07−R003

エンドユーザ向けアプリケーション統合環境の
研究開発報告書

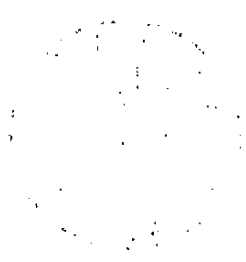
平成8年3月

JIPDEC

財団法人　日本情報処理開発協会

この報告書は、日本自転車振興会から競輪収益の一部である機械工業振興資金の補助を受けて平成７年度に実施した「エンドユーザ向けアプリケーション統合環境の研究開発」の成果の一部をとりまとめたものであります。

KEIRIN この報告書は、競輪の補助金を受けて作成したものです。



はじめに

パーソナルコンピュータ（PC）の小形化、低廉化により、個人ベースでのコンピュータ利用が拡大してきています。また、インターネットに代表されるネットワーク技術の進展によって、多種多様な情報の発信、アクセスも個人ベースで行える環境が整備されつつあります。

こうしたエンドユーザをとりまくコンピュータ環境の変化の中で、企業などのエンドユーザは、PCやネットワークを駆使した新しい業務改善や仕事の質的向上が求められています。

エンドユーザにより近い部分での情報化にあっては、コンピュータを用いて行いたい仕事や利用形態が多種多様であることから、従来のエンドユーザコンピューティングで用いられていた定型的なアプリケーションや汎用パッケージソフトだけではもはや不十分であり、エンドユーザが自ら、あるいは情報部門の支援を受けながら、自分の仕事や業務に最適なシステムをPC上に構築し活用する、より進化したエンドユーザコンピューティングが必要になります。

こうした背景のもとに、当協会では、平成７年度より２年計画で、「エンドユーザ向けアプリケーション統合環境の研究開発」事業を実施することにいたしました。

実施に当たりましては、大学、企業などの研究者、専門家からなる「エンドユーザ向けアプリケーション統合環境に関する調査研究ワーキンググループ（主査　中所武司　明治大学理工学部情報科学科教授）を設け、要素技術、構築技術などについて調査検討を行うほか、平成７年６月には国際会議に海外調査員を派遣し、最新技術情報の収集を図りました。

本報告書は、平成７年度における研究成果の一部を中間報告として取りまとめたものです。第１章は新しいエンドユーザコンピューティング像や技術動向、第２章はコンポーネントウェアを中心とした新しいアプリケーション構築環境となっております。

最後に、本研究開発に当たって、ご指導ご協力を頂いた、中所主査を始め各委員の方々及び関係各位に深甚なる謝意を表する次第です。

平成８年３月

財団法人日本情報処理開発協会

エンドユーザ向けアプリケーション統合環境に関する調査研究

ワーキンググループメンバー名簿　（敬称略）

主　査　中所　武司　明治大学理工学部情報科学科教授

委　員　青山　幹雄　新潟工科大学情報電子工学科教授

委　員　佐治　信之　日本電気（株）クライアントサーバソフト技術研究所開発環境技術部技術課長

委　員　澤谷由里子　日本アイ・ビー・エム（株）APソフトウェアテクニカルマーケティング

委　員　萩原　正義　マイクロソフト（株）ビジネスシステム事業部市場開発部担当課長

委　員　原　　祐貴　（株）富士通研究所ソフトウェア研究部

委　員　増石　哲也　（株）日立製作所情報・通信開発本部ミドルウェア開発センタ主任研究員

委　員　山崎　訓由　新日鉄情報通信システム（株）技術開発部技術企画室長

委　員　山本修一郎　日本電信電話（株）NTTソフトウェア研究所ソフトウェア技術研究部主幹研究員

委　員　向山　　博　（財）日本情報処理開発協会開発研究室主任研究員

執筆協力者　宇野 浩司　日本サン・マイクロシステムズ（株）システム応用技術本部技術サポート部

目　次

1. 技 術 動 向

1．技術動向

1．1　エンドユーザコンピューティングの動向

マルチメディアやインターネットというキーワードに代表される情報化社会の進展にともない、新しい時代に対応した新しいアプリケーションの作り方が模索されており、エンドユーザ主導のアプリケーション開発の時代が迫っているという印象が強い。本節では、幾つかの視点からエンドユーザコンピューティングの位置付けを試みる。

1．1．1　背景

現在、広く利用されているコンピュータのアーキテクチャの基本となっているプログラム内蔵方式が確立して、ほぼ５０年になろうとしているが、この間、アプリケーションソフトウェアの開発は情報処理の専門家によって行われ、利用者はそれをエンドユーザとして使用することに専念してきた。

このような時代において、ソフトウェア開発環境は、８０年代半ばまではプログラマなどの情報処理技術者の作業効率の向上が主目的であった。その後のCASE（Computer-Aided Software Engineering）ツールにおいても、大規模高信頼ソフトウェアの開発に携わる種々の立場の情報処理の専門家をソフトウェア工学的な観点で支援するのが主目的であった。

しかし、高度情報化社会を迎えた現在では、情報システムは、すでに業務の効率化といった従来の目的を越えて、広く社会生活の中に入り込んできている。そして、そのような情報システムの利用に関して、従来の情報システム部門に対比したエンドユーザ部門という範囲を越えて、あらゆる人達がエンドユーザになりつつある。このような状況は以下のような理由によるところが大きく、今後は、エンドユーザコンピューティングを支援する環境が重要になろう。

(1)情報のパーソナル化

ハードウェアの低廉化と共に、パソコン、ワークステーションなどの普及がめざましく、マルチメディアデータのコンピュータによる処理が容易化するにつれて、OA分野を中心にエンドユーザが急増している。「物のパーソナル化」が「情報のパーソナル化」を促進している。

(2)情報のグローバル化

コンピュータの利用形態が大型機によるホスト集中型からパソコン、ワークステーション主体のネットワーク型へと移行すると共に、分散コンピーティングやインターネットの普及がめざましく、エンドユーザがコンピュータを直接操作して、離れた場所に点在するデータにアクセスする機会が急増している。「システムのグローバル化」が「情報のグローバル化」を促進している。

(3)情報の資源化

コンピュータの利用目的が、従来の業務の合理化から業務革新や経営戦略の実現へと広がるにつれて、さまざまな業務担当者がエンドユーザとしてコンピュータを利用しはじめており、「情報の資源化」を促進している。

オフィスにおけるエンドユーザの業務を大ざっぱに分類すると、従来は定形業務と非定形業務に分けられた。定形業務はビジネス分野の帳票処理に代表されるような企業の基幹業務にかかわるものである。そのためのアプリケーションプログラムは、通常は、エンドユーザの要求をききながら、企業内の情報システム部門が開発あるいは導入して、エンドユーザに提供してきた。保守、拡張も情報システム部門が行い、エンドユーザは定められたユーザインタフェースに従ってコンピュータを使用するだけであった。一方、非定形業務については、表計算に代表されるような市販のOAソフトウェアを購入して利用してきた。

しかしながら、コンピュータの利用目的が、従来の業務の合理化から業務革新や経営戦略の具現化へと広がるにつれて、基幹業務と個人の非定形業務の連携が必要になってきている。この傾向は、コンピュータシステムが分散コンピューティングの方向に進展するにつれていっそう加速されている。その結果、例えば従来の表計算ソフトウェアと基幹業務データベースをつないだり、データベース検索ソフトウェアにデータの加工、編集機能を追加したりして、非定形業務の支援システムを作り上げることが重要になってきている。さらに、CSCW（Computer-Supported Cooperative Work）のように、別々の場所にいるグループが協力しあって業務を遂行していくためのグループウェアが求められている。

このような状況下でタイムリーに必要なソフトウェアを手にいれていくためには、もはや情報システム部門が提供するシステムや市販のソフトウェアパッケージを単に利用するだけでなく、業務担当者は、積極的に自ら使うソフトウェアの開発に関与していく必要がある。このようなエンドユーザコンピューティングのための開発環境としては、情報処理の専門家を対象にしたような従来の手続き型パラダイムに基づくプログラミングの支援ではなく、業務の専門家が自らの業務の知識を基本にしたプログラミングを可能とする新しいパラダイムに基づいた支援機能が必要がある。

1．1．2　多視点からの考察

ここで、エンドユーザコンピューティングの位置付けを明確にするために、次のような５つの視点からエンドユーザコンピューティングについて考察する。

・エンドユーザコンピューティングの定義

・ソフトウェア産業論

・ソフトウェア生産技術

・プログラミング言語

・開発環境

(1) エンドユーザコンピューティングの定義

はじめに、ここで議論するエンドユーザコンピューティングは、「誰が、何のために、どのように利用する技術」であるかを明確にするために、エンドユーザ、対象ソフトウェアおよび開発・保守形態に関して一応の定義をしておく。

(a) エンドユーザ

一般に、エンドユーザとして少なくとも以下の3種類が考えられる。

(i) 基幹業務担当者

例えば、銀行のようなユーザ企業において、システム部門に対するエンドユーザ部門に所属する人達で、利用するソフトウェアはシステム部門が開発し、提供してくれる。

(ii) 業務の専門家

一般にオフィスワーカーといわれるような人達で、DB検索や表計算などに市販のアプリケーションパッケージを利用する。

(iii) 一般ユーザ

例えば、日常生活の中で銀行のATMを利用するような一般の人達で、将来、マルチメディア時代の主要ユーザとなる。

ここでは、(i)と(ii)のエンドユーザに対象を絞るが、特に非定型業務のコンピュータ化という視点では今後急速に増大すると思われる(ii)の方により重点を置く。

(b) 対象ソフトウェア

ここでは、上記の(ii)のエンドユーザが利用するような、オフィス業務用アプリケーションが中心となる。従って、分散協調型ソフトウェアが主な対象になり、ワークフローシステム、グループウェア、エージェントシステムなども含まれる。規模的には、中、小規模のアプリケーションソフトウェアを想定することになるが、ネットワーク接続するによりシステムとしては大規模化することもある。

(c) 開発・保守形態

エンドユーザ主体の開発を念頭に置くならば、問題領域の分析によるドメインモデルの構築が重要である。即ち、問題領域のモデルを作成し、そのモデル上でのシミュレーションによりモデルの妥当性を検証した後、実用に際しては、そのモデルをインタプリタにより実行するか、あるいは必要に応じてプログラムを自動生成するという方法である。比較的小規模のものについては、ドメインモデルの構築とプログラミングが一体化して、核の部分からだんだん機能を拡張していくようなプロトタイピング方式で開発する

ことになろう。

このように最終的にはエンドユーザが自らの業務のアプリケーションソフトウェアを自ら開発し、自ら利用することを理想とするが、その実現に向けての技術課題は多い。そこで、マイルストーンとして、エンドユーザが主体でシステムエンジニアの助けを借りて開発するが、保守はエンドユーザだけで行うレベルが当面の目標となるであろう。

(2) ソフトウェア産業論

次に、エンドユーザコンピューティングをソフトウェア産業の発展過程という視点でとらえるならば、表1.1-1にも示すように、労働集約型産業、知識集約型産業に続くものとして位置付けることができる。

(a) 労働集約型産業

これまでのソフトウェア産業は労働集約型産業であり、生産コストあたりの生産量という生産性の効率向上が重要であった。ステップ数／人月あるいはステップ単価を改善するために、CASEツール等を用いて自動化率を向上させ、人海戦術からの脱皮の努力をしてきた。しかし、この生産性の尺度の致命的欠陥は、ソフトウェアの価値（質）を規模（量）ではかることである。そのため、ソフトウェアの受託開発において、生産コストを発注者がすべて支払う場合は、ステップ数が多いほど価格（価値）が上がることになってしまい、受注者側の技術力向上の努力がおろそかになってしまう。このような矛盾は、バブル経済崩壊とともにソフトウェア産業が不況業種の代表になってしまったことによくあらわれている。

(b) 知識集約型産業

ソフトウェアの生産性は本来以下のように定義されるべきである。

ソフトウェアの生産性＝生産物の価値／生産コスト

この「生産物の価値」はユーザの視点で決まるべきものであり、高品質のソフトウェアやベストセラーのアプリケーション・パッケージの価値は高い。このようなソフトウェアの開発には、業務の知識と情報処理技術の双方が必要である。

表1.1-1　ソフトウェア産業論

ソフトウェア産業形態	主要な技術職	主要技術
労働集約型産業	プログラマ	自動化（CASE）
知識集約型産業	設計者	標準化（パッケージ）
ポスト知識集約型産業 （知恵集約型産業）	業務専門家	エンドユーザ コンピューティング

今後、ソフトウェアのビジネスは、アプリケーション・パッケージやその基となるコンポーネントウェアの分野とシステム・インテグレーションの分野に2分化して発展していくと思われるが、この場合、種々の分野対応の業務の知識と情報処理の知識をノウハウとして蓄積することが必要である。

(c) ポスト知識集約型産業

情報処理システムが、業務の効率化ではなく、業務革新や経営戦略の具現化に用いられるようになると、ソフトウェアの開発においても効率よりも効果が重要視される。何を作るかが最も大事であり、それを決めるのは業務専門家である。業務専門家が知恵をしぼって効果的なアプリケーションをタイムリーに作っていくためには、エンドユーザ自身が開発でき、かつ保守拡張ができる必要がある。対象は異なるが、例えば、最近注目されているRAD（Rapid Application Development）においてもエンドユーザが全工程に参加することが重要な成功要因になっていることと同じである。

このようなエンドユーザコンピューティングを実現するためには、そのためのツールや環境を提供しなければならない。自動化ツールや標準パッケージなどの情報処理技術を統合した上に、きめ細かく分類された応用分野対応（domain-specific, application-oriented）の業務の知識に基づいたアプリケーション・フレームワークの構築が必須である。潜在ユーザとしての業務の専門家の数を考えると、市場規模は現在の情報サービス産業のそれ（数兆円規模）を大きく上回ることになる。ポスト知識集約型産業は、知恵集約型産業ともいえる。

(3) ソフトウェア生産技術

第3の視点として、エンドユーザコンピューティングをソフトウェア生産技術の観点から考察する。情報化社会に対応して、一般にソフトウェアの規模と量と質に関するソフトウェア生産技術への要求は急速に増大しているが、ソフトウェアの問題は多種多様であり、何を開発するか、誰が使うか、誰が開発するか、等々の条件によって問題が異なることが多い。

一般的にメーカの視点（作る立場）から、ソフトウェア生産性に関する解決手段をあげると、以下の5項目が考えられる。

・開発対象（ソフトウェア）の標準化
・開発工程（プロセス）の定式化、自動化
・開発手段（ツール）の高機能化
・開発者（プログラマ）の技術力向上
・業務専門家（ユーザ）による開発可能化

これらは、新規開発量の抑制、工数削減、作業効率の向上、労働の質の向上などの形で直接、間接に「規模」、「量」、「質」の問題の解決に寄与する。従来は最初の4項目に努力が払われてきたが、これから

は５番目のエンドユーザコンピューティングも視野に入れる必要がある。

次に、これらの解決手段を、ユーザのソフトウェア入手方法というユーザの視点（使う立場）で見直すと、表1.1-2に示すような「標準化」、「自動化」、「情報処理技術者の自由業化」、「エンドユーザコンピューティング」の４種類のシナリオ案を描くことができる。「標準化」と「自動化」に関しては従来から相当の努力が行われてきた。「情報処理技術者の自由業化」は、日本の社会構造の変化を伴うため少し先の話になる。当面は「エンドユーザコンピューティング」が大きな鍵になると思われる。以下にこれらの解決方法のシナリオについて簡単に述べておく。

(a) 標準化シナリオ

標準化の基本は共通化による共有化であり、古くて新しい問題である。これまでソフトウェアの標準化には次のような幾つかの段階があった。

・同一組織内での部品化、再利用

・同一組織内、複数機種間での共通化

・複数組織間、複数機種間での共通化

・コンポーネントウェアおよびアプリケーションパッケージの業界標準化

第１段階は、従来からプログラムの部品化、再利用技術として研究がなされてきたものである。第２段階は、アプリケーション・ソフトウェアの異機種間の移行性を高めるために、アプリケーション・ソフトウェアの標準的なアーキテクチャを設定し、プログラミング言語、ユーザインタフェース、データベース管理、通信管理などの外部仕様を統一するもので、８０年代後半にメーカ側から提案されたものである。現在は、第３段階であり、次のような特徴に代表されるオープンシステムの時代である。

・異機種／異種プラットフォーム間でのアプリケーションの移行性

・異機種／異種プラットフォーム間の接続性／相互運用性

表1.1-2　ユーザ視点でのソフトウェア入手の問題の解決

解決手段	ユーザのソフトウェア入手方法
標準化	適正価格の市販パッケージ購入
自動化	要求使用を提示して自動生成
情報処理技術者の自由業化	優秀な技術者に高額で依頼
エンドユーザコンピューティング	自分で作成

９０年代のシステム構成は、大まかには、図1.1-1に示すように、応用ソフトウェア、ミドルウェア、基本ソフトウェア、ハードウェアの４階層で表現される。ミドルウェアや基本ソフトウェアとアプリケーションとの間のAPI（Application Programming Interface）をできるだけ業界標準や国際標準にして、移行性や相互接続性を可能とする。特にミドルウェアの役割は大きい。

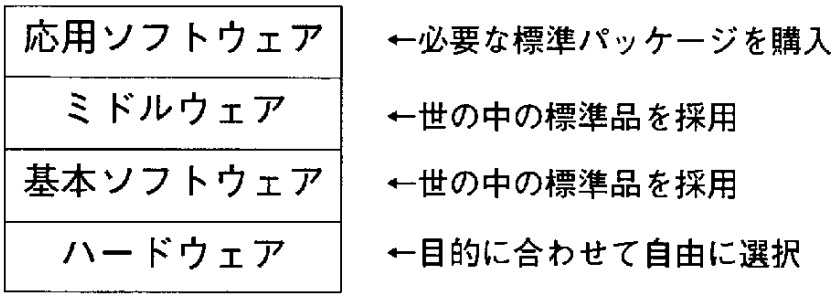


図1.1-1　オープンシステムのシステム構成

最後の段階は、特定用途のアプリケーション・パッケージの中からベストセラーになったものが業界標準になっていく。OAソフト、RDBソフトなどでその傾向が強い。銀行システムではメガステップオーダのパッケージもある。ただし、変化の激しい分野で、１つのアプリケーションパッケージがいつまでも業界標準ではありえず、また変化に対応した機能拡張によるマンモス化は、エンドユーザにとって好ましいことではない。現実には、むしろ、コンポーネントウェアの流通が促進され、コンポーネントの差し替えによってエンドユーザに必要な機能のみを取り込んでいくスタイルになるものと思われる。

このように標準化が進めば、ユーザは、自分の業務に必要な市販のアプリケーションパッケージやコンポーネントウェアを適正な価格で買ってくることができる。

(b) 自動化シナリオ

「自動化」、即ち「自動プログラミング」は、プログラム内蔵方式のコンピュータの発明以来のソフトウェア技術者の夢である。かっては、アセンブラやコンパイラが自動プログラミング技術と呼ばれた時代もあった。プログラムの記述形式は、機械語からアセンブラ、高級言語へと発展し、機械語への展開率の向上という意味での「自動化」を実現したが、プログラムを手続き的に記述するというスタイルは本質的に変わっていない。現在は、仕様記述言語と仕様からのプログラム自動生成技術の研究が活発に行なわれている。

最終的には、ユーザは、自分の業務の用語で要求定義を記述するだけで、プログラムを自動生成することができる。

(c) 情報処理技術者の自由業化シナリオ

現在、情報処理技術者の社会的待遇が他に比べて明らかに優位にあるということはないが、今後、急激な変化がありえる。勿論、この業務は個々の技術者の能力差が非常に大きいので、建築士などの専門家と同じように個人の能力に応じた収入を得られるように自由業化するべき分野である。銀行オンラインシステムや電話交換システムなど、社会的責任の大きいシステムの数が増加するので、優秀な情報処理技術者の高収入は保証されるであろう。

その結果として、ユーザは、お金さえ払えば、優秀な技術者が作る良質のソフトウェアをタイムリーに入手できる。

(d) エンドユーザコンピューティングシナリオ

ユーザ業務を大まかに定形業務と非定形業務に分けて考えると、定形業務については、「標準化」シナリオあるいは「自動プログラミング」シナリオによって解決する。一方、非定形業務については、自分の業務に固有の部分のソフトウェアを作成する必要がある。今後急激な増大が予想されるこのようなソフトウェアはユーザ自身が自分で開発できることが理想である。

エンドユーザコンピューティングが発展すれば、ユーザは、"プログラミング"を意識することなく、自分の業務をメタファベースで、あるいは業務の言葉を用いて簡単にコンピュータ化できる。

(4) プログラミング言語

第4の視点として、エンドユーザコンピューティングをプログラミング言語の観点から考察する。プログラミング言語は、当初からソフトウェア生産性向上に大きな役割を果たしてきた技術である。プログラミング言語の研究開発に関するこれまでのマクロなトレンドを表1.1-3のように要約することができる。

表1.1-3　プログラミングパラダイムの変遷

時期	目的	内容
１９６０年代	量的高級化	記述水準の向上
１９７０年代	質的高級化	プログラミング方法論
１９８０年代	脱手続型パラダイム	宣言的記述
１９９０年代	エンドユーザ コンピューティング	脱プログラミング

(a) 高級言語化

１９６０年代は、高価なコンピュータを効率良く利用することが重要であった。このようなコンピュータ利用の初期の段階に、プログラムの実行効率をあまり低下させないで生産効率をあげるために、アセンブリ言語からFORTRAN、COBOL、PL/Iなどの高級言語への移行が行われた。これは、機械語への展開率の向上という意味で「量的高級化」といえる。

(b) 構造化

次に１９７０年代は、ソフトウェアの大規模化という第１次のソフトウェア危機への対応が重要であった。そのため、高信頼性と保守性を重視したプログラミングスタイルの研究が行なわれ、構造化技法などによりプログラムの理解容易性を実現する「質的高級化」が追求された。

(c) 脱手続型パラダイム

そこで１９８０年代には、ソフトウェアの生産性、信頼性、保守性に関する諸問題は、本質的に手続き型のプログラミングパラダイムに起因するという考えから、論理型、関数型などの非手続き型のプログラミングパラダイムとそれをベースにした言語の研究が盛んに行なわれるようになった。

(d) エンドユーザコンピューティング

最近では、情報処理の専門家でなくてもソフトウェアを開発できるように、非手続き型プログラミングの概念をより進めて「脱プログラミング」を実現するエンドユーザコンピューティングの研究が盛んである。エンドユーザがプログラミング言語を用いないで業務のコンピュータ化を行なえるための現実的な技術としては、業務向け簡易言語、日本語プログラミング、ビジュアルプログラミング、プロダクションシステム等がある。いずれも脱プログラミングのコンセプトに基づいているが、現状ではまだ「脱プログラミング言語」のレベルであり、プログラミングの概念やセンスが不要というわけではない。

(5) 開発環境

最後に、エンドユーザコンピューティングを開発環境の視点から考察する。開発環境とは、狭い意味では、ソフトウェアの開発のためにコンピュータ上で利用するツールの集合であるが、実際にはそれらのツールが提供する開発技法に加えて、それらのツールの操作対象となるデータの管理、それらのツールを有効活用するための開発方法論やプロセス管理、さらには開発担当者まで含めたプロジェクト管理も関連している。むしろ開発方法論の実現手段として、あるいはプロセス管理やプロジェクト管理の一環として開発環境ないしツールがあると考えるべきである。したがって、開発環境の目的は、一般には「良いものを早く安く作ること」すなわち高品質ソフトウェアの高効率生産を促進することである。

開発環境をツール史の視点からみると次のようにまとめられる。その概要を表1.1-4に示す。

表1.1-4　ソフトウェア開発環境の変遷

年　代	特　　徴
５０年代 ｜ ６０年代	●バッチ型プログラミングツール ・高級言語コンパイラ、ソースファイル操作ユーティリティ
７０年代 前半	●対話型プログラミングツール *～バッチ型処理から対話型（ＴＳＳ）処理へパラダイムシフト～* ・上流工程：ドキュメント作成支援 ・下流工程：ツールボックス（エディタ，デバッガなど）
７０年代 後半 ｜ ８０年代 前半	●統合プログラミング環境 *～開発手段（ツール）の高機能化から* *開発工程（プロセス）の定式化へパラダイムシフト～* ・ユーザインタフェース、プログラミング用データベース共通化 ・特定プログラミング言語向き環境（構造エディタなど）
８０年代 後半	●統合開発環境 *～製造工程(how-to-make)中心から* *計画・分析・設計工程(what-to-make)中心へパラダイムシフト～* ・ＣＡＳＥツール ・ユーザインタフェース統合、リポジトリによるデータ統合
９０年代 前半	●オープンシステム *～開発工程（プロセス）の定式化から* *開発対象（プロダクト）の標準化へパラダイムシフト～* ・開発環境のプラットフォーム、ミドルウェアの共通化 ・分散環境化
９０年代 後半	●エンドユーザコンピューティング *～生産者中心の視点から利用者中心の視点へパラダイムシフト～* ・コンポーネントウェア、ビジュアルプログラミング

(a) ５０－６０年代

ソフトウェアツールの歴史は、１９４９年のプログラム内蔵方式のコンピュータの実用化に始まる。プログラムの記述形式は、機械語からアセンブラ、さらに高級言語へと発展した。この時期には、コンパイラやプログラムのソースファイル操作用のユーティリティなどのツールがバッチ処理形態で使われていた。

(b) ７０年代前半

ソフトウェア工学の研究初期の１９７０年代は、上流工程に注目した要求定義技法や設計技法の研究が盛んに行なわれたが、コンピュータによる支援という観点では、ドキュメント作成支援程度の自動化しか

実現できなかった。そのため、下流工程で利用するプログラミングツールの個別開発が中心に行なわれた。UNIXに見られるような、いわゆるツールボックス型といわれるものである。この時期には、ツールの利用形態が、バッチ処理形態から対話処理（TSS処理）形態へと移っていった。

(c) ７０年代後半－８０年代前半

その後、プログラミングツールの統合化が進んだ。この時期に従来の開発手段（ツール）の高機能化から開発工程（プロセス）の定式化へのパラダイムシフトが生じ、環境という表現が一般化した。当時の統合プログラミング環境は、ユーザインタフェースとプログラミング用データベースの共通化によるツール統合が基本であった。ツール間の連携は、特定のプログラミング言語を対象にした言語指向プログラミング環境が先行する形で発展した。上流工程に関しては、構造化技法が定着し始めた時期であり、データフロー図やプログラム構造図などの作図やドキュメンテーションが支援されるようになった。同時に構造化図式を対象としたエディタも利用され、ビジュアル化が重視された。

(d) ８０年代後半

ソフトウェア開発の真の難しさは上流工程にあるという認識から、再び上流工程の方法論や技法のツール化が積極的に試みられ始めた。how-to-makeが主体の製造工程からwhat-to-makeを重視する計画、分析、設計工程へと関心が移っていった。このような上流工程支援を含む統合開発環境を統合CASEと呼ぶ。ユーザインタフェースに関しては、ワークステーションの進歩により使い勝手の良いGUI（Graphical User Interface）が一般化した。データの管理に関しては、リポジトリによるライフサイクル支援がある。

(e) ９０年代前半

最近では、情報処理技術者の不足（正確に言えば、技術不足）とアプリケーション・ソフトウェアの複雑化の挟み撃ちにあって、CASEへの期待が高まっている。オープンシステムの時代とともに、開発工程（プロセス）の定式化・自動化から開発対象（ソフトウェア）の標準化へのパラダイムシフトが生じている。プロセス・イノベーションよりもプロダクト・イノベーションが求められている。

(f) ９０年代後半

このようなトレンドの中で、ソフトウェアに関してもアプリケーションがより重要となり、生産者中心の視点から利用者中心の視点へパラダイムシフトが起きている。エンドユーザコンピューティングが注目され、新しいアプリケーションの作り方が模索されている。コンポーネントウェアやビジュアルプログラミングの技術が発展し、分野別にアプリケーション・ソフトウェアの標準アーキテクチャと部品ライブラリを用意し、業務モデルから部品合成によりアプリケーションを生成するようになる。

1．1．3　主要技術

エンドユーザコンピューティングのための技術として、5年前に業務向け簡易言語、日本語プログラミング、ビジュアルプログラミング、プロダクションシステムを候補[7]に上げたが、最近、発展がめざましいのはビジュアルプログラミングである。

(1) ビジュアルプログラミング

ビジュアルプログラミングの発展の経緯を以下のような流れでとらえることができる。

・オープンシステム時代のソフトウェア・アーキテクチャの要請

→・オブジェクト指向技術による階層化と部品化

→・コンポーネントウェアの出現

→・ビジュアルプログラミングの実用化

(a) オープンシステム時代のソフトウェア・アーキテクチャ

最近の情報システム構築に関する大きな変化であるオープンシステム化においては、異機種コンピュータ間でのアプリケーションソフトウェアの接続性や移行性の要求に応えるために、アプリケーションソフトウェアのアーキテクチャをできるだけ標準的に作る努力がなされており、図1.1-1にも示したように、以下のようなソフトウェア構成をとる。

・階層化：ハード、基本ソフト、ミドルウェア、応用ソフトウェアの階層構造化。

・部品化：各階層を部品の集合で構成。

・統一インタフェース化：階層間のインタフェースの統一。

(b) オブジェクト指向技術による階層化と部品化

このようなアーキテクチャを実現するための技術としてオブジェクト指向技術が注目され、既にプログラミングの分野では実用的効果をあげている。社会的要請から最近発展の著しいオープンシステムと四半世紀の歴史を有するオブジェクト指向技術との歴史的出会いとも言えるものである。

オブジェクト指向概念の基本的な要件として、以下の4項目をとりあげる。

・分散協調型計算モデル

・データ抽象化機能

・クラスからのインスタンス生成機能

・クラスの階層化と継承機能

これらは各々独立した概念であるが、ソフトウェア開発のどの局面に注目するかによって、適用すべき概念とその効果が決まる。上記の階層化、部品化、統一インタフェース化と関連では以下の効果が期待される。

(i) 開発者のための部品化、再利用

・データ抽象化により、独立性の強い、汎用的な機能部品を作りやすい。

・階層化と継承機能により、部品ライブラリを構築しやすい。

・インスタンス生成機能により、部品のカストマイズがしやすい。

・分散協調型モデルに基づくプログラム設計により、部品の抽出、利用が容易である。

(ii) 保守者のための拡張性と移行性

・階層化と継承機能により、機能追加が容易である。

・データ抽象化により、機能変更や他機種への移行が容易である。

(iii) 利用者のための操作性と統一性

・データ抽象化により、画面上のオブジェクトの機能の１つを選ぶという直接操作で簡単に対話できる。

・種々の応用ソフトウェアのユーザインタフェースは共通部品での構築により、統一できる。

(c) コンポーネントウェア

このように従来からオブジェクト指向プログラミングにおいて、オブジェクトの部品化・再利用の利点が期待されていたが、抽象データ型のレベルのクラスライブラリを提供しただけでは、なかなか部品の有効活用が難しい。部品としての粒度が小さすぎて、プログラミング技術レベルの知識が要求されるのが現状である。

そこで、アプリケーションと部品の粒度とのこのようなギャップを埋めるために、最近では以下のような３種類の粒度のコンポーネントが考えられている。

・アプリケーションフレームワーク

・デザインパターン

・クラスまたはオブジェクト

フレームワークは、アプリケーションのソフトウェアアーキテクチャ、クラスは、アプリケーション用部品、デザインパターンは、フレームワークに部品を埋め込むときの工法／定石という位置づけで、上記の記述順に粒度が大、中、小である。各々のレベルでドメイン固有のものと汎用性の高いものがある。

最近、特に複数のクラスを組み合わせてそれらの典型的な協調作業をパターン化した手法が注目されている。Eric Gammaらは、オブジェクト指向プログラミングの経験から２３の典型的なデザインパターンを抽出し、「オブジェクト指向ソフトウェア設計における、特定の問題に対する” simple and elegant” な解」として利用することを薦めている。これらのデザインパターンでは、「特定のコンテキストの中で一般性のある設計問題を解くためにカストマイズされた、オブジェクト／クラスの協調」をパターン化している。

コンポーネントウェア普及のためのインフラも実用レベルのものが出始めている。コンポーネントウェ

ア関連のミドルウェアとして、分散環境下でのオブジェクト管理を共通化して相互運用性を確保したり、アプリケーション間連携を容易にするためのプラットフォームが普及しはじめている。その例として、CORBA（OMG）、OLE（Microsoft）、OpenDoc（CI Labs）などがあげられる。

(d) ビジュアルプログラミングの変遷

このようなコンポーネントウェアが充実しても、プログラミングの本質的な難しさを克服できなければエンドユーザコンピューティングツールにはならない。これまで以下の３つの観点からのプログラムの視覚化の研究が活発に行なわれてきた。

・視覚的ユーザインタフェース

・視覚的編集

・視覚的言語

第１の視覚的ユーザインタフェースは、プログラミング環境を用いてプログラムを開発するときに、システム側から提供される種々の情報をグラフィカルに表示し、ユーザ側からの入力もそのグラフィカルな画面上で行なえるようにするものである。

第２の視覚的編集は、従来のテキストエディタに代わり、プログラミング言語の文法規則の基本構造をテンプレートとして表示し、文法的に正しいプログラムを誘導するものである。構造エディタや構造化図式を用いた図式エディタがすでに使われている。

第３の視覚的言語は、本来的に視覚的表現を基本とした言語であり、その視覚化に用いる図式の表現形式の違いにより、フォーム指向言語、グラフ指向言語、アイコン指向言語に分類される。

フォーム指向言語は、表形式を基本とするもので、OA分野のスプレッドシートを用いたフォームベースプログラミングが代表的である。グラフ指向言語は、ノードとアークで構成される有向グラフ表現を基本とするもので、データフロー図、状態遷移図などがある。アイコン指向言語は、ビジュアルオブジェクト（アイコン）とその関係表現を基本とするものである。基本的な処理単位をアイコンとして用意しておき、画面上でアイコンとアイコンを結ぶことによってプログラムを作成していく。

以上のビジュアルプログラミング技術のうち、視覚的ユーザインタフェースと視覚的編集はプログラミング言語の概念から開放されていないので、エンドユーザコンピューティングという観点では、不十分であろう。

最近、この視覚的言語レベルの製品としてコンポーネントウェアを用いたビジュアルプログラミングツールがいろいろと出始めている。例えば、IntelligentPad（富士通、日立ソフト）、HOLON/VP（NEC）、APPGALLERY（日立）、VisualAge（IBM）などがある。いずれもオブジェクト指向技術をベースにビジュアルに部品を組み合わせてアプリケーションを構築していく方法や、部品あるいはそれをアプリケーショ

ンに組み込んだインスタンスがオブジェクト指向プログラミング言語で表現されている点などが共通している。部品の組み合わせ方法も部品の重ね合せによる方法か部品間を線で結ぶ方法が一般的である。

(2) その他

エンドユーザが業務のコンピュータ化を行なえるための現実的な技術の例として、ビジュアルプログラミング以外のものについても簡単に説明しておく。

(a) 第4世代言語

エンドユーザコンピューティングを指向した業務向け簡易言語として第4世代言語（4GL）がある。第4世代言語という名前は、第1世代言語（機械語）、第2世代言語（アセンブラ）、第3世代言語（コンパイラ言語）に続く言語という意味で付けられた。James Martinは、4GLを１３ヶ条の特徴で定義したが、その主なものは、非職業的プログラマが使用可能であること、アプリケーションプログラムの記述ステップ数と作成工数がCOBOLより1桁少ないこと、非手続き的記述形式の採用、などである。

4GLユーザの９８％がエンドユーザだけでのソフトウェア開発可能性を否定しているという調査結果もあるが、その理由は、現在の4GLが、COBOLと比較した展開率の向上という「量的高級化」アプローチに基づいて設計されており、プログラミング技術を必要としているためと思われる。

今後、4GLのような業務向け簡易言語をエンドユーザコンピューティングのツールとしていくためには、業務の言葉で要求仕様を記述でき、業務知識のデータベースを用いてプログラムを自動生成できるようにする必要がある。

(b) 日本語プログラミング

職業的プログラマでないエンドユーザにとって、プログラムの日本語表現は書き易さ、読み易さの点で魅力的である。特にアプリケーションプログラムの保守性に不可欠であるプログラムの理解容易性の向上には、プログラム構造がきれいであることと共に、プログラムの各文の意味がわかりやすいことが重要である。

プログラムの日本語表現には次のようなレベルがある。

(i) 既存のプログラミング言語でデータ名や手続き名に日本語を使う。

(ii) 既存のプログラミング言語のif, endなどのキーワードも含め、すべて日本語で記述する。

(iii) 既存のプログラミング言語相当の記述レベルで独自の文法規則を持つ日本語プログラミング言語で記述する。

(iv) 仕様記述言語として日本語を導入し、プログラムを自動生成する。

現在の実用レベルは(ii)、(iii)が主で、4GLに組み込まれているものが多い。今後は(iv)が主流になっていくと思われるが、次のような技術課題がある。

・業務用語の使用に対応した業務用語辞書の作成保守機能

・日本語記述レベルですべて処理するための開発保守環境

・日本語表現のあいまいさを回避する文法規則

日本語化が進んでいる分野としてデータベース検索のコマンド・インタフェースがある。例えば、データベース検索用自然語インタフェースを用いて入力された日本語文をSQL文に変換して実行してくれる。

日本語プログラミングのニーズが大きいもう１つの分野は、金融オンラインシステムのようなメガステップオーダの大規模高信頼ソフトウェアの開発である。これまでは、ユーザの情報システム部門、ソフトウェアハウス、メーカが協力して、COBOLやPL/Iなどのプログラミング言語を用いて開発してきたが、これまで以上の大規模システムの開発は以下のような理由で無理がある。

・計画段階では数年先の稼働時の機能設計は不可能。

・開発管理が規模的に不可能。

・開発中や稼働後の機能変更への迅速な対応が不可能。

これに対し、日本語プログラミング技術を適用すれば、プログラムの仕様が業務仕様書レベルになり、エンドユーザがプログラムの開発や保守に関与できるので、上記の問題は軽くなる。

(c) 人工知能言語

知識ベースシステムやエキスパートシステムが種々の分野で実用になっているが、その理由は、"推論機能"というよりはむしろ"簡易プログラミング"としての魅力にある。職業的プログラマでない業務専門家が業務の言葉で表現できる利点が大きい。さらに、もっともよく用いられている知識表現であるプロダクションルールの場合は記述形式が「もし～ならば、～せよ」という１つのパターンに限定されており、かつ基本的には記述順序が自由であるので、業務知識の記述、追加、修正が容易である。

従来の手続き型プログラムが、N. Wirth の図式で、

プログラム＝アルゴリズム＋データ構造

と表現されたのに対応させると、知識ベースシステムでは、

知識ベースシステム＝推論エンジン＋知識ベース

となるが、アルゴリズムに対応する推論エンジンは既に用意されている。したがって、ユーザは、データに対応する知識だけを記述すればよい。

分散コンピューティングの世界では、このようなルールベースシステムがグループウェアやエージェントとしてお互いにメッセージをやり取りして分散協調システムを形成することになる。

1．1．4　おわりに

新しいアプリケーションの作り方として、最近のコンポーネントウェアとビジュアルプログラミングの動きは注目に値する。コンポーネントの流通市場が確立するような本格的な実用化に向けての課題は多いと思われるが、夢も多い。高度情報化社会を真に豊かな社会とするためには、エンドユーザコンピューティングの成否が鍵である。その成功の秘訣は、従来の生産者中心の視点から利用者中心の視点への転換ではないだろうか。

従来のソフトウェア工学の分野では、論文になった新しい技術が実用になる道のりは意外とけわしく、早くても１０年はかかるが、それは一握りの成功例にすぎない。技術移転の難しさの原因の１つは研究開発におけるエンドユーザの視点の欠如と思われる。幸いにもエンドユーザコンピューティングの分野においては、さすがにエンドユーザの視点の欠如はないであろうが、他の分野以上にエンドユーザの視点に立って、身近なところから始めて本質に迫るというアプローチが必要である。

【参考文献】

1) 青山幹雄：コンポーネントウェア：部品組み立て型ソフトウェア開発技術, 情報処理, 37, 1, 71-79 (1996).

2) 鯵坂恒夫 編集：特集「ソフトウェア生産環境」, コンピュータソフトウェア, 10, 2, (1993).

3) 中所：開発環境, 情報処理ハンドブック, 情報処理学会編, オ-ム社, 747-758 (1995).

4) 中所：M-base：「ドメインモデル≡計算モデル」を志向したアプリケーションソフトウェア開発環境の基本概念, 情報処理学会ソフトウェア工学研究会資料, 95-SE-104, 104-4, 25-32 (1995).

5) 中所：エンドユーザコンピューティング　－ソフトウェア危機回避のシナリオ－, 情報処理, 32, 8, 950-960 (1991).

6) 日本情報処理開発協会(編)：情報化白書1995, コンピュータ・エージ社(1995).

7) Ambler,A.L. and Burnett,M.M. : Influence of visual technology on the evolution of language environments, IEEE Computer, 22, 10, 9-22 (1989).

8) Gamma, E., et al. : Design pattern, Addison Wesley (1995).

9) Grudin, J., Computer-supported cooperative work : history and focus, IEEE Computer, 27, 5, 19-26 (1994).

10) Helm, R. : Patterns in practice, Proc. OOPSLA'95, 337-341 (1995).

11) Jones, C. : End-user programming, IEEE Computer, 28, 9, pp.68-70 (1995).

12) Maes, P. : Agents that reduce work and information overload, Comm. ACM, 37, 7, 30-40 (1994).

13) Malone, T., Lai, K. and Fry, C. : Experiments with Oval : a radically tailorable tool for cooperative work, Proc.

CSCW92, 289-297 (1992).

14) Martin,J. : Fourth generation languages, Prentice-Hall (1985).

15) McLean, E., et al. : Converging end-user and corporate computing, Comm. ACM, 36, 12, 79-92 (1993).

16) Medina-Mora, R., et al. : The action workflow approach to workflow management technology, Proc. CSCW92, 1-10 (1992).

17) Riecken, D. (Ed.) : Special issue " Intelligent agents," Comm. ACM, 37, 7 (1994).

18) Smith, R. B. (Moderator) : Prototype-based languages: object lessons from class-free programming (Panel) : Proc. OOPSLA'94, 102-112 (1994).

1．2　コンポーネントウェア

1．2．1　概要

コンポーネントウェアは、パーソナルコンピュータ（PC）のウィンドウ上で視覚的に部品を組み合せてアプリケーションを構築するための新しいソフトウェア技術である。エンドユーザやシステムエンジニア自らが市販の部品COTS（Components-Off-The-Shelf）を組み合せてアプリケーションを構築する基盤を提供する。

コンポーネントウェアは次の３つの点でソフトウェア開発のあり方、ひいてはソフトウェア産業構造に根本的な変革を迫る。

(1)部品組み立て型ソフトウェア開発の実現

従来のソフトウェアの部品化・再利用やオブジェクト指向による再利用技術の成果と問題点を踏まえて、プラグ& プレイ型ソフトウェア部品の再利用を実現するための実践的技術を体系化し、それを実働可能なソフトウェアとして提供する。

(2)EUC（エンドユーザコンピューティング）の実現

PC上でエンドユーザがアプリケーションを組み立てる具体的方法を提供するので、ソフトウェア開発の新しい分野と市場を拓く。

(3)ネットワーク上での分散システム構築概念の変革

さらに、コンポーネントウェアはネットワーク上で分散するオブジェクトが相互に連携してシステムを構築するアーキテクチャを提供する。したがって、従来の再利用のようにシステムを構築するために、部品を一箇所に集める必要はない。ネットワーク上に分散する部品を分散したまま利用してアプリケーションを実現する新しいシステムアーキテクチャの構築方法を実現する。

ここでは、コンポーネントウェアの核となる要素技術を整理してその全体像を示す。

1．2．2　コンポーネントウェアとは

(1)コンポーネントウェアとは

コンポーネントウェア（Componentware）は,広義には『オブジェクト指向に基づき部品の組立てによりシステムを開発する技術の体系』と言える[2, 3]。英語ではComponent-Based SoftwareやComponent Softwareとも呼ぶ。狭義には、このような部品組立によるアプリケーション開発を支援す開発環境やシステムの総称としても使われている。Byte誌のComponetwareに関する記事[15, 19]により広く知られるようになった。

コンポーネントウェアは従来のオブジェクト指向による再利用技術と何が違うのであろうか。図1.2-1の

例題は、ある地方自治体における人口分布の集計報告書である。最初の要約説明にあわせ、データベースの商品別売上データの検索、集計結果が表とグラフで示されている。

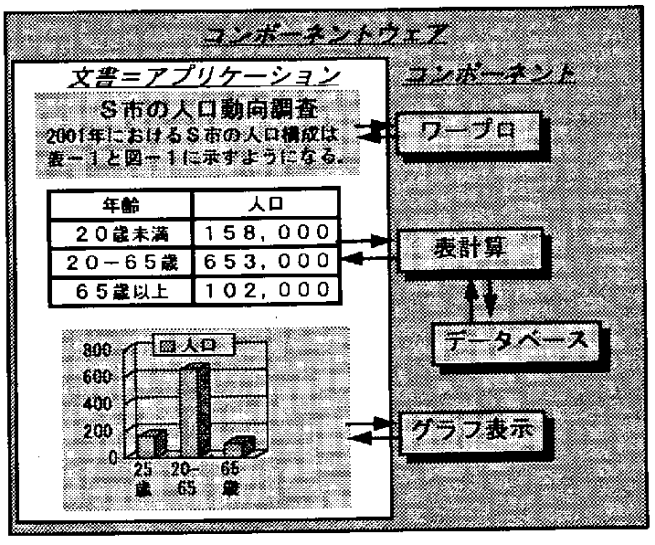


図1.2-1　コンポーネントウェアとは

コンポーネントウェアでは、この文書（Document）を一つのアプリケーションと考える。オブジェクト指向と対比すれば、オブジェクトはテキストやグラフをカプセル化した文書に、データは文書中のテキストやグラフに、メソッドがテキストやグラフを操作するワードプロセッサや表計算ソフトに相当する。したがって、文書はオブジェクト指向のオブジェクトと同様に広い意味で用いる。このような文書を複合文書（Compound Document）と呼ぶ。複合文書は、従来の文書の概念を越えて，プログラムやデータベースを貼る台紙と見なせる。ハードウェアに例えれば、CPU やメモリを組み込んだPCのマザーボードに相当する。

利用する部品は、この例のようなワードプロセッサや表計算ソフトといった市販パッケージからいくつかのメソッドを提供するオブジェクトまで多様である。コンポーネントウェアでは、このように『文書』のメタファにより部品を組み合せる方法を提案する。

さらに、複合文書はネスト構造をとり、他の複合文書を部品として利用できる。複合文書を組み合せると、より高度で大規模なアプリケーションを構築できる。この例でも、市町村毎の報告書をまとめると県全体の人口分布を作成するアプリケーションとなる。このような報告書などは、実際のビジネスの基本的な要素であり、ビジネスオブジェクトとも呼ばれている。

(2) なぜ、コンポーネントウェアか

コンポーネントウェアが出現した背景をニーズとシーズの両面から次のように理解できるだろう。

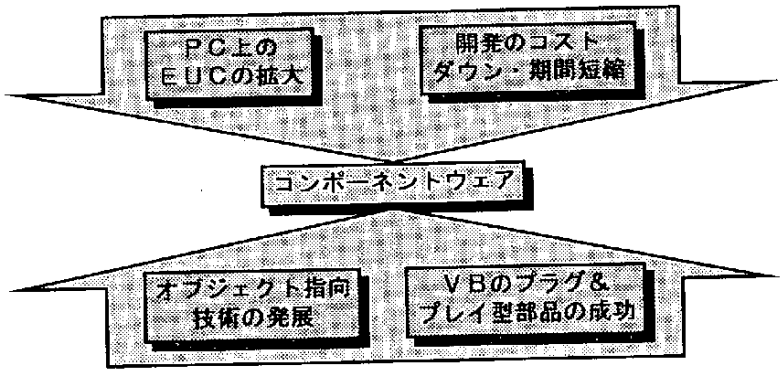


図1.2-2　コンポーネントウェアの背景

(a) ニーズ

(i) パソコン上でのEUC

パソコンのビジネス利用の普及に伴い、パソコン上で処理すべき作業も複雑になった。このため、表計算ソフトやデータベースを組み合せて、複雑な仕事を実行できるアプリケーションを簡単に開発するEUC（エンド・ユーザ・コンピューティング）の必要性が高まった。

(ii) ソフトウェア開発の大幅なコストダウンの必要性

パソコンなどのハードウェアの価格低下に見合ったソフトウェア開発コストの低減が要求されている。さらに、開発期間の短縮が競争力の鍵となっている。しかし、従来のコーディングを主体とした開発方法では限界がある。

(b) シーズ

(i) オブジェクト指向の発展

オブジェクト指向の発展に伴い、フレームワークや協調オブジェクトなどの再利用の要素技術が成熟してきた。

(ii) 使いものになるプラグ& プレイ型再利用部品の出現Visual Basicの再利用部品であるカスタムコントロール（そのファイル識別子から、通常VBX と呼ばれる）が多くのベンダから販売され、商業的に成功を収めた。今や、ベンダ数も２００社に達し, ソフトウェア部品市場が形成されつつある[9]。この部品は、組み込んですぐ使えることからプラグ& プレイ型ソフトウェア部品とも呼ばれるようになった。

(3) コンポーネントウェアのアプローチ

コンポーネントウェアは、従来の再利用では実現できなかった次のような技術的なアプローチにより特徴づけられる。

(a) プラグ& プレイ型再利用

ハードウェア部品のように、組み込んですぐ使える部品再利用を目標とする。従来の再利用は、ソースコードの再利用であり、その実装を利用するホワイトボックス型再利用であった。コンポーネントウェアでは、オブジェクトコードを再利用するブラックボックス型再利用へと発想を転換する。これは、次の点で、アプリケーションを開発方法の転換を意味する。

(i) 部品の独立性の実現：部品の実装が物理的にも隠蔽されるので、部品を独立した製品として販売、流通、利用できる。

(ii) コーディングレスによるアプリケーション構築：利用者が必要な部品を組み合せてアプリケーションが構築できる。プログラミングの非専門家でもアプリケーションが構築できる道を開く。

(iii) 保守性の向上：コーディング量が減るので保守の生産性・品質が向上する。

(b) サービス（インタフェース）の再利用

ブラックボックス型再利用では実装が隠蔽されるので部品のインタフェース、すなわち提供するサービスが再利用の対象となる。この概念は、クライアント／サーバシステムの開発に適している。さらに、従来の再利用とは異なり、部品はネットワーク上に分散していてもよいので、アプリケーションはネットワーク上に分散する部品の協調によって実現される。アプリケーション構築の概念や方法を変えるものである。

(c) 文書（タスク）指向

従来のアプリケーション構築では、仕様に基づき、プログラムに順次データやファイルなどの情報を渡して処理を行うものであった。コンポーネントウェアでは仕様を記述した文書がアプリケーションであると言ってよい。文書は、なすべきタスクの文書化である。このため、コンポーネントウェアは、文書指向(Document-Oriented), あるいは、タスク指向（Task-Oriented）とも形容される。アプリケーションの実現とは、文書にタスクを実現するために必要な部品を組み込むことである。

(d) アーキテクチャ支援：協調オブジェクト

部品を再利用するためには、ハードウェアやOSによらず部品を組み込むことができる標準的な基盤ソフトウェアが必要である。コンポーネントウェアでは、ベンダ、版数、プロセッサなどの違いを越えてオブジェクトを組み合せるための協調オブジェクト（Interoperable Objects）環境ソフトウェアを提供する。協調オブジェクトはミドルウェアの一つであり、分散したオブジェクトの連携を強調する場合には分散オブジェクト（Distributed Objects）とも呼ばれる[14]。

(4) コンポーネントウェアとオブジェクト指向

コンポーネントウェアはオブジェクト指向技術を基礎としているが、実現されたシステムでは必ずしも

オブジェクト指向の技術を全て利用しているとは限らない。表1.2-1は,概念的にオブジェクト指向による再利用とコンポーネントウェアのアプローチを対比したものである。

表1.2-1　オブジェクト指向とコンポーネントウェア

従来のオブジェクト指向再利用	コンポーネントウェア
ホワイトボックス	ブラックボックス
ソースコード*	オブジェクトコード
クラス	インスタンス
実装	インタフェース
静的リンク主体	動的リンク主体

とくに、コードを再利用するための実装の継承（Implementation Inheritance）は、上位クラスの変更が下位クラスに影響を及ぼす,いわゆる脆弱な基底クラス問題（Flagile Base Class Problem）を引き起こすので、その利用については、コンポーネントウェアの中でも議論が分れている。

1．2．3　コンポーネントウェアの要素技術

(1)コンポーネントウェアの技術体系

コンポーネントウェアは多様な技術の集成である。ここでは、図1.2-3に示す3階層に整理して紹介する。

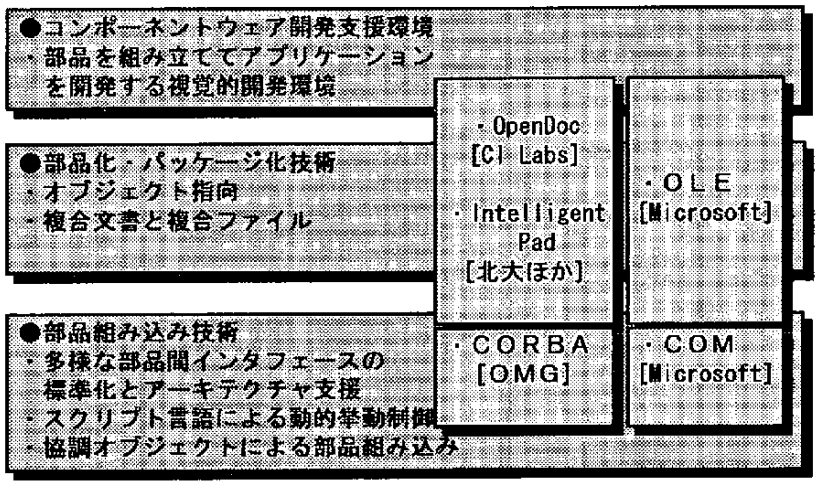


図1.2-3　コンポーネントウェアの技術体系

(2)部品パッケージ技術

部品は単一のプログラムやオブジェクトよりも複数のプログラムやオブジェクトにより実現されること

が多いので、複数のオブジェクトをまとめて再利用する仕組みが必要である。このために開発されたのが、複合文書とそれを支える複合文書ソフトウェアアーキテクチャである。

(a) 複合文書

図1.2-4 に複合文書と複合ファイルシステムの概念を示す。複合文書とは、オブジェクト群をパッケージ化してそのインタフェースを標準化したものである。データと手続きをカプセル化してオブジェクトが構成されるように、複数のオブジェクトをカプセル化したものと解釈できる。

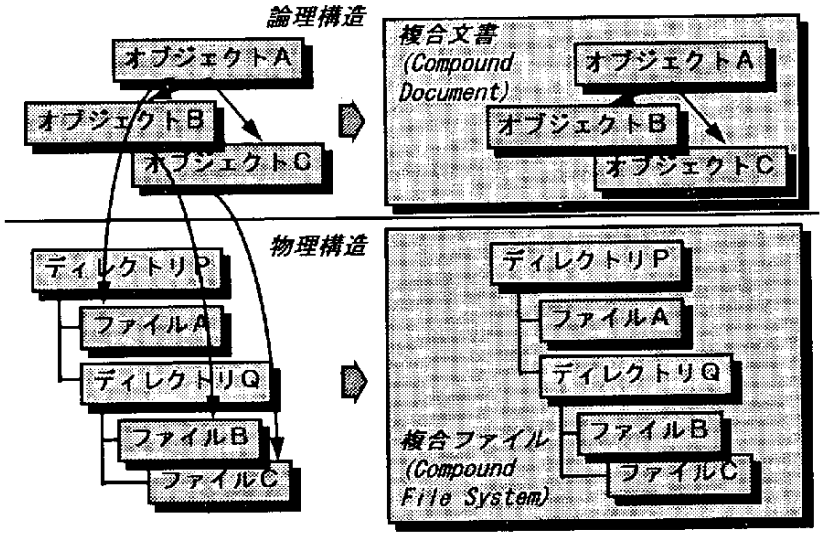


図1.2-4　複合文書と複合ファイルシステムの概念

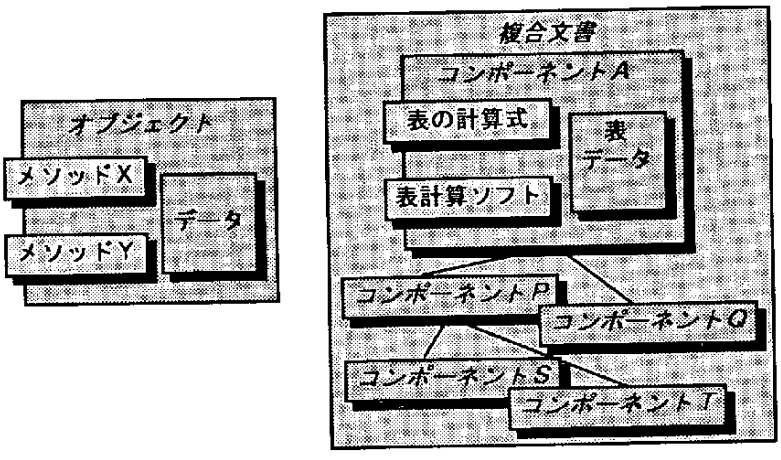


図1.2-5　オブジェクトと複合文書

図1.2-5 にOpenDoc の複合文書の構造をオブジェクト指向のオブジェクトの構造と対比して示す。複合文書の内容は、部品のデータ形式に対応したビューヤ、あるいはパーツエディタで参照、更新できる。

複合文書を構成する要素の名称はコンポーネントウェアのシステムによって異なる。代表例として、OLE（Object Linking and Embedding）、OpenDoc、IPad（IntelligentPad）の複合文書の要素名の例を表1.2-2に示す。

表1.2-2　複合文書システムの例

システム名	ＯＬＥ	ＯｐｅｎＤｏｃ	IntelligentPad
文書全体	コンテナ	コンテナ	親パッド
文書内部品（階層上位から）	文書，ページ，サイト	パーツ	子パッド
文書格納ファイル形式	複合ファイル	Ｂｅｎｔｏファイル	フレームモデル

(b) 複合ファイルシステム

複合ファイルシステム（Compound File System）は、構造化ストレージ（Structured Storage）とも呼ばれ，複合文書を格納するファイルシステムである。従来のファイルシステムでは複数のオブジェクトやデータを効率良く管理できない。複合ファイルは、階層構造をとる複数のファイルやディレクトリをまとめて一つのファイルとして管理する。OLE の複合ファイルシステムの構造を図1.2-6 に示す。ページとテナントは、従来のファイルシステムのディレクトリに相当する。このため、複合ファイルシステムは、『ファイル内ファイルシステム』とも呼ばれている。なお、OpenDoc のBento ファイルシステムはわが国の弁当に由来すると言われている。

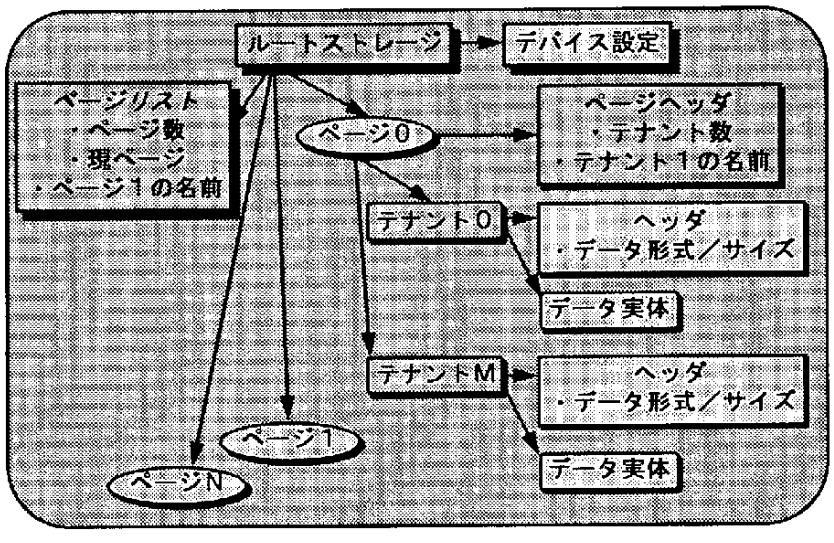


図1.2-6 複合ファイルシステムの例

(3) 部品組込み技術

多様な部品を組み立ててアプリケーションを構築するためには、部品間のインタフェースを統一する必要がある。コンポーネントウェアでは、図1.2-7 に示すように、プログラム間インタフェースだけではなく、データやユーザインタフェースなどのソフトウェア部品間の多様なインタフェースを標準化している。

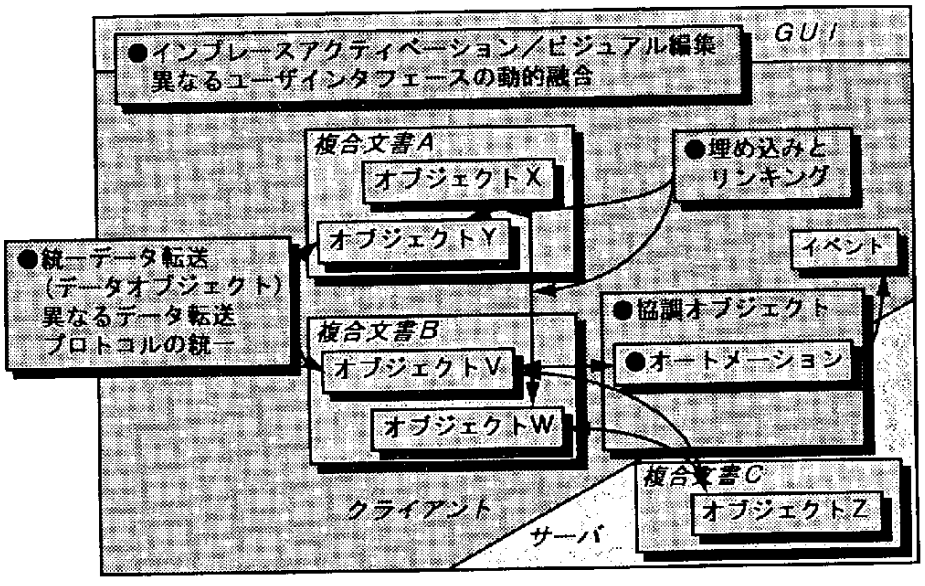


図1.2-7 部品間インタフェースの統合

(a) 部品の組込み：埋込みとリンキング

複合文書に部品を組込む方法として、図1.2-8 に示す 2 つの方法が提供されている。

(i) 埋込み（Embedding）：オブジェクト全体を文書の中に組込む方法。データ量が少ない場合や処理速度が要求される場合に用いる。

(ii) リンキング（Linking）：オブジェクトの実体は文書の外や他の文書内にあり、ポインタにより参照する方法。データ量が多い場合や共通データの場合に用いる。リンキングでは、オブジェクトの移動に対してリンクを保持する機構も提供されている。

なお、IntelligentPadでは『紙』のメタファに基づく『貼り合せ』の概念によってオブジェクトを統一的に結合する方法を提案している。

(b) 起動インタフェースの統一：協調オブジェクト

ベンダ、物理的位置、実装言語などの違いによらず静的、あるいは動的にオブジェクトのメソッドを起動できる仕組み。次の協調オブジェクトの項で詳細に述べる。

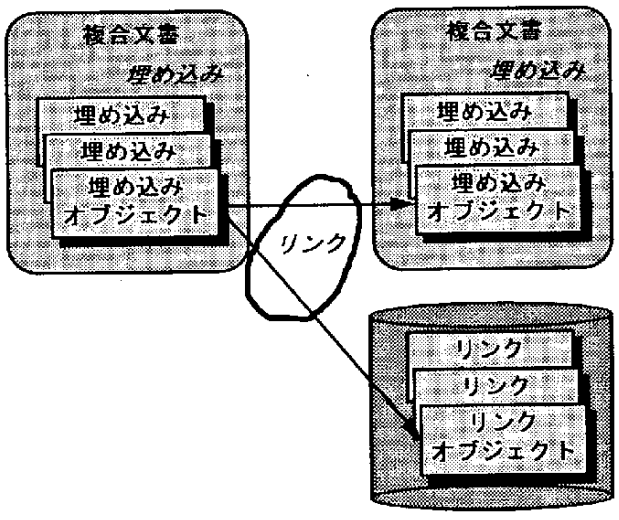


図1.2-8　埋込みとリンク

(c) データ交換の統一：OLE のデータオブジェクト

OLE ではデータオブジェクトと呼ばれるアプリケーション間でのデータ交換用標準オブジェクトが提供されている。データオブジェクトを使うと、データ転送のプロトコルを統一できる。たとえば、従来使われていた、クリップボード、ドラッグ& ドロップ、DDE（Dynamic Data Exchange）などのデータ交換プロトコルを意識することなく部品間で統一的にデータ転送が可能である。

(d) ユーザインタフェースの統一：インプレースアクティベーション（ビジュアル編集）

インプレースアクティベーション（In-place Activation）はビジュアル編集（Visual editing）とも呼ばれ、オブジェクトごとに異なる操作環境を融合する仕組みである。たとえば、ワープロの文書中に張り込んだ表計算のシートを編集する場合、従来は、文書（この場合、表のファイル）を表計算ソフトに渡す必要があった。インプレースアクティベーションでは、逆に、文書内にアプリケーションの起動プロトコルを提供し、ワープロの編集ウィンドウに表計算ソフトのウィンドウを融合する。この結果、同一ウィンドウ内で異なるオブジェクト（アプリケーションや部品）をそのユーザインタフェースによらず結合できる。一方、利用者は、メディアやユーザインタフェースの違いを意識せずに異なるアプリケーションを利用できる。

1．2．4　協調オブジェクト

(1) コンポーネントウェアと協調オブジェクト

異なるベンダや言語で実装された部品の提供するサービスを位置によらず再利用するための仕組みが協

調オブジェクトである。コンポーネントを実現する基盤技術である。クライアント/サーバシステム上でオブジェクト指向を用いてアプリケーションを構築するためにも重要な役割りを果たす。

オブジェクト間の協調を実現するためには、次の2 つの技術が必要である。

(a) オブジェクト間標準プロトコルの提供：オブジェクト間で言語などの実装に依存しない標準プロトコルの提供。

(b) 分散オブジェクト管理の提供：異なるプロセスやプロセッサ（すなわちメモリ空間）間でオブジェクトの物理的な位置によらない管理機構の提供。

(2) オブジェクト間協調の問題点とアプローチ

協調オブジェクトの技術は、図1.2-9示すようなオブジェクトが協調する上での問題点に対するアプローチを採る。

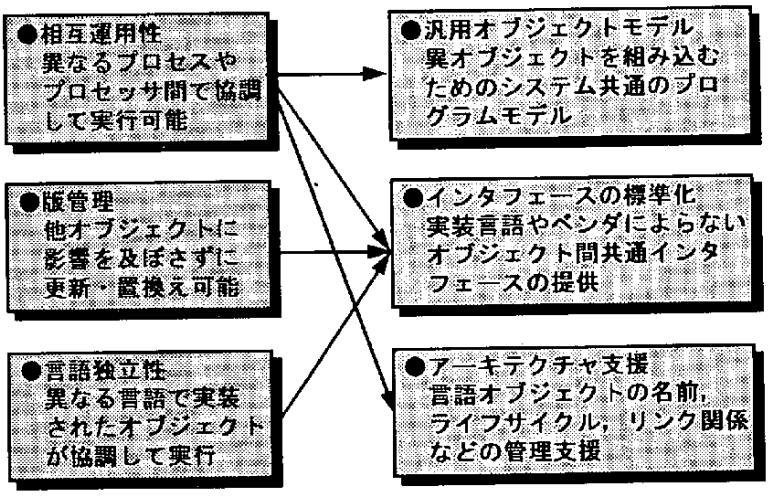


図1.2-9 オブジェクト協調の問題とアプローチ

(3) 古典モデルから汎用モデルへ

従来のオブジェクト指向では、『まずクラスありき』とも言えるように、クラスの構造が中心となっていた。これを、古典モデル（Classical Model）と呼ぶ場合がある。一方, オブジェクトが協調するには、オブジェクト間インタフェースがむしろ重要である。『まずインタフェース（＝仕様）』ありきで、インタフェースと実装は分離されるべきだと考える[16]。

(4) IDL の意義

IDL は、次の点でオブジェクト間のインタフェースを統一する役割を果たす。

(a) 仕様（インタフェース）と実装の分離

インタフェースの言語独立、版数独立、ベンダ独立の実現

(b) 強い型づけによるインタフェース間の整合性の保証

(c) アーキテクチャ支援：名前の管理とオブジェクトの静的、動的起動。

(5) IDL の実際

IDL にはオブジェクトのシグネチャ（Signature）と呼ばれるメソッドや関数の名前、パラメータ、型などの対が定義できる。図1.2-10にOMG（Object Management Group）で標準化されたCORBA（Common Object Request Broker Architecture）のIDL 記述例を示す。文法的にはC++ 言語と類似しているが、仕様は言語とは独立である。

```
/* IDL Definition: CORP.idl */
module CORP
{
 . . . . . .
typedef long    EmpID;  //型定義
typedef long    DepID;

interface Employee      // インタフェース(クラス)定義
    {
            attribute EmpData     personal_data;
    readonly attribute Department  department_obj;
    void promote  (in char         new_obj_class); //メソッド定義
     . . . . . .
    void transfer (in Department   new_dep_obj);   //メソッド定義
}; // Employeeインタフェース定義

}; // module CORPの終り
```

図1.2-10　CORBA のIDL の記述例

一方、COM の使用するIDL はC 言語に依存した実現となっている。

(6) IDL の処理

CORBA では、図1.2-11に示すように、IDL を介して静的起動と動的起動を統一的に処理できる。

(a) IDL はコンパイルされるとスタブとスケルトンを生成し静的に起動する。

(b) インタフェースリポジトリを介して動的なメソッドサーチを行い、該当したメソッドを起動する。

OMG は、CORBA1.2でC に，またCORBA でC++ とSmalltalk へのスタブ／スケルトンの生成仕様（バインディングと呼ぶ）を規定している。Ada95, COBOLへのマッピングも検討されている。

COM では、静的起動と動的起動は別々のIDL を使用する。動的起動を行うために、図1.2-12に示すオートメーション（Automation）と呼ばれる機構がある。動的起動を行うためのIDL ともいえるオブジェクト

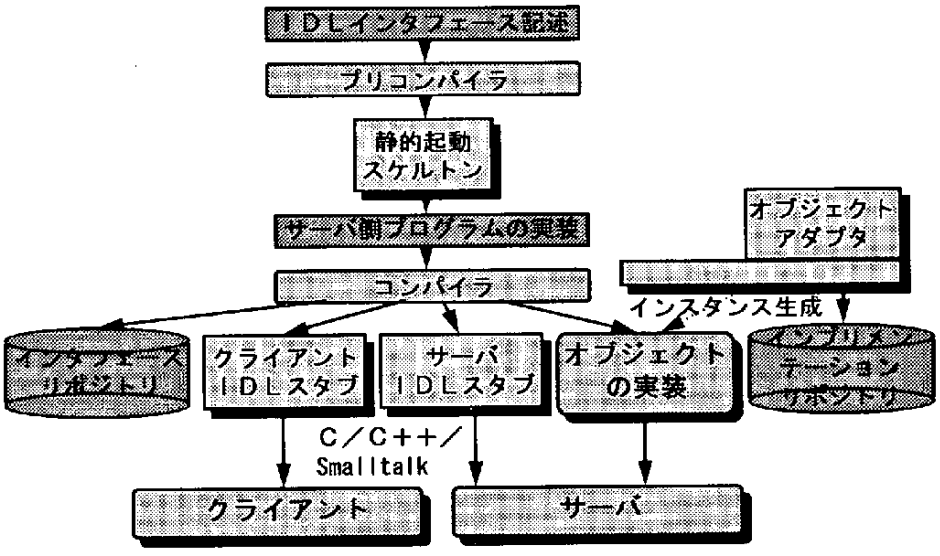


図1.2-11 IDLのコンパイル

記述言語ODL（Object Description Language）によりインタフェースを定義し、CORBAのインタフェースリポジトリに相当する型ライブラリに登録（Registry）する。クライアントであるコントローラからの動的呼び出しに対するサーバの名前や位置は型ライブラリを用いて解消される。

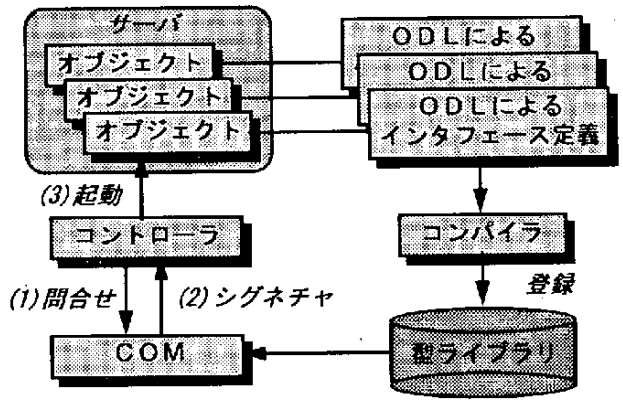


図1.2-12 OLEのオートメーション

(7) アーキテクチャ支援

協調オブジェクトを支援する環境としては、OMGのCORBAとMicrosoftのCOM（Component Object Model）が事実上の標準といえる。CORBAは多くのベンダによって実現されている。

また、MicrosoftとDECはOLE/COMとCORBAとの連携を実現するOLE Integration（旧称：COM

（Common Object Model））も開発している。

(a) CORBA

CORBA は、図1.2-13に示すように、クライアント／サーバアーキテクチャを採る。

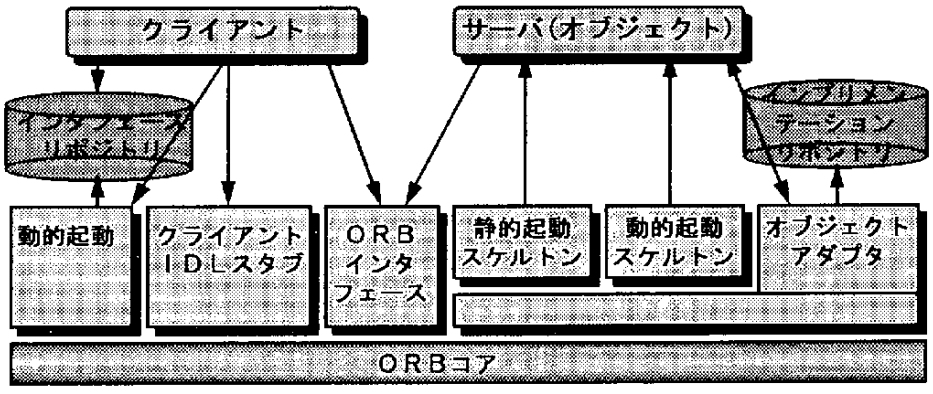


図1.2-13　CORBA アーキテクチャ

CORBA の主な特徴は次の 3 つである。

(i) 分散オブジェクトモデルの支援：処理環境上で分散したオブジェクトが協調可能。

(ii) 言語独立：IDL は特定の実装言語から独立。

(iii) 単一のIDL で静的、動的な起動が可能。

(iv) 仕様がオープンである。

(b) COM

図1.2-14にCOM のアーキテクチャを示す。クライアント／サーバアーキテクチャを採る。ただ、現段階では同一プロセッサ内での連携に留まっている。

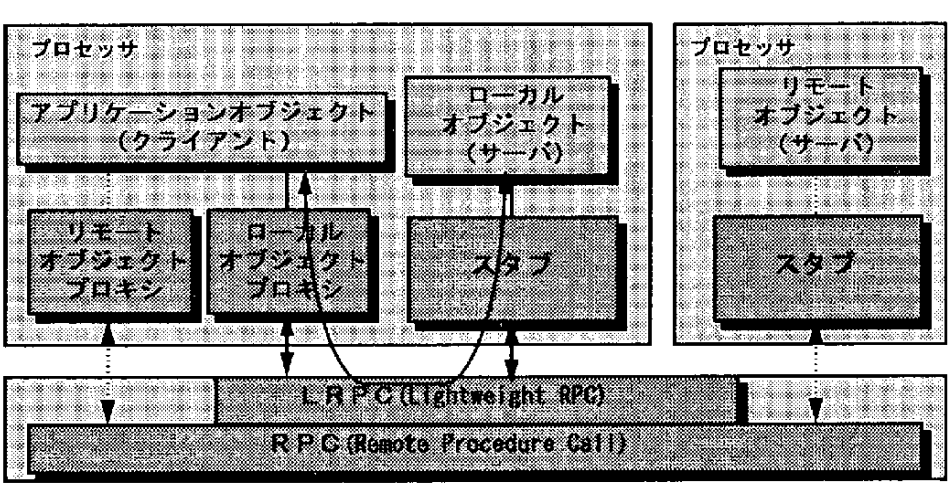


図1.2-14　COM アーキテクチャ

サーバオブジェクトがクライアントと同一プロセッサ内にあっても異なるプロセッサにあっても呼出し手続は同一であるとしている。同一プロセッサ内でのサーバオブジェクトの呼出しは軽量RPC（LRPC: Light-weight RPC）と呼ぶ。

(8) インターネット上でのオブジェクト協調

CORBA1.2では異なるベンダのCORBA間で連携ができない問題であった。このため、CORBA2.0では、図1.2-15に示すように、異なる実装のCORBA 間連携としてGIOP（General Inter-ORB Protocols）が、インターネット上でのCORBA 間連携としてIIOP（Internet Inter-ORB Protocols）が規定された[13, 14]。これは、ネットワーク上でソフトウェア再利用を行う機構として興味深い技術である。

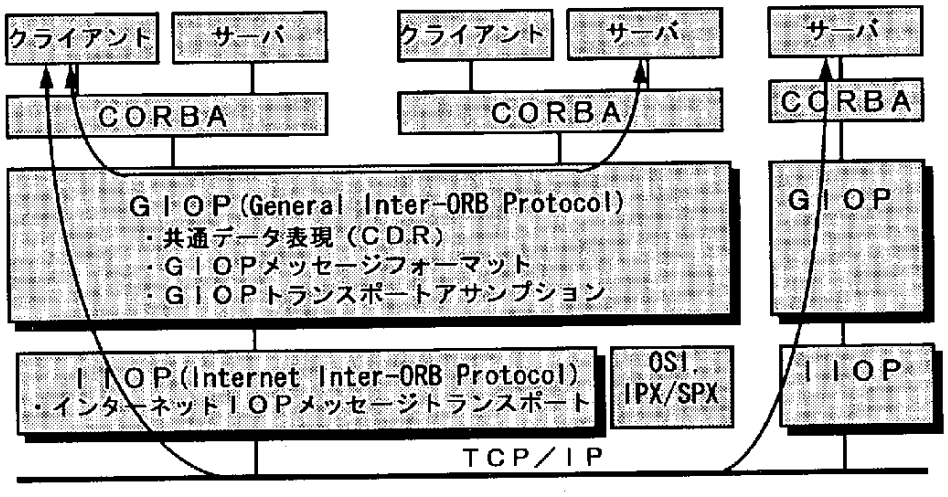


図1.2-15 インターネット上でのCORBA 連携

(9) スクリプティング

コンポーネントウェアではコーディングレスを追求しているが、部品のきめ細かい挙動や部品群の協調的あるいは集団的な挙動を制御するために外部から実行を制御できる必要がある。このため、スクリプト言語を用いて、部品の実行を制御する機構が提供されている。OpenDoc, OLE ではスクリプト言語として任意の言語が利用できる。実際には、OLE ではVisual Basic（3.0以降）などを用いる。OpenDoc のOSA（Open Scripting Architecture）では、Apple Scriptなどの幾つかのスクリプト言語が用意されている。OSA でのスクリプティングの実行例を図1.2-16に示す。パーツを操作するためのイベントをスクリプト言語とは独立に標準化し、セマンテックイベントと呼んでいる。これは、イベントに対するIDL と考えてよい。スクリプト言語ごとに作られるOSA スクリプティング・ランタイムがスクリプティング言語固有のイベントを標準のセマンティクイベントに変換する。したがって、OpenDoc 内部ではスクリプト言語によ

らず統一したイベントを使用できる。

COM のオートメーションは、スクリプティングによる部品の操作も意味する。部品（群）を操作する一連の手続きを自動化し、あるまとまった機能を実現する意味である。オートメーションの起源は、表計算ソフトの一連の手続きを自動実行する『マクロ言語』を汎用化することにあった。

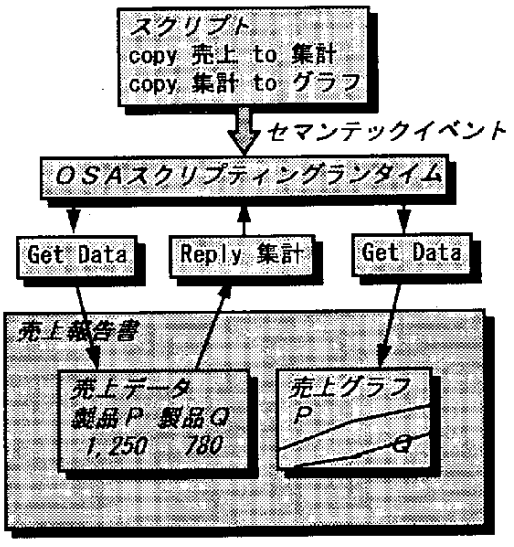


図1.2-16　OSA によるスクリプティングの例

１．２．５　コンポーネントウェアの開発環境と実現例

(1) アプローチ

コンポーネントウェアによる開発環境を実現するアプローチは大別して次の２つがある。

(a) 複合文書のアプローチ：複合文書を中心として、その組立てによってアプリケーションを開発する方法。

(b) オブジェクト指向開発環境の拡張：Smalltalk の開発環境を拡張して、視覚的に部品を結合してアプリケーションを開発する方法。

表1.2-3 にこれらのアプローチによるコンポーネントウェアの実現例と動作基盤環境を示す。

(2) 複合文書による実現例

複合文書によるコンポーネントウェアの例として、OpenDoc, OLE, IntelligentPadを紹介する。

(a) OpenDoc [12, 14, 20]

OpenDoc は Apple, IBM, Taligent, Novell, SunSoft などにより設立された非営利団体である CI Labs（Component Integration Laboratories）により開発と標準化が進められている。そのため、プラットフォームなどのオープン性が強調されており、仕様も公開されている。

表1.2-3 コンポーネントウェア支援環境のアプローチ

アプローチ	アーキテクチャ	システム名	開発元
複合文書	分散処理	OpenDoc	CI Labs: Apple他注1
複合文書	分散処理	CommonPoint	Taligent
複合文書	集中処理	ＯＬＥ	Microsoft
複合文書	集中処理	APPGALLERY	日立
複合文書	集中処理	IntelligentPad	ＩＰコンソーシアム注2
ＯＯ開発環境	分散処理	VisualAge	ＩＢＭ
ＯＯ開発環境	分散処理	HOLON/VP	ＮＥＣ
ＯＯ開発環境	集中処理	PARTSWorkbench	ParcPlace-Digitalk
ＯＯ開発環境	集中処理	VisualWorks	ParcPlace-Digitalk

注１：Component Integration Laboratories＝Apple，IBMなどが1993年に設立した非営利団体
注２：ＩＰコンソーシアム＝北大の田中教授を中心に富士通，日立などが設立した非営利団体

図1.2-17にOpenDoc のアーキテクチャを示す。協調オブジェクト環境としてIBM で開発されたCORBA準拠のSOM（System Object Model）/DSOM（Distributed SOM） を採用している。また、複合文書の技術やBento ファイルシステムはApple で開発されたものである。MacOS とOS/2 Warp, AIX版のOpenDoc 1.0 と開発キットが提供されている。Windows 版は１９９６年が公開予定されている。

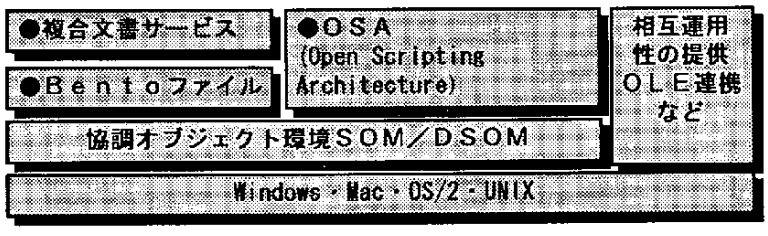


図1.2-17 OpenDocのアーキテクチャ

(b) OLE [5]

OLE（オーレと呼ぶ）はMicrosoft で開発され, Windows 上で利用できるコンポーネントウェアである。OLE のアーキテクチャを図1.2-18に示す。

協調オブジェクト環境としては、同社で開発されたCOM を利用している。現在は実行OSがウィンドウに限定されているが、UNIX, MVS への移植計画も発表されている。

Visual Basicの部品であるカスタムコントロール（VBX）やOLE 準拠のカスタムコントロール（OCX）として多くの部品が数十～数千ドルで市販されている。

OLE の上で視覚的に部品を結合してアプリケーションを構築する開発環境も開発されている。例えば、

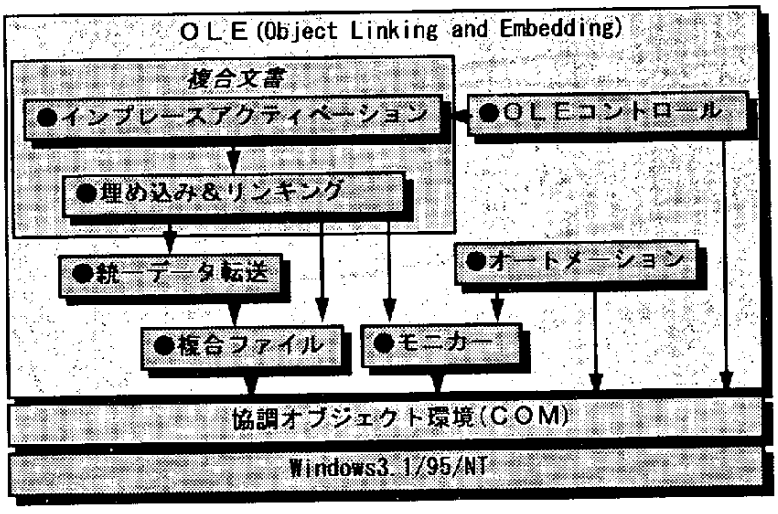


図1.2-18　OLE のアーキテクチャ

２．１節で紹介する日立のAPPGALLERYでは、部品をキャンバスと呼ぶ台紙の上で結びアプリケーションを構築できる。

(c) IPad（IntelligentPad）[6, 11]

IPadは北海道大学の田中教授により提案され、IPコンソーシアムにより開発が進められている。図1.2-19に、そのアーキテクチャを示す。詳細は、２．３節に示す。マルチプラットフォームを指向し、OLE との連携を図るなど、他のコンポーネントウェアとの連携も支援している。さらに、データベース連携などIPadの部品も開発、提供されている。

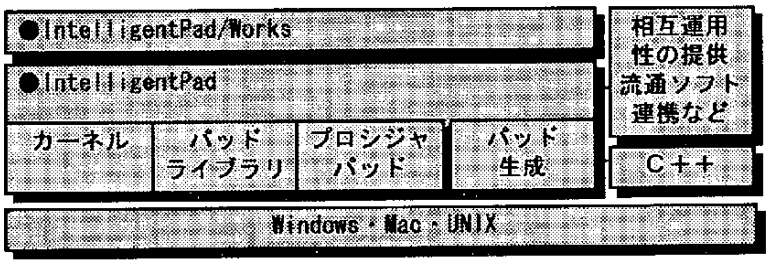


図1.2-19　IntelligentPadのアーキテクチャ

(d) CORBA 共通ファシリティアーキテクチャ [14]

OMG のBOM（Business Object Management）SIGでは、図1.2-20に示すように、協調オブジェクト環境ORB の上に共通ファシリティアーキテクチャとして、業務アプリケーションの構成要素となる部品を２

階層で標準化している。

(i) ドメイン独立な共通ファシリティ（Horizontal Common Facility）：複合文書、複合ファイル、オートメーションなどのコンポーネントウェアを実現する基本アーキテクチャである。OpenDoc などのコンポーネントウェアの要素技術が提案されている。

(ii) ドメイン固有な共通ファシリティ（Vertical Common Facility）：通信、製造、会計など特定アプリケーションドメインに固有なオブジェクト群である。

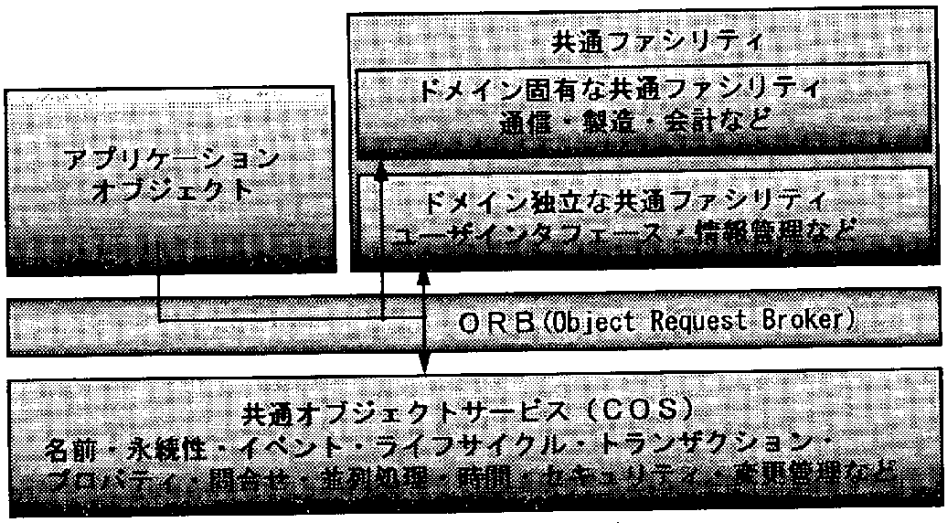


図1.2-20 CORBA 共通ファシリティアーキテクチャ

(e) CommonPoint [10, 18]

CommonPoint は、IBM, Apple, HPにより設立されたTaligentが開発しているアプリケーションフレームワークを中心とする開発支援環境である。図1.2-21にアーキテクチャを示す。

CommonPoint が提供する １００個のフレームワークは １,７３０個のクラスから成り, ５３,０００個のメソッドを公開する。プログラム規模は ６００K行に達している。

(f) Java [7]

Java はSun Microsystemsで開発されたC++ をベースとするオブジェクト指向言語である。Javaの実行環境であるHot Javaを用いて、インターネットのWWW ブラウザの上でJavaで作成した部品アプレット（Applet）をダウンロードし実行することができる。Javaによって、WWW 上でアニメーションや対話的な実行が可能となる。

(3) オブジェクト指向開発環境の拡張による実現例

このアプローチを採る実現例としては、VisualAge, HOLON/VP, PARTSWorkbench などがある。いずれ

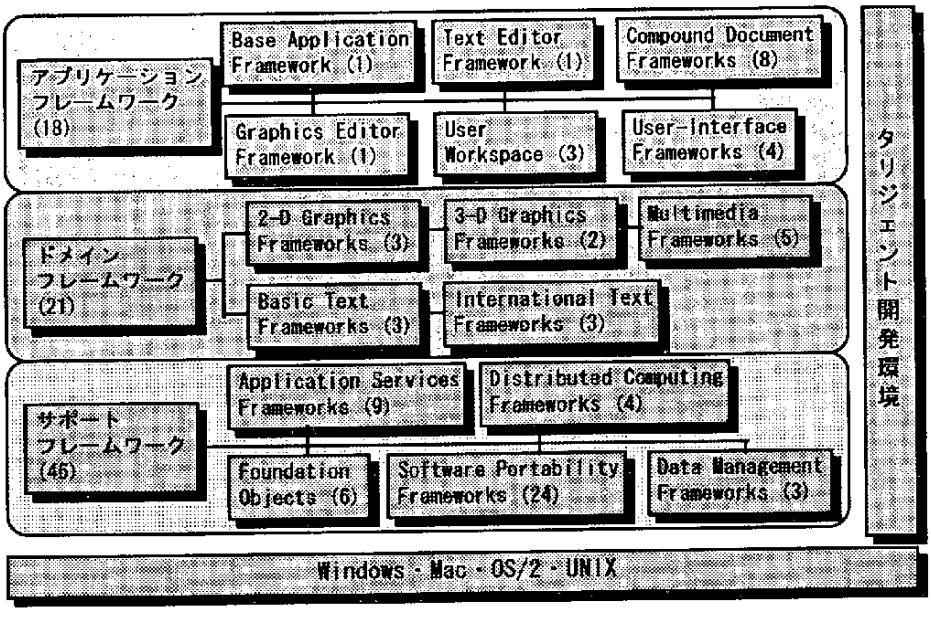


図1.2-21　CommonPointのアーキテクチャ

も、オブジェクト指向言語による視覚的プログラミング支援環境として実現され、製品化されている。ウィンドウ上で部品を線でつなぐことにより部品を結合できる。これらの例では、Smalltalk 開発環境を基盤環境として利用するシステムが多い。また、実際に、GUI やデータベースアクセスなどの部品も提供されている。

また、C やCOBOL など他の言語で作成された部品をラッピングして利用するためのラッパーや処理速度を向上するためにC++ のコードを生成しコンパイルする機構など、実践的な工夫が凝らされているシステムもある。

1．2．6　コンポーネントウェアによる開発形態の変化

コンポーネントウェアがソフトウェア開発形態に及ぼす影響はまだ明らかになっていない。ここでは、いくつかの事例から、開発形態と開発プロセスの面で、開発のあり方を示そう。

(1) コンポーネントウェアによる開発形態

(a) 部品組立て型開発：コーディングレスの開発：市販パッケージや部品の組合せによりアプリケーションを構築できる。

(b) ビジュアル開発：ハードウェア設計における結線のようにソフトウェア部品を線で視覚的に結合してアプリケーションを設計できる。

(c) プロトタイピング開発：部品を組立てて迅速にプロトタイプが構築できる。

(d) EUC 指向：エンドユーザがワードプロセッサ、表計算、グラフ出力などのパッケージソフトウェアを組み合せてOAなどのアプリケーションを構築できる。

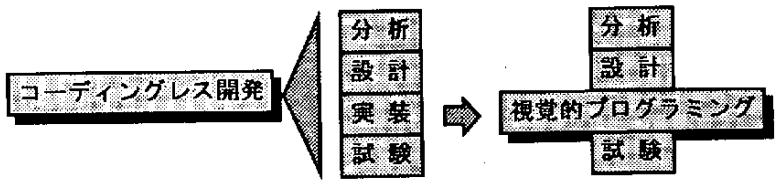
(e) クライアント／サーバシステムの開発：多くのコンポーネントウェアがクライアント／サーバアーキテクチャを基礎としているので、クライアント／サーバシステムのアプリケーション開発に適する。さらに、ネットワークに上に分散する部品をクライアントから再利用するような新しいアプリケーション構築方法の道を開く。

(2) コンポーネントウェアの開発プロセス

コンポーネントウェアによる開発形態は、次の2つがあるだろう。

(a) エンドユーザによる組立て型開発

既存のパッケージソフトウェアなどの粗粒度の部品を組み合せて、OAなどのより高度なサービスを実現する[4]。Visual Basicでアプリケーションを開発した多くの事例が報告されている[9]。この場合、図1.2-22に示すように, インクリメンタルでイテラティブな開発プロセスが適しているだろう。



システム	複合文書を基礎とする	ＯＯ開発環境の発展
開発形態	コーディングレス開発	視覚的プログラミング
開発者	エンドユーザ・ＳＥ	プログラマ（専門家）
適用工程	全工程	実装（プログラミング）
開発方法	市販パッケージや部品を組み合せてアプリケーションを構築	オブジェクト指向開発の一環としてフレームワークや部品を組み合せてアプリケーション開発
適用範囲	ＯＡなど比較的限定	限定なし

図1.2-22　コンポーネントウェアの開発プロセス

(b) 専門家によるアプリケーションの開発

中・大規模アプリケーションの開発では、開発形態に応じてコンポーネントウェアを使い分ける。

(i) CASE環境として利用：従来型開発プロセスにおいて, 下流CASEとしてコンポーネントウェアを利用する。

(ii) システムインテグレーション：エンドユーザでは実現が困難なアプリケーションを部品の組立てによっ

て実現したり、顧客の要求に応じてカスタマイズする。

(3) コンポーネントウェアの適用評価

コンポーネントウェアは適用が始まったばかりであり、その評価はまだ定まっていない。実践の評価を積む必要がある。しかし、従来のVisual Basicのコントロールを用いた実践の評価が一つの目安になる。また、文献[20]では、C プログラムによる共通問題をOLE, COM, SOM 上で実装を行い、作成プログラム量などを比較している。

1．2．7　今後の動向と課題

(1) 研究・開発の動向

(a) 米国における研究・開発の動向

米国では、コンポーネントウェアをソフトウェア産業の競争力強化策と位置づけて、研究、開発を活発に取り組んでいる[17]。研究面では、NIST（National Institute of Standards Technology）の支援の下に１．５億ドルの予算でコンポーネントウェアに関する５年間の研究プロジェクトが始まった。初年度として、９４年から１１のプロジェクトが４,０００万ドルの予算で始まった。いずれも、実践的な技術開発を目指している点に特徴がある。

一方、産業界でも、Microsoft やCI Labs など実際の商品開発と結びついて活発な開発が行われている。

(b) わが国における研究・開発の動向

わが国では、IntelligentPadなどが開発・商品化されている。しかし、ソフトウェア産業界全体での本分野への取組みは、あまり活発ではない。とくに、この分野では標準化が重要な鍵であることから、研究・開発の先行性や仕様の公開などが重要となる。今後、わが国でも本分野での研究・開発が推進されることを期待したい。

(2) 標準化

コンポーネントウェアでは、標準化は本質的な問題である。現段階では、事実上の標準として、次の２つがある[1]。

(a) CI Labs のOpenDoc とOMG のCORBA

(b) Microsoft のOLE/COM

この２ つのシステム間でのオブジェクト協調のための仕様も検討されている。

さらに、アプリケーション領域にも踏み込んで, 特定業種やドメインごとの部品の標準化も検討されている[8]。業種ごとに部品が標準化されるとアプリケーションを構築する上で飛躍的な生産性向上が望める一方、設計や実装の上で制約となるおそれもあることに留意すべきである。

(3) 部品の流通

インターネットを利用した部品の販売・流通が始まっている。たとえば、９５年１０月から米国のObjectSoft社がWWW を利用してOLE 部品の販売、流通を行うOLE Brokerを始めた。OpenDoc の部品もAppleやIBM のWWW サイトからダウンロード可能である。インターネットによって部品を世界中の利用者に直接販売できる。わが国のソフトウェアベンダにとっても大きなビジネスチャンスである。

(4) 今後の課題

コンポーネントウェアを開発・適用するためには、まだ解決すべき問題も多い。技術面と管理面から、いくつかの課題を挙げよう。

(a) 技術面の課題

(i) 再利用のためのオブジェクトモデルとその方法：とくに、意見が分かれている継承の利用をどうするか。

(ii) コンポーネントウェアによる開発方法論の確立：アプリケーションを部品の組立てによって構築するための方法。

(iii) 大規模基幹アプリケーションへの適用方法：規模の増大にともなう性能の低下や複雑度の増大に対応する方法。

(iv) 部品抽出方法：オブジェクト指向におけるオブジェクトの抽出と同様、部品抽出方法はコンポーネントウェアを実践する鍵となる。

(b) 管理面の課題

(i) コンポーネントウェアによる開発プロセスの確立

(ii) 標準化のためのプロセスの組込み：組織的に部品を抽出し、標準化する規則を開発プロセスの中に組み込む必要がある。

(iii) 構成管理・変更管理：コンポーネントウェアを実現する上で構成管理構成管理・変更管理は必須である。

(iv) 部品とそれを組み込んだ製品の品質保証・品質管理機能が同一であれば、性能や品質が部品の競争力の鍵となる。一方、外部から購入した部品によって構築したシステム全体の品質を保証する方法も開発する必要がある。

(v) 部品の流通方法とその著作権の問題：ネットワークによる部品の流通が自然に行われている。今後は、分散オブジェクト環境の開発によって、分散した部品を利用する技術の開発が必要と思われる。

(vi) 教育：オブジェクト指向の導入でも同様であるが、コンポーネントウェアを使いこなすためには、導入教育が欠かせない。CommonPoint の例では、フレームワークを理解し、使えるようになるには４～５

け月要したとの報告がある[10]。

(vii) プロジェクト管理：コンポーネントウェアでは、従来より開発規模が格段に減る可能性があるので、メトリクスや見積り方法、開発のコストモデルなど、プロジェクト管理のあらゆる面で見直しが必要である。

(viii) 導入のリスク管理

新しい技術の導入にあたっては、導入リスクの評価を行い、その低減を図る必要がある。パイロットプロジェクトによる評価が有効である。

1．2．8　まとめ：コンポーネントウェアの挑戦

ソフトウェア開発の生産性と品質を向上するためには『作らない』ことであると現場では言われてきた。コンポーネントウェアは、『コーディングレス』あるいは『zero-lines-of-code programming』を目指す様々な技術の集成である。

これまでソフトウェアの再利用に関して多くの研究・開発が行われてきたが、技術面、アーキテクチャ支援の面で、再利用を実現する仕組みに欠けていた。コンポーネントウェアは、従来とは異なるアプローチを採り、プラグ& プレイ型ソフトウェア部品による再利用の実現を図る。とくに、クライアント／サーバシステムやウィンドウ上でアプリケーションを開発するためには、コンポーネントウェアが提供する技術が有用である。

コンポーネントウェアによってアプリケーションの姿が大きく変わろうとしている。従来、単一（モノリシック）システムとして提供さていたアプリケーションがより小さなアプリケーションの集まりとして提供されるであろう。さらに、これらのアプリケーションはネットワークの上に分散しているかもしれない。

さらに、コンポーネントウェアは一からコーディングをする従来型ソフトウェア開発方法に対する根本的な見直しを迫るだろう。従来型開発方法では、ちょっとした機能のソフトウェアでも千万円単位のコストがかかることは珍しくない。ソフトウェアがハードウェアのおまけではなく、独立した産業として成立するためには、他産業なみに生産性を高める必要がある。とくに、ハードウェア価格の急激な低下によりソフトウェアの価格は相対的に高くなっている。ソフトウェア産業が『手工業的生産』から『近代的工業』へ脱皮するためには、標準部品の提供と、部品を組み合わせてシステムを構築する方法の開発が必要である。コンポーネントウェアは部品組立によるソフトウェア開発を実現する具体的なソフトウェアアーキテクチャを提供し、ソフトウェアの産業革命を促すと言ってよい。

さらに、産業界全体でソフトウェアの再利用が普及するためには、再利用がビジネスとして成立する必

要がある。コンポーネントウェアによって、ソフトウェア部品市場が出現した。ソフトウェアの産業構造にも変化の兆しが見える。コンポーネントウェアは、ソフトウェア産業が部品開発産業（コンポーネントベンダ）と部品組立て産業（アプリケーションベンダ）に分化する起爆剤となるかもしれない。

このように、コンポーネントウェアは今後のソフトウェア開発のあり方を根本的に変える可能性を秘めた技術である。

【参考文献】

1) R. M. Adler: Emerging Standards for Component Software, IEEE Computer, Vol. 28, No. 3, pp. 68-77 (1995).

2) 青山幹雄：コンポーネントと協調オブジェクトの動向, オブジェクト指向 '95シンポジウム論文集, pp. 27-37 (1995).

3) 青山幹雄：コンポーネントウェア：部品組立て型ソフトウェア開発技術, 情報処理, Vol. 37, No.1, pp. 71-79 (1996).

4) 青山幹雄：部品組立て型ソフトウェアプロセスモデル、ソフトウェアの新しい構成・統合技術に関するワークショップ論文集、ソフトウェア科学会 (1996).

5) K. Brockschmidt: Inside OLE, 2nd ed., Microsoft Press (1995)[初版は翻訳あり：エー・ピーラボ／長尾高弘(訳): Inside OLE2, アスキー出版局 (1995)].

6) 富士通, 日立ソフトウェアエンジニアリング：インテリジェントパッド (1995).

7) J. Gosling and H. McGilton: The Java Language Environment, Sun Microsystems (1995).

8) 星野友彦：コンポーネントウェアが来た, 日経コンピュータ, 1995年８月７日号, pp. 86-97.

9) 井上望, 小川弘晃： Windows上の開発ツール徹底活用：多様なソフト部品を再利用し連携させる, 日経オープンシステム, No. 20, pp. 152-187 (1994).

10) W. Myers, Taligent's CommonPoint: The Promise of Objects, IEEE Computer, Vol. 28, No. 3, pp. 78-83 (1995).

11) 長崎祥, 田中譲：シンセティック・メディアシステム: IntelligentPad, コンピュータソフトウェア, Vol. 11, No. 1, pp. 36-48 (1994).

12) 新居雅行：OpenDoc GENESIS, ソフトバンク, 1996.

13) OMG, CORBA 2.0/Interoperability, 1995.

14) R. Orfali, et al.: The Essential Distributed Objects Survival Guide, John Wiley & Sons (1995).

15) D. Pountain and C. Szyperski: Extensible Software Systems, Byte, Vol. 19, No. 5, pp. 57-62 (1994).

16) 佐藤広行, 大野邦夫：共通オブジェクトリクエストブローカアーキテクチャCORBA 1.1, 情報処理,

Vol. 35, No. 9, pp. 853-858 (1994).

17) D. Sims: US Funds Large R&D Program on Reuse, IEEE Software, Vol. 11, No. 4, pp. 107-108 (1994).

18) Taligent: The Power of Frameworks, Addison-Wesley, 1995.

19) J. Udell: Componentware, Byte, Vol. 19, No. 5, pp. 46-56 (1994).

20) R. Valdes, et al.: The Interoperable Objects Revolution, Dr. Dobb's Special Report, Vol. 19, No. 16 (1994).

1．3　コンポーネントウェアを活用したシステム構築事例

1．3．1　コンポーネントウェア開発の現状

コンポーネントウェアを使った典型的な開発では、情報システム部門でソフト部品を使ったプロトタイプ開発を行い、ユーザ部門はそれをカスタマイズすると言う業務分担である。しかし、一般的な企業では、まだ、エンドユーザがカスタマイズを行なえるレベルに達しておらず、コンポーネントウェア開発は、情報システム部門の生産性向上の施策として取り入れられている段階と認識される。しかし、ビー・エム・ダブリュー（BMW）などの特定の企業では、コンポーネントウェアを使ったエンドユーザ主導の開発が徐々に実現してきている。

コンポーネントウェアを用いた開発プロセスは、従来のような、ウォータフォール型開発ではなく、プロトタイプによるスパイラル型開発によって、ユーザー要求仕様を早い段階で確認することができ、リリース後の仕様の食い違いよる手戻りの回避が可能になる。また、ビジュアル開発ツールによる画面設計／開発の効率化と合せ高い生産性が実現できる。一般的なコンポーネントウェア開発プロセスは、要求仕様分析・基本設計・部品設計・プロトタイプ構築・開発・テストとなる。コンポーネントウェア開発の特徴は、「部品設計」である。この部品設計は、Visual Basicのカスタムコントロール（以後、VBXと呼ぶ）など流通ソフト部品の調査・受け入れ試験・調整及び自製ソフト部品の再利用あるいは新規開発を行うプロセスである。このプロセスにおいて如何に要求仕様に合致する高品質の部品を一般流通市場及び自社内で効率的に調達できるかが重要な課題である。特に、サードパーティが提供するさまざまなソフト部品を最大限に利用することと自社内における共通ソフト部品の開発と再利用の促進が肝要である。後述する事例においても、部品設計段階の成否が全体開発生産性の高低に大きく起因することが確認されている。逆に、流通ソフト部品が存在しても、パラメータ設定に手間取ったり、品質が必ずしも良くなかった場合、大きな業務負荷となり、全体の開発生産性を悪化させる事態となる。従来、ウォータフォール型開発では、詳細設計・プログラム製作が全体の業務負荷の５０％を占めていたが、コンポーネントウェア開発では、「部品設計」の業務負荷が開発全体の大きな部分を占めることになろう。

1．3．2　コンポーネントウェア開発事例

コンポーネントウェア開発事例を紹介し、開発生産性や開発プロセスについて考察する。但し、分析できる事例が現状では少ないため、一般化できるレベルには至っていない。ここに示す事例において、開発生産性を議論する場合のシステム規模は、画面数・帳票数及びステップ数を使用する。ここで、ステップ数は、画面オブジェクト数、プロパティ定義数及びVBステップ数の総計で表現した。

(1) 事例１－FAXオンデマンドシステム－

(a) 概要

FAXオンデマンドシステムは、遠隔地からの要求によってサーバーに登録されたイメージデータをFAXで送信するシステムである。機器構成及びシステム構成を図1.3-1に示す。図中のイメージサーバーは、イメージスキャナーからドキュメントを読み取りDBへ格納する。FAXサーバーは、FAX配送要求を受信し、イメージサーバーのDBを検索して、所定の電話番号にFAXを送信する。

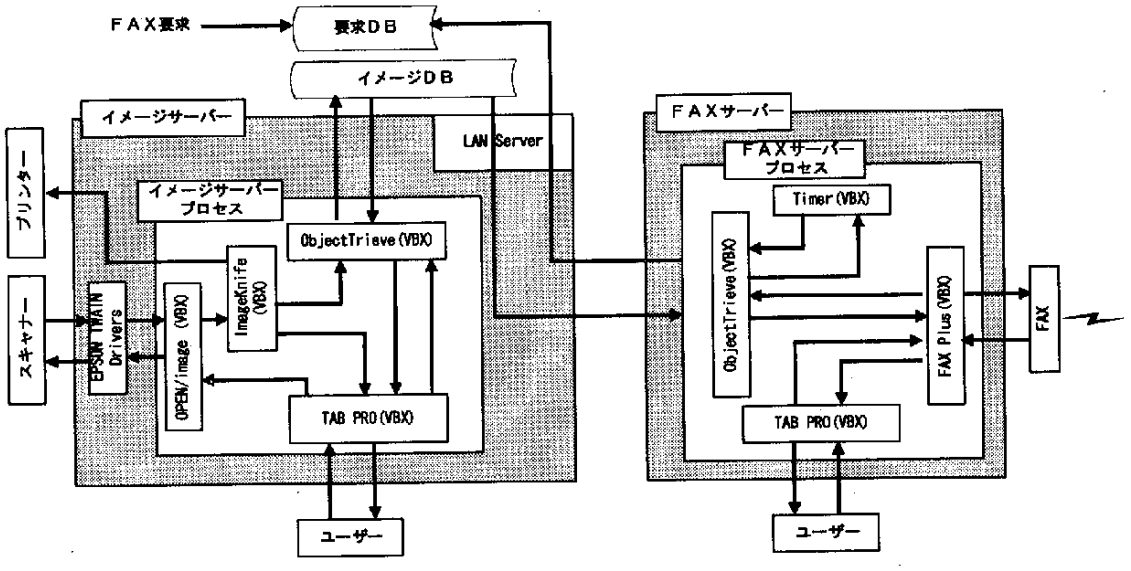


図1.3-1　FAXオンデマンドシステムの構造

(b) 開発結果

本システムの開発は、通信系及びDB系の機能はすべてVBXを使用し、その他のアプリケーションは、VBで開発することとした。使用したVBXは、以下の５種類である。

・イメージデータ処理　：ImageKnife
・DB機能　：ObjectBtrieve
・スキャナ読み取り　：OPEN/Image
・FAX送信処理　：FAXPlus
・GUI　：TAB/PRO

本システムは、鉄鋼業薄板物流における製品不具合処置方法の連絡手段として開発され、システム規模は、４画面、２４７９ステップとなった。

このシステムの開発には、中堅のSE（経験１０年）が合計４５人日を要した。そのフェーズ別の開発日数は、基本設計：４人日、部品開発：２０人日、プロトタイプ開発・本開発：２０人日、テスト：１人日

である。

(c) 分析

(i) このシステムの開発では、要求仕様が既に明確に決定されており、要求仕様分析に要する負荷はほとんどなかった反面、部品設計に全体の約半分のマンパワーを要した。これは、FAXやイメージスキャナーを制御するコンポーネントウェアのパラメータ設定・調整に多くの時間を要したことが原因である。

(ii) コンポーネントウェア利用による大幅な生産性向上が図れた。特に、FAX送信部の開発をC言語による開発と比較すると、開発工期で１０分の１（２０日が２日）、開発量で５分の１（１０００ステップが２００ステップ）となった。

(2) 事例２－出張管理システム－

(a) 概要

出張管理システムは、担当者の出張申請・上司の決済・経理部門の承認と言う一連の出張業務を支援するワークフロー的なシステムである。また、交通費の自動計算や出張関連情報を一元管理し部門別出張費用

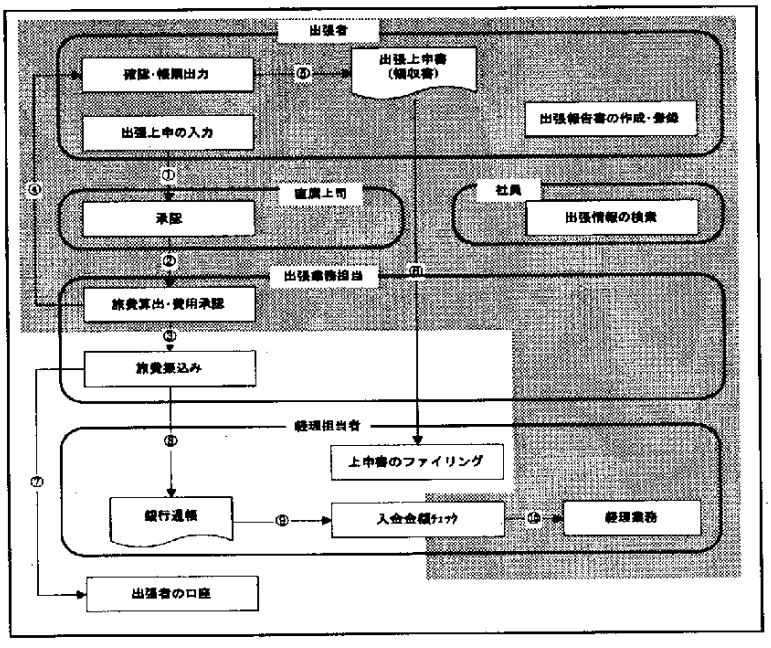


図1.3-2　出張管理業務フローとシステム化範囲

の予算対実績管理などの機能も装備する。図1.3-2に今回開発したシステムの対象範囲を示す。

(b) 開発結果

本システムの開発では、ユーザ部門、ここでは経理部門との仕様検討を経て、ソフト部品やパッケージの選定を行った。図1.3-3にソフトウェア・機器構成を示す。各パッケージ及びツール間の連携には、OLE

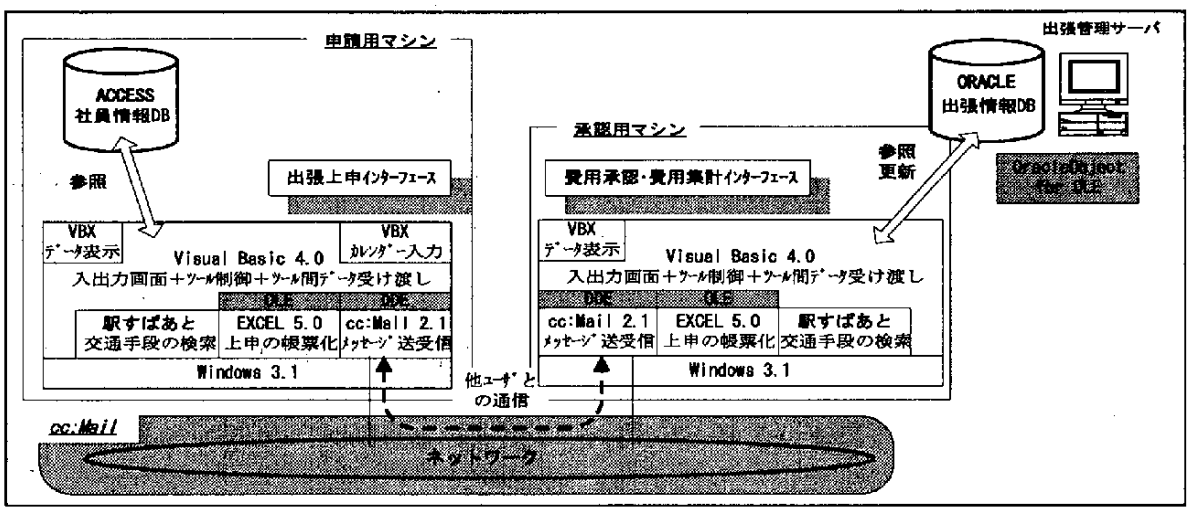


図1.3-3　出張管理システムの構造

（Object Linking and Embedding）及びDDL（Dynamic Data Exchange）を活用した。また、全国対象の交通費計算については、機能の充実度及び簡単に入手出来る事などの理由から、株式会社ヴァル研究所の「駅すぱあと」を採用した。使用したVBX及びパッケージを以下に示す。

・GUI関連　　：Spred*20,Calendar

・DB関連　　：VB Jet,Oracleo bjects for OLE

・交通費計算　　：駅すぱあと（ヴァル研究所）

本システム規模は、８画面、２帳票、１８３３ステップとなった。このシステムの開発には、初級のSEが合計６３人日を要した。そのフェーズ別の負荷は、要求仕様分析：２０人日、部品開発：８人日、基本設計：１１人日、プロトタイプ開発・本開発：２２人日、テスト：２人日である。

(c) 分析

(i) VBXコンポーネントを利用してユーザインタフェースを開発したことによって、VBのみで開発した場合に比較して、開発工期は３分の１、開発ステップ数は半分で実現できた。

(ii) DDE及びOLEを活用した場合とC言語によるプログラミングとを比較すると、メッセージングシステムの部分では、DDEとcc:Mailを連携すると、従来１０週間必要としたプログラミングがわずか１～２日程度で実現出来た。また、EXCELとOLEを連携した帳票システムの場合、従来２ヶ月必要としたものが１日程度で実現出来る様になった。

(iii) プロトタイプ開発で業務部門と仕様確認を行うことによって、ユーザの要求仕様分析結果との不整合を抽出でき、手戻り防止に大きく貢献することが確認できた。これは、常識的な結論ではあるが、特に、コンポーネントウェア開発の場合、レスポンスやユーザインタフェースの仕様上に不整合があった場合、

再度、ソフト部品調査から行う必要があり、プロトタイプ開発は生産性向上の極めて重要な開発プロセスであることが認識される。

1．3．3　コンポーネントウェア開発の課題

ここに示した事例以外にもコンポーネントウェア開発事例は増加してきており、今後、事例の積み重ねによって知見の蓄積を図る必要がある。今回示した事例から得られた今後の課題を述べる。

(i) コンポーネントウェア開発では、部品設計・調査の負荷を如何に軽減できるかが重要な課題であることは、事例紹介の中でも述べた。コンポーネントの調査や動作確認の効率化を推進するために、知見の共有化を企業単位あるいは特定のグループ毎に行っていく必要がある。

(ii) コンポーネントウェア開発プロセスの確立は、コンポーネントウェアの適用が広がるに従って、重要な課題となってくる。特に、ユーザとシステムインテグレータとの業務分担や契約形態及びソフト部品の品質保証、プロジェクト管理など中大規模システムや基幹業務への適用が進展するに従って開発プロセス上多くの課題が顕在化してくる可能性がある。

(iii) また、開発規模の予測や開発生産性の設定など見積り技術の早期確立が望まれる。1．3．2項の冒頭でも述べた様に、今回の事例の開発規模も画面・帳票・ステップ数で便宜的に表現したが、実務的には、標準化された見積り技法の早期確立が必要である。

(iv) 今回の2つの事例から、部品設計・調査を含めて、2～3画面／人・月＋600～1200ステップ／人・月程度の開発生産性が実現できた。さらに、生産性の向上を図るには、2章で紹介される「新しいアプリケーション構築環境」の導入・定着とEUC化が必要である。

【参考文献】

1) 星野友彦：コンポーネントウェアが来た, 日経コンピュータ, 1995年8月7日号,pp86-97.

2．　新しいアプリケーション構築環境

2．新しいアプリケーション構築環境

2．1　APPGALLERY

APPGALLERYは、エンドユーザでもアプリケーション・プログラムを組み上げることができるような環境を提供するために開発されたソフトウェア開発ツールである。エンドユーザとプログラマとが共同してアプリケーション・プログラムを開発する環境を想定している。APPGALLERYは、２つのユーザ・インタフェースをもつ。１つはエンドユーザが部品を組み立ててアプリケーション・プログラムを構築するためのビジュアル環境で、もう１つはプログラマが部品を作ったりエンドユーザ用の環境を拡張したりするための環境である。

2．1．1　背景

(1) 解決しようとしている問題

ソフトウェア開発において解決されていない問題として、エンドユーザとプログラマの間のコミュケーション・ギャップがあげられる。プログラマは、エンドユーザとコミュニケートして彼らの要求を分析し、プログラムの仕様を規定しなければならない。エンドユーザは、自分たちの業務の専門用語は使うのに、プログラマが使うコンピュータの専門用語を理解できない。このギャップは、プログラマはどんなソフトウェアが必要とされているかを知らず、エンドユーザは自分たちが必要としているプログラムの作り方を知らないというジレンマと解釈することもできる。

(2) アプローチ

この問題を解決するために多くの試行が繰り返されてきた。スプレッドシートなどのエンドユーザ向き計算ツールや、ビジネス分野向きのクラスライブラリ[1]などがあげられる。しかし、いままで提案されてきたもののなかには、ここで提案するような、プログラマとエンドユーザととインタフェースをとり、共同作業を支援するような開発環境は見られない。Martin [2] はエンドユーザとプログラマとの共同開発形態であるJAD（Joint Application Development）を提唱してるが、この中でもエンドユーザとプログラマとが実現フェーズにおいても共同作業するということには言及されていない。

(3) APPGALLERYの役割

APPGALLERYは、２つのインタフェースをもつプログラム開発環境である。１つめのインタフェースは、エンドユーザが部品を組み立ててアプリケーション・プログラムを作るための使うインタフェースである。もう１つのインタフェースは、プログラマが、部品を作ったり、エンドユーザ用のインタフェース

を拡張するためのものである。エンドユーザは、基本的には、APPGALLERYが提供している部品や、購入してきた部品を組み立ててアプリケーション・プログラムを構築する。プログラマは、エンドユーザにもアプリケーション・プログラムが組み立てられるように、部品を用意したり、部品間をつなぐためのツール（APPGALLERYでは「スタッフ」と呼んでいる）を用意する。この２種類の作業を実現する環境をAPPGALLERYは提供する。

(4) APPGALLERYが提唱する開発スタイル

APPGALLERYというツールが想定している開発スタイルをEUCAP（End-User Customizable Application Program）と呼んでいる。プログラマは、エンドユーザがアプリケーション･プログラムを作るための部品とツールを作る。エンドユーザは、プログラマが作った部品やツール、APPGALLERYが基本的に提供している部品やツールを使って、アプリケーション･プログラムをビジュアルに組み立てる。

(5) もうひとつの背景 …ソフトウェア部品の規格化

APPGALLERYのようなツールが登場した背景には、OLE2（Object Linking and Embedding 2）やVBX（Visual Basic Extension）に代表されるように、ソフトウェア部品の規格化が進んできたということがあげられる。ソフトウェアの部品化については、２０年ほど前から研究が進められてきたが、特定の組織内での再利用の事例しか報告されていなかった。しかし、ここへ来て、VBXというソフトウェア部品の規格と作り方が公開され、サードベンダーを中心としたVBX部品の市場が形成されるに至った。OLE2部品もこれに続いている。APPGALLERYでは、部品の規格として、OLE2部品を採用している。OLE2オートメーション機能をサポートしている部品（OCX（OLE Custom Control）を含む）は、APPGALLERYの部品として機能する。

２．１．２　構成要素

(1) エンドユーザ向け開発環境

(a) キャンバス

キャンバスは、部品を組み立てるための土台である。図2.1-１に示すように、キャンバスの上に部品を乗せ、コネクタで組み立てる。「キャンバスに部品を持ってきて、コネクタでつなぐ」というのが、APPGALLERYにおけるプログラミング作業である。（図2.1-1, 図2.1-2, 図2.1-3, 図2.1-4）部品は、OLE2オートメーションのインタフェースを備えている必要がある。したがって、部品と外部とのインタフェースは、通常のOLE2オートメーション部品と同様に、プロパティ、メソッドとで構成される。プロパティは、オブジェクト指向で言うところの、スロット、メンバ、アトリビュートと同様のもので、部品内のデータを格納するための領域である。メソッドは外部から呼び出すことができるプログラムである。

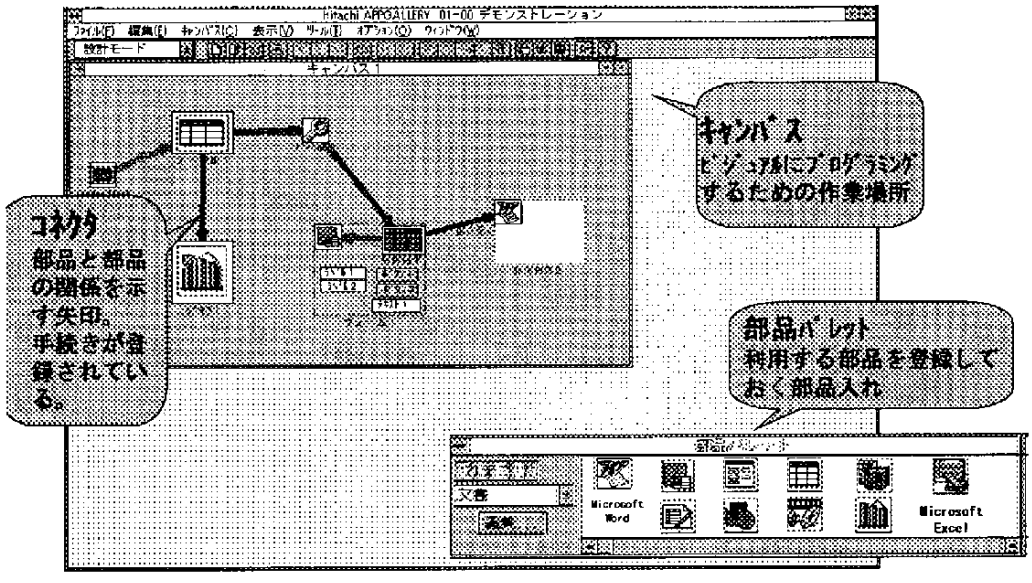


図2.1-1　APPGALLERYのプログラミング

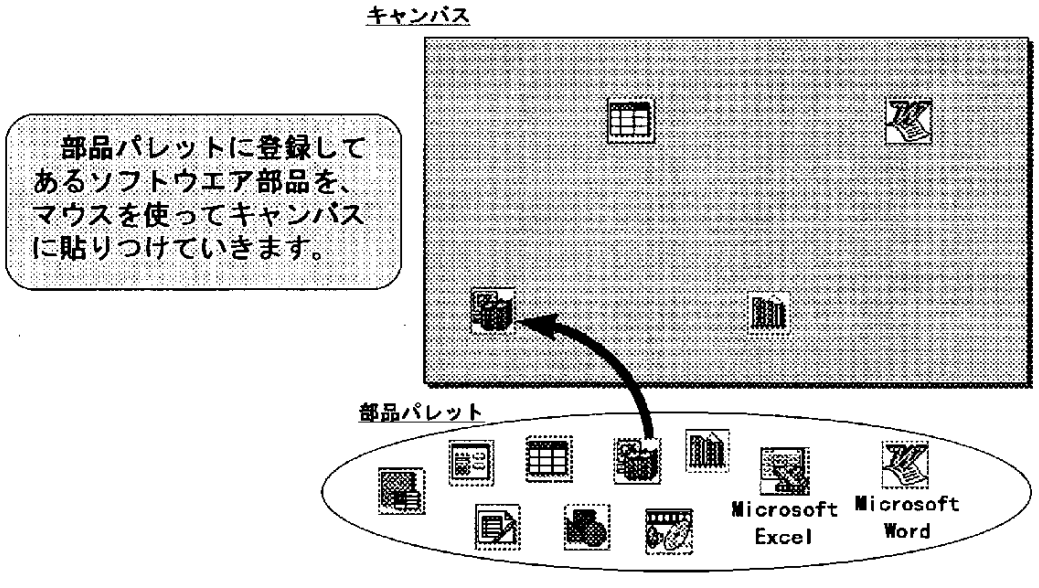


図2.1-2　アプリケーションの開発－1

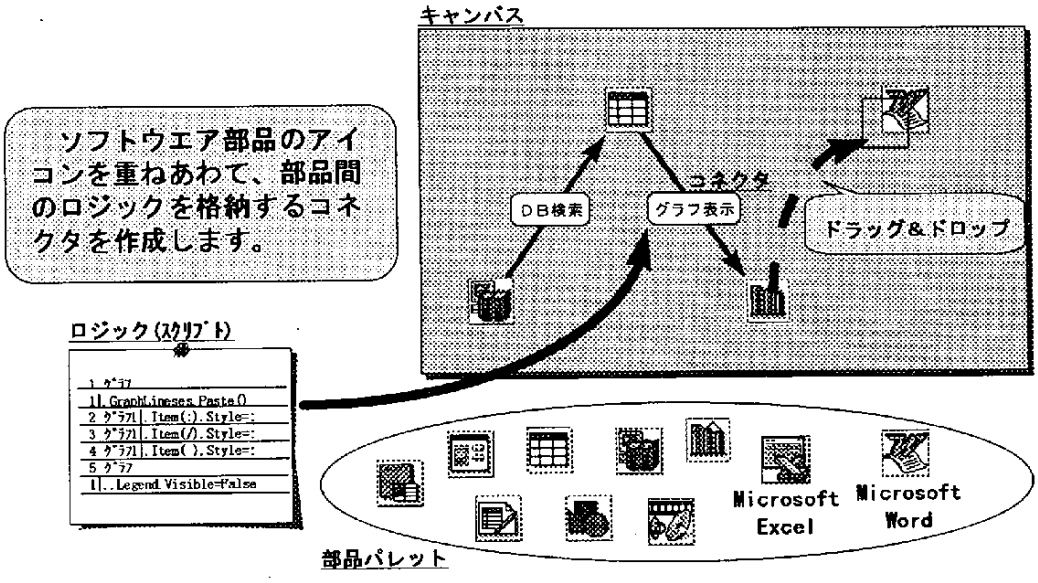


図2.1-3　アプリケーションの開発ー2

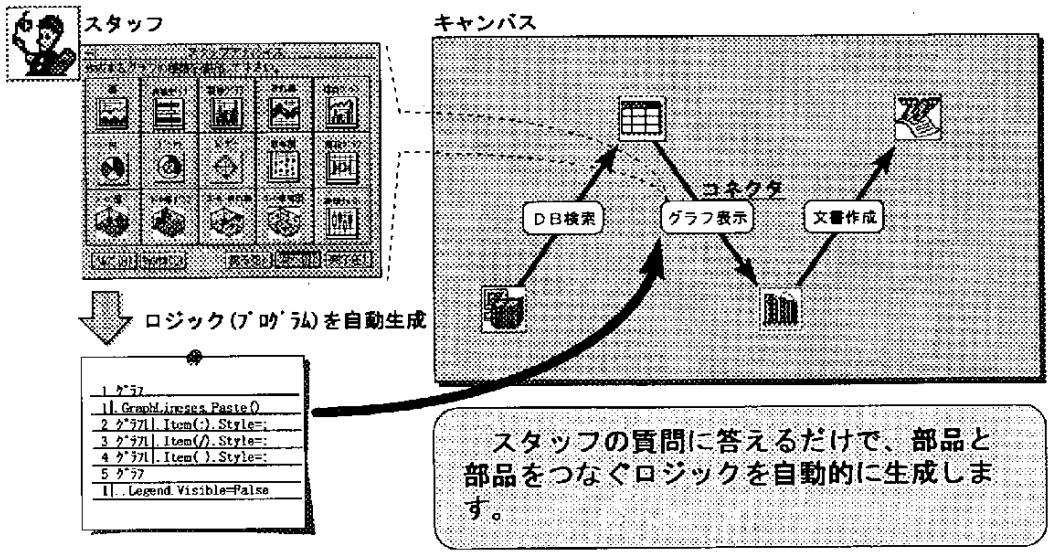


図2.1-4　アプリケーションの開発ー4

(b) コネクタ

コネクタは、部品と部品とをつなぐ矢印としてキャンバス上に表示される。実体は、矢印でつながれた部品のプロパティやメソッドをアクセスするプログラムである。このプログラムを特にコネクタ・プロシージャと呼ぶ。コネクタ・プロシージャは、APPGALLERY BASIC言語で記述される。このAPPGALLERY BASIC言語は、ANSI規格のBASIC言語のサブセットに独自の記述方法を追加したものである。

(c) 部品パレット

キャンバスに部品を持ってくるための部品庫として働くのが、部品パレットである。部品パレットには、APPGALLERYが提供する部品のほかに、ユーザが登録した市販のOLE2部品が並べられている。ここからドラッグ・アンド・ドロップの操作で、キャンバス上に部品を乗せる。市販の部品を部品パレットに登録するには、「部品のインポート」というメニューで実行する。

(d) スタッフ…部品間をつなぐツール

次に、部品間をコネクタでつなぐ操作が必要である。2つの部品をドラッグ・アンド・ドロップで重ねると、その部品間にコネクタが生成される。生成されたコネクタには、コネクタ・プロシージャが含まれていない。つまり空のコネクタが生成されただけである。コネクタ・プロシージャを生成するためのツールが、スタッフと呼ばれるものである。コネクタをマウスで選択して、メニューから「スタッフの起動」を選ぶと、選択されているコネクタがつないでいる部品の種類を見て、該当する部品の組み合わせに対応するスタッフを検索する。スタッフが存在すれば、そのスタッフ一覧を表示し、エンドユーザに選択して起動してもらう。スタッフは、通常は対話型プログラムであり、ユーザにどのようなつなぎかたをするのかについてのオプションを問い合わせる。スタッフは、ユーザが選択したオプションに応じたコネクタ・プロシージャをBASICプログラムとして生成する。テーブル部品とグラフ部品とでスタッフの例をあげる。テーブル部品とグラフ部品とをつなぐコネクタに対して「スタッフを起動」というメニューを選択すると、テーブル部品とグラフ部品を関連付ける典型的な「テーブルのデータをグラフ化する」コネクタ・プロシージャを生成するスタッフがメニューに現れる。これを選択すると当該スタッフが起動される。このスタッフは、グラフのタイプを選択する画面、テーブルのデータを縦方向に読むのか横方向に読むのかを指定する画面などをもっている。ユーザがこれらのオプションを指定すると、スタッフはテーブルのデータからグラフを作成するコネクタ・プロシージャを生成する。これにより、BASIC言語でプログラミングしないエンドユーザでも部品間の連係動作を定義することができ、アプリケーション・システムを構築することが可能になる。他には、画面遷移プログラムを生成するスタッフとデータベースのレコードデータをフォーム上でカード型データベース風にアクセスするプログラムを生成するスタッフとが説明されている。（図2.1-5, 図2.1-6）空のコネクタを生成してからスタッフを起動するという冗長な操作を避けるために、「ス

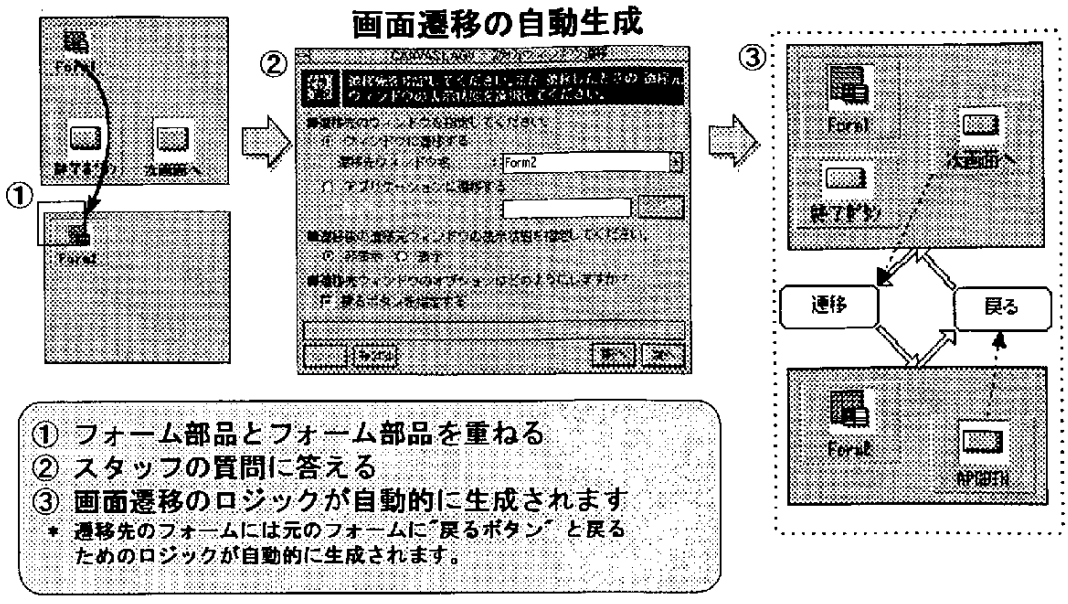


図2.1-5　スタッフの例（1）画面遷移

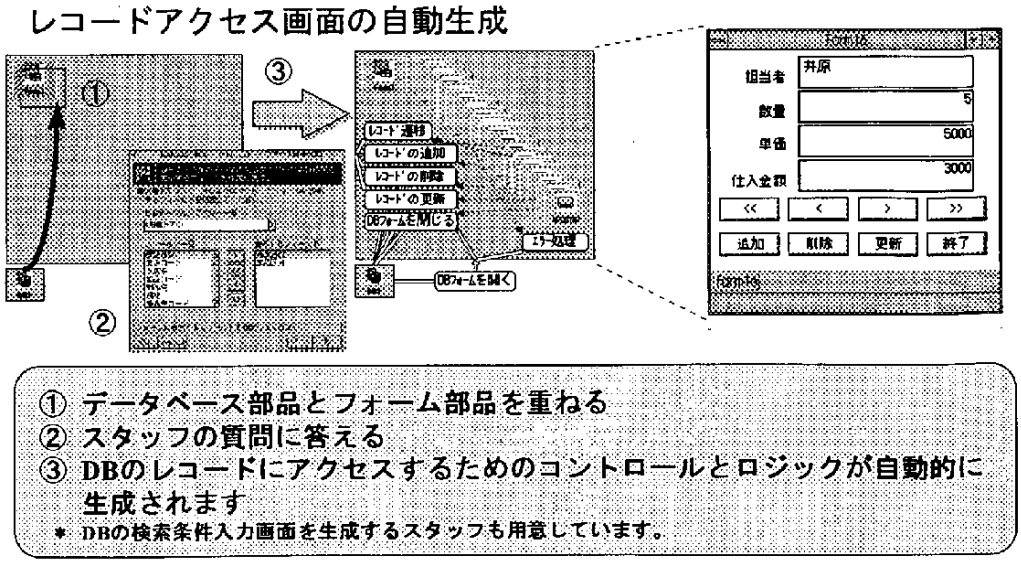


図2.1-6　スタッフの例（2）レコードアクセス画面

タッフの自動起動」というスイッチがある。このスイッチをオンにすると、部品を部品の上にドラッグ・アンド・ドロップするとコネクタ生成と同時にスタッフが起動される。

(2) プログラマ向け開発環境

(a) 部品ビルダ

プログラマが部品を作る場合にはいろいろなケースが考えられる。たとえば、APPGALLERYで提供している部品では汎用的すぎて、もっと業務よりの部品でないとエンドユーザがアプリケーション・システムを組み立てることができないケース。あるいは、汎用的な部品のインタフェースをより限定して、取り扱いやすい部品にするケース。また、エンドユーザにアクセスを許すデータを限定したいとき、自由なアクセスを禁止する部品を作るようなケースなどが考えられる。APPGALLERYの部品ビルダ（図2.1-7）は、OLE2オートメーション部品を作る機能を提供する。部品を構成するのは、プロパティ、メソッド、イベントである。

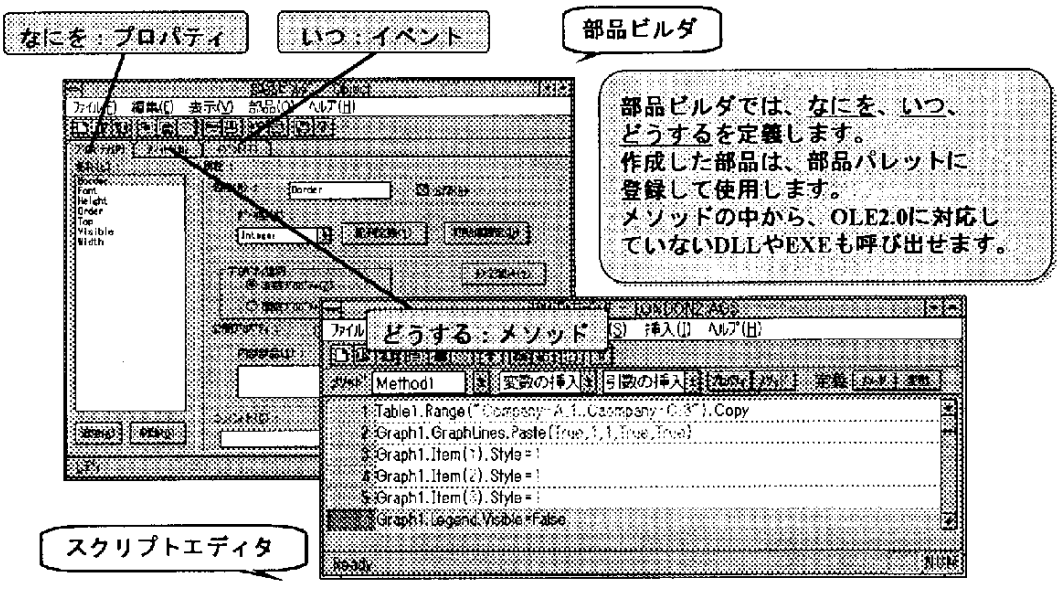


図2.1-7　部品ビルダ

プロパティは、オブジェクト指向で言うところの、スロット、メンバ、アトリビュートと同様のもので、部品内のデータを格納するための領域である。部品ビルダでは、プロパティの定義、データタイプ指定などをするための機能を提供している。イベントは、外部に対して発行するもので、イベントによって他の部品のメソッドが起動されたりする。部品ビルダは、各部品が発行するイベントを登録する機能を持っている。メソッドは、外部から呼び出すことができるプログラムである。部品ビルダでは、BASIC言語でメソッドを記述するための機能を提供している。部品ビルダからメソッド定義を実行すると、スクリプト・

エディタが呼び出される。スクリプト・エディタは、BASIC言語のキーワードなどを認識して色分けして表示する機能を持つエディタである。プログラマは、このスクリプト・エディタを使って、BASIC言語でメソッドを記述する。部品ビルダで作成した部品を部品パレットに登録すると、キャンバスに貼り付けられるようになる。部品は、キャンバス上に貼り付けられコネクタからアクセスされるほか、さらに大きな部品に包含され呼び出されることも可能である。複数の部品を含んだキャンバス全体を部品化することも可能である。

(b) 外部部品のインポート

APPGALLERYで部品として取り扱えるのは、OCXを含むOLE2オートメーション部品すべてである。したがって、部品の入手方法としては、以下の4つが考えられる。

(i) APPGALLERYに組み込まれている部品

(ii) APPGALLERYの部品ビルダで作成した部品

(iii) 市場で販売されてるシュリンクラップ・ソフトウェアで、OLE2オートメーション機能をサポートしているもの

(iv) 他の開発環境を使って作成したOLE2オートメーションをサポートしているソフトウェア

ここで外部部品と言っているのは、APPGALLERY以外から入手した下の2つ（(iii)(iv)）を指す。

(c) スタッフ・マネージャ

APPGALLERYが提供している部品用のスタッフは、いくつか組み込まれている。たとえば、例としてあげた表データからグラフを作成するスタッフは組み込まれている。しかし、ユーザが作成した部品（上記(ii)(iv)）、購入した部品（上記(iii)）については、スタッフを作らないと、BASICをプログラミングしないエンドユーザ用の環境としては整わない。また、APPGALLERYが提供している部品についても、用途が特定できたり、スタッフマネージャは、ユーザスタッフを作成するためのツールである。スタッフマネージャを開くと、作成するスタッフで関連付ける部品を選ぶ。

(d) キャンバス…スタッフ作成

スタッフマネージャで、関連付ける部品を選ぶと、スタッフマネージャは、キャンバスを立ち上げる。このキャンバスはスタッフ作成用のもので、初期画面に先ほどユーザが選択した関連付ける部品を表現するアイコンがならべられている。上記以外は、通常のキャンバスと同様である。スタッフ作成作業はプログラマが行うと想定しているため、キャンバス上の部品をつなぐコネクタのコネクタプロシージャは、APPGALLERY　BASIC言語で記述する。

(e) キャンバス…アプリケーション作成

以上、アプリケーション開発そのものはエンドユーザが行い、そのための環境づくりをプログラマが行

うというシナリオで、APPGALLERYの説明を行ってきた。実際には、プログラマがAPPGALLERYを使って、直接アプリケーション・システムを開発する可能性が高い。理由の一つは、エンドユーザ自身がプログラマを作るべきだという認識が共通認識にまでは至っていないからである。また、仮にエンドユーザにアプリケーション・システムを構築してもらう体制ができていたとしても、プログラマがアプリケーション・システムを開発する可能性は多く残る。たとえば、アプリケーション・システムの例題をプログラマが開発し、エンドユーザはこの例題アプリケーション・システムを修正する開発スタイルは、効果を発揮すると考えられる。

プログラマがキャンバスを使ってプログラミングするのは、スタッフ作成と同様である。すなわち、部品をキャンバスにコピーし、部品間をコネクタでつなぎ、コネクタを開いてAPPGALLERY BASIC言語でプログラムを記述する。

(3) APPGALLERY提供部品

APPGALLERYでは、ビジネス・アプリケーション・システムを構築するための部品群をバンドルして提供している。

(a) フォーム／コントロール部品

フォーム部品は、OCXコンテナとして実現されているウィンドウである。フォーム部品の上に、OCXとして実現されているコントロール部品を乗せて画面を作る。フォーム部品は、キャンバス上で、大きいアイコンとして表示される。フォーム部品のアイコンの上にコントロールなどのアイコンを乗せることができる。コントロール部品としては、ボタン、リスト・ボックス、テキスト・ボックスなど代表的なものが組み込まれている。フォーム部品を作るためのエディタは、フォーム・エディタである。フォーム・エディタは、いわゆるＧＵＩエディタである。キャンバス上のフォーム・アイコンをダブル・クリックするとフォーム・エディタが起動される。フォーム部品に関係するスタッフとしては、フォーム間の画面遷移を実現するスタッフが組み込まれている。どちらのフォームが遷移元・遷移先か、遷移後は遷移元フォームは表示するのかしないのか、などのオプションをもっており、画面遷移に必要となるボタン類を生成する。

(b) テーブル部品

テーブル部品は、データベースやスプレッド・シートで用いられる表形式のデータを表現するための部品である。計算式を持っている。データベース・アクセス部品との間ではデータ検索のためのスタッフが用意されている。どのデータベースから検索するか、どういう項目を検索するのかなどのオプションを持っている。グラフ部品との間では、テーブルのデータをグラフ化するスタッフが用意されている。棒グラフ・円グラフなどのグラフの種類や、テーブルのデータを縦方向に読むか横方向に読むかなどのオプションが

ある。

(c) グラフ部品

棒グラフ、折れ線グラフ、円グラフなど、3次元グラフを含め約15種類のグラフを作るための部品である。テーブル部品との間に、テーブルのデータをグラフ化するためのスタッフが提供されている。

(d) レポート部品

ＲＤＢに格納されているデータは、リレーショナル形式なので、横に属性の列がくる表形式になることが多い。属性のたくさんあるデータをそのままプリントしようと思うと、縦も横もページからはみ出してしまうことになり、プリント結果はとても読みにくいものになってしまう。データベースのレポート・ツールはこの種の問題を解決するものである。データベースの表形式データを、ページ上にレイアウトして、ユーザにとって見やすい形の帳票形式に変換するものである。APPGALLERYのレポート部品は、通常のレポート部品と大きく異なり、「例題ベースで帳票フォーマットを作ることができる」という特徴をもっている。リレーショナル形式のテーブルの１レコードを例として選び、そのデータを画面上に配置していくと、その例を元に他のレコードを配置するという仕掛けである。この結果、同じ形式のテーブルを入力するとそのテーブルのデータを配置印刷するプログラムができあがる。対象を限定してはいるものの、一種の例題ベースプログラミング（Programming by Example）を実現している。

テーブル部品との間に、テーブルのデータをレポート化するためのスタッフが用意されている。

(e) データベース・アクセス部品

データベース・アクセス部品は、ODBC（Open Database Connectivity）をサポートしているデータベースとのアクセスを実現する部品である。データベース・アクセス部品をダブル・クリックすると、クエリー・エディタが立ち上がる。クエリー・エディタは、データベース検索命令をSQL（System Query Language）で生成するビジュアル環境である。データベースのフィールド一覧が表示され、検索対象フィールド、フィールド間ジョイン関係などの検索条件ををビジュアルに指定すると、ODBC準拠のSQLを生成する。データベース・アクセス部品には、上でのべたテーブルとのスタッフのほか、データベースの項目からカード型ＤＢと同じ機能を提供するフォームを生成するスタッフが提供されている。

(f) マルチメディア部品

APPGALLERYとは別売のものだが、マルチメディア部品がある。これは、ファイルに格納された動画・静止画・音声を取り込んで、表示・演奏などを行うことができる。スタッフとしては、テキスト・ボックスのファイル名に格納されているコンテンツをプレイするためのスタッフと、画面切り替えイフェクト（カーテン・オープン、ブラインド・オープン、ワイプアウトなど）や切り替え時間などを変更するためのフォームを作成するスタッフが組み込まれている。

2．1．3　適用事例…保険会社の契約変更システム

ある保険会社の契約変更システムにAPPGALLERYが適用された事例がある。保険業の契約変更業務には、複雑な処理手順（チェック、保険料計算、データの転記、帳票作成などの作業フロー）がある。また、この種の業務処理手順は変更される可能性が高く、開発効率向上とともに、拡張性、保守性について考慮する必要がある。APPGALLERY採用により、処理手順がビジュアルに定義できるため、業務処理手順全体を見渡すことができ、また、開発効率、拡張性、保守性の向上が見込まれる。適用業務としては、本部・契約変更課における保険契約内容の変更などの、オンライン処理できない複雑な案件を対象としている。システム全体は、ワークフローシステムであるFlowmateを中心とする。APPGALLERYからは、OLE2インタフェースをもつFlowmate部品を経由してワークフローの中に流れている案件オブジェクトをアクセスする。保険契約に関係する計算は、EXCELをOLE2オートメーション経由で計算サーバとして利用している。APPGALLERYを利用して、契約変更に関係する４４業務に対して、２４３個のアプリケーション・システムを開発した。各アプリケーション・システムで１つずつのキャンバスを利用しているため、計２４３キャンバス、ステップ数は９７kステップに及んだ。しかし、このうち９５％（１kステップを除く大半）は、スタッフを用いて生成した。スタッフとしては２４kステップをユーザスタッフとして作成した。このほかに１９個のユーザ部品４kステップを作成した。APPGALLERYを適用した結果、アプリケーション・システムの９５％以上がノンプログラミングで開発でき、工数の大幅削減を実現した。処理手順をビジュアルに定義できるため、短期間で仕様を確立することができた。

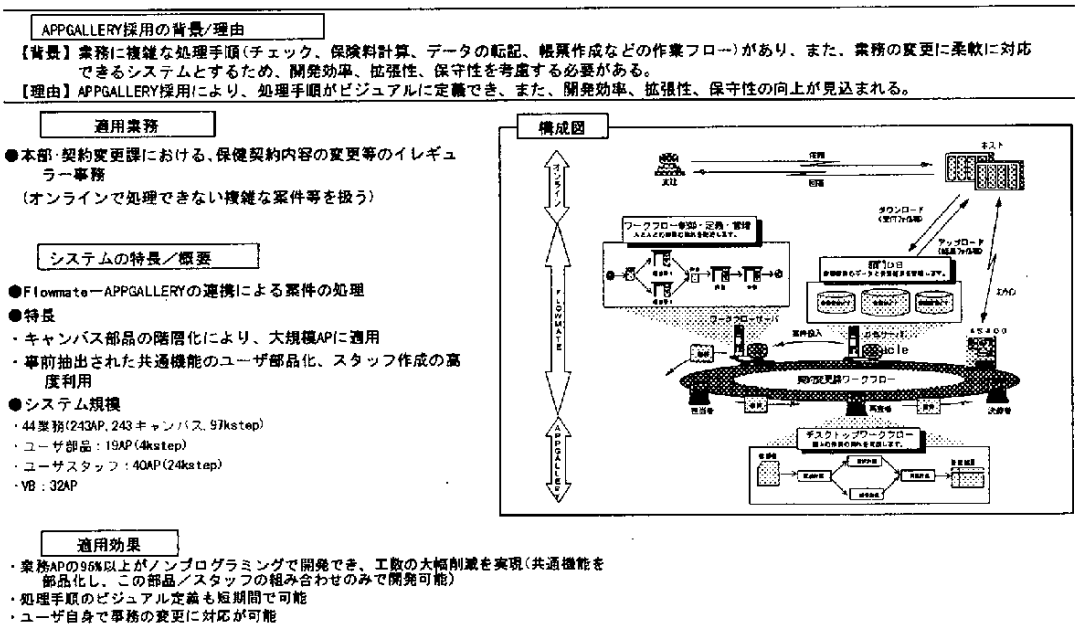


図2.1-8　保険業での適用事例

【参考文献】

1) Christopher Stone: What's the fuss over Business Objects?, First Class, Spring 1995, Object Management Group.

2) James Martin: Rapid Application Development, Macmillan (1991).

2．2　HOLON/VP

HOLON/VPはビジュアルに部品を組み合わせてクライアントサーバシステムを構築する統合開発環境である。

2．2．1　HOLON/VP の特徴

HOLON/VP は、パーソナルコンピュータ、サーバーマシン、メインフレーム、データベースがネットワークでつながれ展開されてゆく、これからの企業情報システムのためのソフトウェア構築環境を提供するものである。

様々な機能モジュールを部品化し、それらを効率良く組み合わせてシステムを構築するためには、各ソフトウェアモジュールを規定のプロトコルに合わせてコンポーネント化し、それらをシステムとして組み上げる（組み合わせる）仕掛けを提供することが必要である。

HOLON/VPでは、部品を組み合わせる技術としてオブジェクト指向技術に注目し、全ての部品はオブジェクトとして扱えるようになっている。HOLON/VPの特徴であるビジュアルプログラミングにおいても、部品（オブジェクト）と動作（メソッド）を画面上で扱うことができ、部品間の結線によって、メソッドの呼び出しやデータの受け渡しをビジュアルに記述することができる。

全ての部品は、C++言語によって作成されたものとHOLON/VPで提供されているスクリプト言語によって作成されたものに分類される。前者は、既存部品やシステム提供部品として予め与えられている部品と考えることができる。それら既存のC++部品をベースにして、後者によってシステム固有の部品を作り上げてゆく。両者の部品を混在し、プロトタイピング時（モジュールの実行検証時）とコンパイル時（デリバリ時）で使用することを可能にするHOLON/VPのコンポーネント化技術については、2．2．3項で述べる。

(1) 部品化と再利用

情報の収集や管理、抽出、分析、計画やプレゼンテーションなどの業務の機能は、適切なかたまりでそれぞれを得意なツールで作成し、流用/再生産しながら組み合わせ、全体として業務を構築していく必要がある。

HOLON/VPでは、各種部品の生成環境と、部品を統合する環境の両面について、一環したサービスを提供している。 図2.2-1に、HOLON/VP の部品生成と実行のアーキテクチャを示すが、マルチメディア、GUI、データベースなど、それぞれの部品向けのエディタを持ち、生成された部品は Hologram と呼ばれるスクリプト言語に変換され、インタプリタによって検証実行を行うことができる。さらに、コンパイル／デリバリ機能により、C++ のプログラムに変換後コンパイルされて単体の実行モジュールとなり、運

用マシン上で効率良く実行することができる。できあがった部品はオブジェクトとして、オブジェクト間の関係も含め資産管理の対象となる。

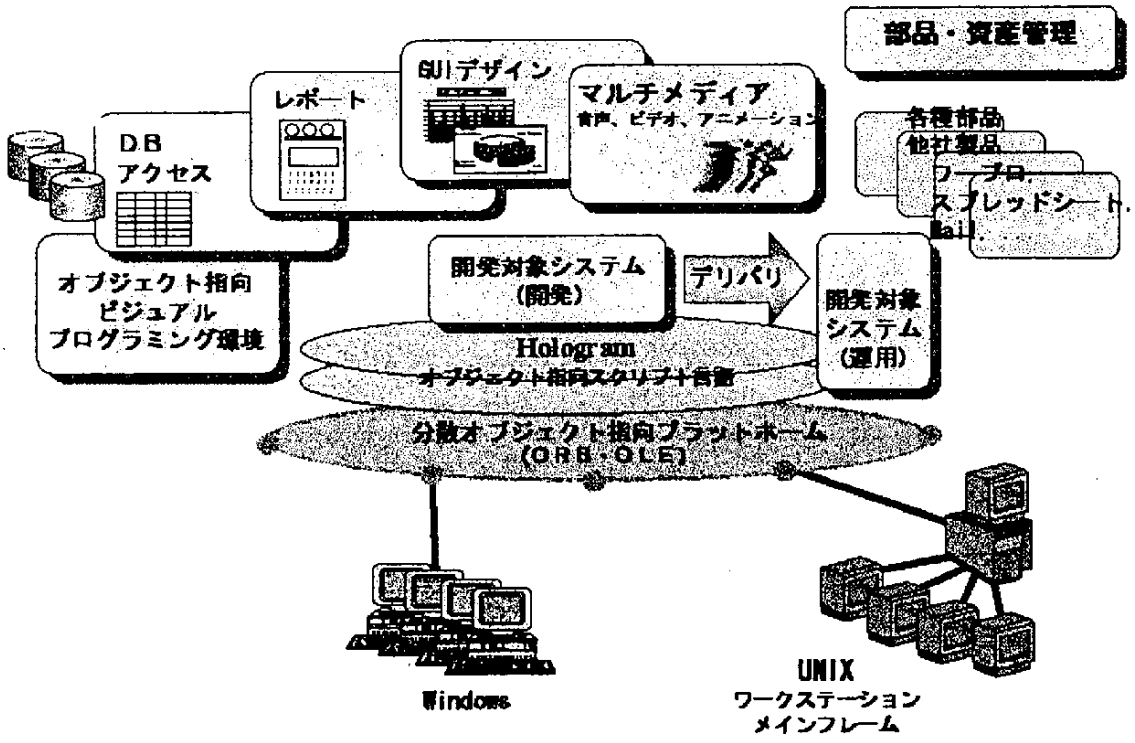


図2.2-1 HOLON/VPのシステム構成

図2.2-2に、HOLON/VPの部品間の結合を示す。上述の部品生成環境で作られた部品は、画面上でビジュアルに組み立てていくことができる。画面上に□（Boxiconと呼ぶ）で現れているのが部品であり、部品の結合はこれらの四角を結線していくことにより簡単に行うことができる。

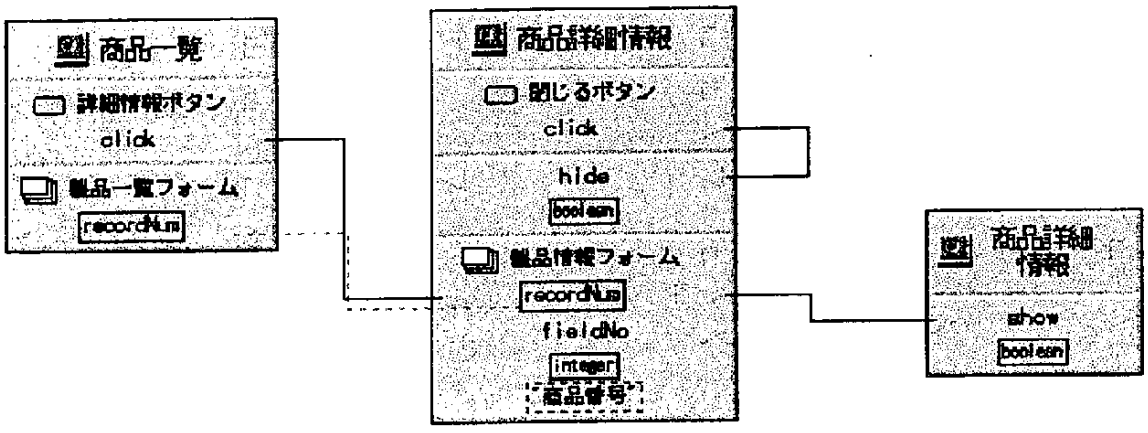


図2.2-2 ビジュアルプログラミングにおける部品間の結線例

また、この関係図はそのままシステム構成のドキュメンテーションと見做すことができ、システムの全体像を容易にみてとることができる。

(2) ネットワークアプリケーションの構築

前述の部品やそのかたまりは、ネットワーク上に配置され互いに協調しながら実行されなければならない。このときの分散の形態は、従来のサーバ中心の静的な垂直分散形態から、徐々に、より広範なネットワークアプリケーション形態へと広がりつつある。

HOLON/VP では、分散オブジェクト指向プラットフォームとして、ORB を実装しており、ORB 上での様々な分散アプリケーションを組むことができる。

複数のサーバーオブジェクトの配置、オブジェクトの配置の自由な変更、ローカルなオブジェクト呼び出しとリモート呼び出しの違いを意識させない、などの実装上の特徴を持ち、開発時にも、上記の部品結合と同じ考え方で、ビジュアルにクライアントマシンですべての開発が可能である。

(3) マルチメディアアプリケーションの構築

コンピュータが扱うメディアは、従来の文字、数値から文書、写真、音声、ビデオ、などと広がりつつあり、業務アプリケーションも多様なメディアへの対応が求められている。

HOLON/VP では、プレゼンテーションの中にイメージ、ビデオなどの様々なメディアを貼りつけることができる。また、ビデオ、各種音声を合成し 1 本のメディアとして（これをコンポーズドメディアと呼ぶ）作成する環境を提供しており、作成したコンポーズドメディアは同様にプレゼンテーションの中に貼りつけることができる。

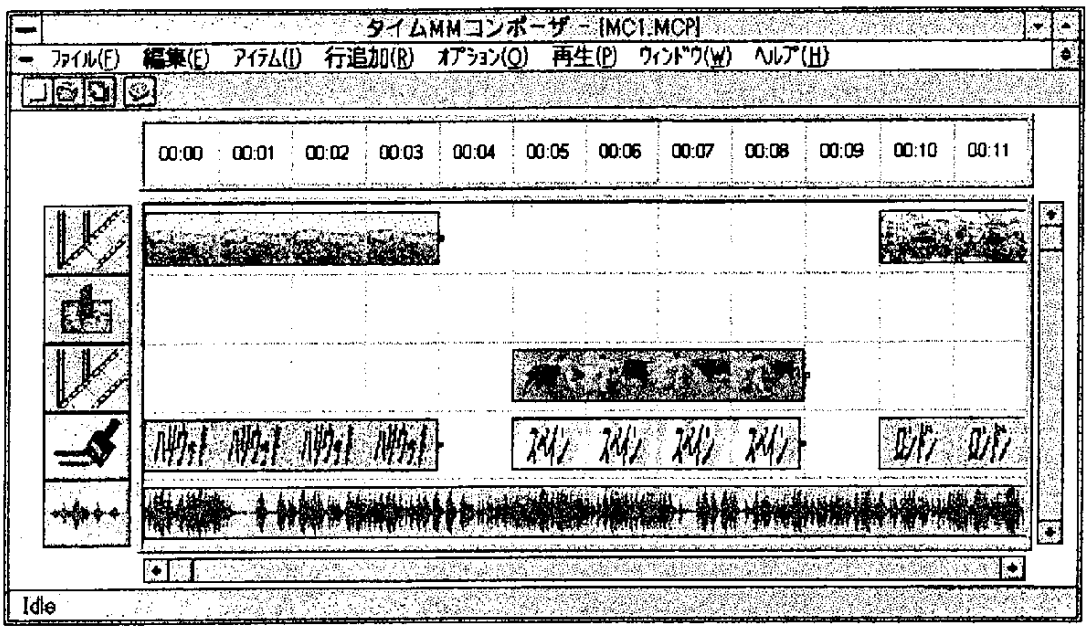


図2.2-3　コンポーズドメディアの作成例

図2.2-3に、HOLON/VP でコンポーズドメディアを作成する例を示す。これは時間を横軸として縦に素材メディアを置き、１本の合成メディアを作成している例である。素材と素材のつなぎには多彩な効果を用いることができる。

２．２．２　HOLON/VP によるアプリケーションの構築

(１) 部品の組み立て

HOLON/VP によるアプリケーションの構築で、一番の特徴は部品を組み立てる方法にある。HOLON/VP では、既に存在する部品、新しく作成した部品のいずれも、アプリケーションコンポーザと呼ばれるツールの画面上で １つのアイコン（Boxicon と呼ばれる）として表現される。

図2.2-4に簡単な例を示す。これはメニュー部品とデータベースを検索するキュエリ部品（売上高1）とフォームを画面に表示する画面部品（支店別表示）の３種の部品から組み立てた例である。Boxicon は、一番上にタイトルと呼ばれる部分があり、部品の名前や種別を示す。２番目以降は項目と呼ばれ、メソッドや部品に内包される小さな部品をあらわす。

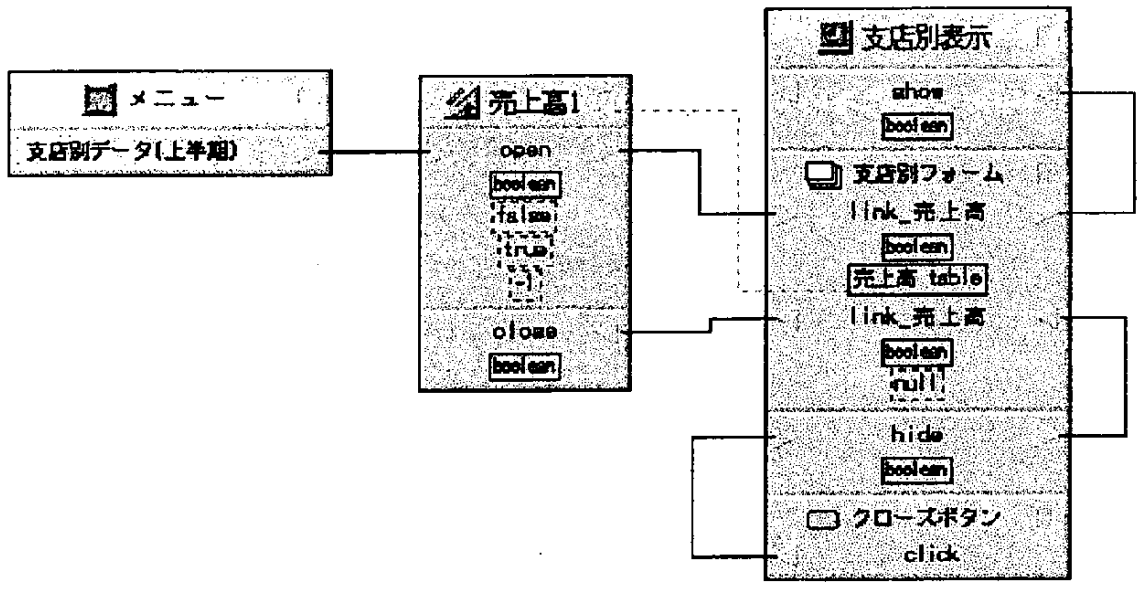


図2.2-4　メニュー部品, キュエリ部品, フォーム部品の結線例

図2.2-4で部品間は線でつないでいるが、実線は制御、破線はデータを示し、例えばメニューの支店別データを選べば売上高1 の open メソッドが動き、支店別表示の支店別フォームの link メソッドが動き、ついで支店別表示（画面）が show メソッドにより表示されるということを示している。また、このときに支店別フォームはキュエリの結果としての売上高データを入力する。

部品の組み立ては、このように部品を結線していくことにより行う。こうすることにより、複雑なアプリケーションも部品とその構成が一目でわかり、再利用/再構成などもきわめて容易になる。

(2) 部品の作成

個々の部品の作成は、それぞれ専用のエディタ環境で作成するか、Hologram（スクリプト言語）で作成するか、またはC++言語で作成することもできる。以下に代表的な部品の作成方法を示す。

(a) データベース部品

データベース関連の部品としては、データベースそのもの、テーブル、レコード、キュエリなどがあるが、これらはデータベースモデラーと呼ばれるツールを用いて作成する。

図2.2-5にデータベースモデラーの画面例を示す。これは、データベースのテーブルを定義しているところである。できあがった部品はクラス-オブジェクトとして生成されるが、データベースにかかわるクラス-オブジェクト間の全体の関係をオブジェクトダイアグラムを用いて表現することもできる（図の左奥“データモデル：受注”がオブジェクトダイアグラムを示す）。

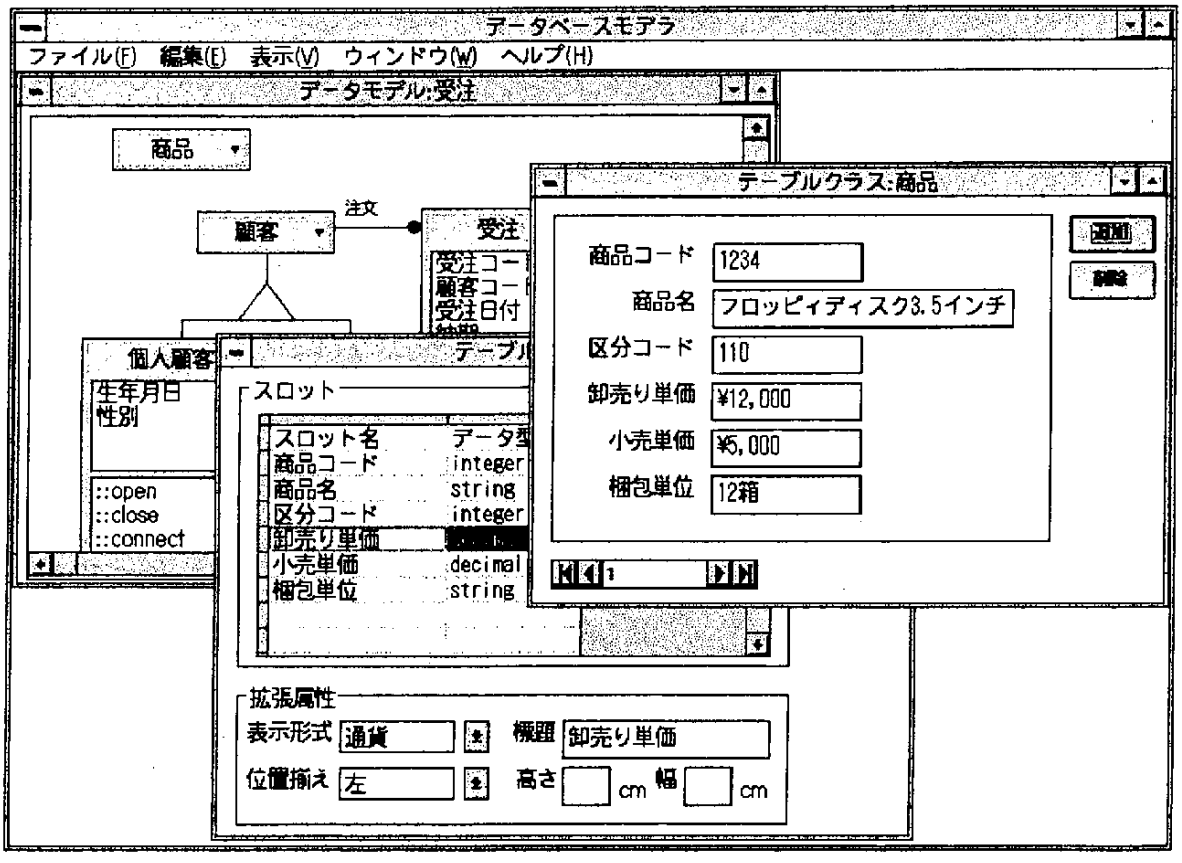


図2.2-5　データベースモデラーの画面例

データモデルは、HOLON/VPが内包するデータベースはもちろん、ORACLE、Infomix、RIQS、などHOLON/VPがアクセスできるデータベースならどのようなものでも表現することができる。

(b) テーブル/フォーム部品

データベースのテーブルは、データベース中のデータそのものを意味するが、アプリケーションで操作したり、画面に表示したりするためのデータを持つ部品の代表的なものとして、テーブル、フォームがある。

テーブルはデータを蓄積、表示するための構造、フォームは台紙のようなもので、その上にテーブルやグラフ、ラベルやボタンなどの種々の部品を載せたものである。

(c) マルチメディア部品

HOLON/VPでは、プレゼンテーション中に通常のGUIに加え、次のものを組み込むことができる。

・図形

・ビットマップイメージ

・AVIビデオ

・MIDIオーディオ

・WAVEオーディオ

・CDオーディオ

・コンポーズドメディア

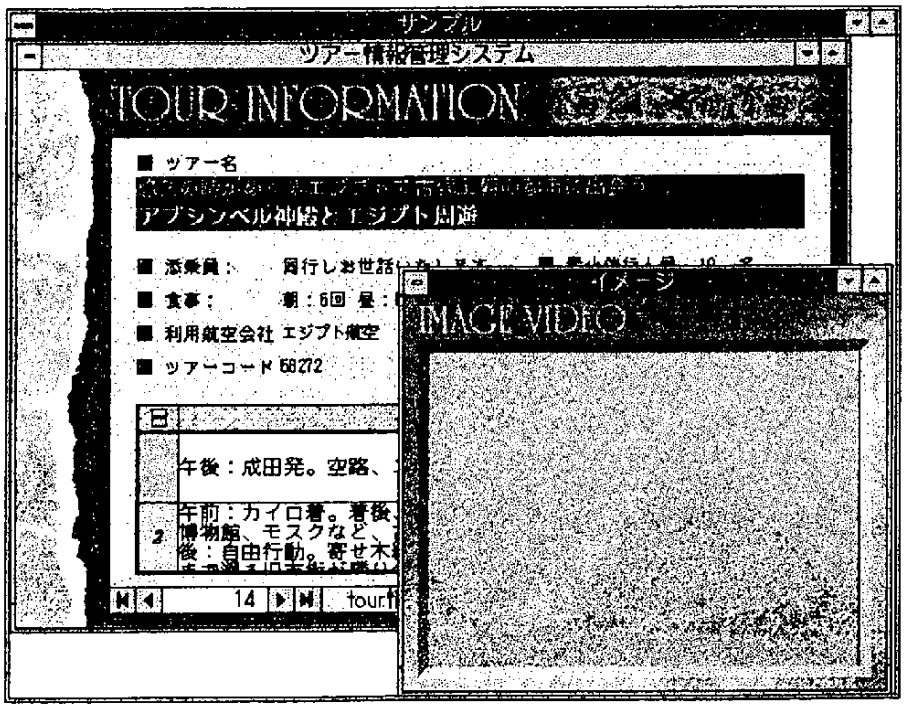


図2.2-6　マルチメディア業務の画面例

コンポーズドメディアは、HOLON/VPが持つマルチメディアオーサリング環境により作成されたメディアで、何本かのビデオ、オーディオを素材として１本のメディアとしてまとめたものである。

図2.2-6にマルチメディアを含む業務画面の例を示す。この例は、旅行ガイドを行うためにビデオ映像と音楽, 解説 を入れて効果を出しているものである。HOLON/VPでは、このような例でもマルチメディア部品を貼りつけ、前述のように結線するだけで簡単にアプリケーションを作ることができる。

(d) プレゼンテーション部品

プレゼンテーション部品は画面上の１個のウィンドウを構成する部品で、次の種類のものがある。

・プッシュボタン、ビットマップボタン、チェックボタン、ラジオボタンなどのボタン類

・リストボックス、コンボボックスなどの選択部品

・１行及び複数行のテキスト、矩形ラベルなどのテキスト部品

・水平/垂直スクロールバー

・線分、図形など

・表

・グラフ

・フォーム

・マルチメディア部品（前述）

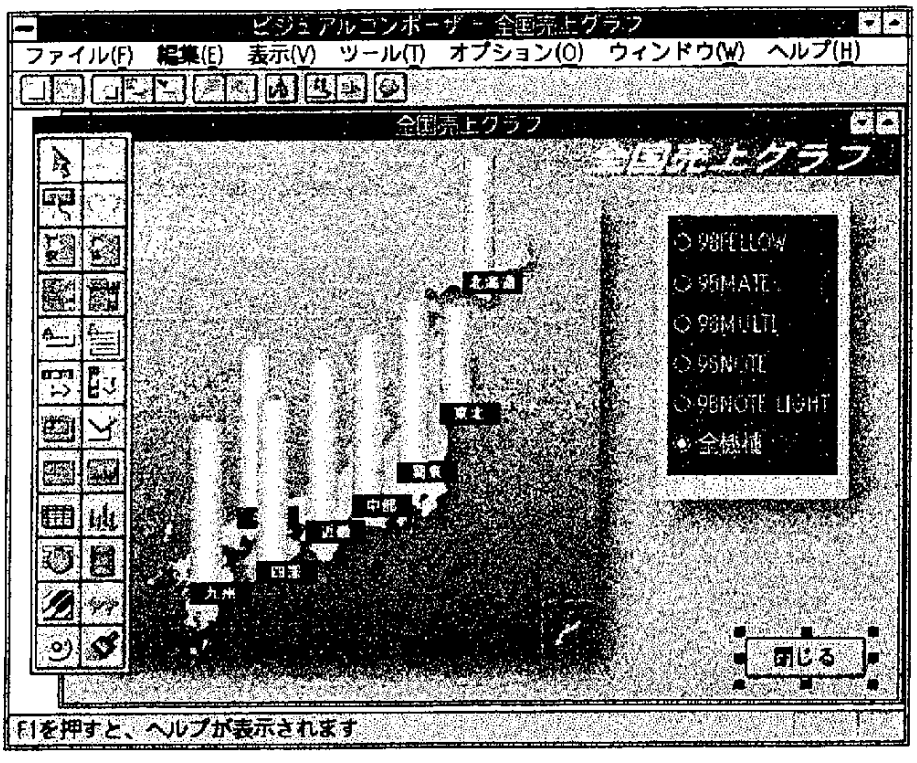


図2.2-7　プレゼンテーションデサイナによる画面設計例

これらの部品を作りだすとともに、１枚のウィンドウをこれらの部品を用いて組み立てるのは、プレゼンテーションデザイナと呼ばれるツールで、組み立てられたウィンドウは １つの部品となる。

図2.2-7にプレゼンテーションデザイナで画面を作成している例を示す。これは、イメージ、ボタンなどを組み合せているもので、部品パレット上から部品種別を選び、カストマイズしレイアウトを決めてゆくことで、自由に簡単に作成していくことができる。

(e) レポート部品

レポート部品は、ファイル内のデータをレイアウトし印刷するためのもので、レポートエディタを用いて作成する。

レポートの形式は、各種ヘッダー/フッダーと明細を含み、グラフ、ビットマップイメージなどを組みこむことができる。レポートエディタ上でパレットからこれらの構成要素を選び、自由にレイアウトしていくことにより作成していくことができる。また、プレビュー機能があり、エディタ上で作成しながらできあがるレポートを確認できる。

図2.2-8にレポートエディタでの作成例を示す。この例は、ヘッダー、明細とグラフを組み合わせた例である。

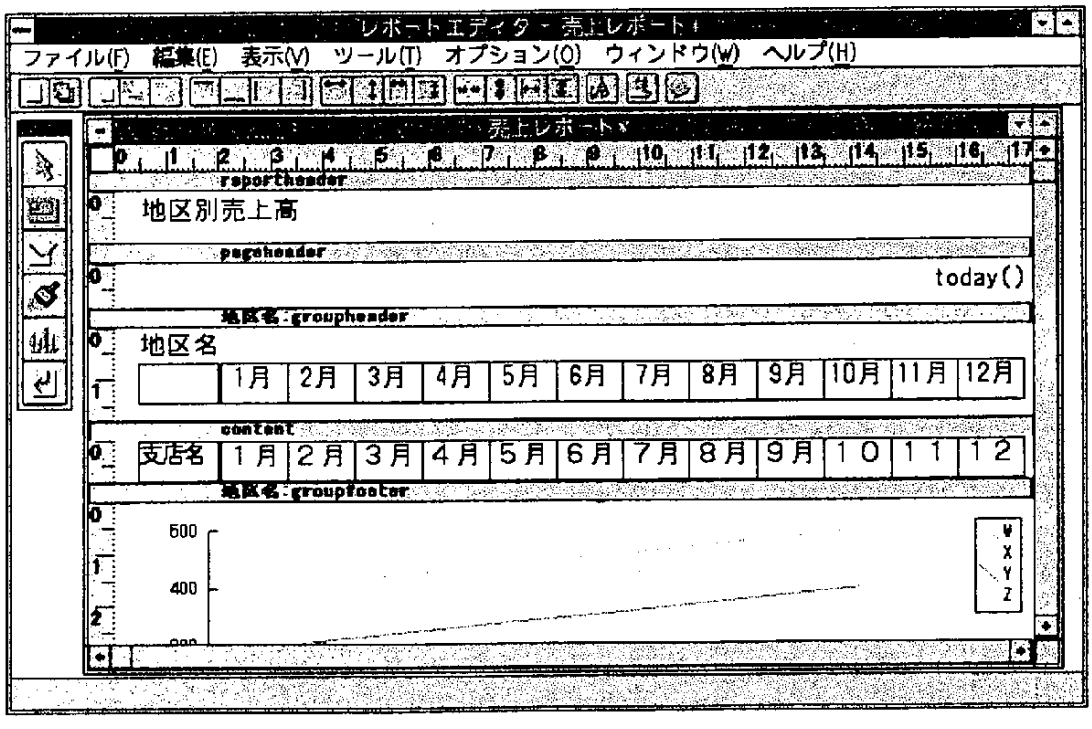


図2.2-8　レポートエディタによる出力票の作成例

(f) ネットワーク部品

HOLON/VP では、ORB を用いた通信のために、いくつかのメソッドが用意されており、これを用いて

ORB 上のアプリケーションを動作させることができる。通信のために必要な記述は、概略次のようなものである。

・メッセージ受け取りメソッドの定義

・ORB サーバとの接続/接断

・宛先の獲得、送信

図2.2-9にORBを用いて通信を行うプログラムの例を示す。図の“サーバにデータを１件送る”、“サーバに接続する”などの部品が ORB メソッドを含む Hologram 部品である。

なお、Windows上では、OLE を経由して他のプロセスと通信を行うことも可能である。

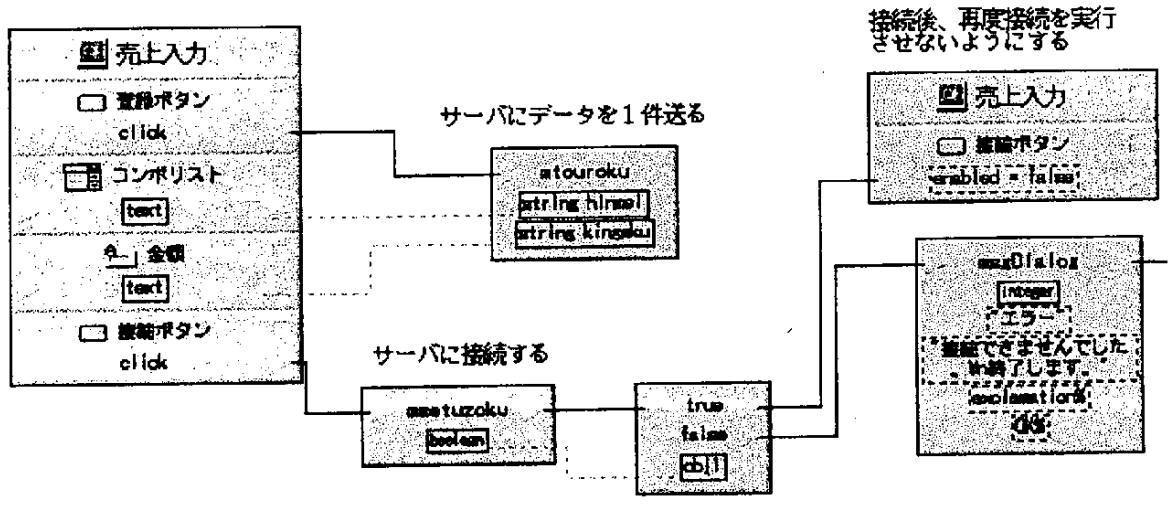


図2.2-9　ORBを用いた分散システム構成例

(3) スクリプト言語

HOLON/VP は、Hologram（ホログラム）と呼ばれるオブジェクト指向スクリプト言語を持つ。HOLON/VP の各ツールで生成された部品は、すべてHologramから操作することができる。また、Hologram で部品を作ることもできる。

Hologram は、インタプリタモードとコンパイルモードの両方を持ち、コンパイルすれば高速な C++ ソースコードが生成される。

図2.2-10に Hologram の記述例（クラス定義、メソッド定義）を示す。

```
class 預金;
    残高 : integer := 0;
end 預金;

method 年利 (x:fixed, y:fixed, z:integer) : fixed;
begin
    y := 1 + y / 100;
    loop i : integer in 1..z;
    do
        x := x * y;
    end loop;
    return x;
end 年利;
```

図2.2-10 Hologramプログラム例

(4) 部品の管理

HOLON/VP での部品の管理は、各アプリケーションを構成する部品を管理するプロジェクトと、共通に引用され用いられる部品を管理するライブラリの 2 階層によって行われる。これらの中で、部品は互いの関係（クラス関係、参照関係）を保ったまま管理される。

例えば部品のコピーを行うと、1 個の孤立した部品だけがコピーされるのではなく、その部品が動作するのに必要な参照部品も含めてコピーされる。

部品管理と前述のアプリケーションコンポーザ（部品を組み立てるツール）とは連動しており、部品管理に登録されている部品は、一動作でアプリケーションコンポーザ上に組み立てていくことができる。

(5) アプリケーションの生成

部品を作成し、組み立てたものはインタプリタにより実行させてもよいが、本番実行となりエディタやインタプリタなどの開発環境が不要になった場合、これらをはずし、プログラムをコンパイルし、実行マシンへインストールするために、パッケージングを行うことができる。

(6) 業務システムの運用

ネットワーク型のアプリケーションを運用する場合、ORB をインストールしておくことにより、前述のとおり簡易に柔軟なアプリケーション配置が可能になる。また、それぞれのアプリケーション内においても、改変や発展は部品の追加/改変や組変えによって行われ、しかもそれらをビジュアルに容易に行うことができる。

2．2．3　コンポーネント化技術

(1) ビジュアルプログラミング

様々な機能モジュールはC++やHologramで記述され部品化されており、それらはBoxiconと呼ばれる箱でビジュアルなオブジェクトとして表現される。Boxiconには、そのオブジェクトが備える機能（メソッド）が明示されているので、オブジェクト間のメソッドの呼び出しは、Boxiconどうしをビジュアルに結線することで表現することができる。これは1つの有効なソフトウェアコンポーネント化／結合技術である。そしてこの結合をスムーズに行うために、部品化に際してHOLON/VP部品としてのルールが定められている。このルールはオブジェクト指向技術における、クラス、インスタンス、メソッド、スロット、既定義のデータ型、という形式で表現されており、それがそのままHologram言語の仕様にもなっている。

(2) Hologramの特徴

多くの有用なクラスライブラリがC++で記述され、利用可能になっており、これらのライブラリを活用し取り込む機構が必須になっている。一方、HOLON/VPのようなエンドユーザ向けの開発環境で、直接C++のライブラリを提供し、C++をエンドユーザ向け言語として提供することは得策ではない。なぜならC++はユーザ向けの会話的プログラミング用としては全く不向きだからである。そこで、これらのライブラリを呼び出すことのできるインタプリタ言語（スクリプト言語）を開発し提供した。これがHologram言語である。

Hologramのスクリプト言語としての外観は2．2．2項（3）で述べたとおりであるが、コンポーネント／結合技術の観点から見たHologram言語の特徴は次のとおりである。

(a) インタプリタ言語であるとともに、トランスレータによってC++に変換後、ネーティブコードにコンパイルすることができる。

(b) 基盤となるC++ライブラリのコンパイルドコードは、Hologramのインタプリット時と、Hologramをコンパイルした後の双方で同一のコードを使うことができる。

(c) インタプリタは（コンパイルされた）C++のクラスオブジェクトの構造を知っており、C++ライブラリのクラスオブジェクトを生成したり、オブジェクトのメソッドを呼び出すことができる。

(d) Hologramで（コンパイルされた）C++のクラスから導出したクラスを定義することができ、そのメソッドを定義し、オブジェクトを生成し、メソッドを呼び出すことができる。

(e) Hologramで導出されたクラスは、インタプリット時もそれをコンパイルした後も、動作は同一である。

図2.2-11にC++ライブラリとHologramプログラムのインタプリット時とコンパイル（デリバリ）時における関係を示す.

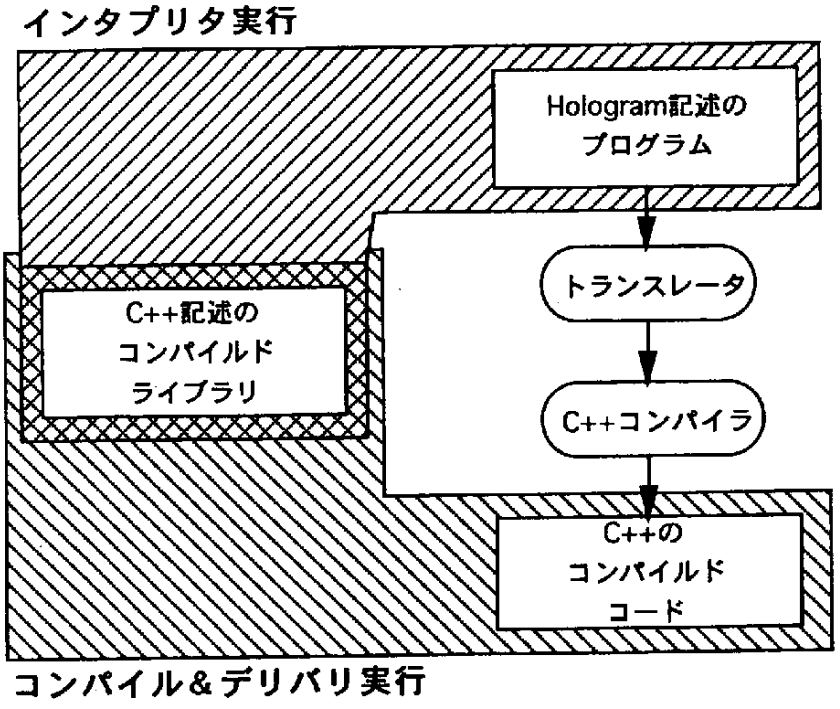


図2.2-11 C++ライブラリとHologramプログラム

これらの特徴を実現するためには、C++のインタプリタのようなものを実装すれば良いことになるが、実際には、C++フルセットのインタプリタは実装せずに、C++ライブラリの記述の方にある程度の制約を設けることによって、処理系全体の規模や複雑度とのバランスをとった。それがHologram言語処理系という形で具現化されている。C++ライブラリ側の制約としては、期間的、開発リソース的制約により、現在のところ、

・単一継承クラスのみ。

・全てのメンバ関数は仮想関数として定義されなければならない。

・型変換オペレータは定義できない。

・各クラスのコンストラクタはディフォルトのコンストラクタのみ定義可能。

・引数と返却値の型は“オブジェクトへのポインタ”でなければならない。

等となっている。

(3) C++メタオブジェクト技術

このC++インタプリタ部分は、Lisp系言語であるScheme処理系上で稼働しているCLOSとよばれるオブジェクト指向のシステムを用いて実装されている。CLOSは元々はCommon Lisp上に実装されたオブジェクト指向システム（CLOS = Common Lisp Object System）である。

このCLOSは、“メタオブジェクトプロトコル”と呼ばれるオブジェクト指向システムの“メタな”システムを備えているところに特徴があり、このメタオブジェクト上に、複数の多様なオブジェクト指向システムを実現することができる。我々はこのメタオブジェクト機構を用いて、HOLON/VP用に制限されたC++の機能を比較的容易に実現することができた。

CLOSのような汎用のオブジェクト指向システムのプラットフォーム上に実現しているので、C++の動作の実現（模倣）効率は最適化されているとは言い難いが、一方、C++に直接コンパイル＆デリバリが可能なため、運用時には最大限の性能が得られる構成になっている。

2．2．4 今後の業務の展望とその対応

ビジネス分野の企業情報システムは、パーソナルコンピュータの普及、インターネット、イントラネットの発展や、企業を取り巻く環境の変化により、常に変革を要求されている。アプリケーションの構築という場面でも、

・ネットワーク化されていくアプリケーションをどのように構築するか

・同様にネットワーク上の多様なデータベースをどのように処理するか

・プレゼンテーション、メディアの進化への追随

などの新しい課題、また新しい意味での複雑さ、規模の問題に直面している。

HOLON/VPが主に対象としているのは、このような分野であるが、今まで紹介された機能は、このような目的に応えていくための次のような考え方を基本としている。

・複雑、大規模な部品の組み合わせをどのように容易にするか

→[アプリケーションコンポーザ]

・多様なデータベース処理をどのように統一化するか

→[データベースモデラー]

・マルチメディア プレゼンテーションをどのように容易にするか

→[メディア コンポーザ]

・ネットワーク化する業務をどのように容易に構築、運用するか

→[ORBとネットワークアプリケーション構築環境]

また、これらの課題に適切に対応できる開発技法や方法論についても合わせて提供していかなければならない。

2．3 IntelligentPadとMAG

2．3．1 はじめに

IntelligentPadは北海道大学の田中譲教授によって提案されたシンセティック・メディアシステムであり、計算機上に実現されている様々なメディアやアプリケーション・プログラムや各種サービス・プログラムを電子的な紙として統一的に扱うことを目標としている[1]。このIntelligentPadのコンセプトに基づいて、分散環境での協調的なエンドユーザコンピューティング基盤を目指しているのが、MAG（Media AGent）である[3]。ここでは、まず前半で新しいプログラミングパラダイムとしてみたIntelligentPadの概要について述べた後、後半でMAGについてやや詳しく説明を行う。

2．3．2 IntelligentPadの概要

パッドは、ユーザによる直接操作が可能な可視的オブジェクトである。パッドが他のパッドに対して公開している内部状態やメソッドはスロットとして表現される。IntelligentPadにおけるプログラミングは、パッド間のスロットを結び付けてパッドを組み合せることによって行われるが、このパッドの組み合せが画面上でパッドを貼り合わせるという非常に視覚的で直感的な作業によって達成されることがIntelligentPadの特徴である。

図2.3-1にスライダパッドの上に数値パッドを貼り合わせた例を示す[2]。各々のパッドにおいてvalueスロットがプライマリ・スロットとして定義されていれば、この貼り合わせによって自動的にvalueスロット間の結合が生成される。このようにして生成された合成パッドにおいてスライダを動かすとそれに合わせて数値パッドの数値も自動的に変更される。これはスロット結合によって、親パッドのスロット値の変更が子パッドに伝播することによって行われる。

商用のIntelligentPadでは、貼り合わせの他に「結線による結合」の機能が提供されている。これは画面上で離れたパッド間でデータの連係を可能とする機能である。結線の機能にはパッドを画面上に自由に配置できるという利点があるが、その半面、画面上からデータ連係の関係がわかりにくいという欠点がある。このため、特に複雑なプログラムを作成する際には、結線による結合はなるべく行わないのが望ましい。

パッドを再利用可能なオブジェクトとするためには、基本部品として用意するパッドを、できるだけ機能が単純で汎用的な部品としてつくる必要がある。田中教授によれば、このような基本パッドは次のように分類することが出来る。

(1) 主に入力器として利用されるパッド群

ボタン、スライドバー、キーボード入力、音声入力パッドなど

(2) 主に出力器として利用されるパッド群

ラベル、ディジタルディスプレイ、さまざまなメータ、ビジネスチャートなど

(3)データを保持するためのパッド群

配列パッドなど

(4)パッドのジオメトリ管理を行うパッド群

縦方向や横方向、スタック状にパッドを配列させるパッドなど

(5)データの変換を行うパッド群

データ型変換や数値のスケール変換のパッドなど

(6)特定のアプリケーション向けのパッド群

数値計算、外部機器制御を行うパッドなど

(7)システムの機能を拡張するパッドなど

協調作業を可能とするパッドなど

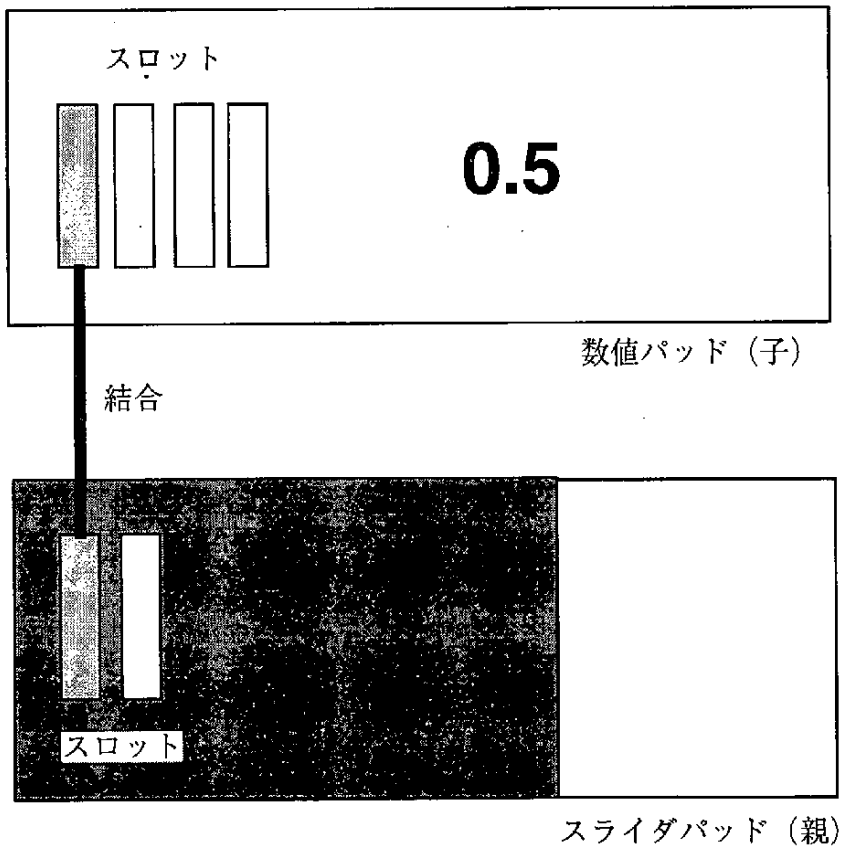


図2.3-1　データの連携

2．3．3　IntelligentPadの計算モデル

パッドは、MVCモデルによって実現されている。MVCモデルは、Smalltalk-80において、ウィンドウのモデリングに利用されている。MVCモデルでは、１つのウィンドウの機能はモデル、ビュー、コントローラという３つのオブジェクトの組合せによって実現される。モデル部は、内部状態とその操作機構を持つ。ビュー部は、モデルの持つ状態の表示を行う。コントローラ部は、ユーザからのウィンドウに対する操作を処理する。パッドは、図2.3-2のように各オブジェクト間でメッセージの交換を行う。Smalltalk-80ではコントロール部からモデル部などへもメッセージを送ることができるが、パッドではこれらを利用しない。また、MVCモデルでは、異なる種類のビューが一つのモデルを共有することが可能であるが、パッドではユーザがこれを行うことはできない。IntelligentPadにおいて、ユーザが扱うオブジェクトはの最小単位は、MとVとCの各オブジェクトではなくてMVCが一体となって定義されるパッドであり、このことがIntellignetPadの大きな特徴となっている。

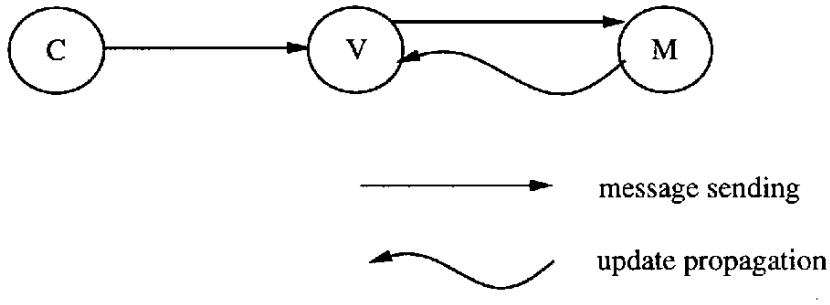


図2.3-2　パッドのMVC

IntelligentPadでは、内部機構はフレーム・モデルによってモデリングされており、パッドが他のパッドに対して公開している内部状態やメソッドは、スロットとして表現される。すべてのスロットは、スロット名、書き込み手続き、読み出し手続きを持つ。スロットへの書き込みは、set<slotname> <value>というメッセージによって行われ、スロットからの読み出しは、get <slotname> というメッセージによって行われる。

IntelligentPadでは、貼り合わせ関係にあるパッド間で授受されるメッセージを標準化している。これらのメッセージにはset, update, gimmeなどがある。set <slotname> <value> は、各パッドが自分の親パッドが持つ<slotname>で指定されるスロットに対して値<value>を送るためのメッセージである。ここで値<value>は、プライマリ・スロットから読み出された値である。プライマリ・スロットは、そのパッドの内容内部状態を代表する値が入っているスロットである。updateは、各パッドが自分のすべての子パッド

に対して、自分の内部状態が変化したことを知らせて状態更新を要求するためのメッセージである。gimme <slotname>は、自分の親パッドが持つ<slotname>で指定されるスロットから値を読み出してくるためのメッセージである。これらのメッセージは、それぞれのパッドごとに禁止することができる。これらのメッセージを制御することにより、自身と親パッドとの機能連係の仕方を指定することができる。

2．3．4　MAGの概要

MAG（Media AGent）は、IntelligentPadのコンセプトに基づいて、分散環境での協調的なエンドユーザコンピューティング基盤を目指すシステムである。MAGでは、ビジュアルな連係構造を実現するアーキテクチャにおいて、オブジェクトの埋め込みが構成する木構造をデータがフローすることで連係するというIntelligentPadの考え方よりも自由度を持たせる（図2.3-3）。MAGは次のような基本概念から構成される。

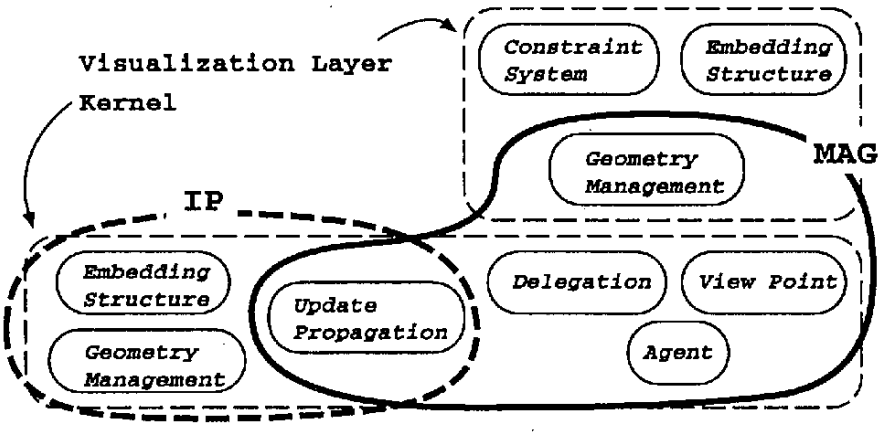


図2.3-3　MAGシステムの構成

(1) 基本データ構造は Observable、Observer、Context である。

(a) Observer/Observable：複数のスロットをインタフェースとして持つオブジェクトである。スロットはある型のデータを保持し、データへのread/write アクセスで起動される付加手続きを持つ。スロットはObserver から名前で参照される。あるスロットのデータが変更されると、そのスロットを参照するObserver の update メソッドを起動する。

(b) Context：Observable をノードとする木構造である。各 Observableはただ1 つの Context に属する。木構造は Delegation プロトコルにとって意味を持つ。

(2) 基本プロトコルは Update Propagation、Delegation である。

(a) Update Propagation：上記の Observable のスロットが変更されたとき、それを参照する Observer の update メソッドを起動する、ことを言う。

(b) Delegation：Observer が Context 内の Observable のスロットを参照するときに発生するプロトコル。Observer が、ある Observable のスロットを名前で参照すると、そのスロットが Observable にないと、Observable が属する Context 内の親方向に最も近いスロットを獲得する機構。

(3) magとは、参照関係を持つ Observer と Observable の集まりのことである（図2.3-4）。参照関係はmagの中で閉じている必要はない。mag同士の連携とは、他のmagとの次の関係付けをいう。

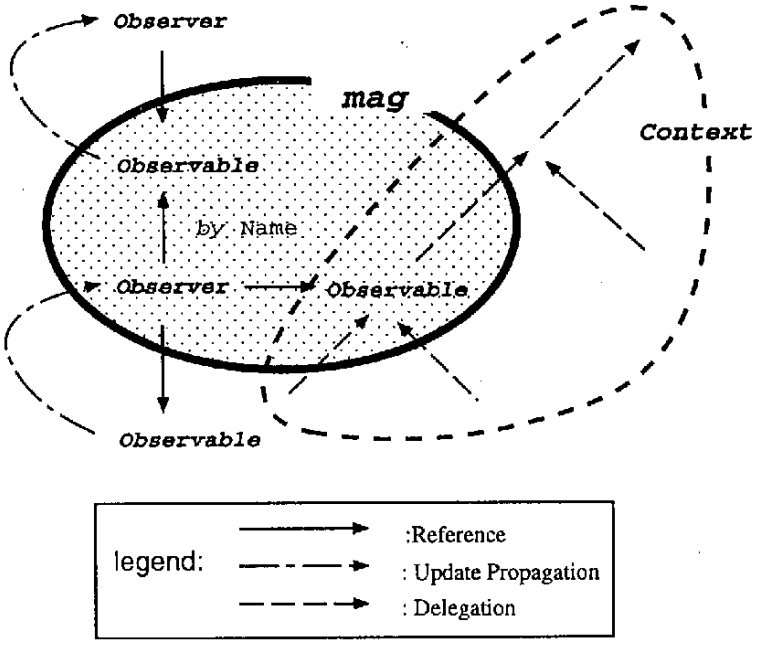


図2.3-4　magの構造

(a) 他のmagに属す Observer から、自身の Observer が参照される。逆に、他のmagに属す Observable を、自身の Observer が参照する。

(b) 自身の Observable が Context に属すため、自身の Observable への参照が、Delegation により Context に属する他の Observable を所有するmagへの参照になる。

(4) magは移動、複写、永続化が可能である。

(5) 表示、及び、イベント処理は、Observable の導出クラスライブラリとして、提供される。また、ジェオメトリカルな親子関係の管理は、Observer の導出クラスライブラリとして提供される。これらの導出さ

れたObservableとObserverから構成されるmagをｖmagと呼ぶ。ｖmagで構成される複合オブジェクトは、表示、及びイベント処理用のスロット：draw/request/allocate/pickに対するUpdate Propagationの効果として、張り合わせ構造を出現させる（図2.3-5）。これらのジェオメトリプロトコルは、レイアウト制御を柔軟に行うためにInterViewsのGlyph間プロトコルを参考にしている。

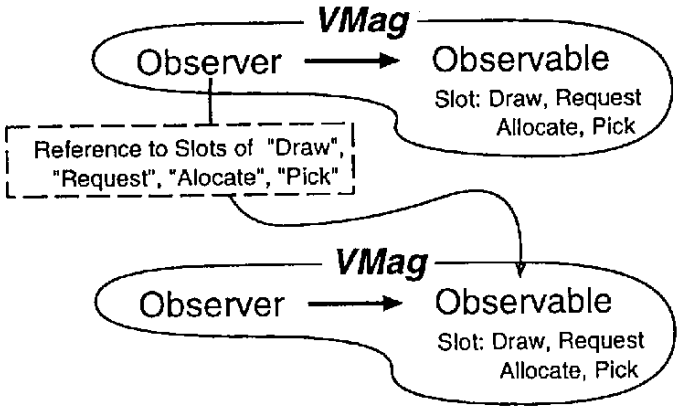


図2.3-5　vmagの構造とジェオメトリプロトコルのフロー制御

2．3．5　MAGの実現

前述の基本アーキテクチャに基づき、magの原理は次のように実現される。

(1)情報構造の可視化

(a)動的なクラス階層定義

Observableを Contextへ埋め込むことは、オブジェクト指向におけるクラス階層の定義を動的に行うことに相当し（図2.3-6）、通常の静的な継承構造がインスタンスのなす木構造として可視化される。各Observableはクラス間の差分を表すため、部品の機能分割をより細かく直交して行うことが出来る。

(b)データフローの可視化

Observer - Observable間の Update Propagationプロトコルに伴って発生するデータのフロー制御はObserverが行う。MAGはデータフロー制御の可視化を支援しないが、MAGの上位層として実装されうるシステムによる支援について述べる。

例１：Intelligent Pad

次のような特殊なｖmagをｐａｄと呼ぶ。”Observerが行うデータフロー制御は、自身のObservableの特定スロット（主スロット）と、ジェオメトリカルな親ｐａｄが持つ１つのスロット間に対してのみ行う”。よって、ｐａｄ間のデータフローは、ｐａｄの埋め込みが形作る木構造によって可視化される（図2.3-7）。

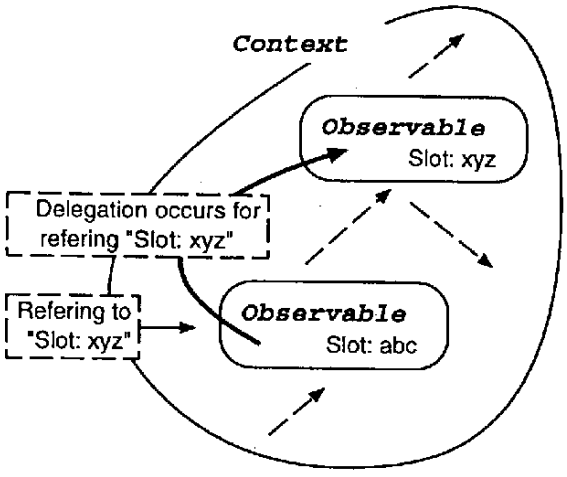


図2.3-6 Contextの木構造とデリゲーション

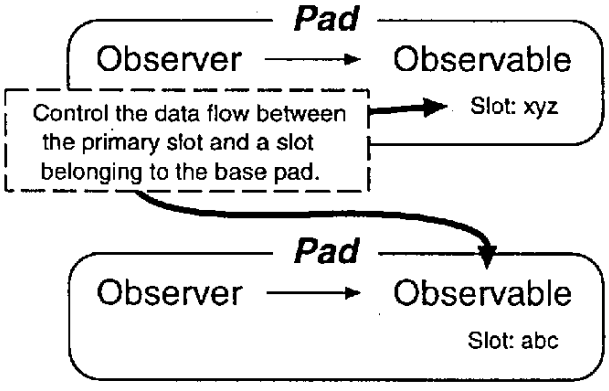


図2.3-7 Pad間のデータフロー制御

例２：制約解消システム

スロット値の更新に伴う Update Propagation(1)によって、複数のObservable に渡る関係の制約を保持する Observer が制約解消エンジンに制約解消を依頼し(2)、Observer が解消結果をスロット値として設定する(3)ことで、データのフロー制御を行う。エンドユーザは、スロット間の論理的な関係を、制約条件として宣言的に表現すればよいので、データのフロー制御を可視化していることになる（図2.3-8）。

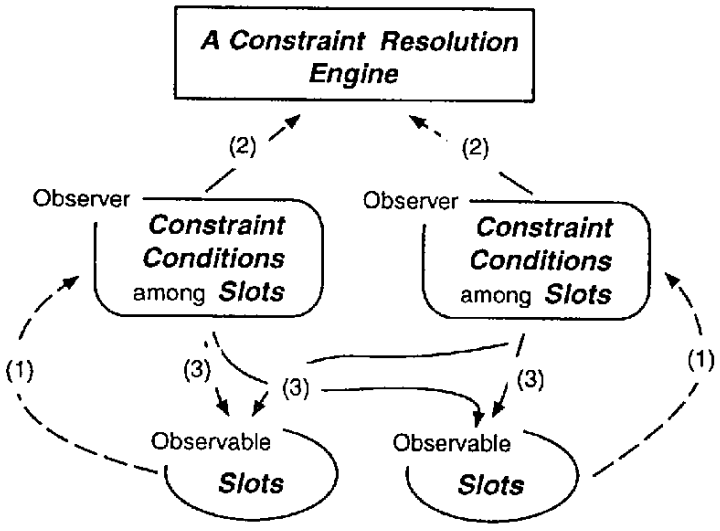


図2.3-8　mag上の制約解消システム

(2) ネットワーク透過性

ネットワーク上に於いて、インタラクティブな機能を有する情報の性能/一貫性/機密保護を保証した送信と共有を次のように提供する。

(a) 分散複合オブジェクト

MAGでは、Context が表す木構造、及び、Observer → Observable の参照関係を、ネットワークに跨って定義するための NetObserver/NetObservable 機構を用意している（図2.3-9）。本機構は完全に一般的なので、Observer/Observable と本機構はお互いを全く意識する必要はない。この一般性は、”Observer は Observable を名前によってのみ参照する”こと、”Observer から Observable へのメッセージはupdate のみである”こと、及び、”実行時データ型チェックができる”こと、に基づく。これによって、ネットワークに跨って連携するmag同士が複合オブジェクトを構成することができる。NetObservable は、ネットワーク上に分散させる時点で、セキュリティのため、認証され、権限を付与される。

(b) ユーザインタラクションの分散共有

v magの”視点”とは、v magの表示とイベントを双方向に共有するもう1つのv magである。視点の付加されたv magをシーンと呼ぶ。シーンと視点とは、NetObserver/NetObservable の導出クラス Viewpoint- Observer/ViewpointObservable とで実装され（図2.3-10）、pseudo canvasと呼ばれる仮想的なディスプレイ機構を持つ。また、視点は、座標変換によって見栄えを変更したり、ネットワークに跨ることができる。

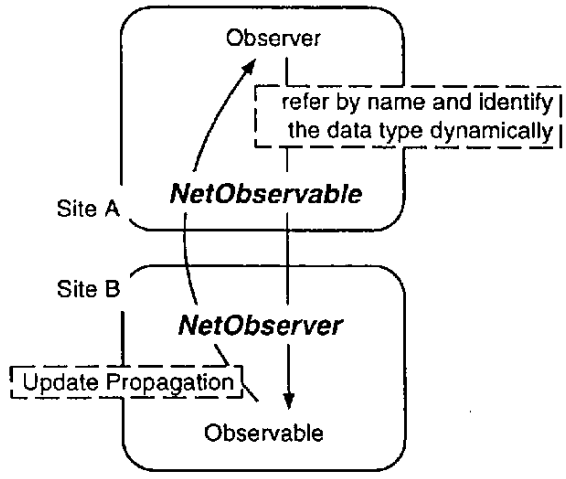


図2.3-9 ネットワークを越えた参照機構

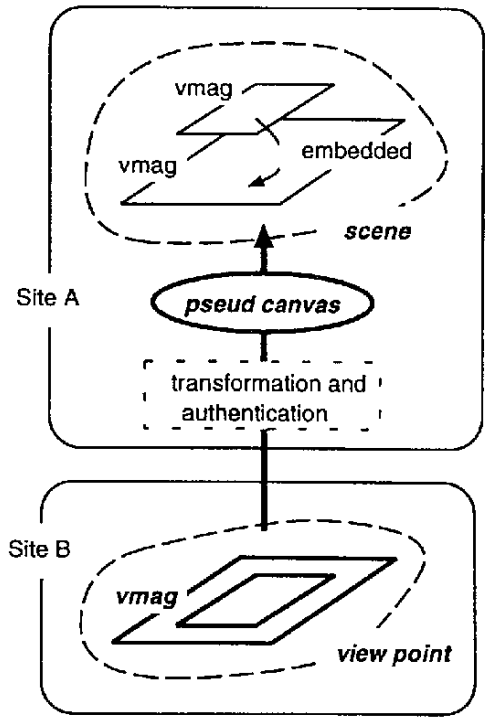


図2.3-10 視点

更に、視点は階層化することができる（図2.3-11）。図2.3-11では、site A の複合magの viewpoint が単体のmagとしてsite B にある。その view point は、site B のある scene の 1 つの要素になることができる。

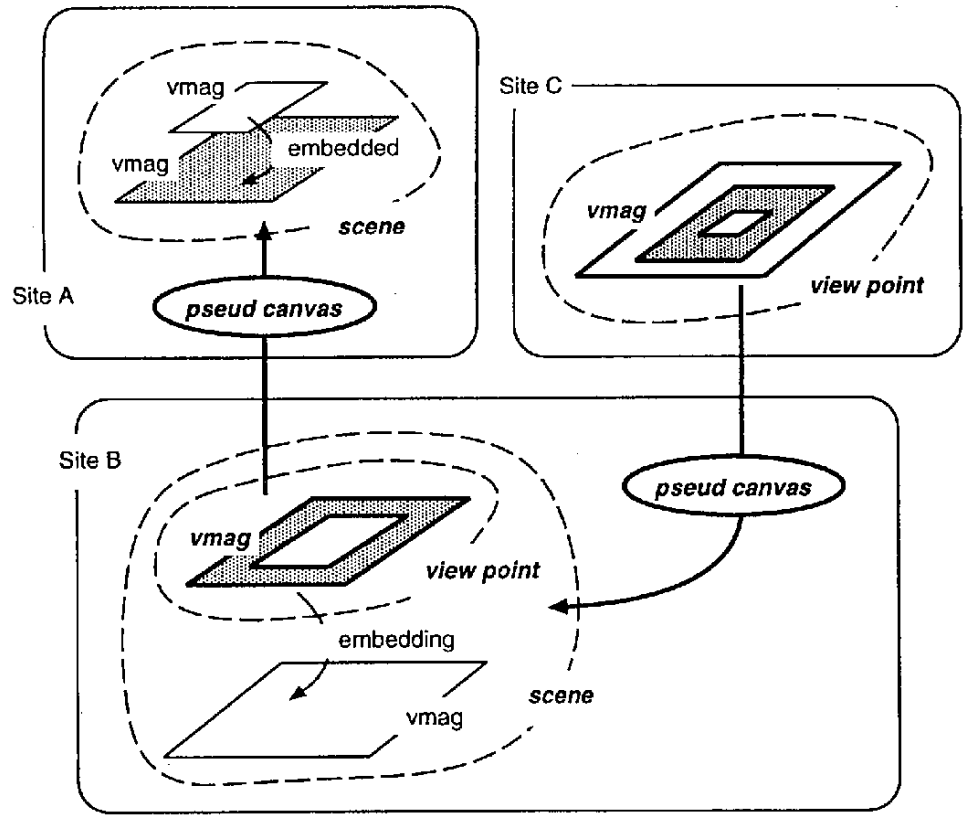


図2.3-11　階層的な視点

(c) エージェント技術

本技術は情報が自律的にネットワーク上を移動するための手段を与える。本技術は次の利点を有する。(i)疎結合された環境での分散処理、(ii)サーバ資源のマイグレーションの回避、(iii)通信量軽減（コネクションを維持する必要がないので）エージェント技術の実現に必要な機構には以下のものがある（図2.3-12）。

(i) magのインスタンスマイグレーションを行う。magの自律的動作の過程で”remote fork”機構を実行すると、別サイト上に自身の複製を作成した後に別サイトで処理を続行する。

(ii) magの自律的な動作は個々のmagに固有なので、動作記述定義が実行形式としてあらかじめ各ホスト上に存在すると仮定できない。よって、インスタンスマイグレーションと共に、動作定義（実装によってはクラス定義）マイグレーションを行う。

(iii) magの移動時に、保持する全ての視点を同時に移動する。本機構は、ハイパーテキスト機能をもつエージェントを実現するために有用である。

(iv) 移動したmagが、新しいホスト上のmagとインタラクションするために。相手magを特定する方法としてはデータベース検索に相当する手段が必要であるが、今後の課題である。

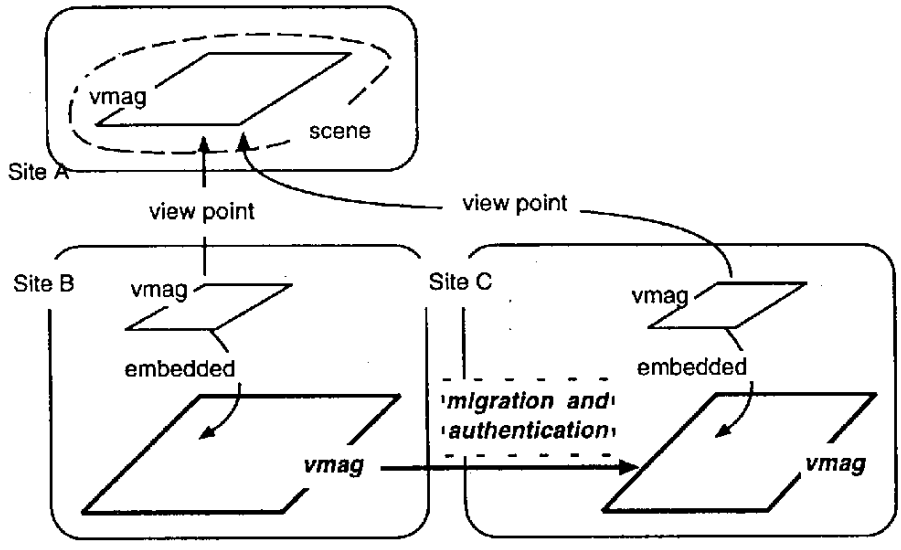


図2.3-12　magの移動と視点の保存

2．3．6　MAGの特徴

(1)更新伝播機構、デリゲーション機構、制約関係によって複合オブジェクトの制御構造を動的に定義でき、複合オブジェクトを作成するための機能合成力が高い。この事は、OLEやIntelProShareのプロセスやウィンドウに比べて、部品の粒度が高く情報表現の自由度が格段に優れていることを意味する。

(2)複合オブジェクトの制御構造は可視的であり再構築が容易である。この事は、エンドユーザによって複合オブジェクトの機能分解、再編集が繰り返され、従来に比べて部品化が飛躍的に促進されることを意味し、同時に、従来の複合オブジェクトが持っていたアプリケーション依存性の問題も回避される。

(3)複合オブジェクトはネットワーク透過的に階層的に構築されることができ、その透過性が、表示内容と双方の操作にまで及ぶ事ができるため、サイバースペースのような協調作業場を構築することができる。

(4)視点はサーバのシーンの内容を知る必要がない。この事は、従来のサーバ／クライアントのようにスケルトン／スタブという静的な関係付けが不要である事を意味し、クライアントは視点を経由してサーバの資源と動的に連携することができる。

(5)シーンの実体は一つである。この事は、従来の実体を複写して共有する方式に比べて、サーバ／クライアント間の矛盾が発生しない事を意味し、ファイルサーバやCPUサーバにあるmagとの連携を可能にする。

(6)更新伝播、デリゲーション、共有を基本とする計算モデルで明確化し、データフローの 可視化の層を分離した。この事は、MAGが、EUCのためのアプリケーションフレームワークを模索するための、よりプリミティブな実験場であることを意味する。

一例として、他のコンポーネントウェアの複合文書との比較を下表に示す。

表2.3-1　MAGと他のコンポーネントウェアの複合文書との比較

	MAG	OLE(OpenDoc)	Netscape+Java
分散共有	○	○	×
リンクの階層	多層	1層	多層
エージェント機能	○	×	×
動的機能合成	○	×	×
複合オブジェクト再編集	○	×	×
定義体移動／セキュリティ	○	×	○
リポシトリ管理	×	△	△

2．3．7　MAGの適用例

ここではMAGの適用例として、WWW上での情報の発信／受信について述べる。MAGが最も適しているのは、その場の思いつきや工夫のスケッチ風に表現し、使い捨てるような領域だからである。MAGの複合オブジェクトが従来のページに相当し、情報の表現や解釈、及び分散環境下での情報の共有・転送・ナビゲーションを中心とした検索を統合的に支援する。ハイパーリンクはホットであり、これを経由してMAGのフレームワークが与えるプロトコルでmagが連携する。ハイパーリンクを通じて操作や表示を共有できるので、分散協調作業場を実現することもできる。更に、MAGページは、自律性を持つエージェントとして、メイル運搬やページの検索依頼も可能である。

(1)リンクオブジェクトと機能合成及び再編集が可能なハイパーテキスト。

図2.3-13の様に、他システム上のMAG（ページ）Xへのハイパーリンクを示すアンカをMAG（ページ）Yに貼っておく。アンカーをクリックすると図2.3-14の様に、以下のいずれかの動作をする。

(a) Xが他システムからDown Loadされて、Yにホットリンクされる。

(b) Xが他システム上にLoadされて、Xの視点がYにホットリンクさ れる。他システム上のMAGカーネルの立ち上げはhttpdに依頼する。

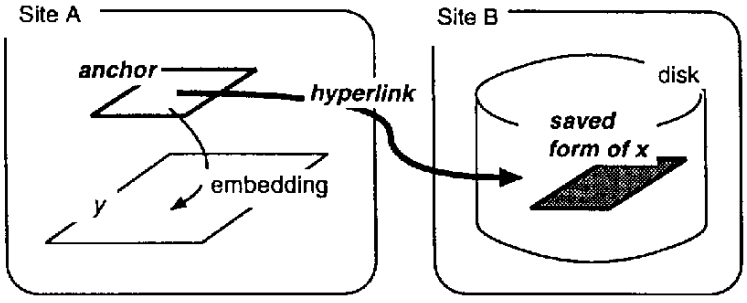


図2.3-13 アンカー

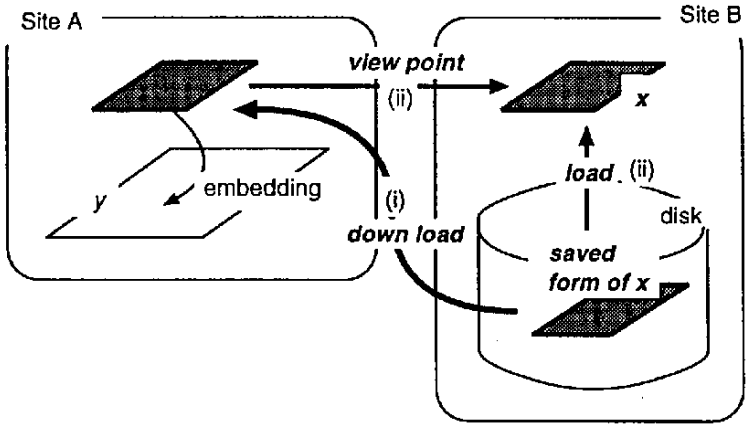


図2.3-14 アンカーをクリックした時の動作

(2) 協調場としてのハイパーテキストにリンク先のオブジェクトとして参加する。

図2.3-15 (i) の様に、他システム上の協調作業場のへ視点に、v magの”遠隔埋め込み”を行うことにより、v magの視点と協調作業場とのホットリンクが確立される。

(3) 自律的メイルとしてのハイパーテキスト。

前述したようにhot なハイパーリンクの一貫性を保ったままネットワーク上を移動することができる。更に、再編集機能を持つためフォームフロー処理も可能となる。

(4) 参加型エージェント。

(2)のv magが自律的に働く場合で、Aが入り込んだ協調場に、視点を経由して、適時、人が参加する。電子マーケットで商品選択する等。また、(2)の形態のネットワーク負荷軽減対策という見方もできる。

図2.3-15では Site A にある vmag が他の Site B にある協調作業場に移動して(i')、協調作業を行う。次に、

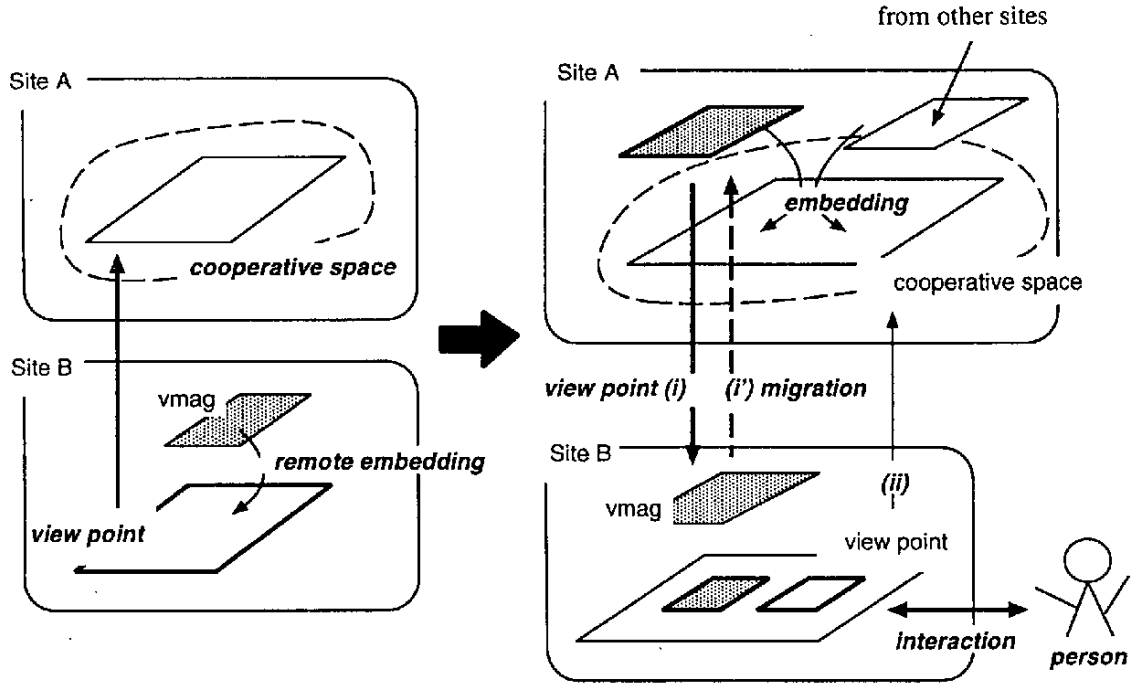


図2.3-15　協調作業場

協調作業場の view pointが、作業者のいる site に送られ(ii)、作業者はその視点を通じて協調作業場とインタラクションを行う。(i)～(ii)は図中の番号を示す。

2．3．8　おわりに

エンドユーザのためのソフトウェア構築環境としてみたIntelligentPadとMAGについて述べた。IntelligentPadは、更新伝搬のメカニズムなどの点で他のビジュアルプログラミングツールとは大きく異なった特徴を備えており、真にオリジナリティを持った純国産の技術として今度の発展が期待されている。

【参考文献】

1) 長崎祥,田中譲: シンセティック・メディアシステム : IntelligentPad, コンピュータソフトウェア, Vol.11, No. 1, pp.36-46 (1994).

2) インテリジェントパッド: アスキー出版 (1995).

3) Suresh, K.G., and Kainuma, T.: A Distributed Platform for Multimedia End-User Computing, International Symposium on Software Engineering for the Next Generation (1996).

2．4　OLEによるアプリケーション統合環境

マイクロソフトのOLE（Object Linking and Embedding）は、アプリケーションコンポーネント統合技術としてWindows NT、Windows 95などの製品に組み込まれている。アプリケーションをデータとそのデータを操作する関数をまとめたプログラム単位コンポーネント（オブジェクト）の連携により構成するため、OLEはコンポーネントオブジェクトモデル（COM)）とよぶ連携メカニズムとプロトコルの基盤を持ち、優れた操作性、システム拡張性を提供する。COMによりアプリケーションはコンポーネントの組み合わせた「コンポーネントウェア」となる。コンポーネントは再利用可能なソフトウェア単位であり、あるコンポーネントを他社製アプリケーションへ組込むことが可能である。たとえば、あるスペルチェッカを複数の他社製ワープロソフトに組み込んだり、トランザクションモニタを複数のデータベースサーバーと連携させるコンポーネントが考えられる。一方、従来のアプリケーションは単一構造（モノリシック）から成るため、機能の一部の削除や置き換えは困難であった。

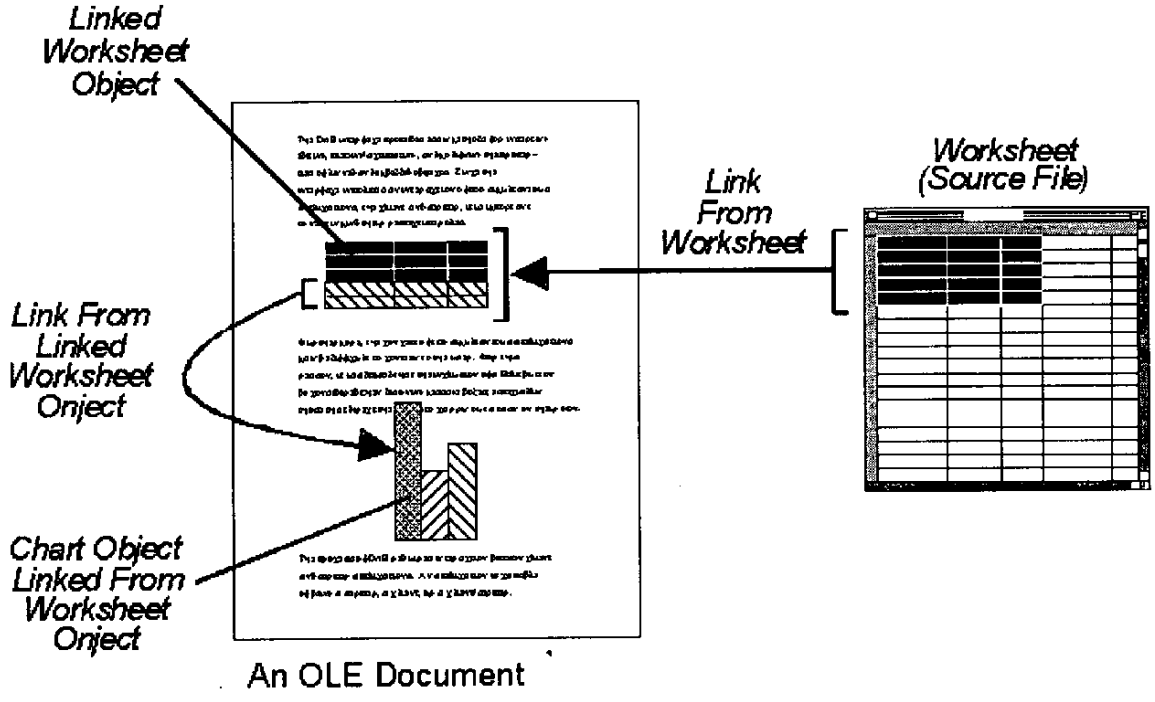
OLEの主要機能には、複合ドキュメントと呼ぶ図形や表などが張り込まれたドキュメントの作成を容易とするビジュアルエディティング、オブジェクトの効率的な記憶方式を提供する構造化ストレージ、異なるバージョンのコンポーネント間の連携を可能とするバージョン管理、後述のOLEオートメーションやOLEコントロールなどがある。さらに、ビジネスオブジェクトとして業界別オブジェクトの標準化と仕様策定が進んでいる。金融証券市況データのリアルタイム転送とデータ処理、流通POSでの装置制御とアプリケーション連携、地方自治体向け人名処理システムでの漢字処理、CAD/CAMシステムなどがあり、既に一部のソフトウェアコンポーネントが提供されている。従来のOLEの機能はどちらかと言うと事務処理系ソフトの効率化とコンポーネント化という意味合いが強かったが、OSのオブジェクト指向化の進歩に合わせて、システム基盤技術、ミドルウェアを中心とした分散システム機能への機能拡張も進んでいる。これらは、データベースアクセス機能、メッセージング機能、トランザクション機能、非同期型のミドルウェアであるキューイング機能などである。このうち、データベースアクセス機能、メッセージング機能は既に提供されており以下のコンポーネントウェアの構築例で紹介する。トランザクション機能、キューイング機能は現在米国で仕様を策定中である。

以下では、OLE複合ドキュメント、OLEによるコンポーネントウェア構築の基本技術であるOLEオートメーションとOLEコントロールの概要、コンポーネントウェアの構築例、および業界別オブジェクトを紹介する。

2．4．1　OLE複合ドキュメント

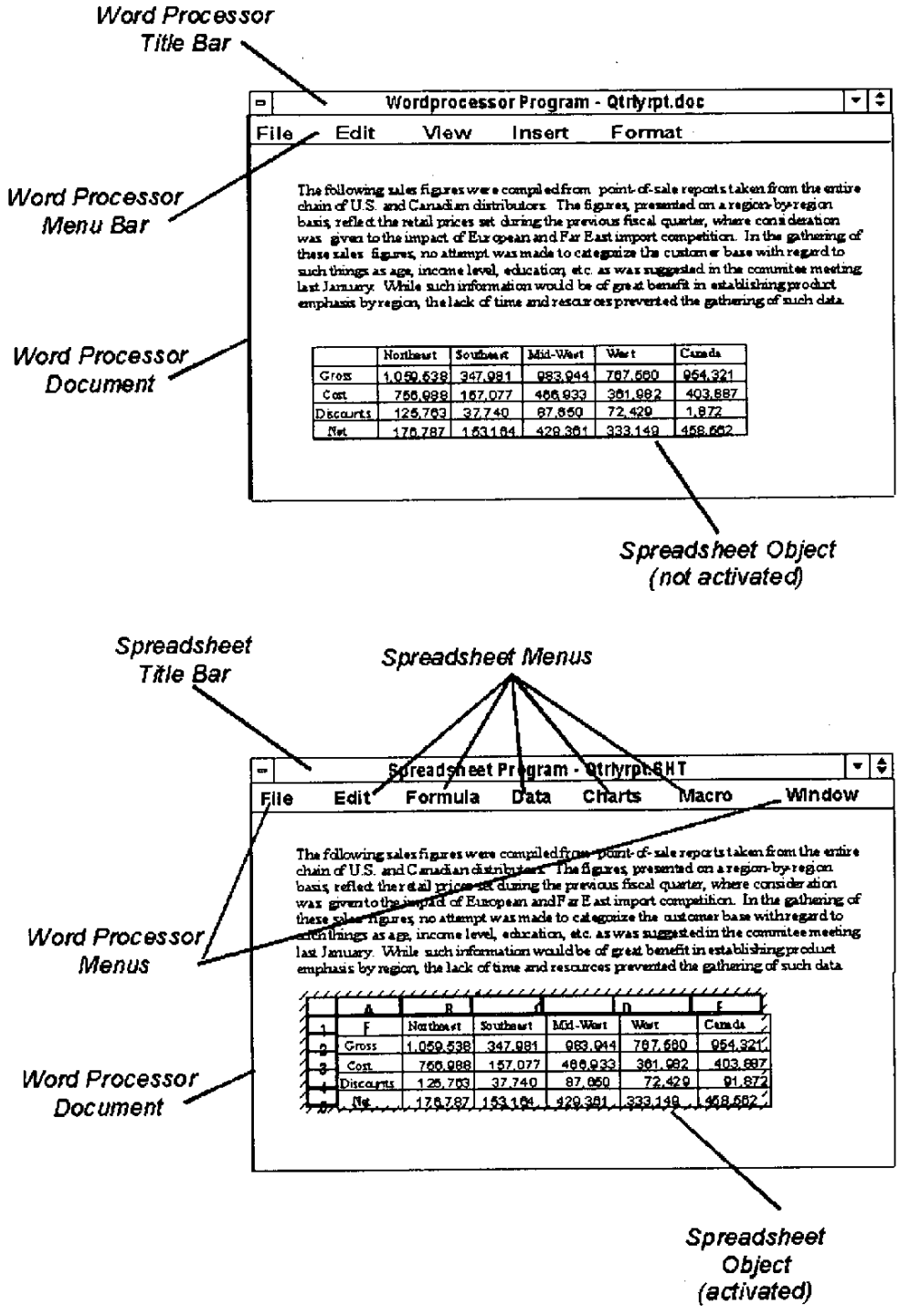
OLE複合ドキュメントは、ワープロ文書、スプレッドシート（表計算ソフト）の表の一定領域、などさ

まざまなOLEオブジェクトを貼り込んだものである。OLE複合ドキュメントに対して、ユーザーは各OLEオブジェクトがどのアプリケーションで作られたかを意識することなく、マウス操作だけでオブジェクトの編集処理を進められる。画面に現れるメニューは自動的に切り替わり、もともとオブジェクトを作ったアプリケーションと、複合ドキュメントを持つアプリケーションのメニューが統合された状態になる。ツールバー、ウインドウタイトル、ステータスバーなども変わる。これがOLE複合ドキュメントのインプレースアクティベーション（オブジェクトのある場所での起動）である。



〔右側の表計算ソフトの表（ワークシートオブジェクト）の一部の領域が左側のワープロ文書にリンクし、さらに同じ文書内のグラフ（チャートオブジェクト）にリンクしている。ワークシートオブジェクトに変更が加わると、ワープロ文書の表とグラフがそれに応じて変更される。〕

図2.4-1　OLE複合ドキュメントの例



〔上が起動前のオブジェクト、下が起動中のオブジェクトーOLE複合ドキュメントのオブジェクトの起動には、マウスでオブジェクトをダブルクリックする。オブジェクトが起動されると、オブジェクトを編集するためのメニュー、ツールバー、ウィンドウタイトル、ステータスバー、ヘルプなどが起動中のオブジェクト用に変更される。ユーザはどのオブジェクトの編集処理にどのアプリケーションを呼び出す必要があるかを意識しないで操作ができる。〕

図2.4-2 OLE複合ドキュメントのインプレースアクティベーション

OLEはオブジェクト指向で言うカプセル化技術に基づいて動作する。OLEオブジェクトに対する基本操作は、標準仕様として共通に定義してある。OLEオブジェクトを利用する側のソフトウエアはこの仕様に従い、表示、編集などの操作を指示することができ、実際の操作はオブジェクト自身が担当する。

以下、OLE複合ドキュメントの機能を実現するメカニズムについて、説明していく。

(1) OLEオブジェクトとは

OLEオブジェクトは、独立したソフトウエア部品（コンポーネント）と見ることができる。コンポーネントは再利用可能なものとして、オブジェクトのデータにアクセスする種々のプログラム（クライアントアプリケーションあるいはOLE複合ドキュメントではOLEコンテナーと呼ぶ）から利用できる点にメリットがある。OLEではこのことを実現するため、オブジェクトとクライアントとの関係は、公開された標準仕様として定義している。

オブジェクト指向技術では、オブジェクトとはデータとデータを操作する手続き（メソッド）を一体にしたものを言い、データへのアクセスは必ずメソッドを介して行うことになっている。これをデータのカプセル化と呼び、ちょうどデータをオブジェクトというカプセルに封じ込めた感じになる。カプセル化は、クライアントがオブジェクトのデータを操作するとき、オブジェクトのデータ構造（オブジェクトの内部管理情報を外に見えないように構造化したデータ）の詳細を意識せずにすむようにする。

これにより、クライアントアプリケーションは、アクセスするデータに依存しないものになる。オブジェクト内のデータ構造に変更があってもクライアントは影響を受けないこと、データへのアクセスがメソッドの操作範囲内に限定されるため不正な参照や変更をなくせること、が利点となる。オブジェクトの開発者は、クライアントに提供するメソッドさえ変更しなければ、自由にデータ構造を変更したり、メソッドの処理アルゴリズムを変更したりできる。

(2) 識別子を持つOLEオブジェクト

コンポーネントソフトウエアとしてのOLEオブジェクトは、ワープロ文書、スプレッドシートの表の一定の領域、HTMLのリンクページ、CADデータの図形要素、クエリーによるデータベーステーブルの一部、電子メールのアドレス帳、トランザクション処理のコンテキストといったものである。また、スペルチェック、マルチメディア画像の再生、受信FAXの表示などといったアプリケーション機能の一部、表計算ソフトのチャート機能、データベース管理機能といったアプリケーション機能の論理的なまとまりもOLEオブジェクトとして扱える。

プログラミング言語のための従来のオブジェクト指向技術とは、オブジェクトの大きさ（カバーする機能の範囲）は異なる。オブジェクト指向言語でのオブジェクトといえば、ダイアログボックスのようなグラフィカルユーザーインタフェース部品やリストのようなものを指すことが多かった。

OLEオブジェクトが、各オブジェクトを特徴づけるクラス識別子（CLSID）を持つのも特徴である。例えば、MS-Wordバージョン６の文書は特定のCLSIDを持ち、クライアントが一太郎など他の文書やMS-Wordの別バージョンの文書と区別するのに用いる。ここで、CLSIDはMS-Wordバージョン6の文書同士の識別には使わない。これらは同一CLSIDをもった異なる文書オブジェクト（クラスのインスタンス）であり、従来どおりファイル名で識別する。

ここで、注意すべき点は、MS-Wordバージョン6のCLSIDは文書データだけでなく、そのデータ構造を特定する点である。データ構造はMS-Wordの他のバージョンのデータ構造とも異なり、その構造にアクセスするメソッドも一般には異なる。たとえば、新しいバージョンで追加された機能は、古いバージョンの文書にアクセスするメソッドとしては存在しえない。従ってCLSIDは、文書データとデータにアクセスするメソッドとをまとめた特徴を代表している。

(3) クライアントはオブジェクトのメソッドを呼ぶだけ

OLEオブジェクトとして、CADの図形データを考えてみよう。CAD製品のメーカーとバージョンの違いによってCLSIDを変え、互いにオブジェクトを識別することになる。ここで、OLEオブジェクトにアクセスするクライアントは、CAD図形データを貼り付け、表示、編集するCADアプリケーションである。

このクライアントは、複数の図形データを貼り付け、組み合わせてCAD図形を仕上げるため、内部的に現在貼り付けられているすべての図形データ（OLEオブジェクト）をリスト構造を用いて管理する必要がある。OLEオブジェクトがクライアントと異なるメーカーのCAD製品の図形データであっても、CLSIDをもつことから、OLEはオブジェクトを処理するCADアプリケーション（OLEサーバー）を知ることができる。このとき、OLEはWindows 95あるいはWindows NTのレジストリデータ（システム設定管理用のデータベース）を参照して情報を得る。

貼り付けられた各製品のOLEオブジェクトは、異なるCLSIDであっても、複合ドキュメントのOLE標準インタフェース（注：OLEでいう機能別のメソッドグループ。たとえば、複合ドキュメント、OLEオートメーションのインターフェイスなどがある）や、オブジェクト基本操作（開く、閉じる、保存、削除、切り取り、コピー、貼り付け、選択、表示、編集など）は共通に呼び出せるようになっている。これにより、各OLEオブジェクトの内部の実現の方法が異なっていても、クライアントはその違いを意識することなくオブジェクトのメソッドやOLEインタフェースを呼び出せば良いのである。

たとえば、ある図形をマウスで選択してから削除する場合を考える。図形データがマウスのクリックで選択されたとき、クライアントはその図形データオブジェクトへのポインタをクライアントの内部変数（たとえば選択オブジェクトと命名する）として格納しておく。次にキーボードのDELキーが押された時、内部変数が指す選択オブジェクトの削除メソッドを呼び出せば良い。削除処理自体はオブジェクトが自ら

行う。画面の再表示を行うときは、クライアントは貼り付けられているOLEオブジェクトのリスト構造をたどり、すべてのOLEオブジェクトの表示メソッドを順次呼び出す。

同様にクライアントが図形データを保存するときは、すべてのOLEオブジェクトの保存メソッドを順次呼び出す。これに応じてOLEオブジェクトは、クライアントから指定される記憶領域にデータを書き込むが、内部データ構造はクライアントからは見えない。

クライアントはOLEオブジェクトの貼られている位置や大きさ、記憶領域などだけを管理する。OLEオブジェクトに関する処理を実行するときは、そのオブジェクトのメソッドを呼び出すだけである。OLEオブジェクトの完全なカプセル化のもとで、クライアント/サーバー型で動作しているのである（図2.4-3）。

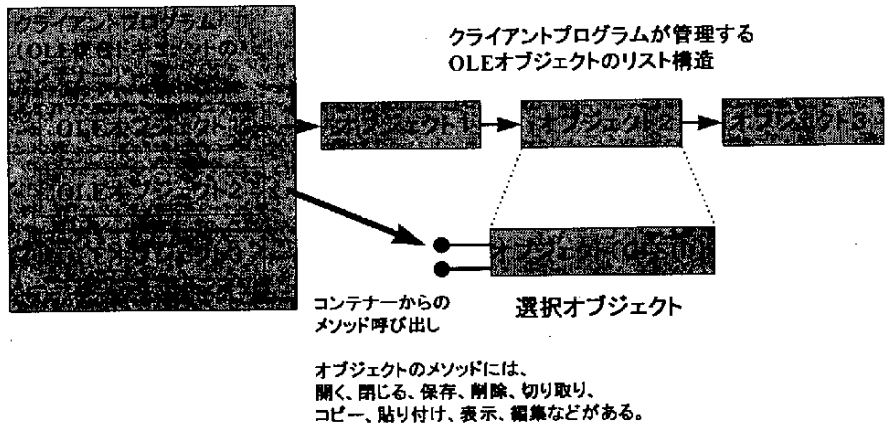


図2.4-3　オブジェクトのカプセル化およびOLEコンテナーからのメソッド呼び出し

(4) OLE対応機能の開発

OLEオブジェクトの処理がクライアント/サーバー型である利点は、特定のデータ（OLEオブジェクト）と、それを表示、編集する複合ドキュメントアプリケーション（クライアント）との関係を切り離せることである。OLEオブジェクトであれば、開発メーカーに関わりなくデータすべてを利用できることになる。

OLE機能の開発方法は大きく分けて、OLE SDK（Software Development Kit）による方法と、Visual C＋＋とMFC（Microsoft Foundation Class）ライブラリによる方法が存在する。OLE SDKによる開発は、標準のOLEインタフェースやMFCでは間に合わない拡張機能の開発、既存コードのOLE対応、Windows以外のプラットフォームとの共通コードによるマルチプラットフォーム対応、MFCなど特定クラスライブラリとの依存を避けたい場合などに用いる。

一方、VC＋＋とMFCライブラリは新規に製品を開発する場合、特にWindowsプラットフォームをター

ゲットとした製品では、OLE SDKに比べて開発効率およびOLE詳細部分のコード品質がきわめて高くなる。MFCのコード資産は、バージョンアップに際しても動作保証がある。

(5)主なOLE複合ドキュメントの機能

クライアント（OLEコンテナー）が装備するべきOLE機能のうち主なものは、複合ドキュメントの読み込み、保存、オブジェクトの挿入と作成、オブジェクトの表示と印刷、オブジェクトの起動と停止、オブジェクトの切り取り、コピー、貼り付け、削除、オブジェクトのドラッグ&ドロップ、マウスクリックによるオブジェクトの選択と起動、オブジェクトの矩形領域のハッチ処理などである。ハッチ処理は選択されたオブジェクトのある矩形領域を目立たせる処理である。

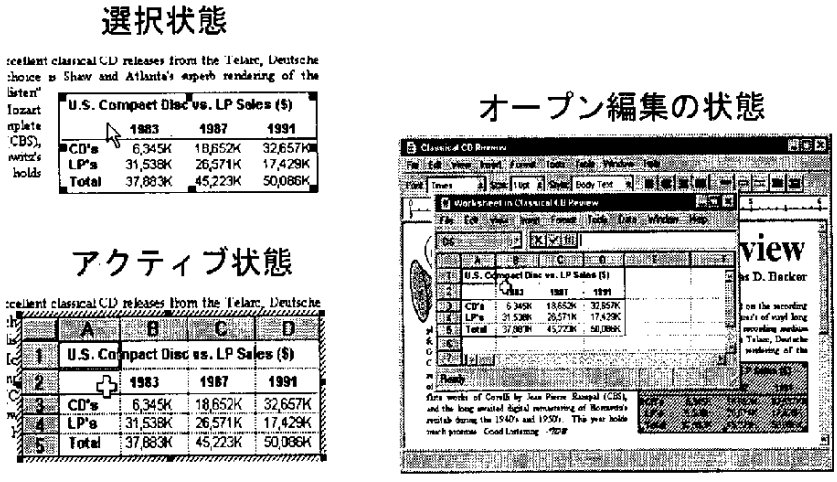


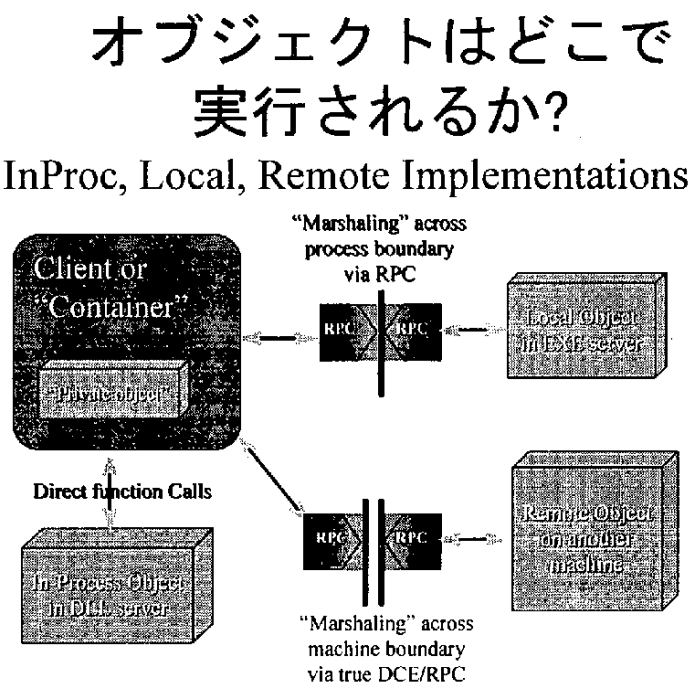
図2.4-4　オブジェクトのハッチ処理ー選択、アクティブ状態、オープン編集状態

また、インプレースアクティベーション機能として、コンテナーと起動オブジェクトのメニュー統合、コンテナーと起動オブジェクト間のツールバーの切り替え、起動オブジェクトのアクセラレータ（ショートカットキー）とフォーカス処理（選択したオブジェクトが目立つようにマウスカーソルなどの形を変える処理）、複数ドキュメントウインドウ間の切り替え、ドキュメントウインドウのリサイズ（サイズ変更）やスクロール処理、起動オブジェクトのヘルプや起動のUndo処理などが必要である。

オブジェクトを処理するOLEサーバー側の主なOLE機能は、OLE関連情報のレジストリ登録、オブジェクト保存のためのインタフェース、オブジェクトデータ転送のためのインタフェース、その他OLE関連情報を管理するためのインタフェース、コンテナーへの通知機能、OLEコンテナーのメニューやタイトルバーの変更処理などである。また、インプレースアクティベーション機能として、オブジェクトの起動と停止、メニュー統合、アクセラレータ処理などが必要である。

(6) OLEサーバーが無くても表示はできる

OLEサーバーには、DLL（ダイナミックリンクライブラリ）による形式とEXE（実行イメージ）による形式による開発方法が存在する。どちらの形式を採用するかはOLEサーバーの開発者の方針による。



〔クライアントプログラムからはその違いが区別されない。〕

図2.4-5　OLEサーバーの開発方法ーDLL形式とEXE形式

一般に、メニューやショートカットキーなど複雑なユーザーインタフェース、複数クライアントから起動されるオブジェクト間でのサーバー状態の共用、OLE 1.0との互換性、オブジェクトのリンクサポート、クライアントからのセキュリティ機能、を提供する必要がある場合はEXE形式のサーバーを用いる。ユーザーインタフェース機能を犠牲にしても、高速な処理を必要とする場合はDLL形式のサーバーを採用することになる。

OLE D&Mの場合、ユーザーインタフェース機能は犠牲にできないのでEXE形式のサーバーによる実装となるだろう。ただし、EXE形式のサーバーの場合であっても、表示処理は「オブジェクトハンドラー」というDLLが担当する。オブジェクトハンドラーはコンテナーと同一アドレス空間で動作するので、表示処理速度は犠牲とならない。またオブジェクトハンドラーが存在することにより、OLEサーバーが無くても複合ドキュメントに貼り込まれたOLEオブジェクトの表示データだけからOLEオブジェクトの表示は可能である。

OLEはオブジェクトハンドラーを図形オブジェクト作成時に作成しておく。オブジェクトの起動時には

オブジェクトハンドラーと起動オブジェクト間で通知コネクションを設定し、以降オブジェクトデータに変更があると、オブジェクトハンドラーに変更通知がなされるようにする。オブジェクトハンドラーは自分の持つ表示キャッシュを更新し、コンテナーに変更を通知するとともに表示キャッシュを渡す。こうしてコンテナーはデータならびに表示の変更を行う。

2．4．2　OLEオートメーション

OLEオートメーションは、あるアプリケーション（オートメーションクライアント）が他のアプリケーションコンポーネント（オートメーションサーバー）機能を操作する標準インターフェイスを提供する。すなわち、アプリケーション機能として利用可能なものは、機能を提供するアプリケーションコンポーネントに処理を任せてしまうということが可能となる。したがって、アプリケーションが実装するのは、他のコンポーネントが提供できない機能だけですむ。これは、オブジェクト指向技術の広い意味での差分プログラミング、すなわち、ソフトウェアの再利用を実現する。ソフトウェアの構築は、１つのオートメーションクライアントを中心とし、オートメーションサーバーを処理に合わせて組み合わせる手法となる。OLEオートメーションは、図2.4-6のように、オートメーションサーバーが外部のアプリケーションに対してオブジェクトの機能を見せることで実現される。

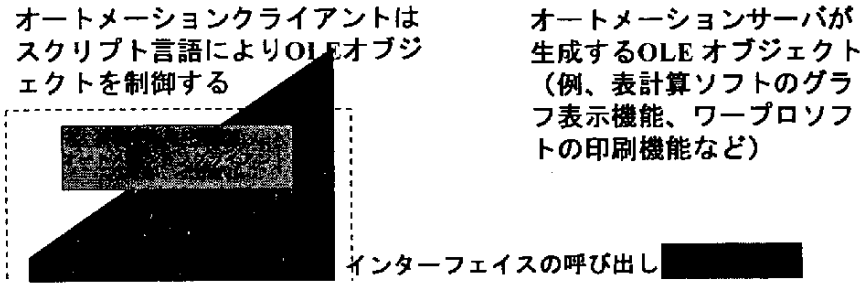


図2.4-6　オートメーションクライアントとサーバーの連携

外部のアプリケーション、たとえば、Visual Basicのようなプログラミングツールはオブジェクトの機能にアクセスし、繰り返しや条件判定のような通常のプログラム処理を使って、定型業務処理を記述することが可能である。また、これをアプリケーション付属のマクロ言語で記述することも可能である。OLEオートメーションサーバー内にあるオブジェクトは、外部に機能やデータを公開するためメソッドとプロパティを持っている。メソッドはオブジェクトにある動作を実行させるメンバ関数であり、プロパティはオブジェ

クトの状態の設定、読み取りを行うメンバ関数である。図2.4-7はExcelがもつ１３０余りのオブジェクトの主要なものを示す。このように、１つのアプリケーションExcelはオブジェクトの集合体から構成されており、ソフトウェアコンポーネントの観点からは１つのアプリケーションとして実現される必然性は必ずしもなく、近い将来、バージョンアップを続けて肥大化するアプリケーションやシステムは、より小さいオブジェクトとして再構築されるだろう。

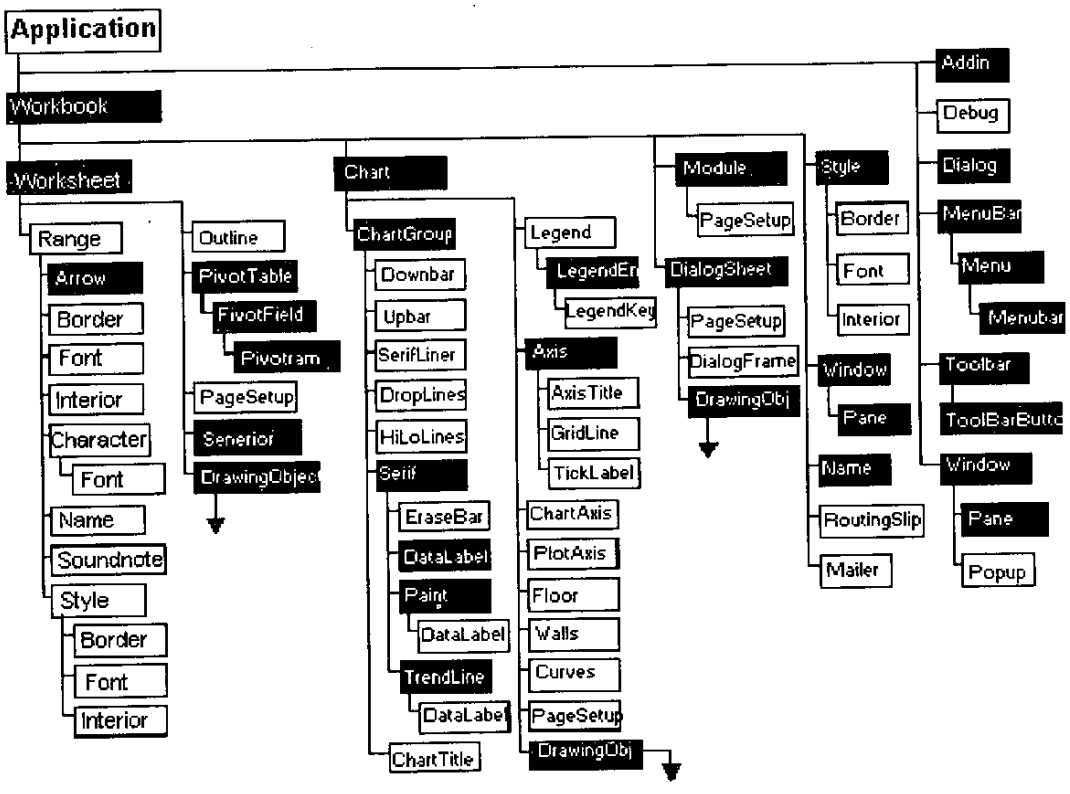


図2.4-7　Excelのオブジェクト構造

2．4．3　OLEコントロール

高機能なソフトウェアコンポーネントは従来よりVisual Basicカスタムコントロール（VBX）で実現され、多くの優れたコンポーネント製品が提供されている。OLEコントロールはVBXの機能を受け継ぎ、さらにVBXに比べ以下の優れた機能を提供する。

(1) コンテナーとコントロール間の標準インターフェイス（OLE 2.0）

OLEコントロールを埋めこむコンテナー（OLEクライアント）とOLEコントロール間の相互作用はすべて標準OLE 2.0インターフェイスで実現されている。OLEコントロールは、コンテナーと同一プロセスで動作しユーザが両者の見た目や操作性の違いを意識しないようにさせている。

(2) Windows 95、Windows NTへの32ビット対応

VBXが16ビットであるのに対して、OLEコントロールは16/32ビットいずれにも対応する。また、Macintoshなど他のプラットフォームでのサポートも予定されている。

(3) 複数開発環境によるサポート

VBXがVisual Basic開発環境でだけサポートされるのに対して、OLEコントロールは今後出荷されるすべてのOLEコンテナーでサポートされる。

(4) カスタムメソッド

VBXはカスタムメソッドをサポートしないので不満が多かったが、OLEコントロールはカスタムメソッドをサポートする。カスタムメソッドは画面再表示などの標準メソッド以外のメソッドを追加し、コンポーネントのカスタマイズを可能とする機能である。

(5) コンテナーとのUIネゴシエーション

OLEのビジュアルエディティングでは、たとえば、コンテナーとコントロールのもつメニュー項目の融合、コンテナーとコントロールのヘルプ切替え、コントロールウィンドウの移動や大きさの変更、画面スクロールなどがサポートされる。また、OLEコントロールにはストックプロパティと呼ばれる標準プロパティがあり、コントロールの背景色、ウィンドウ境界のスタイル、キャプション、フォント、フォントの色などのプロパティをコントロールの外部から簡単に操作できる。たとえば、プロパティページと呼ぶダイアログを使用して、グラフィカルにプロパティの設定の変更が可能である。これらに加え、OLEコントロールにはコンテナーがコントロールに対してプロパティ情報を伝達する機能がある。たとえば、コンテナーのフォームの背景色やフォントの変更に合わせて、コントロールの背景色やフォントの設定を同一にしたい場合に使用される。これにより、コンテナーのフォームとコントロールの外見上の違いをなくし、両者をシームレスに見せることが可能となる。

(6) コントロール記述のためのタイプライブラリ

外部のアプリケーション、マクロ言語やツールがコントロールにアクセスし、その機能を操作するためには、コントロールがもつOLEオートメーションのメソッドとプロパティを記述するタイプ情報が必要となる。タイプ情報はObject Description Language（ODL）で書かれ、コンパイルされてタイプライブラリとなる。OLEコントロールがロードされる時、まず、コンテナーはそのタイプライブラリを読み取り、オブジェクトを生成して、そのOLEインターフェイスと通信する。

(7) 柔軟性のあるライセンシング

カスタムコントロールがバイナリで流通するソフトウェアコンポーネントとして成功するには、柔軟なライセンシングスキームが重要となる。VBXはライセンスファイル（.LIC）を添付して、常時あるいは実

行時のみコントロールの作成が可能となるような機能が提供される。OLEコントロールは、これに加え、設計時のみコントロールを作成できる機能があり、開発者向けの設計時ライセンスとエンドユーザ向けの実行時ライセンスの区別ができる。

2．4．4　コンポーネントウェアの構築例

以下ではコンポーネントウェアの構築例を2つ紹介する。

(1) OLEオートメーションとOLEコントロールによるデータベースアクセス例

Visual Basic V4.0には以下のようなOLEオートメーションとOLEコントロールを使用したデータベースアクセス機能がある。

(a) OLEオートメーションによるデータベースアクセス

Visual Basic V4.0にはデータアクセスオブジェクト（DAO）と呼ばれるOLEオートメーションを利用したデータベースアクセス機能がある。付属のJetエンジン（Microsoft Accessのデータベースエンジン）の機能をオブジェクト化し、そのプロパティやメソッドを操作することでデータベースへのアクセスが可能である。

(b) OLEコントロールによるデータベースアクセス

Visual Basic V4.0はOLEコントロールを埋め込むコンテナー機能を持つので、OLEコントロールのもつ32ビット対応、カスタムメソッド、コンテナーとのUIネゴシエーションなどの拡張機能が利用できる。すなわち、Visual Basic V4.0にデータベースアクセス用のOLEコントロールを貼りつけ、OLEコントロールのプロパティやメソッドを操作するユーザインターフェイスを通じて、データベースのデータ表示や更新を行える。実際のアプリケーションでは、他の機能を持つOLEコントロールと組み合わせて画面設計され、OLEコントロールが他のユーザインターフェイスのコンポーネントと区別されることはない。すなわち、ユーザはまったくOLEコントロールの使用を意識しない操作が可能である。

(2) OLEオートメーションによるメッセージングソリューション例

メッセージングアプリケーションをコンポーネントウェアの考え方で再構築すると、ユーザインターフェイスを処理するメッセージングクライアントと、サービスプロバイダとして、メッセージ送受信（メッセージトランスポートプロバイダ）、メッセージ保存と読み取り（メッセージストアプロバイダ）、メッセージアドレス管理（アドレスプロバイダ）の各コンポーネントから構築される。ここで、メッセージングクライアントとサービスプロバイダ間の標準インターフェイスを定義し、メッセージングクライアントとサービスプロバイダの相互依存性を標準化することで、複数のメッセージングクライアントがサービスプロバイダを共有し、サービスプロバイダの各コンポーネントの選択、置き換えが可能となる。標準インターフェ

イスは、従来のAPI（Application Programming Interface）ベースの他に、グループウェア機能などを外部のアプリケーションに提供するためのOLEオートメーションのインターフェイス（OLE Messaging）、主にデータ表示のためのユーザインターフェイスコンポーネントであるOLEコントロールの３つが考えられる。

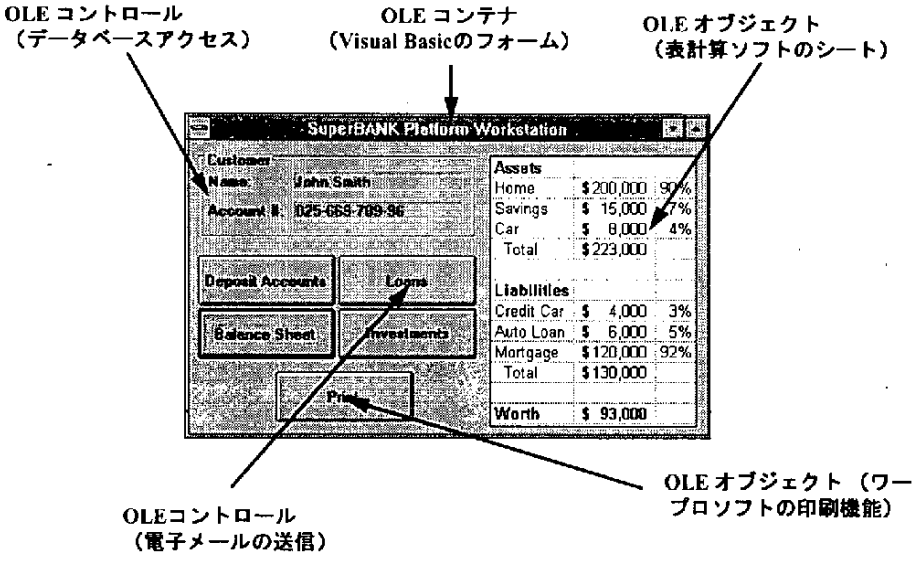
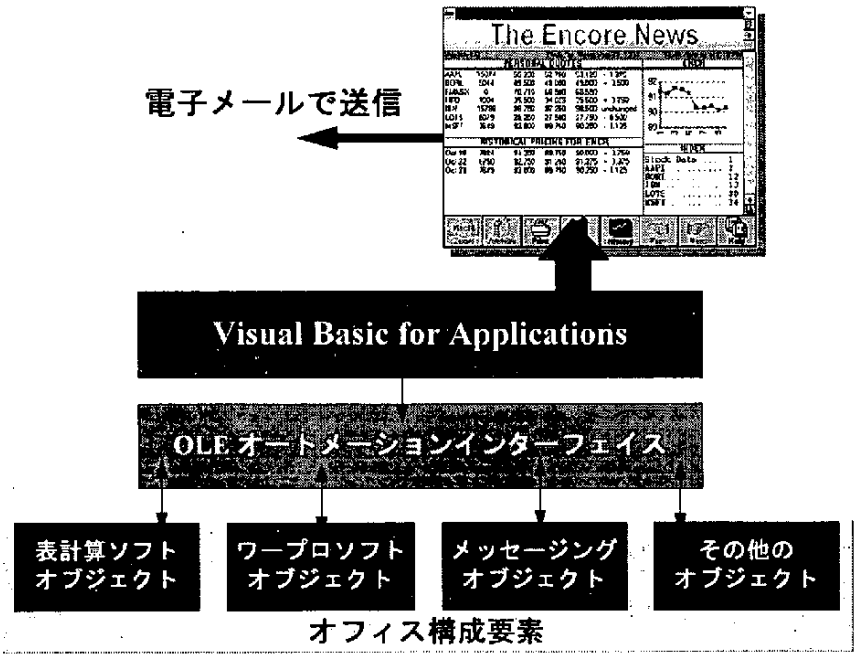


図2.4-8　OLEコントロールを使用したデータベースアクセスの例



〔報告書作成とメール送信の自動化〕

図2.4-9　OLEオートメーションを使用したメッセージングソリューション例

OLEオートメーションには、AddressEntry（メッセージアドレス指定）、Attachment（添付文書）、Folder（メッセージフォルダ）、Message（メッセージ）などのオブジェクトがある。たとえば、ワープロ文書で作った月例報告書に、表計算ソフトでデータベースから読み込んで作った月間売り上げデータグラフを添付し、それを電子メールで送信する業務作業を毎月末に自動的に行うような場合を考える。以下のように、OLEオートメーションを利用したVisual Basicの簡単なプログラムを作ることで、従来のAPIベースの開発に比べ飛躍的な効率で定型業務の自動化が可能となる。（図2.4-9）

(a) ワープロソフトにOLEオートメーションで文書の読み込みを要求

(b) 読み込んだ文書をMessageオブジェクトのTextプロパティに設定

(c) 表計算ソフトにOLEオートメーションでデータベース検索、結果をグラフ化、グラフの保存を要求

(d) 保存したグラフをAttachmentオブジェクトのReadFromFileメソッドで読み込む

(e) MessageオブジェクトのSubjectプロパティに報告書の主題を設定

(f) AddressEntryオブジェクトのAddressプロパティに送信先アドレスを設定

(g) MessageオブジェクトのSendメソッドを呼び出し、電子メールを送信

以上の２つの例から、実際のアプリケーション構築で、どのオブジェクトのインターフェイスを使用すべきかは設計に依存するが、OLEによるコンポーネントウェアの構築はアプリケーションの柔軟性を高め、カスタムアプリケーションの構築、アプリケーションの保守や拡張が容易となることが分かるであろう。

２．４．５　OLEによる業種別オブジェクト

OLEによる業種別オブジェクトは、ハードウェア、ネットワークプロトコル、文字セットなどのシステムインフラ部分の違いをオブジェクトのカプセル化機能を使用して共通化し、共通インフラ上に汎用アプリケーションの構築を可能とする。また、市販ソフトウェアのコンポーネントの既存機能（報告書作成、グラフ表示、印刷など）はできるだけ利用して、各業種別機能だけを共通コンポーネント化していく。たとえば、金融証券市況データのグラフ表示には市販の表計算ソフトのグラフ表示機能をコンポーネントとして使用し、市況データの解析、予測のような業務に特化した機能のコンポーネントを開発することで、コンポーネントの組み合わせによるディーリングシステムを構築することが可能となる。データベースアクセスのような汎用コンポーネントでは、データ検索やデータエントリの機能差が業務に影響することはそれほど多くはない。しかし、業務に特化するコンポーネントは、先の市況データの予測アルゴリズムのように、その違いがディーリング業務の競争力の違いに直接結び付く。したがって、コンポーネントウェアによる機能共通化の流れは、新たにコンポーネントのもつ特異性、質的競争力という次元での差別化を求めることになる。業種別オブジェクトのOLE仕様として、現在までに表2.4-1が提供または検討されている。

表2.4-1 現在までの提供または検討されている業種別オブジェクトのOLE仕様

名称	業種、適用分野と利用するOLE技術	仕様製品化の状況（米国）	仕様製品化の状況（日本）
OLE for Real Time Market Data	ディーリング端末での証券市況データの転送管理や加工をOLEオートメーションで統合	V1.01仕様および製品化予定あり	V1.01仕様検討中
OLE for Extended Kanji Processing (XKP)	外字を含む拡張漢字処理が必要となる地方自治体向けシステムを汎用ツール、アプリで構築するためのOLEコントロールを作成	仕様を英訳中（95年10月予定）	V1.0仕様をNT3.51ベースに拡張中（96年2月V2.0予定）
OLE for Retail POS	レジおよび周辺デバイスをOLEコントロールでサポートし、汎用アプリと統合	V0.91仕様	検討中（96年3月仕様完成予定）
OLE for Design & Modeling	3次元CAD/CAMにおける図形オブジェクトの編集、表示をビジュアルエディティングで行う	95年5月が最新仕様。製品あり	検討中（完成未定）
OLE for Process Control	OLEオートメーションやOLEコントロールを使用したプロセス制御やデバイスの監視	検討中	未定
WOSA/XFS(Windows Open Services Architecture/Extensions for Financial Services)	金融端末での入出力装置の標準インターフェイスを提供(APIベースからOLEインターフェイス化される)	V2.0仕様検討中	V1.0仕様および製品
DMI/MIF(Desktop Management Interface/Management Information Format)	PC、サーバー、プリンタなどのシステム管理の標準インターフェイス(APIベースからのOLEインターフェイス化される)	各社が製品提供または予定	各社が製品提供または予定

このうち、OLE for Real Time Market DataとOLE for XKPの概要を以下で紹介する。

(1) OLE for Real Time Market Data

OLE for Real Time Market Dataは、金融証券市況データごとにネットワークプロトコルや手順が異なる提供会社（フィーダ）からリアルタイムデータを受信し、データの表示や加工を汎用アプリケーションのコンポーネントの機能を組み合わせて実現するシステムである。さらに、提供情報に対する信頼性、セキュリティ機能も求められる。OLE for Real Time Market Dataは、従来から考えられていたオブジェクト指向の処理速度の問題で以下の性能目標を掲げ、リアルタイムシステムへの適用を可能としている。

(a) データは株式、債権、先物などの価格と出来高の情報、ニューズヘッドラインと修正データ

(b) ピーク時、約４００メッセージ/秒。各メッセージ長は約６４バイト

(c) データ変更要求あるいはデータ紛失が通知される。データ転送では、変化した差分のみを転送する場合と最新データのみを転送する場合がある

OLE for Real Time Market Dataはオブジェクト転送技術とOLEオートメーション機能を使用し、転送オブジェクトのデータ形式プロパティを実行時に設定することが可能となっている。（図2.4-11を参照）現在、V1.01の仕様とサンプルコードが公開されており、日本でもこれに合わせて現在４９社からなるオープンマーケットデータ協議会が検討を開始したところである。

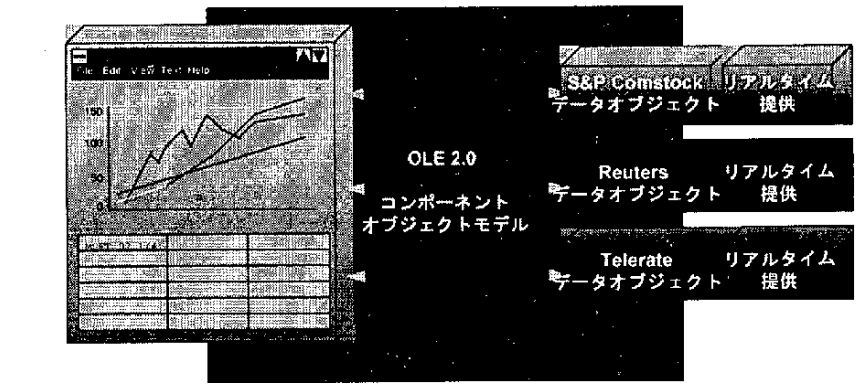


図2.4-10　OLE for Real Time Market Dataのシステム構成

```
Dim XRTObj as Object　　変数をobject型に宣言
Sub Start()
  XRTObj = CreateDataObject("Quotron.Equity.1")　データオブジェクト生成
  props = XRTObj.Requests.CreateProperties("High", "Low", "Last")　転送するデータ形式プロパティの設定
  XRTObj.Requests.Add("MSFT",props)　転送データの指示１（MSFT社のデータ）
  XRTObj.Requests.Add("LOTS", props)　転送データの指示２（LOTS社のデータ）
  n = 1
  For Each Stock In XRTObj.DataItems　転送データを表計算ソフトのセルに張り込み表示
    Cells(1, n) = Stock.Properties("Name")
    Cells(2, n) = Stock.Properties("Last")
    n = n + 1
  Next
End Sub
```

図2.4-11　OLE for Real Time Market Dataのサンプルコード

(２) OLE for XKP（Extended Kanji Processing）

OLE for XKPは、地方自治体および多くの拡張漢字機能を必要とする業種のアプリケーションへの適用を考えている。OLE for XKPは、OLEコントロールを使用し、PCによるクライアント／サーバーシステムで従来のShift JIS文字コード体系では困難であった１８，０００以上の文字をWindows NT上でサポートする。その主な特徴は以下のとおりである。

(a) JIS X0221-1995（ISO/IEC 10646-1:1993, Unicode V1.1）文字コード体系をOSで利用する方法を定めた最

初の仕様

(b) PCによる戸籍/住民基本台帳業務系システムの構築および従来の専用システム間との漢字データ交換

(c) 既存のShift JISアプリケーション（ワープロや表計算ソフト）でJIS X0221-1995文字コードの表示や印刷が可能

OLE for XKP仕様書には、システムに搭載される文字に関する内字の文字セットの規約、外字処理メカニズム、登録された外字を含めた文字入力方法、JIS X0221-1995文字を表示するOLEコントロールのプロパティ、イベント、メソッド、およびデータベース上の文字データの５項目を規定している。

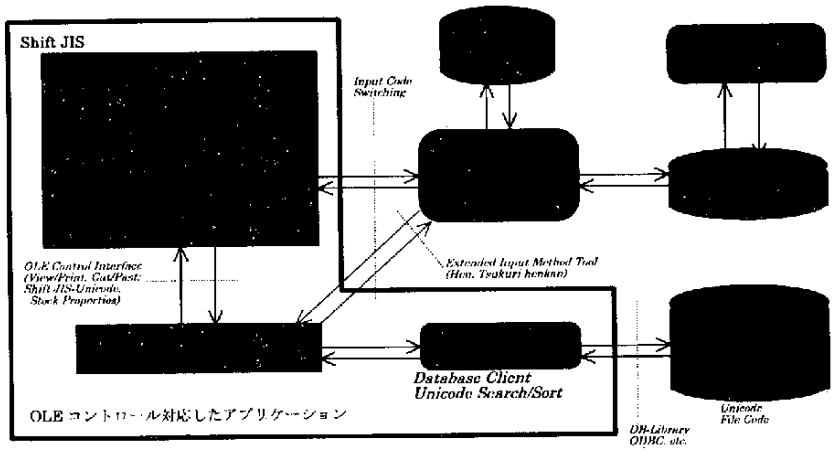


図2.4-12 OLE for XKPのシステム構成

2．4．6 OLEの今後

PCを使用した企業情報システムの構築は、安価で高機能な市販のパッケージソフトを有効利用しつつ、カスタマイズをできるだけ低コストで行うことが重要である。また、高機能化するソフトウェアは同時にその肥大化がシステムに大きな負担を課し、投資コストを増大させる。OLEはこれらのさまざまな問題を解決し、将来までシステム投資を保護する優れた基盤技術である。以下にOLEの主な特長とOLE利用方法の要点をまとめる。

(a) 市販パッケージソフトの機能をコンポーネント単位で有効利用する

(b) カスタマイズでは必要機能のみをコンポーネント化する

(c) 厳格なインターフェイス定義とバージョン管理を利用し、コンポーネント間の連携を組み立てる

(d) コンポーネント単位の機能選択、置き換えが可能で必要なコンポーネントのみをシステムに搭載する

従来のオブジェクト指向プログラミング言語に比べ、OLEはOS、言語などのプラットフォームに依存

しないコンポーネントを構築でき、分散化機能でマルチベンダ、マルチプラットフォームを実現する。ネットワークプロトコルに依存ぜず、RPCやトランザクション処理などの分散処理技術との融合がこの段階で進んでいく。したがって、OLEを複合ドキュメントなどのユーザインターフェイス改善という考え方から、分散サービス抽象化の新しい基盤技術と見ていくことがこれからは重要となるだろう。

【参考文献】

OLEの詳細は以下の情報ソースから得ることができる。

1) Inside OLE

2) Microsoft Systems Journal

3) VC++ CDKマニュアル

4) MSDN（MicroSoft Developer's Network） CD-ROM

5) www.microsoft.comまたはftp.microsoft.com

6) OLE Programmer's Reference

2．5　OpenDoc

2．5．1　OpenDocの目的

エンドユーザが日常行う作業を支援するためのソフトウェアとしてワードプロセッサー、表計算ソフト、ドローイングソフトなど様々なものが開発されてきている。こうしたソフトウェアは、ユーザの様々な機能拡張、機能追加要求に応えるためにバージョンアップが繰り返され、その開発維持や利用が年々困難になってきている。また、ソフトウェアのアプリケーションインタフェースやデータ形式には互換性がないものが多く、組み合わせて利用することが難しかった。

OpenDocは、従来の単一アプリケーションですべてを行うという方法から、機能に対応する小さなコンポーネント（Component）が協調して動作する環境を提供しようとするものである。すなわち、一つのドキュメント内の様々な中身（コンテンツ）ごとに編集モジュールを対応させるという方法である。こうすることにより、最初からすべての機能をもったアプリケーションを作る必要がなく、漸増的開発、アプリケーションの早期実現などが可能となる。

OpenDocにより、ユーザはニーズに合った様々なデータタイプからなる複合ドキュメント（Compound Document）を作ることができる。

OpenDocを用いることにより、アプリケーションの開発者は、コンポーネントを単独で提供したり、いくつかのコンポーネントをパッケージとして提供したりすることができる。後者は従来のアプリケーションに似ているが、ユーザは、いくつかのコンポーネントを好みのものに替えることができる点で異なる。

OpenDocは、１９９３年に、Apple、IBM、Novel、Oracle、Tarigent、SunSoft、WordPerfect、Xeroxの参加の基に作られたコンソーシャムであるCI Lab（Components Integration Lab）によって開発が進められ、１９９５年１１月にバージョン1.0が出荷された。このバージョンは、AppleのMacOSの上で稼働するものであるが、今後AIX、OS/2、Windows用のバージョンが計画されている。

2．5．2　パート

(1) ドキュメント、パート、埋め込み

OpenDocの基本要素は、ドキュメント（document）、パート（Part）、フレーム（Frame）とパートエディタ（Part editor）である。クラスライブラリーを通して、これらのコンポーネントはユーザインタフェースリソース、ドキュメントレイアウトの折衝、蓄積されたコンテナの共有、データリンクの生成を行う。

ドキュメント（アプリケーションでない）は、OpenDocの中心となるものである。OpenDocのドキュメントは、パートエディタと呼ばれる小さなソフトウェアコンポーネントによって作成される。各パートエディタは、限られたデータタイプのみ扱うことができる。こうしたパートエディタは、いちいち起動させ

る必要がない。

OpenDocと従来のアプリケーションには次のような違いがある。

・OpenDocでは、好みのパート例えばグラフィックパートなどを用いてドキュメントを構成できる。

・ドキュメント内に新たなパートを追加できる。

・データを処理するために新たなウインドウを開く必要がない。ドキュメント内で処理できる。

・ユーザはパートエディタを取り替えることができる。

パートエディタは、パートを表示、編集、蓄積する能力を持っている。実行時においては、パートはオブジェクト指向プログラミングでのオブジェクトと同じであり、状態（Stare）と振舞い（Behavior）をもつ。パートデータは状態であり、パートエディタは振舞いである。パートごとにパートエデイタをロードする必要がなく、メモリ内の一つのパートデータは複数のパートを扱う。

OpenDocは、実行時にパートとパートデータの種類に応したパートエディタをリンクさせる。パートビューワ（Part Viewer）は、パートエディタの特殊な形態で、表示や印字はできるが、パートの生成や編集を行うことができない。

OpenDocでは、エディタやビューワに加えて、パートの表示、蓄積、編集を行わないサービス（Service）と呼ばれる特殊なコンポーネントがある。サービスは、パートやドキュメントにソフトウェアサービスを提供するもので、スペルチェックやデータベースアクセスなどがその例である。ドキュメント内でパートは階層的に編成される。

パートはフレーム内に表示される。フレームは、パートの内容を表示するために囲まれた領域である。表示フレームは他のパートの中に埋め込まれる（Embedded）こともある。図2.5-1に含有パート（Containing part）と埋め込まれパートの関係を示す。内側のフレームは、外側パートの埋め込まれフレームである。また、外側パートは、内側パートの含有パートである。

埋め込まれパートは、論理的に含有パートに埋め込まれただけで、含有パートは、埋め込まれパートの特性に関与しない。含有パートは、埋め込まれパートを移動、選択、削除することができる。埋め込まれパートは、埋め込まれフレーム内の描画、イベント操作を行う。

すべてのドキュメントは、最高レベルとしてルートパートをもち、他のすべてのパートはルートパートに埋め込まれる。ルートパートは、ドキュメントの基本レイアウト構造とプリントの振舞いを決める。すべてのパートは、他のパートの中に埋め込まれることができし、ルートパートになれる。ただし、時計のようなパートはルートパートとしてふさわしくない。こうしたパートは、非コンテナパート（Noncontainer parts）と呼ばれる。一方、埋め込んだり、埋め込まれたりすることのできるパートはコンテナパートと呼ばれる。

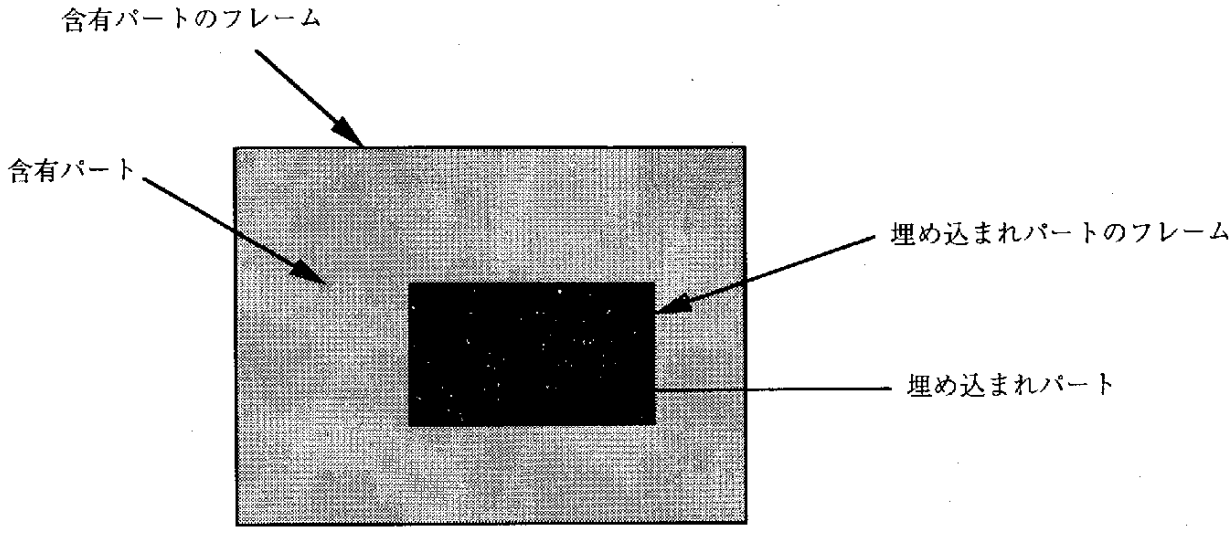


図2.5-1 含有パートと埋め込まれパートの関係

(2) データタイプ

OpenDocは、パートとパートエディタを結びつけるためにファイルタイプに似たパート種類（Part kind）という概念を持っている。OpenDocのドキュメントは、一つのアプリケーションに対応付けることはできないのでファイルタイプという概念は適用できない。ドキュメント内のパートは、各々固有のファイルタイプを必要とする。パート種類は、ISO文字列で表現される。エディタによって作成されたパートはパート種類をもつ。しかし、パートが作成したエディタのみでしか参照でないのでは、利用を甚だしく制限するためパートカテゴリー（Part Category）と呼ばれるパートエディタによって操作できるデータ種類の一般的記述を使ってパートエディタを代用できるようにしている。ユーザは、パートカテゴリーに対して暗黙のエディタを指定することができる。

ユーザがデータをペーストしたとき、データのパート種類あるいはパートカテゴリーとペーストする先のパートのパート種類あるいはパートカテゴリーが比較され、ペーストする先のパートの中に組み込まれる（Incorporate）か、埋め込まれパートになるか決定される。判断は次のように行われる。

・ペーストされるデータのパート種類がペーストする先と同じときは、データは組み込まれる。

・パートカテゴリーが異なるときは、ペーストされるデータは埋め込まれパートとなる。

・パートカテゴリーが同じで、パート種類が異なるときはペーストする先のパートは組み込むように変換

を試みる。変換できないときは、埋め込まれパートとする。

パートを作成したエディタが無い場合、OpenDocは、パート種類、パートカテゴリーの対応した暗黙のエディタを割り付ける。エディタを変更することにより、情報の損失が発生することがある。

(3)パートの表示

OpenDocは、フレーム間のジェオメトリックな関係の管理とプラットフォームに依存したグラフィックシステムへのラッパーを提供している。グラフィックシステムへのラッパーオブジェクトはキャンバス（Canvas）と呼ばれ、ジェオメトリックな形状を記述するオブジェクトはシェイプ（Shape）と呼ばれる。オフセット、拡大／縮小、回転などの2次元変換のためのオブジェクトはトランスフォーム（Transform）と呼ばれる。

キャンバス上にフレームの中身（全体あるいは一部）を表示するためにファセット（Faset）と呼ばれるオブジェクトを使う。フレームは、表示する中身のジェオメトリック関係の制御を行い、オフセットは、含有フレームあるいはウインドウへのフレーム間のジェオメトリック関係の制御を行う。すなわち、フレームオブジェクトは、含有パートと埋め込まれパート間でのスペースレイアウト調整のために使われる。ファセットオブジェクトは、表示されている内容に対するイベント処理などに使われる。

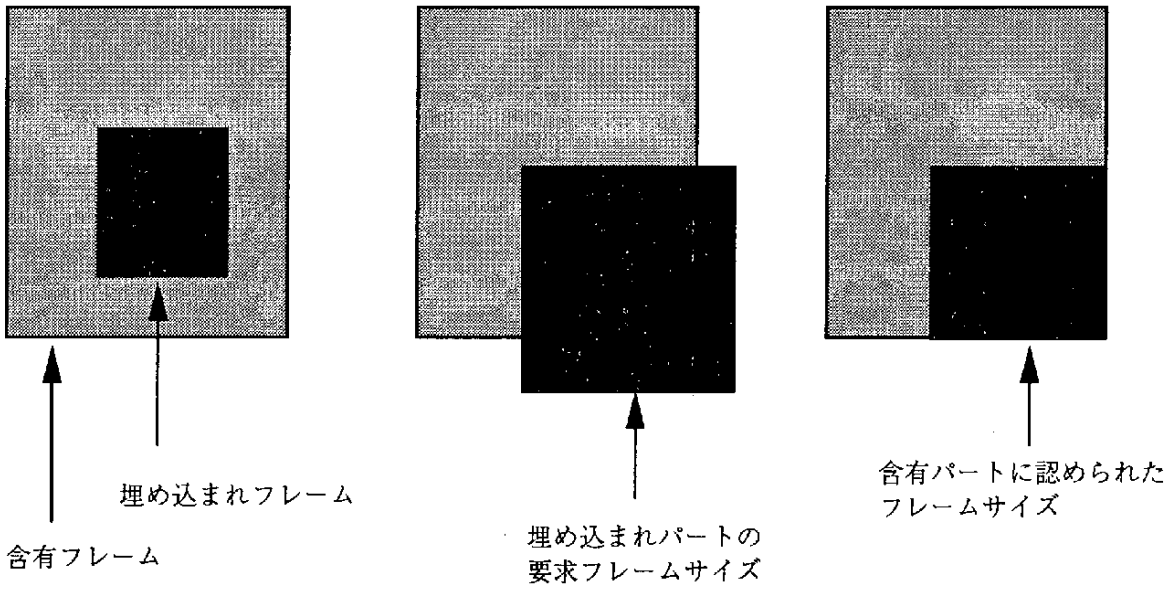


図2.5-2　フレームネゴシエーションの例

OpenDocでのウインドウ表示は通常のアプリケーションと同様であるが、特徴としてプレース内編集（In-place editing）という機能を実現している。ユーザは、埋め込みの深さに関係なく、パートを表示しているフレーム内で直接編集することができる。無論、別ウインドウ（Part Window）として表示し編集することもできる。

埋め込まれパートがサイズを変更したり、他のフレームを付け加えようとする場合は、含有パートとのネゴシエーションを行う必要がある。図2.5-2にフレームネゴシエーションの例を示す。含有パートは、埋め込まれフレームのサイズや形に対する絶対的な制御権を持っている。

2．5．3　イベント操作

多くのパートエディタは、ユーザイベントを介してユーザと相互作用を行う。ユーザイベントとは、ユーザからのアクション、パートやウインドウの活性化、非活性化、他のイベントソースからのメッセージによってオペレーティングシステムから送られるメッセージである。ユーザイベントによって、パートエディタは、パートの再描画、ウインドウの開閉、編集操作などを行う。

OpenDocでは、すべてのユーザイベントをOpenDocがまず受け、その後各パートエディタに渡す方法をとっている。これは、パートエディタが完全なアプリケーションでなく、ある環境の上で実行されていることによる。この環境をドキュメントシェル（Document Shell）という。

OpenDocは、ドキュメントがオープンされるとドキュメントシェルのインスタンスを生成し、ドキュメントウインドウで必要となるすべてのパートエディタをメモリにロードする。そして、各パートエディタはパートのデータを読み込む。ドキュメントシェルは、イベントの位置やメニューのような共有リソースのオーナーを探し、該当するパートエディタにイベントを渡す。

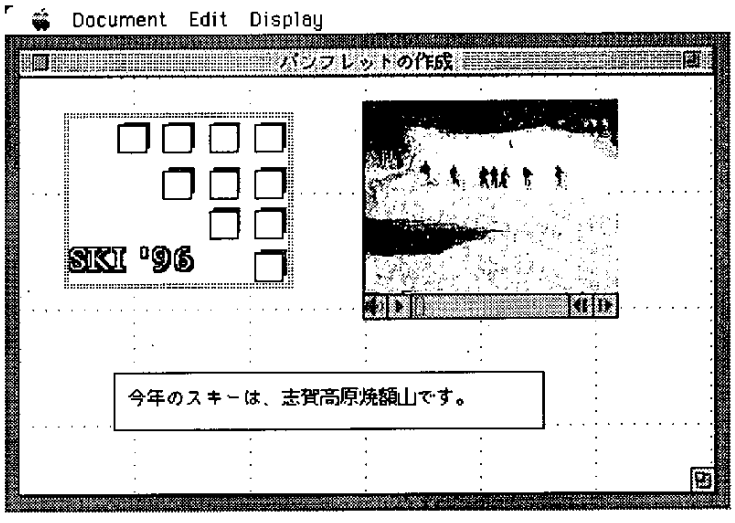
パートはセレクションフォーカス（Selection Focus）を持ったとき編集可能になる。この状態で、パートは更にメニューやキー入力のフォーカスを持つ。

パートが活性化された状態と選択された状態では、その操作が異なっている。パートが選択された状態では、そのフレームの移動、削除、複製などが可能となる。これは、埋め込まれフレームは、含有パートの要素としてみられるからである。パートが活性化された状態では、該当するパートエディタによる操作が可能となる。図2.5-3に埋め込まれパートの各状態を示す。図2.5-3で、左側の図は非活性状態、中央の図は選択状態、右側の図は活性状態を示す。

パートが活性化されると、境界線の形状が変わる以外にも変化が生じる。図2.5-4にパートの活性状態と非活性状態の変化を示す。

図2.5-3　埋め込まれパートの各状態

(1) グラフィックパートがアクティブ



(2) テキストパートがアクティブ

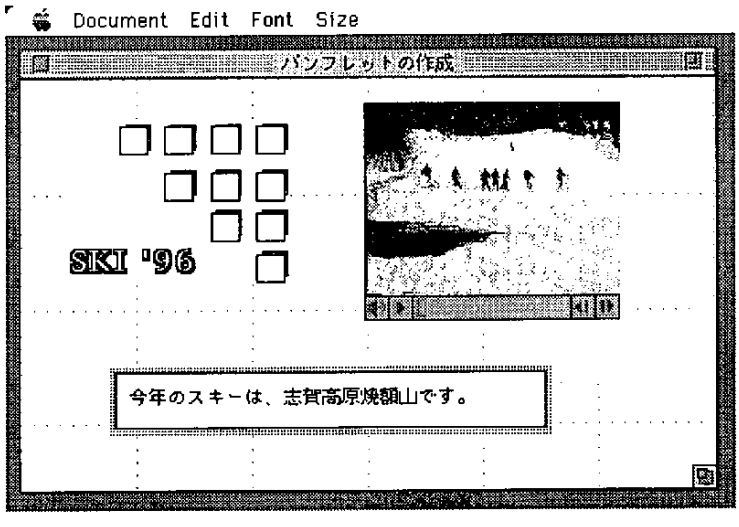


図2.5-4　グラフィックパートの活性状態と非活性状態の変化

グラフィックフレーム内でのマウスクリックなどによりグラフィックが活性化されると、OpenDocはグラフィックフレームの回りにその部分が活性化されていることを示す印を付け、メニューバーにグラフィックエディタのメニューを表示する。この後、テキストパートを活性化すると、グラフィックパートのメニューは消え、テキストパートが、自分のメニューを表示する。

フレームを活性化するのにOpenDocは、インサイド・アウト活性化モデルを使っている。このモデルでは、フレームが入れ子状態になっているとき一番内側のフレームを活性化させる。

2．5．4　ストレージとデータ転送

(1)ストレージ

OpenDocのストレージシステムは、複数のパートエディタが一つのドキュメントを一貫性を保ちながら共有を可能にさせるものでベントウ（Bento、弁当）と呼ばれている。ベントウは、オブジェクト指向データベースシステムではなく、各ファイル内にファイルシステムをもつような構成になっている。

OpenDocストレージシステムは、コンテナ（Container）オブジェクトから構成され、各コンテナオブジェクトは複数のドキュメントオブジェクトをもち、ドキュメントオブジェクトは複数のドラフトオブジェクトを持つ。更に各ドラフトオブジェクトは、複数のストレージユニットオブジェクトをもつ。この関係を図2.5-5に示す。

ドキュメントオブジェクトは、パートやフレームなどに関する情報をもち、ドラフトオブジェクトは、ある時点でのドキュメントの状態をもつ。ドラフトによりドキュメントのバージョン管理を行うことができる。ストレージユニットオブジェクトは、データストリームを蓄積しているところで、各ストレージユニットは、複数のプロパティ（Property）をもち、各プロパティは複数のバリュー（Value）をもつ。バリューは生のバイトデータが入るところである。この関係を図2.5-6に示す。

プロパティ内の各バリューは、同一データの異なった表現形式である。例えば、ある同じ文字列がプレーンテキスト表現とビットマップ表現として蓄積され、必要に応じて一方が使われる。

バリュー内には、他のストレージユニットへの参照を保持できるため、ストレージユニットを階層構造として使うことができる。実際、OpenDocでは、ドキュメントの埋込構造をストレージユニットを使って構成している。

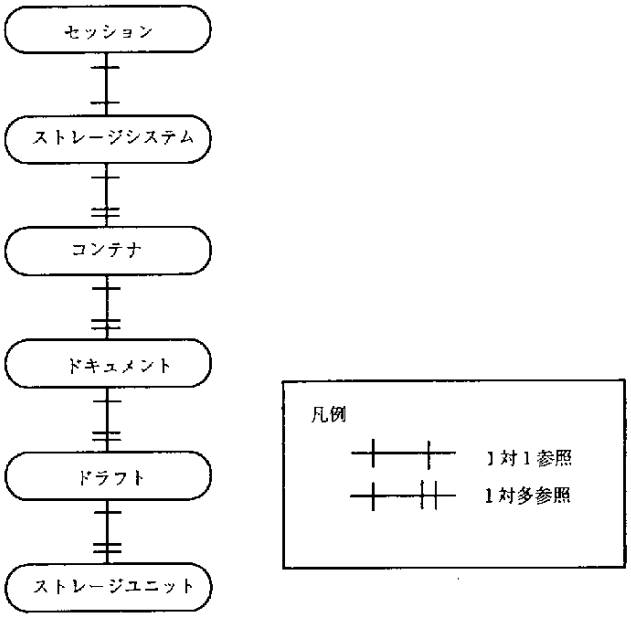


図2.5-5　ストレージシステム・コンテナ・ドキュメントの関係

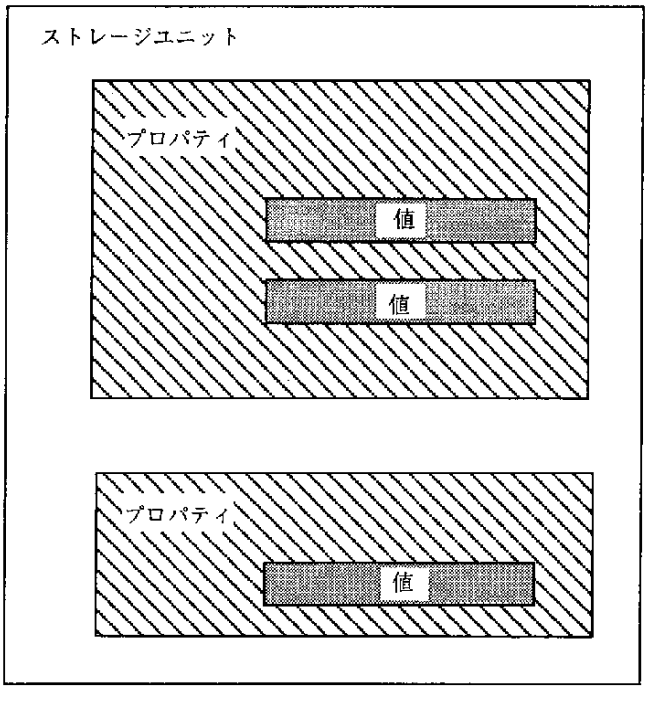


図2.5-6　ストレージユニットの構成

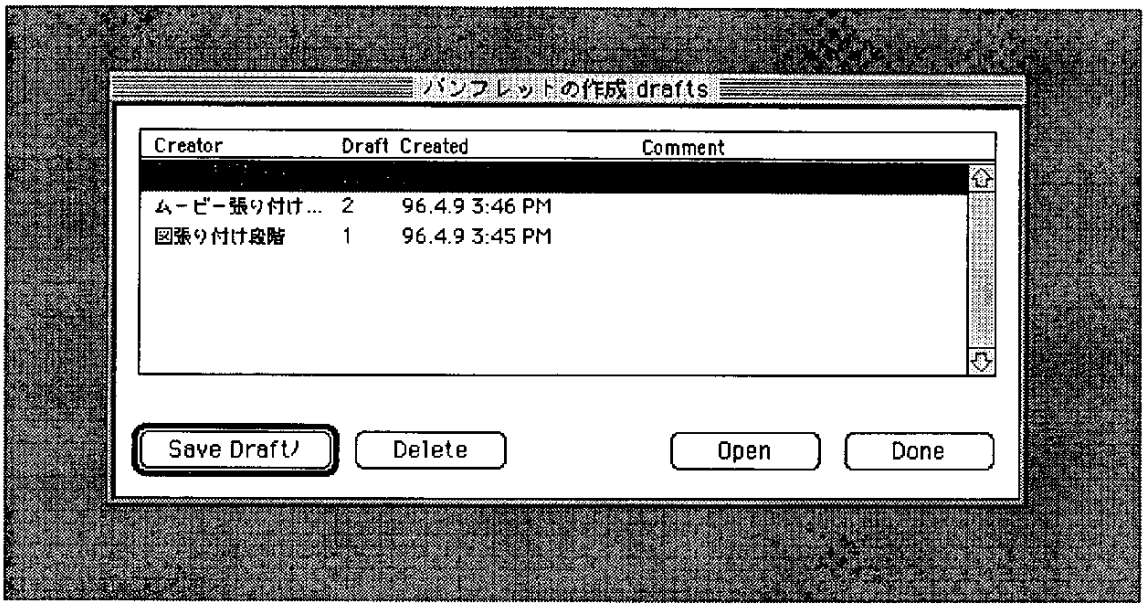


図2.5-7　ドラフト作成時のメッセージ画面

図2.5-7にドラフト作成時のメッセージ画面を示す。ドラフトではこのように、複数の時点でのドキュメントを管理でき、ドラフトを通じて複数のユーザによるドキュメント共有が可能となる。

(2) データ転送

OpenDocでは、クリップボード（Clipboard）、ドラッグ＆ドロップ（Drag and Drop）とリンク（Link）によってデータをパートからパートに転送できる。

クリップボードによるデータ転送は、まず転送元のパート内のコンテンツを選択した後、カット（Cut）又はコピー（Copy）コマンドを実行することによりクリップボードバッファにデータのコピーを作り、次にペースト（Paste）コマンドを実行することによりクリップボードバッファーのデータが転送先パートやドキュメントにコピーされる。

転送先のパートは、データの種類を基に、データを転送先のパートの中に組み込むか（Incorporate）、埋め込むか（embed）を決める。

ドラッグ＆ドロップは、クリップボードによる転送に似ているが、クリップボードバッファーを使わずにより直接的なユーザ操作によって実行される。OpenDocでは、ドキュメント内、ドキュメント間やデスクトップへのドラッグ＆ドロップが可能である。

リンクは、パートのフレーム内に他の場所のデータを最新更新状態で見ることを可能にさせる。すなわ

ち他の場所のデータが更新されると、そのコピーも自動的に更新される。元データは、同じフレーム内、異なったフレーム内、異なったパート内、異なったドキュメント内などどこに存在していてもかまわない。

2．5．5　拡張性

OpenDocには、オブジェクトにプログラミングインタフェースを付加する方法やパート間の直接通信を拡張させる機能をもっている。

プログラミングインタフェースは、OSA（Open Scripting Architecture）と呼ばれるスクリプティングシステムによって実現されている。OpenDocのスクリプティングモデルは、コンテンツモデルに基づいているためコンテンツ中心スクリプティングと言われている。コンテンツモデルは、セマンティックイベントを介してアクセスできるコンテンツオブジェクトから構成されている。このコンテンツモデルを用いて、通常のユーザイベント以外のイベントでコンテンツを操作できる。コンテンツモデルを作成することにより、音声などによる様々な新しい操作が可能になる。

パートエディタをスクリプティングシステムで扱えるようにする方法には、パートエディタをスクリプタブルパートにする方法、レコーダブルパートにする方法、カスタマイザブルパートにする方法の３つがある。

パートエディタをスクリプタブルにするためには、まずパートのコンテンツモデルを作り、次にコンテンツオブジェクトとオペレーションの記述を発行しなければならない。その後、スクリプタブルパートはセマンティックインタフェースを介してセマンティックイベントを受け取り実行することになる。

レコーダブルパートは、パートに対する一連の操作をセマンティックイベントに変換するするもので、スクリプトエディタは、それらのセマンティックイベントをスクリプトに変換する。パートエディタをレコーダブルパートにするためには、パートエディタのコードを、インタフェース要素と実行コードに分離させなければならない。

カスタマイザブルパートは、スクリプタブルパートの機能に加えて、イベントにスクリプトを付け加えることができる。

2．5．6　OpenDocクラスライブラリー

OpenDocは、SOM（System Object Model）を用いた共有ライブラリーであり、そのクラス構造は図2.5-8のとおりである。パートエディタは、これらのクラスを用いて作ることになる。次に各クラスの概要を示す。

(1) ODPart

中心となるクラスで、パートエディタは、この抽象クラスのサブクラスとして実現される。ODPartは、６０個のメソッドをカプセル化しており、パートエディタはそれらのメソッドをオーバライドする必要がある。ただし実装は、必要なメソッドのみでよく、他はスタブとして実装すればよい。

(２) ODExtension

OpenDocを拡張するときに使う抽象スーパクラス。

(３) ODSemanticInterface

サブクラス化されたときパートが受け取るセマンティックイベントへのインタフェースとなる。

(４) ODSettingsExtension

サブクラス化されたときセッティング診断ボックスを生成して表示するオブジェクトとなる。

(５) ODDispatchModule

キー入力のようなイベントをパートエディタに分配するために使われる。

(６) ODFocusModule

選択フォーカスのような特別なフォーカスを示す。

(７) ODTransform

描画のために使われるグラフィック変換マトリックスとなる。

(８) ODObject

ほとんどのOpenDocクラスのための抽象スーパクラス。

(９) ODRefCntObject

参照カウントオブジェクトの抽象スーパクラス。これによりOpenDocがメモリーを有効利用することができる。

(１０) ODPersistentObject

パーシスタントオブジェクトに対する抽象スーパクラス。

(１１) ODSession

単一のOpenDocドキュメントをオープンしたりアクセスするためのクラス。

(１２) ODBinding

ドキュメント内のパートとパートエディタを繋げるためのクラス。

(１３) ODStorageSystem、ODContainer、ODDocument、ODDraft、ODStorageUnit

ドキュメントのストレージに関するクラスで、その実装はプラットフォームに依存している。

(１４) ODLink、ODLinkManager、ODLinkSource、ODLinkSpec、ODDragAndDrop、ODClipboard

ロケーション間でのデータ転送に関するクラス。

(15) ODCanvas、ODShape

グラフィックシステムを用いてイメージを表示するクラス。

(16) ODWindowState、ODWindow

プラットフォームに依存したウインドウを表示するクラス。

(17) ODFrame、ODFacet

埋め込みパートのレイアウトを決定するクラス。

(18) ODArbitrator、ODDispatcher

どのパートにユーザイベントを渡すかを制御するクラス。

(19) ODMenuBar、ODUndo

パートエディタからメニューバーへのアクセスを行うクラス。

(20) ODNameResolver、ODMessageInterface

セマンティックイベント操作に関連したクラス。

(21) ODStorageUnitCursor、ODStorageUnitView

特殊な装置内の特殊な値をアクセスするためのクラス。

(22) ODFocusSet

アクティベーションやイベント操作のためのフォーカスを提供するクラス。

(23) ODNameSpaceManager、ODNameSpace

属性と値の組み用のストレージを与えるクラス。

(24) ODObjectNameSpace、ODValueNameSpace

名前スペースを提供するクラス。

(25) ODInfo

EditメニューのInfo項目選択に応じてパート情報ダイアログを表示するクラス。

(26) ODTranslation

プラットフォーム間でのデータ形式の変換を行うクラス。

【参考文献】

1) OpenDoc Programmer's Guide for the MacOS, Apple Computer, Inc. (1995).

(http://www.opendoc.apple.com/dload/documentation.html)

2) Orfali, Robert et al. : The Essential Distributed Objects Survival Guide, John Wiley&Sons, Inc. (1995).

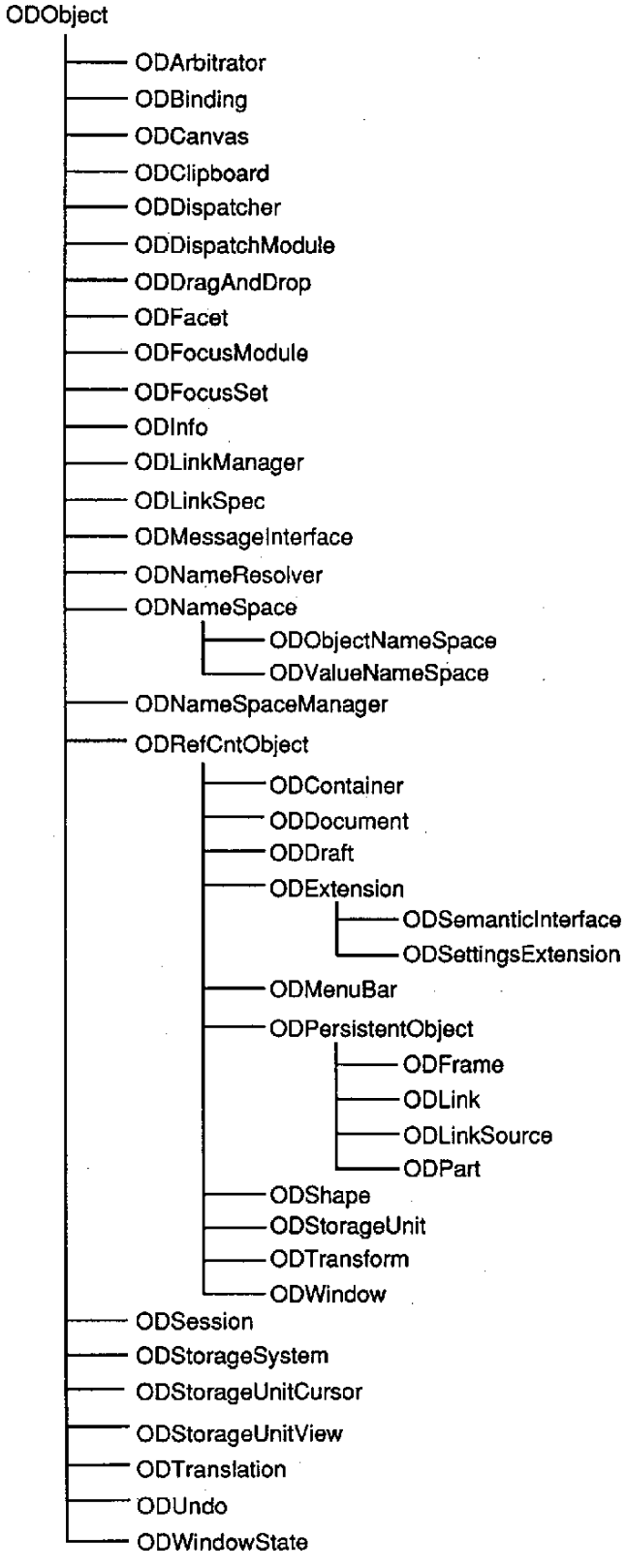


図2.5-8 OpenDocのクラス構造

2．6　VisualAge

2．6．1　オブジェクト指向開発環境

コンピュータ雑誌の広告でも、例えばC++やSmalltalkといった、単に言語のオブジェクト指向対応を強調したものから、より具体的に豊富なクラス・ライブラリー/フレームワークによる開発やブラウザーなどのオブジェクト指向言語の開発環境の整備をセールスポイントとしてあげるものが目につくようになってきた。このような開発環境、クラス・ライブラリーあるいはパーツの充実はオブジェクト指向言語を実際に使ってみようと考えているユーザーにとっては必要不可欠である。というのも、たとえ言語がオブジェクト指向であっても、アプリケーション開発ですぐ使えるようなクラス・ライブラリーとその開発環境の充実なしには、オブジェクト指向技術といえどもユーザーにとってバラ色とはいえなかいからである。

このようなオブジェクト指向開発環境のなかでも、ビジュアル・プログラミング・ツール（以下では、ビジュアル・ビルダーという）の登場は重要である。特に、特殊なオブジェクト指向の知識を持たないユーザーにとって、ビジュアル・ビルダーの登場によって初めてオブジェクト指向のメリットを実感できるようになる。今迄ソフトウェアの再利用といってもコードの切り貼りが主であった。ビジュアル・ビルダーは、このアプリケーション開発をパーツ（ソフトウェア部品）の組み立てに置き換えることによって、大きく変えていくパワーを持っている。GUIプログラミング・ツールが主にウィンドウまわりのコード生成を助けてくれるのに対して、ビジュアル・ビルダーはもう一歩進んでパーツの組み立て工場といったイメージである。

以下では、IBMのVisualAge[1]を中心にビジュアル・ビルダーの創る新しい世界を説明する。まず、概要を説明し、その後その中でも特にコンポーネントウェア構築環境としてビジュアル・プログラミング・ツール（Visual Builder）を取り上げる。

2．6．2　VisualAgeの概要

VisualAgeはIBMのアプリケーション開発環境の中心となる製品であり、言語によってSmalltalk版、C++版、COBOL版がある。ここでは、Smalltalk版をもとにこの製品の概要を説明する。

(1) 部品で構築するビジュアル・プログラミング・ツール

VisualAgeは豊富なパーツ・ライブラリーをもち、その中から部品を選び画面上で接続・設計することによってアプリケーションを作成するビジュアル・ビルダーを持っている。VisualAgeが提供するパーツ・ライブラリーには例えば以下のものがある。

GUI・パーツ

　－ 多数のGUIコントロール・パーツ

データベース・アクセス・パーツ

－ DB２ファミリー・アクセス

－ オラクル・データベース・アクセス

－ ODBCインターフェイスを経由したオラクル、サイベース、その他のデータベース・アクセス

コミュニケーション・トランザクション・パーツ

－ APPC

－ TCP／IP

－ NetBIOS

－ EHLLAPI

－ RPC

－ CICS

－ CPI－C

－ MQI

－ IMS

マルチメディア・パーツ

－ オーサリング機能パーツ

－ マルチメディア各デバイス用パーツ

AS／４００アスセス・パーツ

レポート・パーツ

分散機能パーツ

パーツとして提供されていなくても、製品に含まれている２０００以上あるクラス・ライブラリーのクラスもパーツとして使用できる。また、新しいパーツが必要な場合は、各言語（製品によってSmalltalk、C＋＋、COBOL）で構築することができ、その他の組み込みのパーツ同様パーツ・パレットに登録しパーツとして使用可能である。

(2)チーム開発環境をサポート

VisualAgeにはチーム・プログラミングをサポートするバージョンがある。LAN環境においてクラス／パーツやアプリケーション（アプリケーションとはVisualAge Smalltalkで導入された概念で、クラスの上のカテゴリーである。Smalltalkを用いて大規模アプリケーション開発をサポートするために加えられた。）を共有するのを援助し、以下の機能を提供する。

クラスおよびアプリケーションのバージョン管理

コンフィギュレーション管理

メソッド、クラスおよびアプリケーションのオーナー管理

前提アプリケーションの管理

(3)マルチプラットフォーム対応

VisualAgeで作成したアプリケーションはOS／2、WindowsおよびAIX（UNIX）環境で稼動する。また作成したアプリケーションは無料で各ユーザーへの配布が可能である。つまり、ランタイム・チャージはない。

(4)多言語プログラムとのインターフェイス

CやCOBOLなど、ダイナミック・リンク・ライブラリー（DLL）を通じて既存のコードにアクセスできる。これによって、コードの再利用やスキルの活用を可能にする。また、OMGの提唱するCORBA対応のSOM／DSOMのサーバー・コードも利用できる。C＋＋対応の製品ではダイレクトSOM機能によってC＋＋のクラスをSOMのクラスとしてコンパイルでき、結果としてそのクラスを他の言語から再利用できる。

(5)オブジェクト指向型アプリケーション開発へ容易に移行

VisualAgeはオブジェクト指向型アプリケーション開発への移行を容易にする。多言後とのインターフェイス・サポートやコミュニケーション・トランザクション・パーツを使うことによって既存のコードおよびアプリケーションをいかしながら徐々にVisualAge上のクラス・パーツを開発していくことが可能になる。また、オブジェクト指向技術のスキルがなくてもVisualAgeが提供するパーツを組み合わせることによってアプリケーションを構築し、オブジェクト指向技術の利点を享受できる。また、オブジェクト指向技術を習得した開発者には、新しいパーツを作成する機能により拡張性を提供する。

次にビジュアル・ビルダーの開発環境のメリットを従来の方法と比較しながら説明する。

2．6．3 ビジュアル・ビルダー開発環境

(1)コードを書かなくてもアプリケーションが構築できる

今迄と比べて、作られるプログラム自体が理解しやすくなっている。オブジェクト指向一つのメリットは、C++とかSmalltalkといった記述力の高いオブジェクト指向言語を使うことによって、クラス・ライブラリーを利用して短いコードでアプリケーションを開発することが可能になったことである。ビジュアル・ビルダーでは基本的にコードを書くことすらいらない。パーツとパーツ間のビジュアルな連携の図がプログラムとなる。視覚を使う方法が強力なことは疑うまでもない。実際に同じことをするプログラムをC言語、IBM CSet++[2]のユーザー・インターフェイス・クラスを用いたC++言語とVisualAgeで作った例を図2.6-1～図2.6-3に示す。表現の差が分る。

```
//*****************************************************************************
// Cを使ったサンプル・プログラム
//*****************************************************************************
#define INCL_WIN
#define INCL_DOS
#define INCL_GPIBITMAPS
#include <os2.h>                    //システム・ヘッダー
#include "CloseMe.h"                //アプリケーション・ヘッダー
MRESULT EXPENTRY CloseMeWinProc( HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2);
HAB hab;
ULONG flCreate;
HWND hwnd;                          //メイン・ウインドウのハンドル
HWND hwndc;                         //クライアントのハンドル

//*****************************************************************************
// メイン・プログラム
//*****************************************************************************
main()
{
  QMSG qmsg;                        //メッセージ・キュー
  HMQ hmq;                          //キューのハンドル
  HPS hps;                          //プレゼンテーション空間のハンドル

  hab = WinInitialize((USHORT) NULL); //PMの初期化
  hmq = WinCreateMsgQueue(hab,0);    //メッセージ・キューの作成

  //ウインドウの登録

  WinRegisterClass(hab, "CloseMe", (PFNWP)CloseMeWinProc, CS_SIZEREDRAW, 0);
  flCreate = FCF_SIZEBORDER | FCF_MINMAX | FCF_SYSMENU | FCF_TITLEBAR;

  hwnd = WinCreateStdWindow(HWND_DESKTOP, //メイン・ウインドウの作成
    WS_VISIBLE, &flCreate, "CloseMe", NULL, 0L, NULL, 0, (PHWND)& hwndc);

  WinSetWindowPos(hwnd, HWND_BOTTOM, 400, 100, //ウインドウの位置の設定
    300, 300, SWP_MOVE | SWP_SIZE | SWP_ACTIVATE);

  while ( WinGetMsg(hab, (PQMSG)&qmsg, (HWND)NULL, 0, 0)) //メッセージ・ループ
    WinDispatchMsg(hab, (PQMSG)&qmsg );

  WinDestroyWindow(hwnd);           //終了処理
  WinDestroyMsgQueue( hmq );
  WinTerminate( hab );
  return(0);
}
```

```
//*************************************************************************
//  ウインドウ・プロシージャー
//*************************************************************************
MRESULT EXPENTRY CloseMeWinProc( HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2)
{
  HPS hps;                    //プレゼンテーション空間のハンドル
  RECTL rect;
  switch(msg)
  {
    case WM_COMMAND:
      switch (SHORT1FROMMP(mp1))
      {
        case PB_CLOSE:          //プッシュ・ボタンが押された時
          if(WinSendDlgItemMsg (hwnd,
            PB_CLOSE, BM_QUERYCHECK, MPFROMSHORT(0), 0L))
            //メイン・ウインドウを閉じる
            WinPostMsg( hwnd, WM_QUIT, (MPARAM)0,(MPARAM)0 );
          break;
      }
      break;
    case WM_CREATE:               //プッシュ・ボタンの作成
      WinCreateWindow(hwnd, WC_BUTTON, NLS_3, BS_PUSHBUTTON | WS_VISIBLE,
        70, 100, 130, 70, hwnd, HWND_TOP, PB_CLOSE, 0L, NULL);
      break;
    case WM_PAINT:
      hps = WinBeginPaint( hwnd, (HPS) NULL, (PRECTL)&rect);
      WinFillRect( hps, (PRECTL) &rectl, SYSCLR_WINDOW);
      WinEndPaint( hps );
      break;
    default:
     return(WinDefWindowProc(hwnd,msg,mp1,mp2));
  }
  return(FALSE);
}
```

図2.6-1　Cを用いたサンプル・プログラム

```
//*************************************************************************
// ユーザー・インターフェイスclass libを使ったC++サンプル・プログラム
//*************************************************************************
                        //IBM UI class ヘッダー
#include <iapp.hpp>             //IApplication クラス
#include <ipushbut.hpp>          //IPushButton クラス
#include <isetcv.hpp>           //ISetCanvas クラス
#include "closeme.hpp"           //AHelloWindow Class ヘッダー
#include "closeme.h"            //シンボル定義
```

```
//*************************************************************************
// メイン・プログラム
//*************************************************************************
void main()                          //メイン・プログラム
{
 ACloseMeWindow mainWindow (WND_MAIN); //メイン・ウインドウの作成
 IApplication::current().run();       //アプリケーションの起動
}

//*************************************************************************
// ACloseMeWindow :: ACloseMeWindow - コンストラクター
//*************************************************************************
ACloseMeWindow :: ACloseMeWindow(unsigned long windowId)
 : IFrameWindow ("A Close Me Window", //IFrameWindow コンストラクターの
   windowId)                          //呼び出し
{
 aCanvas = new ISetCanvas(WND_BUTTON, this, this); //キャンバスの作成
 aButton = new IPushButton(PB_CLOSE,  //ボタンの作成
 aCanvas, aCanvas, IRectangle(), IPushButton::defaultStyle() |
   IControl::tabStop);
 aButton -> setText("Close me!");     //テキストの設定
 setClient(aCanvas);                  //キャンバスをクライアントに設定
 aCanvas -> setMargin(ISize(100,200)); //マージンの設定
 handleEventsFor(this);               //イベント・ハンドラーの設定
 sizeTo(ISize(400,400));              //メイン・ウインドウのサイズ
 setFocus();                          //フォーカス
 show();
}

//*************************************************************************
// ACloseMeWindow :: command - コマンド処理メソッド
//*************************************************************************
Boolean ACloseMeWindow :: command(ICommandEvent & cmdEvent)
{

 switch (cmdEvent.commandId()) {      //コンマンドIDの収得
  case PB_CLOSE:                      //プッシュ・ボタンが押された時
    close();                          //メイン・ウインドウを閉じる
    return(true);                     //処理して戻る
    break;
 }
 return(false);                       //処理せずに戻る
}
```

図2.6-2　C++を用いたサンプル・プログラム

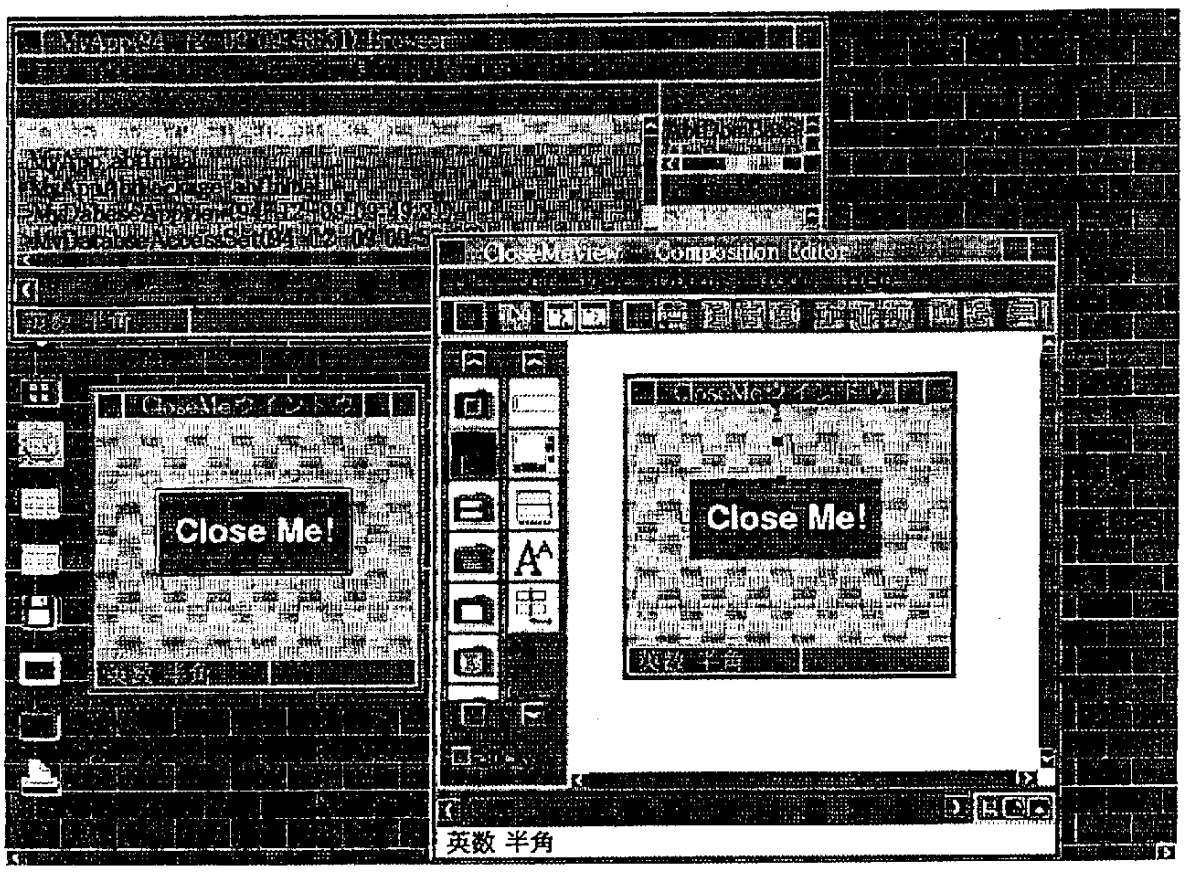


図2.6-3　VisualAgeを用いたサンプル

VisualAgeのコンポーネント・エディター（図2.6-3）を見ながらどのようにアプリケーションを作るのかを説明する。まず、プッシュ・ボタンとか入力フィールドとかいったパーツを集めたパレットがある。ユーザーはそのパーツを使って自分のアプリケーションを作っていく。通常ドラッグ&ドロップのような感覚的にわかりやすい手法で、使用するパーツを組み立てる空間に移動する。パーツ間の連携には製品によってさまざまなやり方があるが、VisualAgeではビジュアルに表現する。パズルを組み合わせるような感じでパーツを組み合わせアプリケーションを作っていく。

VisualAgeの最大の特徴である線を用いた接続の種類について以下に概要を述べる。

(a) 接続タイプの概要（VisualAge C＋＋ V3 より）

VisualAge C＋＋には以下の接続が用意されている。

属性－属性接続	お互いに属性の変化を知らせ変更する
イベント－属性接続	イベントが発生するたびに属性を変更する

イベントーアクション接続	イベントが発生するたびにアクションを呼び出す
イベントーメンバー関数接続	イベントが発生するたびにメンバー関数を呼び出す
属性ーアクション接続	属性が変化するたびにアクションを呼び出す
属性ーメンバー関数接続	属性が変化するたびにメンバー関数を呼び出す
属性ーカスタム・ロジック接続	属性が変化するたびにカスタム・ロジックを呼び出す

なおカスタム・ロジックとは、メンバー関数よりも簡易に作成でき（コンポジション・エディター上で定義できる）パーツの機能・アクションを拡張する方法である。これらの視覚的なアプリケーションの構築によってより幅広いユーザーのアプリケーション開発への参画を可能にする。

(2)使えるのはGUIパーツだけではない

ビジュアル・ビルダー上で使えるパーツはGUI（グラフィカル・ユーザ・インタフェース）だけでなく、ビジュアルではないものも、パーツ（Non-Visual Parts）としてGUIパーツと同様に組み立てていくことが可能である。Smalltalkの世界でおなじみのMVC（Model-View-Controller）のモデル部分まで守備範囲に含まれる。つまりすべてのソフトウェアの要素がパーツとして構築し再利用できる。例えば現在VisualAge

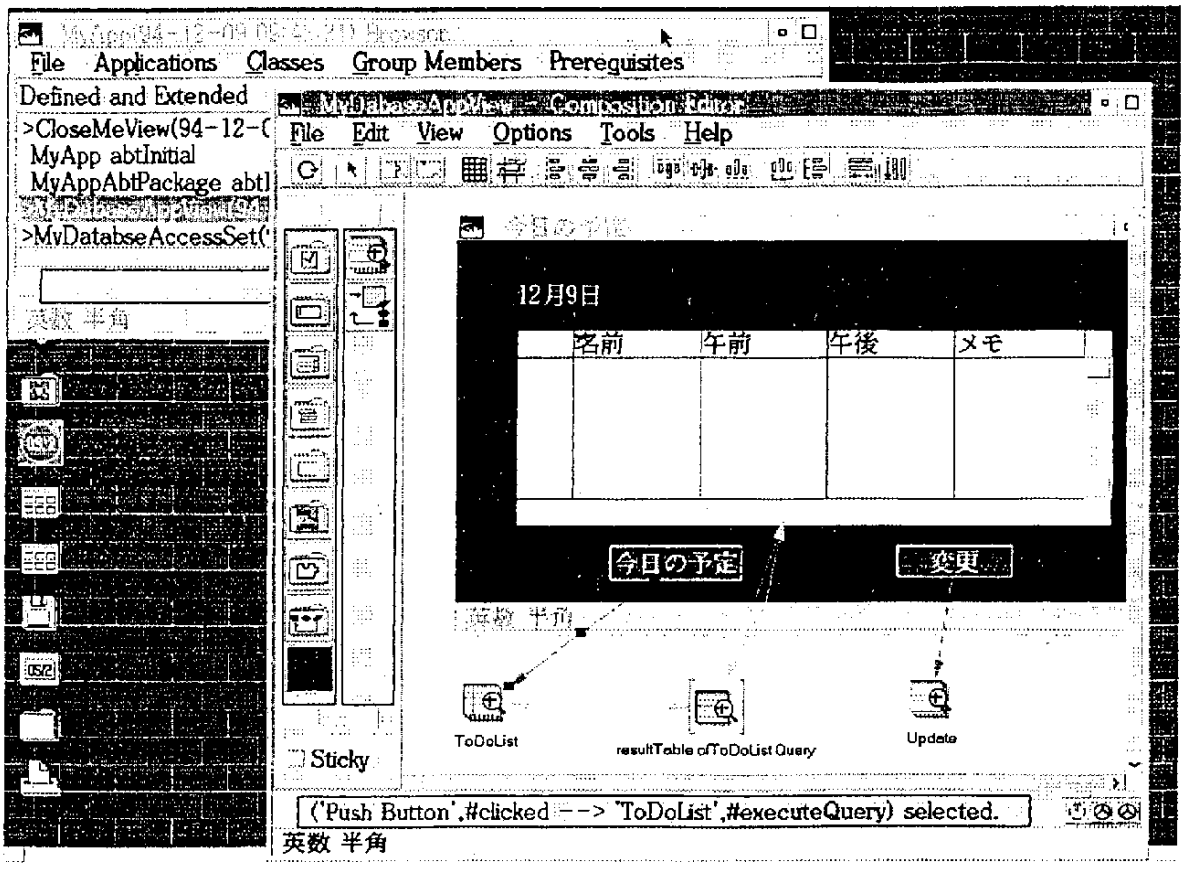


図2.6-4 VisualAgeでデータベース・アクセス・パーツ（nun—visualparts）を使う例

ではデータベース・アクセス、コミュニケーションやトランザクション、マルチメディアのためのパーツが用意されている。また既存のC言語、COBOL言語で作ったモジュール、SOMのオブジェクトもラッパーでかぶせてパーツとして取り込める。図 2.6-4 はNon-VisualPartsであるデータベース・アクセス・パーツを使っている例である。

(3) シームレスなオブジェクト指向開発

GUIはもちろんのこと、それ以外もパーツとして組み立てられることを示した。しかし、あらかじめ提供されているパーツを組み合わせることしかできないとすると、魅力は半減する。ここで開発言語からコンセプトまでオブジェクト指向で作られている製品の強みがでてくる。VisualAgeはユーザーが使う言語もパーツ構築に使われている言語も同じオブジェクト指向言語を使っている。言語自身が持つ継承を使いパーツをカスタマイズしたものや、新しいパーツも同じ言語/開発環境を使って作成可能である。このようなプリミティブなパーツや、あらかじめ提供されているパーツやプリミティブなパーツを組み合わせたコンポジット・パーツを作成することが可能である。また、VisualAgeが提供しているノーティフィケーションのフレームワークを利用するためには、新しくパーツを作る時に特定のクラスを継承する。それを利用することによって、パーツの連携の際の細かい作業を気にすることなく新しいパーツを加えていくことができる。

2．6．4 これからの発展

ビジュアル・ビルダーがパーツの組立工場であることを示した。このような製品の登場によってこれからのソフトウェア開発者は、主に既存のパーツを組み立てることによってアプリケーションを作るアプリケーション・ビルダーと、ビジュアル・ビルダー上にパーツを提供していくパーツ開発者とにわかれていくと思われる。またパーツはそれ自身流通していく。図 2.6-5 はメソッド、属性などユーザーに公開しているパーツの情報をみたり定義するためのエディターである。今後のアプリケーション開発は、このような情報からなるパーツ・カタログから必要なパーツを選択し組み合わせるスタイルに変わって行くであろう。この様なパワフルなツールはオブジェクト指向プログラミングだけに限られるのではなく、オブジェクト指向分析/設計において、例えばユーザーの要求を明らかにするために作られるプロットタイピングやオブジェクト・モデルの検証に使われ、方法論とうまく組合わせて使われる可能性を持っている。

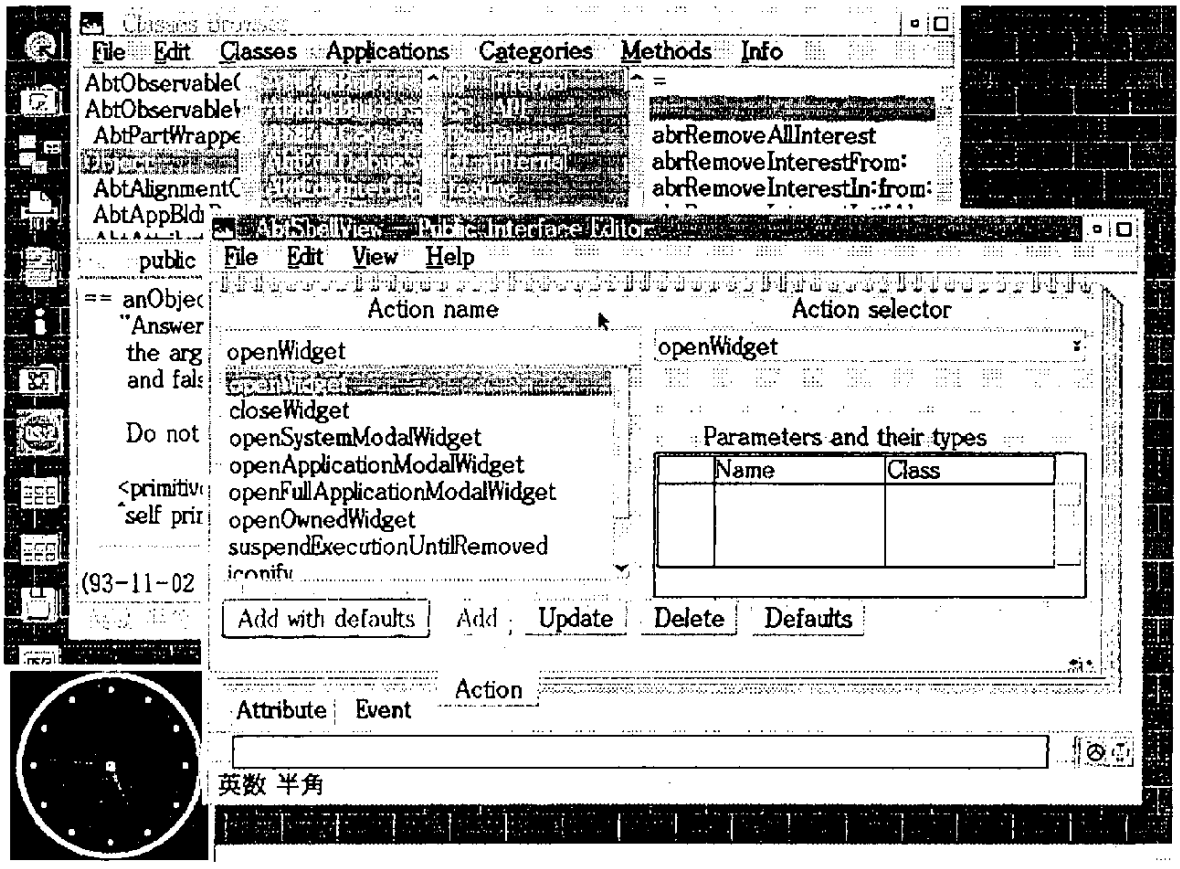


図2.6-5 メソッド、属性などの定義・ブラウズためのパブリック・インターフェイス・エディター

【参考文献】

1) VisualAge: IBM製品. Smalltalk上のビジュアル開発環境.

2) CSet ++: IBM製品. C/C++ツール、OS/2Toolkitおよび統合環境のWorkFrame/2をセットにした開発環境.

2．7　Java プログラミング言語

2．7．1　概要

Java は現在 JDK（Java Developers Kit）1.0 が正式リリースされている。Alpha バージョンのときに含まれていた World Wide Web のブラウザ HotJava は現在別スケジュールで進行している。次回リリース予定の HotJava は Aplha バージョンに比べてかなり多くの機能が含まれているが未発表のため紹介することはできない。

ここでは、Java 言語の以下のような項目について説明する。

1．Java とは

2．Java の文法

3．Java のプログラミング

2．7．2　Java とは

Java 言語は最近 World Wide Web の表示言語としてだけではなく、アプリケーションを記述する言語としても注目されている。特に企業内の情報インフラを整備する言語として注目度が高い。それは、Java 言語が優れた特長を数多くもっているからだと考えられる。Java には以下のような特長がある。

- シンプルでかつ親しみやすい
- オブジェクト指向
- アーキテクチャに依存しない
- 安定性
- ダイナミック特性
- 言語レベルのセキュリティ
- 言語レベルのマルチスレッド

これらの項目について説明する。

(1) シンプルでかつ親しみやすい

Java の文法は非常に C/C++ に似いる。よって、C/C++ のプログラマであれば違和感なく修得することが可能である。また、Java は、C/C++ 上で曖昧または冗長と呼ばれている部分を削除し不足していると言われている部分を追加している。C/C++ から削除された部分は以下のようである。

- プリプロセッサ（typedef, define など）
- 構造体
- 関数

- 多重継承

- Goto 文

- 演算子のオーバーロード

- 自動強制

- ポインタ演算

(2) オブジェクト指向Java はオブジェクト指向の言語である

オブジェクト指向言語の要素と言われる以下のものを満たしている。

- カプセル化（Encapsulation）

- 多態（Polymorphism）

- 継承（Inheritance）

Java はこれ以外にも動的な接続（Dynamic Binding）というものを備えている。通常 C/C++ であれば、ある関数などを変更した場合、関係するすべての関数、ライブラリおよび Class オブジェクトを再コンパイルしなくてはならない。Java では変更点を含む Class のみを再コンパイルするのみで他の部分は再コンパイルする必要がない。これによってプログラムの部品単位での提供がより柔軟にできるようになる。

(3) アーキテクチャに依存しない

Java はコンパイルされたバイナリは変更することなく Java をサポートするすべての環境で動作させることができる。これは、Java が仮想マシーンコードを採用しているためである。Java はコンパイル後仮想マシーンコードと呼ばれるバイトコードに変更される。実行時に各プラットフォーム上のインタプリタがこれを解釈しプラットフォーム上で実行する。このためアプリケーションは今までのように各プラットフォーム用に作成する必要がなくなる。Java のアプリケーションを 1 つ作成すればどのプラットフォームでも起動できるからである。

(4) 安定性

C/C++ のプログラムのバグの ５０％がメモリ管理によるものと言われている。Java ではメモリの参照を行うポインタの演算機能を削除するとともに、自動ガーベジコレクション機能によりプログラマが作成したオブジェクトの削除を意識する必要がなくなっている。つまり C/C++ のような free() を呼び出さなくても Java のインタプリタがオブジェクトを追跡し使用されなくなった時点で自動的に削除テーブルに移動させる。オブジェクト自体の削除も低位のスレッドで自動的にバックグラウンドで行われる。

(5) ダイナミック特性

Java のクラスオブジェクトは実行時にすべてそろっている必要がない。プログラムの中でオブジェクトが生成されるときにリンクする。また、Java はオブジェクトをネットワーク上の任意のマシーンからロー

ドすることも可能になっている。これによって例えばワープロなどのアプリーケーションで最初は基本的な文字の変換/表示の機能だけを実行し、他の機能例えば文字検索などは使用される時にネットワーク上で必要となるオブジェクトを検索/ロード/リンク/実行するということが可能になる。

(6) 言語レベルのセキュリティ

言語レベルのセキュリティはコンパイル時と実行時に区別される。まず、コンパイル時には基本的な文法の検査が行われる。オブジェクト参照はすべてシンボリックリンクになっていて実際のメモリレイアウトは行われない。実行時においてもコンパイル時と同じ文法の検査がインタプリタ上で行われる。これは、仮想バイトコードの仕様をネットワーク上の公開ページから入手し作成された任意のコンパイラのためである（これら任意のコンパイラはJavaの環境として提供されるコンパイラと同様な検査を行っているかどうか判断できない）。Javaのオブジェクトはネットワーク上からもロード可能である。しかし必ずローカルな環境から検索されるためクラスオブジェクトの置き換えを行うことは不可能となっている。また、オブジェクト間の不等アクセスを防ぐためネットーワーク上からロードされたオブジェクトはローカルな環境からロードされたクラスオブジェクトと別領域で管理されオブジェクト間の通信もまた同様に管理されている。これ以外にもネットワークのオブジェクトの 1 つとして以下のようなオブジェクトのロードの制限ができるものを用意している。

- すべてのネットワークのアクセスを禁止する
- HTML ファイルなど接続しているホストのみアクセスを可能にする
- Firewall を介してのみアクセスを可能にする
- すべてのアクセスを可能にする

(7) 言語レベルのスレッド

Java はマルチスレッドのクラスオブジェクトを用意している。これはプリエンプティブに動作し、プログラマは簡単にこれを使用するこができる。また、コンテキストの同期やプライオリティの設定する機能も用意されている。

2．7．3　Java の文法

Java 言語の文法は C/C++ に非常によく似ている。よって、C/C++ のプログラマなら違和感なく取り組むことができる。ただし、C 言語のみを知るエンジニアはオブジェクト指向の概念について事前に学習しておいた方がよい。これは、従来のプログラミングの構造とまったく違う構造を要求されるからである。Java の文法について以下の順序で説明する。

- 基本データ型

- 演算子
- 制御構造
- 例外処理
- 配列
- package と import
- class と interface
- 文字列

(1) 基本データ型

基本データ型の各ビットサイズは固定されている。よって、他の言語のようなマシン依存というとこはない。基本データ型には以下のようなものがある。

- char 16 ビット（unicode）
- boolean 論理値（true/false）
- byte 8 ビット符号付き整数
- byte 16 ビット符号付き整数
- byte 32 ビット符号付き整数
- byte 64 ビット符号付き整数
- byte 32 単精度浮動小数点（IEEE754）
- byte 64 単精度浮動小数点（IEEE754）

また指定方法は以下のように C/C++ と同じである。

```
byte b = 0xf;
short s = 077;
int i = 100;
long l = 1000L;
char c = 'a';
float f1 = 3.14f; float f2 = 6.02e+23f;
double f1 = 3.14d; double f2 = 6.02e-9;
boolean bn = true;
```

ただし、boolean 型は他のデータ型に変更することはできない。

(2) 演算子

演算子は基本的に C/C++ と同様である。ただし、符号なしのデータタイプがないため、C/C++ の符号

なし右シフトをできるよう >>> と >>>= が新しく用意されている。また、',' 演算子は for 文の内部でのみ使用可能となっている。演算子は以下のようである。

= > < ! ~ ? : == <= >= != && !! ++ -- + - * / & ¦ ^ % << >>

+= -= *= /= &= ¦= ^= %= <<= >>= >>> >>>=

(3) 制御文

制御文において C++ および C との違いは以下のようである。

- 条件式は論理値
- goto 文はない
- ラベル付き break と continue がある
- for 文の初期値の設定では型指定が可能である

（ただし、その適用範囲はループ内のみである。）

ラベル付き break と continue は基本的に goto 文と同様である。

例)

```
for(int x = 0; x < 10; x++) {
   if( ... ) {
      continue loop;
   }
}
loop: int a = 1;
```

制御文には以下のようなものがある。

if(条件式) { ... } else if (条件式) { ... } else { ... }

switch(整数(int)){case X: case Y: case Z: default: }

while(条件式){ ... } と do{ ... }while(条件式)

for(初期設定; 条件式; 増減値) { ... }

break, continue, return

条件式?真:偽

(4) 配列

配列はオブジェクトの 1 つとして定義されている。後ほど説明する Class と同種類に属する。多次元配列を指定する場合は C/C++ と同じように配列の配列として定義する。また配列のサイズはlength を使用して取得する。

例)

一次元配列

```
int one[] = new int[10];
int [] one = new int[10];
```

二次元配列

```
int two[] = new int[10][10];
int [] two = new int[10][10];
```

メソッドの戻り値

```
int [] methodOne() { return new int[3]; }
int methodOne() [] { return new int[3]; }
```

初期化

```
int one[] = {1, 2, 3};
int two[][] = {{1, 2, 3}, {4, 5, 6}};
```

配列のサイズの取得。

```
int one[] = new int[10];
int array_size = one.lenght;
```

(5) 例外処理

Java は、C++ の例外処理をモデルとしてエラーのハンドリングを提供している。指定はエラーを生成するメソッドと取得するメソッドの 2 つに分られる。生成するメソッドはメソッド名の次にどのエラーを生成するかを throws を使用してメソッドの指定の次に指定する。また、生成する場所では throw を使用してエラーを生成する。ただし、生成するエラーは必ず Exception クラスのサブクラスでなくてはならない。クラス/サブクラスについては後ほど説明する。

例)

```
public methodOne getFD(int a) throws SomeException {
    if(a == 0)
        throw new SomeException();
}
```

エラーの取得をするメソッドは

```
try {
} catch (例外の型 e1) {
```

```
} catch (例外の型 e2) {
} finally {
}
```

と指定する。finally には例外が発生する/しないにかかわらず必ず処理する必要のある内容を記述する。finally は必ず指定する必要はない。

例)

```
try {
    f = new File("sample");
    file.write("abc);
} catch (IOException e) {
    return;
} finally {
    file.close();
}
```

(6) package

これは、クラスをグループ化し階層的に管理するのに用いる。例えば package として sample.abc と指定するとこの指定を含むすべてのクラスを同一グループとしてJava は取り扱う。これは、後ほど説明するクラスのアクセス制限に影響する。また、これは Class の検索パスとして実行時にも使用される。例えば Sample.class の中に package 名として sample.abc と指定した場合 sample/abc/Sample.class（UNIX の場合）として検索される。

例)

ファイルの最初に次のように指定する。

```
package some;
package some.abc;
```

(7) import

クラスを記述する場合、その中で使用される他のクラスのクラス名を package 名と共に指定しなければならない。ただし java.lang の package に属するものは自動的に指定されるのでその必要はない。また、package内に含まれるすべての class を指定する場合にはワイルドカード '*' を使用することも可能である。

例)

```
import java.util.Vector;
```

```
import java.awt.*;
```

import を使用して必要となるクラスを指定しない場合、必要となる場所で package 名を含むフルパスを使用しなければならない。

例)

```
Font f = new java.awt.Font();
```

(8) クラス

java のオブジェクトはクラス単位で構成される。複数のクラスを含むファイルをコンパイルした場合そのバイナリはクラス単位で生成される。例えば

```
public class Sample {
}
class Child {
}
```

という内容のファイルをコンパイルするとバイナリは Sample.class および Child.class の 2 つのファイルが生成される。java におけるクラスの特長は

- 単一継承
- インタフェースの指定

があげられる。それではこれらの 2 つの内容を含めてクラスの定義の方法について述べる。クラスの定義は次のような要素に分けることができる。

```
修飾子 class クラス名 [extends スーパークラス名] [implements インタフェース名(複数可)] {
    メソッドおよびデータ領域
}
```

例)

```
class AAA {
    int a;
    void abc() { ... };
}
public class BBB extends AAA {
    char b;
    void ccc() { ... };
    void ddd() { ... };
```

}

この例の場合 AAA と BBB の二つのクラスが定義され AAA は BBB のスーパークラスとして定義されている。これらのクラスは

BBB sample = new BBB();

として生成することができ、その付属するメソッドは

sample.ccc();

sample.abc();

と使用することができる。

(a) クラスの修飾子

Java のクラスの修飾子には他のクラスからのアクセスの制限を指定するものとクラス自体の性質を指定する 2 種類に分類される。アクセス制限と性質の修飾子は同時に指定することもできる。アクセスの制限を指定する修飾子には次の 2 種類がある。

- public
- 指定しない

public の場合は、すべてのクラスからアクセスが可能になる。また何も指定しない場合には、同一の package のクラスのみからのアクセスが可能となる。性質を示すものには次の 2 種類がある。

- abstract
- final

abstract とは後ほど説明する abstract メソッドを含むクラスのであることを示す。逆に abstractのメソッドを含むクラスはすべてabstract と指定しなければならない。final とはサブクラスを作成できないクラスのことである。

(b) スーパークラス

スーパークラスとはそのクラスの親つまり継承するクラスのことである。前例の AAA クラスは BBBクラスのスーパークラスである。継承する場合 extends を使用してスーパークラス名を指定する。スーパークラスは必ず 1 つしか指定できない。

(c) インタフェース

インタフェースとはクラス間のメソッドの統合に使用される。インタフェースは implements を使用して複数指定できる。インタフェイスの指定方法については後ほど説明する。

(d) メソッド

メソッドはクラスの内部を構成する要素である。メソッドとは、C/C++ の関数のように記述するが

C/C++ のように外部定義ができない。必ずそのクラス内部でその内容を定義する。

修飾子 戻り値 メソッド名 (引数1, 引数2, ...) {

メソッドの内部定義

}

例)

```
public class Sample {
   public int add(int x, int y) {
      int z;
      z = x + y;
      return z;
   }
}
```

(e) メソッドの修飾子

Java のメソッドの修飾子にはも他のメソッドからのアクセスの制限を指定するものとクラス自体の性質を指定する 2 種類に分類される。アクセス制限と性質の修飾子は同時に設定することもできる。アクセスの制限を指定する修飾子には次の 4 種類がある。

- private
- 指定しない
- protected
- public

private は同一クラスにあるメソッドからしかアクセスできない。修飾子を指定しない場合には同一 package 内のクラスにあるメソッドからしかアクセスすることができない。protected は同一 package およびそのサブクラスにあるメソッドからアクセスが可能である。public の場合は制限なしとなる。性質を示すものには次の 5 種類がある。

- final
- abstract
- static
- synchronized
- native

final とはオーバーライドできないメソッドである。abstract とはその内容を持たないメソッドで戻り値

およびその引数を指定してその内容は継承されるサブクラスで指定するメソッドである。サブクラスでは必ずその内容を定義しなければならない。static とは、クラスのメソッドとも呼ばれそのクラスがインスタンス化されていなくても使用できるメソッドのことである。synchronized とは、Thread を使用する場合、この指定のメソッドが使用されると、そのメソッドが処理を行っている間はそのクラスのインスタンス全体がロックされるスレッドの同期のための指定である。最後に native とはこのメソッドの内部は C 言語で定義するという指定である。

(f) コンストラクタ

コンストラクタとはメソッドの特別な使用方法で、クラスの初期化を行う際に使用するメソッドのことである。通常コンストラクタが指定されていない場合、クラスが生成されるとスーパークラスの引数なしのコンストラクタが起動される。ルートのクラスすなわちすべてのクラスは Object クラスのサブクラスとなっている。よって、すべてのスーパークラスでコンストラクタが指定されていない場合、Object クラスの Object() が起動される。また、コンストラクタは必ずそのコンストラクタを含むクラスのクラス名をメソッド名として使用しなければならない。

例)

```
public class Sample {
  praivete int a;
  public Sample(int i) {
    a = i;
  }
}
Sample sample = new Sample(2);
```

(g) this と super

this と super はそれぞれクラス内部で自分自身とスーパークラスとして使用できる。

```
public class Parent() {
  int a;
}
public class Child() extends Parent {
  int a;
  void aaa () {
```

```
        this.a = 1;
        super.a = 1;
    }
}
```

(h) データ

データもクラスの内部を構成する要素である。使用する基本データタイプおよびクラスを 定義する。

例)

```
private char name[];
private int priority;
protected Thread td;
public static int i = 5;
public static char c = string.charAt(0);
```

(i) データの修飾子

データの修飾子は基本的にはメソッドの修飾子と同様であるが、synchronized と native の指定はない。

(9) インタフェース

インタフェースとは abstract のメソッドの集まりでクラス間のメソッドのインタフェースの継承に使用される。指定方法は以下のようである。

```
修飾子 interface インタフェース名 [extends インタフェース名 (複数可)] {
内部定義
}
```

例)

```
public interface AAA {
    void abc();
}
public interface BBB {
    int efd();
}
public class Sample implements AAA, BBB {
    void abc() { ... };
    int efg() { ... };
```

```
}
public class Child implements AAA {
   void abc() { ... };
}
```

(１０) 文字列

Java において文字列は java.lang package に含まれる String クラスに代入される。

例)

```
String s = "abc";
```

また、文字列はクラス演算子を使用して簡単に接続ができる。

例

```
String a = "aaa";
String b = "bbb";
String abc = a + b + "ccc";
```

ここの abc には "aaabbbccc" が代入される。このとき、C/C++ のように文字列の最後に '¥0' が代入されることはない。char データタイプに文字列を代入した場合には

```
char c[] = {'a', 'b', 'c'};
```

と行わなくてはならない。

２．７．４　Java のプログラミング

ここでは、Java のプログラムのコンパイルから起動までの一連の流れを説明する。

(１) Java のパッケージ（package）

Java でパッケージと呼んでいるものは C/C++ でいうライブラリと同じである。Java には以下のようなパッケージが含まれている。

- java.lang

言語のベースを構成する部分

- java.io

入出力（ファイル I/O など）

- java.net

ネットワーク（Socket, Telnet や URL など）

- java.util

ユーティリティ（Date など）

- java.awt

グラフィカルユーザインタフェース

- java.applet

アプレット

- sun.tool.debug

デバッキング

(2) Java のツール

Java のツールには以下のようなものが含まれている。

- javac

Java のコンパイラ

- java

Java のインタプリタ

- jdb

Java のコマンドラインデバッガ

- javah

C 言語用ファイル生成ツール

- javap

逆アセンブラ

- javadoc

ドキュメント自動生成ツール

- appletviewer

アプレットのみを表示するツール

(3) プログラムの種類

Java には、パッケージ以外に次の 2 種類のプログラムが存在する。

- アプリケーション

- アプレット

アプリケーションとは単独で動作するプログラムのことを言う。HotJava World Wide Web のブラウザや javac コンパイラは Java 言語で記述されたアプリケーションである。アプレットとは、HTML の拡張タグ <applet> を使用して HTML ファイルの中で起動するアプリケーションのことを言う。

(4) Java のコンパイル

Java のプログラムは javac コンパイラを使用してコンパイルする。

javac Sample.java

コンパイルが終了すると *.class というファイルが生成される。これが Java のバイナリである。バイナリは前述したようにクラス単位で生成される。よって、ソースファイルに 2 つのクラス定義が含まれていれば 2 つの *.class ファイルが生成される。

(5) Java の実行

Java のプログラムの実行方法はプログラムの種類によって異なる。アプリケーションの場合、

java Sample

と java インタプリタにクラス名（class は必要ない）を指定すれば起動される。アプレットの場合には、そのプログラムを起動する HTML ファイルをまず作成する。

例)

ファイル sample.html

```
<HTML>
<BODY>
<APPLET> <APPLET CODE = "Sample.class"> </APPLET>
</BODY>
</HTML>
```

作成した HTML ファイルを appletview またはアプレットを起動できる World Wide Web のブラウザで読み込む。appletviewer の場合は以下のようである。

appletviewer sample.html

appletviewr で読み込んだ場合そのファイルに存在するアプレットすべてが動作するように表示される。アプレットタグ以外の記述はすべて無視される。

(6) Java のデバッグ

Java にはデバッグ用のパッケージ（sun.tool.debug）がある。これを使用してデバッグ用のコマンド jdb と appletview の -debug オプションが用意されている。使用方法は以下のようである。

jdb Sample

appletviewer -debug sample.html

jdb は、UNIX の dbx に非常によくにた方法でデバッグを行うことができる。例えば上記コマンドを実行すると '>' というプロンプトが表示される。ここで、dbx と同じようにコマンドを入力してデバッギン

グを行う。例えば run と入力すればプログラムが実行される。

【参考文献】

(1) Java の情報

Java の情報は一般に広く公開されている。以下のようなアドレスから入手することができる。

- http:java.sun.com

- news:comp.lang.java

以前はメールも存在したがこれはすべて news に置き換えられるようである。

(2) Java のドキュメント

Java のドキュメントは http://java.sun.com から簡単に入手することができる。ここにはスペックのドキュメントからプログラミングのチュートリアルまで含まれている。特にプログラミングのチュートリアルは簡単なコマンドラインへの出力からファイルI/O、スレッド、アプレット、グラフィカルユーザインタフェースまでわかりやすく記述されている（チュートリアルは 3/4/96 に更新されその内容はかなり前回のものよりも充実している）。

The Java Language Environment White Paper (http://java.sun.com)

The Java Java Language Tutorial (http://java.sun.com)

Documentation for the 1.0 Release (http://java.sun.com)

2．8　WebBASEとその応用事例

インターネットを使って世界中の情報を入手できるWWW（World Wide Web）が急速に普及している。また、企業内の情報システムにおいても、その操作性からWWWを利用して情報システムを構築したいという要望が高まっている。

NTTでは企業内データベースのデータをWWWの情報ページで容易に利用できるWWWとリレーショナルデータベースの連携機構としてWebBASEを開発した。WebBASEの適用分野としては、複数のページに跨って一貫した処理を行うような業務、例えば、会員制のオンラシンショッピングや、複雑な企業内システムの業務などがある。また、不特定多数の人からの情報検索やアンケートなどの収集向きの業務、例えば、電話番号案内、飛行機や電車などの時刻表検索、図書／文献の検索、FAQなどにも適用できる。主な適用例としてホテルやゴルフ場の検索／予約システム、また、科学技術庁金属材料技術研究所との共同研究により開発した、金属スペクトルデータ検索システムがある。

2．8．1　WWWとデータベース連携の背景

インターネット環境の発展により全世界の情報サーバに接続し情報を入手したり、逆に情報を発信できるWWWが急速に普及してきている。また、オンラインショッピングなどのエレクトロニック・コマース（EC）の実現においてデータベース連携機能が急速に必要となってきた。さらに、企業においても、端末の機種やOSに依存しないこと、操作性が良いこと、安価であることなどからWWWを企業内の情報システムの開発に利用したいという要望が高まっている。このため、企業内のシステムにおいても、WWWを利用して業務サービスを開発したいという情報システム部門が増えている。

しかし、WWWで情報を発信する場合には、HTML（Hyper Text Markup Language)を用いて情報ページを作成する必要があり、一度作成した後で更新しないと、情報ページがすぐに陳腐化してしまうという問題がある。このため、WWWとデータベースの連携方式が注目されている。この理由の一つは、データベースの内容をWWWの情報ページに出力すれば、目的に合った新鮮な情報を随時発信でき、企業内のデータベースをWWWのインタフェースを通してアクセスすることができるためである。

ところが、これまではデータベースとWWWの連携方式はWWWサーバが規定するCGI（Common Gateway Interface）[1,2]に則ったC言語あるいはUNIXのshell等でアプリケーションを開発する必要がある。そこで、CGIによるゲートウェイアプリケーションを用いてWWWとデータベースを連携する実験を行ったところ、以下に示すような問題が明らかになった[1]。

(1)CGIによるプロセス起動が必要なこと、ならびに起動されるプロセスごとにデータベースのオープン／クローズが必要になるため、性能が低下する。

(2)WWWのプロトコルではページ毎に通信の接続／切断が行われるため、複数ページに跨ってデータベースのトランザクション制御を行うことができない。

(3)C言語やschellスクリプトを用いて業務処理ごとにデータベースと情報ページ間の連携処理の作成が必要であり開発効率が低下する。

NTTソフトウェア研究所では、これらの問題を解決するために、HTMLを拡張してSQLや簡易言語で記述されたプログラムを起動できるスクリプト言語機能とセッション処理機能を実装したWebBASEを開発した。WebBASEでは、クライアント／サーバ型のシステム開発のためのミドルソフトウェアVGUIDE [3]をベースにしており、主要なデータベース製品をそのまま使える。また、CGIを用いずにWWWとのインタフェースを利用しているため、高速処理を実現でき、高トラフィック時でもレスポンスが良いという特徴がある。さらに、HTMLとSQLを混在させてサービス業務を記述できるWebBASEスクリプトにより、短期間でサービスを実現できる。

2．8．2　WebBASEのシステム構成

WebBASEのシステム構成を図2.8-1に示す。

(1)WebBASEは一般のWWWサーバ（CERNやNCSA、Netscape communication server等）と同一のマシン上で動作する。

(2)VGUIDEとDBMSを搭載したDBサーバは、WebBASEと同一マシンでも構わないが、ネットワークからの侵入／処理の分散化を考えると別マシンにする方が望ましい。

(3)WWWブラウザとWWWサーバ間は、セッションの開始のみのCGIによるWebBASEの起動の要求、およびその他通常のHTTPの処理を行う。

(4)WWWサーバはセッションモードの場合のみWebBASEをCGI起動する。

(5)WebBASEとVGUIDEの間は、VGUIDEプロトコルによるSQLの実行依頼／結果の取得、リモートプロシジャコール（RPC）／その結果の取得を行う。

(6)VGUIDEとデータベース間は、データベースから見れば、通常のデータベースアクセス処理と同じである。

VGUIDEはクライアント／サーバ型の情報システム開発のためのミドルウェアで、次の特徴を持つ。

(1)データベースアクセス用の簡易言語（4GL）により短期間でシステムを開発できる。

(2)高トラフィックの大規模システムにも適用できるリアルタイム制御機能により、スケーラビリティが高い。

(3)多様なDBMSおよびベンダマシン環境に対応しており、アプリケーションの移植性が高い。

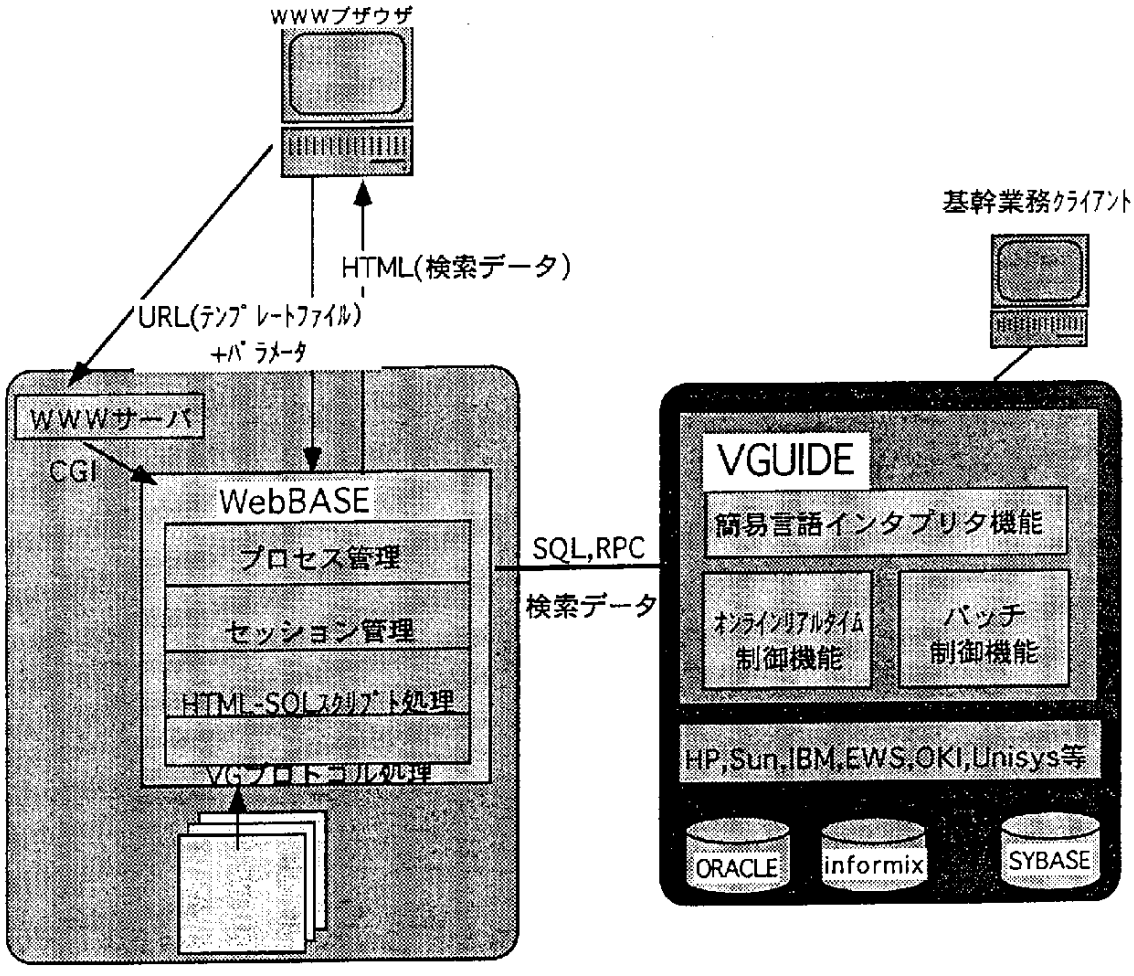


図2.8-1　WebBASEのシステム構成

WebBASEでは、このような特徴を持つVGUIDEをDBサーバに利用することにより以下を可能とした。

(1)DBMSに依存しないオープンな環境をVGUIDEが提供しているのでユーザが今まで基幹業務などで利用しているデータベースをそのままWWWに利用できる。

(2)データベースアクセスに必要な資源を事前に確保しておくTPモニタ機能の利用により高速なレスポンスを実現する。また、DBサーバへの負荷が少ないため、従来の基幹業務にWWWからのサービス業務を新たに追加することができる。

(3)データベース管理システムの差異をVGUIDEが吸収するので、WebBASEでデータベース管理システムの違いを意識する必要がない。

2．8．3　WebBASEの内部処理の流れ

WWWブラウザからの要求に従ってWebBASEプロセスはWWWサーバからCGI起動され、セッションを開始する。WebBASEがデータベースアクセス命令を含むテンプレートファイルの処理を行う。通常のWWWサーバがHTMLファイルや画像ファイルを転送する。図2.8-2にWebBASEの内部処理の流れを示す。

(1)サービス開発者がWebBASEスクリプト言語を用いて、業務サービスの内容に応じて、テンプレートファイルを作成する。

(2)サービス提供の前に、DBサーバおいてVGUIDEを起動しレディ状態にする。この段階で、VGUIDEプロセスは常駐化し、データベースがオープンされた状態になる。

(3)WWWブラウザから、フォームでGETあるいはPOSTによりWebBASEの起動とテンプレートファイル名と必要なパラメータをWWWサーバに要求する。

(4)WWWサーバはCGIによりWebBASEプロセスを起動する。

(5)起動されたWebBASEは一意な識別子を生成する。

(6)指定されたテンプレートファイルを読み込む。

(7)テンプレートファイルに記述されたスクリプトの処理を行う。

(a)起動時に渡されたパラメータを変数への代入。

(b)DBサーバ(VGUIDE)への接続。

(c)拡張子がSQLであれば、DBサーバへ処理の要求。

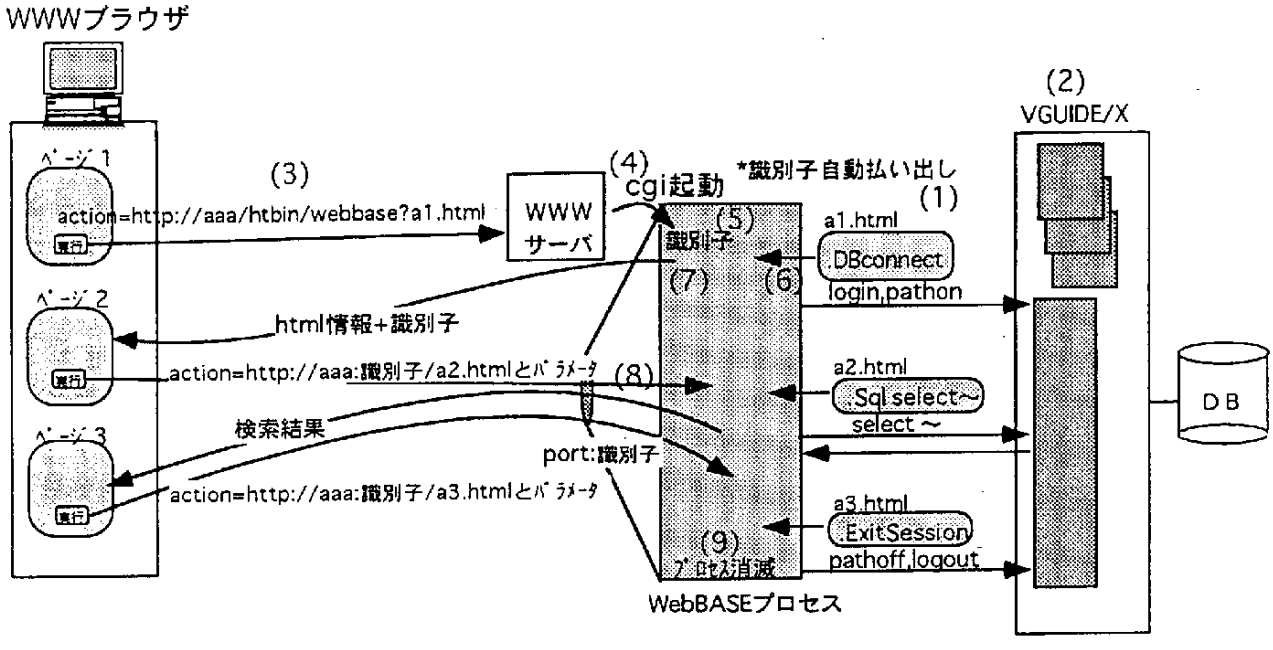


図2.8-2　WebBASEの内部処理の流れ

(d) SQLがSELECTであれば、検索データの取得。

(e) 取得データの変数への代入およびHTMLテンプレートとの合成。

ここで、検索結果はカラム順に変数@1,@2･･･に格納される。

(f) 合成されたデータをHTTPに変換し識別子と共にWWWサーバを経由してブラウザに送信する。

(8) WWWブラウザからの要求は、識別子（port番号と一意データ）を用いて、WWWサーバを経由することなく、サーバ上に常駐しているWebBASEプロセスに対して行われる。WebBASEプロセスが常駐化されている間、変数の内容やDBサーバの排他状態は保持される。要求が来れば(6)~(8)を繰り返す。

(9) テンプレートファイル中にEXITSESSIONコマンドが指定された場合、WebBASEプロセスが終了する。

2．8．4　WebBASEスクリプト

WebBASEスクリプト言語の構成要素は、以下のいずれかである。

(a) 画面表示のためのHTML文

(b) データベースアクセス用のSQL文

(c) VBA互換の処理文

データベースアクセス用のWebBASEスクリプトの基本的な構文には、以下の１２種類がある。ここで、WebBASEスクリプトでは、HTML文(a)と区別するために、拡張部分(b)(c)の行頭にドット（'.'）を付けている。

(1) データベース（VGUIDE）の接続

.DBConnect <DB名>

(2) データベース（VGUIDE）の切断

.DBDisconnect

(3) SQL文の実行

.Sql <SQL文>

(4) 検索結果の値による取得

.LoopFetch

<変数展開並び>

.End Fetch

ここで、<変数展開並び>の形式は以下のようである。

<表示名>: @<カラム番号> {, <表示名>: @<カラム番号> }

ただし、表示名は情報ページ上で表示されるフィールド名であり、カラム番号に設定された検索結果の

値が情報ページ上で対応するフィールドの値として表示される。

(5) 検索結果の文字列による取得

. <変数名> = DBgetcol(<カラム番号>)

(6) 検索結果のファイルへの出力

.DBWriteFile <ファイル名>

(7) RPC（リモートプロシジャコール）の実行

.DBCpexec <CP名> {, <パラメータ変数名>}

(8) RPC実行結果の取得

.DBdownload

(9) 条件分岐文

.If <条件>　Then

.End If

(10) 代入文

.Let <変数名> = <値>

(11) 外部コマンドの実行

`<コマンド名>`

(12) セッション終了

.ExitSession

以下では、WebBASEスクリプトの例を示す。また、このスクリプトに対応する情報ページの表示例を図2.8-3に示す。

(例1) 指定した金額よりも多い給与をもらっている社員を検索する。

```
<HEAD><TITLE>社員データベース</TITLE></HEAD>
<BODY><H1>社員情報検索結果</H>
<HR>
給与が@SAL円以上の社員
.DBConnect "JINJIDB"
.Sql "SELECT ENO,ENAME,JOB,SAL,PHOTO FROM EMP" & DBCndNum("", "SAL" , ">", sal)
.Loopfetch
    顔写真:<img src="/kao/@5.jpg"><p>
    社員番号: @1<p>
```

氏名: @2<p>

役職: @3<p>

給与: @4 円><p>

<HR>

.End Feach

.DBDisconnect

.End

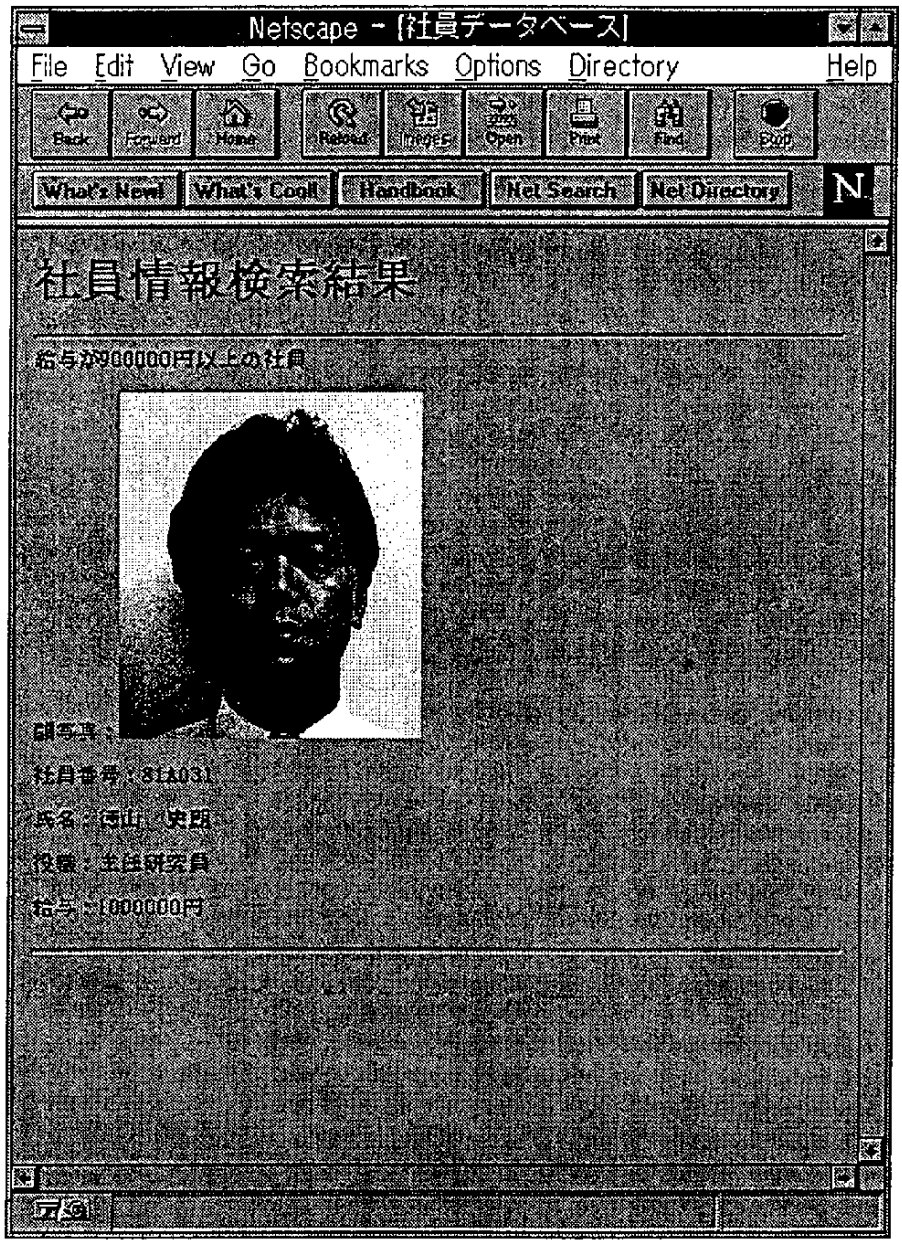


図2.8-3　WebBASEスクリプトを用いた情報ページの表示例

2．8．5　WebBASEを用いたシステム構築事例

WebBASEが最初に適用されたシステムは、筑波にある金属材料技術研究所における物質スペクトル検索システム[2,4]である。このシステムは、９５年９月からインタネット上で情報発信を始めた。その後、表2.8-1 に示すような数多くのシステムがこれまでに開発されている。この表では、システムの分類と各システムが扱う記号情報ならびにマルチメディア情報の内容を示している。WebBASEの適用システムは、情報発信型、情報案内型、情報共有型、EC型に分類できる。

表2.8-1　WebBASEを用いたシステム構築事例

分類	システム事例	記号情報	マルチメディア情報
情報発信	物質スペクトルデータベース	測定情報	グラフ情報
	新聞記事データベース	SGML文書	音声情報
	古代生物データベース	生物情報	グラフィックス情報
情報案内	ＮＴＴディレクトリ	URL情報	
	フリーダイアル検索	電話番号情報	
情報共有	経営管理システム	購入伝票情報	グラフ情報
	技術資料データベース	SGML文書	
	ソフトウェア情報管理システム	SGML文書	
	写真データベース	撮影情報	イメージ情報
	不動産情報システム	不動産情報	イメージ情報
	営業支援	文書情報	
エレクトロニックコマース	ホテル予約	空室情報	イメージ情報
	ゴルフ場予約	空コース情報	イメージ情報
	サービス申込み受付	サービス情報	

(1)情報発信型システムの例としては、先の物質スペクトルデータベースを始め、新聞記事データベースや古代生物データベースがある。新聞記事データベース・システムは、マルチメディア実験の一貫として実現したシステムであり、新聞社から毎日提供されるSGML化された記事情報をデータベース内に蓄積しておき、インタネット上で検索利用できる。

(2)情報案内型システムの例としては、インタネット上で新着ページ情報を検索できるNTTディレクトリ

やフリーダイアル番号を業務種別分類やキーワードにより検索できるフリーダイアル検索システムがある[5]。

(3)情報共有型システムの例としては、経営管理システム[6]を始め、SGML形式で格納された情報を検索してWWWブラウザ上で提示できる技術資料データベースや質問応答事例検索システムがある。また、イメージ情報を検索できる例としては、写真データベースや不動産情報システムがある。情報共有型のシステムは、インタネット上で一般利用者が使うのではなく、企業内で利用するイントラネット型の情報システムである。

(4)EC型システムの例としては、インタネット上で実現されたホテル予約システム、ゴルフ場予約システム、サービス申込み受付システムなどがある。

2．8．6　WebBASEによるシステム開発の効率化

表2.8-1に示した各システムは数週間から数カ月という極めて短期間で開発されている。以下では、WebBASEを利用したマルチメディア情報システムの具体的な事例に基づいて開発期間を示す。

(1)社員情報管理システム

［概要］検索条件を指定し、社員の氏名や年齢、住所および写真を表示するシステム

［特徴］写真データはファイルとしてサーバ上に格納。社員データベース中に写真名を保持。検索結果を一時的に保持しておく機能により一人一ページのカード形式で表示しページめくりが可能。

［開発データ］情報ページ数：3、テーブル数：1、開発期間約2週間

(2)旅行代理店業務システム

［概要］目的地、費用、ホテルクラスなどを検索条件とし、旅行概要および旅行先の写真イメージを表示し、お客様が気に入れば旅行の予約を行うシステム。

［特徴］営業所窓口に設置されている専用アプリケーションによる複雑な検索や予約などの高トラフィック処理とWWWによる検索処理との共存を実現している。

［開発データ］情報ページ数：4、テーブル数：4、開発期間約4週間

(3)スペクトルデータ分析システム

［概要］金属のある環境下で測定されたスペクトルデータをデータベース中に格納しており、種々の検索条件で検索されたスペクトルデータをグラフ表示する。

［特徴］会員はWWWで各種検索条件入力、検索結果のスペクトル詳細データのダウンロード、クライアント上の特殊ビュアーでグラフ化が可能。非会員はWWWで単純な検索条件入力とサーバ上で作成された検索結果のグラフのみを表示

［開発データ］情報ページ数：4、テーブル数：1、開発期間約3週間

(4)質問応答事例検索システム

［概要］窓口やお客様応対で実際に発生した質問事項と、その適切な応答事例をデータベースに関連付けて格納しておき、キーワードや階層分類などの検索条件で検索して表示する。

［特徴］キーワード検索と階層分類に基づく段階的な絞り込みが可能。また、応答事例とともに関連する質問を自動的に提示。

［開発データ］情報ページ数：7、テーブル数：5、開発期間約3週間

これらの事例から分かるように、WWWとデータベースとをWebBASEを用いて連携させることにより、マルチメディア情報システムの構築が容易にしかも短期間にできるようになった。このように従来のシステム構築方法と比較して、WebBASEでは工数を大幅に削減できる理由は、以下のようである。

(a) マルチメディアデータの取得／転送／表示はWWWクライアントの機能を利用することができる

(b) データベースに格納されたマルチメディアデータの管理情報のアクセス処理をWebBASEスクリプト言語で記述するだけで済む

この事実は、クライアントサーバ型の情報システムの開発手法がWWWのような情報ブラウザと、VGUIDEのような分散型アプリケーション開発環境の組み合せにより劇的に変革されることを示している。

2．8．7　WebBASEを用いたダウンサイジング事例

大型コンピュータをホストとし、専用端末からホスト上のデータベース情報を利用するような情報システムの場合、専用端末の数が限られるため、誰でもが自席の端末から自由に必要な予算とその執行状況を迅速かつビジュアルに把握することができない。また、経営企画部などの予算を管理する部門では、予算の計画に対する進捗や実績を全社単位、部単位、グループ単位など種々のレベルでグラフ化するなど、経営状況を多角的に分析する必要がある。ところが、先に述べた大型コンピュータ中心の情報システムでは、このようなきめ細かい非定型業務については考慮されていないため、専用のパソコン端末からSQL文を入力して、その結果を出力した上で、もう一度、EXCELに値を入力してグラフ化する必要があった。このような作業稼働の省力化と操作性の向上ならびに、いつでもどこでも予算状況を見ることができるようにするため、予算データをUNIX上で管理し、VGUIDEとWebBASEを用いてデータの検索を行う経営支援システムを開発した[6]。

(1)経営支援システムの構成

経営支援システムの構成要素は以下の3つである（図2.8-4）。

(a) ホストコンピュータからUNIXサーバにデータを転送する2台のPC

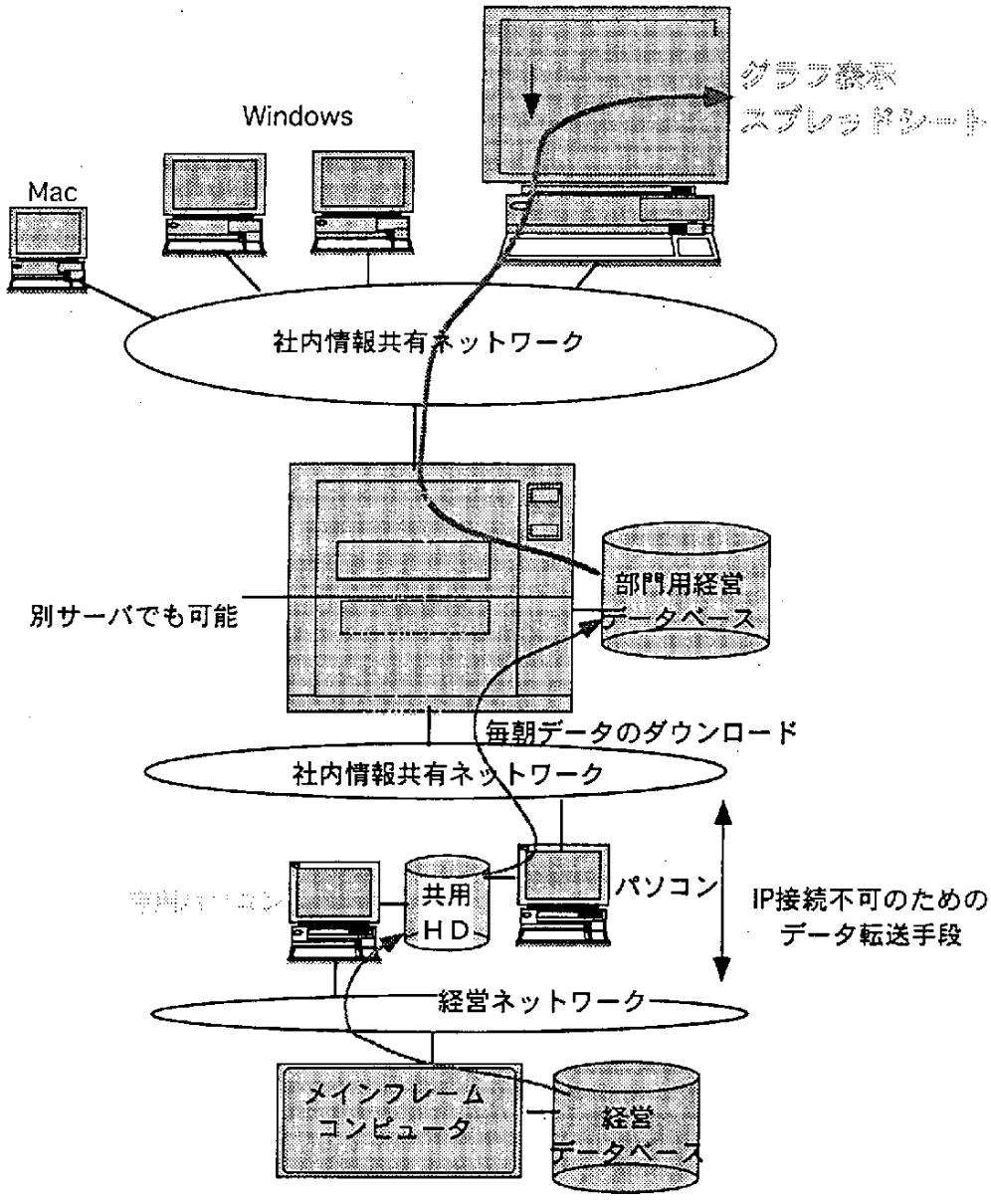


図2.8-4　経営支援システムの構成

(b) データベースを利用するためのVGUIDEとWWWサーバとVGUIDEを連携するWebBASEを搭載したサーバ

(c) WWWブラウザとEXCELをインストールしたサービス利用者の自席にあるWindowsかMacintoshを搭載するパソコン

(2) ホストコンピュータとUNIXサーバの接続方法

ホストコンピュータは経営管理用ネットワークに接続されている。経営管理サービスを行うUNIXサー

バは一般社員が利用する社内情報共有用ネットワークに接続されている。このため、社内に存在する異なるネットワーク間でデータ転送を行う必要があった。２つのネットワークのセキュリティを確保するためには、物理的にIPパケットが通るような接続形態は許されない。この条件の下でデータ転送を行う方法には、電話回線やINS- 64 などの公衆網を利用する方法があるが、この場合、年度末には約１０MBになるデータを毎日転送することになるため、データ転送時間が問題となる。このため、２つのネットワークに接続された２台のPCをSCSIケーブルで接続した共用ハードディスク装置（HD）で連携する方式を採用した。共用HDには２台のPCからの同時アクセス時に矛盾が発生しないように排他制御装置を使用している。この方式には、以下の利点がある。

(a) PCとホスト、サーバ間はTCP/IPであるのでデータ転送時間が速いだけでなく、IPパケットが漏れることはない。

(b)PCを利用するのでホストコンピュータのデータベースにアクセスするためのPCで動作する既存のエミュレータを利用できる。

(c) UNIXサーバとの間はFTP、RSHコマンドを実行する市販プログラムが用意されているので開発工数が少なくて済む。

(３) WWWとデータベースの連携方式

経営支援システムでは、以下の３種のスクリプトファイルを作成した。

(a) 検索条件を設定するためのHTMLファイル

(b) 設定された条件を基にデータベースの接続／検索／データの取得を行うためのWebBASEスクリプトを記述したテンプレートファイル

(c) WebBASEスクリプトで記述できない部分をWebBASEから外部起動して処理するためのawkスクリプトファイル

処理の流れを以下に示す。

(a) 検索条件入力用のHTMLファイルやメニュー用のHTMLファイルは、通常のhttpd（WWWサーバ）がWWWブラウザに送信し、表示される。

(b) 利用者はWWWブラウザに表示された画面上でラジオボタンを選択して検索条件を設定し、実行ボタンをクリックする。

(c) 設定された検索条件とテンプレートファイル名のデータが、WWWブラウザからWebBASEに直接通知される。

(d) WebBASEはテンプレートファイルを読み込み設定された条件を元にSQL文を組立てDBサーバ（VGUIDE）に処理を依頼する。

(e) DBサーバからの処理結果を取得し、HTMLへの加工を行い、WWWブラウザにデータを送信する。

(4) WWWブラウザとEXCELの連携

サービス利用者の自席のWindowsあるいはMacintoshのPCには、NetscapeなどのWWWブラウザとEXCELをインストールしておく。WWWブラウザのHelp Applicationを利用して、Content/typeがtext/csvあるいはtext/sylkの場合にはEXCELを起動するように設定する。また、EXCELに渡されたデータを罫線付きの表やグラフを自動的に作成する場合、そのためのEXCELマクロを組み込む。このマクロはWWWサーバ上に格納されているのでWWWブラウザのファイル転送機能を利用して自分のPCのEXCELのマクロを登録するディレクトリ（Macintoshは特殊ホルダ）にダウンロードする。EXCELのマクロでは、渡されたデータの先頭部分（月別か部別かなど）で自動判断し、罫線処理や集計処理あるいはグラフ処理を行うことができる。

2．8．8　VGUIDEとWebBASEによる分散型システムの構成法

VGUIDEとWebBASEとを組み合わせることにより、図2.8-5に示したような多様な形態のクライアント／サーバ・システムが構成できる。この図で示したコンポーネントを必ずしもすべて利用する必要はない。たとえば、市販GUI製品とリモートSQLを用いることにより、2層モデルを簡単に実現できる。このモデルは小規模な分散システムを簡易に開発する場合に適している。市販GUI製品とRPC、CPの組み合せにより3層モデル[8,9,10]を実現できる。このモデルでは、文献で示したように大規模な分散システムを構築できる。さらに、WWWブラウザとWebBASEを用いてクライアント機能を実現し、WebBASEスクリプトでリモートSQLを利用してデータベースを参照すれば、表示機能をサーバ側にも一部配置した新しい形態の3層モデルを構築できる。このモデルの特徴は、表示機能の開発が従来のGUI製品を用いたプログラミングではなく、HTMLを用いたハイパテキスト作成という、いわば、文書作成へと大幅に簡易化されている点である。上述した経営支援システムの場合、既存システムの情報をWWWサーバから利用する3層モデルにより実現した。すなわち、経営支援システムのデータ層では、ホストコンピュータ上のレガシーシステムのデータベースから必要なデータだけを抽出してUNIX上のデータベースに格納している。経営支援システムの機能層では、VGUIDEとWebBASEを利用して業務アプリケーションをWebBASEスクリプトで作成している。経営支援システムのプレゼンテーション層では市販のNetscapeとEXCELを利用してクライアント側のOSと独立な利用者インタフェースを作成している。このように、WWWを利用する場合でも、最近のクライアント／サーバ型システムで利用が広まりつつある3層モデルを適用することにより、業務の拡張や追加を行う場合、テンプレートファイルの改造・追加で対処でき、維持管理作業を効率化できる可能性が高い。

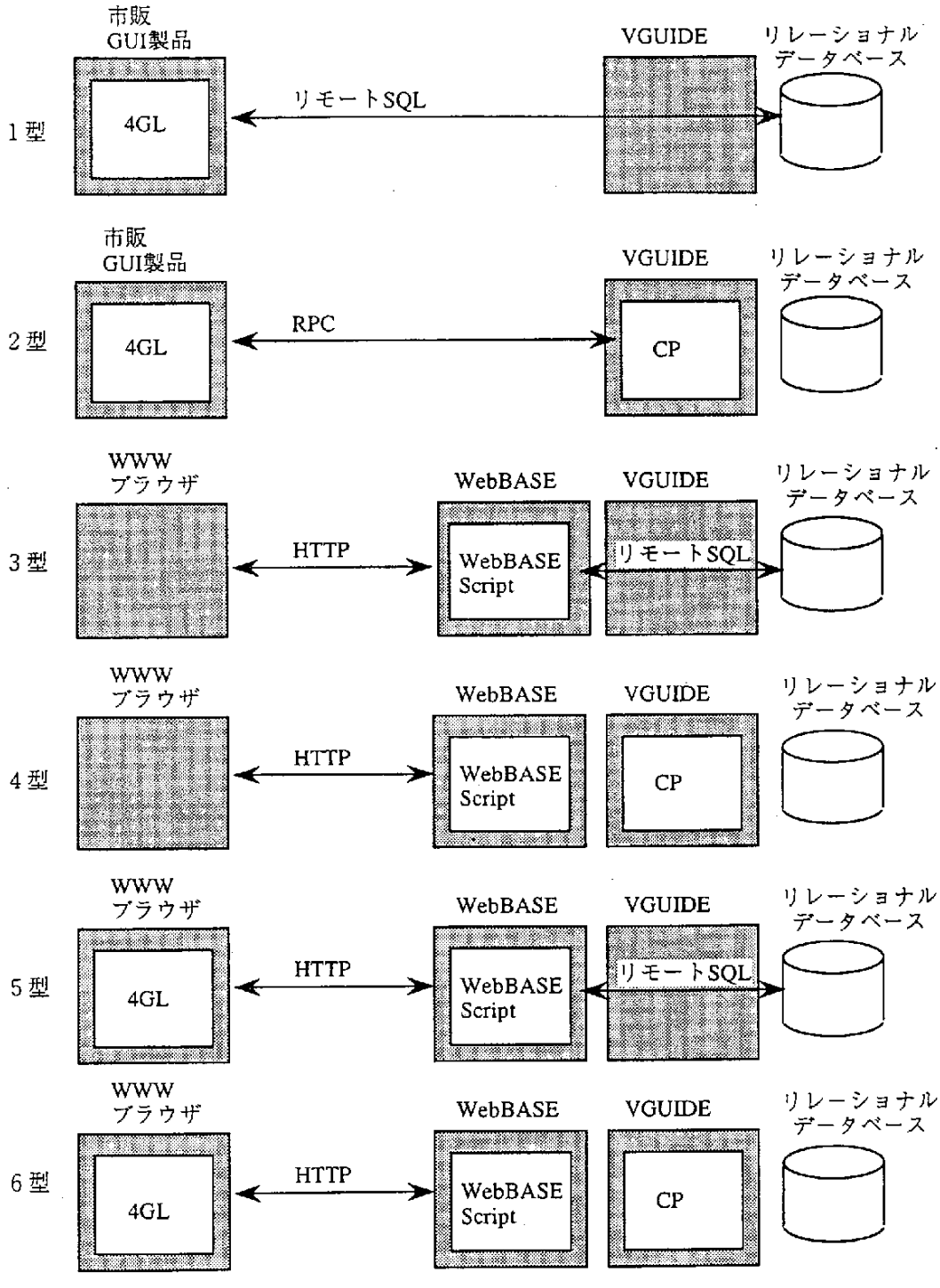


図2.8-5　VGUIDEとWebBASEによる分散型システムの構成法

WebBASEスクリプトから、VGUIDEの4GLであるCPを起動すれば、CPで記述された共通的な業務ロジックを再利用できる。実際、業務サービスを構築する場合、通常はテンプレートファイルにスクリプトを作

成する方が容易であるけれども、WWWブラウザで入力した値を、データベース中の複数のテーブルに格納するなど、複数のSQL文を発行する必要がある場合は、VGUIDE側にコマンドプロシジャCPを作成しておき、WebBASEではRPC実行を記述する方が、性能面／維持管理面では有利である。この場合、サーバ側に表示層と機能層を分割して持つので4層モデルと呼ぶべきだろう。また、WWWブラウザ上にExcellやJavaのアプリケーションを配置することにより、さらに、きめ細かい分散システムを構築できる。とくに、WWWサーバの場合、クライアント側に処理を置かなければ、クライアントサーバ方式の特徴である負荷分散のメリットが享受できないので、同時接続クライアント数が大量になるようなシステムの構築は困難である。このような問題に対処するためには、従来の3層モデルとWWWを用いた4層モデルの併用や、WWWブラウザ側でも何らかの機能層の処理を受け持つ必要がある。いずれにしても、VGUIDEとWebBASEの組み合せにより、多様な形態のクライアントサーバシステム構築に対応できる。今後、業務分野やシステム要求に応じて、適切なシステムのアーキテクチャを選択していく基準を確立していくことが重要になると思われる。

【参考文献】

1) 元田敏浩, 徳丸浩二: WWWとデータベースサービスとの連携方式の検証, 信学技報KBSE-7 (1995).

2) 黒川裕彦, 徳丸浩二, 元田敏浩, 渡部一成: VGUIDEを用いたマルチメディア情報システムの構築, NTT技術ジャーナル, Vol.7, No.12, pp.82- 85 (1995).

3) 川崎隆二, 黒川裕彦, 山本修一郎: 簡易言語による大規模分散型システム構築環境VGUIDEの適用, 情報処理学会情報システム研究会, Vol.56-2, pp9-18 (1995).

4) Yoshihara, K. and Yoshitake, M., Construction of the Surface Analysis Network Database, Journal of Surface Analysis, Vol.1, No.3, pp. 369- 373 (1995).

5) 徳丸浩二, 山本修一郎: WebBASEのマルチメディア・ディレクトリ・システムへの適用, NTT技術ジャーナル, Vol.8, No.5 (1996).

6) 黒川裕彦, 伊藤光恭, 岩城勝博: VGUIDEとWWWを用いたダウンサイジング事例,NTT技術ジャーナル, Vol.8, No.5 (1996).

7) 山下亮, 中垣清文: 最新のコンピュータ技術を用いたサービスフロント業務, NTT技術ジャーナル, Vol.8, No.3 (1996).

8) Gray, J. and Edwards, J., Scale up with TP Monitors, BYTE, APRIL, pp.123- 128 (1995).

9) 神谷芳樹: 3階層モデルを実現するVGUIDE, NTT技術ジャーナル, Vol.7, No.12, pp.80-81 (1995).

10) 高田信一, 畑恵介, 山本修一郎: 3層クライアント／サーバ型情報システムの設計手法, NTT技術ジャーナル, Vol.8, No.1, pp.74- 77 (1996).

11) WebBASEホームページ, http:// robin. sl. cae. ntt. jp/ vgindex.html

平成８年３月発行

発行所　財団法人　日本情報処理開発協会
東京都港区芝公園３丁目５番８号
機 械 振 興 会 館 内
ＴＥＬ（03）3432－9384

印刷所　有限会社　盛光印刷所
東京都千代田区飯田橋４丁目６番３号
宝第３ビル
ＴＥＬ（03）3264－1851

07－R003