，“Smalltalk-80 入門”，サイエンス社，1986.
[4] G. Steele, “Common Lisp : The Language”, 2 nd Edition, Digital Press, 1990.
[5] D. G. Bobrow, et al., “The Common Lisp Object System Specification”, ANSI X 3 J 13 88-002 R, 1988.
[6] G. Kiczales, et al., “The Art of the Metaobject Protocol”, The MIT Press, 1991.
[7] A. Paepcke, ed., “Object-oriented Programming : The CLOS Perspective”, The MIT Press, 1993.
[8] B. Stroustrup, “The C++Programming Language”, Addison Wesley, 1986.
[9] M. Ellis, B. Stroustrup, “The Annoted C++Reference Manual”, Addison Wesley, 1990.
[10] 日本ユニシス，“Tippler ユーザガイド”，1994.
[11] J. H. Walker, et al., “The Symbolics Genera Programming Environment”, IEEE Software, 1987 November pp. 36～45.

執筆者紹介 川 辺 治 之 (Haruyuki Kawabe)

1961 年生．1985 年東京大学理学部数学科卒業．同年日本ユニシス(株)入社．知識システム部所属，Lisp マシン KS-300 シリーズの日本語機能開発・保守，1100/2200 上の Lisp 処理系の開発，KEE/U-6000 の日本語化，TIPPLER の開発・保守に従事．現在，開発三課所属．

TIPPLERのプレゼンテーションモデル

The Presentation Model for TIPPLER

伊勢本 勝一

要 約 TIPPLERのプレゼンテーションモデルとは，ディスプレイ上のウィンドウや帳票はオブジェクトを表示するための枠組に過ぎず，クラスとしては定義しないとする，GUIを取り込むために拡張されたオブジェクトモデルである．プレゼンテーションモデルは，宣言的記述と表示の自動更新機能という二つの大きな特長により，TIPPLERプログラミングの生産性を高めている．その理由は，SmalltalkのMVCモデルと比較すると明らかである．
本稿では，TIPPLERにおけるプレゼンテーションモデルについて，MVCモデルとの比較を中心に説明する．

Abstract With windows and documents on the display screen being no more than frameworks for displaying objects, the presentation model for TIPPLER is positioned as an object model extended to incorporate a GUI, which does not define those objects as any classes. With the help of its two major features: declarative representation and automatic update of displays, the presentation model helps improve productivity in TIPPLER programming. The reason for this are obvious when the model is compared with the MVC model of Smalltalk.

This paper addresses the presentation model for TIPPLER, focusing its comparisons with the MVC model.

1. はじめに

システム化とは，現実世界のモノや事象を計算機上に投影する作業であり，その投影に対し何らかのモデルが使用される．80年代中頃，SmalltalkやC++などオブジェクト指向プログラミング言語がベースとしたオブジェクトモデルは，それまでのモデルの問題点のいくつかを克服したものであり，開発生産性の向上が期待された．

一方，グラフィカルユーザインタフェース（GUI）は，初心者にもとっつきやすく，計算機の利用者の裾野を広げるための優れたユーザインタフェースとして，広く普及した．しかし，GUIプログラミングを行うためには，膨大な手続き呼び出しをはじめとする多くの問題点があり，熟練者でなければ開発が難しいといわれていた．

当時，オブジェクト指向プログラミング言語にとって，GUIをいかに取り込むかが普及のための課題であったのではないかと考える．Smalltalk 80はMVC（Model-View-Controller）モデルという概念によりGUIプログラミングをモデル化し取り込んだ．一方，C++のベンダはGUIビルダの提供によってその課題を解決しようとした．

これに対し，TIPPLERは，オブジェクト指向プログラミング言語Uniscriptの文法に，GUIを完全に取り込んだ形で登場した．GUIを取り込むために導入したのが，

TIPPLERの設計とともに生まれたプレゼンテーションモデルである．

本稿では，TIPPLERにおけるプレゼンテーションモデルについて，MVCモデルとの比較を中心に説明する．

2章では，TIPPLERのプレゼンテーションモデルについて述べ，3章では，Smalltalkのプレゼンテーションモデルについて言及する．最後に両者を比較し，今後の課題について述べる．

2. プレゼンテーションモデル

プレゼンテーションモデルとは，問題対象における本質を表すモノおよび事象のみをオブジェクトと捉え，ディスプレイ上のウィンドウや帳票は，オブジェクトの側面を表示しているに過ぎないとするモデルである．

ディスプレイ上のウィンドウや帳票などはオブジェクトの一つの見方（オブジェクトの側からいえば，見せ方）であり，これをプレゼンテーションとよぶ．

商取り引きの例で説明する．商取り引きという事象が発生したとき，まず伝票にその内容が記され，その後伝票から仕訳帳に転記される．会計係は，商取り引きを伝票で見たり仕訳帳で見たりする．この行為をプレゼンテーションモデルで説明すると，「会計係は商取り引きという事象を，伝票や仕訳帳という枠組を使って見ている」といえる（図1）．会計係は，決して伝票や仕訳帳そのものを見たいわけではなく，データである商取り引きの内容が見たいのである．伝票や仕訳帳は，商取り引きというオブジェクトに対するプレゼンテーションである．

TIPPLERのプレゼンテーションモデルには次の特長がある．

1) 宣言的記述
2) マルチプル・プレゼンテーション
3) 継承およびオーバーロード
4) モジュール化と複合
5) アップデート・リフレッシュプロトコル

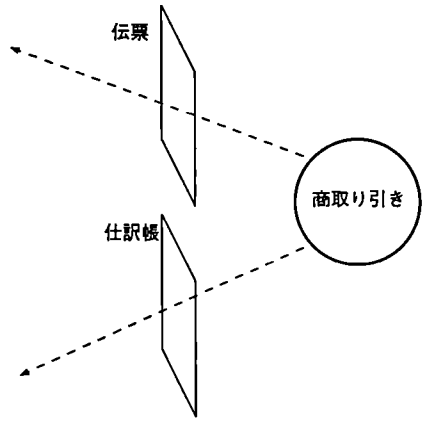


図1 プレゼンテーションモデルの例

以下，個々の特長について説明する．

2.1 宣言的記述

プレゼンテーションモデルは，オブジェクトのユーザインタフェースを前提とし，オブジェクトモデルの中にGUIを取り込んでいる．

オブジェクトを見る場合，通常その全てを見ることはあまりない．人間が一度に把握できる情報量はそれほど多くはなく，見る必要のない属性も多い．したがって，プレゼンテーションモデルでは，どの属性を見るか，すなわち「何を」見るかを最大関心事と捉え，極力手続き的な要素を排除しようとした．その結果，宣言的記述によるプレゼンテーション定義を可能としている．それまでのGUIプログラミングは膨大な手続き呼び出しが必要であったが，TIPPLERは，プレゼンテーションの宣言的記述により，GUIプログラミングの生産性を大きく向上させた．当初，TIPPLERが強力なGUIビルダを持たなかったにも関わらず普及したのは，プレゼンテーションの宣言的記述による高生産性が最も影響していると考える．

TIPPLERでは，プレゼンテーションの宣言的記述を可能にするため，データのさまざまな表示方法を部品（アイテムという）という形で提供している．数値データを表示するための「数値フィールド」やリスト構造のデータを表示するための「スクローリングリスト」などがそれである．図2のウィンドウをTIPPLERのプレゼンテーションとして記述する場合，TIPPLERではどのような記述になるか説明する．frame（フレーム）はプレゼンテーションの定義を表しており，ディスプレイ上ではウィンドウに対応する．モデルとの対応が分るようにクラス定義も掲載しておく（図3）．

図2　TIPPLERのフレームの例

```
class 従業員 ()
   名前      : string
   生年月日 : time
   所属部署 : string
end

frame 個人情報 従業員
   textfield : 名前
   timefield : 生年月日
end
```

図3　コーディング例（1）

このように，クラスのスロットを，アイテム（ここではtextfieldやtimefield）に対

応づけているだけである．

宣言的記述という形でGUIを取り込んでいるため，プラットフォーム間の移植性が高く，移植した場合の他アプリケーションとのユーザインタフェースの親和性が高いといえる．また，簡単にウィンドウを表示するプログラムが書けるため，プロトタイピングが容易である．

2.2 マルチプル・プレゼンテーション

モノの見方は，人や場面によってさまざまであるため，一つのオブジェクト（クラス）に対して，複数のプレゼンテーションを定義できる必要がある．この機能をマルチプル・プレゼンテーションとよぶ．数値データを表で見たり，グラフで見たりすることはマルチプル・プレゼンテーションの一例である．

2.3 プレゼンテーションの継承とオーバーロード

プレゼンテーションは，操作（メソッド）同様オブジェクトにカプセル化され，継承される．これにより，プレゼンテーションにおけるオーバーロードや動的束縛（多態性）を可能とする．

```
class 社員 ()
   名前   : string
   入社日 : date
end

frame 個人情報 社員
   textfield : 名前
   timefield : 入社日
end

class 課長 (社員)
   部下達 : sequence of 社員
end

frame 個人情報 課長
   panel
      textfield : 名前
      timefield : 入社日
   end
   scrollinglist : 部下達
end
```

図4 コーディング例（2）

図4のように定義を行った場合，個人情報というフレームを表示させる文があったとき，表示されるフレームはインスタンスに応じて変化する．すなわち，社員に対応するインスタンスの場合は名前と入社日しか表示されないが，課長に対応するインスタンスの場合は部下の一覧も表示される．

同じメソッドを呼び出した場合でも，インスタンスに応じて異なった振舞いをする機能を動的束縛というが，プレゼンテーションにも同じ機能がある．

2.4 プレゼンテーションのモジュール化と複合

プレゼンテーションは，メソッド同様モジュール化することができる．具体的には，

一つのフレームを複数の部分に分割して定義することができる．分割したフレームはさらに複数の部分に分割することができるため，階層構造をなす．言い換えれば，あるフレームの定義の中で他のフレーム定義を引用できるのである．これにはパネル(panel) というアイテムを用いて行う．

```
frame F C
   panel : F1
   panel : F2
   panel : F3
end

frame F1 C
   panel : F11
   panel : F12
end

frame F2 C
   :
end

frame F3 C
   :
end
```

図 5　コーディング例（3）

図5のような定義がある場合に，フレーム「F」を表示させたとき，図6のようになる．さらに，別のフレーム定義でF11だけを引用するとか，F2とF3を引用するなどといったことが可能である．すなわち，あるフレーム定義を複数の別のフレーム定義で共有することができる．

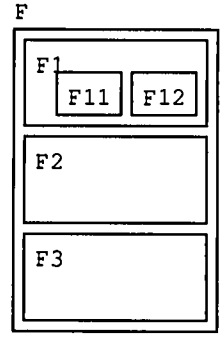


図 6　プレゼンテーションのモジュール化

引用するのは，同一クラスのものに限らない．複合オブジェクト（他のオブジェクトを属性値とする関係）のフレーム定義も，引用する事ができる．言い換えれば，複合オブジェクトを値に持つようなスロットを表示できるということである．これを図7のTIPPLERのコーディング例で説明する．

複合オブジェクトを表示する場合もパネルを使用する．「C」のインスタンスはスロット「x」により，「X」のインスタンスを参照している．フレーム「F」でスロット「x」をパネルにより表示しているのである．この場合，参照している「X」のインスタンスの内容が変化すると，フレーム「F」の中に表示されている「P」も変化する．

これを図示したものが図8である．

このように複雑な構造を持つスロットであっても，一つのアイテムとして表示でき，フレームのモジュール化がメソッド同様に自在にできるのである．

```
class X ()
end

frame P X
end

class C ()
   x : X
end

frame F C
   :
   panel : x using P
end
```

図 7　コーディング例（4）

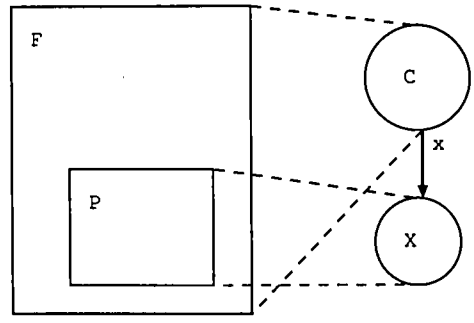


図 8　複合プレゼンテーション

2.5　アップデートリフレッシュプロトコル

プレゼンテーションは，オブジェクトの側面を，ウィンドウなどの枠組をとおしてユーザに見せるためのものである．したがってプレゼンテーションモデルでは，オブジェクトの更新があった場合も，手続きを呼び出すことなしに表示の整合性を保つ必要があるとする．これを，アップデートリフレッシュプロトコルとよぶ．

ユーザからのディスプレイ上のウィンドウ（フレーム）に対する入力操作は，対応するオブジェクト（インスタンス）を直接操作しているという考えである．フレームはインスタンスのデータを見せる枠組であるため，表示の更新を行うためのプログラムは不要である．従来のGUIプログラミングでは，表示の整合性を保つことが最も煩雑であったが，TIPPLERのプレゼンテーションモデルはシステム内部でこれを実現

することにより，開発者の負担を大幅に軽減した．

3. MVCモデル

MVCモデルとは，アプリケーションプログラムをモデル，ビュー，コントローラという三つのカテゴリに分類する概念および機構であり，主にSmalltalkで使われてきた．モデル（Model）・ビュー（View）・コントローラ（Controller）という3種類のオブジェクトとその役割を次のように定めている．

モデル（Model）	：問題対象としてのデータとそのデータに対する操作を行うオブジェクト．ユーザインタフェースに対する依存度は低い．
ビュー（View）	：ディスプレイを通して，モデルからユーザへ情報を提供するオブジェクト．表示対象となるモデルをインスタンス変数として保持する．
コントローラ（Controller）	：ユーザからの入力（マウスやキーボード）を解釈し，モデルやビューに適切な調整を施すオブジェクト．

MVCモデルは，モデルからユーザインタフェースに関連することを排除するという考え方を基本に持つ．すなわち，ユーザインタフェースをアプリケーションプログラムから分離しようとしている．

モデルはデータベースに格納されるべきデータを表現したオブジェクトであり，そのデータに対する操作がカプセル化される．ビューは，モデルのインスタンスをインスタンス変数として参照し，それをどのように表示するかを手続きとして表現する．コントローラは，ユーザからの入力を解釈/制御するための手続きをカプセル化している．コントローラはモデルおよびビューのインスタンスを参照する．3者間の関係は次の図9のようになる．

図9中の実線は明示的な参照を表しており，点線は「依存」を表している．依存とは，モデルの変更をビューに通知するために用意された弱い参照である．

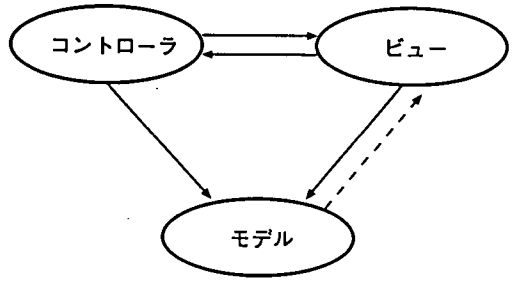


図9　モデル・ビュー・コントローラの関係

3.1 表示の更新メカニズム

MVCモデルでは，表示の更新は次のように行われる．

アクティブコントローラ（ディスプレイ上にウィンドウが複数表示されていても，ユ

ーザが操作できるウィンドウはひとつだけである．MVC モデルでは，操作できるビューに対応するコントローラのことをアクティブコントローラと呼ぶ．)が，ユーザからの入力を拾いそれを解釈すると，参照するモデルに入力に応じた変更を施し，それをモデルに通知する(この通知についてはプロトコルが定められており，Smalltalk では changed というメッセージを送信することにより行う)．変更の通知を受け取ったモデルは，従属するビューのインスタンス（そのモデルを参照しているビューのインス

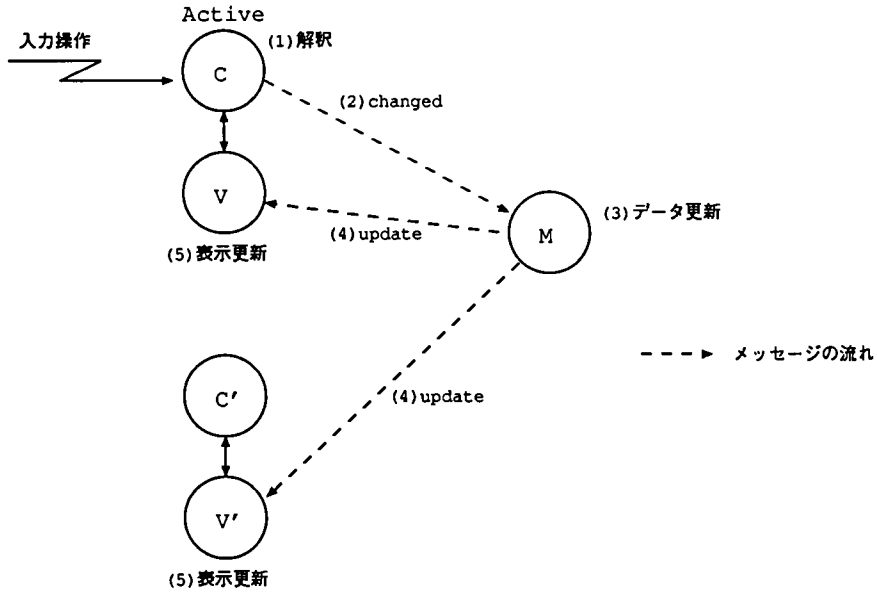


図 10　MVC モデルの表示更新

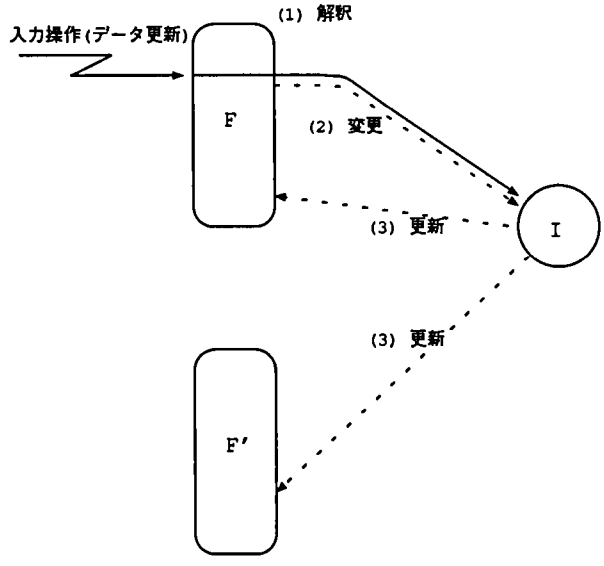


図 11　プレゼンテーションモデルの表示更新

タンス）すべてに対して，変更があったことを放送（ブロードキャスト）する（Smalltalkではupdateというメッセージ送信にて行う）．ビューは，変更があった旨のメッセージを受け取るとモデルを参照し自身の表示を更新する．

これを図で表すと図10のようになる．番号は，Smalltalkのプログラムで記述が必要な部分とその実行順序である．

これに対し，プレゼンテーションモデルのアップデートリフレッシュプロトコルを図11のように表すことができる．

図11中の点線および番号とその説明は，TIPPLERの場合のシステム内部の手続き呼び出しの様子を表している．このように，内部的にはMVCモデルと同様の更新メカニズムを利用しているが，システムに隠蔽することにより，プレゼンテーションへの入力操作がオブジェクトへの直接操作と同じ意味を持つのである．さらに，表示の整合性をとるための手続き呼び出しが不要であるため，開発者の負担を大きく軽減している．

3.2 パネルとサブビュー

プレゼンテーションのモジュール化はパネルというアイテムを用いて行う事は前述したが，MVCモデルでもモジュール化を実現する方法を提供している．

MVCモデルでは表示を手続きにより行うため，モジュール化は手続きのそれと同様に考える．

TIPPLERのパネルに相当するものをサブビューと呼び，ビューのインスタンスはサブビューのリストを保持している．サブビューが従属するビューのことをスーパービューといい，スーパービューを持たないビューをトップビューという．したがって，ビューはトップビューをルートとする木構造を形成する．2.4節で述べたように，TIPPLERのプレゼンテーションモデルではパネルというアイテムを用いた宣言的記述によりフレームの引用を行うことができるが，MVCモデルにおけるスーパービュー・サブビューの関係はすべて手続きによりつくらなければならない．

ビューを表示する場合は，トップビューに対して表示のための手続きを呼び出すことにより行う．ビューは表示のためのメッセージを受信すると，サブビューのリストの要素全てに対して，表示のためのメッセージを送信する．つまり，表示の手続きがトップビューから下位に向かって再帰的に呼び出されるのである．

コントローラにおいても再帰的な手続きが必要である．ユーザからの入力があると，一番適切なコントローラが見つかるまで，トップビューを参照するコントローラから下位に向かって制御が移される．適切なコントローラが見つけ出されるとそのコントローラが動き出し，処理が終り制御が必要なくなると，上位のコントローラに制御が返される．このようなプロトコルを用いているため，ビューは木構造でなければならず，表示の命令はトップビューに対してのみ行えるだけである．

これに対し，TIPPLERのプレゼンテーションモデルでは，フレームが木構造である必要はなく，従属関係を強く定義する必要がないため，どのフレームに対しても表示の命令を下すことができる．

3.3 イベントハンドラ

イベントハンドラとは，ユーザからの入力を制御/解釈するための手続きのことであ

る．MVCモデルではコントローラとしてこれを記述しなければならない．しかし，記述することによりイベントを自由に拾うことができるため，どんなインタフェースも構築可能である．TIPPLERの場合はこれを記述することができないため，たとえばテキストエディタのようなアプリケーションを開発することは困難である．

4. 二つのモデルの比較

前節で述べたことをまとめると表1のようになる．

プレゼンテーションモデルは，

- 宣言的記述によるフレーム定義
- アップデートリフレッシュプロトコル

という二つの特長により，アプリケーション開発の生産性を向上させた．が，一方で低レベルの記述ができないことにより，不向きなアプリケーションが存在するのも確かである．これは，プレゼンテーションモデルとMVCモデルが，それぞれ前提においたアプリケーションの違いによるものである．TIPPLERはビジネスアプリケーションを前提としていたため，

- 生産性のよさ
- 開発工程がシームレスにつながること

などを重要視し，プレゼンテーションモデルを採用している．

表1　プレゼンテーションモデルとMVCモデルの比較

	プレゼンテーションモデル(TIPPLER)	MVCモデル(Smalltalk)
フレーム(またはビュー)の定義	宣言的記述	手続き的記述
変更の通知	不要	updateメッセージ送信
フレーム(またはビュー)の従属関係	閉路のない有向グラフ	有向木
イベントハンドラ	不要(記述できない)	コントローラとして記述

```
class 社員 ()
   名前    : string
   入社日 : date
end

frame 個人情報 社員
   textfield : 名前
   timefield : 入社日
end

class 課長 (社員)
   部下達 : sequence of 社員
end

frame 個人情報 課長
   passpanel
   scrollinglist : 部下達
end
```

図 12　コーディング例（5）

5. TIPPLERの今後の課題

プレゼンテーションモデルという観点から，TIPPLERの今後の課題をまとめた．

1)　フレーム継承の再利用方法の洗練

フレームはクラスにカプセル化され継承されるが，オーバーロードに柔軟性がない．追加定義する場合，メソッドにはパス文により容易に行うことができるが，フレームにはこれに相当する機能が用意されていない．2.3節で示した例で，クラス「課長」のフレーム「個人情報」の中で「名前」と「入社日」を表示している部分は，親クラス「社員」に定義された「個人情報」と同じである．これを図12のように記述できるとよい．

2)　複合オブジェクトの完全な分離

スプレッドシートやグラフなどのアイテムは，インスタンスのリストを表示するためのものであるが，上位のフレーム定義で下位の(リストの要素に相当する)インスタンスのスロット名を記述しなければならない(図13)．

```
frame 部下の表 課長
   spreadsheet
      block
          abbrevcell : "名前"
          abbrevcell : "入社日"
      end
      block : 部下達
          textcell : 名前
          timecell : 入社日
      end
   end
end
```

図 13　コーディング例（6）

この場合，図14のように記述できれば隠蔽性が損なわれない．

```
frame 部下の表 課長
   spreadsheet
      block
          abbrevcell : "名前"
          abbrevcell : "入社日"
      end
      block : 部下達 using 行型情報
   end
end

block 行型情報 社員
   textcell : 名前
   timecell : 入社日
end
```

図 14　コーディング例（7）

3) 帳票への拡張

TIPPLERの帳票機能としてPUIというパッケージが提供されているが，帳票もプレゼンテーションの一種であるため，フレームと同様帳票も宣言的記述を可能とし，言語に取り込めればよい．

6. おわりに

TIPPLERがプレゼンテーションモデルの採用によりGUIを言語の中に取り込んだことは，APIやツールの提供だけがプログラミングの生産性を高めるものではないことを示唆していると思う．これからのプラットフォームソフトウェアやミドルソフトウェアの開発において，大いに参考となる部分があるのではないだろうか．

本稿により，プレゼンテーションモデルの長所や短所を理解し，今後の開発に役立てて頂ければ幸いである．

参考文献

[1] 保科剛，“統合開発支援ツールTIPPLER”，UNISYS技報，Vol. 13 No. 2, 1993.8.

[2] Ward Cunningham，“Smalltalk-80によるアプリケーションプログラムの作り方” ソニー・テクトロニクス(株)AI営業部抄訳，訳校閲竹内郁雄，bit, vol. 18, no. 4, pp. 379-395, 1986.4.

[3] 暦本純一，ユーザインタフェースとオブジェクト指向，ウィンターチュートリアル「オブジェクト指向」日本ソフトウェア科学会，1991.

執筆者紹介 伊勢本 勝一（Shoichi Isemoto）

1990年大阪大学基礎工学部情報工学科卒（グラフ理論専攻），同年日本ユニシス(株)入社．1995年3月までTIPPLERの開発に従事．現在，関西支社金融システム一部に所属．

TIPPLERとオブジェクト指向データベース

TIPPLER and Object-Oriented Databases

大石光太郎

要　約　オブジェクト指向開発環境TIPPLERを使用することにより，リレーショナルデータベースを基にしたクライアント/サーバアプリケーションを容易に開発することができる．また，新しいニーズであるマルチメディアデータを扱うアプリケーションもTIPPLERにより開発可能である．

本稿では，TIPPLERの使用者にとって，より使い易いデータベースアプリケーションの開発環境を目指したとき，どのようなデータベース機能が要請されるかを考察する．そして，それを実現するために，オブジェクト指向データベース（OODB）との統合の枠組を提案する．

現在，TIPPLERを使用してデータベースアプリケーションを作成するとき，オブジェクト指向言語Uniscriptとリレーショナルデータベースの APIライブラリを使用する．このときTIPPLERとしてのクラス定義とリレーショナルデータベースのスキーマ定義をそれぞれ行う必要がある．Uniscriptはオブジェクト指向モデル，リレーショナルデータベースはリレーショナルモデルと，両者のベースとするデータモデルは異なっており，使用者はなんらかのモデル変換の考慮を強いられている．

また，Uniscriptのプログラム空間のオブジェクトとデータベース空間のオブジェクトは同期しておらず，それらの整合性を保つのも使用者の責任となっている．このため，データベース関連のプログラミング量が全体の3割から4割になるとも言われている．

オブジェクト指向データベースは，まだ発展途上であるが，これらの問題のいくつかを解決することができるので，TIPPLERとオブジェクト指向データベースの統合は十分に価値がある．また，先進的なデータベースアプリケーションの開発環境としては，複数のメディアデータの統合（マルチメディアデータベース），分散オブジェクト管理（ORB，分散データベース），インターネット対応（WWW-データベースインタフェース）がクリティカルな課題であると考える．これらの課題を統一的に解決するためのアーキテクチャ（枠組）を考察する．

Abstract　TIPPLER, which provides an object-oriented applications development environment, helps facilitate the development of client/server applications based on a relational database. It also makes it feasible to develop multimedia data-handling applications which have emerged as one of new user needs.

This paper considers what database functionalities are required for TIPPLER users who desire to have a development environment for much easier-to-use database applications, and also proposes the architecture by which to integrate TIPPLER with advanced object-oriented databases for that purpose.

Today, for the development of TIPPLER-based database applications, used are both Uniscript, an object-oriented programming language, and an API library for a specific relational database. That

本稿に記載の会社名，商品名は，一般に各社の商標または登録商標である．

requires a class definition for TIPPLER and a schema definition for a relational database, respectively. The Uniscript language is based on an object-oriented model, while the relational database is on a relational data model; that is, both are based on two different data models; thus forcing applications programmers to consider some transformation of the models between the Uniscript language and the relational database.

Non-synchronization between objects in the Uniscript program space and ones in the database space also holds applications programmers responsible for the consideration of consistency between the two. For this reason, it is said that programming workloads involved in database handling account for somewhere between 30% and 40% of all.

Although the object-oriented database, which helps resolve some of the problems stated above, is still in the process of being developed, it is well worth while integrating TIPPLER and object-oriented databases. In terms of an advanced database applications development environment, the critical database features include the integration of multiple media data (multimedia database), distributed object management (ORB, distributed database) and Internet connection (WWW-database interface). This paper discusses an architecture for resolving these problems in an integrated manner.

1. はじめに

オブジェクト指向開発環境 TIPPLER を使用することにより，リレーショナルデータベースを基にしたクライアント/サーバアプリケーションを容易に開発することができる．また，新しいニーズであるマルチメディアデータを扱うアプリケーションも TIPPLER により開発可能である．

本稿では，TIPPLER の使用者にとって，より使い易いデータベースアプリケーション開発環境を目指したとき，どのようなデータベース機能が要請されるかを考察する．そして，それを実現するために，オブジェクト指向データベース（OODB）との統合の枠組を提案する．第2章では，オブジェクト指向開発環境として持つべき機能要件を概観し，第3章で TIPPLER のデータベース機能の現状について述べる．第4章では，TIPPLER の Uniscript 言語とデータベースの問題をより一般的にとらえるため，プログラミング言語とデータベースの文化の違いについて述べる．第5章では，筆者らが昨年，米国ユニシスで開発中の OODB プロダクト OSMOS を，TIPPLER のユーザデータベースとして実際に試用したときの経験を述べる．第6章では，新しいニーズであるマルチメディア，およびインターネットに関連したデータベースの機能要件について述べる．

第7章ではこれらを踏まえ提案として，マルチメディアデータ管理，分散オブジェクト管理，インターネット連携を考慮したデータベースアプリケーション開発環境のアーキテクチャ（枠組）を考察する．次に TIPPLER と OODB の統合の具体的方法について述べる．

2. オブジェクト指向アプリケーション開発ツールの機能要件

現在，オブジェクト指向アプリケーション開発ツールあるいはオブジェクト指向クライアント/サーバアプリケーション開発ツールなどと呼ばれているオブジェクト指向開発環境プロダクトが，数多く流通している．これらのプロダクトの開発の母体と

なっているのは，3GL言語，4GL言語，CASE，データベース，エキスパートシステムなど多様であるが，進化した結果はその母体の違いにもかかわらず，これらのプロダクトが提供する機能は，似たものとなっている．しかし，対象とするユーザのレベル，アプリケーションの規模のレベル，あるいは対象とするアプリケーションの形態（EUC，OLTP，DSS*）などそのターゲットに応じて，プロダクトごとに力点の置き方が少しずつ異なっている．

個々のプロダクトの特殊性を捨像したとき，オブジェクト指向アプリケーション開発ツールが共通に持つべき機能は，大方以下のようにまとめられる[4]．

- ●オブジェクト指向技術を使用していること．
- ●ユーザインタフェース開発機能を持っていること．
- ●アプリケーションロジックの開発機能を持っていること．
- ●データベースの開発/アクセス機能を持っていること．複数のメディアデータを統合的に扱えること．
- ●ネットワーク機能を持っていること．インターネットと連携できること．
- ●クライアント/サーバ機能を持っていること．

図1は，これらの機能を図で表現したもので，太い線はTIPPLERを表しており，データベース機能とネットワーク機能が相対的に弱いことを示している．

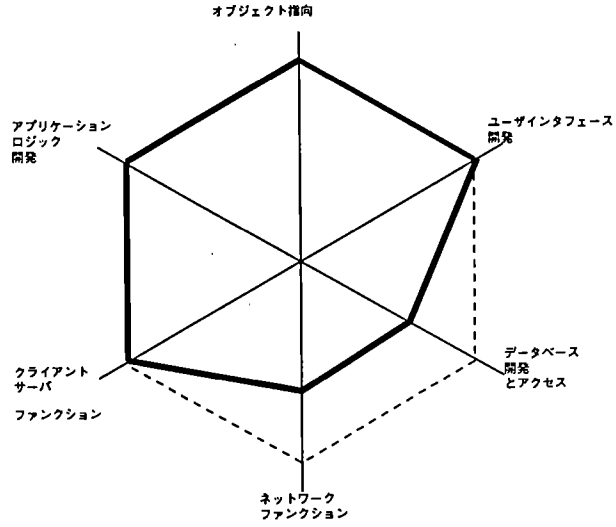


図1　オブジェクト指向アプリケーション開発ツールの機能要件

3. TIPPLERのデータベース機能の現状

現在，TIPPLERは，リレーショナルデータベースのORACLE，Informix**，

---

* EUC: End User Computing, OLTP: Online Transaction Processing,
DSS: Decision Support System

** ORACLEは，Oracle社のリレーショナルデータベース管理システムである．
Informixは，米国Informix社のリレーショナルデータベース管理システムである．

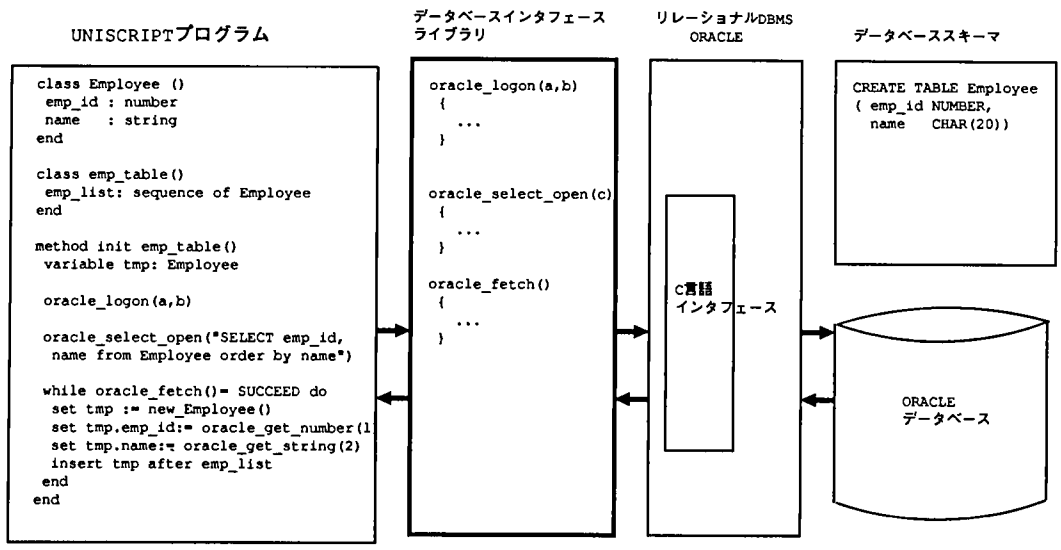


図 2 TIPPLER のデータベースインタフェースライブラリ（ORACLE の場合）

SYBASE*に対するデータベースインタフェースライブラリを提供している．このライブラリを使用することにより，Uniscript プログラムの中から直接 SQL 文を発行し，リレーショナルデータベースをアクセスすることができる．ライブラリは，各データベース管理システムが提供している C 言語インタフェース(Pro * C，ESQL/C，DBLibrary）を使用して作られている．データベース側の視点からは，C 言語インタフェースを使用したアプリケーションプログラムとして見えるし，TIPPLER の Uniscript プログラム側の視点からは，C 言語で記述された外部関数と見ることができる．

図 2 は，すべての従業員情報を名前順に ORACLE のデータベースから読み込むアプリケーションの例を示している．Uniscript プログラムにはクラスとして，Employee クラスと Employee クラスのすべてのインスタンスを Uniscript プログラム空間で保持するための emp-table クラスが定義されている．

一方リレーショナルデータベースのスキーマには，Employee という表が定義されている．Employee 表と Employee クラスは，全く同じものを二重に定義しているわけである．データベースの Employee 表のデータを Uniscript プログラムで使用するには，Uniscript プログラム側で定義した Employee クラスのインスタンスを生成し，次に項目単位にデータを get しては，そのインスタンス内の対応するスロットにデータを設定する．さらに処理上同時に使用するインスタンスはリスト構造で Uniscript プログラム空間に保持するため，insert tmp after emp-list というコマンドを使用者が明示的にプログラミングする必要がある．

例は非常に簡単なものであるが，同時に使用するクラス(表)，スロット(列)あるいはカーソルが多くなったり，エラーの細かい検査などを含めると，データベース関連のコードだけで，Uniscript プログラムコード全体の 3 割から 4 割を占めることもあ

* SYBASE は，米国 SYBASE 社のリレーショナルデータベース管理システムである．

る．ここでの問題を整理すると，次のようになる．

●スキーマをデータベースと Uniscript プログラムで二重定義しなければならない．

●データベース空間のレコードと Uniscript のプログラム空間のオブジェクトが同期していない．

●データベース関連の基本的なロジック(データ変換や整合性検査等)を直接 Uniscript プログラムに記述するので，データベース関連のコード量が多くなる．

これらの問題は，Uniscript がベースにしているオブジェクト指向モデルと，リレーショナルデータベース管理システムがベースにしているリレーショナルモデルのギャップが大きく影響していると考えられる．現在，使用者はこのギャップを埋めるために，大半がモデル変換の考慮をプログラミングしていると言える．

4. プログラミング言語とデータベースの文化の違い

一般的に，プログラミング言語 (PL) とデータベースは，カルチャの違いから来るいくつかの大きな相違点がある．

●永続性 (persistence)

プログラムの実行が終了してもプログラム空間のデータを保存するかどうか．データベースは保存するのを前提とし，PL は保存しないのを前提としている．

●データの共有 (sharing of data)

同一のデータを複数のプログラムで同時に使用するか．データベースは同時使用，PL は同時使用しないのを前提としている．

●大容量データ (large amount of data)

大容量のデータを扱うことを前提にしているかどうか．データベースはそれを第一義の前提としている．

●型としてのクラスと集合としてのクラス (class as type, class as collection)

データベースではクラスは同じ種類のデータの集まりの総称であり，一般的にデータ (インスタンス) を持たないクラスを対象とはしていない．また，オブジェクト指向での継承 (inheritance) において，データベースではインスタンス間の継承という概念が存在する．PL はどちらかというと型によるメソッドの継承を重視している．

リレーショナルデータベースではデータベース言語である SQL を，COBOL や C などの手続き型プログラミング言語に埋め込んで使用している．この二つの言語はデータモデルやデータ構造および操作系も異なっており，これはインピーダンスミスマッチと呼ばれる．アプリケーションプログラマは，二つの言語を学びこれらのインピーダンスミスマッチに対応するため，モデル変換のプログラミングをする必要がある．

プログラミング言語とデータベース言語のギャップを埋めるために，いくつかのアプローチがとられている．

●データベース言語 (DBL) の拡張

たとえばリレーショナルデータベース ORACLE では SQL 言語を拡張し PL/SQL という言語を提供している．PL/SQL は SQL 文に加えフロー制御文 (IF-

THEN-ELSE, EXIT, GOTO など), 反復文 (FOR-LOOP など), 代入文, 例外処理用構文を持つ.

● PL の拡張

現在多くの OODB プロダクトで行っているアプローチで, C や C++などの言語を拡張してデータベースオブジェクトを扱えるようにする.

●新しい言語 DBPL (Database Programming Language)

クライアント/サーバアプリケーション開発環境 FORTE の TOOLS スクリプト言語やオブジェクト指向データベース O^2 の O^2C などがこれに当たるであろう.

TIPPLER の Uniscript は, オブジェクト指向のプログラミング言語であり, それによってアクセスされるリレーショナルデータベースとの関係において, ここで述べたインピーダンスミスマッチが同様に存在している. 第3章で述べたオブジェクト指向モデルとリレーショナルモデルのギャップを埋め, かつ本章で述べたプログラミング言語とデータベースのインピーダンスミスマッチを解決するにはどうしたらよいのであろうか. 次章ではオブジェクト指向データベースを採用することにより, これらの問題がどの程度解決するか, 試行を基に考察する. そして 7.2 節で, オブジェクト指向データベースとの統合について提案をする.

5. TIPPLER と OODB の統合試行

筆者らは昨年, 米国ユニシスが開発中であるオブジェクト指向データベース OSMOS のプロトタイプ版を使用し, TIPPLER によるデータベースアプリケーションを試作する機会を得た. この作業は, TIPPLER と組み合わせて, オブジェクト指向データベースを使用するとどのような有用性があり, またどのような解決すべき課題があるのかを見究めることが目的であった. OSMOS は, セマンティックデータモデルをベースにしたオブジェクト指向データベースであり, 現在も多くの機能が開発中のプロダクトである.

実際に OSMOS を TIPPLER のユーザデータベースとして使用し, 使用者にとって有益であると感じた点は, オブジェクトモデルの親和性と言語の標準性である.

●オブジェクトモデルの親和性

TIPPLER のオブジェクトモデルと OSMOS のオブジェクトモデルは基本的に一致し, 親和性がある. 両者ともに, リレーショナルデータベースが不得意とする多：多の直接表現, セルフリンク関係の直接表現, 複合オブジェクトの直接表現ができるので, TIPPLER とデータベースの間でモデル変換を必要としない.

●言語の標準性

OSMOS は, 他の OODB のように, データベースインタフェースのために C_{++} を一切拡張しておらず, プリコンパイラを必要としない. OSMOS はスキーマに定義されたクラスごとに C++API (標準メソッド群) をスキーマから自動生成する. したがって標準の C++ の知識だけでデータベースアプリケーションを組むことができる.

今後考慮を要すると感じた点は, 以下の事柄である.

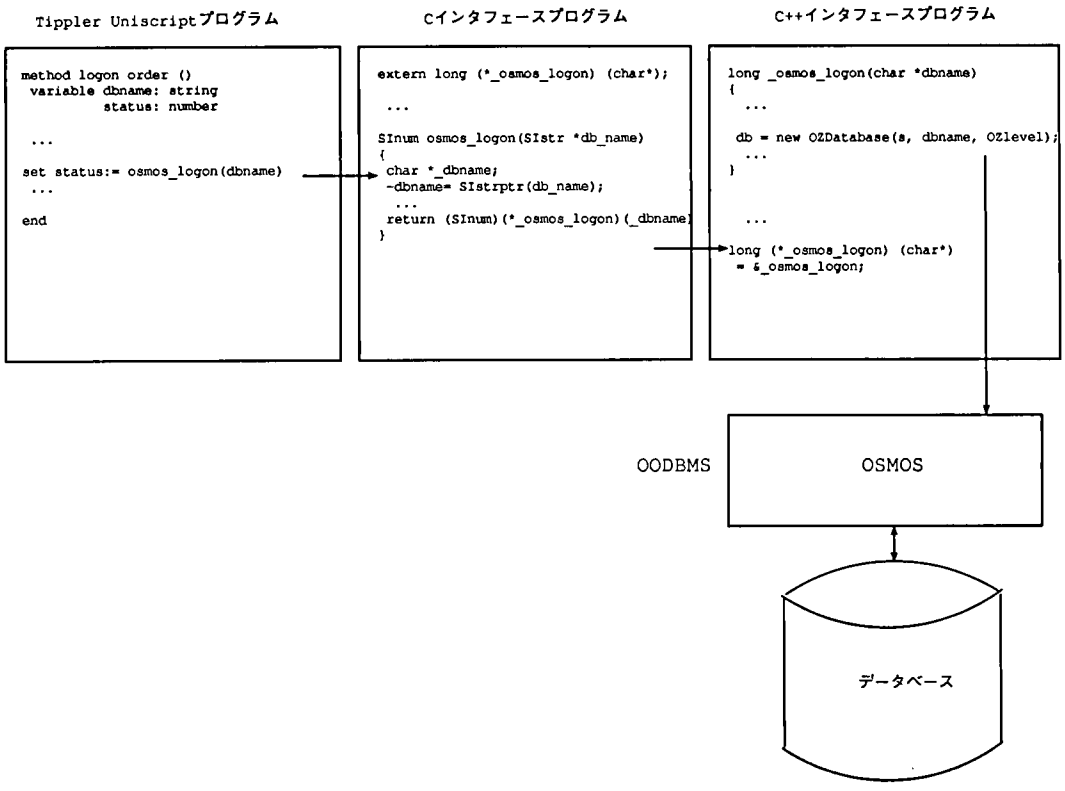


図 3　TIPPLER/OSMOS API

- Uniscript から C++API を呼び出す C API を作る必要がある（図3参照）．TIPPLER の Uniscript 言語は，C言語インタフェースを持っている．一方 OSMOS の API は，C++である．Uniscript から C++を直接呼べないので，間に C API を作る必要がある．(OSMOS は，現在Cインタフェースを持っておりこの問題は解決している．)
- データベース定義と TIPPLER のクラス定義を二重にする必要がある．これは TIPPLER と RDB での問題が TIPPLER と OODB に持ち越されている．
- TIPPLER プログラムがデータベースオブジェクトを使うときは，基本的にすべてのインスタンスを生成しながらデータベースオブジェクトを読んではスロットに内容を設定して，プログラム空間にすべてのインスタンスをそろえた上で処理を始める．また，Uniscript のオブジェクトとデータベースオブジェクトの整合性の保証は使用者の責任である．これも TIPPLER と RDB での問題が TIPPLER と OODB に持ち越されている．
- データ独立性

  データベースのクラス定義をヘッダファイルとしてプログラムにインクルードするので，クラス定義に変更があると，プログラムの再リンクを必要とする．これはいままでデータベースが培ってきたデータ独立性がもののみごとに踏みにじられ無視されていると言える．この状況は流通しているほとんどの OODB に当て

はまる．この問題解決のため，ビュー機能および動的スキーマ進化機能が必要とされるが，まだこれらを支援しているOODBは少ない．

- SQLまたはOQL（Object Query Language）を受渡しするインタフェースがない．

  言語側の構文としてデータベースアクセス機能が融合しているC++やCインタフェースは，C++やCに慣れたプログラマには良い環境であろう．しかしCOBOL, PL/1など高級言語とSQLに慣れたプログラマには，C++やCの枠組でポインタをたどりながら，データベースアクセスのプログラミングをしていくことに先祖返りの感を抱くことであろう．（OSMOSは，現在SQL（CLI: Call Level Interface）インタフェースを持っておりこの問題は解決している．）

6. マルチメディア/インターネットとデータベース

この章では，提案に先立ち，先進的なデータベース管理の機能要件も考慮の範囲に含めるため，マルチメディアとインターネットについて考察する．

現在急速な勢いで，マルチメディアアプリケーションをインターネットを含めた環境において開発するニーズが高まってきている．これにともない，データベース管理システムに対しても，インターネット上に分散したマルチメディアデータを効率良く管理し，アクセスする機能の提供が求められている．ここではマルチメディアデータの管理機能の要件と，インターネットにおける現在のマルチメディアデータの統合方法について概観する．

6.1 マルチメディアデータの管理機能

マルチメディアデータベースの基本要件は，テキスト，図形，画像，音声，動画などメディアが異なっても使用するデータモデルと操作インタフェースを変える必要がないというメディア独立性を保証することである．言い換えれば，マルチメディアデータベースとは，使用者がメディアの違いを意識せずにアクセスできるデータベース管理システムのことである．複数のメディアを扱えるデータベース管理システムは，リレーショナルデータベースやオブジェクト指向データベースとして数多く存在するが，メディア独立性の視点からまだマルチメディアデータベースと呼ぶことはできないと考える．

メディアを統合するアプローチには，メディアごとにライブラリAPIを提供する方式と，マルチメディアデータモデルとして統合して共通の操作言語を提供する方式がある．後者はまだ研究レベルにあり，実用レベルで現在有効なのは前者のメディアごとのライブラリAPI方式である．マルチメディアデータベースの基本要件をまとめると以下のようになる．マルチメディアデータベースは，オブジェクト指向である必要性はないが，オブジェクト指向技術は，マルチメディアデータベースにとって，データモデルの統合化や操作言語の統合化に現在最も有効と考えられる．

- ●複数のメディアデータを統合するデータモデル
- ●巨大で複雑な構造を持つメディアデータの扱い
- ●メディア共通処理とメディア依存処理を統合できる操作言語
- ●異なるメディアの同期制御

- 時系列，連続メディアデータの扱い
- メディアデータ圧縮機能
- 内容検索/ファジー検索機能
- DBMS としての基本的データベース管理機能

マルチメディアデータベースを実現するアーキテクチャとしては，以下の三つが考えられる[12]．

- 単一アーキテクチャ　：1 DBMS：n メディアデータベース
- 主－従アーキテクチャ：1 主 DBMS：n 従 DBMS：n メディアデータベース
- 協調型アーキテクチャ：1 統合用インタフェース：通信：n DBMS：n メディアデータベース

単一アーキテクチャは，一つのデータベース管理システムで複数のメディアデータを扱うものであるが，これはまだ研究レベルにある．現在有効なアプローチは，協調型アーキテクチャでインターネットの World Wide Web で実現している技術もこのタイプであると考えられる．

6.2 インターネットにおけるメディア統合

前節で述べた協調型アーキテクチャの一種と考えられるアプローチが World Wide Web（WWW）によって採用されている（図 4 参照）．WWW は，インターネットにおける広域分散ハイパーメディア情報検索システムである．インターネット上のテキストやイメージ，音声などのメディアデータを参照するリンク（ハイパーリンクと呼ばれる）を自分の文書の中に自由に記述できる．

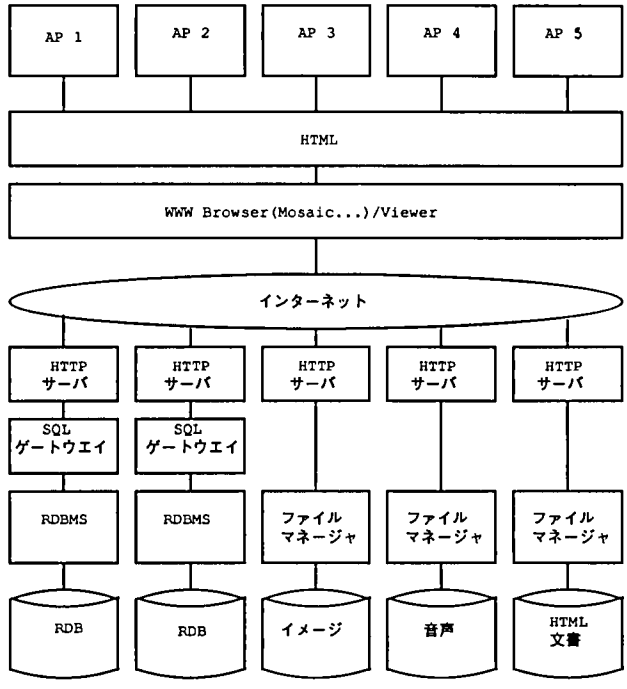


図 4　WWW におけるメディア統合

これは，電子文書を記述するための標準言語 SGML (Standard Generalized Markup Language) を基に作られた HTML (Hyper Text Markup Language) と呼ばれる言語で記述される．こうして記述された HTML 文書を実際に表示するのが WWW ブラウザの役目であり，Mosaic，Netscape，MacWeb など多くのプロダクトが提供されている．リンクは URL (Uniform Resource Locator) と呼ばれ，メディアを問わず統一された形で，インターネット上のリソースの物理的なアドレスを指定する (図5参照)．

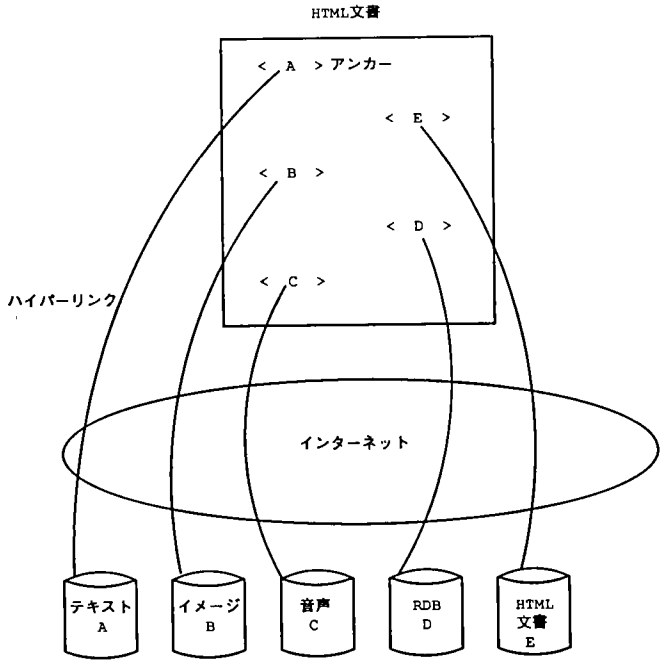


図 5 HTML 文書

最近では，HTML 文書の中から，リレーショナルデータベースをアクセスできるインタフェース (SQL Gateway など) が提供されており，一層統合が進んでいる．しかし，WWW のメディア統合は，リソースのインターネット内の物理的アドレスをクライアントとしての HTML 文書が直接指定するというアプローチであり，リソースのアドレス変更の影響をクライアントがダイレクトに被るという問題を持っている．リソースの物理的位置からの透過性をクライアントに提供する分散オブジェクト環境における ORB (Object Request Broker) や分散データベースにおけるディレクトリに相当する機能がインターネットにおいても要請される．

7. 提　　案

ここでは，今まで述べてきたデータベース関連の課題や機能要件を踏まえ，先進的なデータベースアプリケーションの開発環境のアーキテクチャ (枠組) を考察する．

次にオブジェクト指向データベースとの統合について焦点を絞り，具体的な改善項目について述べる．

7.1 アーキテクチャ

オブジェクト指向データベース，マルチメディアデータ管理，分散オブジェクト管理，インターネット連携を考慮したデータベースアプリケーションの開発環境のアーキテクチャは，図6のように，八つの層（レイヤ）から構成される．

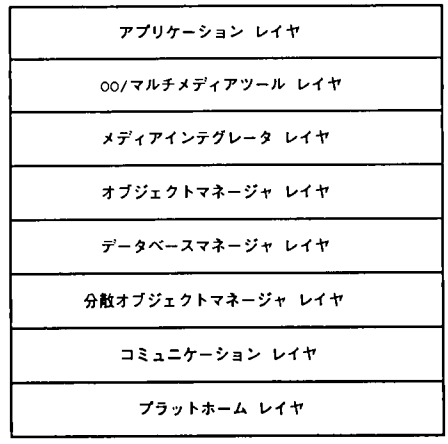


図 6　レイヤ

- ●第八層：アプリケーション層は，オブジェクト指向/マルチメディアを使用したユーザアプリケーションである．
- ●第七層：オブジェクト指向/マルチメディアツール層は，TIPPLER をはじめ，オープンなツールや言語である．
- ●第六層：メディアインテグレーション層は，メディア間の相互関係や同期など複数のメディアデータから構成される複合体（複合オブジェクト）を管理するインタフェースである．たとえば World Wide Web における Netscape や Mosaic などのブラウザが果たしている機能性である．
- ●第五層：オブジェクトマネージャ層は，オブジェクトに対するリクエストを一括して受け付ける．またオブジェクトバッファを管理する．
- ●第四層：データベースマネージャ層は，ローカルなデータベースを管理するオブジェクト指向データベース管理システムである．ここには各種のリポジトリサービス機能も含まれる．
- ●第三層：分散オブジェクトマネージャ層は，リモートのデータベースやリモートのオブジェクトへのアクセスを管理するオブジェクトリクエストブローカ（ORB）である．
- ●第二層：コミュニケーション層は，通信ネットワークによりメッセージやオブジェクトを転送する．
- ●第一層：プラットホーム層は，ハードウェアやオペレーティングシステムである．

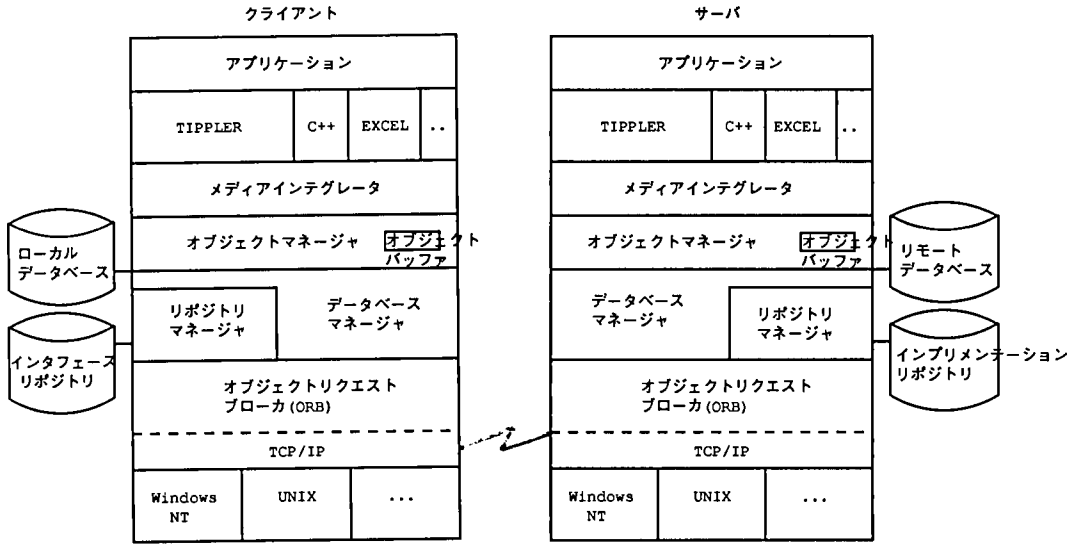


図 7　分散オブジェクト環境としてのアーキテクチャ

このアーキテクチャは，分散オブジェクト環境あるいは分散データベース環境を対象にしており，クライアントとサーバの役割を固定化してはいない．一つのノードが同時にクライアントかつサーバになることができる．ここでは，クライアント/サーバ形態のデータベースアプリケーションの視点からアーキテクチャを検証する（図7参照）．

ユーザアプリケーションからデータベースオブジェクトに関するリクエストが出されたとき，リクエストはオブジェクトマネージャが受ける．オブジェクトマネージャは，オブジェクトバッファをサーチし要求されたオブジェクトがキャッシュされていればオブジェクトバッファ内のそのオブジェクトへのポインタを返す．

オブジェクトバッファに要求されたオブジェクトが存在しないとき，ローカルなデータベースのオブジェクトが対象であれば，ローカルなデータベース管理システムを呼び出す．そして得られたオブジェクトをオブジェクトバッファにキャッシングし，ユーザアプリケーションにそのオブジェクトへのポインタを返す．

要求されたオブジェクトがローカルなデータベースのものでない場合には，オブジェクトリクエストブローカを呼び出す．オブジェクトリクエストブローカは，インタフェースリポジトリによりオブジェクトの呼び出し方を知り，インプリメンテーションリポジトリ（オブジェクトのディレクトリ情報）により，オブジェクトが存在するリモート（サーバ）を知り，そのリモートサイトのデータベースマネージャにリクエストを送信する．

メディア統合の視点からアーキテクチャを説明する(図8参照)．このアーキテクチャは，第六章で述べたマルチメディアデータベース実現のための三つのアプローチのうち，協調型アーキテクチャを採用してしている．すなわち一つの統合用インタフェースとしてメディアインテグレータ，通信機構としてオブジェクトリクエストブローカ，そしてインターネットに分散された各種のメディアデータベースを管理する複数

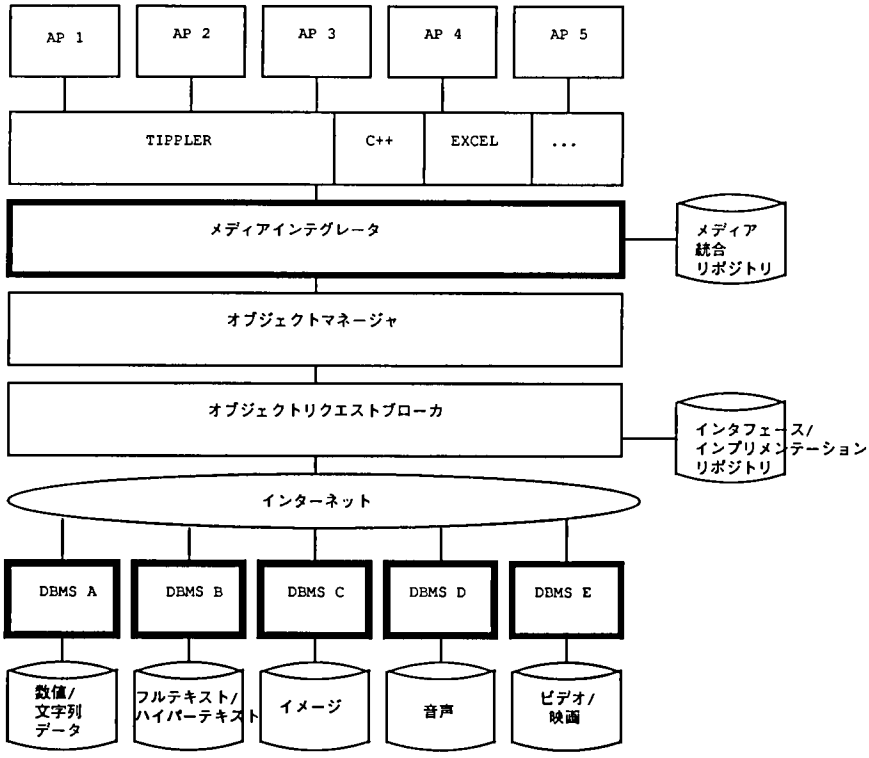


図 8　メディア統合としてのアーキテクチャ

のデータベース管理システムという構図である．このアーキテクチャは，World Wide Web において使用されており，実用の技術レベルにある．

アーキテクチャの中には，インタフェースリポジトリ，インプリメンテーションリポジトリ，メディア統合リポジトリというリポジトリが組み込まれている．リポジトリはアプリケーションシステムを構成する要素，構造，それらの間の関係についての情報を一元的に管理するデータベースである．インフラストラクチャとしてリポジトリをアプリケーション開発環境の中に内蔵することにより，そこから生成されるアプリケーションシステムもロバストなものとなるであろう．リポジトリは共有資源であるから，データベース管理システムが管理するデータベースであることが望ましく，オブジェクト指向技術とマルチメディアデータを支援できるものが理想的である．(UNISYS の UREP (Universal Repository) は，そのようなニーズに応えるために開発された本格的なリポジトリ管理システムである．)

7.2 オブジェクト指向データベースの統合

TIPPLER が標準で支援する使用者データベースとしてオブジェクト指向データベース (OODB) を統合するために，OODB が最低限満たすべき条件は，以下のものである．

- TIPPLER のオブジェクトモデルとの親和性
- C アプリケーションプログラムインタフェース

● SQL または OQL インタフェース

●マルチメディアデータの格納/アクセス機能

●リレーショナルデータベースと同等以上の処理性能

これらの条件を満足する OODB を得た上で，以下のような改造が TIPPLER に必要となるであろう．

●クラス定義の統合

OODB のスキーマから TIPPLER のクラス定義を生成するか，逆に TIPPLER のクラス定義から OODB スキーマを生成できる機能のどちらかが必要である．データ中心アプローチの観点からすると，OODB スキーマから TIPPLER のクラス定義を生成するのが順序であろう．

● TIPPLER プログラミング言語 Uniscript の拡張

Uniscript の言語仕様として SQL (OQL) 相当のデータベースアクセス機能，トランザクション処理用コマンドを支援するように拡張する．これは，プログラミング言語とデータベース言語のギャップを埋めるために，新しい言語を作成するアプローチと言える．Uniscript の言語仕様に，データベース機能をモデル化して反映することにより，プログラミング言語とデータベースのインピーダンスミスマッチがより高いレベルで解消できる．

● TIPPLER の GUI 機能とデータベース開発/アクセス機能の統合

TIPPLER の GUI 機能にデータベーススキーマ定義機能とフレーム/データベース連携機能を追加する．データベーススキーマ定義機能は，TIPPLER のクラス定義機能と統合する．フレーム/データベース連携機能は，フレーム内のアイテムとデータベースの情報を結び付けるもので，データベースの項目選択 (SQL の自動生成) か，複雑なものは使用者による SQL 指定によって行われる．これにより，基本的なデータベースアプリケーションは，プログラムレスでフレーム定義だけで作成できるようになる．

●プログラム空間のオブジェクト管理

これはアプリケーションプログラム空間のオブジェクトを DBMS あるいは DBMS センシティブ機能 (オブジェクトマネージャなど) の管理下に置くことである．こうすることにより，現在の TIPPLER プログラム空間のオブジェクトと，データベース空間のオブジェクトの非同期な二重管理という状況を大きく改善できる．またクライアント側にバッファ持つことでオブジェクトがヒットすればサーバに要求する負荷が軽減され，効率向上が図れる．以下は参考文献[8] からの引用である．

サーバのページバッファの他にオブジェクトバッファを用意する (図 9 参照)．オブジェクトバッファはクライアント側に配置しクライアントアプリケーションが直接見ることができる共有メモリ (Shared Memory) である．オブジェクトバッファ内のオブジェクトを管理するデータ構造は，Object Descriptor Hash, Object Descriptor, Object である．Object Descriptor Hash は OID をキーとして Object Descriptor をハッシュする．

アプリケーションには Object へのポインタではなく，Object Descriptor へ

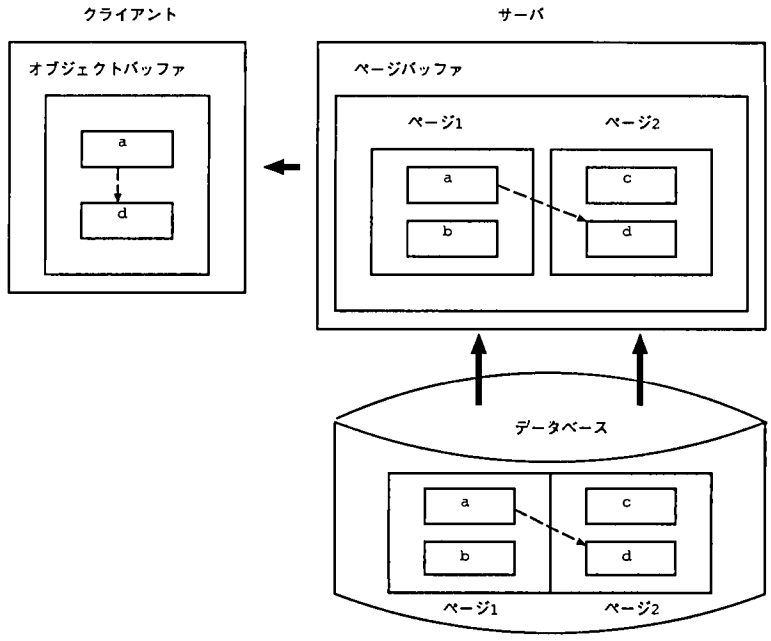


図 9 オブジェクトバッファ

のポインタを返す. Object はオブジェクトバッファの中でメモリのガーベージコレクションによって移動し得る. Object Descriptor には無駄なロック要求をクライアントがしないでも済むようにロック情報も保持する.

動的スキーマ進化機能のためには，オブジェクトバッファ内のオブジェクトを無効化する invalidation 処理が必要となる．また，複合オブジェクトの処理において，OID で示すオブジェクトがオブジェクトバッファにいないときは自動的にデータベースからオブジェクトを fetch する機構を設ける必要がある.

8. ま と め

本稿は，TIPPLER とデータベースとのより緊密な統合について考察してきた．データベースエンジンだけで，開発環境のないデータベースプロダクトの有用性は半減する．ほとんどのビジネスアプリケーションは必ずデータベースを使うのであるからデータベースを持たないアプリケーション開発環境の有用性も半減する．データベースとアプリケーション開発環境の緊密な統合は，両者の有用性を同時に高める.

このとき API アプローチでなく，親和性のあるモデルを持つデータベースを内蔵しているアプリケーション開発環境はさらに有用性を高めるであろう．なぜならスキーマの定義は一回ですみ，モデル変換もいらず，また GUI とデータベースが統合できるので，簡単なデータベースアプリケーションは，プログラムレスで作成できる可能性を持つからである.

オブジェクト指向データベースを統合する副次的利点として，TIPPLER リポジトリのデータベース化を，より整合のとれた形で容易に実現できることが挙げられる.

このリポジトリデータベースは，分散開発環境において，TIPPLER によるオブジェクト指向分析設計方法やデータ（ベース）中心設計の情報基地としての役割を果たすことになるであろう．

参考文献

[1] 保科剛．統合開発支援ツール TIPPLER，UNISYS 技報第 38 号，Vol. 13，No. 2，August 1993.

[2] 日本ユニシス，統合開発支援ツール TIPPLER(Motif 版)開発ガイド実装編，September 1993.

[3] 溝上昌宏，OSMOS と TIPPLER のインタフェースの実現，UNISYS，October 1994.

[4] Object-Oriented Application Development Tool, Object-Oriented Strategies, August 1994.

[5] Timonthy Andrews, Craig Harris, Combining Language and Database Advances in an Object-Oriented Development Environment, Proceeding of OOPSLA'87, pp. 430～440, October 1987.

[6] Toby Bloom, Issues in the Design of Object-Oriented Database Programming Languages, Proceeding of OOPSLA'87, pp. 441～451, October 1987.

[7] J. Eliot, B. Moss, Object-Orientation as Catalyst for Language-Database Integration, edited by W. Kim, F. H. Lochovsky, Object-Oriented Concepts, Databases, and Applications, Addison-Wesley, pp. 583～592, 1989.

[8] W. Kim, Integrating Object-Oriented Programming and Databases, in Introduction to Object-Oriented Databases, The MIT Press, pp. 172～182, 1990.

[9] W. Kim, Architecture, in Introduction to Object-Oriented Databases, The MIT Press, pp. 183～198, 1990.

[10] W. Kim, Architectual Issues in Object-Oriented Databases, JOOP, pp. 29～38, March 1990.

[11] R. G. G. Cattell, Object Data Management-Object-Oriented and Extended Relational Database Systems, Addison-Wesley, 1991.

[12] 情報処理特集 オブジェクト指向データベースシステム，Vol. 32 No. 5，情報処理学会，May 1991.

[13] Dimitris N. Chorafas, Heinrich Steinmann, Object-Oriented Databases, Prentice-Hall, 1993.

[14] 情報処理特集 マルチメディアデータベース，Vol. 28 No. 6，情報処理学会，June 1987.

[15] Klaus Meyer-Wegener, Database Management for Multimedia Applications, edited by J. L. Encarnacao and J. D. Foley, Multimedia, Springer Berlin, pp. 105～119, 1994.

[16] Karl Aberer, Wolfgang Klas, The Impact of Multimedia Data on Database Management Systems, International Computer Science Institute, 1993.

[17] Thomas C. Rakow, Erich J. Neuhold, Michael Lohr, Multimedia Database Systems-The Notions and the Issues, Arbeitspapiere der GMD, Sankt Augustin, February 1995.

執筆者紹介 大 石 光 太 郎 (Kotaro Ooishi)

昭和 46 年 10 月，日本ユニシス(株)入社．シリーズ 2200/1100 のデータベース管理システムの開発，保守，および適用支援業務に従事．現在，知識システム部に所属．

TIPPLER/V のCASE機能－Factory

A CASE Tool for TIPPLER/V—Factory

平 井 誠 人

要 約 我々は，TIPPLER/V（ティプラ・ファイブ）のUniscript言語に対応したCASEツール「TIPPLER/V Factory（ティプラ・ファイブ・ファクトリ）」を開発した．TIPPLER/V Factoryは，Visual BASICなどが備えているGUIビルダ，Smalltalk風のブラウザ，上流CASEツールが備えているクラス構造図エディタの機能を併せ持っている．これらの機能を利用することで，Uniscript言語によるプログラミングを，ビジュアルな環境で行えるようになった．

Factoryの開発では，「ソフトウェアの巨大さと複雑さ」の問題を，どのように克服するかが大きな課題であった．我々は，Factoryの機能を複数のプログラムの集合体として実現することで，見かけ上のソフトウェア規模を押さえた．また，データ構造とアルゴリズムを可能な限り単純化することで，ソフトウェア全体の複雑さを押さえることができた．

Abstract The author's team has developed the CASE tool called "TIPPLER/V Factory" for the Uniscript language of TIPPLER/V (five). TIPPLER/V Factory provides the same GUI builder as is available in Visual BASIC, a Smalltalk-style browser and the functions of the class structure editor embedded in upper CASE tools. The use of these has made Uniscript programming feasible in a visual environment.

The problem with the Factory development was how to get over the problem related to the "hugeness and complexity of software." Our efforts to implement the functions of Factory as a body of separated program modules have led to success in paring down the apparent size of the software package. Also, the maximum simplification of the data structure and algorithm has made it possible to keep the Factory product from being complex.

1. はじめに

TIPPLERは，GUIの存在を前提としたオブジェクト指向開発環境として，野村総合研究所と日本ユニシスによって1990年に開発された．TIPPLERは，C++やSmalltalkとは異なる独自の仕様を持ったUniscript言語を備えている．Uniscript言語は，オブジェクト指向言語に必要なカプセル化，継承，ポリモフィズムの機能を満たしている．一般のオブジェクト指向言語でクラスにカプセル化できるのはデータと手続きの二つだが，Uniscript言語では，この他にframe，menuといったプレゼンテーション定義，およびスロットのアクセス監視を行うdaemonもカプセル化することができる．

Uniscriptの言語仕様にプレゼンテーションの機能が組み込まれているという点は，他の言語にない特長である．C++に比べるとUniscriptの言語水準は高く，プレゼンテーション定義もGUIビルダなしで，十分定義可能なレベルにある．逆の見方を

本稿に記載の会社名，商品名は，一般に各社の商標または登録商標である．

すると，C++は言語で補いきれない部分を，GUI ビルダなどのツールで補強していると言えるだろう．

しかし，C++やVisual BASIC などが普及するにつれて，エンドユーザからは，それらのツールと同等な機能を持つ開発環境を望む声が出てきた．彼らの要求はもっともなものであるが，Uniscript 言語にはC++と同じアプローチが取れない，微妙な問題が存在する．

C++では，プレゼンテーション定義を「リソース」と呼ばれるファイルに書き出す．ここには，ウィンドウに対してどの部品をどの座標に配置するかが記述されている．リソースはC++の言語仕様とは完全に切り離されており，プログラマがエディタでこれらの情報を直接編集することはない．ゆえにC++の開発環境では，このリソース情報をツールによって一方的に生成してやればよい．

それに対して Uniscript 言語では，言語仕様にプレゼンテーション定義が含まれているため，プログラマは当然，エディタを使ってそれらを編集する．Uniscript 言語に対応した GUI ビルダを提供するには，プレゼンテーション定義が，エディタと GUI ビルダの双方から編集可能であるという条件を満たさなければならない．

この他に TIPPLER/V に依存しない問題としては，あらゆるソフトウェアに共通する「巨大さと複雑さ」の問題があった．UNIX 版 TIPPLER の開発環境として既に Uniguide を提供しているが，ソフトウェアの巨大さおよび複雑さから，多くの工数を安定化作業に費やしている．この問題は，1975 年に発刊された F. P. Brooks, Jr.の著書，THE MYTHICAL MAN-MONTH（邦訳『ソフトウェア開発の神話』）によって既に指摘されており，今日のソフトウェア開発においてもなお重要な課題である．Factory の開発では，高機能と保守容易性という，一見，矛盾する課題にも取り組む必要があった．

2. Factory のアーキテクチャ

2.1 Factory の全体構造

今回，我々が開発した Factory の構造を，図1に示す．Factory は，次のように大きく三つの部分に分けることができる．

1) リポジトリ・アクセス・ライブラリ（RAL）
2) 構文解析ライブラリ（RCL）
3) フロントエンド

Factory 全体を下から支えているのは，リポジトリに対する入出力を行う「リポジトリ・アクセス・ライブラリ」（以下 RAL と呼ぶ）および，それを補助する「構文解析ライブラリ」（以下 RCL と呼ぶ）の層である．Factory は，Windows NT のファイルシステム NTFS を利用したリポジトリを持っている．ここには最終的に Uniscript プログラムを生成するのに必要な「クラス定義」と「クラス情報」（method, frame, menu, daemon の情報），およびクラスにカプセル化できない include 文，start 文などの「グローバル情報」が保存されている．また，各フロントエンドで独自に管理している情報，たとえばクラスデザイナのクラスの矩形の座標や，コードジェネレータのコード生成規則なども，リポジトリに保存されている．

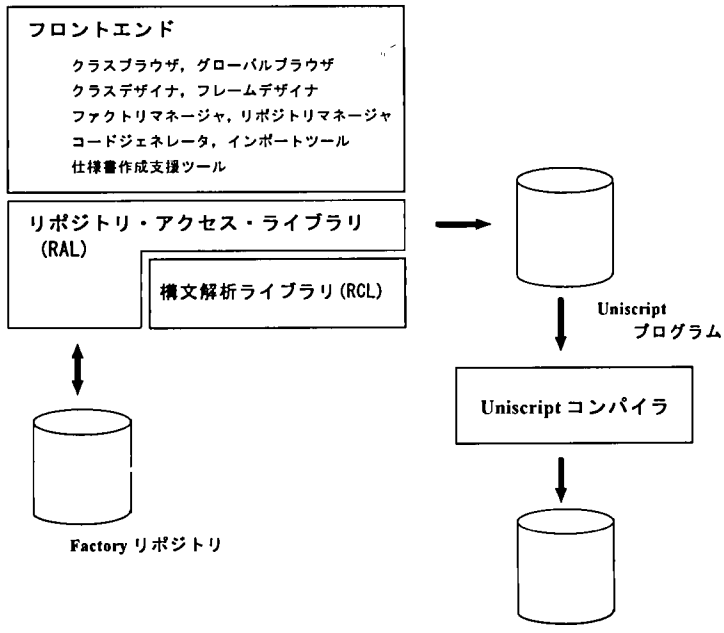


図 1 Factory の構造図

Factory の利用者が直接向き合うフロントエンドは，RAL/RCL の上の層として作られている．各フロントエンドは別々のプログラム，つまり別々の EXE ファイルになっている．したがって，RAL/RCL の層に乗る形で作成すれば，全く別々に開発することができ，各フロントエンド間の結合度も最低限に押さえることができる．

今回，フロントエンドには九つの機能を用意した．これらは，次の三つに分類することができる．

1) デザイナ系
2) ブラウザ系
3) その他

デザイナ系のフロントエンドは，GUI ベースで Uniscript プログラムを編集するためのものである．デザイナ系には，OMT に準拠したクラス構造図が扱える「クラスデザイナ」と，GUI ベースでフレーム定義を行う「フレームデザイナ」がある．

ブラウザ系のフロントエンドは，ソースレベルで Uniscript プログラムを編集するためのものである．ブラウザ系には，クラス定義およびクラスにカプセル化できる method, frame, menu, daemon を編集する「クラスブラウザ」と，クラスにカプセル化できない include 文や start 文などを編集する「グローバルブラウザ」がある．

その他のフロントエンドとしては，Factory のラウンチャとしての役割を果たす「ファクトリマネージャ」，リポジトリの管理を行う「リポジトリマネージャ」，Uniscript コードの生成を行う「コードジェネレータ」，既存の Uniscript プログラムをリポジトリに取り込む「インポートツール」，仕様書作成に必要なデータを生成する「仕様書作成支援ツール」がある．

2.2 リポジトリアクセスとフロントエンド

Factoryは，Windows NTのファイルシステムNTFSをベースとしたリポジトリを持っている．リポジトリには，クラス定義，クラス情報，グローバル情報，その他の情報が保存されている．リポジトリは，複数のアプリケーションに関する情報を保持し，それらを階層的に管理するために，「システム」と「アプリケーション」の二つの概念を持つ．システムの中には，アプリケーションまたは子のシステムのいずれかを複数持つことができる．システムとアプリケーションの概念は，UNIXやMS-DOSのファイルシステムにおけるディレクトリとファイルの関係に似ていると言える．アプリケーションは，一つの実行プログラム（EXEファイル）を作成するために必要なすべての情報を持つ(図2)．ただし，外部のクラスライブラリを参照する場合は，この限りではない．

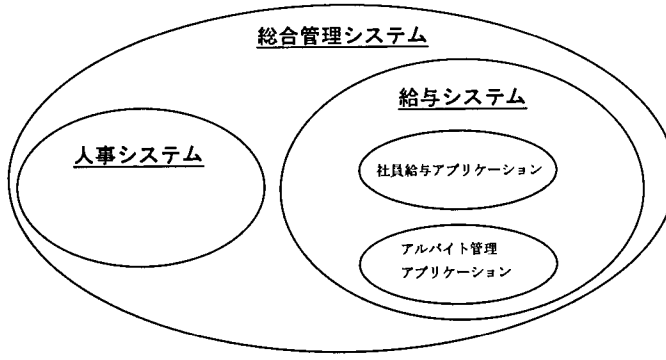


図2　システムとアプリケーション

リポジトリに保存されている情報の中で最も重要な情報は，最終的にUniscriptのソースコードのもととなるクラス定義，クラス情報，グローバル情報の三つである．クラス情報は，メソッド定義，フレーム定義，メニュー定義，デーモン定義に分けることができる．それぞれの定義は，一つのメソッド，あるいは一つのフレームというレベルにまで分割される．それぞれのメソッドやフレームの定義は，「define情報」と「token情報」の対で構成されており，これがリポジトリにおけるデータの最小単位となる．define情報はUniscriptのコードをそのまま記録した情報であり，token情報はdefine情報を文法解析して，トークンに分割した形の情報である(図3)．

define情報とtoken情報は，常に整合性が保たれた状態でリポジトリ上に存在する．フロントエンドは，これらの情報に対してアクセスする訳だが，ブラウザ系のフロントエンド（クラスブラウザ，グローバルブラウザ）はdefine情報側から，デザイナ系のフロントエンド（クラスデザイナ，フレームデザイナ）はtoken側からアクセスする．define情報とtoken情報の整合性を保つための相互変換は，すべてRAL内部で行っている．つまり，ブラウザ系のフロントエンドにはdefine情報だけが見え，デザイナ系のフロントエンドにはtoken情報だけが見えるようになっている．これによって，ブラウザ系とデザイナ系のフロントエンドが互いのことを把握していなくても，常にdefine情報とtoken情報の整合性が保証される．

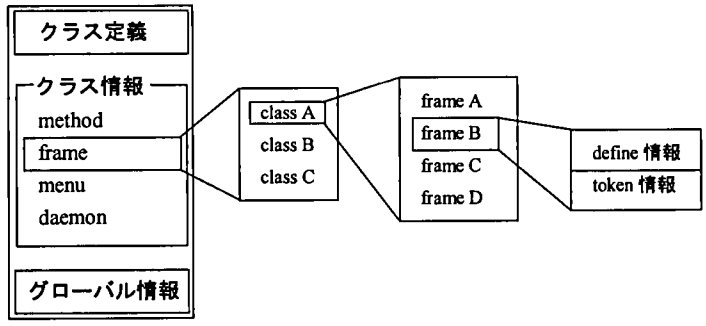


図 3 アプリケーションのデータ構造

3. Factory を利用したソフトウェア開発

3.1 Factory を利用した場合の開発の流れ

Factory を利用した開発では，従来のエディタとコマンドライン・コンパイラのみの開発とは異なり，同時に複数のフロントエンドを利用しながら開発を行う．また，ソースコードをリポジトリの形で管理するという点も，これまでにはなかった新しい考え方である．本章では Factory の各フロントエンドについて説明するが，その前に Factory を利用したソフトウェア開発全体の流れを示しておく（図 4）．

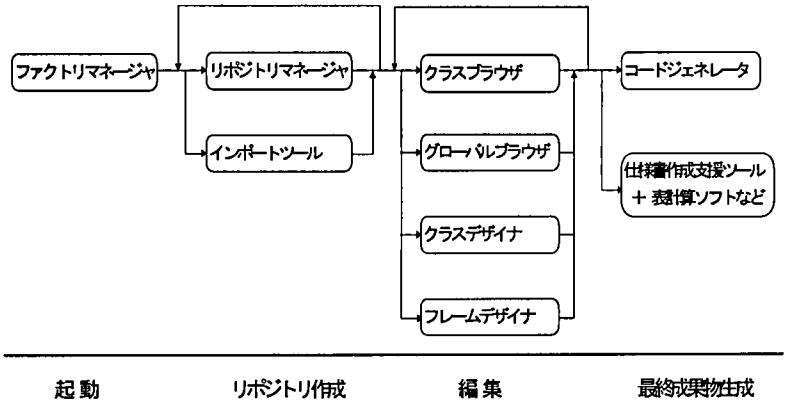


図 4 Factory を利用した場合の開発の流れ

Factory の利用者は，まず，「ファクトリマネージャ」を起動する．他のフロントエンドは，すべてファクトリマネージャによって起動されるようになっている．Factory を起動したら，まず最初に行うべきことはリポジトリの作成である．リポジトリの作成は「リポジトリマネージャ」を使って行う．リポジトリを作成したら通常はプログラムの編集作業に入るわけだが，エディタで作成した既存のプログラムを資産として活用したい場合は，「インポートツール」を利用して，リポジトリに取り込むことができる．

編集作業は，「クラスブラウザ」「グローバルブラウザ」「クラスデザイナ」「フレー

ムデザイナ」の四つのフロントエンドを使って行う．これらのフロントエンドは，並行して利用することができる．

編集作業を終えたら，コードジェネレータを使ってUniscriptコードを生成し，makeを行って最終成果物である実行プログラム（EXEファイル）を作成する．Factoryは，「仕様書作成支援ツール」を提供しており，開発したプログラムのカプセル化情報の一覧をCSV形式ファイルとして出力することができる．開発者は，表計算ソフトウェアなどを利用して，好みの形式の内部仕様書を作成することができる．

以上が，Factoryを利用したときのソフトウェア開発の流れである．

3.2 各フロントエンドの概要

ここでは，Factoryが提供しているフロントエンドの機能について説明する．紙面の都合もあり，ここですべての機能を紹介することはできない．それぞれのフロントエンドの詳細に関しては，マニュアルの『TIPPLER/V Factory CASE機能編』[3]を参照されたい．

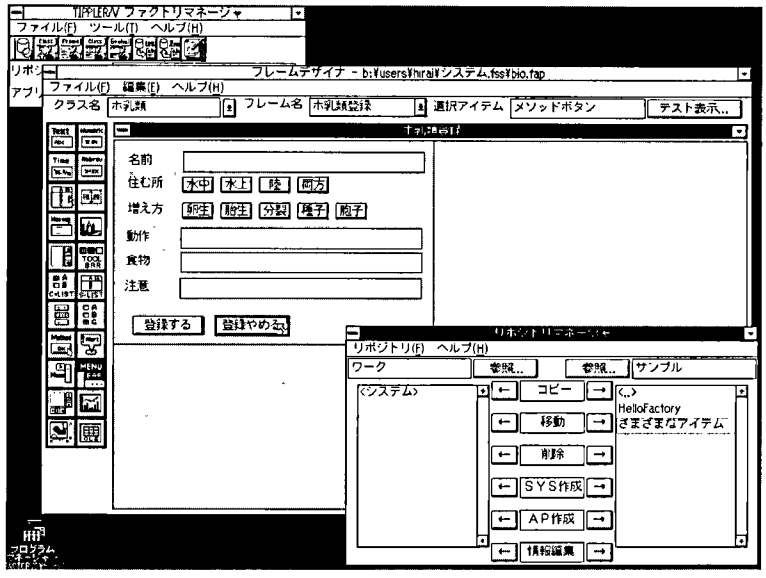


図 5 Factoryの開発環境

3.2.1 ファクトリマネージャ

ファクトリマネージャは，Factory専用のラウンチャ（他のプログラムを起動するツール）の役割を果たすためのフロントエンドである（図5の左上）．一般のラウンチャと異なる点は，Factoryのリポジトリの内部構造を認識し，他のフロントエンドを起動するときに，これからどのアプリケーションを編集するかを知らせる点である．

3.2.2 リポジトリマネージャ

リポジトリマネージャは，Factoryが利用するリポジトリファイルの作成，複写，移動，削除を行うためのフロントエンドである（図5の右下）．Factoryで開発するときは，まず最初にリポジトリマネージャを使って「システム」および「アプリケーション」を作成する．「アプリケーション」の中には，一つの実行プログラムを作成するた

めに必要な情報が保存されている．

リポジトリマネージャは，同時に二つのリポジトリを参照することができ，リポジトリ間でシステムやアプリケーションの複写や移動ができるので，リポジトリに保存されたデータの整理やバックアップを行うのに便利である．

3.2.3 クラスブラウザ

クラスブラウザは，クラス定義およびクラス情報 (method, frame, menu, daemon) をテキストベースで編集するためのフロントエンドである (図 6 中央)．クラスにカプセル化されているメソッドやフレームの一覧リストを常に眺めながら閲覧できるので，よりオブジェクト指向的な視点に立ってプログラミングできる．クラスブラウザは，Factory 利用者にとって，もっとも使用頻度の高いフロントエンドであり，Uniscript プログラムの大部分を記述することができる．

クラスブラウザと次に紹介するグローバルブラウザでは，キーワードによる入力ができるようになっている．これは，Uniscript の命令文を一覧リストから選ぶことによって，命令文のひな型を，カレントのカーソル位置に挿入するための機能である．Uniscript によるプログラミングに慣れていない初心者のことを考慮した機能であるが，熟練したプログラマが文法を確認するための手段としても有用である．

3.2.4 グローバルブラウザ

グローバルブラウザは，クラスにカプセル化できない include 文や start 文を記述するための補助的なブラウザである (図 6 右下)．クラスブラウザとグローバルブラウザを組み合わせれば，すべての Uniscript コードを記述することができる．

3.2.5 クラスデザイナ

クラスデザイナは，J. Rumbaugh の方法論である OMT[2] に準拠したクラス構造図を表示し編集するためのフロントエンドである．クラスデザイナで描画するクラスの

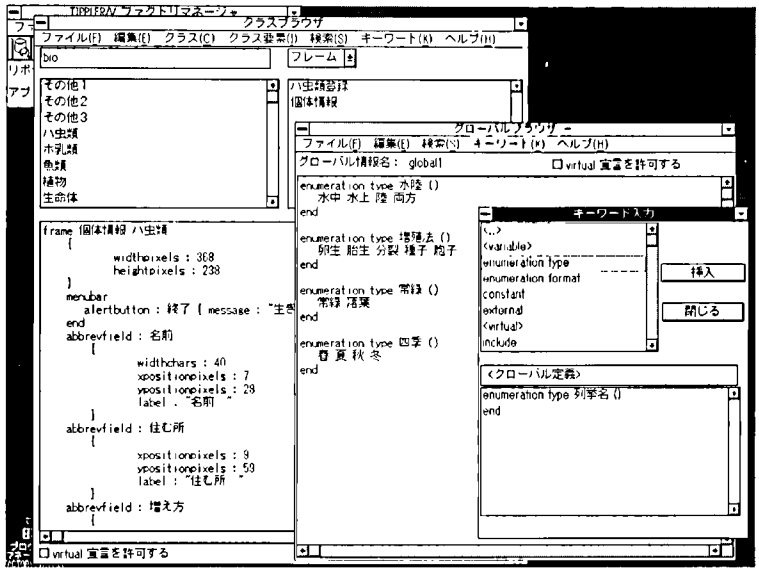


図 6 クラスブラウザとグローバルブラウザ

矩形には，クラス名，スロット名，メソッド名の他に，フレーム名，メニュー名，デーモン名も記述できるように拡張してある．

クラスデザイナはプログラム全体の構造を鳥瞰図的に閲覧できるため，開発の初期段階でクラス構造をデザインする場合に有効である．また，既存のプログラムの改善すべき点を探す場合に全体を眺めるためのツールとしても利用できる．

クラスデザイナからも，クラス定義やクラス情報の作成・削除ができるが，本格的にコーディングを開始した後は，クラスブラウザやフレームデザイナを使って編集し，クラスデザイナでは全体構造を閲覧するのにとどめておくのが良いだろう．

3.2.6 フレームデザイナ

フレームデザイナは，Uniscript 言語に対応した GUI ビルダである（図 5 の中央）．フレームデザイナは frame 定義を GUI ベースで編集するためのフロントエンドであり，フレームデザイナで作成したフレームの定義は，自動的に Uniscript プログラムに変換される．変換された Uniscript コードをクラスブラウザで編集し，その結果を再度フレームデザイナで編集することも可能である．

フレームデザイナは XY 座標ベースで画面を構築する．したがって，フレームのレイアウトが動的に変化する可能性のある自動レイアウト機能（XY 座標を指定しないアイテム）や panel 参照を利用したフレームを定義したい場合は，クラスブラウザを使ってフレームを定義することになる．

3.2.7 コードジェネレータ

コードジェネレータは，リポジトリに保存された情報をもとにして，最終的な成果物である Uniscript プログラムを生成し，make を行って，実行するためのフロントエンドである．コードジェネレータを使えば，従来のようにコマンドラインから mkmf コマンドや make コマンドを実行する必要はなく，すべてメニューを使って作業できる．コードジェネレータは，システム定義ファイルに記述する情報を GUI ベースで編集するなどのカスタマイズ機能も備えている．

3.2.8 インポートツール

インポートツールは，既存の Uniscript プログラムを Factory のリポジトリに取り込むためのフロントエンドである．これまで Factory を使わずに開発してきた Uniscript プログラムも，少ない手間でリポジトリに取り込むことができる．

3.2.9 仕様書作成支援ツール

仕様書作成支援ツールは，Uniscript プログラムの仕様書を作成する上で必要な情報を，CSV 形式のファイルとして出力するためのフロントエンドである．現在のバージョンでは，クラス定義とクラス情報の一覧を出力することができる．このデータを表計算ソフトウェアに取り込んで，利用者の好みに応じた形式の仕様書を作成することができる．

4. Factory の開発における課題と対策

最後に，今回，我々が Factory を開発する上で問題となった点と，我々がとった対策について述べる．

Factory は，UNIX 版の Uniguide と同等の機能を Windows NT 上で提供するため

に開発したが，設計思想は Uniguide のそれと大きく異なっている．Factory を開発する上での最大の課題は，Factory 全体の巨大さと複雑さをできるだけ下げることにあった．それを実現するために，我々は次の方針を立てた．

1) データおよびプログラムの構造をシンプルにする

2) 単機能のツールの集合体として動くようにする

以下，この二点について，さらに詳しく解説する．

4.1 データおよびプログラムの構造をシンプルにする

最近のソフトウェアは，ますます巨大化し，複雑化する傾向にある．巨大で複雑なソフトウェアは，不具合が収束するまでに多くの時間を必要とし，将来の機能拡張を困難なものにする．こうしたソフトウェアは，保守を続けていくにつれて，徐々に複雑さを増し，管理が困難なソフトウェアになっていく．

我々はこの問題に対処するため，可能な限りデータ構造をシンプルなものにするという方針を採った．データ構造をシンプルにすることによってアルゴリズムが単純化されるのであれば，たとえデータ量が増加するなどのデメリットがあっても，単純に実現できるアプローチを選択した．

優秀なプログラマは理想を求める傾向が強いため，結果的に複雑なアプローチを選びがちである．しかし我々は，平均的なプログラマでも十分に理解できるレベルの技術で，目的の機能を実現することに重点を置いた．

また，フロントエンド開発者には，Uniscript 言語のみによるプログラミングを義務づけ，独自に C 言語ルーチンを作成し呼び出すことを禁止した．しかし，Factory のような高機能なソフトウェアを開発していると，どうしても C 言語ルーチンを呼び出す必要に迫られる場合がある．そのような場合は，ライブラリ担当者に欲しい機能を伝え，ライブラリ担当者が共通の低水準ライブラリとしてまとめることにした．ちなみに，Factory 全体における C 言語の利用率は，構文解析ライブラリの RCL を除くと，4%程度に収まっている．

4.2 単機能のツールの集合体として動くようにする

Factory の巨大さと複雑さを下げるためのもう一つの対策は，Factory 全体の機能を，複数のプログラムの集合体として実現することである．これは，機能ごとに別々の実行プログラム (EXE ファイル) を用意することを意味する．UNIX 版の開発環境である Uniguide は，すべての機能を一つの実行プログラムで実現していた．Uniguide がこのようなアプローチを採った背景には，各フロントエンド間でデータの授受を行わなければならないという問題があったからである．Factory では RAL を用意することで，この問題を解決している．

各フロントエンドは，それぞれ 2,000〜7,000 ステップの Uniscript で実現できている (唯一の例外はフレームデザイナで，16,000 ステップである)．このようなステップ数でフロントエンドが構築できた背景には，RAL が比較的高水準なリポジトリアクセス機能を備えていることが挙げられる．RAL は，フロントエンドに対して，次のような機能を提供している．

- リポジトリ管理用関数
- define 情報アクセス関数

- token 情報アクセス関数
- クラス情報リスト取得関数
- フロントエンド依存のアクセス関数

「リポジトリ管理用関数」には，システムやアプリケーションの作成，コピー，削除といった機能が備わっており，フロントエンドからはこれらの関数を呼び出すだけで，ファイルマネージャのような機能を実現することができる．

「define/token 情報アクセス関数」では，クラス名とメソッド名，フレーム名などを指定するだけで，define 情報や token 情報にアクセスすることができる．これ以外に，特定のクラスで定義されているメソッドやフレームなどの一覧リストを取得する関数も用意している．

以上のように，基本的な機能を RAL が提供することで，フロントエンド開発者は低水準な機能の実装に振り回されることがなくなった．彼らは，自分の担当しているフロントエンドが提供すべき機能に集中して，設計や実装を行えばよい．

今回，Factory の機能を複数のフロントエンドに分割することで得られたメリットを，以下に列挙する．

- フロントエンド一つあたりのサイズが小さくなるので，見通しが良くなり，その結果，各フロントエンドの安定性が向上する．
- 将来，大きな機能を追加するときは新しいフロントエンドを増やせばよいので，既存のフロントエンドの複雑さが増すことはない．
- 特定のユーザに依存した機能を要求された場合，独自のフロントエンドを追加することで，特殊なニーズを満たすことができる．
- 各フロントエンドのサイズが小さいので，安定性が低下したフロントエンドを破棄して，再構築することが十分可能である．つまり Factory は，当初からフロントエンドのソフトウェア・ライフサイクルを意識することで，Factory 全体としてのソフトウェア寿命を延ばすことをねらっている．

その昔，巨大な OS であった Multics へのアンチテーゼとして UNIX が開発された．Factory も UNIX と同様なアプローチを採ることで，ソフトウェアの巨大さと複雑さの問題を克服している．ソフトウェア技術者は，複雑で機能を満載したソフトウェアは高く評価するが，シンプルにするための努力に対しては，低い評価しか与えない傾向にあるように思う．

5. おわりに

Factory の開発で我々が採用した「シンプルな設計」および「機能を分割して複雑さを下げる」というアプローチは，おおむね成功したと言える．例外は 16,000 ステップのフレームデザイナだが，このフロントエンドは要求される機能が多く，今回のアプローチでは完全には対処できなかった．フレームデザイナについては，厳格なオブジェクト指向分析設計を行って，再利用可能なクラスに分割する必要がある．部品に分割できれば，サイズの大きなソフトウェアであっても，積極的な意味での破棄と再生を繰り返すソフトウェア・ライフサイクルへの道が開けてくる．

今回の開発では，当然のことながら Uniscript を主力言語として採用した．Uni-

script 言語の機能の中でも，特に列型（sequence）データの存在とメモリ管理機構（ガベージコレクタ）の存在が，生産性の向上に寄与したと思う．デザイナ系のフロントエンドにおいては，TIPPLER/V のプレゼンテーション機能が大きく貢献したことは言うまでもない．ブラウザ系のフロントエンドでは，複数行のテキストが扱えるアイテムが不可欠であったため，TIPPLER/V のコンパイラに対して texteditor アイテムを追加することで，目的の機能を実現した．

今後も Factory に対して，いろいろな機能を追加していく予定である．その土台は，今回の開発で提供することができたと考えている．なお，TIPPLER/V Factory という名前の商品には，UNIX 版 TIPPLER で提供していた帳票作成支援ライブラリ（PUI）と同等の機能も含まれていることを付記しておく．

参考文献
［1］ F. P. Brooks, Jr., ソフトウェア開発の神話，企画センター．
［2］ J. Rumbaugh 他，オブジェクト指向方法論 OMT，トッパン．
［3］ 統合開発支援ツール TIPPLER/V Factory CASE 機能編，日本ユニシス．
［4］ 統合開発支援ツール TIPPLER/V ランゲージリファレンス，日本ユニシス．

執筆者紹介 平 井 誠 人（Makoto Hirai）
昭和61年福岡大学工学部電子工学科卒業．同年日本ユニシス㈱入社．システムプロダクト本部ソフトウェア二部を経て，現在，システム技術本部知識システム部にて TIPPLER の開発，保守業務に従事．情報処理学会会員．

TIPPLERアプリケーションとInterleaf 5の連動

A TIPPLER Application and its Linkage with Interleaf 5

三池良洋

要約 ビジネス・プロセスの国際化，オープン化の流れの中で，企業活動の様々な側面で発生するドキュメントの電子化，標準化が必要となってきている．ドキュメントの効率的な管理が市場における競争力強化につながるためである．また一方では，ソフトウェア産業においてもオブジェクト指向技術やCASEツールを利用したシステム開発によって生産性向上を図ることが大きな課題となっている．

本稿は，ソフトウェア開発に伴うドキュメンテーションに焦点を当て，TIPPLERのCASEツールと連動した仕様書の自動生成の仕組みを，統合文書管理システムInterleaf 5を活用して実現することを目的としたプロトタイプ・モデルの開発を行った際の報告である．本システムの機能概要に加え，開発の背景およびインプリメンテーションを述べていく．

Abstract Amid the current trends toward the global extension of established business processes and the growing adoption by information technology users of distributed open systems, there has been an increasing need to digitize and standardize documents generated in all different scenes of business activities. That is because efficient document management leads to much stronger competitiveness in the marketplace. In the meantime, what the software industry now eagerly desires to do is to achieve higher productivity by developing systems for which both object-oriented technology and CASE tools are used.

Focusing on documentation involved with software development, this paper reports on a newly created prototype model, whose objective is to process the mechanism for the automated generation of software specifications documents with the help of a TIPPLER CASE tool and with the use of Interleaf 5, a document management system.

Besides giving an overview of the functions of the model, this paper describes its implementation as well as why this development effort was needed.

1. 開発の背景と目的

今日の情報処理技術に静かな革命が起きている．その重要なキーワードは“ドキュメント・マネジメント”である．オープン・システムとクライアント・サーバ・コンピューティングという情報処理技術の発達と，競争優位をもたらすビジネス上の最も重要な情報（ドキュメント）がノー・コントロールであることにより生じる問題を解決したいという要請が急増するニーズの背景となっている．これは昨今のCALSをめぐる欧米/日本の動向にも顕著に現れている．オープン化，国際化の流れの中で，ドキュメントの電子化と標準化による情報の共有化と再利用性を高めることで，組織としての競争力を強化することが必要になってきたためである．

また一方，同じく市場の国際的な拡がりの中で，企業が生産する製品の品質管理も大きな課題となっている．この動きの最も大きなものが，ISO 9000シリーズによる品

本稿に記載の会社名，商品名は，一般に各社の商品または登録商標である．

質管理システムである．この命題はなにも製造系メーカばかりのものではない．ソフトウェアという商品を開発・提供する情報サービス産業においても全く同様である．

本稿ではソフトウェア開発にまつわるドキュメンテーションの効率化と品質の向上を実現するための具体的なモデルを紹介する．今回紹介するモデル・システムは，オブジェクト指向技術に基づくシステム設計・開発が一般化する中で，TIPPLER* のCASEツールと統合ドキュメント管理システムInterleaf 5** を統合し，効率的なソフトウェア開発とドキュメント・プロセスを実現することを目的とするものであった．

Interleaf 5とCASEツールの連携ということでは，米国ではHP社製のCASEツールSoftBench*** とInterleaf 5のリンクのためのCASE connectという製品がInterleaf社より販売されている．今後，オブジェクト指向技術とCASEを用いたシステム開発が日本においても一般化することが予想されるため，日本ユニシスのオリジナル製品であるTIPPLERのCASEツールをターゲットに同様の仕組をInterleaf 5で実現することは将来に向けても意義のあることと認識し，モデル・システム開発に臨んだ．

2. 開発方針

TIPPLERのCASEツールとInterleaf 5（以下，IL 5と記す）の統合は，ソフトウェア開発業務におけるドキュメンテーションの効率化と品質の向上が目的である．ソフトウェア開発に関連するドキュメントとして各種仕様書を対象とした．そしてその具体的なシステム要件は次の通りであった．

- TIPPLER CASEツールが生成する様々なチャート類を自動的にドキュメント中に取り込む．
- ドキュメント化されたチャートは元データと動的なリンク関係を保持する（OLE的なリンク機能）．
- ドキュメント化された仕様書の改訂管理を実現する．

これらを実装する仕組みをIL 5，TIPPLER双方の開発により実現した．ただし，開発を行った時点ではTIPPLERのCASEツールが開発中であったため，IL 5と連携するTIPPLERアプリケーション（以下，TIPPLER APと記す）として，簡易ドローイング・ツールであるPresentation Toolをモデルとした．本プロトタイプ・システムの名称はInterleaf-TIPPLERリンク・システム（以下，ITリンク・システムと呼ぶ）とした．

ITリンク・システムの実現に当たり，まず検討したのがデータ交換のためのファイル形式である．データはクラス図やDFD（データ・フロー・ダイアグラム）等であるため，グラフィックデータを表現するファイル・フォーマットとしてPostScript**** やCGM（Computer Graphics Metafile）が考えられた．TIPPLERにはPostScript出力のためのクラスライブラリが既に存在しており，これを利用することも考えたが，

---

* 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，日本ユニシスの登録商標である．
** Interleaf 5はInterleaf社の登録商標である．
*** SoftBenchはHewlett-Packard社の登録商標である．
**** PostScriptはAdobe Systems社の登録商標である．

PostScript は取込み後イメージ（ラスタ）情報となるため，ドキュメント化した後の自由な編集が制約されること，CGM については日本語データ取扱いに関するコンセンサス形成が不十分であること，を考慮した上で今回はドキュメント・データの交換に豊富な実績のある IL 5 のデータ交換のためのファイル形式 Interleaf ASCII* を利用することにした．

本開発は UNIX**ワークステーションである US ファミリ上で行ったが，OLE 的なリンク制御は，TIPPLER for Windows の OLE クラスライブラリと，IL 5 が標準的に備える外部アプリケーションとのリンク機能 ActiveLink を参考に実現した．ActiveLink は MS-Windows*** の OLE などに先駆け IL 5 が実現したコンパウンド・ドキュメント作成のためのモジュールであり，ドキュメントの個々の構成要素をオブジェクト化して管理できる．また，今後リリースの予定される IL 5 の Windows NT 版と，TIPPLER for Windows 間で OLE を用いて実現される連携のイメージと，操作性を含めて統一することも意図して OLE と同様のインタフェースでアプリケーション間リンクを実装した．

このような開発方針に基づき，IL 5 のドキュメントをベースにさまざまなツールを利用しながらオブジェクトを構成・編集する仕組みとして IT リンク・システムを開発し，TIPPLER の CASE ツールに限らず多様なアプリケーションのデータをオブジェクト化できる汎用的な構造を持たせた．

仕様書の改訂管理については IL 5 のオプション機能を用いることとし，特別なカスタマイズはしていない．

3. システムの概要

IT リンク・システムは，TIPPLER AP が生成するグラフィック・データをドキュメント化するために IL 5 に自動的に取り込み，そのリンクをアクティブに保持するシステムである．主な機能を以下に示す．

- TIPPLER AP が生成するグラフィック・データの IL 5 文書への取込み
- OLE 的な動的なリンク管理（以下 IT リンク機能と呼ぶ）
- 元データを作成した TIPPLER AP の自動起動
- IL 5 文書オープン時のデータの自動更新

対象となる TIPPLER AP のデータは picture**** 機能を用いて作画されたグラフィック・データである．TIPPLER AP と IL 5 の二つのプログラム間でのデータの授受は，中間ファイルを用いる形態とした．IT リンク・システムによる連携のイメージの一例を図 1 に示す．

図 1 からも分かる通り TIPPLER AP を使用するユーザは，アプリケーションに付与された IL 5 インタフェースのためのボタンにてデータを保存することにより，容易に IL 5 に絵図情報（テキスト情報を含むグラフィック・データ）を受け渡すことがで

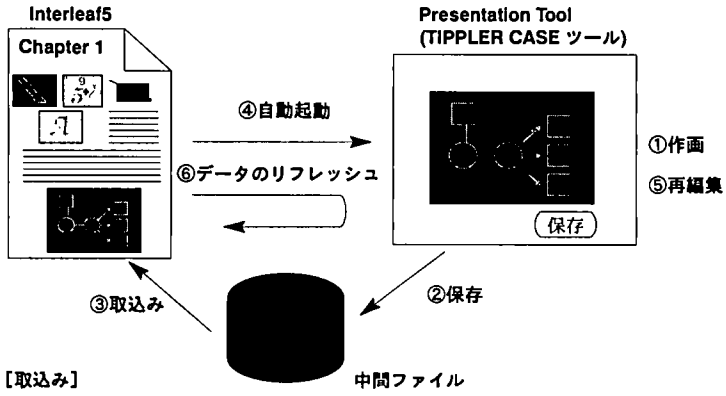
---

* Interleaf ASCII は Interleaf 5 が備える外部アプリケーションとのインタフェースのための公開ファイル・フォーマットである．

** UNIX は，X/Open カンパニーリミテッドがライセンスしている米国ならびに他の国における登録商標である．

*** MS-Windows は MicroSoft 社の商標である．

**** picture は TIPPLER のグラフィック・オブジェクトの一形式である．



① Presentation Tool 内で描画する.
(TIPPLER CASE ツールが自動的に生成するチャートを想定)

② データを保存する. データ交換のための中間ファイルが作成される.

③ IL5 より, 中間ファイルの名称を ID にして仕様書への取込みを指示する.

[編集]

④ IL5 より, 取り込まれたデータの編集を指示することにで, 自動的に Presentation Tool が起動され, 該当の絵図が編集可能な状態になる.

⑤ Presentation Tool 内で再編集する.
(TIPPLER CASE ツールでは例えばプログラム構造が変わった場合などを想定)

⑥ Presentation Tool 内で再保存することにより, IL5 側の文書の絵図が自動的にリフレッシュされる.

図 1　IT リンク・システムによる連携処理

きる. この機能を実装する内部的な構造は, IL 5 インタフェースのための機能群をクラスライブラリとして実現し, 様々な TIPPLER AP にこの機能を簡単に組み込めるような汎用的な仕組みとした.

4. インプリメンテーション

本章では IT リンク・システムの内部的な仕様について述べていく. はじめに TIPPLER AP と IL 5 間でのデータ交換のためのファイル構造を, 次に TIPPLER 側, IL 5 側での IT リンクの実装について順に説明する.

4.1 IT リンク・システムで使用されるファイル

TIPPLER AP と IL 5 間でのデータの授受および IT リンク実現のため 3 種類のファイルを使用している. データ 1 件につきこれら 3 種類のファイルが一つずつ作成され一つの論理的なデータベースを構成する.

1) IT データ・ファイル

このファイルは TIPPLER AP のグラフィック・データを IL 5 に受け渡すためのデータ・ファイルである. ファイル形式は IL 5 が外部アプリケーションとのインタフェースのために用意している IL 5 形式のファイル・フォーマットである. つまり TIPPLER AP のグラフィック・データを TIPPLER AP 側で IL 5 が読み込めるフォーマットでファイルとして記述する. IT リンク・システムでは TIPPLER 側に picture データをこの形式のファイルに書き出すためのクラスライブ

ラリを開発した.

2) インスタンスダンプ・ファイル

インスタンスダンプ・ファイルはITリンク実現のために用いられるTIPPLER側のデータ保存ファイルである.ITデータ・ファイルはIL 5に受け渡されるが,このファイルはTIPPLER APがローカルに使用するファイルである.

TIPPLER APのデータが一度IL 5に取り込まれ,IL 5側でそのデータに対してデータの編集が指示されるとTIPPLER APが自動起動されるが,その際にデータをTIPPLER AP内で再生するために使用される.

3) IT管理ファイル

IT管理ファイルはITリンクを実現するための管理情報を含むメタデータ・ファイルである.該当のデータに対応するITデータ・ファイル名やインスタンスダンプ・ファイル名などを内容として含む.IL 5, TIPPLER APともにこのIT管理ファイルを介してデータ本体を識別・アクセスする.IT管理ファイルは,データ・ファイルと同時にTIPPLER APが作成する.

IT管理ファイルはテキスト形式のファイルであり,以下の四つの情報を含む.

① ITデータ・ファイル名称

② ITデータ・ファイル最終更新日時

③ オブジェクト・プログラム名

④ インスタンスダンプ・ファイル名

②の最終更新日時をもとに,IL 5は①で指定されるITデータ・ファイルに対し,前回の取込み以降,改訂が成されたかどうかをチェックする.③のオブジェクト・プログラム名はITデータ・ファイルを生成したTIPPLER APの実行モジ

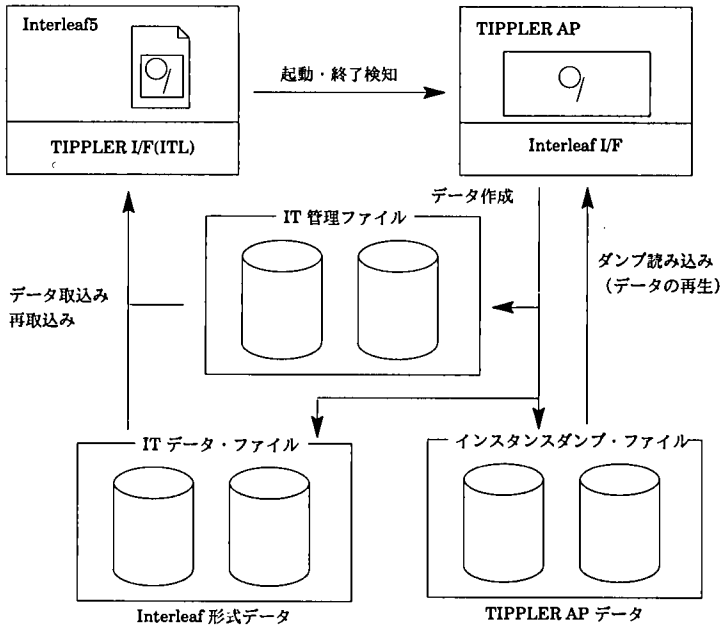


図 2 ITリンク・システムとファイル

ュールの名称である．IL 5から元データを作成したTIPPLER APを起動する際に用いられる．④のインスタンスダンプ・ファイル名はTIPPLER AP自体のデータファイル名称である．

これら三つの中間ファイルとITリンク・システムの関連を図2に示す．

4.2 TIPPLER側API

4.2.1 ITリンクの対象となるTIPPLERのデータ

ITリンク・システムにより，TIPPLER APがpicture機能を用いて作画したグラフィック，文字情報のうち以下のオブジェクトがIL 5に取り込み可能となる．これらの基本要素の組合せにより，TIPPLERのCASEツールが生成するチャート類を構成する図形のほとんどが表現可能と考えた．

- 直線（line）
- 多角形（polygon）
- テキスト（text）
- 矩形（rectangle）
- 円（circle）

4.2.2 TIPPLER AP側でのITリンクの実装

ITリンクを実現するためにTIPPLER AP側でApplication Classという名称のクラスライブラリを開発した．これはTIPPLER for WindowsのOLEクラスライブラリを参考にして実装した．

ITリンク・システムの中で，TIPPLER APの起動のされ方には次の2通りがある．

① TIPPLER APが単独で起動される場合（通常のTIPPLER APの起動のされ方）

② ITリンクによりIL 5からOLE的に起動される場合

である．②の場合，TIPPLER APはIT管理ファイルを参照し，対応するインスタンスダンプ・ファイルから該当のデータを再生する必要がある．また①の場合，②の場合とも，当該APの中でpicture機能によりグラフィック・データを作成し，それをIL 5に受け渡すためにIT管理ファイル，ITデータ・ファイル，インスタンスダンプ・ファイルを作成する必要があるが，これらの制御をこのApplicationClassクラスライブラリが行う．これらのクラスライブラリを利用することによって，様々なTIPPLER APに，IL 5との連携のためのファイル生成等のインタフェース機能を容易に付与（たとえば，メソッド・ボタンとして）することができる．

4.2.3 ITデータ・ファイルの生成

TIPPLER APのpictureオブジェクトをIL 5形式のファイルへ変換するための関数本体としてpicture_markupクラスライブラリを開発した．これは，ITデータ・ファイルを作成するための基本的な関数群である．以下にそれを示す．

- picture_markup_start
- picture_markup_end
- draw_line
- draw_rectangle
- fill_rectangle
- draw_circle
- fill_circle
- draw_text
- draw_polyline
- fill_polygon
- dash_polyline
- dash_line

このうち，picture_markup_start, picture_markup_endはITデータ・ファイル生成に必須な関数であり，それぞれITデータ・ファイルに必要なファイル・ヘッダとフッタを書き出す．この二つの関数の間にpictureのオブジェクト・タイプに応じたIL 5

形式への変換関数がコールされ，IT データ・ファイルが生成される．

4.2.4 picture_markup クラスライブラリと生成される IT データ・ファイル

picture_markup クラスライブラリの呼び出しによって生成される IT データ・ファイルの例を紹介する．

たとえば，draw_line は

```
draw_line (StartX : number,  StartY : number,
           EndX : number,  EndY : number,
           LineWidth : number, Color : string, Sensitive : boolean)
```

という形式で呼び出される．この関数が生成する IT データ・ファイル中での記述は次のようになる．

```
(v 7, 1, 0, ,StartX, StartY, EndX, EndY, Color, 0, LineWidth, 0)
```

この記述の意味は，

v 7：直線を表す IL 5 のオブジェクト・タイプ記号

1：Z 座標（オブジェクトの重なりを制御する）

0：ロックのフラグ

StartX, StartY, EndX, EndY：始点，終点の座標

Color：オブジェクトのカラー

0：オブジェクトの表示，非表示の制御フラグ（0：表示に固定）

LineWidth：線の太さ（point 数）

0：線のパターン（実線，点線等）

である．LineWidth は TIPPLER 側ではピクセル数で表現されているが，IL 5 ではポイント数が用いられるため数値の変換を行っている．

同様に draw_rectangle は

```
draw_rectangle (StartX :number,  StartY :number,
                Width :number,  Hight :number,
                LineWidth :number,  Color :string, Sensitive :boolean)
```

という呼び出し形式であり，生成される IT データ・ファイルのイメージは，

```
(p8, 1, 0, ,5, 7, 0
  (g9, 1, 0,
    (v7, 1, 0,, StartX, StartY, StartX+Width, StartY, Color, 0, LineWidth, 0)
    (v7, 1, 0,, StartX, StartY, StartX, StartY+Height, Color, 0, LineWidth, 0)
    (v7, 1, 0,, StartX+Width, StartY, StartX+Width, StartY+Height, Color, 0,
LineWidth, 0)
    (v7, 1, 0,, StartX, StartY+Height, StartX+Width, StartY+Height, Color, 0,
LineWidth, 0)
  )
)
```

となる．p 8 は多角形を表し，g 9 はグループを表す記号で，ここでは四つの v 7(直線) オブジェクトがこの g 9 記号によってグループ化され多角形 p 8 を作っている．

テキストの場合の例を示す．テキストは，draw_text 関数によって変換される．

draw_textの呼び出しイメージは次の通り．

```
draw_text (StartX :number,  StartY :number,
           String :string,  Font :string,
           Color :string,  Sensitive :boolean)
```

これにより生成されるマークアップは次のようになる．

```
(T16, 1, 0,, StartX, StartY, 7, 127, 5, 7, 127, 1, 0, 2,
<!Page, Width=width Inches, Height=height Inches>
<"IT", Font=Font, Alt Font=Font>
String
<End Text>)
```

TIPPLER APで描画されたテキスト情報はIL 5内では作画領域中のテキストとして展開される．この場合，生成されるイメージは他のオブジェクトに較べると少し複雑である．IL 5側でのテキストのオブジェクト・タイプ記号はT 16である．テキストには文字列フィールドの幅および高さがデータとして必要であり，それぞれwidth, heightで表される．また文字のフォントもデータとして必要である．

このようにpicture_markupクラスライブラリの各関数によって，TIPPLER APにて描画されたpictureのデータが対応するIL 5形式に変換されファイルに保存される（図3参照）．またこの変換の際，TIPPLER側ではグラフィック・オブジェクトの座標はピクセルを単位としているが，IL 5ではインチ系であるため，この単位の変換も行っている．こうして生成されるITデータ・ファイルをIL 5が取り込む．

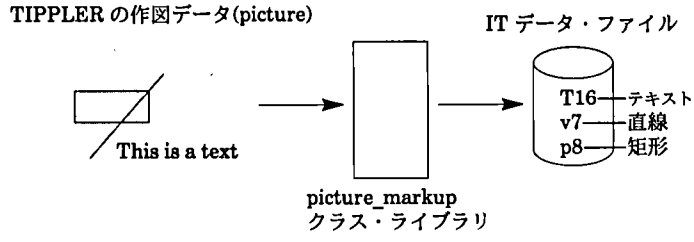


図 3　pictureデータのITデータ・ファイルへの変換

4.3 Interleaf側API

4.3.1 ITデータ・ファイルの取込み

TIPPLER APがITデータ・ファイルとしてIL 5形式のファイルを生成するため，IL 5ではフォーマット変換は不要である．IL 5側よりデータを識別する際，このデータ・ファイルを直接指示するのではなくIT管理ファイルを介して間接的にアクセスする．取込みが指示されIT管理ファイルが指定されると，IL 5はIT管理ファイルに記述されているデータ本体を読み込み，取込み指示がなされた文書にフレーム（IL 5文書内での作画領域）を作成し，その中にTIPPLER APで作成されたグラフィック・データを取り込む．この取込みの際IL 5側ではIL 5文書内での作画領域の大きさを指定するが，この作画領域の大きさに実際のデータをサイズ調整している．これによ

り TIPPLER AP とのリンクが完成する.

4.3.2 IT リンク機能を備えた文書クラス

IL 5 はオブジェクト指向の考え方に基づいて開発されたアプリケーションである. IL 5 のすべてのオブジェクトはあるオブジェクト・クラスのインスタンスとなっており, それぞれがクラスに応じたメソッドを持っている. たとえば IL 5 の個々の文書ファイルはデスクトップ (IL 5 の作業領域のこと) 上でアイコンとして管理されるため, デスクトップ・オブジェクトの一つとして分類されるが, デスクトップ・オブジェクトのクラス階層の中では dt-document-class に属し, open (文書を開く), close (閉じる), save (保存する) などの dt-document-class としてのメソッドを持っている. IL 5 のカスタマイズはこれら標準のクラス (階層上では親となる) に対し, 新しいサブクラス (階層上で子となる) を定義し, そのメソッドを拡張することで実現されることが多い.

今回 IT リンク機能を備えた文書クラスとして dt-document-class のサブクラスとなる it-document-class を定義した. it-document-class は通常の振る舞いに加え, 以下の 2 機能を拡張機能として持つ. その他のメソッドについては親クラスである dt-document-class のメソッドを継承する (図 4 参照).

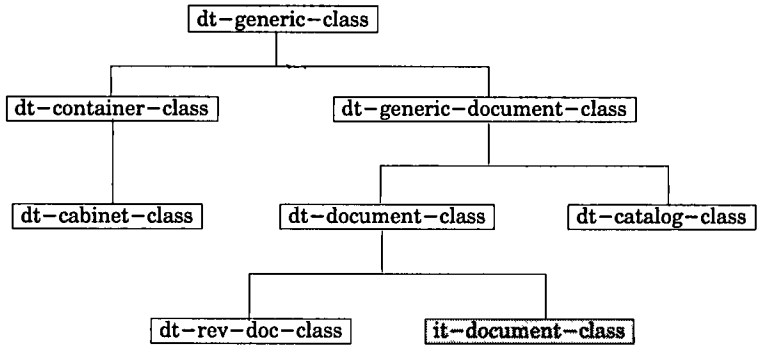


図 4 IL 5 デスクトップ・オブジェクトのクラス階層 (一部)

1) 拡張 open メソッド……ユーザ操作により文書が開かれる際に実行されるメソッドで, 文書中に TIPPLER データが含まれる場合にはその元データである IT データ・ファイルの更新日付をもとに改訂の有無を判断し, 必要なら該当のデータをリフレッシュする.
2) 拡張 close メソッド……ユーザ操作により文書が閉じられた際に実行されるメソッドで, 当該文書より IT リンクにより TIPPLER AP が起動されている場合, 文書を閉じる前にその TIPPLER AP の終了の監視を対象から外す.

4.3.3 IT リンクにより IL 5 から起動された TIPPLER AP の監視

IL 5 に取り込まれた TIPPLER AP のデータに対して IL 5 中から編集が指示された場合, IT 管理ファイルをもとに IT リンク・システムは, 該当のデータを作成した TIPPLER AP を自動的に起動しデータを再生する. この際, IL 5 文書内には複数の TIPPLER データが含まれる可能性があるため, 複数個の TIPPLER AP が起動され

ることも有り得る．動的なリンク保持のためには，起動した一つ以上のTIPPLER APのデータに対する更新を認知しIL 5側データのリフレッシュを行う必要がある．これをIL 5上でデーモン・プログラムとして実装した．ITリンクを含む文書からTIPPLER APが起動されると，IL 5上の管理テーブルにこの情報が記述される．このテーブルは起動されたTIPPLER APの数だけエントリを含む．これをITリンク管理テーブルと呼んだ．TIPPLER AP監視プログラムは，このITリンク管理テーブルを参照しその中に記述されているITリンクの複数のエントリを監視し，それぞれについて以下の処理を行う．

1) TIPPLER APが該当のITデータ・ファイルを更新した場合，その更新を認知し，対応するIL 5側のデータをリフレッシュする．
2) TIPPLER APがデータ更新後，あるいはデータを更新せずに終了した場合，ITリンク管理テーブルから該当のエントリを削除し監視対象から除外する．

この監視プログラムはIL 5セッション一つに対し，一つ起動される．この一つのデーモン・プロセスがすべてのITリンクのセッションを監視する（図5参照）．

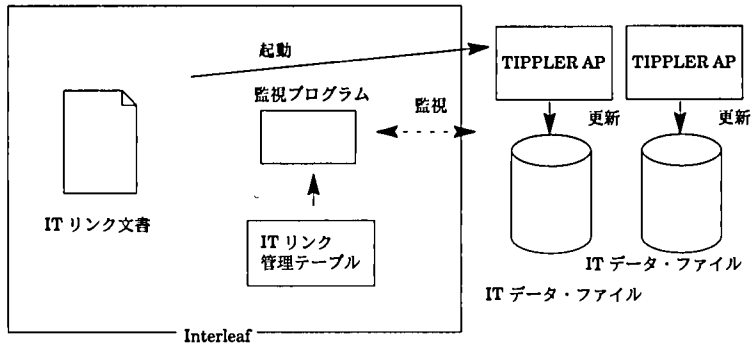


図5　IL 5におけるITリンク制御

4.3.4 ITリンク管理テーブル

IL 5文書とTIPPLER APとの対応関係を保持しているリスト形式の管理用テーブルである．ITリンク管理テーブル内の1件のエントリは1件のITリンク・セッションに対応し，それぞれ以下の情報を含む．

1) ドキュメントID
   IL 5中で文書を識別するための内部的な識別子である（IL 5上でのアドレス）．
2) フレームID
   IL 5文書内でTIPPLERデータが含まれるフレームを識別するための識別子．
3) ITデータ・ファイル名
   対応するITデータ・ファイル名称
4) ITデータ・ファイル最終更新日付
   ITデータ・ファイルの最終更新日付
5) TIPPLER APのプロセスID
   起動したTIPPLER APのIL 5上でのプロセス識別子

図6にITリンク管理テーブルの構造を示す．

このITリンク管理テーブルを利用しながら4.3.3項で説明したデーモン・プログラムが，IL 5側でのITリンクの制御を行う．

以上，ITリンク・システムのインプリメンテーションの実際をTIPPLER側，IL 5側に分け，説明を行ってきた．次章では本プロトタイプ・システムの評価と課題について述べる．

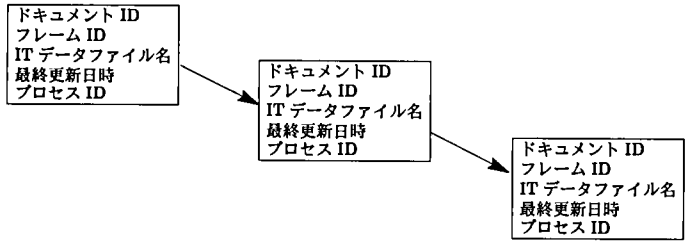


図 6　ITリンク管理テーブル

5. プロトタイプの評価と課題

このプロトタイプの目的は，Presentation ToolをモデルとしてTIPPLER APとIL 5連携の一形態として有効と思われるTIPPLER CASEツールとの統合の可能性を評価することであった．この意味ではTIPPLER CASEツールとの統合が現実的なレベルで可能であることが認識できた．特に，今回picture_markupクラスライブラリとして実装したTIPPLERのpictureオブジェクトのIL 5形式への変換のための関数群は，OLE的なリンク機能を除外しても，TIPPLERからIL 5へのデータ変換のツールとして非常に有効であると思われる．

ただし最終的なTIPPLER CASEツールとIL 5の統合に向けては解決すべき課題がいくつか残っている．主な課題としては次の3点が挙げられる．

1) picture_markupクラスライブラリの拡張

今回はプロトタイプということであり，また開発期間も短かかったためTIPPLERのpictureオブジェクトのうち一部のものしか対応していない．TIPPLERのCASEツールをターゲットとした場合，すべてのpictureオブジェクトに対応する必要は必ずしもないが，汎用的なライブラリとするためには対応を広げる必要があろう．これには技術的な障害は特になく，今回と同様の仕組みで実現可能と思われる．

2) IL 5でのITリンクに必要な管理情報の隠蔽

IL 5側でのITリンクに必要とされる管理情報が，現行のシステムではユーザに対しオープンになってしまっている．このため，ユーザの操作ミス等によりリンク情報が破壊される危険がある．これをガードする仕組みをIL 5側で実装する必要がある．プロトタイプでは管理情報の保持は，IL 5のコンポネント属性と呼ばれるユーザが設定可能な機能を用いた．しかし，ユーザからの情報隠蔽のため，IL 5のsaved-dataという機能を利用すれば可能である．この機能は，文書に含ま

れるデータとは別に，付加的な情報を格納するための機能で，これを用いれば通常のユーザ操作による情報の破壊を防ぐことが可能である．

3) 変換の精度向上

短期間での開発であったため，変換そのものについても精度的な問題が残っている．TIPPLER側で作成した際の線の太さや文字フォント・サイズに対するIL 5側の対応が完全ではない．これはpictureをIL 5形式のデータに変換する際にIL 5側での各種のパラメータ（線の太さ，フォント・サイズ）を標準的に決め打ちしてるものがいくつかあるためで，多様なデータを用いて現実的な変換のロジックを策定する必要がある．

上で挙げたようにいくつかの課題は残っているが，今後のTIPPLERのCASEツールとIL 5との統合の可能性・実現性を評価するという意味では，今回のITリンク・システム・プロトタイプ開発は十分な成果があったものと考える．

6. おわりに

オープン環境におけるシステム開発において，生産性の向上による納期の短縮は大きな命題であり，オブジェクト指向による開発方法論とツールとしてのCASEは今後ますます一般化すると思われる．また，CASEの概念を考えたとき，ドキュメンテーションは軽視できない重要なファクタであるため，特に下流のCASEツールとドキュメンテーション・ツールを統合的に利用できる環境の有効性は大きい．はじめにも述べた通り，英語製品ではSoftBenchとIL 5を統合するためのCASE connectという製品があり活用されている．弊社にはTIPPLERというオブジェクト指向の方法論に則った優れた開発言語があることから，システム開発の生産性向上のため，ドキュメンテーションまでも包含した統合的なCASE環境を提供することの意義は大きい．今後，今回のプロトタイプ開発をベースに最終的なシステムの構築に向け引き続き調査・研究を続けていきたい．

最後に，本プロトタイプ開発にご協力いただいた知識システム部の保科課長，渋谷課長，森西氏をはじめとする関係各位に御礼申しあげたい．

参考文献 [1] 土居範久編，「オブジェクト指向のおはなし」，日本規格協会，1995.
[2] 竹下　享著，「CASE概説—コンピュータ支援ソフトウェア・エンジニアリング」，共立出版株式会社，1991.
[3] 「Interleaf 5 Lisp Reference Manual」，Interleaf, Inc, 1991.
[4] 「The Document Management Guide」，Interleaf, Inc, 1994.

執筆者紹介　三　池　良　洋（Yoshihiro Miike）

1967年生．1990年東京都立大学人文学部卒業．同年日本ユニシス(株)入社．現在オープンソリューション部にてドキュメント管理システム Interleaf 5 関連製品の保守，利用技術研究，販売支援を主に担当．

TIPPLERを使ったオブジェクト指向技術に基づくアプリケーションシステムの開発

The Development of an Object-oriented Technology-based Application System Using TIPPLER

野村潤一郎

要約 TIPPLERは，オブジェクト指向プログラミング言語を中核とする，ソフトウェアの「開発と実行の環境」である．オブジェクト指向技術がソフトウェアの生産性，品質，再利用性などの向上に寄与するといわれている点について，開発の現状を評価するならば，TIPPLERによるソフトウェアの開発者は，未だにオブジェクト指向技術の恩恵に与かっていない．そこには，開発現場の実装技術の未熟さや，オブジェクト指向の問題以外に，教育体制や文化的な慣れなどの諸問題が山積しているため，実質的なオブジェクト指向技術の実践にまでに至っていないこともある．しかし最も大きな原因は，システム分析設計者が実装技術に立ち入らずにソフトウェアを設計していることによる，分析設計と実装の間の大きな隔たりにある．

各種オブジェクト指向方法論にいまひとつ実績の裏づけが無い現時点では，システム分析設計者が，ソフトウェアプロダクト・ライフサイクルの観点から「開発」の位置づけを見直し，「オブジェクト指向の方法論的な研究」と「オブジェクト指向技術の実践による方法論の検証」を行う必要がある．そのためにはまず，システム分析設計者自らがオブジェクト指向技術を適用したプログラム開発を実践し，方法論の検証を積み重ねることが重要であると考える．本稿では，TIPPLERによるオブジェクト指向開発の現状と問題を指摘し，対策について述べる．

Abstract TIPPLER provides a "software development and implementation enviroment" based on an object-oriented programming language. When the alleged contribution of object-oriented technology to higher software productivity, quality and reusability is considered in terms of how software is being developed at present, software developers who use TIPPLER have not benefited from object-oriented technology yet. That is partly due to the fact that such technology has not yet come into practical use because there are hoards of problems related to educational systems, cultural differences and the like other than the immaturity of their on-site implementation skills and problems associated with object-oriented programming. At any rate, the greatest cause is the existence of a wide gap between object-oriented analysis/design and programming, which stems from the designing of software by systems analysts/designers who refuse to step into programming techniques.

Today, when various object-oriented methodologies are not justified yet, if not all, by actual achievements, it is imperative that systems analysts/designers review the positioning of "development" from a software product life cycle aspect and "make methodological studies into object-oriented programming" as well as "verifying the methodology through the practicing of object-oriented technology". That makes the author think it important that those systems analysts/designers take the initiative in

implementing programs, to which object-oriented methodologies are applied, and repeatedly verify such methodologies. This paper is intended to discuss the current TIPPLER-based object-oriented applications development and problems involved as well as how to deal with them

1. はじめに

1991年5月に発売されたTIPPLERは4年を経，この間に多くのアプリケーションがTIPPLERによって開発されてきた．

筆者は20数年にわたってビジネスアプリケーションシステム（以降APとする）の開発に携わってきたが，1991年からTIPPLERによるオブジェクト指向ソフトウェア開発に関わるようになった．そして試行錯誤の結果，TIPPLERによるオブジェクト指向技術によって，ソフトウェア開発の生産性，品質，再利用性の向上が実現できるのではないかという期待を持つようになった．

しかし，開発の現状は遅々として変わっていない．その主な原因は，オブジェクト指向分析設計とプログラム実装との間の大きな隔たりにある．オブジェクト指向の黎明期にあって，構造化手法成熟期の，分析設計工程と実装工程の作業を分担するという習慣が，オブジェクト指向技術の検証や蓄積をはばみ，ソフトウェア開発の変革を遅らせる結果となっている．

方法論と実践が噛み合っていない現状に対し，TIPPLERによるオブジェクト指向開発の現状と問題を指摘し，対策の提示によって，現在の停滞状況を打破する一助としたい．

2. 分析設計の現状と問題

TIPPLERを利用してAPを開発するシステム分析設計者は，"オブジェクト指向をやるぞ"と言って開発に取り組んでいる訳ではない．むしろ多くの場合，20年以上もどっぷりと構造化手法に浸かっていたベテランの人達が，TIPPLERの採用により，突然オブジェクト指向をやる羽目になっているのが実状である．それまではウォーターフォール型の開発形態で，構造化手法に基づき，COBOLのような手続き型言語でゴリゴリとプログラムを開発していた人達ばかりである．当然彼らはTIPPLERを，これまでの手続き型言語と同じように，単なるプログラミング言語の問題としてしか考えていない．ほとんどのシステム分析設計者は「分析設計と実装は切りはなすものだ」として，「自分は実装言語（TIPPLER）には立ち入らないでいいのだ，立ち入るべきでない」という姿勢を取っている．そして多くのシステム分析設計者は，開発上流工程の方法論として設計方法論やCASE，オブジェクト指向データベース，モデリングなどのオブジェクト指向分析設計論を学習している．

しかしその一方で彼らはまた開発事例やプロジェクト事例，大規模開発事例，実用化方法，導入方法などの「実績の情報」を求めている．このことは，実際のシステムの開発に際して，方法論だけでは不安であることを示している．

このように，システム分析設計者はオブジェクト指向分析や設計の方法論を「知識」として知っていながら，結局，全く従来通りの方法で分析や設計を行うことが多い．なぜそうなってしまうのかを以下で述べる．

2.1 オブジェクト指向方法論の現状

方法論と実践との格差を明らかにするために，オブジェクト指向の要点を確認しておく．数十種類あるオブジェクト指向方法論の共通するところを挙げてみる．

1) オブジェクト指向とは「思想」であり，「物の考え方」である．
2) オブジェクト指向分析（OOA）では現実世界を構成する「モノ」に着目する．
3) オブジェクト指向設計（OOD）では「モノ」を主体に，現実世界を自然な姿でコンピュータの世界にモデル化（写像）する．また，ソフトウェア構成要素をオブジェクト単位に部品化し再利用する．
4) オブジェクト指向プログラミング（OOP）では抽象型変数，カプセル化，情報の隠ぺい，継承，ポリモフィズム，クラスライブラリ化などの技術を基盤とする．
5) 開発作業はスパイラルアップ型で行い，ソフトウェアプロダクト・ライフサイクルの観点から各工程間のギャップを少なくする．また，プロトタイピング手法を導入し，速い回転で仕様の明確になった部分から開発を進める．

2.2 システム化対象物の違い

システム分析設計者がオブジェクト指向技術の適用を難しいと考える最初の理由は，調査対象の違いにある．たとえば実際の開発の開始時に，「オブジェクト指向調査（OOR）というのがあるのか」という疑問に突き当たる．

従来の構造化手法に基づく要求調査では，開発者は，依頼者が明確にできないシステム開発の目的や「機能」などを明らかにする．この過程でシステム分析設計者は，業務フローやデータフローダイアグラムなどから，機能の抽出や分析を行う．

ところがオブジェクト指向分析設計の方法論では「システムの機能を考えるな．何をオブジェクトとするかを考えろ」とか，「機能を考えるのは遅いほどよい」といっている．

通常のAPの開発は人事管理とか営業支援といったように，企業の特定業務を対象とする．その範囲内でシステム開発の期間と工数が定められ，ヒアリング対象者が選定される．ヒアリングでシステム分析設計者がエンドユーザから最初に聞き出すことは「何（機能）を望むか」であって，「何がオブジェクトか」ではない．開発を依頼する側は欲しい機能についてヒアリングに応じるのであって，そこにあるモノについて話し合いをすることはない．エンドユーザが要求する機能に触れずに，オブジェクトに注目するようなヒアリングは非常に難しい．

2.3 オブジェクトの認識と抽出

ベテラン分析設計者が次に突き当たる壁がオブジェクトの認識と抽出の作業である．

オブジェクト指向分析設計の方法論では一様に「システムの機能」に注目しないで，「対象業務分野のオブジェクト（モノ）」に注目しろ，と説いている．モノの認識は人間にとって自然で容易であるとしながら，具体的なオブジェクトを抽出する方法としては「名詞に注目しろ…云々」とか，「分析設計者の個々の直感を働かせて」など，システム分析設計者の個々の認識の問題に転化している．

モノの認識は，個々の人間が長い時間を掛けて築き上げてきたものであるため，誰にとっても極めて自然な方法ではあるが，一人ひとりの多様な成長過程に依存するこ

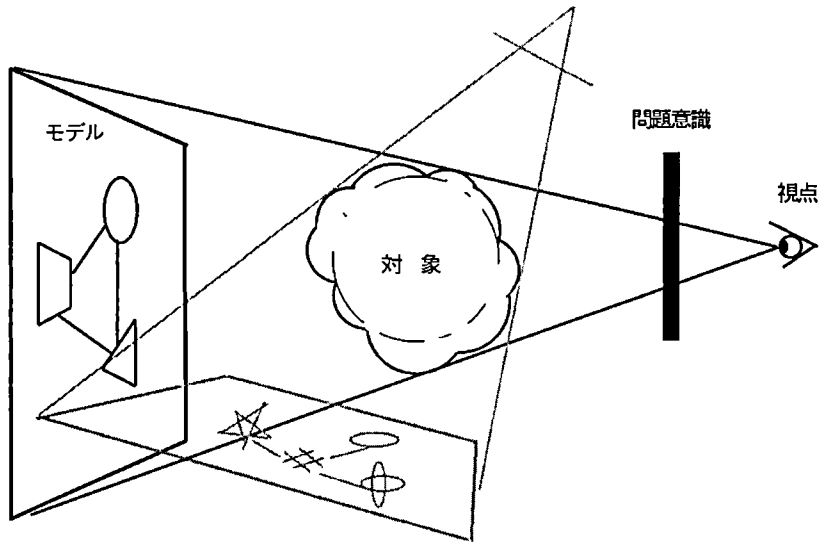


図 1 オブジェクトの認識

とになる．したがってモノの認識は非常に主観的であり，個人の問題意識や置かれた環境に大きく左右される（図1）．

形のあるものであっても，問題意識や視点の違いによって「モノ」の認識が異なることから，サービスや約束，関係といった形のないモノについては，より大きな認識の違いが発生し得る．モノに対する認識の違いは，システム化対象物(オブジェクト)の違いとなり，オブジェクトの違いはシステム構成物の違いとなる．

2.4 プロトタイピング

同じく慣れていない作業にプロトタイピング手法の導入がある．TIPPLERによるAP開発でプロトタイピング手法が導入されていく経緯は以下の通りである．

TIPPLERによるソフトウェア開発の特徴の一つに，グラフィカルユーザインタフェース（以降GUI）機能に富んだAPの開発がある．GUIアプリケーションの開発では多くのケースで，画面のイメージが要求仕様の資料として先行する．エンドユーザが最初に説明すべき「会社概要」や「業務概要」を差し置き，先に画面イメージを要求として提出する場合が少なくない．また，開発者側もエンドユーザの要求を，より具体的に聞き出すことができる材料として，実物に近い画面イメージを作成してヒアリングに臨むことが多い．

提示される画面イメージには，ヒアリングの進行と共に次々と変更が加えられていく．そのため手書きやワープロ，ペイントソフトウェアのような再利用の効かない媒体では，画面イメージの書換の工数が大きくなるため敬遠される．一方TIPPLERでは画面イメージの実装が比較的容易であり，かつ実物であり，再利用が可能であるため，Uniscriptで実装した画面をプロトタイプとしてエンドユーザに提示している．

さらにエンドユーザが，より具体的になった詳細画面や画面の動きを了承すると，開発者もエンドユーザも，システムの骨格が決まっていない要求定義段階で実装したプログラムを，本番用システムにまでレベルアップできると錯覚してしまう．

しかしこのような経緯の結果，開発当初意図しなかったスパイラルアップ型開発に入り込み，プロトタイピング手法が絡みあって，システムの基本仕様が曖昧なまま，画面イメージだけで，本番用システムの実装を進めてしまう．結果は，終わりのないスパイラル型開発の悪循環に陥ってしまう．

3. プログラミングの現状と問題

次にプログラムの実装におけるオブジェクト指向開発の現状と問題について述べる．

TIPPLERによって開発されたAPは概して好評である．とくに計画立案支援のようなシミュレーション系APや，苦情処理や営業支援のようなナビゲーション系APにはエンドユーザの評価の高い事例が多い．ユーザの評価だけから判断すると，TIPPLERによるAPの開発がうまくいっている様に見える．

しかしその開発経過や最終成果物であるシステム仕様書やプログラムをオブジェクト指向の観点から評価すると，高い評価は必ずしもオブジェクト指向技術導入の成果ではない．オブジェクト指向技術がいうところの生産性や品質，再利用性，が高い訳でもない．以下にTIPPLERによるGUIアプリケーションの代表的開発形態と，オブジェクト指向技術の問題について述べる．

3.1 GUI構築ツールとしてのTIPPLER

TIPPLERには図2に示すように4つの主要な機能があるが，TIPPLERによって開発されたAPが概して好評な理由は，TIPPLERの「GUIを実現する機能」によるところが大きい．

保科[1]によればUniscriptの設計概念として，「オブジェクト指向技術についてはオブジェクト指向言語として必要な機能の採用に止め，GUIアプリケーション構築における生産性の向上に注力する．」としている．このGUI実現の機能は，CADやCG以外に優れたマンマシンインタフェース構築実績のないワークステーション/UNIXの世

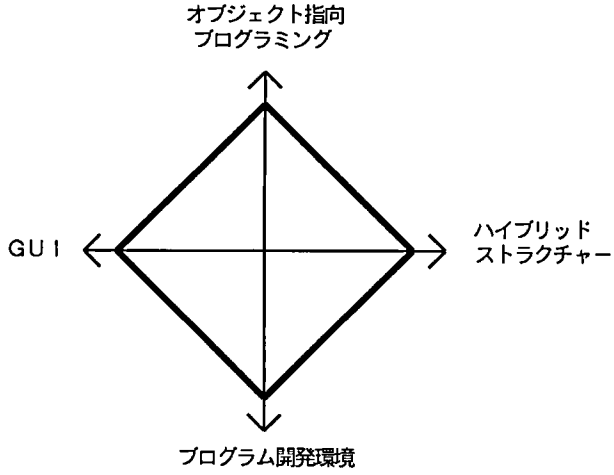


図 2　TIPPLERの機能

界で，APを開発するのに最も大きな効果を発揮している．GUIアプリケーション実装作業の実態は以下の通りである．

1) 画面レイアウトのみの仕様

要求仕様または設計仕様として詳細画面のレイアウトが提示される．このレイアウトの多くは，たとえば生産計画立案者が実際に計画を立てる際に紙や頭の中に描いたり，電卓や表計算ソフトで計算し確認している普段の作業のイメージを基にしている．

2) プログラム実装者による分析と設計

プログラム実装者は詳細画面レイアウトを分析し，画面を構成する属性（データ項目）を定義する．次にプログラムとして実現するために，画面の背景にあるはずのクラスを想定し，クラスの構造と画面のレイアウトを実装する．

3) スクリプト言語によるGUI中心のプログラム実装

リレーショナルデータベースのような外部データベースが設計されていない段階では，簡便的にテキストデータを使用してインスタンスを生成する．クラスの定義からインスタンスデータの作成までは，Uniscript言語のGUI実現の機能によって，極めて簡単に実装することができ，エンドユーザのイメージに沿った詳細な画面を表示することが可能になる．

4) プロトタイプ

エンドユーザがイメージした主要画面を，ある程度プログラムで表示できるようになると，プロトタイプと称して，エンドユーザに見せる．エンドユーザは画面のみのプロトタイプを提示されると，その画面が実際の業務のイメージに近く，しかも要求から実現までが短期間であることから，開発作業に好意的かつ積極的になり，より詳細な要求仕様に言及するようになる．

5) 口頭による機能の説明

さらにプログラム実装者はシステム分析設計者から各画面に求められる機能の説明を口頭で受ける．この中には，複数部門間にまたがる業務フローやデータフローダイアグラムのようなシステム概要仕様のレベルのものから，業務のノウハウのような知識ベース，さらには数値項目の入力データチェックのような子細なプログラム仕様レベルのものまで，様々な内容が入りまじっている．プログラム実装者はこれらの雑多な機能を理解し，分類整理してアプリケーションの画面に対応するクラスに，手続きとして割り当てる．

この様にGUIアプリケーションの開発では，システムの基盤を定義しないまま，システム分析設計者と実装者の会話を基に，プログラムを実装していく．

3.2 GUIアプリケーションと生産性の評価

UniscriptによるGUIアプリケーションプログラムの開発は，生産性や開発工数などの面から，C言語とXライブラリ，GUIツールキットによる開発と比較されることが多い．

保科[1]によればUniscriptでは「GUIアプリケーションの生産性を上げることを主眼にGUIをモデル化し言語仕様に採り込んだ」としている．この点について，画面を中心としたGUIアプリケーションの開発では確かに生産性を上げており，システム開

発者の評価も良い．

しかしここで評価されている高い生産性は，あくまでもUniscriptのGUI実現を重視した言語特性によるものであって，オブジェクト指向技術をプログラム開発に適用した結果によるものではない．以下にその理由について述べる．

3.3 プログラミングとシステム仕様書

TIPPLERによるプログラムの実装は多くの場合，システム分析設計者がオブジェクト分析設計を模索した結果を受けて行われるか，あるいは全くの従来手法で行われる．そのため，システム仕様書やプログラム仕様書が全く作られていないか，または作られていても，オブジェクト指向プログラミングに合っていないことが多い．いずれにしてもUniscriptによるプログラムの実装は，実装者の個人的資質に大きく依存することになる．以下にシステム仕様書の無い場合とある場合の，それぞれの問題を述べる．

3.3.1 システム仕様書の無い場合

システムに関する記録については従来から，その重要性が叫ばれながらも，「今，動いているのが仕様」とか，「プログラムが最新の記録」といわれてきているのが実態である．TIPPLERによる開発の事例でも仕様書を作らない場合が多い．先に述べた「GUIアプリケーション開発の例」は仕様書のない典型的な例で，システム分析設計者とプログラム実装者が一体となってプログラムの設計と実装を展開していく．

仕様書がなく，システム分析設計者にUniscriptの知識がない場合，プログラムすなわちシステムの内容は全面的に実装者にまかされてしまう．この場合，プログラムの実装に関して次の二つの点が問題となる．一つはオブジェクト指向に未熟なプログラミングの問題であり，もう一つは対象業務の理解の問題である．

1) オブジェクト指向に未熟なプログラミングの問題

保科[1]によれば「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言語の特徴を採用することにした．」とある．したがってUniscriptはオブジェクト指向言語であるが，純粋なオブジェクト指向言語ではない．手続き型言語の特徴も含んだ，いわゆる準オブジェクト指向言語である．

Uniscriptが準オブジェクト指向言語であることは，システム仕様書がない場合のプログラム実装作業だけでなく，作られたプログラムの品質や再利用性にまで大きな影響を与えている．

ビジネスアプリケーションの開発では非常によくあるケースであるが，プログラム実装者がCOBOLなどの手続き型言語に馴れている場合，抽象型変数や継承，ポリモフィズムなどの不可解なオブジェクト指向プログラミング技術を回避し，Uniscriptの手続き型機能を多用することになる．たとえば，クラスは処理対象の単なるデータファイルの代わりとなり，処理に必要な中間ファイルのために，本来システムに必要でないクラスが追加され，インスタンスの複写が安易におこなわれるようになる．さらに画面表示のためにプログラムに必要なクラスをシステムに追加し，コピーしたインスタンスの整合性を保つために，多くのデーモンが必要になる．画面表示のためのデータ構造が優先され，オブジェクトを基盤と

するデータ構造はなくなってしまう．手続きの多くは最上位のクラスで行われ，属性（データ項目）の取扱いには参照が頻繁に使われる．

その結果，プログラム中のクラスはオブジェクトとは無関係となり，データ構造と手続きのカプセル化は見かけだけで，情報の隠ぺいは崩れ，一つまたは複数のクラスからなるオブジェクトレベルの部品化（クラスライブラリ化）は不可能となる．したがって，ソフトウェア部品の再利用もなし得ないし，処理効率も悪くなる．プログラムは極端に肥大化し，不整合が発生し，品質が落ちる．

2) 対象業務の理解の問題

プログラム実装者がオブジェクト指向プログラミング技術に熟練している場合であっても，クラス構造やクラス定義がないため，本来システム分析設計者が行うべきオブジェクトの認識やデータの正規化などの作業を行わなければならない．さらにプログラムがシステムとしての機能を提供するためには，オブジェクト間の関係やメッセージの授受について分析設計しなければならない．これはプログラム設計であると同時にシステム分析設計に他ならず，プログラム実装者の仕事ではない．

しかも従来からシステム化の対象となっていた，人事管理，財務管理，物流管理，などの管理系システムは，プログラム実装者が分析設計するには規模が大きすぎる．

また，Uniscript は高水準言語で，従来の手続き型言語に比べて数十倍の記述性があり，また可読性が高いため，COBOL 換算で 50 万ステップ程度の一つのシステムを，一つのプログラムで実現することが可能である．そのためプログラム実装者は要求される機能を全て一つのプログラムに組み込んでしまいがちである．

その結果，システムは適切なモジュールに構造化されず，数百のクラスを抱えた巨大プログラムとなってしまう．

3.3.2 システム仕様書のある場合

GUI の実現をめざしたアプリケーション開発であっても，従来型の工程と仕様を踏襲する場合，画面レイアウトの他に外部データベースや，通信，出力帳票などの仕様書が作られる．

図 3 に従来型の仕様書作成手順を示す．

1) 画面や帳票など出力仕様の調査から詳細設計まで行う．
2) 出力仕様を基に入力仕様の分析から詳細設計まで行う．
3) 出力と入力を結び付ける手続きを設計する．

20 年間変わっていないと思われるこの方法は，TIPPLER によるオブジェクト指向

図 3 従来型の仕様書作成手順

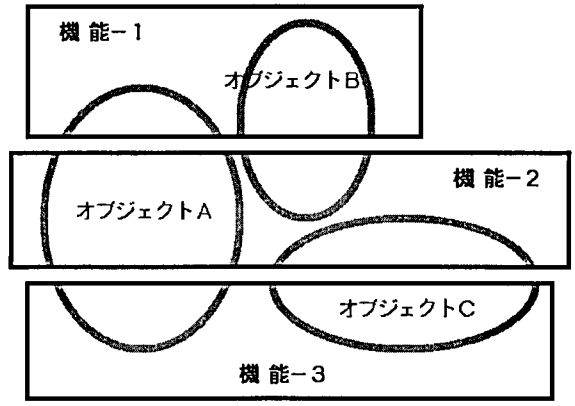


図 4　オブジェクトの分断

開発にとって致命的な問題を抱えている．

1)　画面や帳票の詳細設計が先行することによって，対象物であるオブジェクトの認識作業は放置されてしまう．

2)　データベース設計前の入出力データの調査と詳細設計によって，プログラム処理用の中間ファイル（クラス）の存在を暗黙の前提としてしまう可能性がある．

3)　システムは，細分化された機能に基づいてプログラムレベルで分割されてしまうため，オブジェクトは細分化された機能によって分断されてしまう（図4）．

4)　外部データベースはインスタンスデータの永続保存と管理のために設計されるため，プログラム中に実現されるクラスは単なるファイルにすぎない．

プログラムの実装者はシステムの仕様書に忠実であればあるほど，カプセル化，継承，ポリモフィズムといったオブジェクト指向プログラミング技術を発揮できなくなる（必要としなくなる）．これはプログラム実装者の責任ではない．

分断されたオブジェクトはクラスライブラリとして成り立たない．その結果，ソフトウェアの部品化はできず，再利用性や品質の向上も果たせない．従来と同じように，保守や改造に開発以上の工数の掛かるシステムができ上がってしまうことになる．

このように，仕様書がある場合でも，オブジェクトの認識なしに入出力の詳細が設計されてしまう開発手順は，オブジェクト指向開発にとって致命的である．

4. 対　　策

2章，3章でTIPPLERによるオブジェクト指向開発の現状と問題について述べてきた．しかし筆者の関与した開発で，オブジェクト指向技術を意識的に利用しようとした事例は1件しかなかった．その他の多くの事例では，オブジェクト指向技術を意識する余裕もなく開発に追われている．

ソフトウェアプロダクト・ライフサイクル（図5）の観点からソフトウェアの部品化や再利用，生産性を考えるオブジェクト指向の考え方は，ライフサイクルの一部である，開発（分析/設計/実装）だけを担うこれまでの考え方にとって，工期や工数，開発能力などの開発管理の面でそぐわないところが多い．

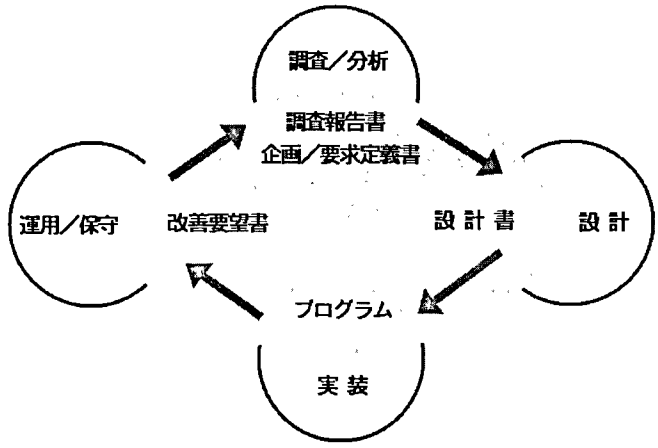


図 5　ソフトウェアプロダクト・ライフサイクル

以下にこれまでの開発事例から得た TIPPLER によるオブジェクト指向開発の，開発管理と分析設計における反省と実践的な対策を述べる．

4.1　オブジェクト指向開発と開発管理

1)　開発リーダの条件

AP の開発においては，たとえ部分的であっても，TIPPLER を利用する場合，開発リーダを含めた全ての開発要員が，TIPPLER によるオブジェクト指向開発を知っていなければならない．特に開発リーダやシステム分析設計者は，TIPPLER の主要機能を熟知していることは当然であるが，Uniscript についても，実装レベルで知っている必要がある．

オブジェクト指向開発における分析設計と実装の間の大きな隔たりは，システム分析設計者にオブジェクト指向プログラム実装技術の無いことに起因している．実装作業が省力化され，分析設計と実装が速い周期で入れ替わる開発では，分析設計者も実装技術を知っている必要がある．開発の上流工程で TIPPLER を意識しなかった結果，設計した仕様が実装ができなかったり，困難となったりして，仕様変更や開発工数が増大した事例もある．

開発リーダやシステム分析設計者は，オブジェクト指向に基づく分析設計から実装までの全ての開発工程を経験することによって，開発管理や分析設計などにおける諸問題への対応の心構えができる．

2)　見積方法

TIPPLER による開発に対し，ファンクションポイント法による見積りに熱心に取り組んでいるグループもあるが，現在のところ適切な見積方法はない．とくにシステム提案時点では，開発物，開発方法，開発能力がこれまで以上に不透明であるため，見積りは TIPPLER による開発経験のある人の，勘と経験と度胸にたよらざるを得ない．

3)　開発体制

開発の一部で C 言語を使う場合も含め，TIPPLER によってのみシステムを開

発する場合，開発要員は多くても10人以下の小集団が望ましい．

4)　開発要員

開発要員には工程別専任的資質ではなく，全工程の作業をこなせる資質が求められる．それらの条件が満たされない場合には，開発開始初期から全工程に対応する資質を育成する予定を組むと良い．たとえば実装言語の経験しかない要員であっても，エンドユーザが認識するオブジェクトの理解のために，ヒアリングなどの調査作業から参加させるようにする．また，システム開発経験の少ないプログラム実装者に対しては，構造化手法の習得機会を設定する．

5)　スパイラルアップ型の採用

スパイラルアップ型で開発を行う場合は，調査から検収までのサイクルを，2〜3週間程度の速い回転で並行しながら繰り返す（図6）．

開発は，システムの骨組みから枝葉へと進め，明確になった部分から小回り良く次の作業へ移行する．不明確な部分の解明に長時間を費やさずに，次回の作業に回す．スパイラルアップ型の開発で管理する対象は，開発工程ではなく，システム化対象物の明確になっていく度合いである．

6)　開発ツール

現在のところAPのオブジェクト指向開発全体をカバーする適切なツールはない．TIPPLERによるAPの開発では，TIPPLERによって開発ツールを作成し，開発環境を整備することを推奨する．TIPPLERは「人間の知的な活動を支援するソフトウェア」を作成する道具である．システムを開発することが知的な活動の一つであるとすれば，システム開発者は当然自らの活動を支援するソフトウェアをオブジェクト指向に基づいて，自然に楽に作ることができるはずである．

この様な試みによって，開発リーダやシステム分析設計者は，ソフトウェアプロダクト・ライフサイクルのすべてのフェーズで，オブジェクト指向技術の適用

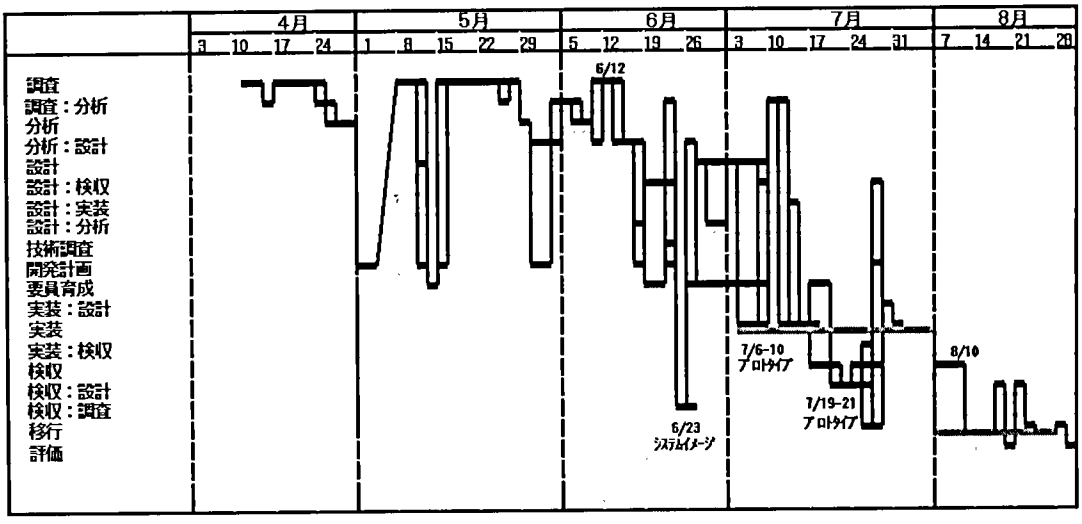


図 6　スパイラルアップ型開発事例

を実践できるとともに，オブジェクト指向方法論の検証によって，より実践的な技術の蓄積と共有化が推進できると考える．

7) オブジェクト指向に基づく開発の実践と記録

これは本稿の主張であるが，なによりもオブジェクト指向に基づくAPの開発を実践することである．これなくしては何も始まらない．さらに，できるかぎり作業の実績や経緯を電子的に記録し，実践の結果と方法論を照合する．これに基づいてより良い技術の集積と共有化を計り，オブジェクト指向技術によるソフトウエア開発の生産性，品質，再利用性の向上を試みることである．

4.2 分析と設計

1) オブジェクト指向技術の適切性を検討する

APの全てをオブジェクト指向で開発するすることは，困難であると判断している．したがって開発リーダはシステムに求められている目的を十分に理解するとともに，オブジェクト指向プログラミング技術と利用する言語の特性を十分に理解し，開発物と開発技術や道具の適切性を検討する必要がある．

システム化の対象業務は図7のようにタイプ分けすることができる．

これらのタイプ分けによって，開発方法の適切性を判断することができる．TIPPLERは営業支援システムのようなナビゲーション型や計画立案支援のようなシミュレーション型のAP開発に適している．

2) 業務モデルとモジュール分割

企業にはそれぞれの経営方針に基づいた独自の組織や業務モデルがある（図8）．

大規模システムの開発は，オブジェクト指向による実績が少ないため，従来，構造化手法でシステムを構築してきた経験者の知識が必要である．システムを従来の構造化手法に基づいて，業務モデルを適切なモジュールに分割し，小集団で開発可能な規模にする必要がある．

3) プロトタイピング

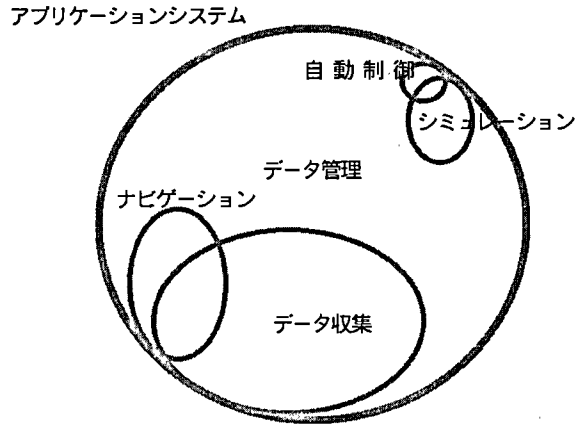


図7 システム化対象業務のタイプ分け

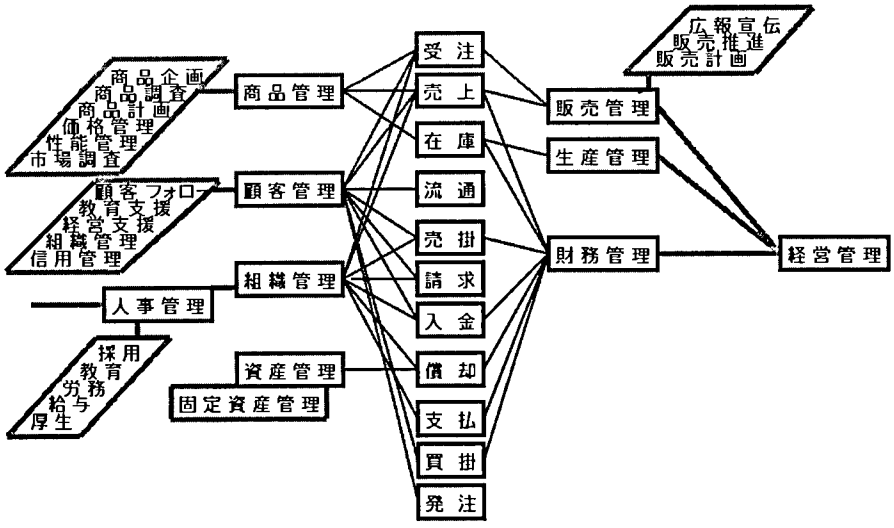


図 8　業務モデル事例

要求調査では，プロトタイピング手法の導入が，有効であるか否かを判断する．一般的にシミュレーション系やナビゲーション系のAPは，エンドユーザの判断(業務知識)がシステムの有効性を決定づけるため，プロトタイピング手法の導入は効果的である．しかし処理手順が明確になっている業務の場合，あまり効果はない．

プロトタイピング手法には，プロトタイプを育成（機能追加）しながら本番システムを作り上げる方法と，プロトタイプをあくまでもエンドユーザから要求を聞き出すのための材料として利用する方法がある．

TIPPLERによるAPの開発ではGUI実現の開発が多い反面，TIPPLERのWYSIWYGの特性などから，プロトタイプを育成していく方法は推奨しない．

システムの開発はよりエンドユーザ主導型の傾向にあり，エンドユーザに対する要求調査がシステム開発成功の鍵となっている．

TIPPLERによるAPの開発においても，プロトタイプはエンドユーザとのコミュニケーションを円滑にし，エンドユーザの開発に対する参加意識を高めると同時に，ユーザ自身の業務に対するより深い学習をもたらす可能性がある．

したがって，プロトタイピング手法の導入は，システム開発における位置づけ(図9)や目的，期間や工数などを明確にした上で導入すれば，非常に効果がある．

4)　オブジェクトの認識

一人でオブジェクトの抽出を行った場合，個性の強いシステムとなる．また，複数の場合，システム分析設計者の間で認識の違いが発生する．この違いを一致させる作業は非常に労力を必要とするため，完璧な一致のための努力は推奨しない．

最終的にはモデルの定義であるので，少数（2〜3人）のシステム分析設計者によるオブジェクトの抽出と，協議による合意の必要性を認めることが現実的な方

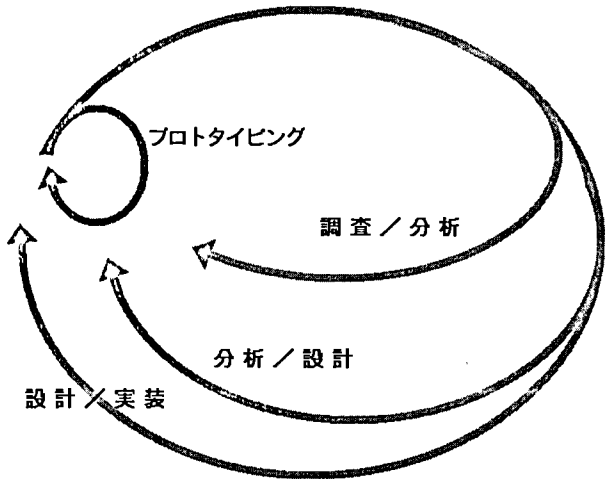


図 9　プロトタイピング位置づけ

法である．

5)　クラスライブラリ，部品化，再利用，非再利用性

AP を開発する場合，整数や実数といった基本的なクラスライブラリは細かすぎてあまりメリットを感じない．AP 開発の規模や機能を考慮すると，システムの構成要素となる中規模 AP 程度のライブラリが，作らなくて済む，実績があれば安心などの現実的なメリットを感じる．しかし現時点で，世の中にはこのような規模のクラスライブラリは販売されていない．

一方，一つの企業の中で，クラスライブラリが他の AP で再利用されるだろうか，という疑問もある．理由は，企業内の業務モデルでは「業務」は重複していないはずであり，一つの「モノ」に対し，複数の異なる業務が重複して同じ働き掛けをすることはないはずだからである（図 8）．

再利用の可能性として，同一企業内で部門レベルの AP の横展開やカスタマイズの場合が考えられる．また情報システム部門やソフトウェア産業における，システムの保守や改良では再利用が生産性の向上に大きく貢献すると考える．

6)　システム仕様書

TIPPLER によるオブジェクト指向 AP 開発では，システム仕様書とプログラム仕様書およびプログラムの境界が不明確である．しかしシステム分析設計者は以下の範囲について責任を持つべきであると考える．

①　システム概要の定義
②　システム化対象物（オブジェクト仕様）の定義
③　システムのモジュール分割
④　各モジュールの定義
　（クラス，大域変数，列挙などモジュール全体にかかわる定義）

TIPPLER により AP をスパイラルアップ型で開発する場合，従来のように完璧な設計書を作成してから一斉に実装作業に入れるとは限らない．概要から詳細

へと，明確になった部分から仕様を整備していく．

5. おわりに

実践による理論の検証は科学的な態度として情報処理技術にも欠かせないものでありながら，構造化技術の成熟とAP開発組織の肥大化が，システム開発技術の空洞化をもたらしている．AP開発者が，方法を考え，実行し，検証する本来の姿勢に立ち戻るならば，オブジェクト指向技術の蓄積と共有化が推進できるとともに，ソフトウェア開発のあり方を変えることができると考える．

本稿ではできるだけ多くの事例を基に，オブジェクト指向開発に共通する実態を述べている．しかし意識的なオブジェクト指向開発は極めて少なく，実践を基盤にした方法論の検証を行うにはまだまだ不十分であった．

また，本稿では個別の事例を引用したわけではないが，これまでに多くの開発で一緒に試行錯誤して下さった方々，情報を提供してくださった方々に心から感謝するしだいである．

参考文献　[1] 保科　剛　著,「統合開発支援ツールTIPPLER」, 技報第38号, 日本ユニシス, 1993年8月.

[2] Roland Vonk 著, 黒田純一郎　訳「プロトタイピング」共立出版株式会社,1992年9月1日.

執筆者紹介　野　村　潤　一　郎 (Jun-ichirou Nomura)

1945年生, 71年横浜市立大学文理学部心理卒業. 同年日本ユニシス(株)入社. 人事部, 財務部, 管理システム部, CGシステム部, 製造工業システム部をへて主に業務システムの開発と技術支援を担当. 現在, システム技術本部知識システム部に所属.

マルチメディアとTIPPLER

Multimedia and TIPPLER

佐々木 勝信

要 約 A社がその技術検証の一環としてマルチメディア指向の営業窓口支援システムの開発を行った．開発に用いたのは日本ユニシスの開発言語のTIPPLER for Windowsである．

システム構築は，OLE，DDEといったWindowsアーキテクチャを利用することで，容易に実現することができた．OLEはマルチメディアデータの表示に，DDEはホストとのデータ連携に利用している．

本稿では，上記システムの開発を例に，TIPPLER for Windowsを用いたマルチメディアシステムの構築方法と，その可能性について述べる．

Abstract Company A has developed a multimedia-oriented, user service support system to verify part of its own information technologies. This implementation entailed the adoption of "TIPPLER for Windows", an object-oriented programming tool developed by Nihon Unisys, Ltd.

The use of Windows architectures such as OLE and DDE helped make it easy to create the system. The former (OLE) was used for the display of multimedia data and the latter (DDE) for data communications with the host computer.

By referring to this development effort, this paper describes a methodology for building a multimedia system based on "TIPPLER for Windows" and explores its potentialities.

1. はじめに

ここ数年のコンピュータの進歩には目を見張るものがある．高速なCPU，大容量の記憶装置の登場，高速なネットワークの発展，GUIを主体としたOS，挙げればきりがないほどである．それら技術革新はPC/AT互換機と呼ばれるパソコン(以下，PCと記述)を中心に発展し，以前とは比較にならないほど高性能なPCが安価で市場に登場した．高性能なPCが普及していくことで，CPU処理能力の違いから，エンジニアリング・ワークステーションクラスでしか実現できなかった高度な処理，たとえば画像処理，音声解析がより身近に扱えるようになった．当然ながら，あちらこちらから色々な利用方法が出てきた．印刷処理を主体にしたDTP (DeskTop Publishing)，ビデオ編集を行うDTV (DeskTop Video) などが例に挙げれる．それらを基にマルチメディアという言葉が登場した．最近では，インターネットの人気上昇に伴い，マルチメディアブームとなっており，その実用性を本格的に研究している企業も多い．本稿で採り上げたA社も，マルチメディアの実用化を研究している企業の一つである．

今回，A社と「営業窓口業務のマルチメディア化」について技術検証を行った．この検証はA社の関連会社であるB社が主体になって行い，弊社(日本ユニシス)は機種の選定，開発の支援を行う事となった．開発機には技術革新の目覚ましいPC/AT互

本稿に記載の会社名，商品名は，一般に各社の商標または登録商標である．

換機，OS を MS-Windows 3.1 (以下，Windows と記述) に決定した．また，開発言語には弊社の TIPPLER for Windows (以下，TIPPLER と記述) が採用された．

本稿では，「営業窓口業務のマルチメディア化」を通して，TIPPLER を用いたマルチメディアを扱ったシステムの構築方法と，その可能性について述べる．

2. 営業窓口業務について

2.1 営業窓口業務の概要

A 社の営業業務は，その業務内容・業務特性から以下のように分類できる．

1) 基幹系業務

A 社の基本業務として，受付～集金にいたる一連の定型業務および，それらに関わる問い合わせ対応業務 (図 1)

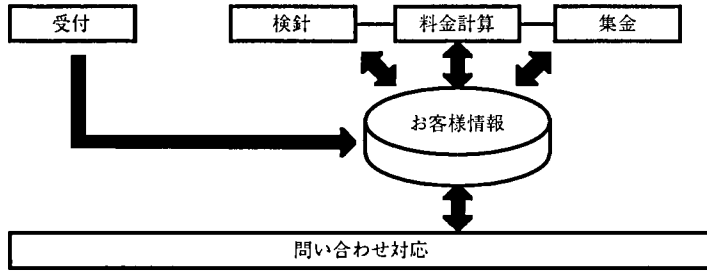


図 1　基幹系産業概要

2) 新規サービス業務

企業イメージ向上や地域への貢献を目的として，A 社の基本業務の理解と顧客満足度の向上や，付加価値サービスへの業務拡大を行うための業務 (図 2)

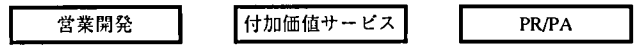


図 2　新規サービス業務概要

その中の窓口業務となるのは「受付」「問い合わせ対応」などで，これらの窓口業務は回答レスポンスの早さや回答の正確さが要求される．しかし，それらは窓口担当者の熟練度に左右されるため，その回答精度は様々なのが現状である．

2.2 営業窓口業務へのマルチメディアの適用

現行システムはホスト集中型のシステムで，担当者はダム端末を使って作業を行っている．現状の問題点としては，

1) システム操作の拡張性が低い

現行システムはダム端末上で稼働しており，メニュー方式とコマンド方式が用意されている．コマンド方式は習得に時間がかかるがレスポンスが早いため，熟練者はメニュー形式をほとんど利用していない．

2) コード入力方式のためデータ入力が面倒

銀行名や地域名などを数字のコードで入力するため，担当者は変換表を見なが

ら入力・参照を行わなければならない．

3) 関連情報を参照するのが面倒

現行システムには，システムが必要とする情報のみを保持しているため，たとえば顧客の住所から具体的な位置を調べるためには，地図を開いて調べなければならない．担当している検針業者の名前なども他の冊子を調べなければならず，より詳しい情報を調べるのが困難である．

などが挙げられる．このような問題点を認識し，

1) GUI化によりシステム操作の習得性を向上

システムをGUI化することで，コマンドによる操作を排除する．ボタン選択を操作の主体にすることで，操作性の向上と，その習得性を高める．

また習得性向上の一環として，GUIの操作内容をムービーに録画し，ヘルプシステムとすることで，直感的なマニュアルを提供する．

2) リスト選択方式の採用による入力の簡素化

項目入力をリストからの選択入力にすることで，変換表からのコード変換作業をなくし，入力ミスの低減と情報の可読性を高める．

3) 簡便な関連情報選択手順の提供

地図・検針員情報をシステムに取り込むことで，関連情報を調べるために他のメディアを参照する作業（たとえば，冊子を調べたりする）を省く．

を実現する．すなわち，現行システムが文字情報のみのシングルメディアに対して，本システムは動画・音声などの異なる情報を一度に扱う．これをマルチメディアと定義し，今回の開発に取り掛かることとした．

適用対象とした現行システムの画面は，以下のような処理を行うものに限定した．

契約情報……お客様の契約内容の検索に使用する（地図との連携）

料金情報……お客様の過去13か月の使用料金を表示する(使用料金のグラフ表示)

支払方法……使用料金の支払方法の入力に使用する（入力の簡素化）

3. マルチメディア指向営業窓口支援システム

3.1 システムに求められる要件

3.1.1 システムの短期開発

今回のシステム開発にあてられた期間は約3か月と短く，かつ開発担当者もWindowsやGUI構築は初体験であった．そのため，求められる要求には，できる限り容易に実現できる方法で対応する必要があった．

また，開発手法には，B社よりGUI構築や短期開発に適しているとして，設計者とプログラマが開発中の画面を見ながら，処理条件の確認や変更などを行うプロトタイピング手法を用いて進めていきたいとの要望があった．

3.1.2 GUI画面のレイアウト

現行システムはIBM社ホスト・(以下，ホスト）上のIMSと呼ばれるデータベースシステムの1アプリケーションとして稼働している（図3）．操作端末はDOSマシン上に3270端末エミュレータを実装して使用している．

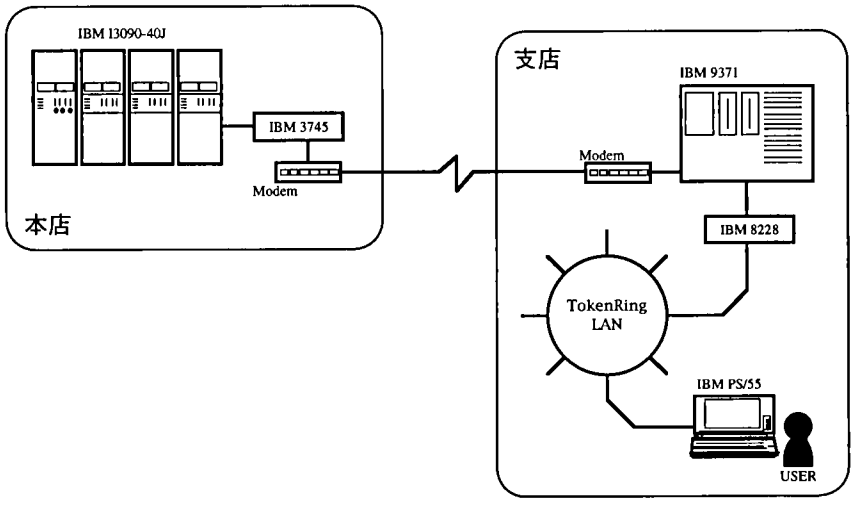


図 3 現行のシステム構成

本システムの開発は，現行システムの一部に対して行うため，現行システムとの混在が基本要求とされた．そのため GUI 画面（以下，GUI）の設計においても，現行システムのエミュレーション画面と比較して，できる限り違和感のない画面を利用者に提供して欲しいとの要求があった．

3.1.3 現行の営業窓口システムとの連携

本システムは，端末エミュレータで実行している現行システムと同等の処理を，GUI から実行できるようにしなければならない．そのため，GUI とホスト間でのデータ連携を行う必要があった．

3.1.4 静止画・動画・音声の取り扱い

営業窓口業務に必要な文字情報に加え，次に示す付加情報を提供または記録することが求められた．

静止画……地図・検針員の顔写真など
動　画……システムの操作マニュアル
音　声……電話の通話内容の記録

3.1.5 外部機器の制御

静止画は表示するだけではなく，保守のため取り込みも行えなければならない．また画面に表示される顧客の電話番号を元に電話を自動発信し，その通話内容を記録できる機能も必要とされた．そのため，イメージスキャナ，デジタルスチルカメラといった画像取り込み装置や電話機をシステムから制御する必要がある．

3.2 システムの構成

3.2.1 ネットワークの構成

本システムのネットワーク構成を図 4 に示す．本システムの開発をホスト側の構成に影響を与えずに実現するため，端末を DOS マシンから DOS/V マシンへ変更し，

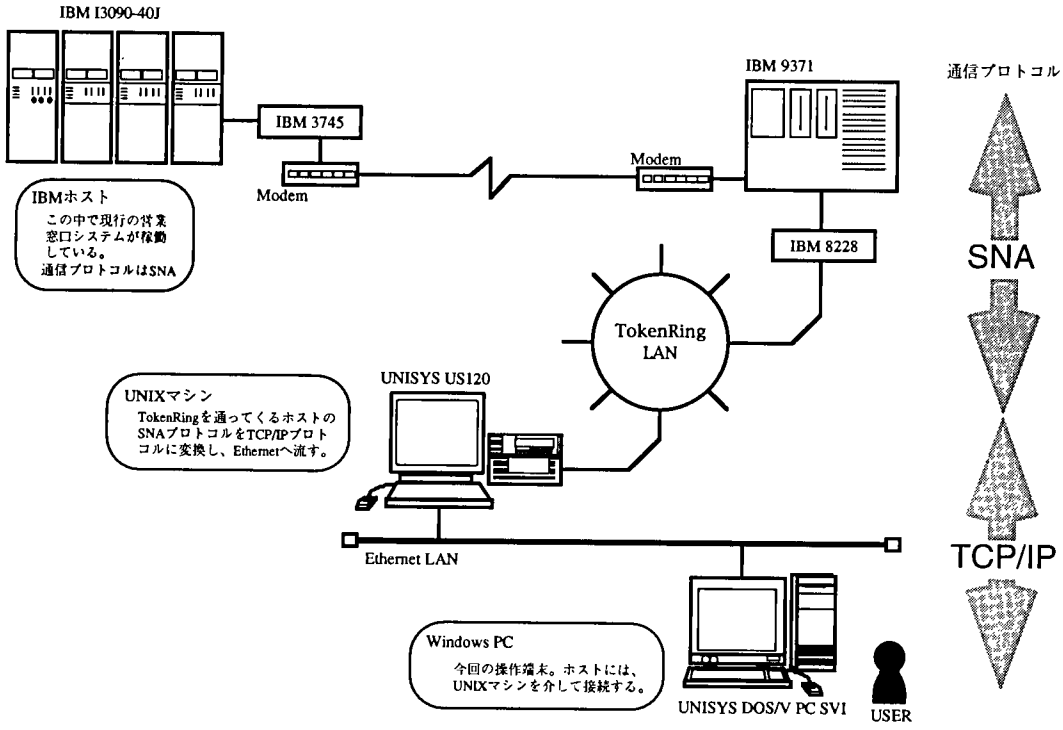


図 4　ネットワーク構成図

1)　端末エミュレータを用いてホスト上の現行システムを実行

2)　DDE (Dynamic Data Exchange) を利用して，端末エミュレータに表示されたデータを GUI へ受け渡す

ことで，ホストとのデータ連携を図ることにした．そのため端末エミュレータには，

- Windows 対応であること
- IBM 3270 系のエミュレーションが可能であること
- DDE に対応していること

の三つの条件に合致した川鉄情報システム(株)の OpenWay II を採用した．

さらに OpenWay II のホスト接続の方法が TN 方式* のため，通信プロトコルには TCP/IP を採用していた．それに対応するため，PC とホストの間に SNA・TCP/IP のプロトコル変換を行うゲートウェイを設置した．ゲートウェイには，UNIX マシン US 120 を設置し，大日電子(株)の OCS II を実装した (OCS II は UNIX 上で稼働する)．

3.2.2　ハードウェアの構成

ハードウェア構成は，PC の周辺に関してのみ示す(図 5)．PC はデスクトップ型で十分対応させる予定であったが，拡張スロットを多く使用するため，タワー型を採用した．

* TN 方式：TELNET 上で 6680 や 3270 のデータをやりとりするためのプロトコル．RFC 1041 で規格されている．

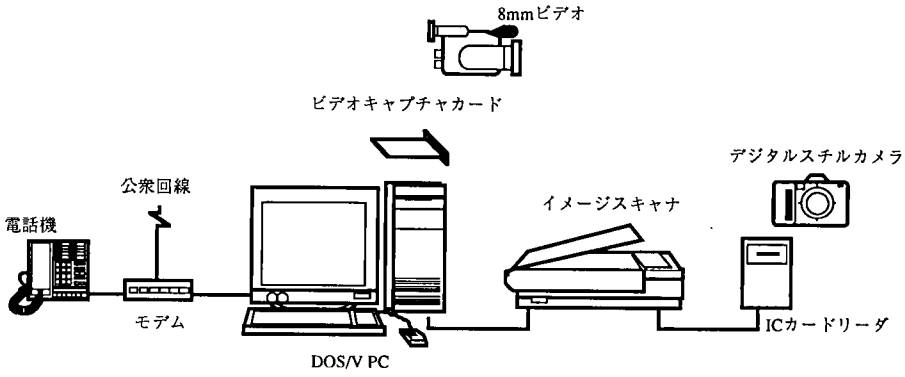


図 5　ハードウェア構成図

4. TIPPLER for Windows での機能実装

4.1 画 面 設 計

端末エミュレータに表示される現行システムの画面は 80 桁 24 行の文字画面になっている．これを GUI 化するにあたり，現行システムの画面レイアウトを尊重する方針をとったため，ほとんどはテキストフィールド* の集合で，各フィールドの項目名(たとえば，「丁目」「支払方法」) の一部をボタンにした．

TIPPLER で画面設計を行う場合，Uniscript の panel, layout 属性で大まかな指定を行い，あとの細かい配置は TIPPLER にまかせるのが普通である．しかし，今回は端末エミュレータに表示される現行システムに近似させなければならないため，各アイテムの配置は，xpositionchars, ypositionchars 属性を用いた座標指定で行った．座標をピクセル単位にしなかったのは，表示する文字のフォントサイズが変更** されたときに対応するためである．

4.2 現行システムとのデータ連携 (DDE)

4.2.1 現行システムとの関係

現行システムの操作方法について説明する．現行システムの画面の右上には，業務コードと呼ばれるユニークな番号が各画面に定義されている***．異なる画面に切り替える場合は，画面上の接続業務のフィールドに，切り替えたい画面の業務コードを入力し，「送信キー」を押す．この画面切り替え処理を GUI でも行わせるため，「送信キー」に相当する処理を GUI に持たせる必要があった．

今回は現行システムの数ある画面のうち 3 画面だけを GUI 化する．営業オンラインシステム全体から見ると，GUI 化された画面と現状通りのダム端末画面が混在するシステムとなる．そのため，GUI と端末エミュレータを必要に応じて切り替えて使う必要があり，図 6 に示すようなイメージで互いの切り替えを設計した．

---

* TIPPLER の Uniscript で記述するところの textfield と abbrevfield.

** TIPPLER for Windows では TIPPLER.INI に記述されている FONTALIAS セクションを変更することで，任意のフォントを表示フォントに指定できる．

*** 今回作成した画面の業務コードはそれぞれ，契約情報：FM 11，料金情報：FM 13，支払方法：FI 31 である．

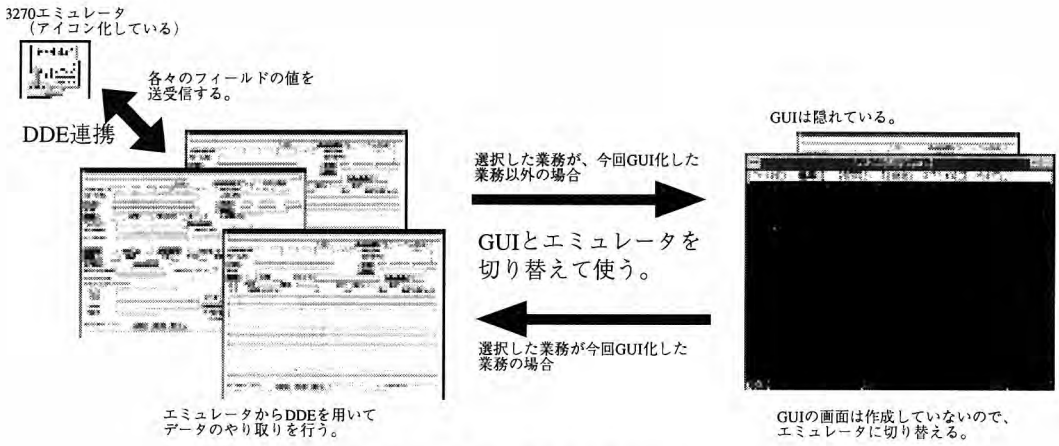


図 6　画面切り換えのイメージ

4.2.2　DDE 機能の実装（その 1）

現行システムとのデータ連係に利用する DDE 機能の実装は，2 段階に分けて行った．1 段階目は単純に端末エミュレータと GUI 間でデータの送受信を行うだけの機能を実装した．DDE 機能を用いたデータの送受信には Uniscript で用意されている DDEput()，DDEget()の関数を用い，端末エミュレータに表示されているフィールドごとにデータの送受信を行った．

送信/受信を行うトリガーには，それぞれ「送信」「受信」というボタンを GUI に用意することで実現した．「送信」では，GUI 画面に入力された各フィールドの値を端末エミュレータ上の対応するフィールドに送信し，最後に端末エミュレータの「送信キー」に相当する命令を DDE を用いて実行させる．この処理で，端末エミュレータを介して入力した値をホストに送信できた．「受信」は，端末エミュレータ上に表示されているフィールドの値をそのまま GUI 側に取り込むだけである．ただし，この方法は端末エミュレータの値が更新されたかどうかは利用者が目で確認しなければならないという問題があった．

端末エミュレータでは，「送信キー」を押すだけで，

1)　ホストへデータを送信

2)　結果をホストから受信し，画面を更新

の処理を行うが，1 段階目では，GUI は 1)と 2)のステップを一度に行うことができなかった．それは，GUI が受信を開始するタイミングをつかめなかったためである．端末エミュレータがホストの処理結果を受け，画面の更新が完了した時点で GUI は受信を開始すればよいのだが，このトリガーの監視方法がうまく考えつかなかった(図 7)．

4.2.3　DDE 機能の実装（その 2）

2 段階目の実装で注目したのが DDE の同期更新という機能である．この機能は，指定したサーバ* 上のアイテムの値が更新されると，その値をクライアントに通知する

* DDE では，データ送受信などのサービスを要求する側をクライアント，サービス要求に応答する側をサーバと定義している．

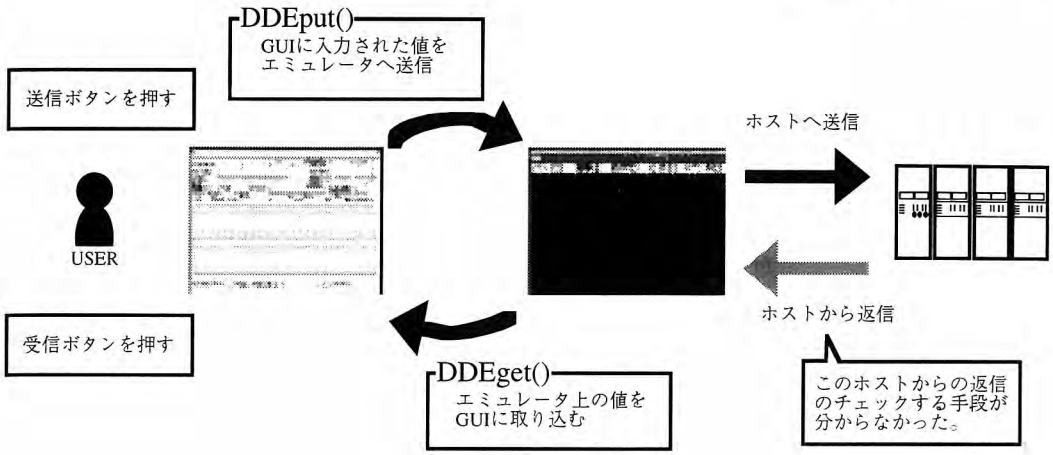


図 7　DDE 機能の実装（その 1）

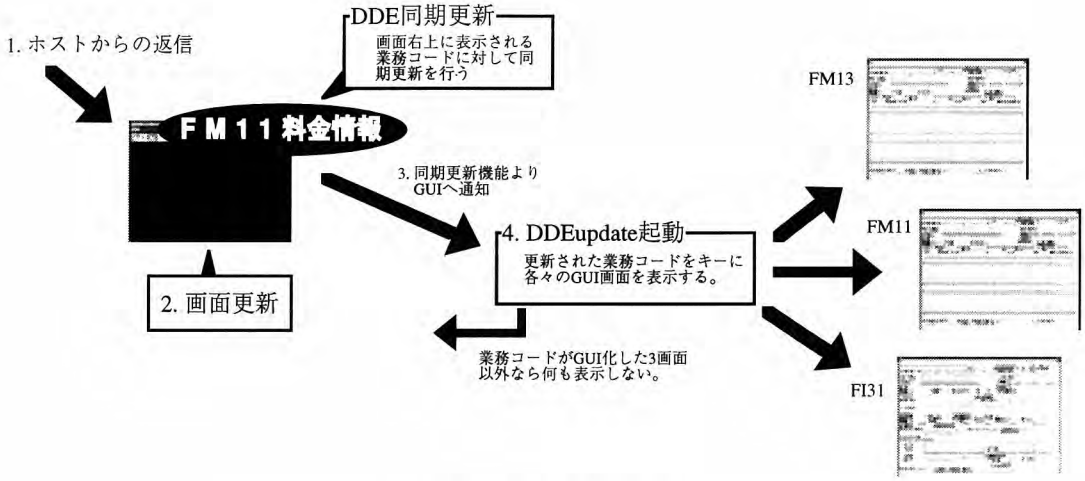


図 8　DDE 機能の実装（その 2）

もので，TIPPLER では DDEadviceStart 関数で実装できる．さらに，DDEupdate という名称のメソッドを定義することで同期更新が起きたとき，TIPPLER に任意の処理を行わせることができる．

この DDE の同期更新機能と，DDEupdate メソッドを利用し，図 8 に示す手順のデータ連携を実装した．2 段階目の実装で当初のイメージを実現できたのだが，時間の都合上デモンストレーションには間に合わなかった．そのため実装は実験レベルの確認となった．

4.3　静止画・動画・音声の張り付け（OLE）

文字情報以外の情報として，静止画・動画・音声を取り扱えるようにすることが検証の一つであった．静止画・動画・音声は前述したとおり，以下の内容を示すのに使用した．

静止画……地図・検針員の顔写真など
動　画……システムの操作マニュアル
音　声……電話の通話内容の記録

TIPPLER ではプレゼンテーションに bitmap 属性や picture 環境を持っており，BMP 形式の静止画の表示は可能であったが，動画や音声は扱える属性を持っていない．そのため動画と音声は OLE（Object Linking and Embedding）を用いて扱うこととした．また，静止画についても BMP 形式以外のものは OLE を用いて扱うこととした．

4.3.1 静止画の表示

静止画の表示には，主にプレゼンテーションの bitmap 属性を使い，BMP 形式の画像ファイルを表示した．BMP 形式以外の画像として，実験的に cmp 形式* のファイルを OLE を用いて表示した．結果，bitmap 属性で表示するのと変わりなく表示できた．bitmap 属性で表示した画像をスクロールするより，OLE で表示した画像をスクロールする方が，より滑らかに行えることがわかった．

地図を表示する際に，画像の拡大・縮小を要求されていたが，今回使用した TIP-

図 9　静止画を張り付けた例

* IC カードリーダ付属ユーティリティのオリジナル形式．ユーティリティが OLE 対応であったため，OLE で張り付けることができた．

PLER では，まだ bitmap 属性の拡大・縮小機能は提供されていなかった．そのため，あらかじめ拡大・縮小した画像を切り換えて表示させることで対処した（図 9）．

4.3.2 動画の表示

動画のファイル形式として取り扱ったのは，Video for Windows* と ScreenCam** の 2 種類である．前者は VTR 録画した動画を，後者は Windows 上のオペレーションを録画した動画を表示することに利用した．これらはすべて OLE を用いて表示している．

4.3.3 音声の取り込み・再生

音声も動画と同じく OLE を利用した．音声は，静止画や動画と異なり表示（再生）だけではなく，取り込み（録音）の必要もあった．取り込みの方法は，可能な限り短期間で実現できる方法ということで，音源ボードに付属してきた録音ユーティリティを利用することにした．その手順は，

1) TIPPLER から，録音ユーティリティを起動
2) ユーティリティで音声を録音
3) 録音した内容をクリップボードに保存させ，ユーティリティを終了
4) TIPPLER は，クリップボードの内容を自身に張り付けファイルに保存

である．多少面倒ではあるが，この方法により音声の取り込みが容易に実現できた（図 10）．

4.4 外部機器の制御

今回使用した外部機器は次のものが挙げられる．

イメージスキャナ……地図，写真などの取り込みに使用
IC カードリーダ ……スチルカメラで撮影した映像の取り込みに使用
ビデオキャプチャカード……VTR から動画の取り込みに使用

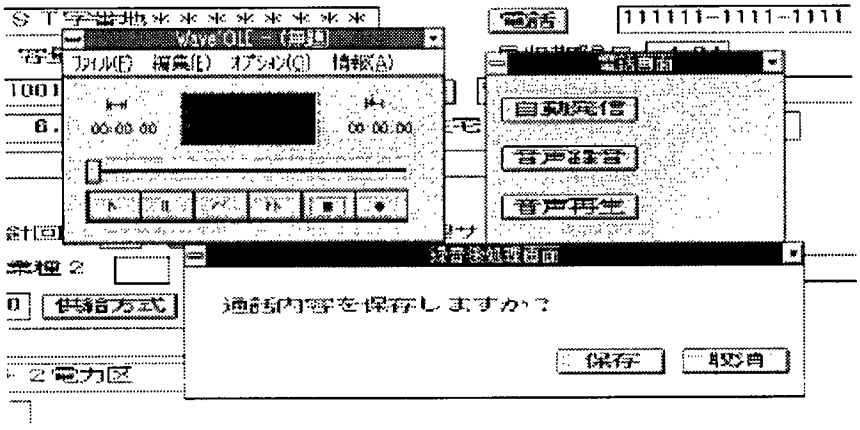


図 10 通話内容の録音画面

* Video for Windows は米国 MicroSoft 社の登録商標である．
** Windows の操作内容をムービとして録画するソフトウェア．Lotus 社の登録商標である．

モデム……電話の自動発信に使用

ビデオキャプチャカード以外の機器に関しては，GUI から制御できるよう設計したが，時間的な理由からイメージスキャナと IC カードリーダは，付属してきたユーティリティを利用する方法で対応した．イメージスキャナは，その API が公開されていないため，GUI で制御を行うのは難しいが，IC カードリーダに関しては，TIPPLER 用にクラスライブラリ* が用意されているため，GUI からの制御は可能である．

モデムの制御は，AT コマンドを記述したテキストファイルを，モデムが接続されているポート (COM 1 とか COM 2) に対してコピーすることで実現した．ダイヤル発信を行う AT コマンドのバッチファイルを GUI から起動するのだが，Uniscript の system 文で起動すると DOS 画面に切り替わるため，見た目がよくなかった．そのため system 文の替わりに WindowsAPI の WinExec 関数を用い，バッチファイルをアイコンの状態で実行するようにした．

各々の装置に対して API を利用し，システムとして一体化するのがスマートであったが，期間的な制約もあり，今回は各々の装置付属のユーティリティを GUI から起動することで対応した．

5. 評価と課題

本システムについて，開発要求者の A 社および開発主体の B 社より次の観点から評価を受けた．

1) 開発生産性
2) マルチメディアの業務適用の可能性
3) ホストプログラムを継承したシステム構築

以下に A 社および B 社の評価を記述し，その評価に対する筆者の意見を述べる．

1) 開発生産性

開発生産性については，TIPPLER の生産性とプロトタイピング手法の生産性について評価している．

TIPPLER の生産性については，Uniscript 言語の習得性と，Uniscript とプログラミング生産性に分けて評価している．

Uniscript（TIPPLER）の習得性についての評価は以下のとおりである．

- 5 日間の教育コース受講で，初心者でも Uniscript 言語の基礎を習得できる
- TIPPLER の基本概念であるオブジェクト指向を理解するのは短期間では難しい

また，Uniscript（TIPPLER）のプログラミング生産性についての評価は以下のとおりである．

- C の 6.6 倍の生産性がある（Uniscript のソースライン数と C に変換後のソースライン数の比較による）

プロトタイピング手法については，次のような評価であった．

---

* クラスライブラリ：「画像処理 for TIPPLER」(株)ジャパンテクニカルソフトウェアより発売．

・設計者とプログラマの意思の統一がはかれるため，手直しを減少できる
・口頭指示が多いため，議事録などの管理を行わないと長期開発には向かない

2)　マルチメディアの業務適用の可能性

静止画，音声，動画などのマルチメディアデータを扱った評価も，その作成過程と保守・運用面の2通りで評価を受けた．

作成過程について，

・専用機器を用いることで，容易にデータを作成することができる
・データのアレンジには，多少の専門知識を必要とするため容易ではない
・多くの機器を導入すると，互いに干渉し合ってシステムが不安定になりやすい

との評価を受けた．

保守・運用面では，

・データの圧縮を行えなかったため，データ量が膨大になった
・今回，データは全てPC上に置いたが，実業務では現実的ではない
・表示，再生を始めるまでのレスポンスが多少遅い

と少々辛口の評価となった．

3)　ホストプログラムを継承したシステム構築

ホストプログラムの継承方法として，「端末エミュレータ＋DDE連携」の手順を提示し，弊社（日本ユニシス）側が全面的に支援する形で構築していった．その評価を以下に示す．

・ホストとのデータ連携は，大きな問題もなく正常に処理できた
・DDEに対する造詣がないとプログラミングは非常に困難である
・データの送受信について，まだまだ検証の余地がある

以上の評価を受けて，筆者の意見を述べる．

1)について

TIPPLERの習得性の高さについては以前から定評がある．今回，プログラミング支援を行って気付いたことだが，変数，メソッド名などを日本語で表記できるという事が，その習得性に一役買っているようだった．

プロトタイピング手法については，TIPPLERとは切り離されて評価されたように受け取れる．しかし，プロトタイピングによって指摘された変更や修正をすぐさまプログラムに反映できたのは，TIPPLERの保守性・改修性の高さによるところが大きい．TIPPLER自体がこのプロトタイピング手法に適した言語系なのだろう．ただし，実業務を考えた場合には，上流工程でのシステム設計は必須となる．

2)について

マルチメディアの業務適用を考えた場合，「データ生成手段は問題ないが，運用・保守にはまだまだ考慮の余地がある」という評価となろう．ただし，データ生成が可能な環境を用意するには，PCについてかなり高度な知識を必要とする．今回，さまざまな機器を導入することになったが，機器の組み合わせによっては，

正常動作しなかったり，ハングアップしたりとトラブルが絶えなかった．この件については，Windows 95 と PCI バスの「プラグアンドプレイ」技術に注目したい．

保守・運用面にて，データの圧縮が行えなかったとあるが，これは本システムの開発を容易にするため TIPPLER の picture 環境で BMP 形式の静止画を表示していたことを指す．BMP 形式は無圧縮の画像フォーマットのため，そのデータ量も相当な大きさになる．JPEG, TIFF などの圧縮フォーマットを利用するには，OLE を利用して TIPPLER に張り付けることで対処ができる．

今回は，画像データを無圧縮で保存していたため，ディスクからの読み込みに要する時間が大きくなり，レスポンスの低下につながった．実運用においては，以下のような方法が一般的であろう．

- 静止画や動画は圧縮フォーマットにてサーバに格納する
- クライアントとして，本システムのような操作画面を提供する
- クライアントからの要求により，サーバから画像データを転送する
- クライアント側で展開して GUI に表示する

その際，問題となるのは画像データの転送速度とクライアントにおける展開時間である．転送時間は，データの圧縮率を上げれば小さくすることが可能であるが，その半面クライアントにおける展開に時間がかかっては，実用に耐え得ない．動画などでは，データの転送を受けつつ，展開するなどの技術が求められる．すでに圧縮データを展開する CPU などが販売され始めており，今後の技術革新に注目していきたい．

3) について

DDE を利用したことにより，ホスト側にはなんら手を加えることなく，データ連携を行うことができた．これは，かなり画期的なことだと思う．ただ，細かい点については，まだまだ検証の余地は残っている．今回の場合でも，ブランクを送信できなかったり，半角文字が全角文字として受信される現象が確認された．この件については，時間的理由から未解決となっている．

DDE の同期更新を利用したデータ連携は，デモに間に合わせることはできなかったが，当初のイメージ通りのデータ連携を実現することができた．この同期更新，とくに TIPPLER の DDEupdate メソッドは，利用のしかたによって色々な可能性が見えてきそうである．

このように，マルチメディアデータの利用や，ホストとのデータ連携が短期間で実現できたのは，TIPPLER が OLE, DDE といった Windows アーキテクチャを積極的に取り入れてきたためだといえる．もしこれらが支援されていなければ，今回のシステム開発は，これほど容易には実現できなかっただろう．

TIPPLER は UNIX から Windows へ移植されたばかりにもかかわらず OLE, DDE といった Windows の文化に馴染んでいこうとする姿勢は素晴らしいが，プレゼンテーション部分には不満が残る部分がある．

TIPPLER は，中規模～大規模なプログラム開発も視野に入れているため，画面上の項目は相対的な位置関係を指定すれば，実際の配置は TIPPLER に任せることができ

るようになっている．これは新規開発のシステムにおいては有効であるが，本システムのように利用者に違和感を与えないように現行システムと近似させたGUIを作成するのは難しかった．とくにTIPPLERでは，画面設計をソースのコーディングで行うため，実際の画面レイアウトを確認するには，プログラムを実行してみなければならない．画面設計/変更をより容易にするために，画面レイアウトの作成をグラフィカルに行える支援機能が必須と思われる．

またWindowsアプリケーションは，ツールバーやボタンバーといった目新しいメタファー，カラフルなカラーリングなどを利用して，使いやすさをアピールしている．これらをTIPPLERで実現するのは困難であった．

今後のTIPPLERのリリースにおいては，UNIXからWindowsへの移植に際してもOLE, DDEといったWindows文化に対応したように，プレゼンテーションについてもWindows文化に沿った設計思想を取り入れていくことを期待したい*．

6. おわりに

本システム検証では，動画や音声を扱う事をマルチメディアと定義して，その実用性について検証してきた．筆者が思うに，このマルチメディアを本システムで一番効果的に利用していたのは，システムの操作説明部分であろう．現在，マルチメディアと呼ばれ市場に出まわっているCD-ROMタイトルもエンターテイメントを除くと，教育ものが大半を占める．このことから，マルチメディアはCAIなどの教育分野，もしくは今回のようなヘルプシステムの強化に用いる事が，用途として適しているのではないだろうか．

マルチメディアという言葉の響きに，様々な夢を抱く方も多いと思う．今回の開発を通して，その陰の部分も少なからず垣間見る事ができた．マルチメディアということで，動画や，音声といった今までは専門家が扱っていたデータも，一般に扱えるようになった．それ自体は，むしろ喜ばしい事であるが，そのデータを扱う(作成する)側としては，多少なりともその分野の専門知識を必要とする．静止画をスキャナから，動画をVTRから，音声をマイクから取り込んだとして，その質や精度を上げるための2次加工が必要になる．たとえば，

- ノイズのない音声データ作成するには，どのようなフィルタリングを施せばよいのか
- 静止画の色彩を統一するには，どのパラメータを調整すればよいのか
- 複数の動画を合成する際に，どの部分にマスキングを設定すればよいのか

などだ．これらの加工方法を素人が発想するのはなかなか難く，ある種のセンスを要求される．道具があっても使いこなせなければ意味がないのだ．

マルチメディアとは，利用者には操作説明などに利用する事で，必要以上の知識を補うための手助けとなる．しかし，製作者には，業務知識，プログラム知識以外に，動画や音声などのデータを扱うための知識までを要求する．マルチメディアという言葉は，その万能ぶりばかり目立つが，あまりその語感に浮き足立たず，足元をきちん

* TIPPLER/Vでは，ツールバー，ボタンバー，画面レイアウト支援機能がサポートされている．

と見据えておくのは，マルチメディア実用を真に考える上で必要な事だと思う．

最後に，この場を借りて本システムの開発ならびに本稿の執筆にあたりご指導，ご協力いただいた方々に感謝の意を表する．

執筆者紹介 佐々木 勝 信 (Masanobu Sasaki)

1972 年生．1993 年国立一関工業高等専門学校化学工学科卒業．1993 年日本ユニシス(株)入社．社会公共システム本部社会公共システム部エネルギーシステム 1 課に所属．客先サービス担当．

TIPPLERの製造業における スケジューリングシステムへの適用事例

An Application of TIPPLER to a Scheduling System for the Manufacturing Industry

田頭　浩

要　約　製造業において，生産現場の作業計画を立案するスケジューリングシステムは，今，最も各企業が重視しているものの一つであろう．しかしながら，このようなスケジューリングシステムは，構築に多くの時間と費用がかかる割には，あまりうまくいっていないのも実状であろう．今回は，このようなスケジューリングシステムを業務モデルサブシステム（テンプレート）として，TIPPLERを利用して開発した．

開発に当たっては，業務モデルサブシステムへの要件を整理し，スケジューリング業務手順のモデル化，部品化構造，GUIの駆使などで対応した．

スケジューリング業務手順のモデル化では，スケジューリング業務の流れをモデル化し，システム機能のレイヤー化を図った．部品化構造では，ユーザ要件により変化する部分と比較的安定している部分に分離し，さらに階層化の概念を盛り込み，各層別にそれぞれの単位での部品化を図り，改良・保守の容易性を狙った．GUIの駆使では，TIPPLERのガントチャート機能を利用し，スケジューリング結果を直接，マウス操作で調整し，その結果を即座に確認できる機能を開発した．本稿では，それらの対応について，TIPPLERでどのように実現していったかを開発の事例として報告する．

Abstract　It can be said that all businesses in the manufacturing industry attach the greatest importance to the scheduling system which helps formulate work plans on the shop floor as one of their pursuits. The fact, however, would be that such a system does not function well enough to deserve a lot of time and cost required for its creation. The author's team has just completed the building of a scheduling system as an application subsystem model (template) with the use of TIPPLER.

For the development, the requirements of the subsystem model were first specified, and the next efforts included the modeling of a scheduling flow, the designing of a system diagram based on parts structure, and the adoption of a GUI for the user interface.

The modeling of the scheduling flow involved the layering of the system functions. The parts structure urged the separation of segments which vary depending on user requirements from comparatively stable ones and then the implementation of the hierarchical structure approach for the segmentation of each structural layer as individual unit parts to ensure the ease of modification and maintenance. For the GUI, for which TIPPLER's Gantt chart feature was used, the new development was the function that enables users to directly adjust the results of scheduling by executing mouse operations, thus making it possible to immediately have access to the adjusted schedules.

As a development example, this paper is intended to report how TIPPLER has served to make these

本稿に記載の会社名，商品名は，一般に各社の商標または登録商標である．

functions available.

1. はじめに

製造業において，生産現場の日々の活動を円滑に効率良く行うことは，最も重要なことである．そのために，生産現場の作業計画を立てるためのスケジューリングシステムは，従来より多く構築されている．しかしながら，これまでのシステムはハードウェア性能や構築ツール（または，開発言語）などの制約から，処理に時間がかかる，使い勝手が悪い，システムの構築に時間と費用が多くかかる割に拡張性がない，など多くの問題があった．

近年，オブジェクト指向技術の実用化のきざし，ミドルソフトウェアの充実，開発環境のオープン化，低価格で高性能なワークステーションやパソコンの利用の拡大など，徐々に，情報システム技術環境は整備されてきている．その中でも，ミドルソフトウェアとしてのTIPPLERは，オブジェクト指向プログラミングの実現，GUI（グラフィカル・ユーザ・インタフェース）の実現，ハイブリッド・ストラクチャの実現など，これまでの問題を解決するための多くの機能を備えている．本稿では，TIPPLERの持つこれらの機能を利用して，新しいスケジューリングシステムを，いかに開発したかの事例を報告する．

2. 開発の背景

製造業においてシステムの出発は計画にある．計画をきちんと立てることで

① 全社全工場で計画（情報）を共有化する

② 情報に基づいて各部門が一致して行動する（自律経営）ことにより経営効果を高める

③ 工場/職場をその時点において最適状況にもっていく

などを確立すべきである．

そのためには，戦略的なスケジューリングシステムを構築することが重要となる．

スケジューリングシステムは人中心の業務であるため，現状では，限られた専門家による試行錯誤やこれまでの経験を頼りに判断する部分などが多くある．また，製造する対象物や生産設備なども影響し，各企業でルールや制約条件などのスケジュール要件に格差があり，なかなか汎用化が難しい状況にある．

そのために，汎用のスケジューリング・パッケージでは，ユーザの要求に対応するためのカスタマイズが必須であったが，カスタマイズを前提にしていない堅い構造のパッケージが多く，なかなか十分には，ユーザの要求に対応しきれないことが多いようである．

また，個別に開発する場合，何もない状態からスケジューリングのルールや制約条件などを専門家の知識から引き出すには多大な時間が必要であるが，引き出しに成功した場合も，現在の環境条件下でのやり方をシステム化するに止まり，市場動向の変化などによるスケジューリング方針の変更などが発生した場合は，システムを再構築する必要が発生する．したがって，限定された比較的小さいシステムであるが，システム化しにくい分野であり，これまで，システム化があまりうまくいっていないのが

実状である．そこで，今最も各企業が重視しているスケジューリングシステムの業務モデルサブシステム（テンプレート）を開発することとなった．

3. スケジューリング・テンプレートへの要件

テンプレート（業務モデルサブシステム）への要件として，以下のようなものがある．

① 業種/業態別の業務単位程度の大きさのサブシステムで，部品の組立によって作られ，そのままでも動かすことができる

② 見本/原型（モデル）としての役割を担い，ユーザのシステム化業務領域に対して，

・トップや利用部門へのデモ/刺激

・あるべき姿の抽出剤

・システム開発の参考

・モデフィケーションの原型など

として利用できる

このような要件を受けて，スケジューリングの業務モデルサブシステム（以下，スケジューリング・テンプレートと呼ぶ）への要件を整理し，開発目標とした．

また，スケジューリング・テンプレートは，スケジューリングのタイプに合わせて以下の2種類を開発した．

タイプ1は，単一職場/ライン/機械/工程への割付型で，ライン/機械/設備に対するオーダの納期・ルール・制約条件などに沿って優先順割付を行うもの

タイプ2は，複数工程間連動型で，シリアル，ネットワーク，オーバラップという異なる種類の工程を連動させ，能力有限の割付を行うもの

本稿では，スケジューリング・テンプレートのタイプ1（以下，スケジューリング・テンプレートIと呼ぶ）について記述する．

スケジューリング・テンプレートIの要件を整理した開発目標は以下の通りである．

① 同様な形態（機械加工/組立業）の企業/工場への適用度が高い
→オブジェクト指向での設計/開発

② ユーザの運用時の多様な要求に柔軟に対応できる
→システムの機能とソフトウェア構造のレイヤ化

③ ユーザニーズに合わせて，仕組みの変更が容易にできる
→データベース使用の有無，エキスパート型エンジン使用の有無などのオプション化

④ 実戦的な機能を標準として装備している
→混流生産，治工具条件，ライン特性，オーダ確定，既計画への上乗せ再計画，ロットまとめ/分割，ビジブル評価，調整などの多くの機能の標準装備

このような開発目標を実現するための開発支援ツールとしてオブジェクト指向プログラミングの実現，ハイブリッド・ストラクチャの実現，GUIの実現，などの機能をバランス良く持っているTIPPLERを採用した．

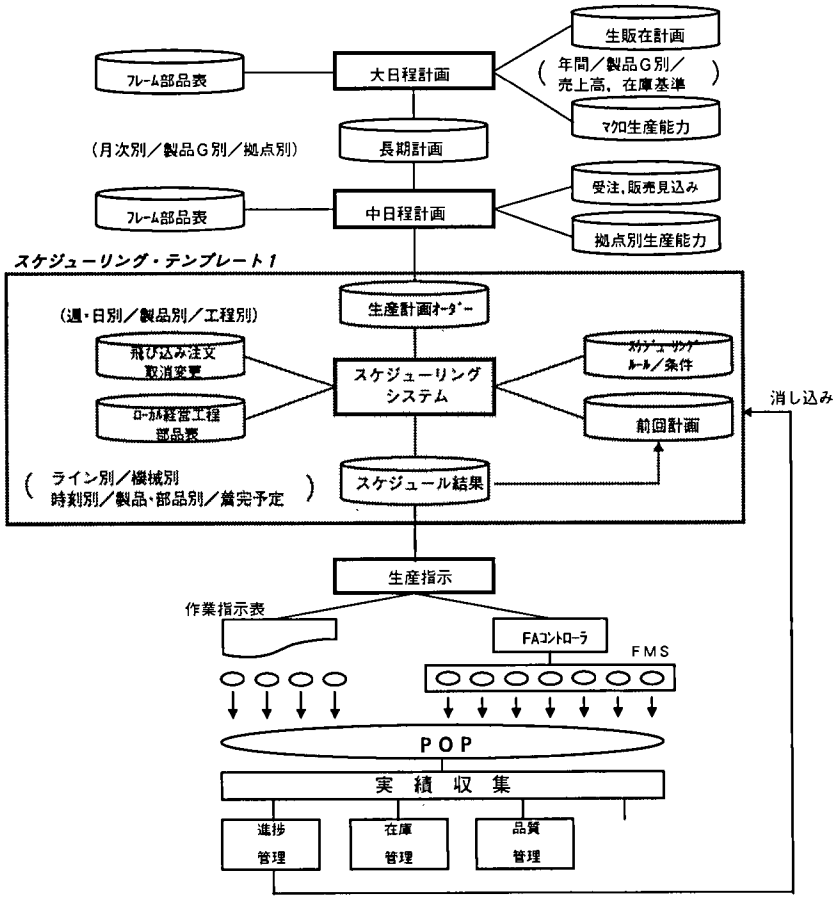


図 1　スケジューリング・テンプレート I の位置付け

4. スケジューリング・テンプレート I の概要

スケジューリング・テンプレート I の概要を紹介する．

まず，業務サブシステムとしての位置付けは，他のサブシステムとデータを共有し，連動して動くことを想定しており，生産現場の割付と作業計画を立案する部分である(図 1)．

次に，スケジューリング・テンプレートの基本構造は，システムの機能とソフトウェアの構造をレイヤ化したものになっている．ソフトウェア構造としては，インタフェース部分を本体部分と切り放してレイヤ化してある．このことにより，たとえば，使用するデータベースソフトウェアを変更したりすることが容易にできるようになっている．また，システム機能もレイヤ化してある．これは，後述のスケジューリング業務手順のモデル化と部品化構造で実現できたものであるが，レイヤ化することにより，ユーザの運用時の違いなどの変化に強い構造になっている (図 2)．

また，主な標準装備機能は，システムの機能レイヤごとにあり，図 3 に示すような機能が準備されている．

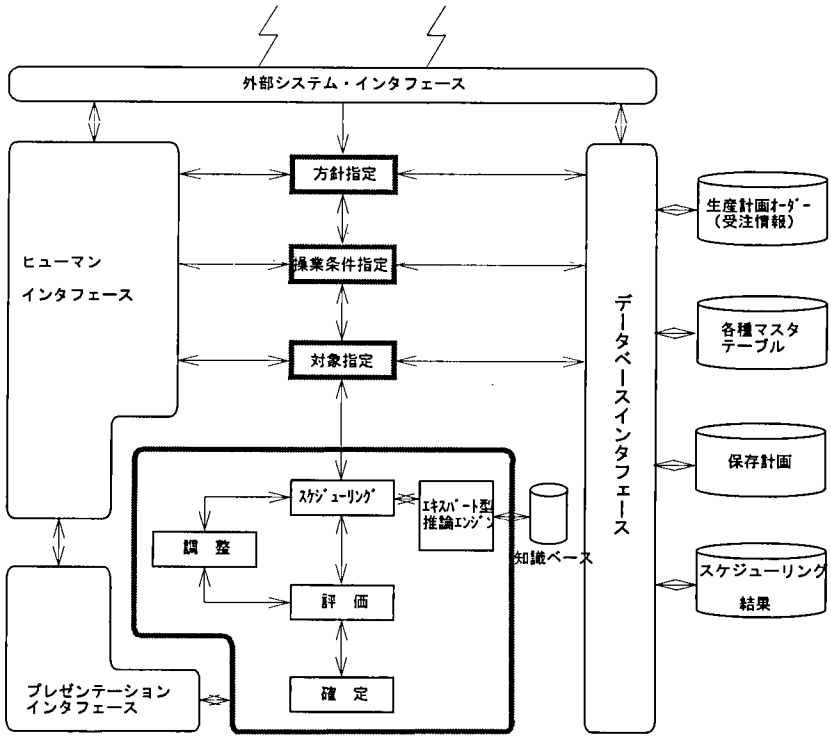


図 2　スケジューリング・テンプレートⅠの基本構造

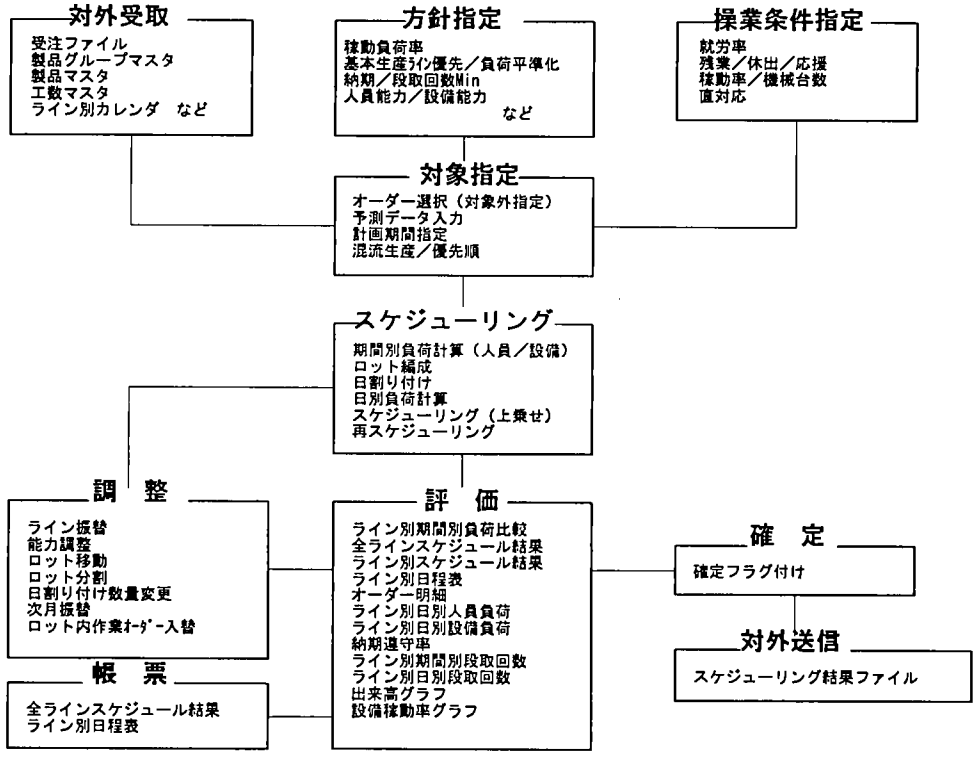


図 3　スケジューリング・テンプレートⅠの主な標準装備機能

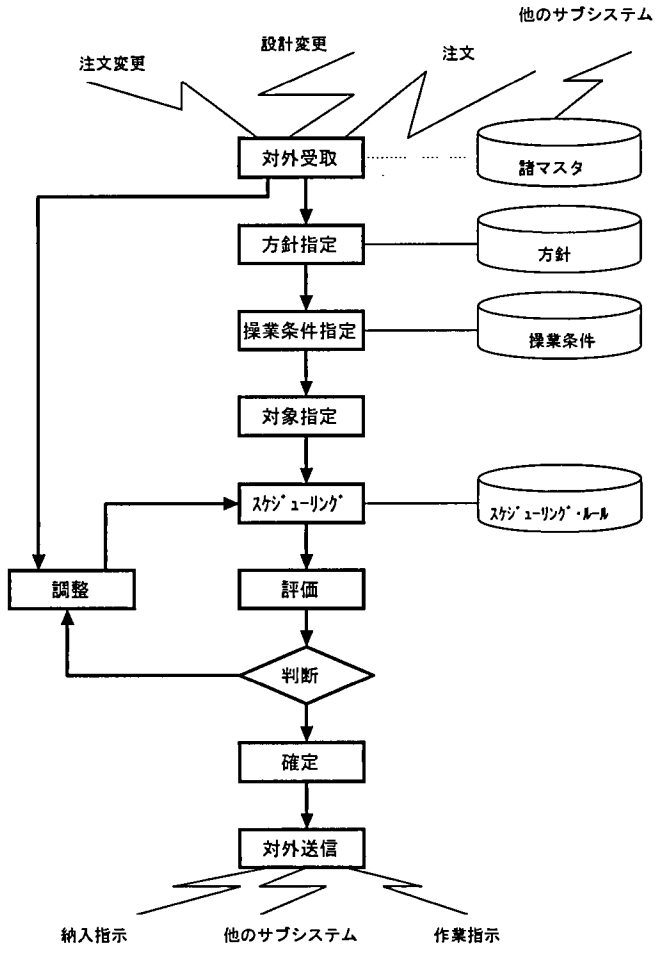


図 4　スケジューリング業務手順モデル

5. TIPPLER によるスケジューリング・テンプレート I の開発

5.1　スケジューリング業務手順のモデル化

スケジューリング・テンプレート I を開発するにあたり，スケジューリング業務の流れをモデル化した（図 4）.

ここでは，計画担当者が，実際にスケジューリングする場面を想定し，どのような手順でスケジューリングしていくかを洗い出し，業務に必要な機能を層別すると共に，それらの機能の流れを整理した．また，機能を層別する場合，パラメータ化して，運用時に変更できるようにしたいものは，さらにそれらの機能を層別した.

5.1.1　スケジューリング・テンプレート I のシステム機能

スケジューリング業務手順のモデル化により洗い出したスケジューリング・テンプレート I の各々の機能について概説する.

1)　対外受取

スケジューリングに必要な情報を他のサブシステムから受け取り，スケジューリング・テンプレートの入力情報として扱えるようにする機能を位置付けた．こ

れにより，他のサブシステムとの接続・同期化が容易にできる．

2)　方針指定

スケジューリングする場合に，生産効率を優先して生産の順序を決めていくか，または，納期を優先して生産の順序を決めていくか，など市場動向の変化などにより変わる可能性のあるものを方針という形でスケジューリングの実行時に設定できる機能を位置付けた．この機能により，市場動向の変化などによりスケジューリング方針が変わってもシステムを変更・修正することなく，運用できることになる．

3)　操業条件指定

現場の操業状態を的確にシステムに与えられるように，直体制，休日出勤，応援，設備の稼働率など多くの要素を実行時に現場から直接指示できる機能を位置付けた．これにより，スケジューリングを実行する場合に，現在の職場の状況を常に正確に反映することができることになる．

4)　対 象 指 定

計画を立てる期間やこれから計画するオーダに対する変更・取り消しなどの情報を指定する機能を位置付けた．このことにより，たとえば，これまで月単位でのスケジューリングであったものが，管理精度の向上などにより，半月単位とか週単位に変更されるような場合でも，システムを変更・修正することなく，運用を変更することができることになる．また，突然の注文変更や飛び込み注文などへの対応もここの機能とした．

5)　スケジューリング

ルールや制約条件に沿って計画を立案する，いわゆるスケジューリング・エンジンの機能を位置付けた．機能としては，全オーダ，全機械/全設備を対象に，全体を一括で計算するものと，変更の発生した部分についてのみ再計算するものとを用意した．

6)　評　　価

スケジューリングした結果を様々な尺度で総合的に評価できるように，計画全体のガントチャート表示，ライン別の負荷グラフ表示，全ライン全期間での段取り回数一覧表表示，出来高グラフ表示など様々な観点からの結果の表示機能を位置付けた．結果を評価する場合，はじめに結果全体をマクロ的に捉え，問題がある部分については，ドリルダウンして詳細に結果を評価することができるような機能にした．また，評価を定量的にできるように数値データの表示機能も合わせて用意した．この機能により，結果の善し悪しの判断基準を公開できることになり，将来的には，専門家のみが可能であったスケジューリング業務を一般事務職に移管することも夢ではなくなると思われる．

7)　調　　整

スケジューリングの結果を判断した後，計画として問題があるものに対して，生産能力の変更や，オーダの分割，納期の変更，といった結果に対する調整を行う機能を位置付けた．最終的な計画には，人の判断による調整が加味できることになる．また，調整の方法は，できるだけ操作を容易にし，また，調整した結果

を即座に確認できることも重要と考えた．

8) 確　　定

スケジューリングの結果を判断し，良しとしたものを計画の確定とすることができる機能を位置付けた．

9) 対 外 送 信

スケジューリングの結果の情報を他のサブシステムへ引き渡す機能を位置付けた．「対外受取」の部分を含めて，外部のサブシステムとのインタフェースをとる機能である．

5.1.2 スケジューリング・テンプレートⅠでの業務の流れ

次に，スケジューリング・テンプレートⅠを使用する場合の業務処理の流れについて概説する．

計画担当者がスケジューリング業務を行う場合，まず，今回のスケジューリングの対象データを他のサブシステムなどから受け取ることになる．次に，スケジューリングの方針や，操業条件，対象といったものを指定し，現場の状況の変化などを今回のスケジューリングに反映できるようにする．その後で，画面上よりの指示でスケジューリングする．スケジューリングが一通り終了すると，スケジューリングの結果が自動的に画面上に表示されるので，その結果を評価し，調整の必要性を判断し，必要があれば，スケジューリングの結果を調整する．能力情報などを変更することで調整する場合などは，能力情報を変更したものに対して，再度，スケジューリングをやり直し，また，評価・判断を行うことになる．評価，判断，調整，再スケジューリングのサイクルを繰り返し，スケジューリングの結果が最適になったところで，スケジューリングの結果データを確定し，結果データなどを他のサブシステムへ引き渡すことになる．この業務の流れで解るように，スケジューリング・テンプレートⅠでは，結果を人が判断し，最適な計画を作成することを想定している．

5.2 部品化構造

5.2.1 階　層　化

部品化を検討する上で，階層化の概念を導入した．全体構造を1)表示層，2)論理表現層，3)物理表現層の3階層で定義した（図5）．

1) 表　示　層

表示層は，計算機内の論理的な情報を，人の目で見える形式で表現するための

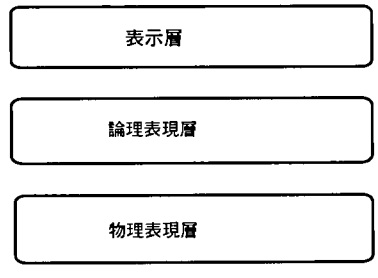


図 5　階層化概念図

階層であり，計算機内と外部とのインタフェースを図ることを目的とする．画面や帳票などは，この階層に含める．

表示層においては，表示情報は論理表現層で表現された情報から導出し，また，論理表現情報の更新は，論理表現層に依頼することにより実現する．

2)　論理表現層

論理表現層は，スケジューリング・テンプレートIの基本となる情報を表現する階層であり，画面表示の都合や物理的な表現形式などにとらわれない本来の情報を表現する．

他の階層は，論理表現層で表現された情報を基に（継承，参照などで）構成される．

3)　物理表現層

物理表現層は，論理表現された情報と情報に永続性を持たせるための物理的な表現とを対応付けるための階層である．

物理表現には，UNIX* ファイル，リレーショナルデータベース，などがあり，何れを選択するかにより容易に可換な構造とする．

階層化の概念を基に，TIPPLERのオブジェクト指向プログラミングを利用し，各階層ごとに，クラス構造を設計した．

5.2.2 部　品　化

階層化の定義に基づき，各階層ごとに部品化の基本方針を検討した（図6）．

1)　表示層には，画面と帳票を位置付け，それぞれの部品化の基本方針を以下のようにした．

①　画面表示部品

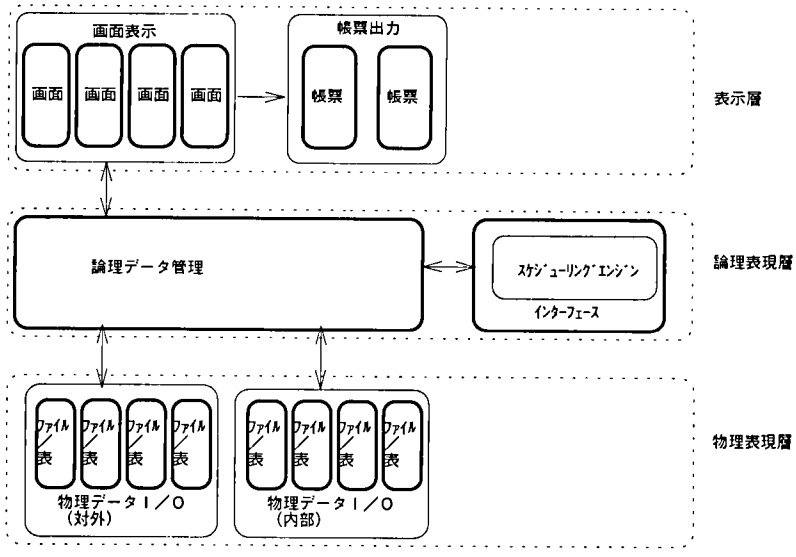


図 6　部品化概念図

* UNIXはX/Openカンパニーリミテッドがライセンスしている米国ならびに他の国における登録商標である．

・原則として，1機能を表現する画面群を一つの部品とし，この単位にTIPPLERでの実装上のクラス構造を検討する．ここでの1機能を表現する画面群とは，関連性の深い画面の集合を意味する．

・汎用性の高い画面（たとえば，テンキー画面など）の場合は，それ自体を単位部品とする．

・画面表示での編集メソッドは部品に内装するが，処理メソッドは本来のメソッドの呼び出しを内装する．

② 帳票出力部品

・1帳票を一つの部品し，この単位に，クラス構造を検討する．
実装では，TIPPLER-PUI（Printing User Interface）を使用する．

2) 論理表現層には，論理情報（オブジェクト）を管理する部分とスケジューリングを行うスケジューリング・エンジンを位置付け，それぞれの部品化の基本方針を以下のようにした．

① 論理データ管理部品

・スケジューリング・テンプレートIの基本的な論理情報の全てのオブジェクトを保持するものであり，全てのオブジェクトが関連性を持っていることから，この単位に，TIPPLERでのクラス構造を検討する．

② スケジューリング・エンジン部品

・本来は，「スケジューリングする」というのは，振る舞いであり，論理データ管理部品内にメソッドとして定義すべきものであるが，スケジューリング機能を可換な形式で実装する必要があるので，論理データ管理部品とは別に，スケジューリング・エンジンのクラス構造を持つものとする．

・エキスパート型のエンジン部品と人判断型のエンジン部品とを実装する．

3) 物理表現層では，物理データとのI/O(インプット，アウトプット)部分を位置付け，部品化の基本方針とする．ただし，実装では，外部のサブシステムとのデータ交換を前提にしているデータと，スケジューリング・テンプレートI内で閉じて使用するデータとに分けて，別の部品とする．

① 物理データI/O（対外）部品

・ファイル，表の単位に一つの部品とし，クラス構造を検討するが，物理的な表現形式に依存する部分が大きいため，全体を制御するための管理クラスを検討する．

・物理的な表現形式として，UNIXファイルとORACLE* データベースを実装する．このとき，ORACLEデータベースには，TIPPLER−ORACLE-APIを使用する．

② 物理データI/O（内部）部品

・物理データI/O部品（外部）と同じ構造を持つが，スケジューリング・テンプレートIの内部で使用するデータを扱うので，物理的な表現形式として，UNIXファイルのみを実装する．

---

* ORACLE：オラクル社のリレーショナル・データベース（登録商標）

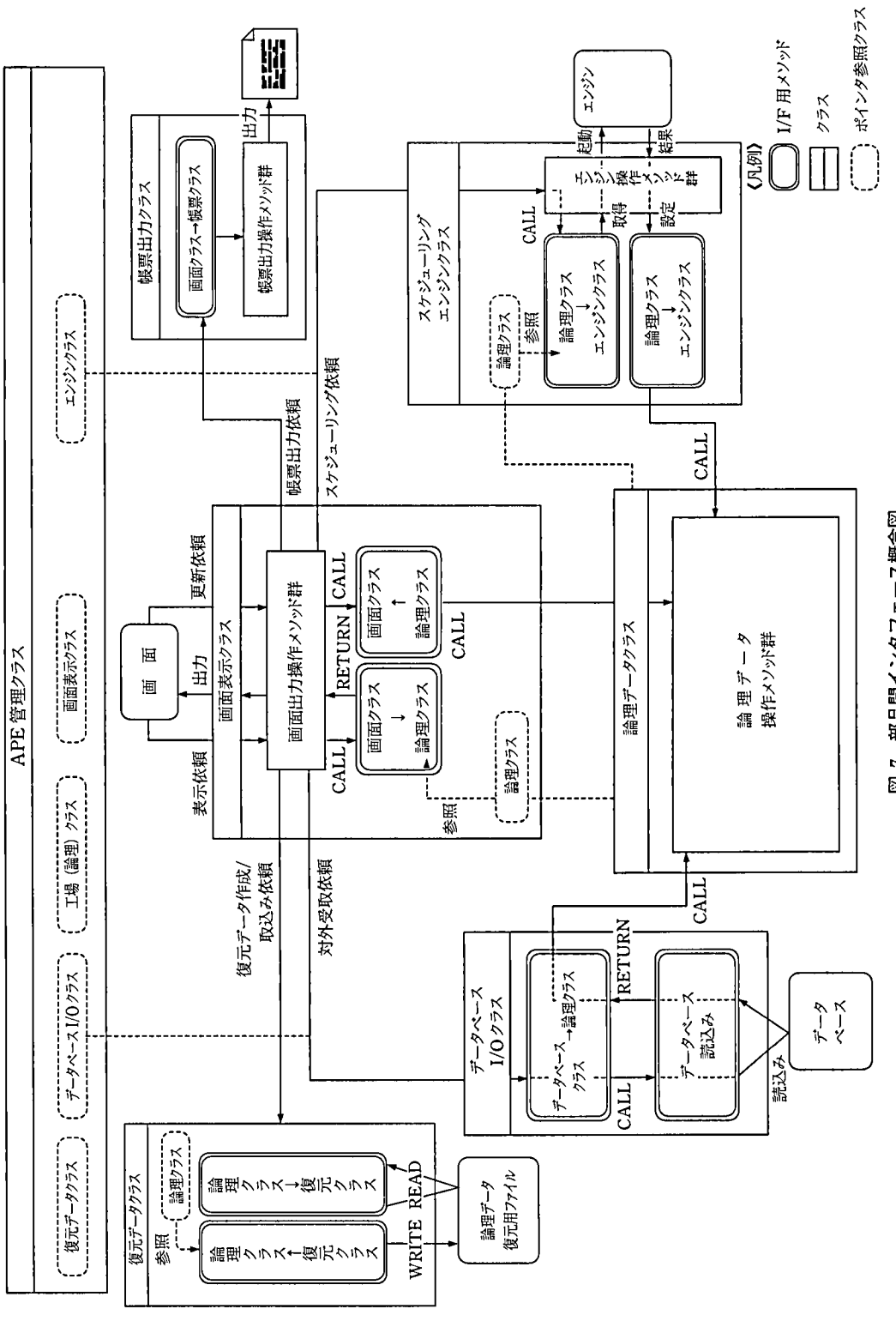


図7　部品間インタフェース概念図

5.2.3 部品間インタフェース

部品化で定義した各部品を有機的に結合するために，部品間インタフェースを定義した．部品間インタフェースは，各部品を構成するクラス(群)のメソッド群として実装する．また，部品間インタフェースの機能は，他のクラスのインスタンス内容の変更は行わないことを前提とし，基本的に，以下のように定義した（図7）．

① 他の部品（クラス）からのデータの取得（他の部品に内装してあるメソッドの呼び出し，またはポインタによる直接参照）

② ①で取得したデータの自クラスへの受け渡し

③ 自クラスのデータ更新に伴う，他の部品（クラス）への変更依頼（他の部品に内装してある変更メソッドの呼び出し）

5.3 GUIの駆使

スケジューリング・テンプレートIでは，現場の計画担当者の使い勝手の良さも重要な要素として捉えた．そのために，GUIをベースに設計，開発を行った．この部分の開発では，とくにTIPPLERが威力を発揮した．代表的な例としては，スケジューリングした結果の全体日程情報をガントチャートアイテム（TIPPLERの標準装備機能）を使用して開発した．

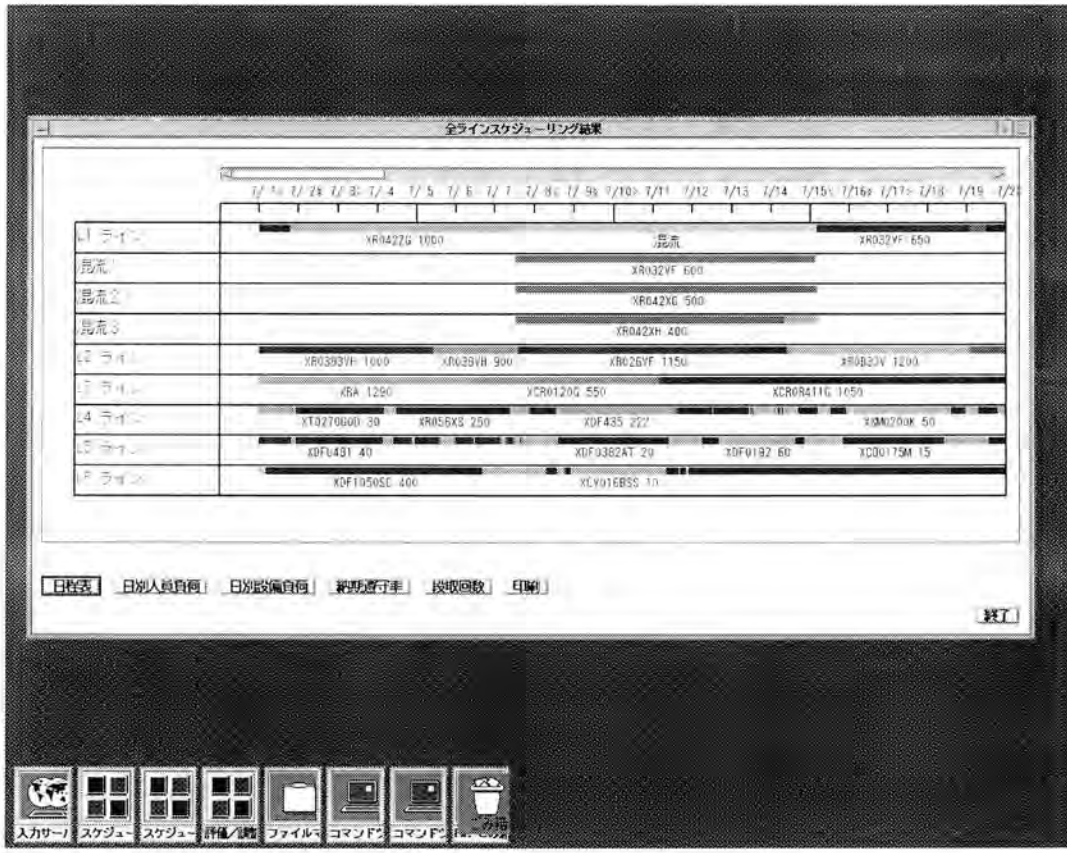


図 8 全ラインスケジュール結果のガントチャートの例

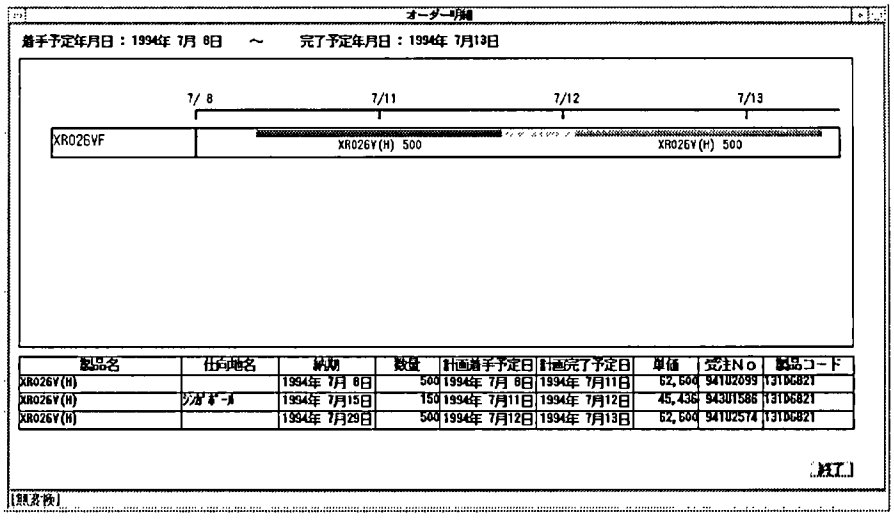


図 9　ロットの詳細情報確認画面の例

全体日程情報ガントチャートでは，全ラインに対しての割付状況を一目で確認できるように，ライン別に生産予定ロットの日程をガントチャート形式で表示した(図8)．このとき視覚的に問題点を把握できるよう，納期に対する生産の予定を，赤系色は納期遅れの発生するもの，緑系色は特に問題のないもの，青系色は納期に対して生産が早すぎるものとし，色を分けて表示した．さらに，問題のあるロットに対して，その問題状況を詳細に確認できる機能（ドリルダウン）も内蔵した．問題のあるロットをマウス操作で選択すると，その詳細情報が表示されるようにした（図9)．

また，同じ画面上に調整機能も内蔵させた．評価画面上から直接，マウス操作で問題のあるロットを捕まえて，移動させたい日付けのところに落とすことで，日程を調整できるようにした．このとき，問題のあるロットの移動に伴い影響が発生する他のロットの計画は，自動的に再スケジューリングされて，調整後の結果がそのまま再表示される．もちろん，他の評価項目の結果も同時に再スケジューリング結果に同期して自動的に更新される．このような機能を装備することで，計画担当者が計画を立案する業務の作業量は飛躍的に軽減すると思われる．

6. スケジューリング・テンプレートI開発に当たってのTIPPLERの評価

述べてきたように，オブジェクト指向プログラミング，ハイブリッド・ストラクチャー，GUIなどTIPPLERの持つ多くの特徴を利用して開発したため，スケジューリング・テンプレートIは，多くの実践的な機能を備えたものになり，そのままでも動かすことができるまでの完成度になった．

スケジューリング・テンプレートIの部品化を検討する上で，オブジェクト指向技術を利用して設計を行った．各部品ごとには設計したクラス構造がある程度，実装時のクラス構造にも使用できたが，そのまま利用できた訳でもない．特に関係オブジェクトやオブジェクトを連結する部分のインタフェースの記述は，工夫が必要であった．

しかしながら，概ね，オブジェクト指向プログラミングができることにより，あまり違和感なく実装のクラス構造に展開することができたと考えている．

また，開発言語としては，実際に比較したデータはないが，実装を約3か月で終了している点から，C言語などより高い生産性で開発できたものと考えている．さらに，今回のシステムは，ユーザ・インタフェース部分が非常に重要であった．この部分の開発において，操作の容易性，見栄え（グラフや表などを組み合わせて表現できる）の良さなどを追求する上では，TIPPLERのframeの概念が有効であったと考えている．とくに，frame内へのメソッドの組み込み機能は，操作性の向上の面で有効であった．たとえば，スケジューリング業務を行う上では，評価画面上から直接，調整作業を行い，同時に自動的に再スケジューリングする機能は必ず欲しい機能である．このような機能を実現するためには，frame内へのメソッドの組み込み機能は，欠かせないものであった．

しかしながら，開発環境としてのTIPPLERに望む点もある．デバッグ，テストの工程において，強力なツールが必要である．インタープリタ機能もあったが，まだ実用レベルでは，不十分なものであり，今回の開発ではあまり効果をあげられなかった．今後の課題として，この部分の機能強化を節に望んでいる．

7. おわりに

実際にスケジューリング・テンプレートIを利用して，ユーザのアプリケーションシステムを構築する上では，新しいシステム化のアプローチが必要になる．スケジューリング・テンプレートIを利用した新しいシステム化のアプローチでは，初めに，スケジューリング・テンプレートIを使用してあるべき姿をイメージし，その後で，ユーザの要求をスケジューリング業務手順モデルのシステム機能ごとに整理していく．このようにすることで，ユーザの曖昧な要求を整理でき，さらに，これまでより，短期間，低工数で，ユーザの実状に合ったアプリケーションシステムを構築することができるようになる．

今回開発したスケジューリング・テンプレートIは，機械加工/組立業の生産現場の割付，作業計画を業務範囲としている．従って，他の業種/業態に対する適用は勧めにくい．他の業種/業態に対するテンプレートの充実も今後行っていく必要があると考えている．さらに，部品に関しても，かなり限定した形での部品になっている．この点も，テンプレートの充実と合わせて，今後，改良・充実していく必要があると考えている．

参考文献 [1] J. ランボー, M. ブラハ, W. プレメラニ, F. エディ. W. ローレンセン, オブジェクト指向方法論 OMT, トッパン
[2] 神近博三, 低成長時代の主役に躍り出たスケジューリングシステム, 日径コンピュータ, 1993
[3] システム制御情報学会,「生産スケジューリング・シンポジウム '95 講演論文集」, 1995

執筆者紹介　田　頭　　浩（Hiroshi Tagashira）

1958年生．1980年上智大学理工学部数学科卒業．同年日本ユニシス(株)に入社．製造業における生産管理システムの適用サービス，製造アプリケーションの開発に従事．現在，製造工業システム第2本部生産システム部に所属．

クライアント/サーバによるシステム構築とTIPPLER

多田　哲

1. はじめに

PCの高機能化と低価格化は，企業内におけるオフィスの環境を劇的に変えようとしています．また，一人一台のPCがある環境では，PCなしでは全く仕事ができない程その有用性を高めています．そして今や，これらのマシンがネットワークを通して互いに接続され，会社の基幹システムそのものになろうとしているのです．

このような環境の変化に対応して，TIPPLERもその稼働プラットフォームをPC環境に拡大すべく開発，商品提供を行ってまいりました．ここでは，TIPPLERのPC版であるTIPPLER for Windowsからこの程発表したTIPPLER/V（WindowsNT版）の開発に至った背景を考察する事により，TIPPLER/Vの位置づけとPCによるクライアント/サーバ環境の構築について述べてみました．

2. TIPPLER開発の背景

2.1 TIPPLER（UNIX）版の開発背景

TIPPLERは，1991年に株式会社野村総合研究所と日本ユニシス株式会社が共同で開発し，販売を開始した統合開発環境です．

当時，コンピュータシステムはデータを“Process”する技術だけでなく，“Informate”する事を要求され始めた頃でした．言い換えると，データを誰かある人に与える時に，いかにその人にとって意味のある情報として与えられるか，またはある人がデータを操りながらいかに自分にとって必要な形に加工できるかが，コンピュータシステムに求められるようになってきたのです（いわゆる「知的活動支援システム」）．

このようなシステムには，使いやすいユーザインタフェースと思考を妨げないレスポンススピードが不可欠なものでした．ところが，当時PCはMS-DOSのキャラクタベースのシステムが中心で，今話題のWindowsはまだ立ち上がっておらず，Apple社のMacintoshのウィンドウシステムが先行して高いユーザインタフェースを実現している程度でした．一方，PCの心臓部であるCPUもまだスピードが遅くかつ高価でした（もちろん”現在と比べれば”の話ですが）．また，そのシステムの信頼性もPCではまだ不安なところがありました．

そこで，処理のスピードと操作性の面から当時広くエンジニアリングワークステーショ

ンとして利用されていた UNIX システムをターゲットにソフトウェア開発環境を開発，提供したのが TIPPLER です．

UNIX システムは，当時の PC とは比べものにならないほど速い CPU，30 MB～40 MB のメモリが使え，処理スピード，Windows の操作感など，我々の目的にぴったり一致した環境でした．ところが，もともとエンジニアリング分野を得意としていた UNIX ワークステーションには，前述したようなアプリケーションを作成する適切な開発環境がほとんど無かったのです．C 言語や C＋＋言語ではあまりにも生産性が低く，使い易いアプリケーションを作るには大変な期間と工数を必要としていました．

結局「知的活動支援アプリケーション」を開発するプラットフォームとして自分たちで欲しい開発環境は自らが作らなければならなかったのです．それが TIPPLER(知プラ)です．この開発コンセプトが市場にマッチし，UNIX 版の TIPPLER は，大変な評価を得て数多くのアプリケーションが開発されユーザに利用されました．代表的なものとして

TIPPLER-ALM（資産負債管理システム）

TIPPLER-PSS（店舗等での要員スケジューリングンシステム）

などがアプリケーションとしてもベストセラーになりました．また，AI エンジンや数値分析用のエンジンのユーザインタフェース部分を TIPPLER で作成し，工程計画やバスダイヤ編成システムなどの使いやすいアプリケーションが構築されてきました．この間，弊社内の SE の 4 人に 1 人は TIPPLER プログラミングの教育を受け，弊社におけるワークステーションでの開発業務のほとんど全てに何らかの形で TIPPLER が採用されました．このように TIPPLER は実際に使われることによってその機能を充実させ，信頼性を上げてきました．

2.2　TIPPLER の Windows 3.1 版の開発背景

ところが時代の流れは早く，PC の性能の向上はすさまじいと言って良いほど急激でした．ソフトウェアではなんと言っても DOS/V OS と Windows システムの出現が大きなインパクトでした．

DOS/V OS は寡占状態の日本の PC 市場を世界に開く役割を果たしました．（日本語の表示処理をソフトウェアで行えるようになったのでハードウェアは世界的に一つの仕様ですむことになったわけです．）この結果 Compaq や DELL などの PC メーカが相次いで日本に上陸を果たし，PC ハードウェアの価格が一挙に半分以下に低下しました．最近では，秋葉原は売り上げの 50 %以上を PC が占め，家庭電器街ではなく PC 街だと言われています．

また，Windows 3.1 は発売一年で世界で 140 万本の出荷を記録し，ソフトウェアとしては記録的なヒット商品になりました．

この二つの出来事は，ハードウェア価格をも一挙に引き下げました．PC でもメモリ 10 MB 以上が当たり前で，内蔵ハードディスク 1 GB などというのも珍しくなくなりました．（つい 5～6 年前は 40 MB のハードディスクでさえ感動的な容量に感じたものです．）また Windows 3.1 の使いやすい操作感は，ユーザに理解しやすいアプリケーションを作成するのに必要な要件を満たし，ディスクやメモリは大量生産のおかげで急激に安くなり，また，

Intel 社のチップは以前のワークステーションと遜色ないスピードを持つようになって，システム開発者はディスクやメモリの大きさの制限をあまり意識せずにプログラムを開発する事が出来るようになったのです．

このような環境の変化により，4 年前の TIPPLER 開発当時望んだ環境が PC 上で実現できるようになってきたわけです．そこで 1993 年より TIPPLER for Windows の開発を開始し，翌年 7 月より販売を開始しました．もちろんここでは，TIPPLER の UNIX 版で培われた様々な機能やノウハウが十分盛り込まれた開発環境として提供されました．ところが実際にアプリケーションを開発してみると，いわゆる「MS-DOS の壁」にどうしても突き当たることになりました．つまり，Windows 3.1 と言えども，またいくら物理的に大きなメモリを持とうとも，どうしても 16 bitOS である MS-DOS ではアプリケーションに限界があるということです．

ここで，我々は TIPPLER の”使われ方”を再認識する事にもなったのです．すなわち，”TIPPLER を使って開発するアプリケーションは非常に複雑で高度なものが多く，単なる SQL 発行クライアントを開発するだけにはとどまらない”という事実でした．いつも問題になったのは，1 セグメント 64 KB の制限や，コンベンショナルメモリ 640 KB の壁でした．TIPPLER で開発するような高度なアプリケーションを動かすには，“config. sys”や“autoexec. bat”などを様々に調整しなければならず，さらにそれらを TCP/IP のネットワーク環境に適合させるにはかなりの高度な知識とテクニックが要求されました．

2. 3 TIPPLER/V の登場背景

前述したような環境は，PC をサーバまたはワークステーションとして利用していく場

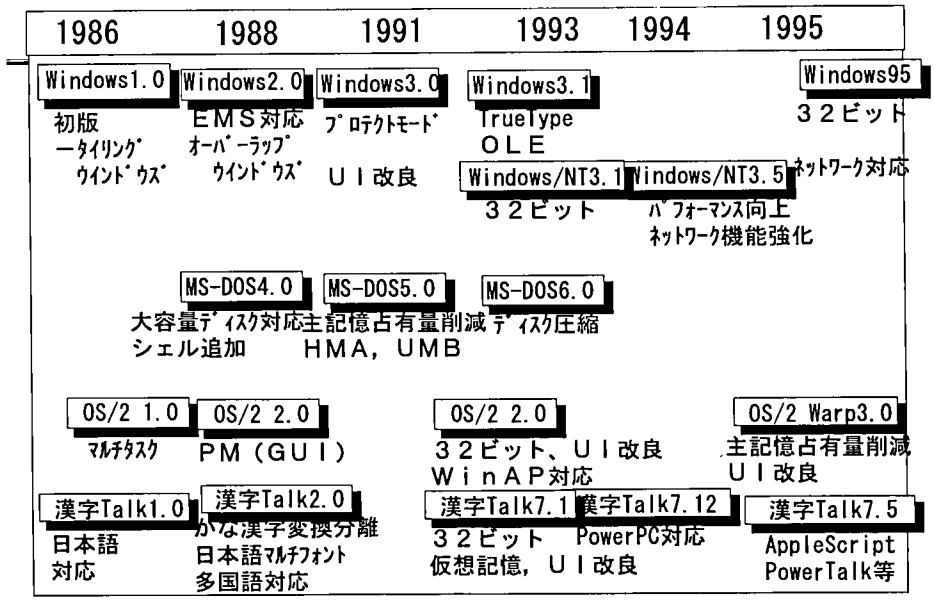


図 1　各 OS（日本語版）の変遷の系譜

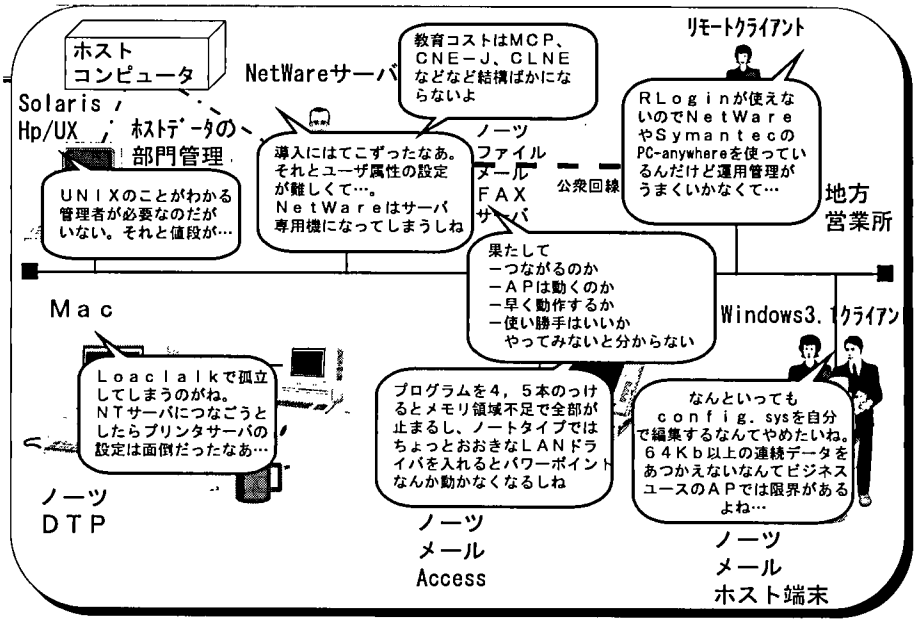


図 2 クライアント/サーバ混合環境での現状の問題点

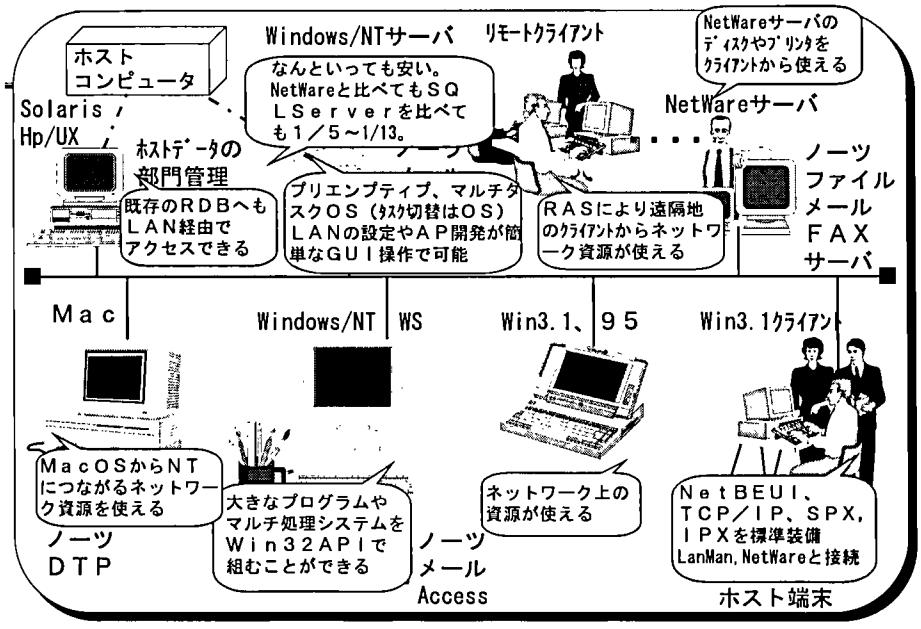


図 3 Windows/NT サーバ環境の特徴

合誰でも突き当たる壁でした．これらの壁を取り払い，UNIX に十分対抗できる OS として発表されたのが WindowsNT です．これは，完全に 32 bitOS として開発され，最新の CPU チップの能力を 100 %引き出す能力を持った OS です（図 1，図 2，図 3）．この段階に来てやっと低価格な DOS/V マシンを使ってかなり高度なアプリケーションが作成できる環境を手に入れることになったのです．TIPPLER/V は，この様な，“PC によるエンタープライズシステム構築”，“PC の高度なビジネスワークステーションとしての利用”をターゲットにした新しい展開の基本開発環境として位置づけられます．

3. WindowsNT 時代の業務領域の見方と TIPPLER/V の適用範囲

それでは，WindowsNT や Windows 95 が発表された現在，TIPPLER/V のようなアプリケーション開発ツールを適用するにはどのような分野があるのでしょうか．ここでは，図 4 に示すように，縦軸に“その業務の定型度合い”を，そして横軸に“その業務の取り扱うデータ/トランザクションの量”を取って様々な業務をマッピングしてみました．この図の“データ量またはトランザクション量”が小さく“定型度合い”が低いエリアには，いわゆるエンドユーザ・コンピューティング（以下 EUC）ツール（ワープロ，表計算，DTPR など）が位置づけられ，“データ量またはトランザクション量”が大きく“定型度合い”が高いエリアは，企業の基幹業務系のアプリケーションが考えられます．言い換えると，左下のエリアに行けば行くほど PC の得意な範疇で，右上のエリアに行けば行くほど汎用機の得意な範疇と言う事もできます．

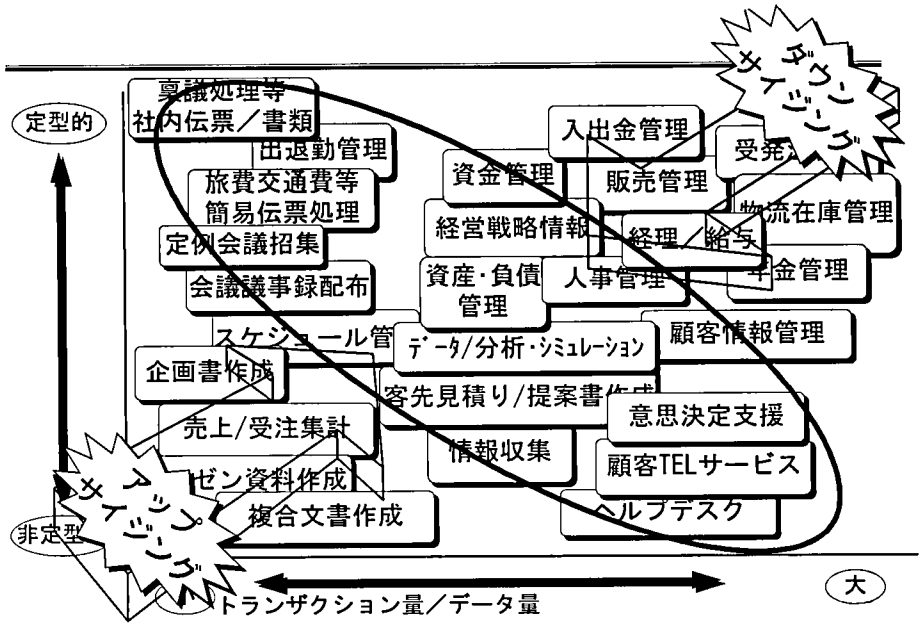


図 4　業務アプリケーション対象領域

WindowsNT もしくは UNIX サーバと PC クライアントを LAN で接続したいわゆるクライアント/サーバ型のシステム（以下 C/SS）は，この図のちょうど中心あたりの楕円で囲んだエリアを守備範囲とするとみると，“ダウンサイズ”とはなにもすべての汎用機上のアプリケーションを C/SS に移植させることではなくて，コストと実行パフォーマンスのバランスにおいて最適なアプリケーションを C/SS の上で開発していくことを指すと考えられます．また，PC を LAN を経由してサーバに接続させることは PC のディスクエリアを拡大して，データを共用することのみではなく，情報を共有することによりグループによる生産性の向上に寄与するようなアプリケーションを構築していくことを指し，これをアップサイズと呼んでもかまわないと考えています．

このような“ダウンサイズ”，“アップサイズ”は PC の使用現場では同時並行的に起こりつつありますが，このようにマッピングしてみると，この図のちょうど中心あたりが今現在ちょっとした谷間になっていることが理解できます．すなわち，ダウンサイジンング的発想とアップサイジング的発想がぶつかりあい，ハードウェアと OS は十分な機能を持ちながら使いやすい開発環境があまり提供されていない分野となっています．

TIPPLER/V は，（あるいは WindowsNT）は，まさしくこのエリアをターゲットとして開発されたものです．これから当然要求される“PC の基幹業務系への適用”と言う流れの中で，TIPPLER/V の機能性が発揮されることになるでしょう．

4. TIPPLER/V の位置づけと特徴

TIPPLER の UNIX 版は，弊社が顧客向けアプリケーションを開発するのに使われたり，お客様が開発ツールとして利用され，その都度ご要望に応えて機能拡張をしてきました．たとえば，グラフはスキャターグラフやレーダーチャート，ガントチャートなど 11 種類以上を提供しています．さらに，UNIX 版の特徴としてマルチプラットフォーム化を推進し，最初は弊社の US ファミリ，次に SUN マシンでも（つまり弊社の OEM マシン以外でも）提供できるようにしました．次にヒューレットパッカード社のワークステーションへの移植を行いました．これによりワークステーションマーケットの大部分をカバーすることができる事となりました．これらの UNIX ワークステーションの世界で蓄積されたノウハウと新しい時代の要望である C/SS の開発ツールとして作成したのが TIPPLER/V です．

以下，TIPPLER/V の位置づけと利用形態を中心にして，これからのシステム構築の方向性を探ってみたいと思います．

4.1 TIPPLER/V の位置づけ

先に述べたように，今までの C/SS は，接続されるクライアントの数もせいぜい 10 台程度で，アプリケーションもデータベースを検索するだけの（更新の無い）ものが大多数でした．この実体はデータベースサーバと GUI を備えた端末が組合わさり，SQL とデータをやりとりするだけで，真の意味でクライアントとサーバの関係にはほど遠いものがあり，クライアント側は情報表示装置，サーバもデーター時保管庫でしかなかったともいえます．

現在の顧客における C/SS 環境においては，それでも部門処理としては十分であるとい

う場合もあります．つまり，C/SS ではホストからダウンロードされたデータやエンドユーザ部門で蓄積しているデータをいかに見やすく加工し分析するかのみをその目的としているケースです．この場合においては図 5 のようにニーズが PC の利用形態の複合化に従って，スタンドアロンからグループコンピューティングのニーズへと変化していく（図では左から右）と考えられます．TIPPLER for Windows が昨年競合した領域がここで，図 5 に示したような要望が次々とお客様からあがってきたのでした．しかしこのような領域のみでの競合では，TIPPLER の本来のポテンシャルは発揮できないと考えています．

WindowsNT や Windows 95 が普及してくると，UNIX で開発してきたような本格的なアプリケーションの，32 bit クライアント PC による本格的な C/SS での開発が，言い換えると大規模なエンタープライズシステムへの導入に向けた開発が始まると考えています．先進企業では，すでに長期計画のもと汎用機の思い切った削減と WindowsNT による基幹業務構築が始まっています．

ごく初期におけるクライアントアプリケーションの構築には，マイクロソフト社の EXCEL や VisualBasic，または Access のような EUC ツールを使うことが多かったようです．ところがこのようなツールは個人的に使うには非常に使いやすいツールですが，大規模な開発になるといくつか問題が起きているようです．まず，大規模開発では当然何人もの開発者が関係してプログラム開発にあたります．その中では，設計をするもの，プログラミングをするものなど仕事の内容においてもいくつかの作業種類が発生します．このようなプロフェショナルなシステム開発作業においては，本来当たり前ですが，仕様書(外部仕様書，内部仕様書，テスト仕様書など)が必要になります．ところが EUC ツールはも

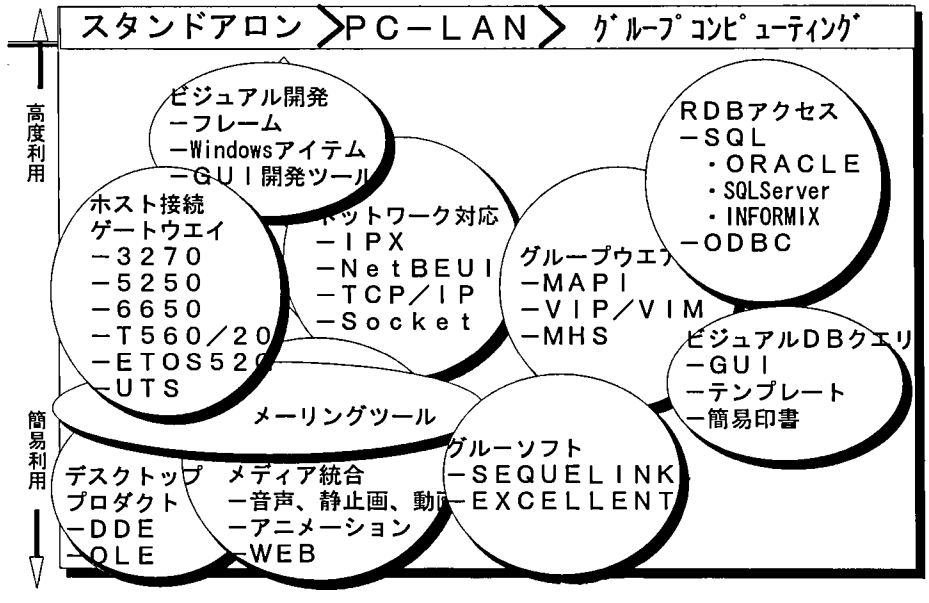


図 5　アップサイジング時に開発ツールに求められる要件

ともと個人での利用しか考えられていないので仕様書による開発がとてもやりにくくなってしまいます．また，このことと関連して保守性の問題があります．アプリケーションは通常最初に作成したままで使われることは希で，多くの場合変更が発生することはみなさん経験していらっしゃる事だと思います．ところがプロフェッショナルが作成した業務アプリケーションは，プログラミングした人と保守する人が必ず同じとは限りません．むしろ，多くの場合別の人が保守するのが通例でしょう．保守のためにプログラム仕様書を何らかの形で残すことは不可欠です．

また，ごく初期段階を過ぎるとデータベースフロントエンドプログラム専用の開発ツールが出てきました．PowerBuilder や SQLWindows などが代表例です．これらは，専用開発ツールだけあって，データベースに SQL を投げて結果を表示するだけですむプログラムでは，非常に高い生産性を示しました．Windows 3.1 クライアントの機能ではせいぜいこのレベルまでの利用にしか耐えられなかったと言うことができると思います．

しかし，PC の機能向上と WindowsNT の登場は，この程度のクライアントアプリケーションではとてもその機能を使いこなす事はできず，より高機能な開発環境が要求されてきます．このニーズこそが TIPPLER/V 開発の最大の要因となっています．

現在，クライアント/サーバシステムの開発にもやはりプログラミング言語が必要であることが再認識されています．また，それがオブジェクト指向であればなおさらプログラミングが楽になり，変更も容易でしょう．TIPPLER/V がオブジェクト指向言語である点が，保守性やプログラミングをより容易にし，それらの生産性を向上させていると考えています．オブジェクト指向について一言で言うと TIPPLER は TIPPLER 自身で扱うデータを一元管理しています．つまり，クラス定義という形でデータ構造の継承関係（親子の関係の集合体と考えて下さい）を定義しておけば，プログラマはそのデータの一元性に関してプログラミングの考慮がいらないわけです．実はこれはとても大変なことで，たとえば人事のアプリケーションを想定した場合，ある人の所属を変えたとします．そのとき別の Window を開いていて，その課の構成員一覧を表示していたとしましょう．データを一元管理していない言語では，所属の変更を構成員を表示している Window に反映させるには何かしらのプログラミングが必要になります．ところが TIPPLER では，その個人のデータは必ず一元管理していますので，プログラマがこのことに関して何の考慮をしなくても，全てに対して変更が反映されます．この場合だと，所属を変更すれば，その構成員一覧を示す Window でも即座に構成員が変更されるのです．TIPPLER/V はまさにこの分野での活用を意識して作成された開発環境なのです．また，変更の容易性はプロトタイピングという新しい開発手順を容易に実現できるようになりました．システムの利用者に直接見てもらいながらの開発は，でき上がったシステムの満足度向上に貢献します．

前とは違った形での適用範囲のイメージを図 6 に示します．横軸は必要とされる機能範囲を示し，縦軸は利用者（開発者）がどういった規模のアプリケーションを開発するかを示しています．この図でも示しているとおり TIPPLER/V は，部門あるいは会社といった単位で利用する，ある程度定型の基幹業務の開発に向いています．

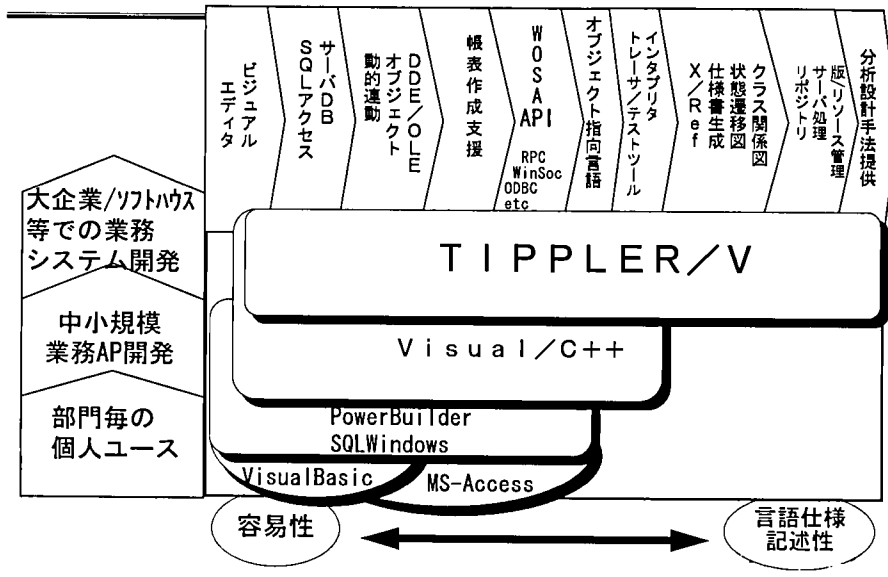


図 6　使用する規模と必要とされる機能

4.2　TIPPLER/V の特徴

このような企業単位のシステムを C/SS 上で実現させるためには，既存のシステムとの併存を前提とした移行方法の提供や，いわゆる「Legacy Wrapping」と言われる既存システムを生かしながらの新システムの導入が必須となります．また，部門ごとの規模での C/SS 利用ではあまり考慮する必要のなかった，運用管理やホストとのトランザクション連動，既存ソフトウェアを活かす移行ツール，ホスト SE パワーを活用する COBOL などの言語の移植，大規模グループ開発のための CASE 機能，多部門間をまたがるワークフロー，既存アプリケーションプログラムとの操作感統一のためのユーザインタフェースの提供など（図 7，図 8）一気に考えるべきファクターが増大します．これらすべてを TIPPLER/V が提供することはできませんが，これらの図にあげられたような他のプロダクトと連動する API（アプリケーションプログラムインタフェース）を TIPPLER/V は数多く装備していることは非常に大きな特徴といえます．

いわゆるデータベースフロントエンドツールが持たず，TIPPLER/V が持つ特徴をまとめると次のようになります．

■開発/保守/教育生産性が高い

—オブジェクト指向の開発手法（OMT 準拠）を提供している．

—オブジェクト指向の言語 Uniscript を使用できる．

—大規模開発の支援機能がある．

—ソースコードの量が少なく保守しやすい．

—簡易言語 Uniscript のみの学習でよく教育効率がいい．

—テストツールが完備している．

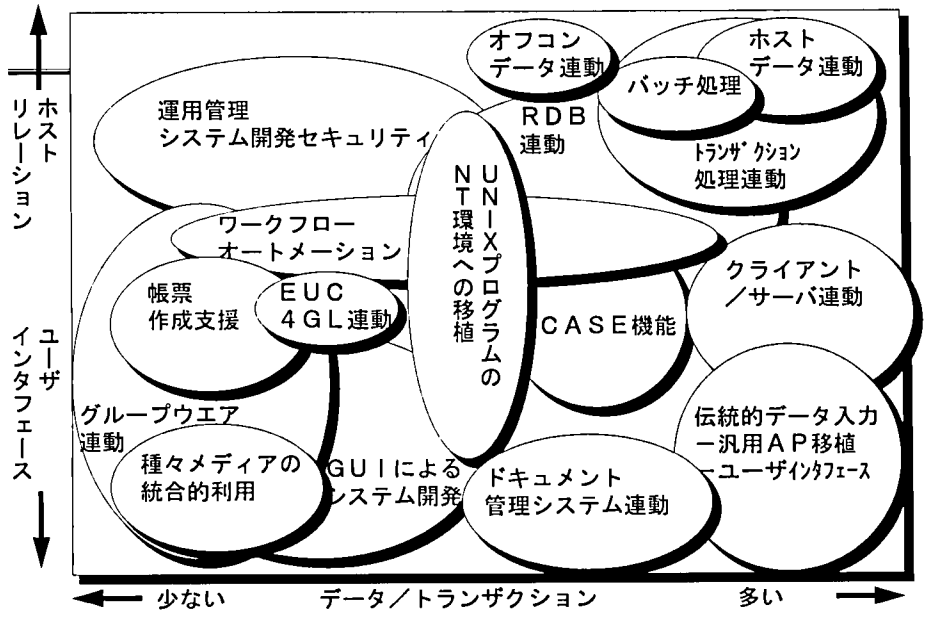


図 7 WindowsNT での開発環境

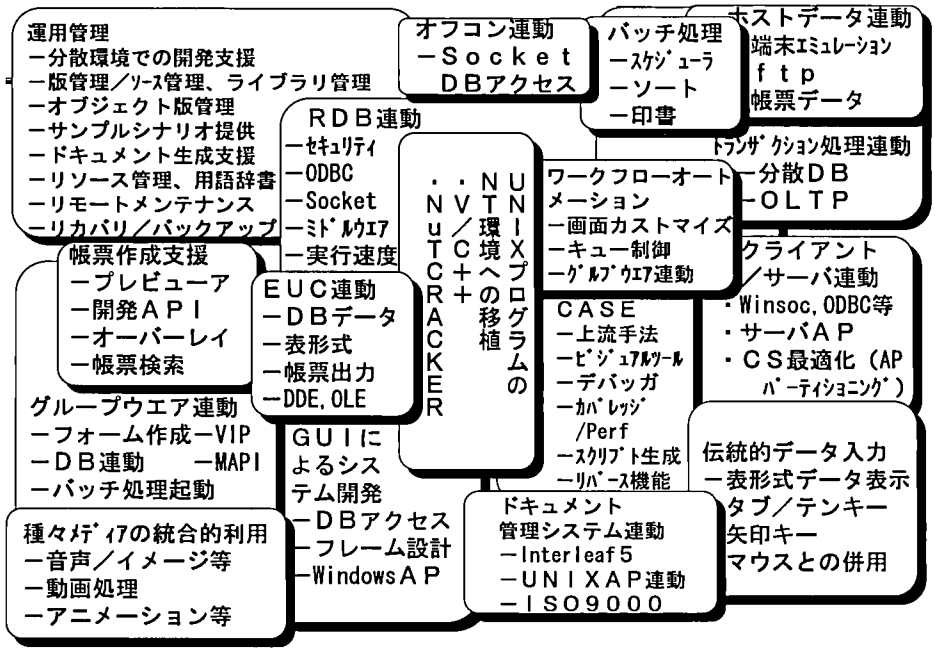


図 8 WindowsNT での開発環境に要望される機能

■基幹業務に必要な機能を備えている
—ORACLE や Sybase, ODBC などのデータベースインタフェースを備えている.
—ソート機能を持っている.
—TCP/IP, Socket, などのネットワーク経由の手順をサポートしている.
—GUI 開発はもちろん, カーソル制御, 矢印キー, tab キーなどの制御が可能である.
■実行性能が高い
—オブジェクトコードは V/C++コンパイラにより生成されるため高速である.
—サーバとクライアントの間でトランザクション転送ができデータベースアクセス高速化のためのカストマイズが可能である.
■日本語諸機能がサポートされている.
—日本語によるプログラミングが可能である.
—帳票作成支援機能があり, オーバーレイによる帳票設計, 開発, 実行が可能.
—帳票データの蓄積, 検索, 編集が可能である.

5. TIPPLER/V の仕組み

それでは, ここで簡単に TIPPLER/V の構造について述べてみましょう (図 9). TIPPLER の中心となるのは, Uniscript と言う独自の言語です. TIPPLER は, フレームと呼ばれる画面上の Window のデザインや, メソッドという処理手続きなど全てのコーディングが Uniscript という言語で記述されます. これが図 9 の上から二つ目の箱に示された Uniscript 言語プログラムの部分です. その上の二つの四角は, この Uniscript を記述するための過程です. 左の四角のテキストエディタとは, Uniscript を記述するために必要です

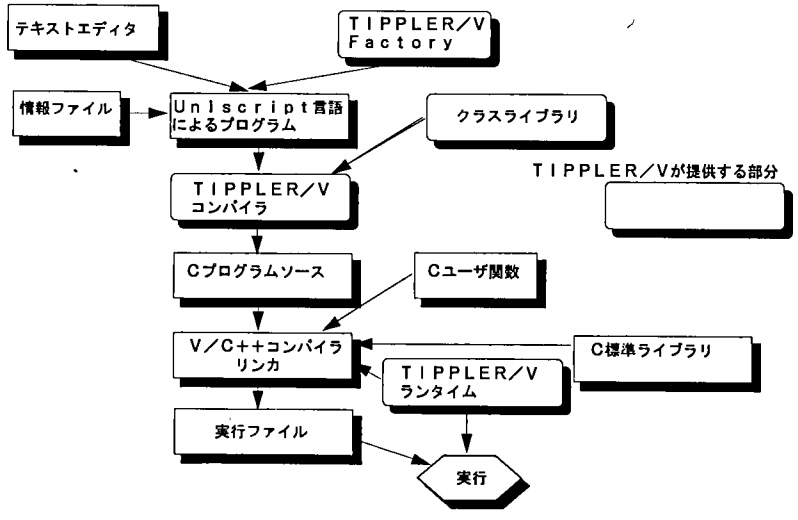


図 9 TIPPLER/V の構造

が，使い慣れたどんなエディタでもワープロソフトでも構いません．一番上右側の四角の TIPPLER/V Factory とは，TIPPLER/V の提供するツール群で，ビジュアルな画面デザインや，見たい部分や相互参照情報を表示するブラウザ，クラス図をビジュアルに示すなどの開発に大変有効な機能がたくさん入っています．

TIPPLER を様々な目的で活用する際，非常に有効なのが右側の四角のクラスライブラリです．これは，色々な処理をライブラリ化したかたまりのファイル群で，様々な機能を実現させたライブラリがあります．この中には TIPPLER が提供している OLE クラスライブラリ，データベースと接続するための ODBC, SEQUELINK や Excellent といったミドルソフトウェアとのインタフェースを提供するクラスライブラリなどの他に，ユーザの作成したグラフやマルチメディア機器などのインタフェースなどたくさんのライブラリがあります．

このようなクラスライブラリなどを活用して記述された Uniscript のソースをコンパイルするのが TIPPLER/V コンパイラです．TIPPLER/V コンパイラは，Uniscript のソースを Visual C++言語のソースコードに変換します．これをさらに Visual C++言語のコンパイラを使ってコンパイルしたものが TIPPLER/V のオブジェクトコードとなって，TIPPLER/V のアプリケーションが完成します．

これが，大まかな TIPPLER/V のアプリケーションができる仕組みです．もうお気づきでしょうが，TIPPLER/V でアプリケーションを開発した際の最終生成物は，Visual C++言語のオブジェクトコードです．この結果，TIPPLER/V で作ったアプリケーションの実効速度はすばらしく早いものになっていますし，Visual C++言語を介した他のアプリケーションとの親和性がとても高くなっています．

6. TIPPLER/V を使ったシステム

TIPPLER/V を使った最も代表的なクライアント/サーバシステムでは，サーバの WindowsNT 上にリレーショナルデータベース (RDB) を置き，クライアントを NT ワークステーションとすると，サーバとクライアントとによる協調的な負荷分散を行いながら，実行効率のよいかなり複雑なアプリケーションを構築することができます．また，データベース上の数値を使った様々なシミュレーションプログラムや処理を組み上げるには，TIPPLER/V の様な言語体系を備えた環境が必要になります．

たとえば，TIPPLER/V はトランザクション転送という機能を備えています．これを利用するとサーバ上の TIPPLER とクライアント上の TIPPLER/V 間で直接通信してデータを共有する事ができます．これにより，負荷を分散した統合的なクライアント/サーバシステムを構築する事が可能です．すなわち，サーバ側でデータを総て TIPPLER/V のインスタンスとして取り込み，クライアントサイドはデータ処理だけを行うなどプロセスの分散を計った真の意味のクライアント/サーバを構築する事ができるわけです．

また，WindowsNT をワークステーションとして適用する機会も今後増加すると考えられます．基幹業務では，ファイルシステムのセキュリティなどが重要な要素になりますが，WindowsNT はそれらの機能を十分備えているからです．TIPPLER/V は，UNIX の実績

が示すとおりワークステーションソフトとしても有効です．最初に述べた，工程計画，人員配置などシミュレーション機能を要求されるソフトや，エラー対応などを作り込む必要がある場合など，言語機能が充実しているTIPPLER/Vが非常に有効です．もし，TIPPLER/V以外でこれらの仕組みを作り込もうとするとVisual C++位しか考えられませんが，生産性と保守性においては比較にならないほどTIPPLER/Vは優れてるでしょう．もちろんTIPPLER/VはWindowsの標準機能であるOLEやDDE機能をサポートしています．これは，TIPPLER/Vとその他のPCソフトウェア間の連携を可能にしています．たとえば，TIPPLER/VでRDBより引き出した情報を，マイクロソフト社のExcelに張り付けてグラフ化したり，お絵かきツールで描いた絵をTIPPLER/Vの画面に張り付けたりといった事が簡単にできるのです．このようなEUCツールとの共存は，企業内のコンピュータシステムそのもののあり方を変える可能性を持っています．

最近注目を浴びているグループウエアやワークフローツールのGUIの開発ツールとしてもその生産性や保守性から大規模なシステムではTIPPLER/Vを選択肢に入れるべきと考えます．

TIPPLER/Vを使ったシステム開発はまだまだ始まったばかりで，たくさんの事例があるわけではありませんが，ここ1〜2年の内にはこのような形のC/SSやワークステーションシステムが多くの企業内の基幹システムに採用されることとなるでしょう．

7. TIPPLER/Vの今後の計画

マイクロソフト社はWindowsNTに加え，11月23日にはWindows 95の発売も発表しています．弊社ではすでにWindows 95のベータ版においてTIPPLER/Vの動作確認を開始しました．TIPPLER/Vもこのような動きに同期した機能やモジュールを提供していく予定です．TIPPLER/Vではこれらの機能を生かして

—ビジュアルな開発ツールFactoryの機能拡充

—インスタンス転送機能の強化

など新しいツールの提供や機能拡張を実施していく予定です．また，クライアント/サーバ環境における開発ツールとしては数少ない純国産ツールとしてのメリットを生かして，ユーザの方の声をどんどん取り入れていくつもりです．(その結果生まれたのが帳票作成ツールやデータ入力の際のタブキー/矢印キーによるカーソル制御ライブラリです．米国製のソフトウェアでは，日本における帳票出力の要望に応えられないようです．)

また，近日中にインターネットを利用した情報サービスを開始します．ここでは，各種の情報をリリースやご意見をお聞かせいただく場として運営します．また，ユーザが作成したクラスライブラリなどを互いに発表して利用する場とも拡充していきます．

8. おわりに

以上述べてきたように，5年以上にわたるTIPPLER開発の過程から，TIPPLER/Vが持つ機能とマーケットのニーズの明確化，言い換えるとTIPPLER/Vを適用すべき分野の明確化がかなり進んだと考えます．

今後は，いかに市場ニーズを先取りして的確なタイミングでソフトウェアをリリースしていくか，またそのための，手順確立，組織的サポートがより厳しく要求されると考えています．本文の様なマーケティング議論をどのようにして今後の製品開発に反映させていくことができるかが，製品開発/マーケティング両サイドからみて，成功の重要なファクタとして認識されることが必要と考えます．

執筆者紹介 多 田 哲（Tetsu Tada）

1954 年生．大阪大学工学部金属材料工学科卒業．1977 年 4 月日本ユニシス(株)入社．製造流通分野のシステムサービスを担当，1988 年日本ユニシス情報システムに出向．ホテル/レストラン POS システム等流通/サービス分野の開発・サービスを担当．現在オープンソフトウェア事業部マーケティング企画部長．TIPPLER の他 Staffware, Lotus Notes 等の企画マーケティングを担当．

■

TransIT Open/OLTP

日本ユニシスでは本年8月より2200シリーズ，Aシリーズ，U6000シリーズおよびWindows NTとWindows 3.1で稼働する分散トランザクション処理システム構築用基盤ソフトウェア「TransIT Open/OLTP」（トランスアイティ・オープン・オーエルティーピー）の販売を開始した．

本製品はクライアント/サーバ方式に基づいた，2階層または3階層の分散トランザクション・システムを構築する基盤ソフトウェアであり，ユニシスのメインフレームから他社PCまで，広範なハードウェア・プラットフォームに対応している．大規模なエンタプライズ・クライアント/サーバ・システムはもとより，中小規模のクライアント/サーバ・システム構築の基盤ソフトウェアとして使用できる．

1. OLTPの概要

1.1 OLTP（On-Line Transaction Processing）とは

現代の企業活動は，コンピュータ・システムへの依存度をますます強めてきている．そして今日では，企業規模の大小を問わず，コンピュータ・システムなしには，正常な企業活動をほとんど考えることができないくらい密接に関係するようになってきている．

たとえば，銀行の預金や為替のオンライン・システム，航空会社の座席予約システム，工場内のCIM，流通業におけるPOSシステムなどをその例として挙げることができる．これらのシステムは，オンライン・トランザクション処理（OLTP: On-Line Transaction Processing）と呼ばれるデータ処理の一形態となっている．つまり，OLTPとは「端末機から通信回線を通して送られてくるトランザクションを受け，データベースを参照/更新して処理を実行し，その結果をオンタイムに応答する処理形態」といえる．

1.2 OLTPのの変遷

初期のコンピュータ利用は，バッチ処理（一括処理）方式からスタートしている．その後，コンピュータ，通信回線，および端末装置の効果的な結合によって，理想的な情報処理システムとして注目されたOLTPは，1960年代から一部の企業で稼働し始め，1970年代には多くの企業でも導入が続いた．そして，適用分野の拡大やトランザクション量の急増によって，大量のトランザクションを高速で処理する必要性から，当時は，OLTPはメインフレームを中核とする汎用機での中央集中型OLTPとして利用が進んできた．

OLTPの利用が進むにつれ，OLTPシステムはしだいに大規模・複雑化し，また処理すべきトランザクションの量も膨大になってきた．そのため，OLTPについて次のようなニーズが高まってきた．

- 大量のトランザクションを効率良く処理できるOLTPを構築したい．
- 短期間でシステムを開発したい．
- 投下したコストを早期に回収できるシステムを構築したい．
- 運用，保守しやすいシステムを構築したい．
- 環境の変化に容易に対応できるよう，柔軟性のあるシステム，拡張が容易なシステムを構築したい．

一方，ハードウェア技術の急速な進歩によるコンピュータの小型化・高性能化・低価格化は，ダウンサイジングの潮流を生み，クライアント/サーバによる分散化と部門コンピューティングが進むにつれ，クライアント/サーバによる分散環境の中でOLTPの機能を使いたいというニーズが高まってきた．

クライアント/サーバ型のシステムの多くは，UNIX OSなどのオープン・システムを使用している．しかし，これらのシステムのOSはOLTPをサポートする機能は弱く，また環境も整ってい

Windows，Windows NT，Visual Basic，Visual C/C++は米国Microsoft社の商標である．

UNIXはX/Openカンパニーリミテッドがライセンスしている米国ならびに他の国における登録商標である．

本稿に記載の会社名，商品名は，一般に各社の商標または登録商標である．

なかった．ところが，最近の分散化やオープン化の流れの中で，OSの改良や機能強化が行われた結果，オープン・システムでも本格的なOLTPが可能となってきた．

オープン・システムをベースとするOLTPとしては，業界標準のX/Open DTPモデルに基づいたオープンなOLTPがあり，クライアント/サーバを前提に分散トランザクション処理の考えを取り入れている．こうした新しいOLTPによって，前述のようなニーズにかなり対応可能となってきている．

1.3　クライアント/サーバ・モデル

X/Open DTPモデルはクライアント/サーバ型処理形態を基本としている．これはアプリケーション・プログラムのうち，処理を要求する側（クライアント）と，これを実際に処理する側（サーバ）とに分割した処理形態である．

従来のトランザクション処理では，端末側からホストへ処理を一方的に依頼するという方法で行われていた．そのため，ホストへの負荷が集中し，トランザクション量が増加するほど処理効率が悪くなるという問題を抱えていた．

この問題を解決するために，一つのトランザクションの処理を，ユーザ・インタフェース処理を中心とする処理部分（クライアント）と，データベースの更新等を中心とする処理部分（サーバ）に分担して処理するという考えが生まれてきた．こうしたクライアント/サーバ型システムでは，クライアント特有の処理と，サーバ特有の処理にそれぞれ適したコンピュータを選択してシステムを構成することによって，効率の高いトランザクション処理が可能となった．

クライアント/サーバ型システムでは，アプリケーション・プログラム（AP）は，クライアント側のクライアントAPとサーバ側のサーバAPから構成され，それぞれの概要は次のとおりである．

1)　クライアントAP

使用者が作成するアプリケーション・プログラムであり，使用者とのインタフェースを担当し処理する．具体的にはGUI(Graphical User Interface)等を提供し，使用者の処理要求の正当性を検査した後，正当ならばサービス要求を生成し，サーバに処理を依頼する．クライアントAPは，サーバからサービス応答を受信後，使用者に処理結果を提示し，処理を終了する．

2)　サーバAP

使用者が作成するサービスの集合である．サーバAPは個々のサービス要求を処理し，処理結果をクライアントに送信後，新たなサービス要求を受け付ける．

クライアント/サーバ・モデルは，分散トランザクション処理を実現するための最適な処理形態である．分散トランザクション処理システムにクライアント/サーバ・モデルのアーキテクチャを用いることにより，アプリケーションがモジュール化

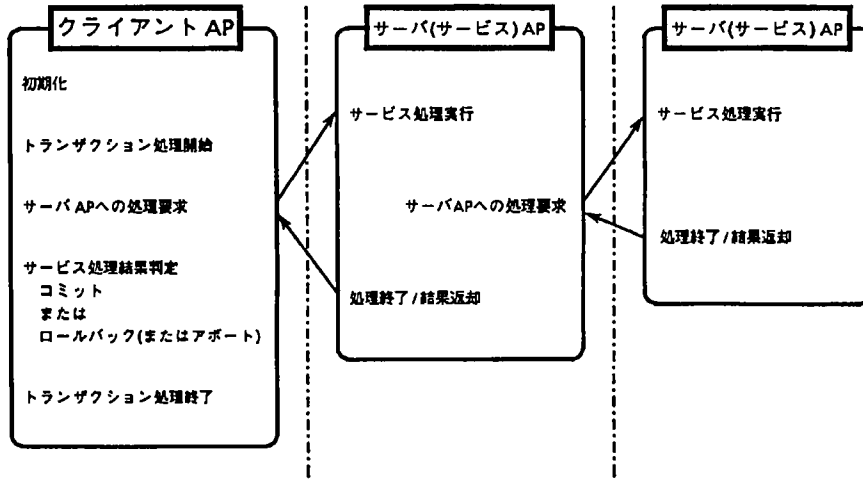


図1　分散トランザクション処理の制御の流れ

でき，拡張が容易となる．

図1は，異なるシステムに存在するデータベースをアクセスするようなトランザクションの処理の流れを，クライアント/サーバ・モデルを基に示したものである．なお，サーバは必要ならばクライアントのように別のサーバにその処理を要求することができる．

1.4 トランザクション処理と ACID 特性

OLTP システムがトランザクションを処理する上で非常に重要な要件がある．X/Open では，この四つの特性を定義し，トランザクションを処理する上で，これらの特性を保証することが必要な要件であるとしている．以上四つの特性を，各特性の英字1文字をとって，ACID 特性という．

- 原子性（Atomicity）
  トランザクション処理完了時のデータベースの状態は，すべてが更新済みであるか，または元の状態に戻されているかのいずれか一方の状態でなければならない．
- 一貫性（Consistency）
  トランザクション処理の終了状態にかかわらず，データベースの内容は一貫性がとれた状態であること．途中の処理順序に関わらずに，処理結果は一定でなければならない．
- 独立性（Isolation）
  多くのトランザクション処理を並行して実行する場合でも，個々のトランザクションの処理は分離し，独立して行える必要がある．異なるトランザクションの処理は，その過程で互いに影響し合うことのないように，独立して処理しなければならない．
- 耐久性（Durability）
  トランザクション処理の結果変化した状態は，そのままの状態で保たれなければならない．仮にそれ以降で障害が発生しても，処理済みの結果に影響を与えてはならない．

1.5 X/Open DTP モデル

X/Open とは，ベンダ（コンピュータ・メーカ）と利用企業によって構成される非営利の国際組織（法人）である．この組織は特定の環境（ハードウェア/ソフトウェア）に依存しないマルチベンダ用の共通アプリケーション環境（CAE: Common Application Environment）を開発することを目的としている．

たとえば，UNIX のアプリケーション・プログラム・インタフェース（API）を制定し，これらの仕様を公開している．このように X/Open ではさまざまな標準を策定しており，このうち分散トランザクション処理（Distributed Transaction Processing）について策定した標準が X/Open DTP モデルである．

分散トランザクション処理（DTP）では，一つのトランザクション処理において，複数の分散されたデータベースを更新する必要があり，このことはそのトランザクション処理の正常終了または異常終了に対応して関連するデータベースの整合性を保証しなければならないことを意味している．DTP モデルでは，このような分散された複数のデータベースにまたがったトランザクションをグローバル・トランザクションと呼び，トランザクション・マネージャ（TM）が中心となって分散されたデータベースの整合性を保証するために調整している．これに対して，従来の非分散環境におけるトランザクションをローカル・トランザクションと呼んで区別している．

X/Open DTP モデルは，一つのトランザクション処理システムを次の四つの構成要素（ソフトウェア）の集合として定義している．

- アプリケーション・プログラム（AP）
- トランザクション・マネージャ（TM）
- リソース・マネージャ（RM）
- コミュニケーション・リソース・マネージャ（cRM）

図2に，各構成要素の関係を示す．また，その内容について以下に説明する．

1) AP
AP とは業務に応じた処理を行う構成要素である．代表的な業務として，飛行機の座席予約や銀行の口座操作などがある．トランザクションの開始/終了などの管理は TM に，トランザクションとして必要なリソースの操作は RM に，他のシステムとの通信は cRM にそれぞれ処理を要求し，応答を受けることによって一つのトランザクションの処理結果を得る．

2) TM
TM はトランザクションの調整と制御を行う構成要素であり，そのために必要な管理機能を持っている．たとえば，トランザクションが複数のリソースを使用する場合には，

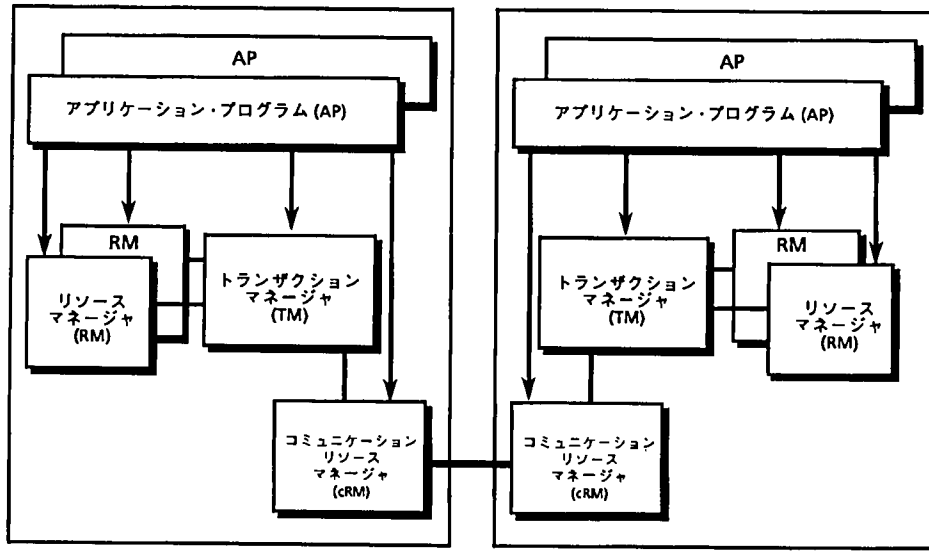


図 2　X/Open DTP モデルの構成要素

複数の RM 間にわたって整合性をとるための調整を行う.

3)　RM

RM とは，トランザクション処理に必要なデータベース，ファイルなどのリソースを管理する構成要素であり，AP に対してリソースへのアクセス手段を提供する．AP がトランザクションとして複数の RM を用いる場合，各 RM は他の RM とは独立しているため，トランザクション全体の ACID 特性は TM によって制御される．RM は TM の指示に従って，自身が管理するリソースの ACID 特性を保証しなければならない．また RM は，複数の AP で共有することができる.

4)　cRM

cRM とは通信リソースを管理する RM であり，AP が他のトランザクション処理システムに存在する AP と連携（通信）して分散トランザクションを処理する場合に必要な構成要素である．cRM は，バッファやコネクションなどの通信リソース管理機能および通信プロトコルの制御機能を持っている．cRM 間の通信プロトコルには，ISO で規格化された OSI-TP を使用できる.

1.6　分散データベース・システムとの違い

TM および RM を用いた DTP モデルは，分散データベース・モデルと機能的に似ているためによく比較される.

クライアント/サーバ・システムではよく参照される「ガートナー・グループの分散コンピューティングにおけるアプリケーション・モデル」では，DTP モデルはクライアントおよびサーバの双方にアプリケーション機能を包含する「機能分散型」に位置付けられるが，分散データベース・モデルはクライアントにアプリケーション機能を持ちサーバにデータ管理機能を持つ「リモートデータ管理型」に位置付けられ，データベースを分散して配置し，クライアントからはあたかも一つのデータベースとしてアクセスできるしくみを提供する.

クライアントからのデータベースへのアクセス要求は SQL 文によって行われ，分散されたデータベース間は，個々のデータベース独自のプロトコルと方式によって別のデータベースサーバに転送されて，データをアクセスし，結果がクライアントへ返される.

一方 DTP モデルでは，クライアントからのサーバへの要求はトランザクション送信という形をとっていることが異なっている.

分散データベース・システムと比べると，DTP モデルは次のような特徴をもつと考えられる.

1)　クライアント/サーバ間の通信量

DTPモデルは基本的にクライアントからトランザクションによってサーバに処理要求する方式をとり，サーバ側のTMがデータベースに必要なSQL文を出す．このために，サーバ内でのデータベースアクセス回数が多くても，クライアント/サーバ間のトラフィックは一回ですむために，通信量のオーバヘッドが一般的な分散データベースよりは少ないといえる．

2) 大規模システムのサポート

DTPモデルでは，一つのサーバ・プロセスは複数のクライアントからのトランザクションを処理でき，トランザクション量の大きさに応じてサーバ・プロセス数を調整できる．このために大規模なトランザクション処理システムに対応しやすいといえる．

3) トランザクションの制御

DTPモデルでは，クライアントからのトランザクション要求をいったんTMが受けてからサーバに渡す．このためにTMによって個々のトランザクションごとの処理優先度づけを持たしたきめの細かな制御が可能である．

4) 業務アプリケーション・プログラムの組み込み

DTPモデルでは，個々のトランザクション処理システムに要求される業務処理を，アプリケーション・プログラムによって組み込むことが容易であり，柔軟なシステムを構築できる．

ただし，どのようなOLTPシステムを構築する場合においても，基盤となるシステム開発技術が重要であり，特にOLTPシステム設計には技術力のあるコンサルテーションが必要であるといえる．

2. TransIT Open/OLTPの概要

2.1 TransIT Open/OLTPとは

TransIT Open/OLTPは，業界標準のクライアント/サーバ・アーキテクチャに基づき，部門規模から全社規模の基幹業務処理までをクライアント/サーバ・システムで構築するための基盤となるユニシスのミドルウェアである．

TransIT Open/OLTPは，X/Open DTPモデルに準拠したOLTPの一つであるTUXEDOをベースとした製品であり，Windows搭載のPCからPCサーバ，UNIXサーバ，そしてエンタプライズ・サーバまでをサポートしている．

2.2 TransIT Open/OLTPの商品

TransIT Open/OLTPは，サーバ用ソフトウェアとクライアント用ソフトウェアに分類され，おのおの以下の商品を用意している．

1) サーバ用ソフトウェア

① TransIT Open/OLTP for Windows NT

マイクロソフト社のOSであるWindows NT上で稼働するサーバ・ソフトウェアである．

TransIT Open/OLTP for Windows搭載のWindows PC，U 6000シリーズ，Aシリーズ，2200シリーズ上のTransIT Open/OLTP，および他社UNIX機上のTUXEDOとの間でクライアント/サーバ型分散トランザクション処理を実現する．

② TransIT Open/OLTP for UNIX

U 6000シリーズ上で稼働するソフトウェアで，利用形態に応じて次のパッケージ商品がある．

(a) TransIT Open/OLTP TM

ノベル社のTPモニタであるTUXEDOに相当する商品である．TransIT Open/OLTP for Windows搭載のWindows PC，U 6000シリーズ，Aシリーズや2200シリーズ上のTransIT Open/OLTP，および他社UNIX上のTUXEDOとの間で，クライアント/サーバ型分散トランザクション処理を実現する．

(b) TransIT Open/OLTP OSI-TPキット

OSI-TPプロトコルで他のTransIT Open/OLTPと接続し，トランザクション処理を行う場合に必要なオプションである．2200シリーズやAシリーズと接続する場合に必要である．

③ TransIT Open/OLTP for Aシリーズ

Aシリーズ上で稼働するサーバ・ソフトウェアで，利用形態に応じて以下のパ

ッケージを用意している．

(a) TransIT Open/OLTP (C/S)
TransIT Open/OLTP for Windows搭載のWindows PCとの間でクライアント/サーバ処理を実現する．また，T27K端末を使用した既存アプリケーションとの共存および移行も可能である．

(b) TransIT Open/OLTP (2 PC)
単一Aシリーズ内でDMS IIの複数のデータベースを構築し，それらの間での統合データベース・リカバリを実現するためのオプション・ソフトウェアである（2相コミット）．

(c) TransIT Open/OLTP (DTP)
Aシリーズ同士，2200シリーズおよびU6000シリーズとの間で，連携する分散トランザクション処理システムを構築する場合に適用する．ただし，2相コミット処理を実現する場合には，別途TransIT Open/OLTP (2 PC) が必要である（平成8年度出荷予定）．

④ TransIT Open/OLTP for 2200シリーズ
2200シリーズ上で稼働するサーバ・ソフトウェアで，利用形態に応じて次のパッケージ商品を用意している．

(a) TransIT Open/OLTP (C/S)
2200シリーズとTransIT Open/OLTP for Windows搭載のWindows PCとの間で，クライアント/サーバ処理を実現する．

(b) TransIT Open/OLTP (DTP)
2200シリーズ同士，Aシリーズおよび U6000シリーズとの間で，連携する分散トランザクション処理システムを構築する場合に適用する．ただし，2相コミット処理を実現する場合には，別途TransIT Open/OLTP (2 PC) が必要である．

(c) TransIT Open/OLTP (2 PC)
TransIT Open/OLTP (DTP) と組み合わせて，2200シリーズ同士，およびU6000シリーズとの間で2相コミット処理を用いた統合データベース・リカバリを実現する．

2) クライアント用ソフトウェア
TransIT Open/OLTP for Windows
Windows上で稼働するクライアント・ソフトウェアである．
各サーバ・ソフトウェアとの間でクライアント/サーバ型のトランザクション処理を実現するための各種ライブラリで構成されている．
また，クライアント・アプリケーション開発環境として，次のソフトウェアとの連携用ライブラリも提供している．
・Visual Basic
・Visual C/C++
・PowerBuilder

2.3 TransIT Open/OLTPの主要機能

TransIT Open/OLTPの主な機能は，次のとおりである．

1) トランザクション管理機能
TransIT Open/OLTPは，X/Open DTPモデルに基づいた分散トランザクション処理を実現するために，次のトランザクション管理機能を持っている．

① 経路選択
クライアント・プログラムは，サーバ・プログラムの位置を意識することなく処理を要求することができる．

② データ依存型経路選択
データの値によって処理するサーバ・アプリケーションを選択することができる．

③ 分散ノード処理
複数のシステムにわたる整合性のとれたトランザクション処理が可能である．

④ 優先処理管理
クライアントからの処理要求の優先度を設定することで，トランザクション処理に求められる重要なレスポンスを保障するための優先処理制御機能を持っている．
なお，上記機能中②，④については，Aシリーズおよび2200シリーズを除く．

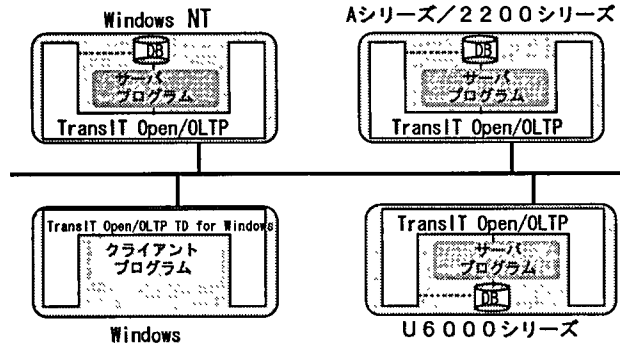


図 3 TransIT Open/OLTP 構成図

2) アプリケーション開発環境

次のクライアント・アプリケーションの開発支援ソフトウェアをサポートしている.

- Visual Basic
- Visual C/C++
- PowerBuilder

さらに, OLE(Object Linking-and Embedding) のサポートにより, 各種ビジネス・ソフトウェアとの連携を実現している. そのため, ユーザは TransIT Open/OLTP を利用して, PC 上から企業内の分散データをリアルタイムにアクセスして PC 上に引き出し, 各種ビジネス・ソフトウェアを利用して自由に加工したり, その結果を OLTP システムに渡すことができる.

2.4 TransIT Open/OLTP のシステム構成

TransIT Open/OLTP は前述のとおり, クライアント・ソフトウェアとサーバ・ソフトウェアから構成されている. したがって, システム構成としてはクライアント・ソフトウェアとサーバ・ソフトウェアによるクライアント/サーバ接続と, サーバ・ソフトウェア間でのクライアント/サーバ接続 (サーバ/サーバ接続) の 2 つがある. 前者は 2 階層のシステム構成で, 後者は 3 階層のシステム構成で必要となる接続形態であり, 日本ユニシスの提供する各種プラットフォーム, および他社機上の TUXEDO を自由に組み合わせて, 拡張性の高い分散トランザクション処理システムを構成することができる (図 3).

掲載論文梗概

A社がその技術検証の一環としてTIPPLER for windows* を使用しマルチメディア指向の営業窓口支援システムの開発を行った．佐々木勝信は**マルチメディアとTIPPLER**の中で，A社営業窓口業務のマルチメディア化を通して，TIPPLERを用いたマルチメディアを扱ったシステムの構築方法と，その可能性について述べている．

*windowsは，米国Microsoft社の商標である．

製造業において，生産現場の作業計画を立案するスケジューリングシステムは，今最も各企業が重視しているものの一つである．しかし，多くの時間と費用がかかる割にはあまりうまくいっていないのも実情である．この様なスケジューリングシステムを業務モデルサブシステムとしてTIPPLERにより開発した．開発に当たっては，システムへの要件に対して，要件の整理，スケジューリング業務手順のモデル化，部品化構造，GUIの駆使等で対応した．田頭浩は**TIPPLERの製造業におけるスケジューリングシステムへの適用事例**の中で，それらの対応について，TIPPLERでどのように実現していったかを開発の事例として報告している．

PCの高機能化と低価格化は，企業内におけるオフィスの環境を劇的に変え，また，PCのある環境では，PCなしでは仕事ができない程その有用性を高めている．そして今やこれらのマシンがネットワークにより接続され，会社の基幹システムになろうとしている．このような環境の変化に対応し，TIPPLERもその稼働プラットフォームをPC環境に拡大すべく開発，商品提供を行ってきた．多田哲は**クライアント/サーバによるシステム構築とTIPPLER**の中で，TIPPLER for windows* からTIPPLER/V（windowsNT* 版）の開発に至った背景を考察する事によりTIPPLER/Vの位置付けとPCによるクライアント/サーバ環境の構築について述べている．

*windows，windowsNTは，米国Microsoft社の商標である．

掲載論文梗概

ISSN 0914-9996

技　報
UNISYS TECHNOLOGY REVIEW
Vol. 15　No. 3　(No. 47)

発　行　日　平成7年11月30日
編集発行人　吉　村　賢　一
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

UNISYS

日本ユニシス株式会社
本社 東京都江東区豊洲 1-1-1 〒135 電話03-5546-4111(大代表)

ビジネスを変える
超並列技術と世界標準。

OPUS&Oracle

ユニシスから、世界標準の「Oracle7」を搭載した、最新の超並列オープン・サーバ「OPUS」新登場。

■意思決定を支える新しいプラットフォームの誕生です。

情報システムは、いま、メインフレーム上の膨大なデータを、新しい形で有効活用する分散／並列型のコンピューティングへと向かっています。なかでも期待を集めているのが、大量のデータをタイムリーに分析・活用し、マーケティング戦略や顧客サービスに役立てるDSS(意思決定支援システム)。そして、このDSS構築の鍵を握るのが、優れた並列サーバと柔軟なデータベースの組み合わせです。この課題に応えるため、オープン・システムで確かな実績をもつユニシスと、リレーショナル・データベース管理システムの世界シェアNo.1※1を誇るオラクルは、お互いの力を最大限に活かしあう、新しいDSS構築のためのプラットフォームを実現しました。

※1「UNIX Server Database Revenue & Share 1993-1994」出典：1995年4月 Dataquest

■ビジネスの規模に応じた最適なシステムを構築します。

OPUSは、最大128個のPentiumプロセッサを格子状に相互接続する独自のSPP(スケーラブル・パラレル・プロセッシング)技術によって、卓越した処理速度と拡張性を備えており、ビジネスの規模にあった効率的なシステム構築が可能です。また、標準UNIXに準拠した分散OS(SVR4／MK)を提供することで、データの分散を意識することなく、スムーズに操作・運用管理ができるシングル・システム・イメージを実現。そしてさらに、クライアントからの複雑な問い合わせを、複数のプロセッサで分割・処理するOracleの並列処理技術が、OPUSの処理能力を最大限に引き出し、高速かつ大規模なデータベースの構築を容易にしています。

〈with IT〉
日本ユニシスは、先進のIT(情報技術)で、
お客様に最適なソリューションを提供します。