０８－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９年３月

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８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、エンドユーザは、パーソナルコンピュータやネットワークを駆使した新しい業務改善や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したい機能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、アプリケーション統合環境のモデル、組み立て型開発の方法論、応用事例等について調査検討を行ったとともに、平成８年１０月には国際会議に海外調査員を派遣し、オブジェクト指向やソフトウェアアーキテクチャに関する最新技術情報を収集しました。

本報告書は、平成８年度における研究成果を取りまとめたものです。第１章は技術動向、第２章はアプリケーション統合環境、第３章は開発方法論、第４章はアプリケーション開発事例となっております。

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９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）富士通研究所マルチメディアシステム研究所 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技術企画部主任研究員

執筆協力者（敬称略）

安達　　進　（株）テクノプロジェクトミドルウェア推進部部長

小山　清美　（株）日立製作所情報システム事業部生産技術部

酒井　之子　日本アイ・ビー・エム（株）AD技術支援

田岡　賢輔　日本テキサス・インスツルメンツ（株）ソフトウェア事業部カスタマ・サービスマネージャ

津田　　宏　（株）富士通研究所マルチメディアシステム研究所メディア統合研究部

戸松　豊和　日本サン・マイクロシステムズ（株）システム技術本部第２技術サポート部

初田　賢司　（株）日立製作所情報システム事業部生産技術部主任技師

目　次

1．技術動向 …… 1

1．1　CORBA …… 1

1．2　ActiveX …… 14
1．2．1　はじめに 14
1．2．2　ActiveXとコンポーネントオブジェクトモデル 18
1．2．3　分散オブジェクトシステム 27
1．2．4　インターネット/イントラネットコンテンツ制作とアプリケーション開発の新手法ActiveX とは 32
1．2．5　ActiveXとMicrosoft Windowsの進化 35

1．3　Java Beans …… 38
1．3．1　Java Beans の概要 38
1．3．2　Java Beans の基礎 43
1．3．3　Java Beans を実現するための仕組み 44
1．3．4　Java Beans の利用例 48
1．3．5　BDK と BeanBox 50

1．4　ソフトウェアエージェント …… 51
1．4．1　エージェントについて 51
1．4．2　ソフトウェアエージェントの概要 52
1．4．3　Telescript 55

1．5　Web-CORBA連携 …… 62
1．5．1　背景 62
1．5．2　3階層モデルにもとづくWeb-CORBA連携システム 65
1．5．3　事例 68
1．5．4　まとめ 72

1．6　Web-データベース連携 …… 73
1．6．1　はじめに 73
1．6．2　WWW－データベース連携の要素技術 74
1．6．3　WWW－データベース連携ツールの分類 78
1．6．4　WWW－データベース連携アプリケーションの適用形態 83
1．6．5　WWW－データベース連携の課題 85
1．6．6　まとめ 88

2．アプリケーション統合環境 …… 89

2．1　エンドユーザ向けアプリケーション統合環境 …… 89
2．1．1　コンポーネントベース開発 89
2．1．2　エンドユーザプログラミング 98
2．1．3　アプリケーション統合環境モデル 108

2．2　Composer/Arranger/WebCenter …… 112
2．2．1　Composerとモデル・ベース開発 112
2．2．2　コンポーネント・ベース開発 117

2．2．3　部品の利用　125
2．2．4　まとめ　128

3．開発方法論 …… 129

3．1　概　説 …… 129

3．2　HIPACE for APPGALLERY …… 138
3．2．1　HIPACE for APPGALLERYの特徴　138
3．2．2　HIPACE for APPGALLERYの開発プロセス　141
3．2．3　まとめ　148
3．3　テンプレート型開発方法論 …… 149
3．3．1　はじめに　149
3．3．2　テンプレート型開発の概要　149
3．3．3　ビジネスオブジェクトと4層アーキテクチャ　154
3．3．4　テンプレートの開発と利用方法　157
3．3．5　おわりに　164
3．4　M-base：「ドメインモデル≡計算モデル」を志向した開発技法 …… 165
3．4．1　はじめに　165
3．4．2　目的と対象　165
3．4．3　モデリングプロセスの概要　166
3．4．4　開発環境の概要　168
3．4．5　モデリング＆シミュレーション　169
3．4．6　スクリプト言語　172
3．4．7　コンポーネントウェア抽出実験　174
3．4．8　おわりに　176
3．5　Visual Modeling Technique（VMT） …… 177
3．5．1　VMTのコンポーネント　177
3．5．2　VMT ライフ・サイクル・モデル　178
3．5．3　VMT開発プロセス　179
3．5．4　VMTモデリング・プロセス　180

4．アプリケーション開発事例 …… 183

4．1　APPGALLERYによる開発事例 …… 183
4．1．1　日本生命の概要　183
4．1．2　契約管理業務の概要と問題点　183
4．1．3　システム概要　184
4．1．4　APPGALLERYの適用　185
4．1．5　性能・運用上の留意点　191
4．1．6　今後の課題　192
4．1．7　おわりに　192
4．2　HOLON/VPと適用事例 …… 193
4．2．1　HOLON/VP の特徴　193

4．2．2　アプリケーションの構築　194
4．2．3　HOLON/VPの効果と今後の展開　201
4．2．4　HOLON/VPの適用事例　202
4．2．5　コンポーネントウェアの可能性　203
4．3　IPアプリケーション開発事例　205
4．3．1　はじめに　205
4．3．2　医薬品開発プロセスとGCP/PMS　205
4．3．3　GCP/PMS支援システムとは　206
4．3．4　CRFの設計と入力　208
4．3．5　システムの特長　210
4．3．6　なぜIPか　211
4．3．7　IPでどう作るか　212
4．3．8　おわりに　216
4．4　VisualAgeの事例　217
4．4．1　VisualAge for Smalltalk＆ネットワークコンピューティング　217
4．4．2　適用事例１・・・A会社　コールセンター受付システム　218
4．4．3　適用事例２・・・B会社　システムアドミニストレーションシステム　219
4．4．4　適用事例３・・・C会社　ディレクトリーサービスシステム　219
4．4．5　VisualAge Smalltalkとオブジェクト指向　220
4．4．6　適用事例４・・・D会社　商品分析支援システム　220
4．4．7　適用事例５・・・E会社　220
4．4．8　適用事例６・・・地図パーツ　221

1. 技 術 動 向

1．技術動向

1．1　CORBA

(1) CORBAのアーキテクチャ

CORBA(Common Object Request Broker Architecture)はオブジェクト指向でクライアント／サーバアーキテクチャのアプリケーションを開発するために必要となるオブジェクト連携機構としてOMG(Object Management Group)により規定された[8, 10]。このような機構を分散オブジェクト環境と呼ぶ。OMGでは、CORBAを中心にアプリケーションを構築するために必要となる様々な部品やインタフェースに関する広範な標準が検討されている。本節では、1997年1月時点の資料に基づいてCORBAと関連する分散オブジェクト環境について紹介する。

(a) OMA参照モデル

分散オブジェクト環境CORBA全体の、いわば参照アーキテクチャがOMA(Object Management Architecture)として規定されている[7]。アーキテクチャは、その構成要素と要素間の関係として定義する。OMAでは、それぞれ、オブジェクトモデルと参照モデルとして規定されている。

図1.1-1の参照モデルに示すように、OMAは次の5つの要素から構成される。アプリケーションを構成するアプリケーションオブジェクト群がCORBAを介して共通サービスやドメインサービスを利用しながら要求された機能を実現する。

図1.1-1　OMA参照モデル

(i) ORB(Object Request Broker)

ORBはＯＭＡの核である。分散したオブジェクトを組み込むためのソフトウェアバスである。

(ii) CORBAサービス

CORBAを実現するために必要となる様々なサービス（機能）を提供するオブジェクト群である。オブジェクトの名前管理や生成・消滅などのライフサイクル管理などが規定されている。

(iii) ファシリティオブジェクトあるいはCORBAファシリティ

CORBA上でアプリケーションを実現するために必要となるドメインに依存しないコンポーネント群である。複合文書の管理や国際化などが規定されている。

(iv) ドメインオブジェクト

特定ドメインのアプリケーションを実現するためのコンポーネントである。会計、運輸、製造、エレクトロニックコマース、通信、医療など様々なドメインのコンポーネントが検討されている。

(v) アプリケーションオブジェクト

アプリケーションを構成するオブジェクト群であり、個々のアプリケーション開発者が提供する。

(b) CORBAのオブジェクトモデル

CORBAは、オブジェクトの実装よりオブジェクト連携の機構を規定するので、オブジェクトはインタフェースで表わされる。CORBAのインタフェース定義は図1.1-2に示す構造をとる。

CORBAの1つのオブジェクトは１つの識別名をもつインタフェースとしてインタフェース定義言語(IDL: Interface Definition Language)で定義される。1つのインタフェースは1つもしくはそれ以上のオペレーションから成る。オペレーションはメソッドに相当する。オペレーションは、実行形態を規定する実行セマンティクス(Execution Semantics)、戻り値の型、識別子(Identifier)、すなわちオペレーションの名前、シグネチャ(Signature)で定義される。シグネチャはオペレーションのパラメータ、例外定義、クライアントのプロパティ情報を渡すコンテキスト定義から成る。パラメータは入出力のモード、型、パラメータ名で定義される。整数型などのデータ型が標準で規定されている。また、BAD_PARAM、BAD_OPERATIONなど26種類の標準例外が規定されている。インタフェースはインタフェース継承(Interface Inheritance)を通して上位インタフェースのオペレーションや属性なを再利用できる。

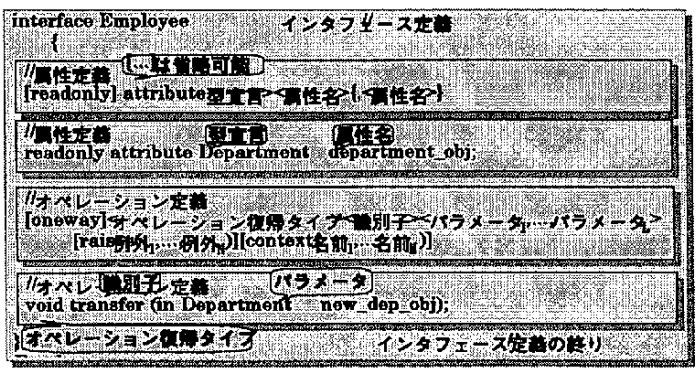


図1.1-2　CORBAのインタフェースモデル

CORBAでは、オブジェクトの実行セマンティクスとして次の2つのモデルが規定されている。

(i)最大１回(At-Most-Once)：オペレーションの最初に実行セマンティクスを省略した場合に取られる。これは、起動がかならず成功する場合や、失敗時の例外が指定されていれば正確に１度(Exactly-Once)起動される。

(ii) ベストエフォート(Best-Effort)：最大1回でかつone-wayが指定された場合のように応答を要しない起動方法。従って、出力のあるパラメータは指定できない。

(２) ORB

(a) ORBアーキテクチャ

ORB (Object Request Broker)のアーキテクチャを図1.1-3に示す。

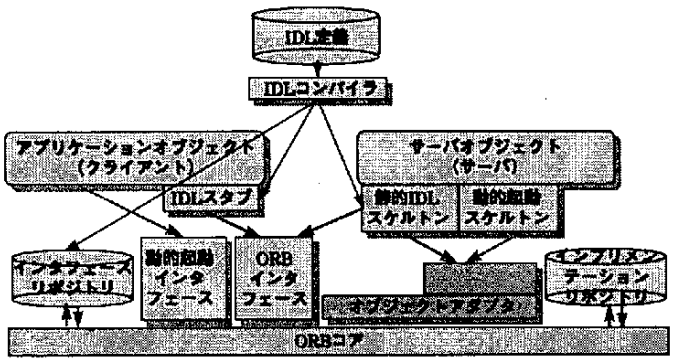


図1.1-3　CORBAアーキテクチャ

ORBアーキテクチャは次の要素から構成される。留意すべきは、ORBそのものがクライアント／サーバアーキテクチャをとることによりオブジェクトの分散に対する柔軟なアーキテクチャとなっていることである。すなわち、クライアント側はサーバ内のオブジェクトの詳細などを知る必要がなく、一種の情報隠蔽が可能となる。

(i) ORBコア：ORBのカーネルであり、上位の要素間のコミュニケーションなどの基本的なサービスを提供する。

(ii) IDLスタブ：特定の言語で実装されたクライアントオブジェクトのインタフェースとORB共通のインタフェース表現との整合をとる。

(iii) 動的起動インタフェース：動的起動を行うためにインタフェースリポジトリからインタフェース情報を取得し、リクエストを生成する。

(iv) インタフェースリポジトリ：IDLで定義されたインタフェースのパラメータの型や戻り値などのイン

タフェース定義情報を格納する永続オブジェクト群である。クライアントからサーバオブジェクトを動的に起動する際やORB間での連携の際にインタフェース定義情報を提供する。一つのインタフェースリポジトリはRepositoryを最上位とする階層構造のオブジェクト群として定義され、それ自身のインタフェースも規定されている。

(v) ORBインタフェース：ORBの基礎的なサービス（機能）であって個々のオブジェクトアダプタやサーバオブジェクトによらない共通のサービスを規定する。例えば、オブジェクトリファレンスの文字列への変換(object_to_string)など。

(vi) IDLスケルトン（静的・動的）：スタブに対応してサーバ側でオブジェクトの特定の実装インタフェースとORB共通のインタフェースとの整合をとる。

(vii) オブジェクトアダプタ：サーバオブジェクトの生成・消滅、オペレーション起動などサーバオブジェクトの管理を行う。オブジェクトアダプタとして、基本オブジェクトアダプタBOA(Basic Object Adapter)、ファイルなどの永続オブジェクトに対するLOA(Library Object Adapter)、オブジェクト指向データベースに対するODA(Object-Oriented Database Adapter)がCORBAの仕様に例示されている。

(viii) インプリメンテーションリポジトリ：サーバ上でORBがオブジェクトの生成や位置を知るために必要な情報を格納する。CORBA仕様では、実装依存として構造などの規定はない。

(b) IDLと処理系

(i) インタフェース定義言語IDL (Interface Definition Language)

IDLは異なるプログラム言語で実装されたクライアント／サーバ間で、クライアントからサーバにあるオブジェクトのオペレーションを遠隔起動するために、インタフェースを記述する言語である。CORBAのIDLはC++に類似したシンタクスを持つが、独立した言語仕様である。図1.1-4にIDLの記述例を示す。

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

interface Employee    // インタフェース定義
  {
          attribute EmpData     personal data;
  readonly attribute Department  department_obj;
  void promote  (in char      new_obj_class); //オペレーション定義
    ・・・・・・
  void transfer (in Department   new_dep_obj);  //オペレーション定義
};                    // Employeeインタフェース定義の終り

};                    // module CORPの終り
```

図1.1-4　IDLの記述例

module文は名前のスコープを制御する。異なるmoduleのインタフェースは

module名：：インタフェース名

の形式で参照する。

(ii) 言語マッピング

IDLで記述したインタフェースから実装言語に対する整合を取るスタブとスケルトンがIDLコンパイラによって生成される。このような特定言語への対応づけを言語マッピングと呼ぶ。現在、表1.1-1に示す言語マッピングが規定あるいは検討されている。

表1.1-1 CORBA-IDL言語マッピング

マッピング言語	標準化状況	仕様／RFP (Doc. No.)
C	1991年12月CORBA 1.1採択	91-12-01
C++	1994年12月採択	Version 1.0: 94-09-14, Version 1.1: 96-01-13
Smalltalk	1994年12月採択	94-11-08
Ada	1996年3月採択	95-05-16
Java	1996年8月1日RFP発行	RFP: orbos/96-08-01
COBOL	1995年12月10日RFP発行	RFP: 96-12-10

(iii) IDLの処理

図1.1-4に例示したIDL定義はIDLコンパイラによりクライアントとサーバの実装に応じて、それぞれ、スタブとスケルトンを生成する。これらは、図1.1-3のアーキテクチャで示したように、実装言語などの違いによるインタフェース定義を詰め替えるマーシャリング(Marshaling)を行う。

(iv)インタフェースの継承

図1.1-5にインタフェースの継承の例を示す。Managerインタフェースが Employeeインタフェースからpromoteやtransferオペレーションを継承している。さらに、Managerインタフェースには異動の承認をするapprove_transferオペレーションが追加されている。

(c)オブジェクト起動の機構

(i) 静的起動と動的起動

静的起動：コンパイル／リンク時に起動先のオブジェクト名が決まる場合である。IDLをコンパイルするとクライアントとサーバのインタフェースの整合をとるスタブとスケルトンが生成される。

動的起動：起動時にファクトリオブジェクトなどによってサーバオブジェクト（インスタンス）が生成される場合には、コンパイル／リンク時に起動先のオブジェクト名を決定できない。クライアントオブジェ

```
module CORP
 ・・・・
interface Employee
 {
        attribute EmpData    personal data;
 readonly attribute Department  department_obj;
 void promote  (in char        new_obj_class);
  ・・・・・
 void transfer (in Department  new_dep_obj);
 };

interface Manager : Employee  //インタフェースの単一継承
 {
 void approve_transfer (in Employee   employee_obj,  //メソッドの追加
                        in Department current_department,
                        in Department new_department);
 };
```

図1.1-5　インタフェース継承の例

クトはインタフェースリポジトリから起動情報を得て、サーバオブジェクトに対する起動リクエストを生成する。

(ii) 起動の同期と非同期

静的起動では通常の手続き呼び出しと同様、同期オペレーション(Synchronous Operation)のみである。

CORBAでは非同期な起動を支援していないが、動的起動では、同期オペレーションに加え遅延同期オペレーション(Deferred Synchronous Operation)が可能である。この場合、起動される側のオペレーションの完了を待たずに制御を呼び出し側に戻すことが可能であるので、非同期起動を擬似できる。オペレーションが完了したかどうかを検出するために、get_responseあるいはget_next_responseオペレーションを用いる。

(3) 他の分散オブジェクト環境

(a) DCOM (Distributed Component Object Model) [ディーコムと呼ぶ]

DCOMはマイクロソフト社により開発されたActiveX/OLEを実現するための分散オブジェクト環境である。

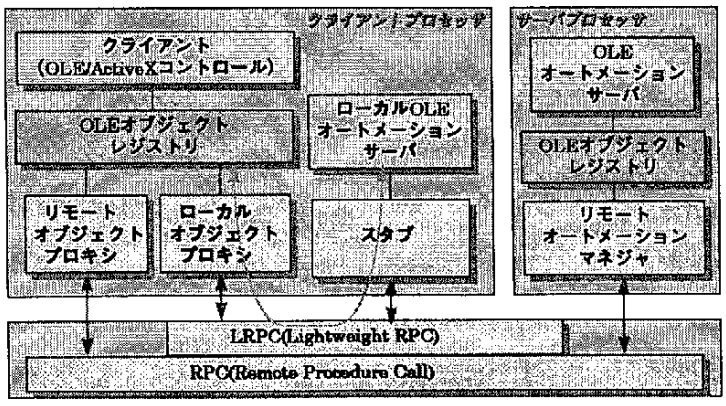


図1.1-6　DCOMのアーキテクチャ

図1.1-6に示すように、クライアント／サーバアーキテクチャをとる。CORBAのスタブとスケルトンにそれぞれ対応してプロキシとスタブと呼ぶ。Windows95/NTで実行可能である。

COM/DCOMの場合、動的起動にためのインタフェース定義はODL(Object Description Language)と呼ぶIDLによりインタフェースリポジトリに相当する型ライブラリ(Type Library)に登録(Registry)される。

(b) Java遠隔メソッド起動RMI(Remote Method Invocation)

Javaでは、ネットワーク上に分散しているJavaオブジェクトの遠隔メソッド起動(RMI)アーキテクチャが、図1.1-7に示す階層モデルで規定されている[2]。

(i) スタブ／スケルトン層：CORABA同様、起動インタフェースのマーシャリングを行う。rmicコンパイラが生成する。

(ii) リモートリファレンス層：サーバオブジェクトに対する異なる起動セマンティクスの整合を取る。たとえば、単一オブジェクトかレプリケーテッドオブジェクトか、あるいはサーバオブジェクトが常時起動しているかどうかなどをクライアントから隠蔽する。

(iii) トランスポート層：クライアント／サーバ間でのコネクションの設定と管理を行う。

Java RMIではURL形式で起動先のオブジェクトを指定できる。起動のためのオブジェクトの位置情報はサーバ上のRMIレジストリが提供する。

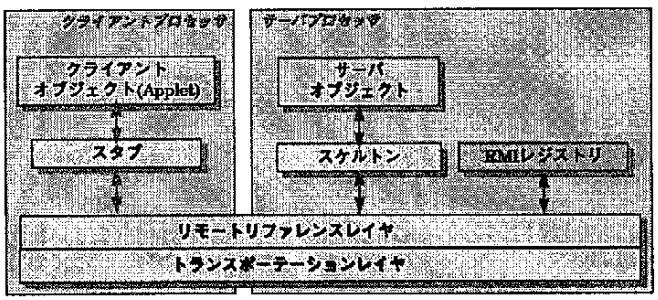


図1.1-7 Java RMIアーキテクチャ

(c) HORB

HORB(ホーブ)は電子技術総合研究所の平野氏が開発したJava上の分散オブジェクト環境である[4, 5]。HORBはCORBAにはない次のような特長がある。

(i) HORB URLと呼ぶURLと同等のオブジェクト指定が可能。

(ii) 同期に加え非同期なメソッド起動の実現。

(iii) サーバからクライアントのメソッドが起動可能。

(iv) Javaで実装されているため、プラットフォーム独立。

HORBでは、サーバオブジェクトをhorbcでコンパイルするとクライアント側のプロキシとサーバ側のスケルトンが生成できる。

(4) アーキテクチャ間インターオペラビリティ

CORBAを用いて分散オブジェクトアプリケーションを構築するためには次のような異なる分散オブジェクト環境間でのインターオペラビリティを確保する必要がある。

・異なるCORBA間のインターオペラビリティ

・CORBAとDCOM(ActiveX/OLE)間のインターオペラビリティ

・CORBAとWeb間のインターオペラビリティ

(a) ORB間連携：GIOPとIIOP

CORBA 1.2で規定されたオブジェクト間のインタオペラビリティは単一ORB内に限定されていたため、複数のサーバからなるクライアント／サーバシステムではサーバ間での連携がとれないという問題があった。1994年12月に改訂されたCORBA 2.0で、異なるCORBA間の連携プロトコルが規定された。これは、図1.1-8に示す階層構造になっている。GIOP(General Inter-ORB Protocol)がORB間連携の一般的プロトコルを規定する。特に、これをTCP/IP上で実行するためにIIOP(Internet Inter-ORB Protocol)が規定されている。すでに、多くのCORBAベンダがCORBA 2.0準拠のブローカを製品として提供している。

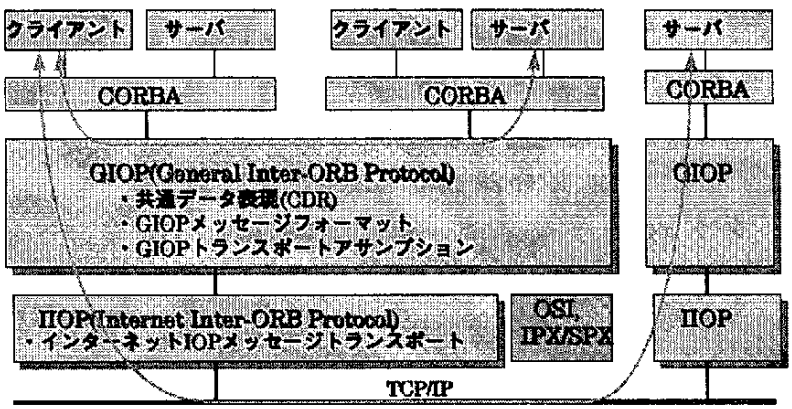


図1.1-8　CORBA間連携

(b) CORBA-COM/OLE連携

ウィンドウズ上でCOM/OLEによる多くのアプリケーションが提供されていることから、CORBAとCOM/OLE間でのオブジェクト連携が必要となる。OMGで検討されている。一方、幾つかのCORBAベンダからこの間のブリッジ機能が提供されている。

(c) CORBA-Web連携

CORBAとWebとの連携も重要なテーマである。次の3つの方法に分類されている[12]。

(i) CGIを介したCORBA-Web連携：CGIプログラムがCORBA起動リクエストを生成する。

(ii) Webサーバ上でWebサーバとCORBAを統合：Webサーバプログラムが CORBAクライアントとなる。

(iii) Webブラウザから直接CORBAオブジェクトへアクセスする。

(d) データベース連携

CORBAでは、図1.1-9に示すように、IDLを介して、ファイルやデータベースへ統一的にアクセスできる。

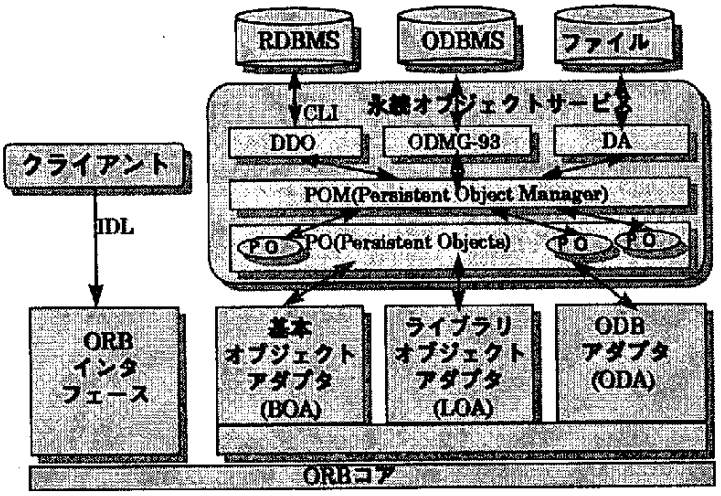


図1.1-9　CORBAデータベース連携アーキテクチャ

(4) CORBAサービス

(a) オブジェクトサービス

オブジェクトサービスはCORBAを実現するために必要なコンポーネントである。表1.1-2に現在検討中を含めたオブジェクトサービスを示す。

(b) ファシリティオブジェクト

ファシリティオブジェクトは、特定のドメインによらずアプリケーションを構築するために必要となる基本的なコンポーネントである。表1.1-3に現在検討中を含むファシリティオブジェクトを示す。

表1.1-2 CORBAサービス

RFP	サービス名称	サービス内容	仕様(Doc. No.)
1	名前(Naming)	階層的な名前付けと参照・ナビゲーション	94-01-01
1	イベント	特定のオブジェクトへのイベント分配・通知	94-01-01
1	ライフサイクル	オブジェクトの生成・削除・複写・移動	94-01-01
1	永続性	永続オブジェクトを実現する各種サービス，ODB インタフェース	94-10-07
2	関係(Relationship)	オブジェクト間の関係(所有・包含・参照・作成など)の保持・管理	94-05-06
2	並列処理	複数オブジェクト間のアクセス競合処理	94-05-08
2	トランザクション	フラット／ネステッドトランザクションの実現	94-08-04
2	エクスターナリゼーション(Externalization)	オブジェクトの情報をストリームデータとして外部と交換(複写・移動)	94-09-15
3	セキュリティ	ORB 内と ORB 間でのセキュリティ確保	orbos/96-08-03
3	時間(Time)	時刻の通知とタイミングを取るイベントの設定・解除	orbos/96-10-02
4	問い合わせ(Query)	評価機による任意のオブジェクトに対する問い合わせ(SQL,OQL など)を可能とする.	95-01-01
4	ライセンシング	オブジェクトの利用数・期間などの管理・利用情報の収集	95-03-23
4	プロパティ	クライアントからサーバオブジェクトのプロパティを動的に変更	95-06-01
5	コレクション	スタック，リストなどの基礎的なコレクションオブジェクトの標準化	orbos/96-05-05
5	トレーダ	サービスの仲介．サービスから最適なサーバ（オブジェクト）の組合せを提示	orbos/96-05-06
5	スタートアップ	ORB 起動時のオブジェクトの起動と初期設定	—

表1.1-3 ファシリティオブジェクト

RFP	ファシリティ名称	ファシリティ内容	仕様／RFP (Doc. No.)
—	システム管理	システム管理の参照モデル	仕様: 95-12-02
1	複合プレゼンテーションと交換(Compound Presentation and Interchange)	複合文書とその交換[Distributed Document Component として標準化]	仕様 1.0：95-12-20 RFP 1.1:
2	国際化と時間(Internationalization and Time)	各国文字コード，通貨，日付，時刻などの表示と操作	RFP: 95-01-31
3	データ交換と移動エージェント(Data Interchange and Mobile Agent)	オブジェクト間でのデータ交換フォーマット	RFP: 95-11-03
4	ビジネスオブジェクト(Common Business Objects and Business Object)	[サービスオブジェクトのカテゴリへ変更された]	—
5	メタオブジェクト(Meta-Object)	オブジェクトとオブジェクト間の関係を定義するメタオブジェクトの定義とその管理方法．メタオブジェクトファシリティと他のファシリティとの関係.	RFP: cf/96-05-02
6	印刷(Printing)	プリントサーバ管理(スケジューリング，スプーリングなど)	RFP: cf/96-04-03
7	アジア(マルチバイト言語)入力(Asian Input)	日本語などのマルチバイト入力の方法(J-SIG の提案).	

(c) ドメインオブジェクト

ドメインオブジェクトは特定のアプリケーションドメインにおける標準コンポーネントのインタフェースを定義する。表1.1-4に現在検討中のドメインオブジェクトを示す。

表1.1-4 ドメインオブジェクト

ドメイン	ドメイン名称	ドメインの内容	仕様／RFP (Doc. No.)
ビジネス	ビジネスオブジェクト (Common Business Objects and Business Object Facility)	ビジネスアプリケーションのための共通コンポーネント.	RFP: cf/96-01-04
	運輸(Transportation)	運輸共通と空，海上，ハイウェイ，鉄道の４ドメインのオブジェクト	RFI: trans/96-11-01
製造	高水準要求仕様(High-Level Requirements)	製造業における企業モデル.	RFI1 mfg/96-06-02
	製造(Manufacturing)	MRP(Material Requirement Planning)およびMRP-IIのための標準コンポーネント.	RFI2: mfg/96-09-02
	プロダクトデータ管理 (Product Data Management Enablers)	次の８項目が挙がっている. 1)エンジニアリング活動 2)変更作業 3)製造プロセスとプロダクト 4)ドキュメント管理 5)プロダクト構造定義 6)プロダクト有効性定義 7)構成管理 8)試験・保守・診断情報	RFP1: mfg/96-08-01
エレクトロニックコマース	電子決済(Payment)	電子通貨 (Electronic Cash)，クレジットカード，その他のマイレージプログラムなどの非通貨を含む決済のプロトコル.	RFP1: ec/96-11-02
	資産管理(Asset and Content Management)	電子資産の生成，保管，検索，配布などの基本的サービス.	RFI1: ec/96-08-03
	EC を実現する基盤技術とサービス (Enabling Technologies and Services)	電子カタログ，情報／サービスブローカ（仲介），フィルタリング，ネゴシエーションなど.	RFI2: ec/96-11-03
通信	連続メディアストリーム制御 (Control and Management of A/V Streams)	オブジェクト間での音声や画像などの連続メディアの通信制御	RFP1: telecom/96-08-01
会計	通貨(Currency)	通貨とその単位の表現ならびに計算方法.	RFP1: finance/96-09-04
	保険(Insurance)	保険業務のための標準コンポーネント.	RFI: finance/96-09-05
医療	患者情報 (Patient Identification Services)	患者情報の識別と交換.	RFP: corbamed/96-11-02
	医療共通(Common Facilities)	—	RFI: corbamed/96-01-01

(5) リアルタイムCORBA

ORBOS(ORB/Object Services)タスクフォースとリアルタイムプラットフォームTCが共同してリアルタイムシステムのためのCORBA(CORBA/RT)の検討を進めている[OMG PTC Doc. No. ORBOS/96-09-02]。ホワイトペーパが提起している検討内容は広範で、スケジューリング、リアルタイムコミュニケーションプロトコル、フォールトトレラントな実装、QoS(Quality of Service)の管理、軽量化(Lightweight CORBA)などが挙げられている。

適用ドメインとしては、通信（ネットワーク管理、インテリジェントネットワーク）、航空／防衛、製造、金融、エレクトロニックコマースなどが挙がっている。

(6) CORBAの応用

(a) 共有フレームワーク(Sharable Framework)

CORBAのドメインオブジェクトに対応するビジネスドメインの基盤となるコンポーネントを図1.1-10に示すような階層構造のフレームワーク群としてサーバ上に提供する。各階層は次のような構成となり、Javaで実装される。

(i) ベース：コレクションなどの基礎的なオブジェクトとセキュリティなどのファシリティサービス。

(ii) 共通ビジネスオブジェクト：住所、通貨などのビジネス分野における共通オブジェクト。

(iii) コアビジネスプロセス：元帳(Ledgers)、倉庫などのビジネスの中核となるオブジェクト群。

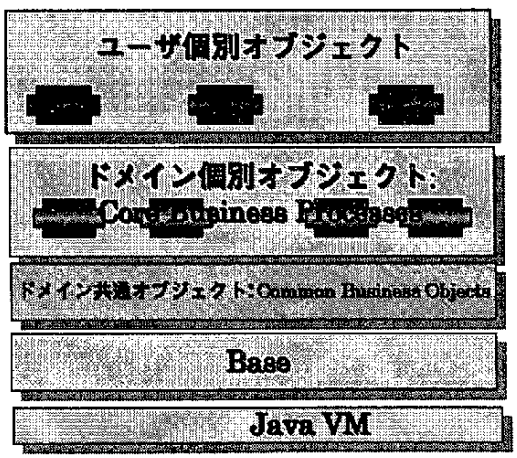


図1.1-10 共通フレームワークのアーキテクチャ

(b) ネットワーク管理ソフトウェアアーキテクチャ

ネットワーク管理は多数の類似な機器を管理することからオブジェクト指向との親和性が良く、ネットワーク管理のためのアプリケーションフレームワークが開発されている。OMGの通信ドメインのタスクフォースではCORBAをベースとするネットワーク管理アーキテクチャのホワイトペーパが提出されている[9]。

(7) CORBAの課題

多くのベンダからCORBAの実装が出そろい、CORBAの本格適用の気運が高まっている。しかし、CORBAの標準が実装の詳細を規定していないことや、性能面などでのスケーラビリティが明確になっていないことなど、実践面でのリスクがある。これは、公表されたCORBAの適用経験が少ないことも一因である。事例[3]のように、適用経験の発表が望まれる。

また、インターネットとWebの急速な発展にともない、現行のソフトウェアアーキテクチャの妥当性についても議論があるところである。OMGにおける標準化も多岐に渡っており、その進捗もテーマによって異なるなど、課題も少なくない[6]。しかし、今後、オブジェクト指向により分散アプリケーションを構築する上で、基盤となる分散オブジェクト環境は必須である[1]。その意味で、CORBAについて理解を深める必要がある。

【参考文献】

1) 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, Jan. 1996, pp. 71-79.

2) Javasoft： Java Remote Method Invocation Specification, Revision 1.2, Dec 1996, http://www.javasoft.com/.

3) 岩本 仁：ORBの使用例：情報技術コンソーシアムの場合、Computer Today、No. 77, Jan. 1997, pp. 27-36.

4) 中原真則、平野 聡：飛べオブジェクト！－HORBプログラミングマジック[前編]、bit、Vol. 28, No. 10, Oct. 1996, pp. 4-15.

5) 中原真則、平野 聡：飛べオブジェクト！－HORBプログラミングマジック[後編]、bit、Vol. 28, No. 11, Nov. 1996, pp. 53-60.

6) 大野邦夫：転機を迎えた分散オブジェクト、Computer Today、No. 77, Jan. 1997, pp. 4-14.

7) OMG： Object Management Architecture Guide Rev. 2.0, TC Document 92-11-01, OMG, 1992.

8) OMG： CORBA 2.0, TC Document ptc/96-03-04, OMG, 1996.

9) OMG： CORBA-Based Telecommunication Network Management System, Telecoms Domain Task Force White Paper, OMG, May 1996.

10) 小野沢 博文：分散オブジェクト指向技術CORBA、ソフト・リサーチ・センター、1996.

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

12) 鈴木純一：CORBA, Web, Java ORB、Computer Today、No. 77, Jan. 1997, pp. 15-26.

1．2　ActiveX

1．2．1　はじめに

ビジネスでは情報システム技術を長期的な競争力を生み出す戦略ツールとして利用するケースがますます増えている。企業は、コスト削減とワークフローの管理を改善するためだけでなく、より優れた製品とサービスを顧客に提供するために情報技術に注目するようになってきた。同時に、クライアント・サーバ処理といった、ハードウェア/ソフトウェア技術の進歩によって、情報システムがこうした新しいニーズを満足できるようになった。とはいえ、情報技術の戦略的利用は企業の情報部門に与える負荷を増大させることにもなった。これらの部門には、今やより優れたシステムをより速いペースで配備し、同時にソフトウェア技術におけるめまぐるしい進歩に遅れを取らないことが要求されている。様々な開発ツールがある程度の負荷の軽減に役立ってきたが、ほとんどの企業の情報部門では、依然として抱えているアプリケーションが増え続ける状況に直面しており、さらに毎日複雑なプログラミングとシステム展開の問題に取り組んでいる。ところが、オブジェクト技術は情報部門が新しい革新的なアプリケーションを配備する際に、本質的な利益を提供し得る潜在能力を備えている。そしてまた、こうしたアプリケーションは、企業がダイナミックで競争の激しい経済的課題に対処できるよう手助けをする。

(1)オブジェクト技術：未来は今から始まる

オブジェクト指向プログラミングはオブジェクト技術の一形態である。コンピュータ用ソフトウェアの開発では、オブジェクト指向プログラミングは従来の構造化プログラミングや設計とは異なるモデルを提供する。簡単に言えば、オブジェクト指向プログラミングとは差し替え、修正、再利用がより簡単に行える自己完結型モジュールを構築することによってソフトウェアを開発する方法である。しかし、オブジェクト指向プログラミング技術はアプリケーション開発を簡略化するというニーズに十分対処するまでには至っていない。この点、新しい形態のオブジェクト技術は企業に対してより具体的な利益を提供する。この形態のオブジェクト技術はオブジェクト対応システムソフトウェアと呼ばれ、これによって、ベンダーが供給する既製のソフトウェアコンポーネントを購入し、完全なビジネスソリューションへ統合することができる。コンポーネントソフトウェアによって、情報部門はより柔軟で、高い品質のシステムを提供でき、また、システムのプログラミング、インプリメンテーション、および保守に要する資源と時間を縮小することができる。

(2)オブジェクトの事業における利点

MicrosoftのActiveXはコンポーネントの概念にもとづいている。コンポーネントは比較的に簡単にほかのベンダのコンポーネントと互換性を保つことができる再利用可能のソフトウェアコンポーネントである。たとえば複数のベンダの各種のワードプロセッサと互換性があるような1社から販売されるスペルチェッ

カもその一つである。また多くのデータベースサーバとの会話を制御する特殊なトランザクションモニタでも構わない。これと対照的に従来のアプリケーションは広い範囲の機能が一体となってパッケージされており、そのほとんどが削除したり代替品と交換したりできない。コンポーネントソフトウェアを使えば、ソフトウェアの設計、制作、販売、使用および再利用がもっと生産的に行えるようになる。また、ソフトウェアベンダ、エンドユーザ、および会社にとって重要な意味がある。

* ベンダにとっては、コンポーネントソフトウェアは他社のアプリケーションや広範に使われているオペレーティングシステムと対話できる1個のモデルを提供する。これは基本的に書き換える必要なしに現在使っているアプリケーションに追加できるだけでなく、アプリケーションをモジュール化する機会を提供し、システム機能を適切に置き換えることができる。コンポーネントソフトウェアの出現によって、小さな、中間の、および大規模のベンダにとって一層多様な市場とニッチが作り易くなる。

* ユーザにとっては、コンポーネントソフトウェアを使えば生産性を高めながらソフトウェアの選択の幅をさらに広げられる。ユーザはコンポーネントソフトウェアの可能性を理解できるので、特殊なコンポーネントをローカルなソフトウェア販売の経路で購入しアプリケーションに付加する需要が増加することになる。

* 会社にとっては、コンポーネントソフトウェアは会社の計算のコストを下げ、情報システムの仕事の効率を上げ、会社の計算ユーザをもっと生産的にすることができる。情報システム開発者にとっては汎用ソフトウェアコンポーネントを開発する時間が減り、接続のためのコンポーネントや仕事に特有な要求を解決するコンポーネントを開発する時間が増加する。コンポーネントのアーキテクチャを有効に利用するために現在使っているアプリケーションを書き直す必要はない。そのかわりに会社の開発者は受け継いだアプリケーションを隠蔽するオブジェクト指向の「包み紙」を作ることができ、その操作やデータをネットワークの他のソフトウェアコンポーネントが使うことができるオブジェクトにすることができる。

(3) オブジェクト技術とは何か。現実のビジネス問題の解決にどのように役立つのか

オブジェクトとは、1組のデータとその関連処理情報からなる自己完結型のソフトウェアモジュールである。オブジェクト技術の重要な特長は、1つのオブジェクトがすべてのデータと処理をカプセル化し、プログラマーやユーザから内部の複雑さを見えなくしている点である。これによって、オブジェクトはいったん定義されてしまえば、簡単に利用できるようになっている。また、オブジェクトはより簡単に誤用から保護することができるが、これは唯一、メソッドと呼ばれる明確に規定されたインタフェースを経てアクセスできるようになっているためである。さらに、すべてのインプリメンテーション詳細が他のオブジェクト(他のソフトウェアモジュール)から隠されているため、同じシステム内の別のオブジェクトに影響を与えずに、簡単にそのオブジェクトの内部情報を修正することができる。その結果、オブジェクトベース

のシステムはプロシージャ・ベースのシステムに比べ、ずっとフレキシブルで簡単に維持することができる。

(4)オブジェクト対応システムソフトウェアとオブジェクト指向プログラミング

この2つの技術は基本的な概念で一致する部分があるものの、オブジェクト技術としての問題と目標はそれぞれ異なっている。実際、オブジェクト技術という用語は、その違いを考慮せず各分野に適用している場合が多いことから、正確な形で定義することは困難である。オブジェクト指向プログラミング言語および開発ツールは、ソースコードの形でオブジェクト定義を行うことによる自己完結型のカスタムアプリケーションの作成に役立つ。これらの言語ベースのオブジェクト定義は異なるアプリケーション間で共有・再利用することができる。しかしながら、オブジェクト指向プログラミング言語は、独立したアプリケーションを他のカスタムアプリケーションまたはパッケージソフトへ統合する手段を提供するわけではない。メーカが異なり、また使用されたプログラミング言語も異なる多様なオブジェクト同士を自由に連係できるようにする、という必要性を満たすわけでもない。この重要なニーズを満たすには、オブジェクト対応技術もシステムソフトウェアに組み込む必要があり、また、これらのシステムソフトウェア機能を利用するようにアプリケーションを設計しなければならない。

(5)オブジェクト対応システムソフトウェア

オブジェクト技術の革新におけるシステムソフトウェアの役割は非常に重要であるが、その問題は複雑である。オブジェクト指向プログラミングはシングルプロセスとして動作する個々のアプリケーションを構築するために使用することができるが、システムソフトウェアは多種多様なアプリケーション間の橋渡しとなる必要がある。オブジェクト対応システムソフトウェアは、オブジェクト指向プログラミングとは反対に、コンポーネントソフトウェアの作成を可能にするもので、次のような重要なニーズに対処する。

* 異なる企業によって、異なるプログラミング言語で書かれたコンポーネント同士の円滑な統合。
* アプリケーションとマシンの境界を越えた(すなわちネットワークを越えた)コンポーネント通信を容易にする総合的なオブジェクトモデル。
* 分散システムのオペレーションを破壊しない、ソフトウェアコンポーネントの自律的なエンハンスおよびアップグレード。

企業組織が自らのビジネスプロセスの最適化とリエンジニアリングを行う場合、様々な情報、アプリケーション、システムを、柔軟な、ワークフローに合わせた手法を用いて完全に統合できるようにする必要がある。オブジェクト指向システム環境なら、こうしたことが可能である。オブジェクト指向システムのような環境では、多数のコンポーネント同士が対話しながら作業を行うことによって、ほとんどすべての処理が達成される。アプリケーションからワークフォームをドラッグして、プリンタアイコンやメッセージ

送信ボックスに移動するといった単純なオペレーションでさえ、互いに通信する数十のソフトウェアコンポーネントを必要とする場合がある。互いに協調する必要のあるソフトウェアコンポーネントが数多くあると、コンポーネントオブジェクトが何であるかを形式的に定義し、また、コンポーネント同士が対話する際の規則についても定義する、優れたオブジェクトモデルをもつことが重要である。これがオブジェクト対応システムソフトウェアの役割である。その理由は、単に任意のプログラム内部にオブジェクトクラスをインプリメントする方法ではなく、自律的なソフトウェアコンポーネントが外界と対話を行う方法について定義しているからである。

適切な設計が行われれば、オブジェクト対応システムソフトウェアによって、アプリケーション間の境界を越えたコンポーネントの利用・再利用が可能になる。オブジェクト対応システムソフトウェアは、異なる会社の異なるプログラマーによって書かれたコンポーネントが、別のコンポーネントをインプリメントする方法には制約を与えずに、よく知られている一貫的な手法で動作することを保証する。もっと技術的な言葉を使えば、オブジェクト対応システムソフトウェアは、プログラミング言語に依存しないバイナリオブジェクトのインタフェースを定義している。さらにこのようなモデルは、たとえ分散システム内のコンポーネントが個々にアップグレード、または差し替えられた場合でも、コンポーネント間の接続を確実に有効にするメカニズムを定義している。

一方、オブジェクト指向プログラミング言語は、プログラミング言語(ソースコード)の標準を定義しているため、そのオブジェクトは言語およびインプリメンテーションの双方に依存することになる。オブジェクト指向プログラミング言語はオブジェクトがアップグレードされた際に、オブジェクトに依存するアプリケーションのリコンパイルと再配布が同時に行えることを想定している。分散システムでは通常、異なる会社の異なるプログラミングチームによってコンポーネントが供給されているため、こうしたことは非現実的である。しかし、オブジェクト対応システムソフトウェアがあれば、次のようなことが可能である。

* そのコンポーネントアプリケーションを設計したのが誰であるか、そのソフトウェアコンポーネントをプログラムするにあたって、どの言語や開発ツールが使われたか、といったことに関わらず、ユーザはテキスト、グラフィックス、レポート、さらにマルチメディアクリップといった、アプリケーション間の境界を越えてあらゆる情報を表現するオブジェクトを操作することができる。

* システムオブジェクト規格に基づいた標準のプログラミングインタフェースを通じて、市販のパッケージコンポーネントが相互に対話することができ、また、ビジネスラインのソリューション全体に統合することができる。コンポーネントソフトウェアはソフトウェア産業により効率的で生産的なモデルを提供し、新しいビジネスソリューションを提供するために必要なプログラミング作業と時間を劇的に減少させるであろう。

企業、システムインテグレータ並びにISVは、ますます、オブジェクト指向の開発言語とツールがプログラミング上の利益をもたらし得るという事実を見出そうとしている。しかし、堅牢なオブジェクト対応システムソフトウェアがなければ、オブジェクト技術への期待はその多くが実現されないまま残るであろう。オブジェクト指向プログラミング言語が有益なものではないと言っているわけではない。とはいえ、オブジェクト指向プログラミング言語が取り組もうとしている問題の範囲(最大限にコードを再利用することによって開発作業を軽減すること)は、オブジェクト対応システムソフトウェアが取り組んでいる問題の範囲と異なる。事実、プログラマーはシステムオブジェクトモデルに基づいたソフトウェアコンポーネントをオブジェクト指向プログラミング言語を使って構築しながら、この2つのタイプのオブジェクト技術を同時に使用することができる。

1．2．2 ActiveXとコンポーネントオブジェクトモデル

ActiveXはオブジェクト対応システムソフトウェア技術といえる。ActiveXはComponent Object Model(COM)をベースとした、基幹システムソフトウェアのオブジェクトモデルである。COMの主な機能は、公開された一貫的な手法でソフトウェアコンポーネントが動作することを保証することにある。COMは、いかなるプログラミング言語にも依存しない、オブジェクトに対するバイナリインタフェースを定義することによってこの機能を達成している。COMに準拠したオブジェクトなら、それぞれに固有のインプリメンテーションを使って作成されていても、互いに対話することができる。

COMをサポートするために作成されたオブジェクトをまとめて、コンポーネントオブジェクトと呼んでいる。ActiveXのどの機能も、基本的なオブジェクト間通信の機能を提供するためにCOMに依存している。言い換えれば、COMはActiveXの「オブジェクトバスとオブジェクトサービス」機能を提供する。例えば、ActiveXを使ってあるベンダーから供給されたスプレッドシートオブジェクトを別のベンダーのアプリケーションによって作成されたワープロ文書にシームレスに埋め込むことができる。スプレッドシートとワープロソフトはお互いのインプリメンテーションについて全く知る必要はなく、ActiveXが規定するインタフェースを使って接続する方法だけを知っていればよい。

ActiveXはCOMの上位に構築されたオブジェクト機能のセットである。これらの機能の多くは複合ドキュメントやインターネットに関連しているが、ActiveX自体は、単なる複合ドキュメントやインターネットのアーキテクチャではない。ActiveXは、アプリケーションがスタンドアロンで実行されていても、またはネットワークに分散されていても、他のビジネスアプリケーション、並びにパッケージソフトウェアと簡単に統合することができるカスタムビジネスアプリケーションを作成するための信頼性の高いプラットフォームを提供する。

(1)コンポーネントオブジェクトモデルは標準の通信方法を提供

オブジェクト間の通信に対するバイナリ標準や通信のインタフェースの標準がなければ、プログラマは異なるタイプのアプリケーションと通信するための専用の数多くの手続きを書く、あるいは対話する必要のある他のコンポーネントに依存してアプリケーションを再コンパイルするというううんざりする仕事に直面する。さらにオブジェクトの対話に使うメカニズムが特別に効率的でない限り、サイズと性能の要求からそのメカニズムを絶対に使おうとはしないであろう。プログラマーにシステムや他のアプリケーションと対話するために特定のプログラミング言語を使うことを強制できないので、結局オブジェクト通信は言語と独立でなくてはならない。

Component Object Model はこれらの課題に応じる。ActiveXではアプリケーションはインタフェースと呼ぶ多くの関数呼び出しやメソッドを使い互いに対話し、またシステムと対話する。インタフェースは、ソフトウェアコンポーネントの間の意味的に関連がある操作の集合である。 すべてのCOM オブジェクトは、共用するインタフェースを見つけるためにコンポーネントの間の非常に有効な折衝を行う QueryInterface と呼ぶメソッドをサポートする。インタフェースで定義した機能とCOMのインタフェースである折衝プロトコルによって、ソフトウェアコンポーネントはコンポーネントの必要に応じて単純または複雑な形式で対話できる。またオブジェクトシステム内の変更と大きな変更も許する。新しいインタフェースは、コンポーネント間の現在の連携を乱すことなく安全に導入できる。

最下位レベルではCOMオブジェクトの相互作用は非常に高速で単純である。ソフトウェアコンポーネント間の結合を確立すると、COMオブジェクトにおけるメソッドを実行するのは2個のメモリポインタを使う単純な間接関数呼び出しである。結果として同じアドレス空間でCOMオブジェクトと対話する性能上のオーバヘッドは無視できる。このようにCOMオブジェクトを使うときには、性能の低いパソコン上でさえもなんの障害もない。

COMのモデルは単純でありプログラミング言語に対して独立である。C、C++、Pascal、Ada、Smalltalk、Microsoft Visual Basic プログラミング言語のような、ポインタの構造体を作りポインタによって暗示的または明示的に関数呼び出しができるプログラミング言語ならなんでも、 COMオブジェクトを作れるとともにそれを使うことができる。 オブジェクト指向言語は言語オブジェクトとCOMオブジェクトの間に高レベルのマッピングを提供し、COMプログラミングを容易にするためにクラスライブラリも備えることもできる。

Microsoft Windowsオペレーティングシステムは、新しいサービスを定義する領域でCOMインタフェースを広く使うことを計画している。視覚的な制御、マルチメディアサービスおよび分散サービスなどの多様な機能は、Component Object Model によって定義しプログラムすることを目標にしている。現在の

Win32アプリケーションプログラムインタフェースは必要で十分なサポートを継続する。同時にオブジェクト指向の機能は徐々に普及し、Windows オペレーティングシステム自身の中ではアプリケーション、システム、簡単に交換できるコンポーネントの提供の間の区別は曖昧になるであろう。長期戦略としては、コンポーネントオブジェクトは、開発とコンポーネントソフトウェアの使用を促進するように構成する。ActiveXはコンポーネントオブジェクトの発展の第一歩である。ネットワーク上でオブジェクトが通信できる機能も含めて、広い範囲の追加機能を提供する。

(2) ActiveXの機能

COM自身の他にも、数多くの補助的なActiveX機能がある。ActiveXの主な機能の一部を以下に紹介する。

オブジェクトのリンクと埋め込み（OLE）機能を使うと、ビジュアル編集、アプリケーション間のドラッグとドロップ、オートメーション、およびオブジェクトの構造化記憶などの機能を始めとするアプリケーションコンポーネントを統合するさまざまなメソッドを使うことができる。

* コンポーネントオブジェクトモデル。COMはすべてのインタフェース標準を供給し、ソフトウェアコンポーネントの統合を可能にするすべてのコンポーネント間通信を管理する。COMはバイナリ標準であるため、ソフトウェアコンポーネントはどの言語でも記述でき、また、どのソフトウェアベンダからも供給することができる。しかも、単一アプリケーションにシームレスに統合することができる。

* OLEオートメーション。オートメーションによって、アプリケーションはアプリケーションの内部、または複数のアプリケーションをまたがって動作するコマンドセットを提供できる。例えば、ユーザは文書作成プログラムからコマンドを呼び出して、別のアプリケーションで作成されたスプレッドシート内のある範囲のセルをソートすることができる。

* ActiveXコントロール。ActiveXコントロールはアプリケーションの機能拡張・強化を行うために購入できるCOM対応ソフトウェアコンポーネントである。ActiveXコントロールはカスタム、または市販のActiveX対応アプリケーションで利用できる。Microsoft Visual Basicなどの開発環境の多くが、高品質の既製ソフトウェアコンポーネントを使ったビジネスアプリケーションの構築に向けた効率的な手段としてActiveXコントロールをサポートする。

* OLEドラッグ・アンド・ドロップ。ユーザはあるアプリケーションウィンドウから別のアプリケーションウィンドウへオブジェクトをドラッグしたり、別のオブジェクトの中にオブジェクトをドロップすることができる。

* コンポーネント管理。ActiveXは分散システム内のソフトウェアコンポーネントを、コンポーネントベースのソリューションに対して影響を与えずに、自律的に更新できるようにしている。

* OLEドキュメント。OLEドキュメントはCOM対応アプリケーションで作成されたデータを組み込むこと

ができる複合ドキュメントの一形態である。例えば、COM対応のワープロソフトは、COM対応のスプレッドシートから表やチャートを受け付けることができる。OLEドキュメントは、ビジネス文書に多様な情報を組み込むことによって、ユーザが自分自身のアイディアをより効率的に伝達することを可能にする。チャートや表といった静的な情報を組み込む以外にも、音声やビデオ画像、アニメーションといった動的な情報をも組み込むことができる。また、OLEドキュメントによって複合ドキュメントの作成プロセスが向上することから、ユーザはより生産的な作業が行えるようになる。以下にOLEドキュメント固有の機能を説明する。

* OLEオブジェクトのリンクと埋め込み。オブジェクトのリンク機能によってアプリケーションを他のアプリケーション内部にあるデータオブジェクトにリンクすることができる。例えば、スプレッドシートの表を複数のカスタムビジネスレポートにリンクさせることができ、元のスプレッドシート・アプリケーションの中でこの表に対する変更が行われると、すべてのレポート文書が自動的に更新される。一方、オブジェクトの埋め込みは、オブジェクトが持つデータソースへのリンクを維持せずに、別のドキュメント内部にそのオブジェクトを組み込む機能である。オブジェクトのリンクとオブジェクトの埋め込みでは共に、オブジェクトを供給するアプリケーションがOLEサーバと呼ばれ、オブジェクトを含むアプリケーションがOLEコンテナと呼ばれる。アプリケーションはOLEコンテナ、OLEサーバのどちらにもなることができる。

* OLEビジュアル編集。ビジュアル編集によって、ユーザはテキスト、グラフィックス、音声、ビデオ画像、その他の多様なオブジェクトタイプを組み込みながら、内容豊かな複合ドキュメントを簡単に作成できる。アプリケーションを切り換えながら複合ドキュメントの部品を作る代わりに、ユーザはあくまでそのドキュメント上で作業を続けることができる。別のアプリケーションで作られたスプレッドシートやグラフィックといったオブジェクトの編集を開始すると、オブジェクトが埋め込まれているアプリケーションのメニューやツールは、本来そのオブジェクトを作成したアプリケーション(サーバ)で使用されているメニューやツールに自動的に変更される。こうしてユーザは、別のアプリケーションをアクティブにしたり、切り換えたりせずに、そのドキュメントのコンテキストの中でオブジェクトを編集することができる。

* ネストされたオブジェクトのサポート。オブジェクトは他のオブジェクトの内部で複数のレイヤーにネストすることができる。ユーザは他のオブジェクトの内部にネストされたオブジェクトを直接操作することができ、また、ネストされたオブジェクトへのリンクを確立することができる。

* オブジェクト変換。同じオブジェクトを使って異なるアプリケーションを利用できるよう、オブジェクトの型を変換することができる。例えば、あるスプレッドシートで作成されたオブジェクトは、編集用の別のスプレッドシートアプリケーションが解釈できるように型を変換することができる。

* 最適化されたオブジェクト保存。オブジェクトは必要になるまでディスク上に置かれており、コンテナ

アプリケーションがオープンされる度にメモリにロードされることはない。また、ActiveXには完全処理されたオブジェクトの保存機能があり、オブジェクトのディスクへのコミットとロールバックをサポートしている。この機能によって、ファイルシステムにオブジェクトが保存された際の、データの完全性が確実に保持される。

* 記憶装置に依存しないリンク。ディスク上のファイルとして保存されていなくても、埋め込みオブジェクト同士のリンクを保持することができる。この機能によって、同一ドキュメント、または異なるドキュメントの内部に埋め込まれたオブジェクトは、ファイルシステムによって認識されるかどうかに関わらず、互いのデータを更新することができる。

(3) ActiveXを特徴づけるオートメーション機能

技術的な視点からは、OLEオートメーション機能とActiveXコントロールの提供がActiveXの大きなアドバンテージである。

OLEオートメーションはカスタマーにメリットを提供するために、仮想的に他のアプリケーションを活用する方法で、他のアプリケーションのオブジェクトへのアクセスがユーザアプリケーションを起動することなく可能になる。このオブジェクトユーザは「オートメーションユーザ」呼ばれ、一般にはVisual Basicようなプログラミングツール、もしくはアプリケーションのマクロプログラミング言語機能が相当する。これらのツールの言語を利用することでプログラマ、特に企業内で開発を行っているプログラマは、特定のタスクを処理するために複数アプリケーションをドライブするためのスクリプトを書くことができる。

このようにアプリケーション間を横断するプログラミングスタイルは、今までは存在しなかったがユーザの要求の1つには含まれていたはずで、ユーザが選択したマシン上で複数のオートメーションオブジェクトを提供し、単一のオートメーションクライアントから集中的に処理を実行するセントラルマクロプログラミングのスタイルをとるようになる。

Microsoftは最近「Office Object」という言葉を使い、いわゆるスイート製品「MS Office」をオートメーションプログラミングという視点から見て、すぐに利用できるオブジェクトの集合体として汎用コンポーネントの提供を進めている。実際、リアルタイムで金融データなどを扱うために定義される「WOSA/XRT」は、オートメーションが技術的な意味で中心的な存在となっている。

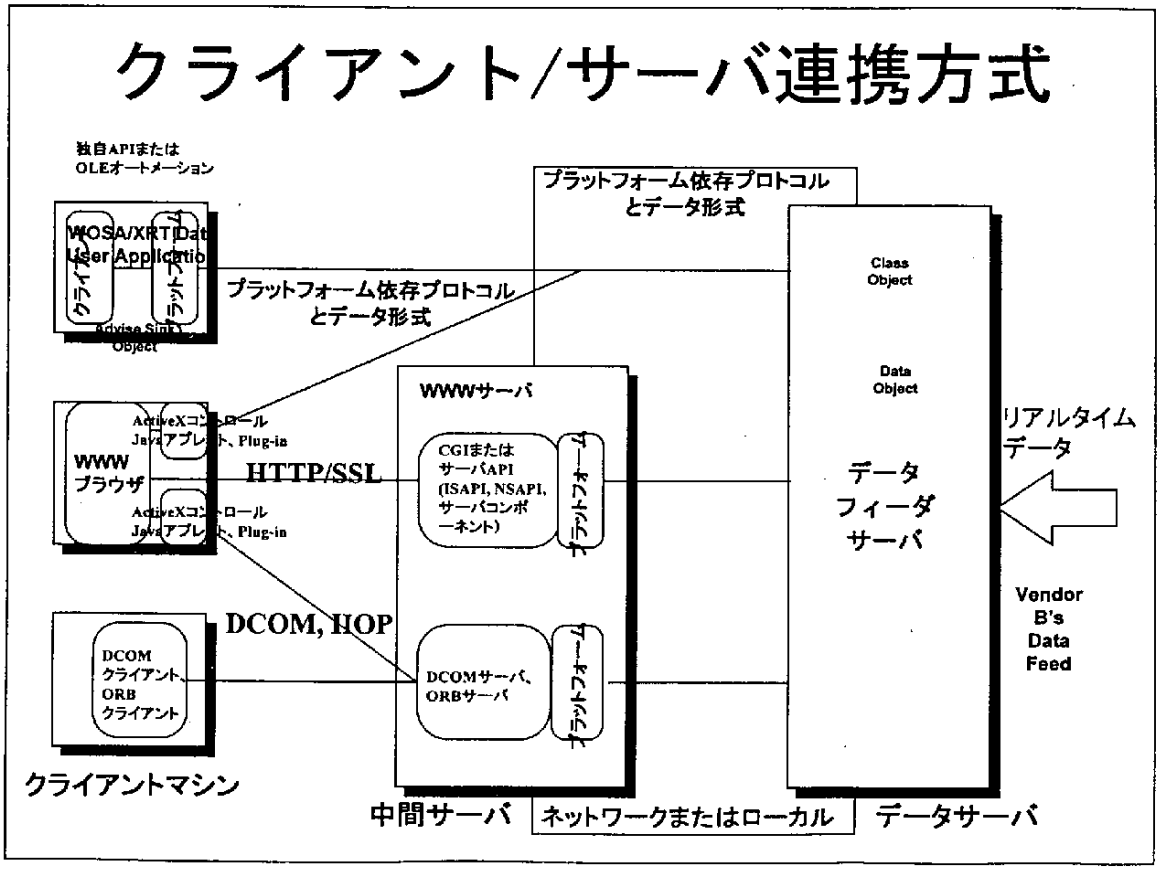


図1.2-1　OLEオートメーション。WOSA/XRTを用いた金融データリアルタイムシステムへの応用

その他のOLEオートメーションの例として、カスタム会計ソフトがOLEオートメーション機能を使ってパッケージソフトのスプレッドシートをアクティブにし、スプレッドシートのセルに対する入力およびフォーマットを行い、さらにスプレッドシートのチャート作成エンジンを利用して四半期レポート用のグラフを作成するというものが考えられる。オートメーション機能によって、配布用の四半期レポートの印刷が行われる前に、グラフはそのレポートに自動的に埋め込まれる。

この例が示すように、企業開発者とシステムインテグレータは、パッケージ化されたコンポーネントソフトウェアを構成部品として使用しながら、OLEオートメーション機能を利用してカスタマイズされたより大規模なビジネスソリューションを迅速に組み立てることができる。OLEオートメーション機能によって、開発者は生産性向上アプリケーション、パッケージ化された垂直市場向けアプリケーション、およびOLE対応アプリケーション/開発ツールといったあらゆるコンポーネントソフトウェアに、カスタムソフトウェアを簡単に統合することができる。一言で言えば、OLEは種類の異なるソフトウェア同士の壁を壊し、ビジネスソリューションを達成するために他のアプリケーションと通信できるようなパッケージ/カ

スタムアプリケーションの作成を可能にする機能である。したがって、システムインテグレータが行う作業の中心は、オートメーションプログラミングで、Visual BasicやVisual Basic for Applications(VBA)またはその互換言語を使ったソフトウェア開発に集約される。Visual Basicプログラミングは、C++などの言語で開発されたコンポーネントを組み立ててアプリケーションを開発する、コンポーネントを束ねるための接着剤としての役割を持っている。このプログラミングスタイルにおけるメリットは、プログラムがシンプルで、高度な処理を簡単に記述できる点である。以下で、実際にオートメーションを利用したプログラム例を見てみる。

```
Command1_Click()
If Combo1.Like "業務組織変更について" Then
Set myDoc = myobj.Workbooks.Open("c:¥demo¥form¥ringi.ff2")
myobject.Visible = True
End If
End Sub
```

図1.2-2 Visual Basicで記述されたプログラム

この例は、コマンドボタンがマウスでクリックされた場合のイベント処理を記述するものである。その中で、コンボボックスの選択内容の条件に従って、スプレッドシートのワークブックにアクセスし、それを画面表示している。

(4) ActiveXコントロール

ActiveXコントロールアーキテクチャは、既存のOLEオートメーションインタフェースを拡張する形で、現在のCOM対応のアプリケーションでActiveXコントロールが動作できるようにする。例えば、ユーザがあるデータベース開発環境に別のActiveXコントロールを挿入して、そのデータベースのもつ機能の範囲を広げることができる。こうした機能としては、財務専門のモジュール、数式編集、化学分析、ランタイム・チュートリアル、メッセージ、あるいはその他のタイプの機能があろう。同一のActiveXコントロールなら、例えば第四世代プログラミング言語といった他の開発ツールとの互換があろう。ユーザはさらに、スプレッドシートやワープロソフトといった生産性向上アプリケーションを含むCOM対応アプリケーションに、直接このコントロールを組み込むこともできるであろう。ActiveXコントロールは、再利用可能なソフトウェアコンポーネントを、カスタマイズされたソフトウェアやパッケージソフトに組み込むための、

速くて、シンプルで、かつ効率のよい標準的方法である。一方、OLEオートメーションは、パッケージソフトを完全なカスタムビジネスソリューションに統合するための方法である。

ActiveXコントロールが提供するGUIコンポーネントとして、ボタンやグラフなどや、データベースとリンクしたグリッドなどが考えられる。これらを例にとり、実際にActiveXコントロールをコンテナに配置してみる。

まず、コンテナアプリケーションを起動し、オブジェクトを配置するための「台紙」（フォーム）を新規に作成する。つぎに、利用可能なActiveXコントロールの一覧メニュー（ツールボタンの場合もある）から配置すべきActiveXコントロールを選び、配置位置をマウスでクリックする。配置されたActiveXコントロールは、フォーム上に自由にレイアウトの変更が可能である。

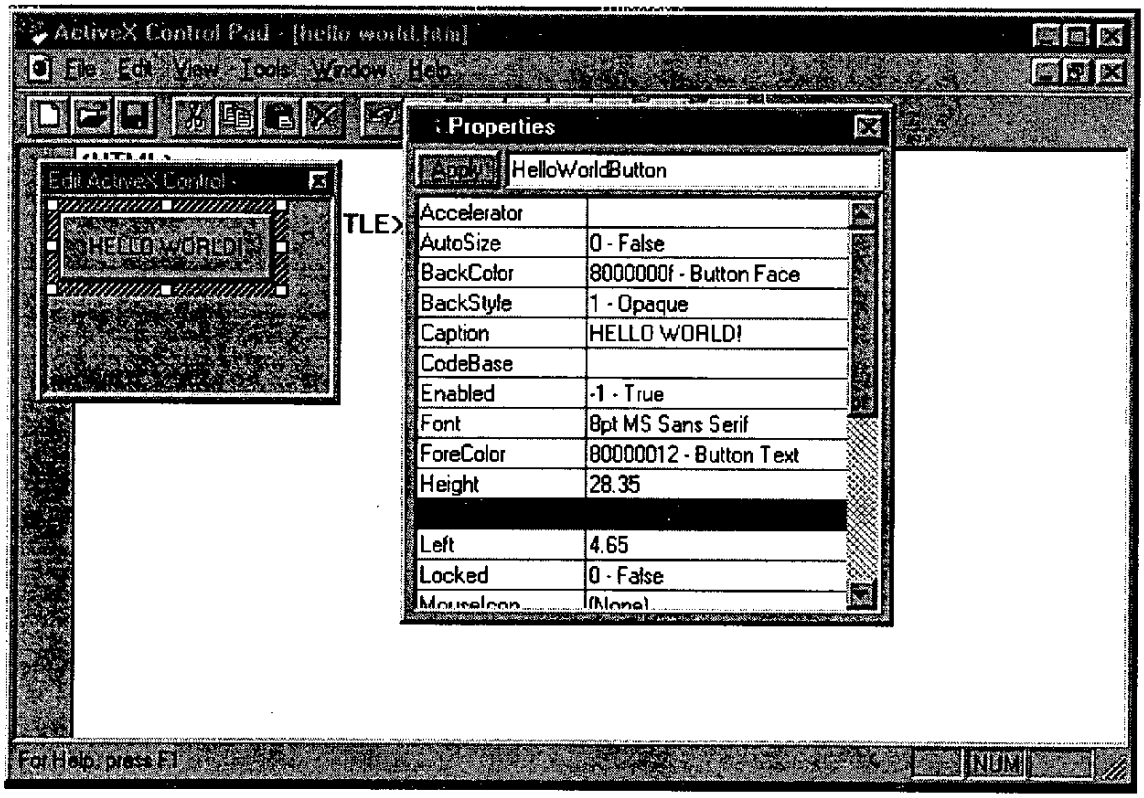
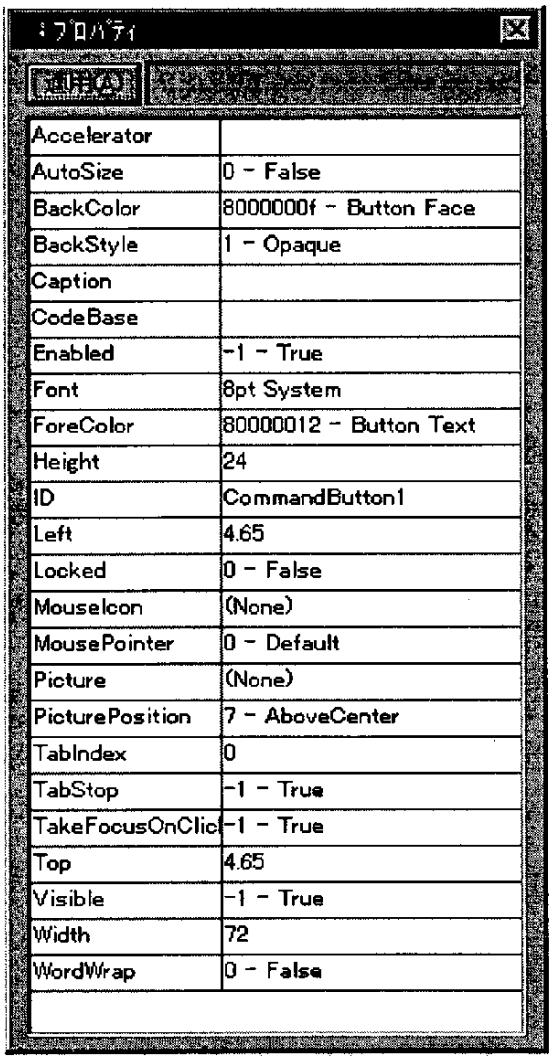


図1.2-3　コンテナアプリケーションにActiveXコントロールを配置する

ところで、表現上「ActiveXコントロールをフォームに配置する」といっているが、正確には「ActiveXコントロールが生成したオブジェクトを複合ドキュメントファイルに書き込んでいる」ということである。さらに具体的に「ActiveXコントロールが生成したオブジェクト」とは、ボタンActiveXコントロールを例

にすると、「ボタンの画像データと、それに与えられた指示に対して実行される機能、その他の設定値」ということになる。複合ドキュメントファイルとは、オブジェクトは構造的に記憶するファイルシステムである。また、この複合ドキュメントファイルをオープンした際、そこに書き込まれたオブジェクトはコンテナのフォームの起動と同時に生成されて、メモリ上に展開される。

ActiveXコントロールのさまざまな設定値を「プロパティ」という。プロパティの設定によってActiveXコントロールが生成するオブジェクトの仕様を変えることができ、設定値を引き数として処理を行うこともできる。例えば、ボタンActiveXコントロールでは図1.2-4のようなプロパティが設定できる。



Accelerator	
AutoSize	0 - False
BackColor	8000000f - Button Face
BackStyle	1 - Opaque
Caption	
CodeBase	
Enabled	-1 - True
Font	8pt System
ForeColor	80000012 - Button Text
Height	24
ID	CommandButton1
Left	4.65
Locked	0 - False
MouseIcon	(None)
MousePointer	0 - Default
Picture	(None)
PicturePosition	7 - AboveCenter
TabIndex	0
TabStop	-1 - True
TakeFocusOnClic	-1 - True
Top	4.65
Visible	-1 - True
Width	72
WordWrap	0 - False

図1.2-4 ボタンActiveXコントロールのプロパティ

プロパティは既定値として設定できるほか、イベントに記述してオートメーションの実行中に設定することもできる。プロパティや次に説明するメソッドなどActiveXコントロールに関する情報は、「タ

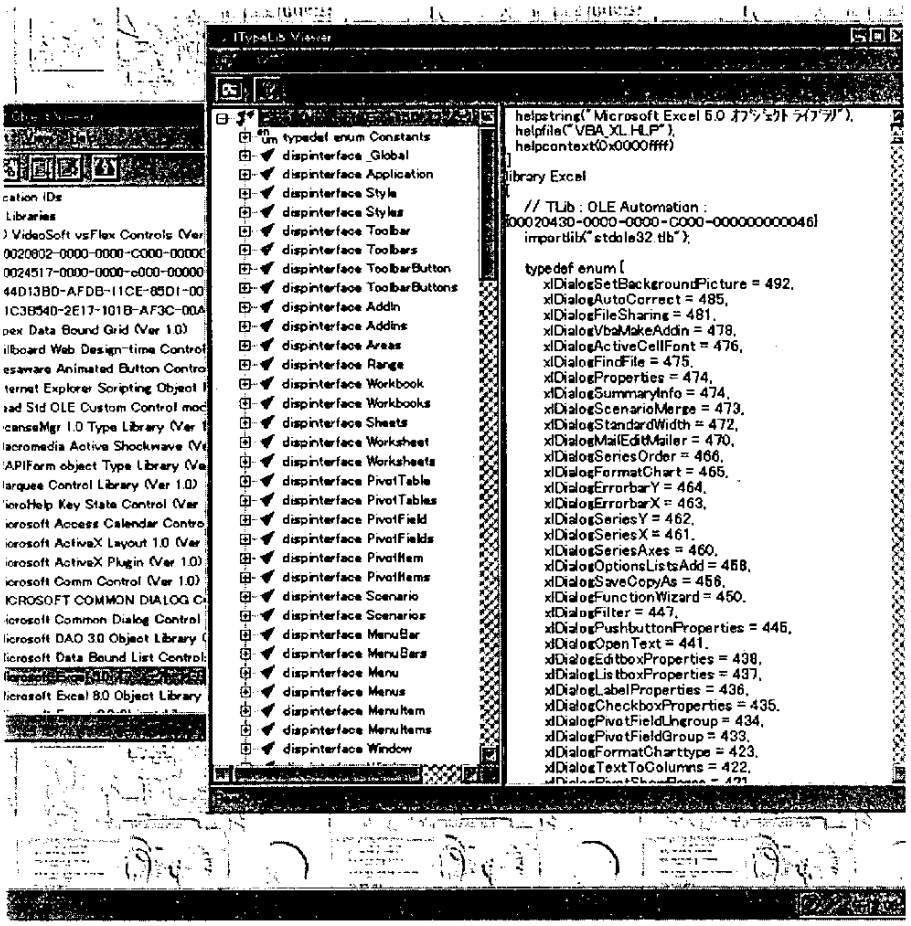


図1.2-5　オブジェクト・ブラウザでタイプライブラリを確認

イプライブラリファイル」に記述されて管理される。タイプライブラリファイルは、サーバ実行ファイルと別ファイルとして存在する場合と、ActiveXコントロールのファイルに内包されている場合がある。

その他、ActiveXコントロールが実行するファンクション、メソッドと、オブジェクトに与えられた指示（イベント）に対して実行するイベント処理がある。メソッドの呼び出し、イベント処理の内容の記述はVisual Basicなどの言語でプログラムする。

1．2．3　分散オブジェクトシステム

COMの主要な特長は、ネットワーク上でコンポーネント間通信を容易にする機能を持つ点である。現在、分散オブジェクトをサポートするCOMは、Windows NT 4.0、Windows 95およびSolarisで提供されている。現在販売されているCOM対応アプリケーションはすべて修正を行うことなく、この分散オブジェクト機能を利用することができる。この機能は、標準ではMicrosoft Remote Procedure Call(RPC)メカニズムを

使用することによって、異なるマシン上で動作するCOM対応アプリケーションのシームレスな相互運用を可能にする。Microsoft RPCは、OpenGroupの分散コンピューティング環境RPC(DCE RPC)と互換性があり、OpenVMS、MVS、AS/400および20以上の種類のUNIXを含む広い範囲のDCEベースのシステムと情報を交換できる。また、オブジェクトを定義するInterface Definition Language（IDL）もDCEと上方互換である。

分散COMの性能を説明するため、先のWOSA/XRTの例、株価をリアルタイムで表示するグラフにリンクされたスプレッドシートを作成する場合を想定してみる。サーバで動作するCOM対応の株価オブジェクトを使って、ユーザは単にスプレッドシートのメニューから「挿入」を選択し、利用可能なオブジェクトのリストから株価オブジェクトを選び、そのオブジェクトをデスクトップ上で動作しているスプレッドシートにリンクさせる。分散COMによって、ユーザはリアルタイムの株価グラフをスプレッドシートの形で受け取るが、株価コンポーネント自身は、生のデータ供給と関連するデータ処理と共に、ネットワーク上のサーバで実行される。加えて、株価オブジェクトはCOMのリンクをサポートする標準COMオブジェクトであることから、追加的なプログラミングを行うことなく既存のCOM対応コンテナアプリケーションに実データを供給することができる。したがって、ワープロ文書や金融取引カスタムアプリケーションの中に簡単に配置することができる。さらに例を挙げてみると、株価オブジェクトがツールバーのようなActiveXコントロールを含む場合がある。このツールバーによってユーザはリアルタイムにグラフ上に表示された株式の銘柄を変更できるようになる。このように個人の利用にあわせたオブジェクトの再利用やカスタマイズを行う場合、コーディングを追加する必要はない。信頼性の高いシステムソフトウェアオブジェクトモデルがもつ真の性能とは、ソースコードを共有するプログラマの領域を越えてオブジェクトの利用範囲を広げている点である。これによってユーザおよび開発者はプログラミングをまったく行わずに、アプリケーションソフトウェアコンポーネントの操作やカスタマイズ、再利用を行うことができる。

分散COMによって、企業開発者は一つのアプリケーションを、それぞれ他のコンピュータ上でも透過的に実行できるような複数のコンポーネントモジュールに分割できるようになる。COMはネットワークに透過性を与えることから、先の例で説明したとおり、ユーザや開発者にとってはこれらのコンポーネントが単一のマシン上にあるように見えるであろう。各コンポーネントを、必要とする処理性能やディスク容量に適応したコンピュータで動作することができる。事実、アイドル状態になっている利用可能な処理能力に応じて、コンポーネントをネットワーク上にダイナミックに配置し、個々のコンポーネントを特定のコンピュータに分散するような設計が可能となる。

ネットワークとオブジェクト指向という異なる役割とツールを持った2つのテクノロジーが融合するとき、予測できないほどのメリットが生まれてくる。ネットワーク上に分散されるオブジェクト間でコミニュ

ケーションが行われ、データの変更や修正が自動的にすべてのオブジェクトに反映されるような環境が実現される。グループウェアなどの概念が実用性をもって具体化されるためには、分散オブジェクトのサポートが前提となるであろう。クライアント／サーバシステムの構築方法も、アプリケーションサーバやActiveXコントロールで構成されるGUIをもったコンテナ環境などをベースとして再検討されることになるであろう。そのために、ソフトウェアのライセンス形態に関する見直しも必要となるであろう。

より近い将来、特定の業務や分野に対応したコンポーネントビルダによるコンソシアムの設立が予測され、このコンソシアムでは、コンポーネント開発に関する規約を作成することになるであろう。複数のベンダーから提供される各種のコンポーネントを効率よく利用するための環境作りが現時点での重要なテーマだからである。ベンダー間で共有できるソフトウェアレイヤと、製品の差別化のために必要となるレイヤを明確にし、コンポーネントビルダ間での競争をより高いレベルで行うことがソフトウェア産業全体の活性化をもたらす。このような考え方が、現在の日本のベンダに最も不足している部分ではないであろうか。国内のソフトウェアベンダがそれぞれの分野においてコンペティタと話し合いの場を持つことで、ソフトウェア部品化への第一歩がはじまる。

(1) 分散COMの動作

分散COM機能のComponent Object Modelの自然な拡張機能である。その最も重要な違いは、操作とデータをオブジェクト間で転送するために使う遠隔手続き呼び出し機能である。COMは「軽量」遠隔手続呼び出し機能（LRPC）を単一コンピュータ間のオブジェクト間通信に使う。LRPCでは、オブジェクトがプロセス境界をまたいで情報を渡し、オペレーティングシステム内のアプリケーションを互いに保護する。言い替えると、LRPCは同じマシン上でプロセス（オブジェクトなど）が別のプロセスと対話するプロセス間通信機能である。分散COMでは、LRPCではなくMicrosoft RPCを使ってネットワークをまたいでオブジェクト通信を行う拡張をした。RPCでは、呼び出したアプリケーションと同じアドレス空間に手続きが存在するかのように、アプリケーションが遠隔手続きを呼ぶことができる。しかし、実際はその手続きは同じマシンの別のプロセスまたはネットワークをまたいだ異なるマシンに存在するかもしらない。呼び出すアプリケーションと対応する手続きの間のデータ転送は透過的に行うので、非分散アプリケーションが使うプログラミングモデルを変更せずに複数のプロセスまたはコンピュータにまたがって動作するアプリケーションを構築できる。RPC自身は、ネットワークプロトコル独立、セキュリティと名前サービス、および互換性のないプロセッサとオペレーティングシステムアーキテクチャの間のデータ変換のサポートなど、分散オブジェクトシステムが必要とする多くの重要な機能を提供する。が、RPCは十分ではない。RPC上のCOMオブジェクト層は、オブジェクト特有の多様な機能を提供するとともに、分散処理の一層の抽象化と単純化を提供する。COMオブジェクト層は、LRPCアーキテクチャ上でオブジェクトのロー

カルインスタンスと非ローカルインスタンス（呼び出しプログラムのアドレス空間外のオブジェクト）の間の透過性をすでに提供している。この透過性を達成することは分散オブジェクトシステムを構築する場合に最も困難な点である。十分なRPC機構（および分散セキュリティのような関連する機能）を追加するには多くの課題があるが、分散COMはCOM全体のアーキテクチャを変更しない。

以下では、クライアントアプリケーションは、異なるマシン上で動作するオブジェクトサーバに結合する。クライアントマシン上で動作するRPCプロキシは、クライアントが同じマシン上であるかのようにオブジェクトサーバと通信させる。クライアントマシン上で動作するネットワークサービスを使い、プロキシは単に関数呼び出しをネットワーク上で転送できる標準RPCメッセージに変換する。サーバマシン上の対応するRPCスタブはクライアントから 標準RPCメッセージを受け取り、それをオブジェクトサーバに渡す。サーバにとってはプロセスはローカルクライアントと通信するのと同じである。ここで、「クライアント」および「サーバ」という言葉は単にオブジェクトサービスでのユーザと提供者を各々指すことに注意する必要がある。クライアントがデスクトップマシンでサーバが大型バックエンドマシンであるというわけではない。実際、COMではアプリケーション間のほとんどの対話は、両者のソフトウェアコンポーネントが互いに同時にクライアントでありサーバでもある、双方向関係を持っている。

(2) 分散オブジェクトはコンピューティングを再定義する

先に述べたように、分散COMは単一のアプリケーションを多数の異なるコンポーネントオブジェクトに分割し、それぞれを異なるコンピュータで動作させることが可能である。ネットワーク全体は巨大な処理能力と容量を持つ1個の大きなコンピュータのように見える。たとえば、データベースアプリケーションは検索エンジン、リポートエンジン、書式ビルダ、トランザクションマネージャなどのコンポーネントの集合として構築できる。これらの各コンポーネントは必要な処理能力、I/Oバンド幅、およびディスク容量に適したマシン上で動作できる。結果として、ソフトウェアが必要とするハードウェア能力と一致することができるので計算はさらに効率的になる。

(3) 他のオブジェクトモデル仕様について

これまで様々なベンダによって異なるオブジェクトモデルが提供されてきたために、オブジェクト対応システムソフトウェア市場は混乱する可能性がある。どんなシステムソフトウェアオブジェクト技術も、それがクロスプラットフォームを実現し相互運用が可能であると主張している場合でさえも、幾つかの大きな疑問点について言及しておかなければならない。これらの疑問点には次のものがある。

＊そのオブジェクトモデルは、利用可能な多くのアプリケーションで採用されている、実績のある技術なのか。あるいは、単なる紙の上での仕様にすぎないのか。

＊そのオブジェクトモデルは、ソフトウェアコンポーネントが同一マシン上のアプリケーション間でシー

ムレスに動作するというニーズに対処しているのか。さらに、コンポーネントは、まったく同じモデルを使ってネットワーク上のマシン間でシームレスに動作できるのか。

* そのオブジェクトモデルは信頼性のある接続メカニズムを提供しているのか。言い換えれば、オブジェクトシステムが脆弱なために、コンポーネントが接続したり、失敗するようなケースがあるのか。

* そのオブジェクトモデルは、オープンでクロスプラットフォームな相互運用性のニーズに対処しているのか。

* そのオブジェクトモデルは、分散アプリケーションに対してセキュリティが確保された環境をサポートしているのか。

* そのオブジェクトモデルによって、アプリケーションコンポーネントとOS機能として提供されているコンポーネントとのシームレスな統合が可能になるのか。すなわち、OS機能の拡張に適したオブジェクトモデルは、そのオペレーティングシステムとアプリケーションとをシームレスに統合するのか。

これらの疑問に対して、分散オブジェクトシステムを構築するための分散COMの技術の要点を以下にまとめる。

* 強力な型。分散オブジェクトシステムは元来無数の互いに識別すべきインタフェースとソフトウェアコンポーネントを持っている。モジュール、オブジェクト、クラス、またはメソッドを発見しバインドするために、可読的な名前を使うシステムはどれも危険な状態にある。複雑なシステムでは可読的な名前同士の衝突の可能性は極めて高いのである。名前による識別の結果、互いに対話するように設計していない複数のソフトウェアコンポーネントが必ず偶然に接続されてしまうことになる。その結果、コンポーネントやシステムにはバグがなく、設計したとおりに動作したとしてもエラーやクラッシュが発生する。これと対照的に、COMは広域的な固有の識別子を使い、すべてのインタフェース、型、およびクラスを識別する。128ビットの整数は、実際上空間と時間を超えて世界中で固有であることが保証できる。可読的な名前は、便利なように割り当てられているだけでローカルな適用範囲しかない。このためにActiveX、COMコンポーネントは無数のオブジェクトを持つネットワーク中でさえ、偶然にオブジェクト、インタフェースまたはメソッドに接続しないことを保証している。

* インプリメント継承の有無。1個のコンポーネントが別のコンポーネントからその機能の一部を「サブクラス」としたりまたは継承するインプリメント継承は、アプリケーションを作成するときに非常に便利な技術である。しかし、分散オブジェクトシステムではこれが問題を生むと多くのエキスパートが結論を出している。この問題は、学術文献で「壊れやすいベースクラス問題」と呼んでいる。インプリメント継承の問題は、インプリメント継承におけるコンポーネントの「契約」または関係がインプリメント階層のなかで明確に定義されておらず、暗黙で不明瞭なことである。親または子コンポーネントがその挙動を急

に変更したとき、関連するコンポーネントの挙動は未定義になるかもしらない。インプリメント階層が、すべてのコンポーネントを同時に修正できるようなプログラマの管理のもとであればこれは問題ではない。しかし、真の分散オブジェクトシステムの複雑なアプリケーションで、関連するコンポーネントの集合を同時に管理、変更することは困難である。それゆえ、インプリメント継承はアプリケーションを構築するためには非常に便利なのであるが、システムオブジェクトモデルでは危険が伴う。COMは「集約」と呼ぶコード再利用機構を持つ。このモデルを使うと、オブジェクトの集合は正しく定義された方法で共に動作し、ほかのソフトウェアコンポーネントには単一のオブジェクトに見える。集約機能はコードの再利用という便宜を与え、同時にすべてのオブジェクトの間の明示的な関係を保持しインプリメント継承の危険を除いてくれる。

*単一プログラムモデル。インプリメント継承に関する問題は、プロセス内とプロセス外またはネットワークをまたぐオブジェクトにとってプログラミングモデルの1問題である。たとえば、インプリメント継承は単一のアドレス空間の外では一般的に動作しない。別の言葉で言うと、プログラマはリモートオブジェクトをサブクラスにはできない。同じように、単一アドレス空間のなかで他のオブジェクトによって自由に操作できるクラスの公開データへのアクセスのような機能は、プロセスやネットワークの境界を越えて動作しない。COMは単一のインタフェースを基礎とするバインドモデルを持ち、ローカルとリモートプログラムモデルの間のどんな違いも避けるように設計されている。

*セキュリティ。現実世界の分散オブジェクトシステムにとって、カプセル化したオブジェクトとデータにセキュリティを確保してアクセスする方法を提供しなくてはならない。分散COMはセキュリティ機能を持つように設計されている。オブジェクトサーバはセキュアなオブジェクト起動をするように設定できるが、COMクライアントは無修正のままセキュアな分散環境に利用することができる。

1．2．4 インターネット／イントラネットコンテンツ制作とアプリケーション開発の新手法ActiveXとは

この数年間、文書検索にWorld Wide Webを使う人が急激に増加し、インターネットの成長は驚くほど大きなものとなっている。この一方で、強力なパワーを持ったデスクトップPCやサーバPCの低価格化が進み、個人生活やビジネス活動での生産性に革命がもたらされている。ActiveXは、World Wide Webとパーソナルコンピュータの能力を組み合わせ、Web利用者相互間のインタラクティブなアプリケーション配布や、より豊かなコミュニケーションを可能にする。ActiveXはまた、ソフトウェア配布、顧客サポート、人事、財務といった分野で使っているアプリケーションをインターネットにのせたいと考えている多くの企業の要望に応える。この環境を実現するために必要なものは、ネットワーキング機能を備え、相互連携が可能なツールやコンポーネントの豊富なセットをアプリケーション開発者に提供でき、インターネットリソー

スとシームレスに通信できるプラットフォームである。ActiveXは、インターネットやイントラネットのインタラクティブなアプリケーションやコンテンツを制作する簡単な手段を、開発者やWebページ制作者に提供する統合プラットフォームである。ActiveXは既存のデスクトップテクノロジの他、Javaのような新しいWebベースのテクノロジも包含しており、パーソナルコンピュータとWebの最良な組み合わせを実現したものである。この組み合わせがもたらす具体的なメリットをいくつか紹介する。

* 内容豊かなインタラクティブコンテンツやアプリケーション
* 高品質なツールと使用可能なテクノロジ
* 標準をベースとした互換性

ActiveX は、ソフトウェア、ハードウェア、ネットワーキングテクノロジ、開発ツール、言語などにおける幅広い選択肢を提供している。ActiveX テクノロジは HTTP、TCP/IP、COMなどの標準をサポートしている。

(1) インターネット/イントラネットアプリケーションのための統合プラットフォーム

ActiveX は、クライアント側プラットフォーム、サーバ側プラットフォーム、開発ツール、オーサリングツール、ネットワーキングテクノロジから構成される。

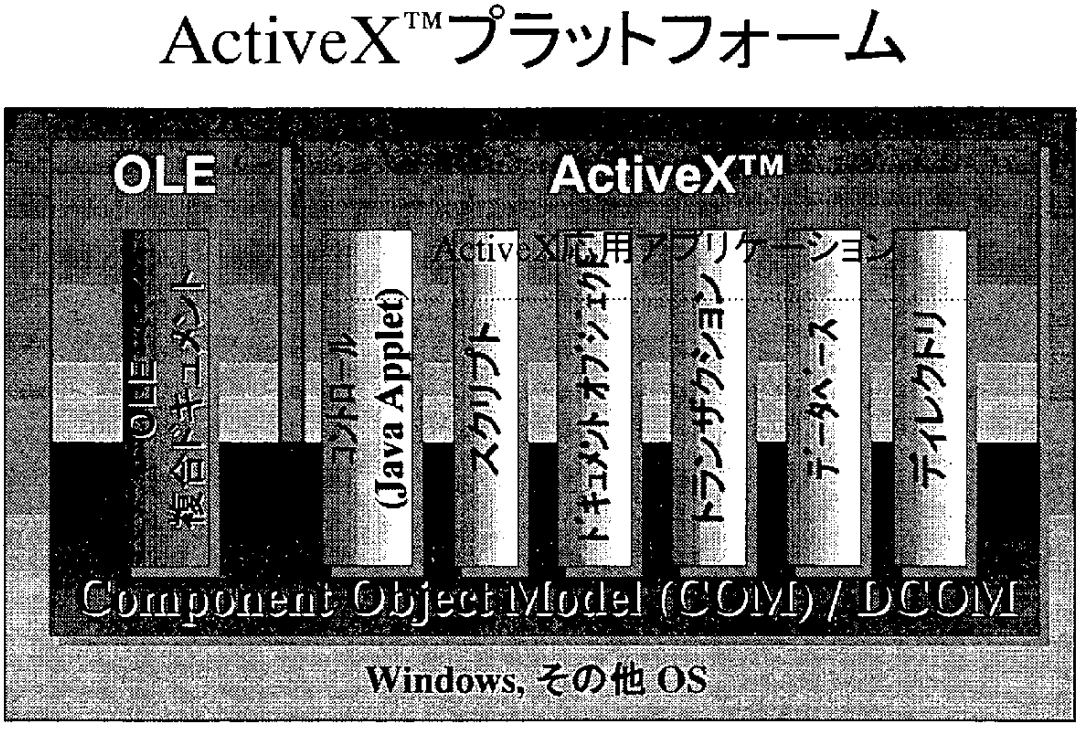


図1.2-6　ActiveXはインターネット/イントラネットアプリケーションのためのオープンプラットフォーム

(2) ActiveXクライアント側プラットフォーム

ActiveXクライアント側プラットフォームは、アプリケーションとコンテンツをエンドユーザシステムで実行、表示するクライアント側プラットフォームで、Microsoft インターネットエクスプローラーがその中心である。JavaアプレットおよびActiveX コントロールを実行、Visual Basic ScriptやJavaScriptスクリプトを実行、ActiveVRML (3Dバーチャルリアリティモデリング)、ActiveMovie (オーディオ/ビデオ)などの異なる種類のコンテンツを表示、編集するActiveX コントロール群の提供がある。

Javaアプレット、ActiveX コントロール、およびそれに付随するVisual Basic Script、JavaScriptなどのスクリプト言語は、Webサーバから自動的にダウンロードされ、クライアント上で表示、実行される。これらは、主に以下のように利用される。

* ActiveXのコントロールは、Webページ内のインタラクティブなオブジェクトであり、ユーザにマウスクリックやキー入力などの対話的な機能を提供する。このためWebサイトでの体験を生き生きしたものとする。

* ActiveXドキュメントは、Microsoft ExcelファイルまたはWordファイルといったHTMLではないドキュメントをWebブラウザで見ることができる。

* ActiveXスクリプトは、ブラウザまたはサーバから、複数のActiveXコントロールやJavaアプレットの動作を統合的に制御する。

* Microsoft Java仮想マシンは、Javaアプレットを実行し、JavaアプレットとActiveXコントロールの統合を可能とする。 ここで注意する点は、一般の認識とは異なり、ActiveXとJavaは競合しないということである。ActiveXは、Javaを含めてすべてのプログラミング言語を拡張する標準的なメカニズムを提供する。ActiveXにより、Java開発者はJavaアプレットをActiveXのプラットフォームと統合でき、Java言語の能力が拡張される。 ActiveXはJavaアプレットを他の言語で作成されたオブジェクトと結びつける。これによって、Javaプログラマは、JavaプログラムからActiveXコントロールに直接リンクできる。同様に、他のプログラミング言語で作成されたCOMオブジェクトをJavaアプレットにリンクできる。ActiveXはこれらのオブジェクトを結びつける接着剤であり、統合化されたプラットフォームを提供する。Javaを含めたプログラミング言語を拡張し、リンクする標準プラットフォームを提供することにより、ActiveXはインタラクティブなWebの開発のための開発者のリソースを最大限に活用する。

(3) ActiveXサーバ側プラットフォーム

ActiveXサーバ側プラットフォームは、クライアントにインターネットサービス、アプリケーション、データ、コンテンツを提供するサーバ側プラットフォームである。アプリケーションやコンテンツは、Webサーバを実行しているサーバ上や別のシステム上に置くことができる。データはSQLデータベース、

またはOpen Database Connectivity (ODBC)をサポートするデータベースであればサーバから接続できる。サーバ側のプラットフォームの提供する中心はMicrosoft Internet Information ServerというWebサーバである。これは、Common Gateway Interface (CGI)アプリケーション、ISAPIダイナミックリンクライブラリ(DLL)、ActiveXサーバスクリプトの機能を提供する。さらに、スクリプトから起動されるOLEオートメーションサーバにより、データベース、メッセージストアなどのバックエンドサーバに接続する3階層型のクライアント／サーバシステムをサポートする。これらは、ActiveXサーバフレームワークとも呼ばれている。

(4) ActiveX開発/オーサリングツール

ActiveX開発ツールには、Microsoft Visual J++といった新しいJava開発ツールもあれば、Microsoft Visual C++、Visual Basic、Borland Delphi、Oracle Power Objectといった既存の製品もある。このような開発環境の柔軟性は、インターネットやイントラネットのアプリケーションの作成にあたって、開発者に幅広い開発ツールおよび言語の選択肢を与えている。インターネットやイントラネット用のコンテンツの作成は、ActiveXオーサリングツールMicrosoft FrontPageなどが利用され、マウスの「ポイントアンドクリック」でActiveXコントロールを配置したWebページが作成できる。

(5) ActiveXオブジェクトモデル

ActiveXオブジェクトモデルの代表的な例は、クライアントプラットフォームである、インターネットエクスプローラーで見ることが出来る。インターネットエクスプローラーのオブジェクトは、MSHTMLオブジェクトとIExplorer Browserオブジェクトという２つのオブジェクトから構成される。両方ともActiveXコントロールとして実装されている。MSHTMLはHTMLビュワー機能を受け持ち、パーサと表示に関する大半のコードを実装する。ActiveXコントロールに加えて、MSHTMLはOLEドキュメントオブジェクト（サーバ）としての機能ももっている。また、オートメーション・インタフェースも提供するが、スタンドアローンのオートメーションサーバとしての利用を前提としていない。もう1つのコンポーネントであるIExplorer Browserは、OLEドキュメントオブジェクトのコンテナで、ハイパーリンクインタフェース機能を提供する。したがって、インターネットエクスプローラーは、IExplorer Browserオブジェクトが MSHTMLオブジェクトを内部に取り込んで、ブラウズ機能を提供するようなコンポーネント間の連携でアプリケーションが構成されている。

1．2．5　ActiveXとMicrosoft Windowsの進化

ActiveX技術は Microsoft のオペレーティングシステムの主要な基礎コンポーネントを提供する。Microsoft Windows NTとWindows 95はActiveXを基礎として情報の作成、アクセス、操作、組み立て、共有を容易に行えるようにしている。ユーザがアプリケーションを意識的に切り替えたり、ネットワークを意

識せずに、内容や特性の質問によって情報を操作する進化したオブジェクト指向環境を提供する。

(1) システムサービスのオブジェクト化

COMは個々のアプリケーションコンポーネントが相互に動作したり通信したりすることを可能にするだけでなく、Windowsのシステムサービスに対してシステムオブジェクトと呼ばれるオブジェクトベースのインタフェースを提供する。現在、こうしたサービスにはメモリ割り当て、ファイル管理、およびデータ転送等がある。この意味で、COMはWindowsをオブジェクト指向オペレーティングシステムに進化させるという戦略的基盤をなしている。オブジェクトサービスは、マルチメディア機能、データアクセス機能、トランザクション機能、ディレクトリ機能、分散セキュリティ、分散ファイルシステムといった機能が含まれ、徐々に普及し、オペレーティングシステム内部を手軽に交換できるコンポーネントを供給していくであろう。その際、既存のソフトウェアが引き続き動作する保証をすることが重要で、COMインタフェースとバージョン管理機能がそれを担う。

【参考文献】

本節の内容に関連した情報の詳細は、Microsoft Webサイトで入手可能である。

1) ActiveX の詳細: http://www.microsoft.com/intdev

2) Java情報: http://www.microsoft.com/devonly

3) Visual Basic Script: http://www.microsoft.com/vbscript

4) Internet Control Pack : http://www.microsoft.com/icp

開発者用の他の資料は、以下の Microsoft ソースで入手できる。

1) Microsoft Developers Network (MSDN): 開発者情報の CD による購読サービス

2) Strategic Whitepapers

3) The Microsoft Object Technology Strategy(098-55163)MIS、ISV、システムコンサルタント、管理者向け

4) OLE Corporate Backgrounder(098-56457) ユーザ、MIS、ISV、システムコンサルタント向け

5) OLE and the Benefits of Component Software(098-56459) ユーザ、MIS、ISV、システムコンサルタント向け

6) OLE Documents(098-56352) ユーザ、MIS、ISV、システムコンサルタント向け

7) OLE Controls(098-55315) MIS、ISV、システムコンサルタント向け

8) Open Systems:Technology Leadership and Collaboration(098-55058) MIS、ISV、システムコンサルタント向け

9) OLE and OpenDoc:Information for Customers(098-56353) MIS、ISV、システムコンサルタント向け

10) Object Strategies:How They Compare(098-55636) MIS、ISV、システムコンサルタント向け

11) OLE, SOM and OpenDoc:A Comparison of Technologies(098-55722) MIS、ISV、システムコンサルタント向け

12) OLE Documents Technical Backgrounder(098-56453) 開発者向け

13) Microsoft OLE:Today and Tomorrow(098-56454) 開発者向け

14) What Is an OLE 2 Application? (098-56455) 開発者向け

15) Developing Applications with OLE 2 (098-56456) 開発者向け

16) OLE Control Specification Overview (098-56458) 開発者向け

17) The Microsoft Foundation Classes (MFC) Whitepaper 開発者向け

18) The OLE 2.0 Programmer's Reference (ISBN 1-55615-628-6 and -629-4) 開発者向け

19) Inside OLE 2.0 (ISBN 1-55615-618-9) 開発者向け

ActiveXオブジェクトモデルを説明するドキュメントとしては、以下のものがある。

1) Component Description Relevant Specifications The ActiveX Object Model Overview

2) The ActiveX Object Model Overview The HTML Scripting Object Model

3) The HTML Scripting Object Model IExplorer Browser

1．3 Java Beans

Java Beans とは、GUIビルダで視覚的にカスタマイズしたり組み立てることのできるソフトウェア部品のことである。Java Beans は、プラットフォームに依存しないJava API を使って実現されるため、ポータブルなソフトウェア部品を作成することができる。 ここでは、Java Beans の概要と特徴を説明し、典型的な利用例を紹介する。

1．3．1 Java Beans の概要

(1) コンポーネントウェア

コンポーネント化技術によってソフトウェア部品を利用できるようになると、これまであったさまざまな問題を解決することができる。一つは、ソフトウェア部品の再利用が進むことによって、ソフトウェア開発 の生産性を向上させることができるようになることである。また、部品自体に価値を持たせられ、ソフトウェア部品の規格を定めれば部品を自由に選択できるようになる。 さらに、これまで小数のソフトベンダーがパッケージソフトのマーケットをコントロールしていたためサードベンダーが参入するのは難しかったが、コンポーネント化技術が普及すれば独自の部品を投入するチャンスが増えるといわれている。

(2) Java Beans の設計目標

Java Beans API は次のような設計目標のもとに設計された。

(a) ポータビリティ

「一度書いたら、いつでもどこでも動く」という性質の実現。

(b) 統一的な API

異なるコンポーネントに対して、違いを吸収するような統一的な API を提供すること。

(c) アプリケーションビルダのサポート

どのアプリケーションビルダからでも Beans を使ってアプリケーションを組み立てられること。

(d) 広い守備範囲

小さく軽快な GUI 部品のために Beans を利用することができる一方で、大きなフレームのために利用することも可能にする。

(3) 既存のコンポーネントとの関係

Java Beans は、既存のコンポーネントウェアである OpenDoc、LiveConnect、ActiveX のそれぞれに対して、コンポーネントを実現するためのインタフェースとして利用することができる。Java Beans は、処理のオーバーヘッドが少なく、ポータブルな部品を作ることができるため、既存のコンポーネントシステ

ムの部品として使われることに適している。

(a) ActiveX/OLE/COM

* Java Beans を OCX の部品として用いる。

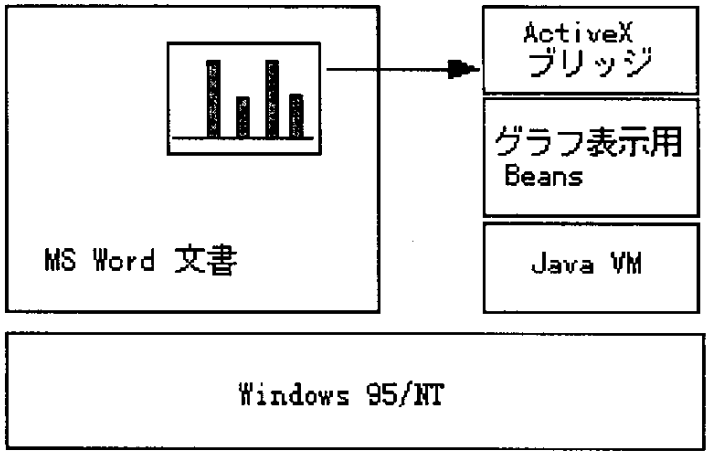


図1.3-1　Java Beans を OCX の部品として用いる例

(b) OpenDoc

* Java Beans を OpenDoc パーツとして用いる。

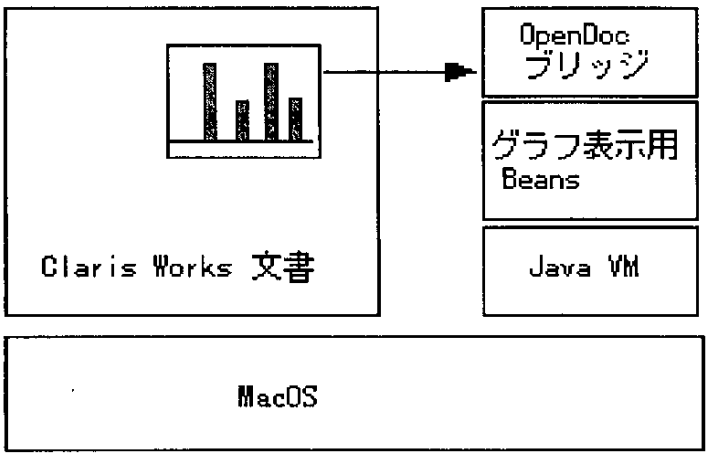


図1.3-2　Java Beans を OpenDoc パーツとして用いる例

(c) LiveConnect

* Java Beans を LiveConnect のモジュールとして用いる。

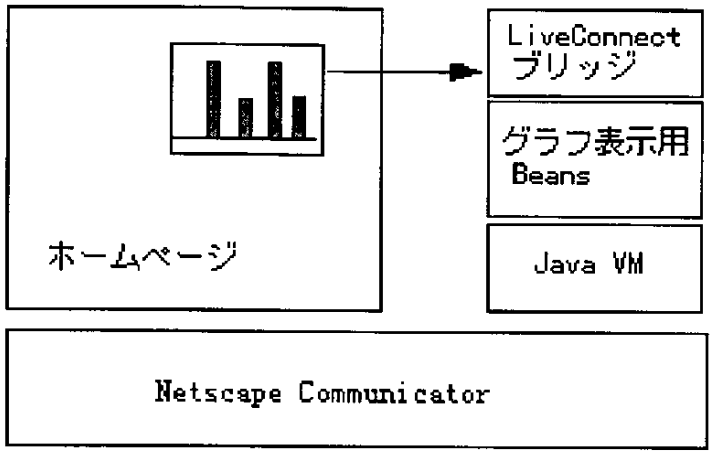


図1.3-3 Java Beans を LiveConnect のモジュールとして用いる例

また、ベースとなるコンポーネントシステムと Java Beans の融合が実現すれば、Java Beans はコンポーネント・ソフトウェアの差異を吸収する標準的なインタフェースとしての役割をはたすようになるであろう。

* Java Beans を OCX 部品のコンテナとして使う。

* Java Beans を OpenDoc パーツのコンテナとして使う。

* Java Beans を LiveConnect のコンテナとして使う。

(4) リモート・オブジェクトとの連携

基本的には Java Beans は、同じ仮想マシン上で組み立てられる。したがって、部品はすべて同じマシン上にあり、しかも同じアドレス空間に配置される。

Java RMI (Remote Method Invocation)や Java IDL(Interface DefinitionLanguage)を用いることによって、Java Beans からリモート・オブジェクトを操作することが可能になる。

(5) 開発プロセス

Java Beans (以後 Beans) はソフトウェア・コンポーネントであるので、Beans の開発者と、Beans を利用する開発者は別々でもよい。また、Beans を利用して別のBeans を作ることも可能である。

アプリケーション開発者は次のような手順で、Beans からアプリケーションを組み立てる。

(a) Beans を配置する

アプリケーション開発者は、まず Beans をアプリケーションのコンテナに配置する。左側のウィンドウには、利用可能な Beans の名前とアイコンが表示されている。

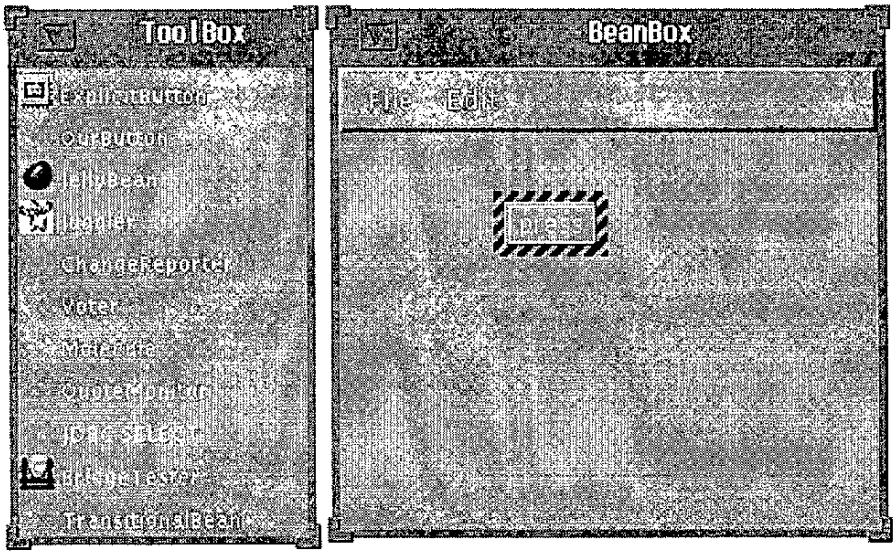


図1.3-4 Beans の配置例

(b) Beans をカスタマイズする

一度配置した Beans に対してカスタマイズを行うことで、その部品の見かけや振る舞いを部分的に修正することができる。

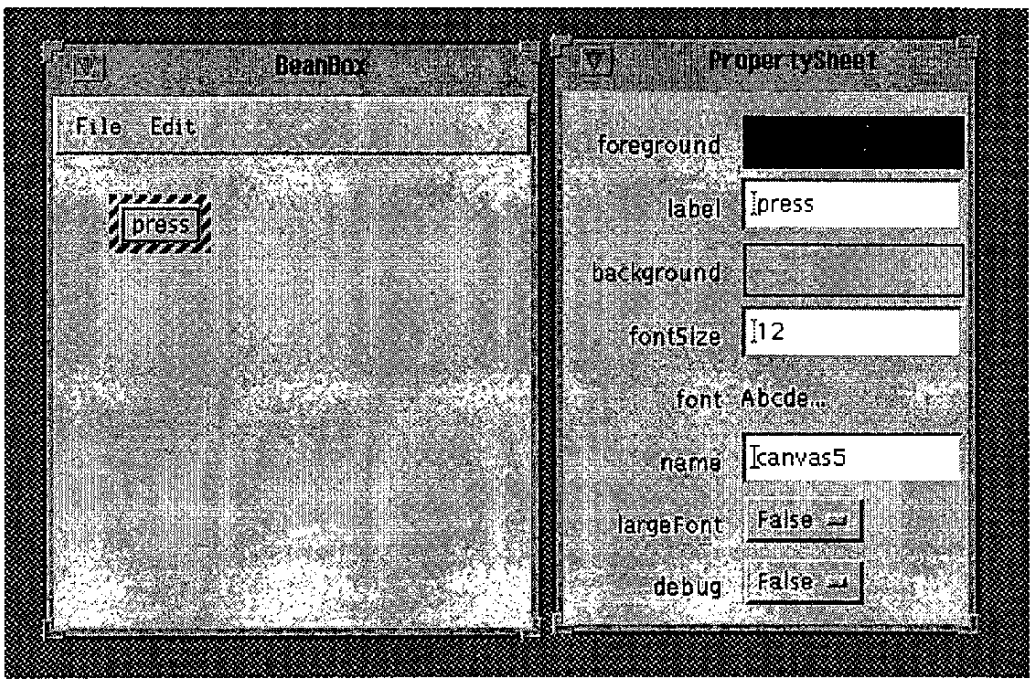


図1.3-5 Beans のカスタマイズ例

(c) イベントをメソッドに結びつける

部品から発生するイベントをどの部品のどのルーチンに結びつけるかを視覚的に指定することができる。図1.3-6は、メニューからイベントを選ぶ場面である。イベントを選んだ後で Beans を選択すると、実行時にその Beans にイベントが送られることになる。

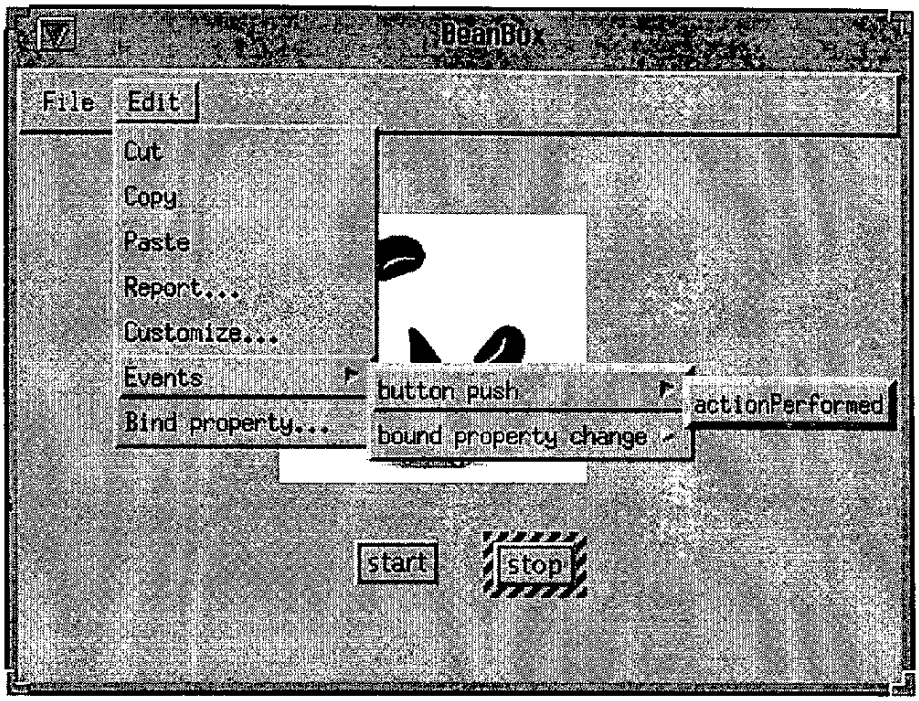


図1.3-6 イベントとメソッドの結付け例

(6) 設計時と実行時

Java Beans は、部品としての使われ方を設計時と実行時で区別できるようになっている。設計時とは、完成した Java Beans を集めてアプリケーションを組み立てる段階であり、実行時とは、エンドユーザがアプリケーションを利用する段階である。

一般に、実行時よりも設計時に多くの情報を提供する必要がある。設計時には、アプリケーション開発者が Beans を部品として視覚的に扱うことでアプリケーションの組み立てが楽になることが期待される。Beans の開発者は柔軟なオプションを用意しておき、アプリケーション開発者は要求を満たすように部品の振る舞いを決める。例えば、ウィザード方式（次々に選択肢を用意することによって情報を引き出す方法）で Beans のカスタマイズを行う方法を提供するとすれば、アプリケーション開発者は選択肢を選ぶことによってその部品を望みどおりに実行させるために必要なコードを生成させることができる。

(7) 何を Beans にするか

Java Beansは、ソフトウェア部品を視覚的に扱って部品の組み立て作業に役立てることを目指すものである。したがって、このような必要性のある部品についてのみJava Beansの形で実現すればよく、必ずしもすべての部品がJava Beansになるわけではない。

(8)部品の視覚性

Java Beansは通常目にみえる視覚的な部品が対象になる。例えば、ボタンという部品も目に見えるBeansになる。

しかし、目に見えない部品もJava Beansになり得る。例えば、タイマーは目に見えない部品に相当する。このような部品も、メソッドを呼び出したり、イベントを送出したり、状態を永続的な形に保存することができる。また、実行時に目に見えない部品も、アプリケーション開発者がその部品をカスタマイズする際にプロパティを視覚的に編集することが可能である。

1．3．2　Java Beansの基礎

Java Beansは、「プロパティ、メソッド、イベント」の三つで部品を特徴づける。

(1)プロパティ

プロパティとは、外からアクセスできるようになっている名前のついた属性である。設計時にはプロパティを視覚的に設定することができる。プログラムからプロパティにアクセスするには、関連するメソッドを利用する。

プロパティはその振る舞い方によって次のように分類される。

単純なプロパティ

一つのインスタンス変数が表すプロパティを指す。読み出し用と書き込み用のメソッドを持つことができる。

インデックス付きプロパティ

配列型の値を持つプロパティを指す。インデックスを指定して配列の要素を取り出すメソッドを用意する。

結合されたプロパティ

値が変更されたときに、他のオブジェクトに必ず通知するようなプロパティを指す。

制約付きプロパティ

値を変更しようとしたときに、その要求を拒否することのできるプロパティを指す。

(2)メソッド

メソッドは、外から呼び出すことのできるJavaのメソッドであり、通常はpublicのJavaのメソッドは

すべて Beans のメソッドとなる。一部のメソッドだけをBeans のメソッドにすることもできる。

(3) イベント

イベントとは、ある部品から、その他の部品に影響を与えることのある出来事が起こったことを表すための仕組みである。JDK 1.1 の AWT のイベントモデルは JDK1.0.2 のものと異なるが、Java Beans のイベントはJDK 1.1 の AWT のイベントモデルをもとにしている。

1．3．3 Java Beans を実現するための仕組み

(1) Reflection API

JDK1.1 の Reflection API を使うと、プログラムの中でクラス定義を調べることができる。例えば、クラス定義から、あるシグネチャを持つメソッドを見つけ、それをオブジェクトとして表し、そのメソッドをダイナミックに起動することができる。あるいは、クラス定義からある名前のプロパティを操作するメソッドを見つけ、それを起動することもできる。

(2) イントロスペクション

アプリケーションビルダに部品を登録したときに、その部品のプロパティ、メソッド、イベントをアプリケーションビルダが調べるための機能をイントロスペクション(introspection)と呼ぶ。

部品の特徴を部品の利用者に見せるためには、二通りの方法がある。一つは、BeanInfo オブジェクトや関連するオブジェクトにプロパティやメソッドやイベントの情報を予め登録しておく方法である。もう一つは、メソッドの名前に規則性を持たせておき、アプリケーションビルダがその命名規則にしたがって部品の情報を得る方法である。Java Beans では、これを設計パターンと呼ぶ。設計パターンはプロパティおよびイベントの名前に関して適用される。

(a) プロパティのための設計パターン

プロパティを読んだり書いたりするには、"get" と "set" をプロパティ名と結合した名前のメソッドを利用する。例えば、Foo というプロパティに対して、次のようなメソッドを使ってプロパティの値を設定したり読みだしたりする。

```
void setFoo(Xyz bar);
Xyz getFoo();
```

インデックスつきのプロパティの場合は、インデックスを示すパラメータを追加する。例えば、次のようになる。

```
void setFoo(int index, Xzy bar);
Xyz getFoo(int index);
```

また、論理型のプロパティを調べるメソッドには、プロパティ名の前に "is" をつけた名前を用いる。例えば、次のようになる。

```
void setFoo(boolean thuth);
boolean isFoo();
```

(b) イベントのための設計パターン

イベントリスナの登録には、EventListener のインタフェース名の最初に "add" をつけた名前のメソッドを用いる。また、同様にイベントリスナの削除には "remove"をつけた名前のメソッドを用いる。例えば、FooListener というインタフェースに対しては、次のようなメソッドが対応する。

```
void addFooListener(FooListerner listener);
void removeFooListener(FooListerner listener);
```

(3) イベント・モデル

Java Beans は、JDK 1.1 のイベント・モデルを使って Beans のイベントを定義する。

イベントハンドラは、リスナ(listener)と呼ばれるクラスで定義する。一つのオブジェクトには複数のリスナを登録できるようになっており、各リスナのメソッドが起動されることによって、オブジェクトのイベントが伝わる。

AWT の各コンポーネントには、イベントの種類に応じてイベントハンドラを登録するメソッドが用意されている。例えば、Button オブジェクトにイベントハンドラを登録するには、public void addActionListener(ActionListener) というメソッド呼び出して、ActionListner 型のリスナを登録する。ActionListener は次のように定義されるインタフェースである。

```
public interface ActionListener extends EventListener {
    public void actionPerformed(ActionEvent e);
}
```

イベントが発生する際に渡したい情報は、java.util.EventObjectのサブクラスで実現し、リスナのメソッドにそのオブジェクトを渡すことによってその情報を伝播させる。例えば、ボタンのように ActionListener を登録している場合は、ActionEvent オブジェクトをパラメータとして受け取る。

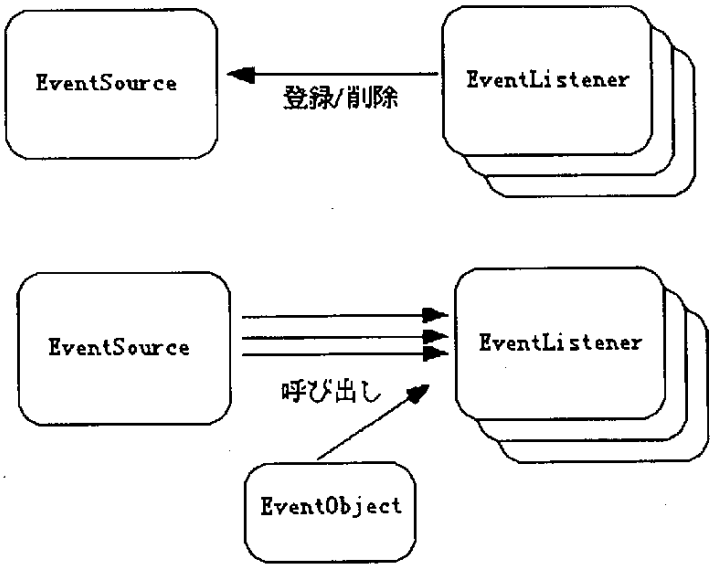


図1.3-7 イベント・モデル

次のプログラムは、JDK 1.1 のイベントモデルでボタンのイベントを処理する例である。

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class B extends Applet implements ActionListener {

    public void actionPerformed(ActionEvent e){
        System.out.println(e);
    }

    public void init(){
        Button b = new Button("press");
        b.addActionListener(this);
        add(b);
    }
}
```

(4) カスタマイズ

設計時にアプリケーション開発者が Beans の見かけや振舞いをカスタマイズするために、視覚的にプロパティを変更する手段が何種類か提供されている。

(a) プロパティ・エディタ

プロパティの値を編集するだけで事足りるような単純な部品については、あらかじめ用意されている単純なプロパティエディタを利用してプロパティを編集し、カスタマイズした結果を保存する。

(b) プロパティ・シート

いくつものプロパティを含んだ Beans をカスタマイズする際には、個々のプロパティの種類に応じたプロパティ・エディタを一つのウィンドウに集めたプロパティシートを利用することができる。

(c) カスタマイザ

複雑な部品の場合は、部品の提供者がカスタマイズのために利用するための自作のプロパティエディタを付属させておき、それを利用する。例えば、ウィザード方式で質問に答えるようなものも考えられる。

(5) 永続性

Java Beans は、部品のインスタンスを永続的な記憶に保存したり復元することができる。永続的な記憶には、カスタマイズのために利用したいプロパティを保存する。アプリケーション開発者は、保存されているプロパティをそのまま利用したり、あるいは、アプリケーションの設計上の要求に応じてプロパティを編集することで、部品の振舞いをカスタマイズすることができる。

OLE や OpenDoc のドキュメントに Java Beans の部品を埋め込んだり、また逆に、Java Beans から OLE や OpenDoc のデータを利用することを実現するために、さまざまな種類のオブジェクトの保存方法をサポートすることが望まれる。

Java のみで実現されている Beans は、Java Object Serialization のメカニズムを利用する。Java 以外の部品が含まれる場合は、現在開発中の externalization の仕組みを利用することになる。

(6) パッケージ化

Java Beans は、個々の部品をそれぞれまとまった形で取り扱う。Java Beans をパッケージ化するために、JAR ファイルと、シリアライゼーションの技術を使う。

JAR ファイルには、部品を実現するのに必要になるクラスファイルやリソースファイルを一まとまりにした Java Beans のパッケージが一つ以上含まれる。Java Beansのために用意された JAR ファイルの機能は、次のとおりである。

* JAR ファイルに対して beans 名を与えて、その beans の情報を取り出す。
* JAR ファイルに対して、含まれるすべての beans を列挙する。beans を列挙するために、どんな Java Beans が含まれているかを記述するインデックスが入る。

1．3．4　Java Beans の利用例

アプリケーションビルダを使ってアプレットを作る場合のシナリオを見てみよう。

(1) アプリケーションビルダと Beans を用意する

開発者	* アプリケーションビルダを購入する。 * 必要になりそうな Java Beans のコンポーネントを購入する。 * アプリケーションビルダに、購入したコンポーネントを登録する。
ツール	* このときアプリケーションビルダは、イントロスペクションの機能を使って、そのコンポーネントのプロパティなどを知ることができる。

(2) Beans を配置する

開発者	* アプリケーション開発者は、新しいアプレットを作るためのメニューを選択する。 * アプリケーション開発者は、Beans のメニューからボタンの部品をドラッグして、新しいアプレットの画面にドロップする。 * また、データベースの内容を表示するような部品を配置する。このような複雑な状態をもつ部品は、初期化された状態を保存されていることが望ましい。
ツール	* アプリケーションビルダは、自動的にボタンを作って配置するようなコードを生成する。 * 永続的な形になっているオブジェクトに対しては、インスタンス化するコードを生成する。

(3) カスタマイズする

開発者	* アプリケーション開発者は、ボタンという部品のプロパティをエディットすることができる。
ツール	* アプリケーションビルダは、BeansInfo オブジェクトを使ってその Beans のカスタマイザが定義されているかどうかを調べることができる。カスタマイザが定義されていない場合には、BeanInfo オブジェクトからその Beans の読み書き可能なプロパティを調べ、汎用のプロパティシートでプロパティの値を編集する。 * カスタマイザが定義されている場合は、アプリケーションビルダはそのカスタマイザのインスタンスを作り、起動する。カスタマイザはいくつかのダイアログボックスを表示しながらプロパティを設定してゆく。

(4) Beans を組み立てる

開発者	* 各部品の機能をどう組み合わせるかを定義する。例えば、ボタンがクリックされた時にどの部品のどの機能を利用するかということを指定する。
	* アプリケーション開発者は、メソッドのソースファイルを編集する。例えば、ボタンをクリックしたときにデータベースの内容を表示する部品(データベースビューア)の表示を更新するコードを次のように記述する。
	class Foobaz extends WombatBuilderApplet {
	public Button button_1;
	public DatabaseViewer viewer_1;
	void button_1_Action(java.awt.event.ActionEvent e){
	viewer_1.update();
	}
	}

ツール	* アプリケーションビルダは、編集されたメソッドからボタンとデータベースビューアを関連づけるコードを自動生成する。クラスの名前は自動生成される。
	class Hookup1234 implements java.awt.event.ActionListener {
	public Foobaz target;
	public void actionPerformed(java.awt.event.ActionEvent e){
	target.button_1_Action(e);
	}
	}
	* アプリケーションビルダは、Hookup1234 クラスのインスタンスを作り、addActionListener メソッドを使ってボタンにイベントハンドラを登録する。

(5) アプレットをパッケージ化する

開発者	—
ツール	* アプリケーションビルダは、Serialization の仕組みを使って Foobaz オブジェクトの内容を Foobaz.ser というファイルに保存する。ボタンとデータベースビューアのインスタンスも保存される。また、これらのオブジェクトの初期化コードもいっしょに保存される。 * アプリケーションビルダは、Foobaz.jar という JAR ファイルを作り、必要になるクラスファイルをすべてその中に入れる。そのアプレットを試すためのHTML ファイルも JAR ファイルに含まれる。

(6)実行する

開発者	* Web ブラウザでアプレットの含まれるページを見ようとすると、Foobaz.jar に含まれる Foobaz.ser からオブジェクトの状態を復元し、クラスファイルを取り出してロードする。 * すべての部品が初期化された後に機能するようになり、アプレットの中に含まれる部品を利用できるようになる。
ツール	—

1．3．5 BDK と BeanBox

BDK (Beans Developers Kit) は Java Beans を開発するためのツールである。 BDKには、Java Beans API のパッケージと BeanBox と呼ばれるサンプル実装のコンテナやデモ用の Beans が含まれている。 BDK は、参考文献の[1]から入手することができる。

【参考文献】

1) http://splash.javasoft.com/beans/

2) http://java.sun.com/

3) http://www.cilabs.org/About/roadmap.html

1．4　ソフトウェアエージェント

1．4．1　エージェントについて

今や「エージェント」という言葉は、かつての「AI」や「人工知能」と似た語感を持ちつつあり、その意味する所も漠然としている。例えば、石田[1]による次のような分類がある。

概念レベルの用法

1. 自律知能を目指す用法　(Autonomous agent、Intelligent agent)

2. 分散知能を目指す用法 (Society of mind、マルチエージェント)

応用レベルの用法

1. Software agent

2. Interface agent

3. Beleivable agent

実装レベルの用法

1. Agent-Oriented Programming

2. Mobile Agent

3. Telescript Agent

概念レベルの用法の分類では、「自律性」とか「知能」とかより抽象的な用語が出てくるのが苦しい所である。ただし、利用者の代理として何か知的な行動を行うものという点では、ある程度のコンセンサスを得ているように思われる。例えば、メイルが来ると教えてくれるUnixのbiffをエージェントと呼ぶのは苦しく、利用者の興味のあるメイルだけを優先的に教えてくれるようなプログラムはエージェント色が強い。

応用レベルの用法の分類は、比較的明解である。画面の中に召し使いが出てきて、それと対話しながら仕事を行う、AppleのKnowledge Navigatorのようなものがインタフェースエージェントである。Believable Agentは、情感を持つように感じられるソフトウェアのことである。ソフトウェアエージェントは、利用者の代わりにネットワーク上で働くプログラム全般を表す。

実装レベルの、Agent-Oriented Programming (AOP) は、スタンフォード大学のShohamにより提唱されているプログラム言語である。Autonomous Agentにおける信念(belief)や目標(goal)を記述することを目指している。Mobile Agentは、ネットワークを移動して何か処理を行うソフトウェアを記述するためのプログラム言語である。Telescript Agentは、Mobile agentの中の一つとみなすことができる。General Magic社のTelescript言語により記述されるエージェントのことである。これらはソフトウェアエージェントの記述言語といえる。

本節では、ソフトウェアエージェントについて、現状を交えながら紹介する。また、その中のモーバイルエージェントであるTelescript言語について、詳しく紹介することにする。

1．4．2 ソフトウェアエージェントの概要

ソフトウェアエージェントとして、利用者の代わりにネットワーク上で動くプログラム全般を表すものと考える。ソフトウェアエージェントはネットワーク上の形態により、Static AgentとMobile Agent (モーバイルエージェント)の二種類に大きく分けることができる。さらに、厳密にはソフトウェアエージェントではないが、関連するネットワークプログラム技術についても本節でいくつか紹介する。

(1) Static Agent

Static Agentは、エージェント自体は動かず、他のエージェントとの間でメッセージ交換を行い、タスクを分割して協調して処理を行うことができる。メッセージ交換は、RPC(リモートプロシージャコール)のような低レベルのものではなく、交渉など高いレベルのプロトコルも用いる。エージェントは等質(ホモジーニアス)なものばかりでなく、異種(ヘテロジーニアス)な場合が多い。そこで、メッセージの交換においては次のような体系が必要となる。

- メッセージに入れる知識を共通に記述する体系
- メッセージ交換のプロトコルの記述体系
- メッセージに含まれる語彙体系の記述

Static Agentの最も有名な例としては、スタンフォード大学で中心的に行われているACL (Agent Communication Language) がある[2]。これは、ARPAがスポンサーとなって行われた、Knowledge Sharing Effortプロジェクトで生まれた、KQML (Knowledge Query and Manipulating Language)[3,4]、KIF(Knowledge Interchange Format) [5]、Ontolinguaといった言語から構成される。

(a) KIF

異種のエージェント間でコミュニケーションを行う共通知識言語がKIFである。KIFは多くの知識表現言語を集約したような言語であり、高階の述語論理に加え、非単調論理や、集合もデータ構造も含まれている。KIFは、それ自身を処理するというよりは、異なる知識表現で表現された知識を交換するための体系である。

(b) KQML

同じ言語で話せば即コミュニケーションが取れるとは限らない。人間のコミュニケーションも、質問に対しては答えるとかのプロトコルにより成立する。エージェント間のメッセージのプロトコルを規定するのが、KQMLである。KQMLではメッセージをパフォーマティブと呼ばれる形式で記述する。パフォーマ

ティブは例えば次のような形式をしている。

tell

:content <expression>

:language <word>

最初にスピーチアクトと呼ばれる、メッセージの意図する行為を表す予約語が来る。例えば、tell(言明)、reply (返信)、ask-one (一人に尋ねる)、ask-all (全員に尋ねる) などがある。続いて、予約語として決まっているパラメータとその値のペアがくる。

(c) Ontolingua

言語とプロトコルが決まってもそれだけでは十分ではない。例えば、あるエージェントでは「WHITE」という語で表す実態は、別のエージェントでは「白」で表すかもしれない。こういった共通な語彙体系のことをオントロジーと呼ぶ。オントロジーを記述言語がOntolinguaである。

(2) モーバイルエージェント

モーバイルエージェント(オブジェクト)は、その名の通り実際にプログラムで記述されたエージェントがネットワークを移動して、移動先のマシンでタスクを行うものである。加藤[6] によれば、

70年代　FTP　ファイルのモビリティ

Telnet　ログインセッションのモビリティ

80年代　X-Windowグラフィックのモビリティ

90年代　WWW　マルチメディアコンテンツのモビリティ

と、計算機の世界でモビリティは発展しており、その次に来るのがモーバイルエージェント(オブジェクト)による、プログラム・データ・計算状態のモビリティであるという。モーバイルエージェントは、何らかのプログラム言語(orスクリプト言語)で記述され、移動先のマシンのリソースを使ってタスクの実行を行う。動作形態の似ているコンピュータウィルスのようにならないためには、作成者の認証や、ネットワーク経路におけるセキュリティ、さらに相手先マシンを簡単にダウンさせないようなセーフティが必要である。

モーバイルエージェントとしては、General Magic社のTelescript(http://www.genmagic.com) が有名である。また、最近では、Javaによるモーバイルエージェントもいくつか発表されている[7]。例えば、日本IBMのAglets Workbench (http://www.trl.ibm.co.jp/aglets)、カリフォルニア大バークレー校で開発されたJava-To-Go(http://ptolemy.eecs.berkeley.edu/dgm/javatools/java-to-go/)、独シュツットガルト大学のMOLEプロジェクト (http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole.html)、米国FTP Softwareによる CyberAgent (http://www.ftp.com/)などがある。他に、米国ダートマスカレッジによるTcl/Tkベースの

AgentTcl(http://www.cs.dartmouth.edu/~agent/) や、 Perlベースの Penguin(http://www.eden.com/~fsg/penguin) 、SunによるNetscape Navigator用のTcl Plug-in (http://www.sun.com/960710/feature1/plugin.html)などもある。

以下、Telescript以外のいくつかのモーバイルエージェントについて紹介する。

(a) Aglets Workbench

Aglets Workbench (AWB) は、日本IBM東京基礎研究所にて開発されたモーバイルエージェントである。Agletsと呼ばれるエージェントが、インターネット上を移動し、移動先のサービスを受けたりデータを獲得したりする。また、移動先で他のAgletsと情報をやりとりしたりすることもできる。Java Aglet API(JAAPI) が提供されており、JavaによりAgletsの動作を記述することができる。

AWBのグループは、モーバイルエージェントの標準化としてAgent Transformation Protocol (ATP) も提唱している(http://www.trl.ibm.co.jp/aglets/atp/index.html)。ATPは、インターネット上でURLによりエージェントの移動先を指定するプロトコルである。また、ATPをはじめとするMobile Agent Facility (MAF)は、OMGに提出されている。AWBのセキュリティ機構は三層構造である。

第1層は、Java言語システムのセキュリティで、Java VMによりコードのチェックなどを行う。第2層は、セキュリティマネージャによるアクセス制御である。ビジュアルエージェントマネージャのTahitiから、ホストにアグレットのファイルアクセスなどが制限できる。第3層は、Java Security APIで、暗号化、デジタル署名、認証などがサポートされる。

なお、現在AWBはAlpha 3が同ホームページよりフリーで公開されている。

(b) Java-To-Go

カリフォルニア大バークレー校で開発されたJava-To-Goは、Javaベースのモーバイルエージェントの実験システムである。

エージェントと、それが移動する場としてのホールサーバ(Hall server)とで構成される。エージェントは、クラス定義、初期状態、エージェントが訪れるホールサーバのリストまた集合から構成される。Telescriptとは違って、あるサーバでの処理をサスペンドして残りをサーバで行うということはできない。その代わりに、サーバに入った時に行うbootと、サーバを出る時に行うwrapupという関数で、内部状態を変更しながら、複数サーバを渡り歩いて処理を行うことができる。

(3) その他のネットワークプログラム技術

厳密な意味ではエージェントではないが、ネットワーク(インターネット)で重要な技術は多い。ここでは、ソフトウェアエージェントに関連の深い技術を取り上げる。

(a) Java

Javaのアプレットも、ネットワークを処理単位が動いて別マシンで実行されるという点では、モーバイ

ルエージェントと似ている。ただし、Telescriptなどのモーバイルエージェントは、エージェントはマシンからマシンへと双方向に渡り歩くのに対して、JavaのアプレットはWWWサーバからWWWブラウザであるクライアントにダウンロードされるのみである。また、利用者が要求することでアプレットの移動が行われるのも、プログラムの中で移動を指定するTelescriptとは異なる点である。

ただし、電総研の平野聡氏により開発されたHORB(http://ring.etl.go.jp/openlab/horb/) や、JDK1.1 における RMI (RemoteMethod Invocation)、前項(2)で説明したJavaにより書かれたモーバイルエージェントなど、Javaも徐々にネットワークプログラム言語としての機能が備わりつつある。

(b) WWWロボット

日々増加の一途をたどるWWWのリソースを検索するには、定期的にWWWにアクセスして随時インデックスを更新し続けるか、情報発信者からのデータを集めて体系的に整理するかの方法が取られている。後者がYahoo! などのディレクトリサービスであり、前者がWWWロボットによるWWW検索である。

WWWロボットは、アンカータグにより結ばれたHTMLの空間を定期的に探索し、追加更新されたページから文字・キーワードベースでインデックスを作成する。このような力まかせなアプローチでは、サーバやネットワークに負荷が大きい。サーバ側で /robots.txtというファイルを作ってロボットがアクセスしてはいけないURLを記述したり、1サイトにアクセスできるのは1分間に1回だけなど、各種の制限を加える方向にある。

1．4．4　Telescript

Telescript[9]は米国General Magic社の提唱するモーバイルエージェントである。Telescriptの技術は長らくGeneral Magic社のスポンサー企業だけに留まっていたが、1995年の10月に方針変更でインターネット(http://www.genmagic.com/)を通じて、処理系を含む情報が公開されるようになった。また、1996年になってWWWとの連携を目的としたシステム (ActiveWeb Tools、およびその後継のTabritz) も発表されて、インターネットをターゲットとしたエージェント技術への転換をとげつつある。

Telescriptはエージェントとプレースによる明解なモデル化が特徴である。また、セキュリティおよびセーフティについても、システムの基底から考えられており、他のモーバイルエージェントにも影響を与えている。

(1) Telescriptのモデル

(a) リモートプログラミング(RP)とリモートプロシージャコール(RPC)

これまでのRPCでは、タスクを行う間ずっとクライアントとサーバの間でセッションを張っていなければならなかった。また、サーバでの処理が固定化して、タスクにおける判断は全てクライアント側で行う

ため、両者の間で頻繁にメッセージのやりとりが行われることになる。それに対して、Telescriptが目指すのはリモートプログラム(RP)である。これは、処理手続きをクライアント側でまとめてサーバに送り、処理自体はサーバで行われるというものである。RPだと、サーバに処理を一度送ったら、コネクションは切ってもかまわない。これは例えば、電話回線でアクセスしているユーザには有効である。この処理手続きのことを、Telescriptではエージェントと呼ぶ。また、エージェントが移動する場のことをプレースと呼ぶ。

(b) エージェント

Telescriptのエージェントは、Telescript言語で記述されるソフトウェアプロセスである。エージェントの移動は、プログラム中のgoという命令で行なわれる。go命令の直前の内部状態のまま、エージェントはネットワークを実際に移動して、移動先のプレースで続きの処理を行なう。

エージェントは、自分が現在いるプレースのオペレーションを呼び出すことで、プレースと情報をやりとりすることができる。また、同じプレースにいるエージェントとミートすることで、互いにオペレーションを呼びあうことができる。

(c) プレース

プレースも、Telescript言語で記述されるソフトウェアプロセスである。プレースは、移動できないことを除いてプログラム上はエージェントと同様に記述される。プレースは、エージェントが移動する場の働きをする。プレースの上にプレース(サブプレース)が乗っていてもよい。移動先のプレースを指定する方法は、プレースのクラス、テレアドレス (URLのようなもの)、テレネーム (プロセスのネットワーク上のIDで、IPアドレスのようなもの) など用途に応じて柔軟に使い分けることができる。図1.4-1は、エージェントとプレースの例である。

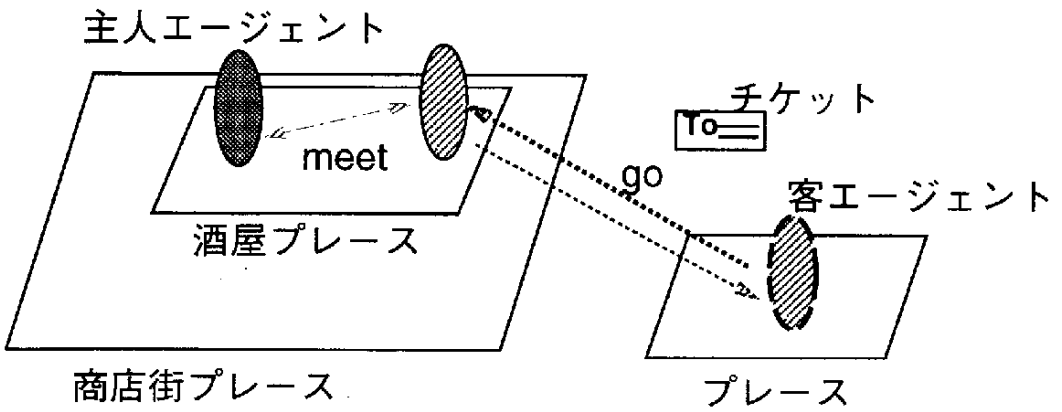


図1.4-1 Telescriptのプレース/エージェントモデル

(2) Telescript言語

Telescript言語には、High TelescriptとLow Telescriptの二種類がある。High Telescriptで書かれたプログラムは、コンパイラでLow Telescriptに変換されて、エンジン(インタープリタ)で実行される。ここでは、High Telescriptだけを対象とする。

Telescript言語は、C++に似た構文を持つオブジェクト指向言語である。モジュール、クラス継承(ミックスインクラスによる疑似多重継承)、オーバーロード、オペレーションやアトリビュートの複数のスコープなどの特徴を持っている。Telescript言語は汎用のプログラム言語であるが、速度の点などから処理の全てを記述するのには適していない。エンジンで提供されている外部プログラムAPIを通じて、C++などによる処理本体を呼び出すことができる。ここでは、Telescript言語のいくつかの特徴について実例を交えながら紹介する。

(a) クラス階層、クラス継承

Telescript言語には60以上の組み込みクラスがある。クラスにはObjectクラスから派生するフレーバークラスと、疑似多重継承を行うミックスインクラスとがある。例えば、Placeクラス、AgentクラスはProcessクラスのサブクラスである。また、プレースの上でエージェント同士がミートをして情報交換をするためには、それぞれ、MeetingPlaceやMeetingAgentといったミックスインクラスを継承したクラスとして記述しなければならない。

(b) エージェント、プレース

Telescriptのプロセス(エージェントとプレース)は、一般に図1.4-2のような形式でコーディングされる。1行目はFujitsuGinzaというPlaceクラスのサブクラス定義の始まりである。2行目は以降のオペレーショ

```
1: FujitsuGinza: class(Place) = (
2:   public
3:    initialize: op(p:xxx) = {
4:      初期化
5:    }
6:    live: sponsored op(Exception| Nil) = {
7:        Sakaya(Permit(1000));
8:          処理内容
9:        loop{};
10:   }
11:   other_op: op(xxx) = { 処理定義 }
12: )
```

図1.4-2 Telescriptのプログラム例1(プロセス)

ンのスコープを表す。ここでは、初期化を行うinitializeおよび、処理メインのオペレーションであるlive、それにother_opが定義されている。7行目はSakayaという他のプロセスを生成している。引数は、このSakayaプロセスの能力を表すパーミットで、この場合は最大1000秒間動くことができることを表す。パーミットについては後述する。SakayaプロセスはFujitsuGinzaの上にできる。特にプレースは9行目のように無限ループでコーディングされることが多い。プレースが終了すると、その上にいるエージェントやプレースも消滅してしまうためである。

(c) エージェントの移動

go(チケット) という簡単な命令でエージェントは移動することができる。チケットには移動先や移動方法が表されている。行き方は、プレースのクラス、テレネーム、テレアドレスなど柔軟な指定ができる。

(d) エージェントのミートおよび情報交換

エージェント同士で情報交換を行うには次の手順で行う。

1. 相手と同じプレースに行く
2. ミート相手をペティションというオブジェクトで指定する
3. meet(ペティション) という命令の結果として相手の参照を得る
4. ミートされた相手は、自分のmeetingオペレーションがプレースから呼ばれる
5. ミート中は相手のオペレーションを互いに実行できる
6. どちらかがパートすることでミート終了

ペティションも、チケットと同様にクラス、テレアドレス、テレネームと柔軟な相手指定が可能である。エージェントの移動とミートを行う典型的なliveオペレーションは次のようになる。ここでは、エージェントが酒屋プレース (Sakayaクラス)に移動し、主人(Masterクラス) とミートする。

```
1:  live: op(Exception! Nil) = {
2:    try {
3:      self.go(Ticket(nil, Sakaya.name));
4:      pet := Petition(nil, Manager.name);
5:      try {
6:        mng := here@Sakaya.meet(pet);
7:        mng@Manager.order(order_list);
8:        here@Sakaya.partAll();
9:      } catch MeetingException {
10:       String("Could not meet").dump();
11:     }
12:   } catch error: Exception { error.dump(); }
13: }
```

図1.4-3 Telescriptのプログラム例2 (ミート)

3行目が移動命令である。引数のTicketはチケットクラスで、第2引数は行き先のクラス指定である。Sakaya.nameは、酒屋クラスのクラス名を表している。4行目以降は移動先のSakayaプレースでの実行となる。4行目でペティションによりミート相手を定義している。ここも第2引数はクラス名による指定である。6行目がmeet命令である。here@Sakayaは、現在いるSakayaプレースを表す。エージェントが現在いるプレースのmeetオペレーションを呼んでいる。その結果、希望する相手への参照をmngにより得ることができる。7行目で、相手(Managerクラス)のorderオペレーションを呼んでいる。8行目で、現在いるSakayaプレースに対して、ミートの修了処理を行っている。9行目以降は例外処理である。最初はミート相手が見つからなかった時の処理、12行目は全体で何か例外が起こった時の処理である。

(e) パーミット

パーミットは、エージェントやプレースが、無限にリソースを消費しないように制限をつけるものである。図1.4-2の7行目のように、プロセスの生成の引数にはパーミットが必須である。パーミットを使い尽くすと、プロセスは消滅する。パーミットで規定できる能力の種類には次のようなものがある。

age : プロセスの稼働時間(秒)

charges : プロセスの計算資源量(テレクリックという単位)

extent : プロセスが使用可能メモリ (オクテット)

プロセス生成時に与えられるパーミットはネイティブパーミットと呼ばれる。移動先のプレースによっては、その能力が全部発揮できない場合もある。また、パーミットの一部を取っておいて、残りのパーミットを使い尽くした時の例外処理に使う方法もある。

(f) 安全性とセキュリティについて

モーバイルエージェントでは、悪意のある利用者がコンピュータウィルスのようなエージェントを作れないように、またエージェントのプログラムのバグでみだりにサーバがダウンしないような仕組みが必要である。前者が安全性(safety)、後者がセキュリティの問題である。

安全性に関連して、Telescript言語には次のような機能がある。

- ポインタ操作を使ったコーディングができない
- メモリ管理はシステムのGCにより行う
- try...catchの例外処理記述
- 実行前にコードベリフィケーション
- プロセスの永続性 (システムがダウンしても復旧できる)

セキュリティに関連して、Telescript言語には次のような機能がある。

- 全てのオブジェクトには、責任者であるオーソリティの情報が明記されている

- パーミットによる資源管理 (無限に動くエージェントは作れない)

- ネットワークにおける、暗号化や認証

(3) 処理系および応用

(a) Tabritz

現在General Magic社のホームページから処理系としてTabritzが提供されている。それには、次のようなTelescriptの開発環境、実行環境、WWW統合環境が含まれている。

- TCP/IPベースで動作するTelescripエンジン

- Webクラスライブラリのフルセット

(TelescriptをWebから使うActive Web Serverライブラリ、Telescriptオブジェクトを HTMLで表現するHTMLサポートライブラリ、TelescriptプロセスからHTTPリクエストを行うWebアクセスライブラリ)

- グラフィカルなTelescript開発統合環境

(ソースレベルデバッガ、クラスブラウザ、プログラムマネージャ)

- NCSA Webサーバ

(b) 応用例

Telescriptやモーバイルエージェント一般のアプリケーションとしては、次のようなものがあげられている [8,9]。

(i) ウォッチング

エージェントをプレースに送って、必要な情報が手に入るまで待つ。例えば、株価がある値段になったら利用者に教えてもらうなどがある。

(ii) サーチング

ネットワーク上のリソース、DBについてのメタ情報を持ち、利用者の要求に応じて適切な(複数の)プレースに行き検索を行って結果を持ち帰る。例えば、ある商品の値段が最も安い、1時間以内で行けるカメラ店を探すなどがある。

(iii) オーケストレーション

上のようなアプリケーションを組み合わせで、複数の異種のプレースを廻って複雑なタスクを行う。例えば、ある作家のセミナーの講演や出版の記録をまとめるのに、エージェントが大学プレースや出版社プレースから異種の情報を検索してまとめるなどがある。

【参考文献】

1) 石田: エージェントを考える,人工知能学会誌, Vol.10, No.5,pp.663--667, Sept. 1995.

2) 西田: ソフトウェアエージェント,人工知能学会誌, Vol.10, No.5,pp.704--711, Sept. 1995.

3) Tim Finin: Knowledge Query and Manipulating Language (http://dis.cs.umass.edu/kqml/), University of Meryland.

4) Tim Finin, Richard Fritzson Don McKay and Robin McEntire: KQML as an Agent Communication Language, 3rd International Conference on Information and Knowledge Management (CIKM94),1994.

5) Genesereth, M.R and Fkes, R.E., : Knowledge Interchange Format Version 3.0 Reference Manual, Report Logic-92-1, Stanford Univ., 1992 (http://logic.stanford.edu/kif.html).

6) 加藤: モーバイルオブジェクト・コンピューティングとデータベース,pp. 93--98, アドバンストデータベースシンポジウム96, 1996.

7) 小野,黒田: Javaの世界を広げる最新エージェントテクノロジー,pp. 134--141, JavaWorld, No.2, IDGコミュニケーションズ, 1996.

8) S.D. Griswold etc.: エージェント技術の現在, Internetworking, Vol.8,pp.31--55, アスキー,1996.

9) 山崎,津田 編訳: Telescript言語入門, アスキー, 1996.

1．5 Web-CORBA連携

本節では、Webと分散オブジェクト技術を用いた新しいタイプの3階層業務システムの基本構造について述べる。

1．5．1 背景

(1) Web

インターネットの爆発的普及は、情報システムの姿に多大な影響をもたらしてきている。インターネットの技術（TCP/IP, 電子メール, WWW[1]など）を企業内ネットワークに適用したイントラネット、企業間ネットワークに適用したエクストラネットなど、インターネットの利用形態が本来のインターネット（The Internet）以外にも拡大してきている。以下では、イントラネットやエクストラネットも含めて「インターネット」と総称して呼ぶことにする。

インターネット、特にWWWの普及により、クライアントPCにはWebブラウザが常備されるようになってきた。さらには、Webブラウザ専用の安価なネットワーク・コンピュータ（以下ではNCと略）や、Webブラウザとして利用できる機能をもつ家庭用テレビが発売されるようになってきている。

これに伴い、アプリケーション・システムのGUIとしてをWebブラウザを使いたいというニーズが増えてきている。これは、「Webブラウザというひとつの対話システムでインターネットばかりでなく、企業内のシステムを含めたすべてのコンピュータシステムを利用できるようにしたい」というユーザサイドのニーズだけによるものではない。「Webブラウザで表示・操作できるようなシステム形態にしておけば、たいていのユーザからアクセス可能になる」というシステムサイドのニーズにももとづいている。また、Webサーバからクライアント・プログラムをダイナミックにダウンロードして実行する技術（Java[2]やActiveX[3]など）によって、CSSシステムの運用コストのかなりを占めるクライアント・ソフトウェアのインストール、バージョン管理のためのコストを大幅削減することができるという運用面でのメリットも享受できるようになる。特にJavaは、Webブラウザがインストールしてあれば、MacintoshやNCを含めた多種のクライアントマシンで実行可能であるという特徴がある。

図1.5-1に典型的なWebシステムの構造を示している。Webサーバ経由で通信プロトコルHTTP（Hyper Text Transfer Protocol）によって、HTML（Hyper Text Markup Language）言語やJava言語のプログラムがWebブラウザにダウンロードされ実行される。HTML言語のプログラムは「ドキュメント」と呼ばれ、ハイパーテキスト機能をもつ文書としてWebブラウザ上に表示される。Java言語のプログラムは「アプレット」と呼ばれ、HTMLにできないさまざまな処理を実現することができる。アプレットの典型的な利用法としては、HTMLではできないダイナミックな表示に使われるか、HTMLにできないデータストリームを

送信するなどの通信に利用されることが多い。HTML中からWebサーバ経由でサーバプログラムを起動することができる。Webサーバからサーバプログラムを起動するインタフェースとして、CGI（Common Gateway Interface）という共通のインタフェースを各種Webサーバがサポートしているので、ここでは「CGIプログラム」と表記している。CGIプログラムの典型的な利用法としては、HTML文書にユーザが入力したデータをもとに、データベースを検索し、検索結果にもとづいてHTML文書を生成するという機能を提供することが多い。

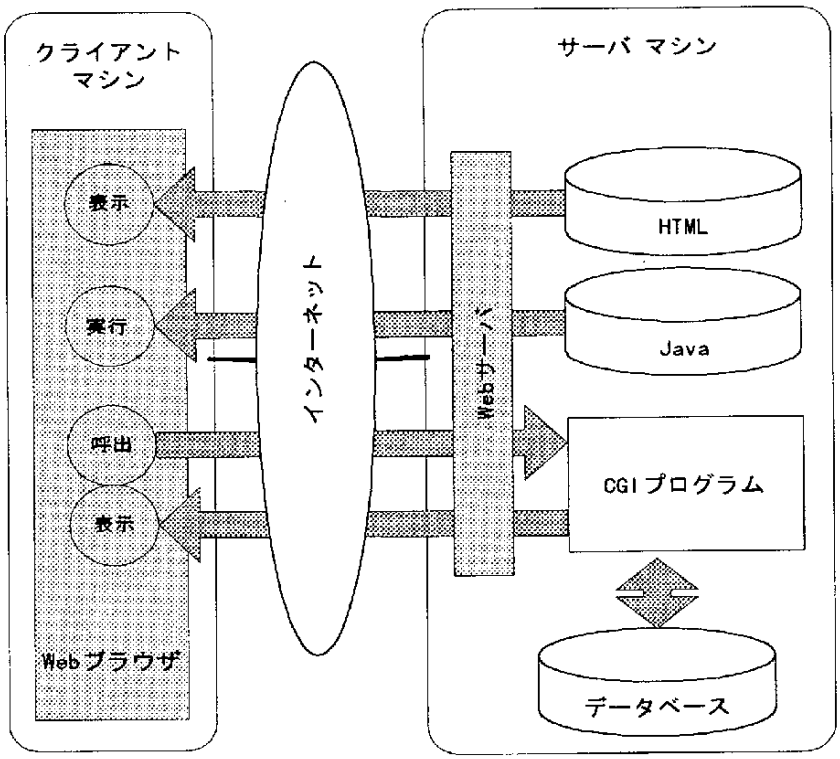


図1.5-1典型的なWebシステム

(2)分散オブジェクト

CORBA（Common Object Request Broker Architecture[4]）やDCOM（Distributed Component Object Model[5]）というに代表される分散オブジェクト技術は、オブジェクト指向パラダイムのもつ優れたソフトウェア工学的特徴を、クライアント･サーバシステムに適用するための基本技術として認知され始めている。サービスを提供するサーバプログラムをオブジェクトとして見立て、サービスを実現するサブルーチン一つ一つをメソッドと見立て、クライアントプログラムはサーバプログラムのもつメソッドを起動することでサービスを受けることができるという考え方である（図1.5-2）。

分散オブジェクト基盤は、上述の形態のクライアントサーバシステムを構築するための枠組みを提供する。サーバプログラムをオブジェクト指向型風にコールするためのRPCメカニズム、サーバプログラムのディレクトリ・サービスなどが提供される。

CORBA2.0には、異なるベンダーが製造したCORBAの通信部分（ORB: Object Request Broker）の間の相互接続を実現するためのプロトコルIIOP（Internet Inter-ORB Protocol）も提供されている。DCOMは、Windows NTの上で動作するのを基本としているが、CORBAはWindows NT、Unix、OS/2など各種サーバOS上で動作するのを基本としている。

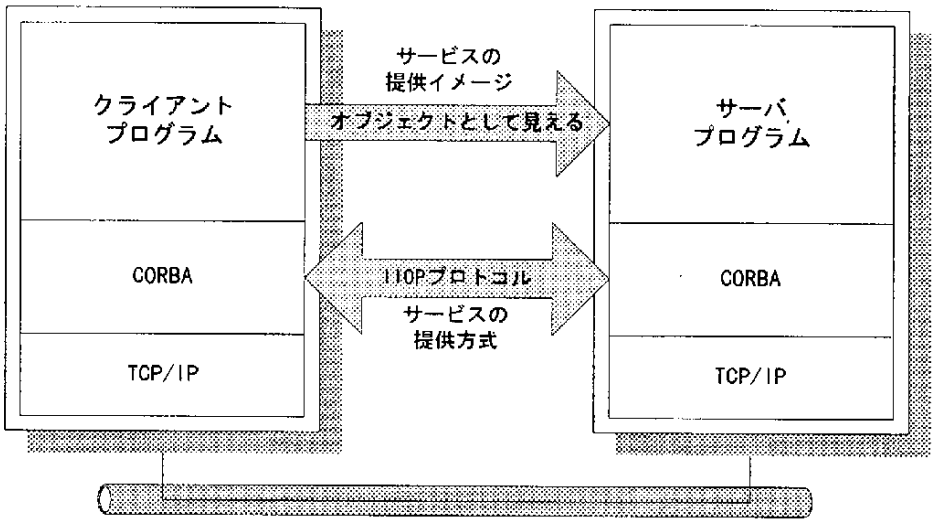


図1.5-2 CORBAによるクライアントサーバシステム

(3) 3階層モデルによるCSSアプリケーション開発

CSS形態でアプリケーションを開発する際、プレゼンテーション・レイヤ、機能レイヤ、データ・レイヤの3階層のオブジェクトに分割した構成で実現すると、各オブジェクトの再利用性を高めることができるという考え方がある。このプレゼンテーション・レイヤ、機能レイヤ、データ・レイヤの3階層のオブジェクトに分割したモデルを3階層モデル[6]と呼ぶ（図1.5-3）。

プレゼンテーションレイヤのオブジェクトは、ユーザ操作イベントによって呼び出され、機能レイヤのオブジェクトのメソッドを呼び出す。機能レイヤのオブジェクトは、データ・レイヤのオブジェクトのメソッドを呼び出すことにより、永続的データを取得する。データ・レイヤのオブジェクトは、永続データをオブジェクトとして表現したものであり、通常はデータベースに永続的データを格納することによって実現する。データ・レイヤのオブジェクトからデータを受け取った機能オブジェクトは、データを変換・集計・他データとの演算などを行い、呼出元のプレゼンテーションレイヤのオブジェクトに返す。

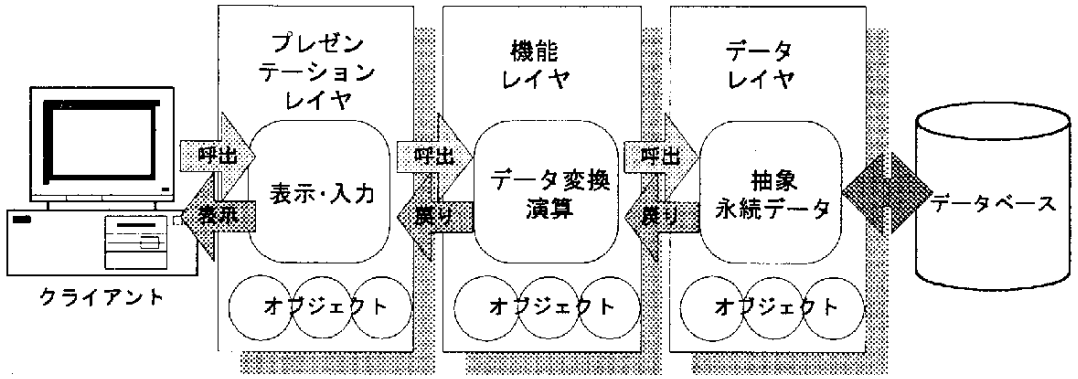


図1.5-3 3階層モデル

1．5．2　3階層モデルにもとづくWeb-CORBA連携システム

3階層モデルに、Webと分散オブジェクトという実装技術を持ち込むと、Webの位置づけ、分散オブジェクトの構成を図1.5-4のように決定することができる。

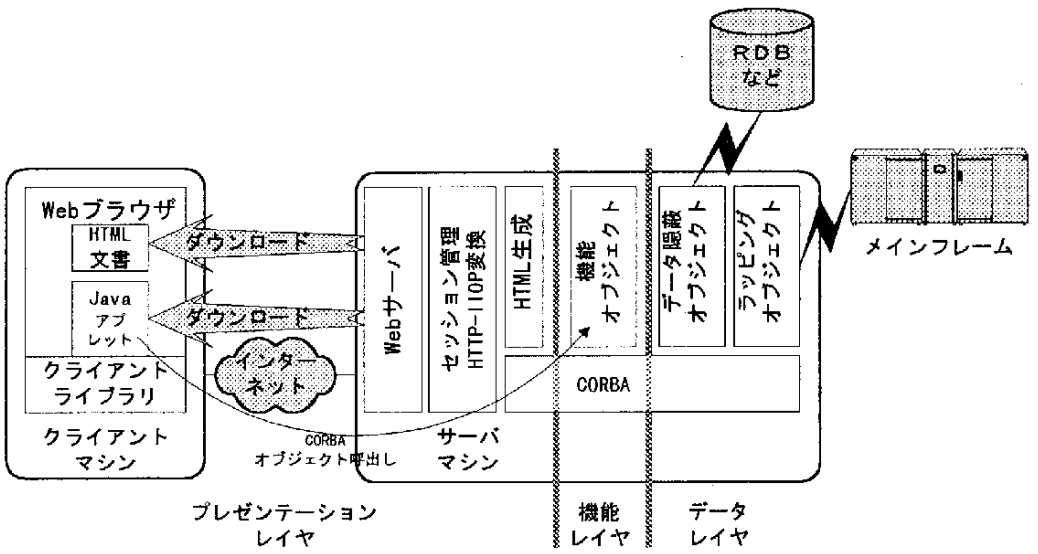


図1.5-4 3階層モデルにもとづくWeb-CORBA連携システム

(1) プレゼンテーション・レイヤ

Webシステムを前提とする3階層モデルにおいては、表示・プログラム実行制御をつかさどるWebブラウザとWebサーバが入り込んでくるため、プレゼンテーション・レイヤが詳細化される。

(a) Webサーバ：クライアント・マシンのWebブラウザとの通信を実現する。

(b) Webブラウザ：クライアント･マシンにおけるユーザインタフェースを担当する。

(c) HTML文書：Webブラウザ上に表示される。アンカーやフォーム、ボタンなどを含み、ユーザからの入力を受け付ける機能を持つ。

(d) HTML生成：HTML文書を動的に生成するモジュールである。HTML文書の枠組みと、その中に埋め込むコンテンツをデータベースから検索し合成してWebサーバに送り出すメカニズムで実現されることが多い（図1.5-5）。

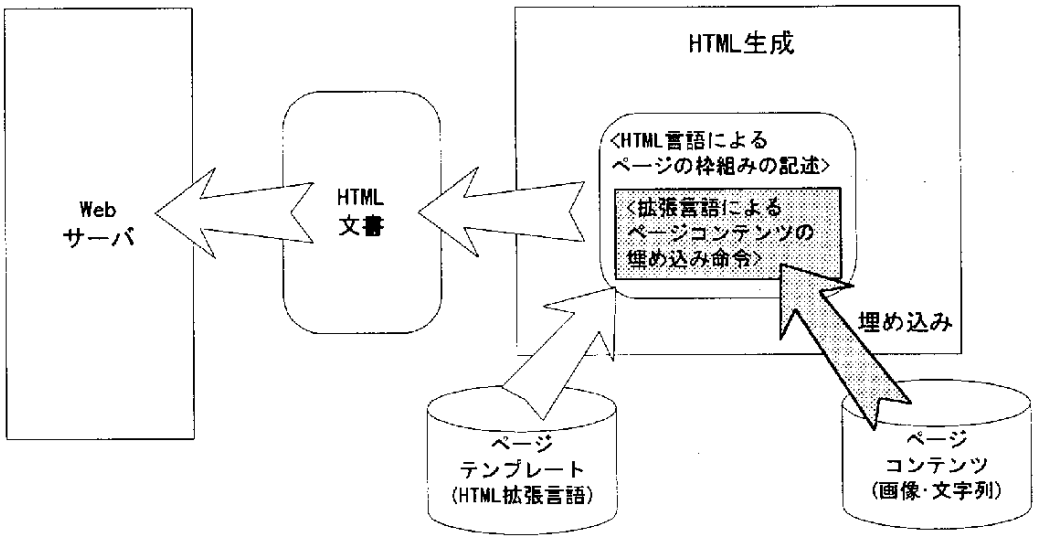


図1.5-5 HTML生成のメカニズム

(e) Javaアプレット：Webサーバ経由で動的にダウンロードされて実行されるプログラムである。サーバのアプレットをバージョンアップするだけで、クライアントで実行されるプログラムを変更することができるので、バージョンアップが予想されるプログラムはJavaアプレットとして開発しておくと便利である。Javaアプレットの機能としては、HTMLでは作れない画面を作ったり、HTTP以外の通信を行ったりするものが多い。なおここでは、Javaアプレットとしたが、ActiveXコントロールなど他にも同等の機能のものがある。

(f) クライアント･ライブラリ：クライアントで動作するdllなどである。バージョンアップされることが少ないプログラム、マシン語で実行されることが重要なほど性能にクリティカルなプログラム、セキュリティの観点から動的にダウンロードすることが不適当なプログラムなどは、Javaアプレットではなくクライアント･ライブラリとして実現される。

(g) HTTP - IIOP変換：WebサーバのうしろでCORBAのアプリケーションプログラムを動かすために、Web

サーバとの通信プロトコルHTTPと、CORBAのプロトコルIIOPとの間の変換を行う機能である。

(h) セッション管理

Webブラウザからアクセスしているセッションが継続しているかどうかを管理する機能である。

(2) 機能レイヤ

Webを導入したことにより、プレゼンテーション・レイヤと機能レイヤの役割分担がはっきりするというメリットが得られる。3階層モデルでシステムを組む際、プレゼンテーション・レイヤと機能レイヤとの境界を作りにくいという問題がよく見受けられる。Webをプレゼンテーション・レイヤとして利用すると、HTMLに依存する範囲をプレゼンテーション・レイヤとして切り出し、HTMLを生成するための基礎データとしてのコンテンツを作り出すところを機能レイヤとして、明確な役割分担が可能になる。

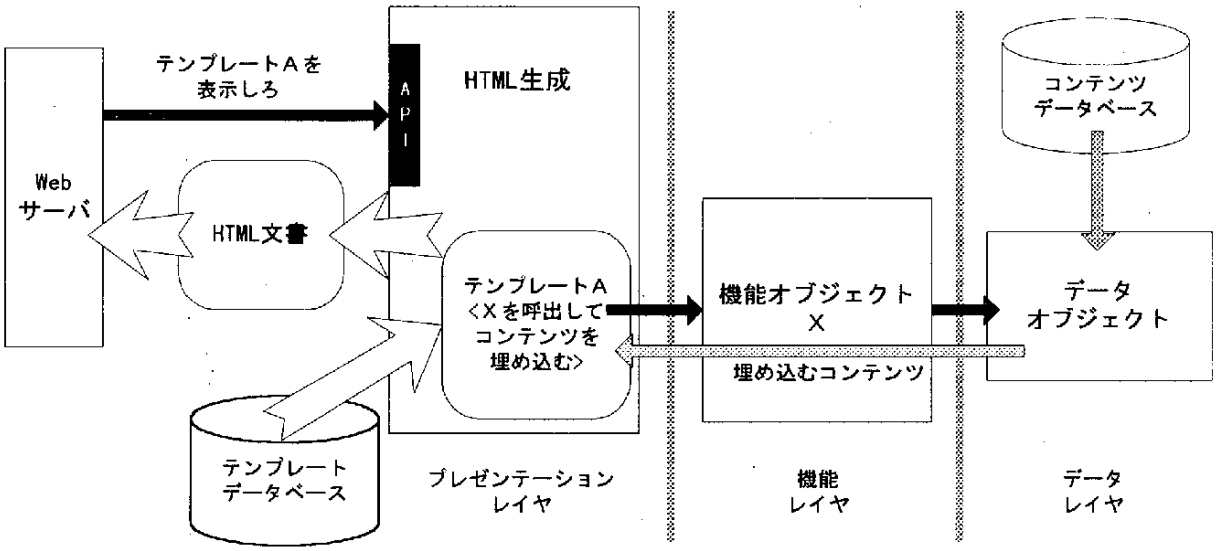


図1.5-6 プレゼンテンション・レイヤと機能レイヤとの役割分担

図1.5-6に役割分担の様子を示した。例として、WebサーバからHTML生成オブジェクトのAPIを通じて「テンプレートAを表示しろ」と起動したとしよう。するとHTML生成オブジェクトはテンプレート・データベースからテンプレートAを読み出してくる。テンプレートはHTMLの拡張言語で記述されている。拡張言語の部分で機能オブジェクトを呼び出して、結果をテンプレートのなかに埋め込む処理が記述されることが多い。この場合、HTML生成オブジェクトは、拡張言語の記述にしたがって、機能オブジェクトを呼び出す。機能オブジェクトは、データ・レイヤのオブジェクトからデータをとってきてリターンする。これをHTML生成オブジェクトがテンプレートの中に埋め込んで、純粋な（拡張言語が含まれていない）HTML文書としてWebサーバに戻して表示する。

(3) データ・レイヤ

Webを導入しても特にデータ・レイヤには影響を与えない。図1.5.4では、データ・レイヤのデータ実体として、サーバのデータベースを隠蔽したオブジェクトと、メインフレームのアプリケーションをラッピングしたオブジェクトとを表記している。特に、現業務にアドオンする場合は、メインフレームとの連携が不可欠になると考えられるため、基本構造としてメインフレームを加えた形としている。

1．5．3　事例

(1) 全体構造

日立コマース・ソリューションが提供するソフトウェア群は、1.5.2項にて述べたWeb - CORBA連携アーキテクチャに準じている。ソフトウェアとしては、以下のものを持つ（図1.5-7）。

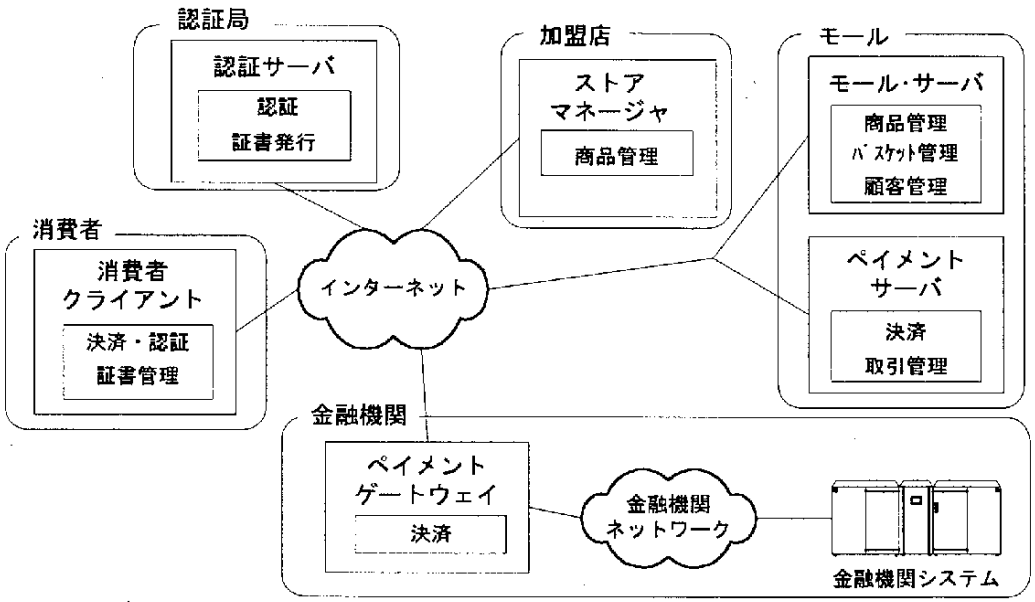


図1.5-7 日立コマース・ソリューションの構造

(a) モール・サーバ

モールをインタネット上のサーバに実現するためのサーバ・ソフトウェアである。

(b) ペイメント・サーバ

モールにおいて、通産省が制定している安全な電子商取引環境SECE（Secure Electronic Commerce Environment）に準拠したプロトコルで電子決済を実現するソフトウェアである。

(c) ペイメント・ゲートウェイ

金融機関において、SECEに準拠したプロトコルで電子決済を実現するソフトウェアである。

(d) 認証サーバ

認証局において、SECEに準拠した決済を行うための認証証書を作成するためのソフトウェアである。

(e) 消費者クライアント

消費者が操作して本システムを利用するために必要なクライアント･ソフトウェアである。

(f) ストア･マネージャ

モールに加盟している店舗（加盟店）から商品情報を登録･改定するため、および注文情報をモール･サーバから取り寄せ内部DBで管理するソフトウェアである。

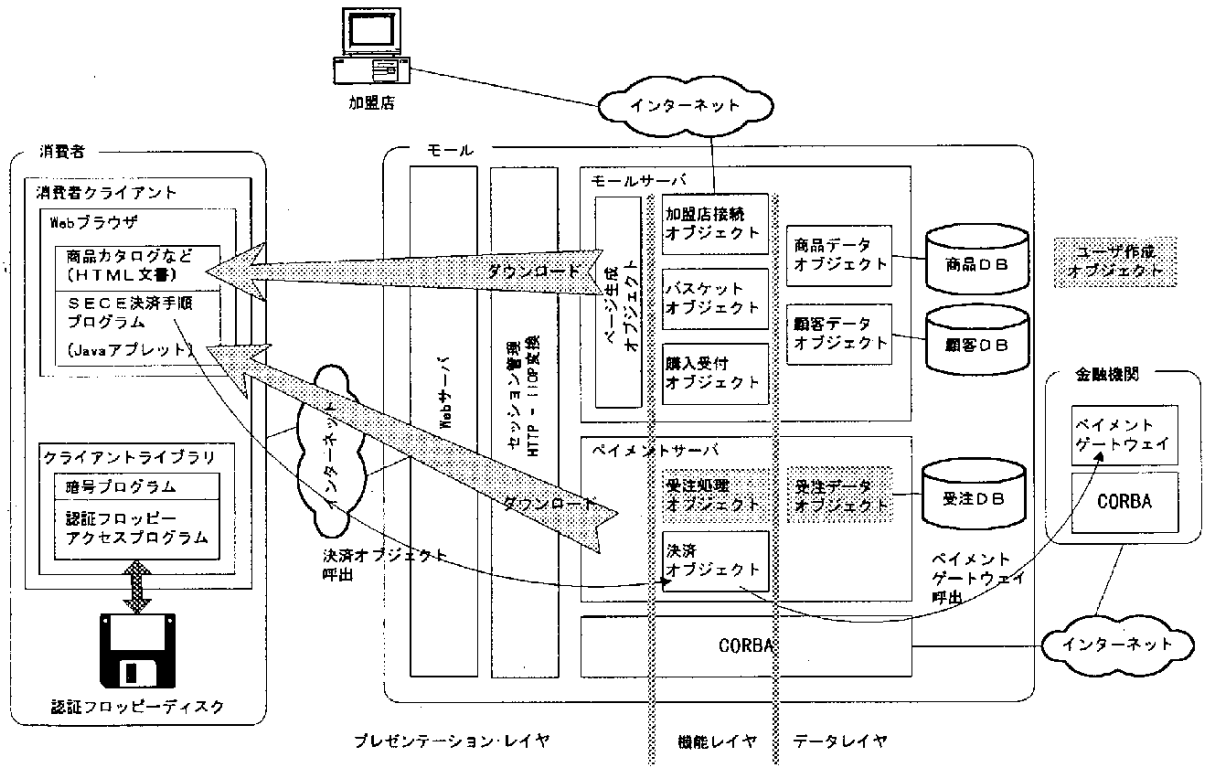


図1.5-8　消費者-モール間でのソフトウェア･アーキテクチャ

(2) 消費者 - モールの間の構造詳細

システム全体の中で、特に消費者とモールとの間について、Web-CORBAアーキテクチャにしたがって詳細に説明する（図1.5-8）。

(a) プレゼンテーション･レイヤ

(i) HTML文書

Webブラウザに、Webサーバ経由でダウンロードされたHTML文書が表示されて、商品カタログのGUIとして機能する。このHTML文書は、モールサーバのページ生成オブジェクト（1.5.2項で言うところの

HTML生成）によって生成されたもので、商品カタログのフォーマットを規定するテンプレートの中に、商品DBのなかの商品データ（商品名、価格、商品画像など）を埋め込んで生成される。

商品カタログ画面で「バスケットに入れる」という操作を行うと、バスケット画面が表示される。バスケットとはいわゆる「買い物かご」のことで、バスケット画面には、購入する候補の商品（「バスケットに入れる」という操作をして「購入」という操作をする前の商品）がリスト形式で表示される。バスケット画面上で購入する商品の個数を変更したり、消費税を含めた価格総額などを確認することができる。このバスケット画面もモールサーバのページ生成機構によって生成されたHTML文書がダウンロードされてGUIとして機能するものである。

バスケット画面で「購入」という操作を行うと、購入受付画面が表示される。ここには、購入する商品のリストと、商品の送り先を入力するフィールドをもつHTML文書が、ページ生成機構によって生成されて表示される。

(ii) 決済アプレット

購入受付画面には、「支払い」ボタンが貼り付けられており、このボタンを押すとSECEの決済手順を実現するJavaアプレットがペイメントサーバからダウンロードされ、実行を開始する。本決済アプレットは、HTTPに包んだIIOPでモールにあるペイメント･サーバの決済オブジェクトと通信してSECEの決済手順を実現する。 消費者クライアント、モール･サーバ、金融機関のペイメント･ゲートウェイの3者の間でSECEの決済手順にしたがって通信しあうことによって、消費者がモールで買い物をした代金を金融機関経由で支払うことができる。

決済手順をアプレットとして実現することにより、決済プロトコルのバージョンアップなどに伴うクライアントプログラムのバージョンアップは必要なくなるというメリットも得られる。

(iii) クライアント･ライブラリ

決済手順の中で、認証フロッピーディスクの中に格納されている暗号鍵を使って電文を暗号化する処理が必要になる。そのため、消費者クライアントの中には、2種類のクライアント/ライブラリが必要になる。

ひとつは、暗号用のライブラリである。これは、決済の安全性を実現するための核技術であるので、セキュリティの観点から動的にダウンロードすることが不適当なプログラムに相当する。言い換えれば、インターネットのどこからかダウンロードしてきた暗号プログラムを使って「安全な決済」と言えるのかということである。そこで、暗号プログラムはクライアント･ライブラリとして消費者クライアントにインストールする方式としている。

クライアント･ライブラリのもう一つは、認証フロッピーディスクをアクセスするプログラムである。

Javaでファイルアクセスを許すとウィルスプログラムがインターネットからダウンロードされて実行されてしまう可能性を許すので、Java言語の仕様としてファイルアクセスを許していない。したがって、認証フロッピーディスクをアクセスするプログラムはクライアント･ライブラリとして実現しなければならない。

(b) 機能レイヤ

商品カタログが表示されている状態では、ページ生成オブジェクトだけが動作しており、図1.5-8でいうところの機能レイヤにはアクセスされてこない。

「バスケットに入れる」という操作によって、バスケット画面に遷移すると、バスケットオブジェクトが動作する。この際、アクセスしてきている消費者をセッションID経由で特定し、それまでにバスケットにいれた商品とあわせてリスト表示する。再び商品カタログに戻ると、また機能レイヤからは離れ、ページ生成オブジェクトだけをアクセスする形態となる。

同様に、「購入」という操作によって、購入受付画面に遷移すると、購入受付オブジェクトが動作する。ここでも、セッションIDを使って消費者を特定し、バスケットの中身を購入商品リストとして表示する。消費者が購入受付画面に入力したお送り先などを受け取り、決済後に利用できるようにDBに保存する。

モールに置かれるペイメント･サーバの決済オブジェクトは、決済アプレットから見て機能オブジェクトとして機能し、金融機関のペイメント･ゲートウェイとやりとりしながら、SECEで規定された決済手順を実現する。決済終了後には、決済オブジェクトから受注処理オブジェクトに対して呼出がかかる。この受注処理オブジェクトは、受注に関わるさまざまな処理を行う機能レイヤのオブジェクトである。

その仕様は商品や取り扱う企業によって大きく異なる。たとえば、受注を受けた後で生産するような場合もあれば、生花のように配達が重要な役割を持つ場合もありさまざまである。そこで、この受注処理オブジェクトはユーザ作成オブジェクトとしておき、決済オブジェクトから呼び出すための出口インタフェースが規定されている。

(c) データ･レイヤ

データ･レイヤのオブジェクトとしては、RDBなどに恒久的に記憶されるDBテーブルがオブジェクトとして見えるものを用意する。商品DBを表現する商品データオブジェクト、顧客DBを表現する顧客データオブジェクトなどがあげられる。たとえば、商品DBのなかには、商品間の関連なども表現されており、これをたどるAPIが商品データオブジェクトに用意されている。

1．5．4　まとめ

以上、Webブラウザをユーザインタフェースとし、Webサーバ経由でオブジェクト指向にもとづいた3階層モデルの業務プログラムを実現するためのソフトウェア･アーキテクチャを述べ、その実例を示した。

Webブラウザがクライアント PCに標準装備される今日においては、HTML文書をベースとするユーザインタフェースが一種の標準インタフェースになりうる状況にある。ここで述べたアーキテクチャは、当面はイントラネットにおける通常の業務プログラムでのニーズが強いと考えられるが、インターネットの応用が現在のようにデータ検索中心のものからさらに一般の機能を持つアプリケーションに拡大されていくにしたがって、より広い分野に適用されていくと考えている。

【参考文献】

1) Berners-Lee, T. et al.: "The World-Wide Web", Comm. ACM, Vol.37, No.8 (Aug 1994).

2) Java言語環境 --- A White Paper, 1995年7月, 日本サン･マイクロシステムズ.

3) Microsoft ActiveX Platform Backgrounder, 1996年6月, マイクロソフト株式会社.

4) The Common Object Request Broker: Architecture and Specification, Revision 2.0, July 1995, Object Management Group.

5) Box, D: "Introducing Distributed COM and the New OLE Features in Windows NT 4.0", Microsoft Systems Journal (May 1996).

6) Donovan, J, J: Business Re-engineering with Information Technology, 1994, Prentice Hall.

1．6　Web-データベース連携

1．6．1　はじめに

WWWが登場したことにより、WWWブラウザをクライアントとするような新しい形態のデータベース・アプリケーションが盛んに構築されるようになってきた。このようなWWW型データベースアプリケーションが注目されるようになった背景には、以下のような理由があると思われる。

(1) WWWでは、HTMLを用いてハイパーテキスト形式の文書を作成する必要がある。ところが、大量のコンテンツをハイパーテキストのリンク構造だけで管理するのは困難である。また、リンクを辿る方法で必要な情報ページを検索するのでは、大量の情報を効率的に検索することが困難である。もし、データベース管理システムとWWWが連携できれば、データベース中に情報ページを構成するための基本的なデータを格納しておき、必要な時点で適切な検索条件でデータベースを検索し、情報ページとして表示できるように検索結果からHTMLを生成することができる。したがって、大量のコンテンツを効率的に管理できるだけでなく検索することもできる。また、データベース内のデータを追加・変更するだけで、実際の情報ページは、必要な時点でデータベースから生成されるので、情報ページを直接修正するよりも効率的である。

(2) もし、データベース管理システムとWWWが連携できれば、対象となるコンテンツを新たにHTMLで作らなくても、すでに企業がデータベース上に蓄積した情報をそのままコンテンツとして利用することができる。したがって、情報共有型のイントラネットを簡単に構築できる。

(3) 従来のクライアント／サーバ型の情報システムに対して、WWWブラウザをデータベース・アプリケーションのクライアントとして利用できれば、システムの構築費用を大幅に削減できる可能性がある。また、WWWブラウザが動作すればクライアント端末の機種の差が問題にならないので、従来のクライアントサーバ・システムよりもはるかに広い範囲でデータベース・アプリケーションを利用できるようになる。さらに、WWWブラウザにより、ユーザインタフェースが自然に統一されるので、複数の異なるデータベースアプリケーションを利用する場合でも、エンドユーザにとっては一つのアプリケーションを操作するのと変わらない。

このように、WWWとデータベース連携技術は、インタネット上のWWW利用環境だけではなく、企業内のWWWを中心とするイントラネット利用環境でも重要な構成要素として着実に成長している。WWW－データベース連携技術を用いた企業情報システムのイメージを図1.6-1に示す。

以下では、WWWとデータベース連携について、基本となる技術的な特徴と、製品動向、適用事例ならびに今後の課題について述べる。

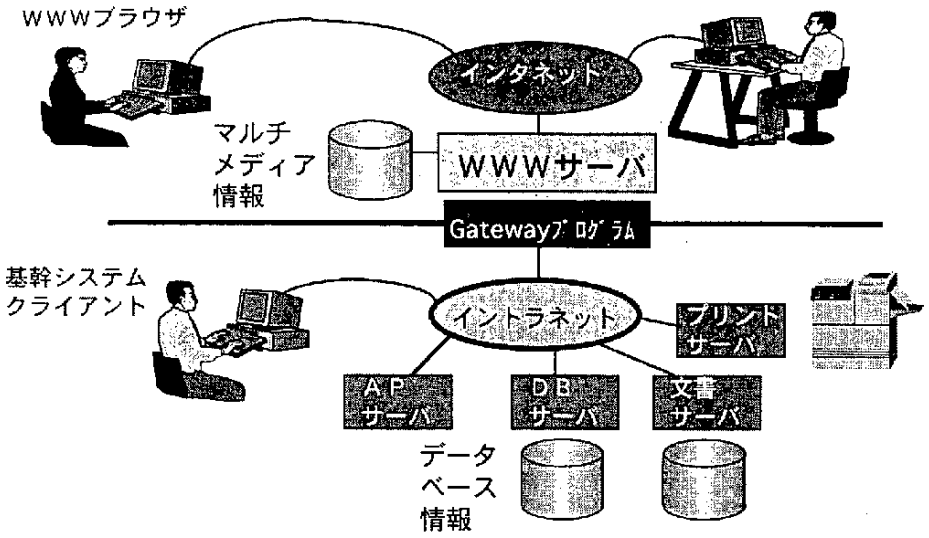


図1.6-1 WWWデータベース連携を用いた情報システム

1．6．2 WWWーデータベース連携の要素技術

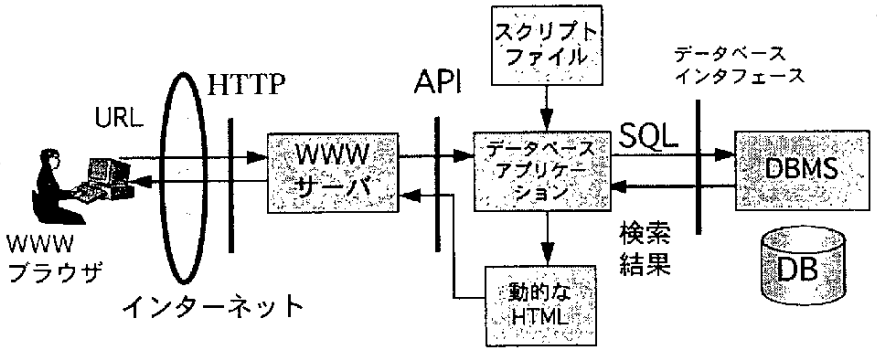
(1) システム構成

WWWーデータベース連携システムの基本的な考え方は、非常に簡単である。データベース・アプリケーションの検索結果をHTML化しておけば、それをWWWで参照できるということである。したがって、従来のデータベースアプリケーションとの違いは、極端に言えば、検索結果が動的にHTML化されWWWブラウザにHTTPで返却されることだけである。

このようなWWWとデータベースを連携するために必要となる構成要素は、WWWブラウザ、データベース・アプリケーションを起動できるWWWサーバ、データベース・アプリケーション、データベース・アプリケーションによる検索結果からHTMLを生成するゲイトウェイ、そしてデータベースである。WWWとデータベース連携システムの基本的なシステム構成を図1.6-2に示す。

WWWーデータベース連携製品の基本的な処理の流れは、次のようになる。まず、WWWブラウザからデータベース・アプリケーションを記述してあるスクリプト・ファイルのURLを指定する。次に、スクリプト・ファイルからデータベース操作のためのSQL文が生成され、データベース管理システムに送信される。最後に、検索結果からHTMLが動的に生成されWWWブラウザに返却される。

このように、基本的な処理の流れは簡単であるが、その実現方式は製品ごとに異なっており、多様な製品が発表されている。以下では、アプリケーション・インタフェース、アプリケーション・ロジックの記述方式、セッション管理方式、データベース検索方式などについて述べる。



WWWサーバとデータベースアプリケーションのインタフェースには、
CGIやWWWサーバベンダが提供するAPIなどがある。

図1.6-2　WWWデータベース連携システムの基本構成

(2)アプリケーション・インタフェース

WWWサーバからデータベース・アプリケーションを起動するインタフェースには、CGI（Common Gateway Interface）方式が最も汎用的で基本的である。しかし、データベースと連携させる場合、CGIではブラウザから要求がある度に、プロセスを起動・終了させるため、環境変数の保存・回復、プロセス資源の確保・開放、データベース資源のオープン・クローズなどが必要となり、性能面で問題があった。

このため、WWWサーバ・ベンダが独自にアプリケーション・インタフェースを独自に提供するようになった。このような各社独自のAPIには、Netscape Communications社のNetscape API (NSAPI) やMicrosoft社のInternet Server API (ISAPI)、Oracle社のWeb Request Broker(WRB)などがある。

(3)アプリケーションロジックの記述方式

WWW－データベース連携アプリケーションの作成では、HTML文、SQL文などのデータベース操作の記述、制御構文を記述する必要がある。したがって、これらをどのようにして記述するかに応じて製品の形態が異なる。このようなアプリケーション・ロジックの記述方式には、テンプレート方式、関数ライブラリ方式、ビジュアル・プログラミング方式がある。

(a) テンプレート方式

HTML文、SQL文、制御構文を、一つのスクリプト言語でまとめて記述する方式である。テンプレート方式の製品事例として、Microsoft社のIDC、 O'Reilly社のCold Fusion 、 Sybase社のWeb.sql、 NTTのWebBASEなどがある。テンプレート方式の利点は、一つのスクリプト言語だけでWWW－データベース連携アプリケーションを作成できるので、従来のプログラミング言語を用いるよりははるかに生産性が高いことである。しかし、テンプレートで利用できる構文や変数に制約がある場合、必ずしも柔軟にアプリケー

ションロジックを記述できないという問題がある。このような制約の例として、IDCでは、一つのテンプレートでは複数のSQL文を記述できないこと、ColodFusionでは、複数のSQL文を記述できるがIF ELSE構文を使えないなどがある。ただし、WebBASEでは、これらの構文上の制約はない。

(b) 関数ライブラリ方式

HTML文やデータベース操作文を既存のプログラミング言語の関数として用意することにより、従来のプログラミングの延長でWWW－データベース連携アプリケーションを作成する方式である。関数ライブラリ方式の製品事例として、Oracle社のWebServerやNetscape Communications社のLiveWire Professionalなどがある。WebServerでは、Oracleのストアドプロシージャの記述言語であるPL/SQLの関数ライブラリとして、HTMLのタグを生成するHTF関数とHTMLの結果を出力するHTP関数を提供する。LiveWire Professionalでは、Java scriptの外部関数としてOpen DataBase Connectivity(ODBC)、 Informix、 Sybase、 Oracleなどのデータベース操作インタフェースを提供する。関数ライブラリ方式の利点は、プログラミング言語をベースにしているので構文上の制約が少なく柔軟性が高いことである。しかし、アプリケーション開発の生産性の面では、HTMLそのものではなくHTMLを生成する関数を学習する必要があること、ベースとするプログラミング言語の学習が必要となるので、テンプレート方式に比較して高いとは言えない。とくに、WebServerの場合、ストアドプロシージャをベースにしているのでJava程の一般性に欠けるし、データベース管理システムもOracleだけに限定されるという問題もある。

(c) ビジュアル・プログラミング方式

ビジュアルにアイコンや処理フローを編集したり、図形エディタ上でアイコンやフローのプロパティを設定することにより、HTML文、SQL文、制御構文からなるアプリケーションロジックを定義する方式である。特定のアプリケーションに限定するものとテンプレートを生成することにより一般的なアプリケーションを作成できる製品とがある。前者の事例としてDEC社のWebQueen/MML、後者の事例としてNTT TE東京のWebGLOBEがある。WebGLOBEでは、GUIで編集した結果からWebBASEのスクリプト言語を生成するようになっており、汎用性が高い。ビジュアル・プログラミング方式の最大の利点は、プログラミング経験のないエンドユーザでも簡単にWWW－データベース連携アプリケーションを作成できることである。

(4) セッション管理方式

検索結果に応じた絞り込み検索のように異なる情報ページ間で情報の引継が必要となるセッション管理をCGIで実現する場合、CGIではブラウザから要求がある度に、プロセスを起動・終了させるため、アプリケーション側でこのようなセッション管理を個別に開発する必要があった。このため、セッション管理を持つようなWWW－データベース連携アプリケーションを効率的に開発するための機能が提供されている。

このようなセッション管理機能の実現では、最初にWWWブラウザから検索要求があったときにアプリケーション・プロセスを常駐させておき、同じWWWブラウザからの2回目以降の検索要求については、このアプリケーション・プロセスがWWWブラウザと通信する方式が一般的である。この場合、WWWブラウザが通信する相手のアプリケーション・プロセスを特定するための識別子を検索条件や検索結果としてのHTMLに付加して、WWWブラウザとアプリケーション・プロセスの間で持ち回る必要がある。図1.6-3に、NTTソフトウェア研究所で開発したWebBASEにおけるセッション管理方式を示す。

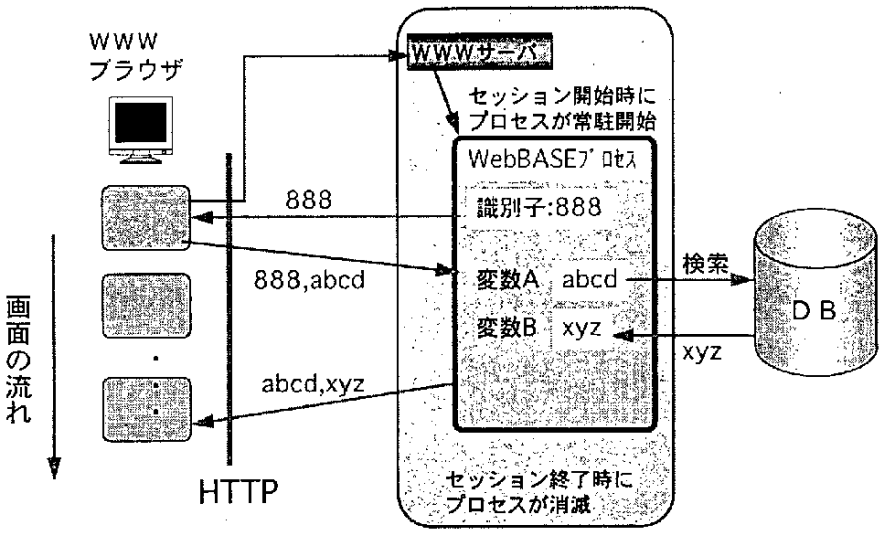


図1.6-3　WebBASEにおけるセッション管理方式の例

(5) データベース検索方式

データベース検索方式には、検索された時点で検索結果からその都度、HTMLを生成するリアルタイム方式と、予めデータベースを検索した結果に基づいてHTMLを生成してWWWサーバに登録しておく一括方式がある。また、これらの中間的な方式として、一度検索して生成したHTMLを保存しておき再利用するキャッシュ方式がある。

(a) リアルタイム方式

データベースの内容が常に変化し続ける更新頻度の高いデータベースでは、リアルタイム方式が適している。そうしないと、HTML化された情報とデータベースに格納された情報の間で食い違いが発生するからである。リアルタイム方式では、検索要求が発生する度にHTMLを生成するので、検索結果が大量に発生する場合、応答性能を低下させる可能性がある。この問題を避けるには、検索を階層化したり、検索結果の個数に応じて絞り込む、一定数づつ結果を提示するなどの段階的な検索ロジックが必要になる。

(b) 一括方式

データベースの更新頻度が高くない場合や、データベース上の情報とWWW上で閲覧する情報の間に多少の時間的なずれがあっても構わない場合には、一括方式を用いることにより簡単にシステムを構築できる。また、データベースをWWWから直接起動しないこと、動的にHTMLを生成しないことから、検索要求時の応答が速いという特徴がある。しかし、データベースの規模が大きくなると、大量のHTMLが更新の度に書き換えられるので、必ずしも参照されないページまでHTML化されてしまうのでディスク容量が無駄になるだけでなく、更新のためのCPU稼働コストも無駄になるという問題がある。

(c) キャッシュ方式

データベースの定型的な検索頻度が高く、データベースの規模も大きい場合、キャッシュ方式が有効である。例えば、ある時間内に同じ検索要求がN回あると仮定すると、もし、最初の1回の検索要求時に生成したHTMLを保存しておき、他のN－1回の検索要求で再利用することができれば、データベース検索時間とHTML生成に必要となる時間を残りの検索要求については削減できる。したがって、キャッシュ方式では、リアルタイム性を損なうことなく、見かけの応答性能を大幅に向上できる可能性がある。

1．6．3　WWW－データベース連携ツールの分類

(1) 製品分類の観点

これまで述べてきたように、WWW－データベース連携ツールの特徴には、アプリケーション・アーキテクチャ、提供形態、アプリケーション記述方式などがある。以下では、これらの観点から製品事例を分類する。

(a) アーキテクチャには、WWWサーバの中に連携機構を内蔵する一体型と、WWWサーバの外部に分離させる独立型がある。独立型には、CGI方式とWWWサーバベンダ固有のAPIを用いる方式の2つがある。

(b) アプリケーション記述方式には、テンプレート方式、関数ライブラリ方式、ビジュアルプログラミング方式などがある。

(c) 提供形態としては、WWWサーバとバンドルする統合製品と連携製品だけで販売する単独製品がある。埋め込み型の場合は、必然的に統合製品になる。アーキテクチャとしては独立型の場合でもOracle社のWebServerのように統合製品として商品化される場合がある。

WWW－データベース連携製品をアーキテクチャとアプリケーション記述方式の観点から分類した結果を表1.6-1に示す。以下では、これらの製品の概要について説明する。

表1.6-1　WWWデータベース連携製品の分類

	一体型	独立型
テンプレート	WebBASE InterServ InfoProvider	IDC Web.sql Cold Fusion
関数ライブラリ	――――	WebServer LiveWire
ビジュアル プログラミング	WebGLOBE WebQueen/MML	――――

(2) 製品事例

(a) IDC

Microsoft社のInternet Database Connector (IDC) は、アーキテクチャ：独立型、アプリケーション記述：テンプレート型、提供形態：統合型のWWW－データベース連携製品である。

IDCは、IISからのODBCインタフェースによるデータベースを操作するためのゲイトウェイ機構である。IDCにより、ODBCインタフェースが利用できるため、Microsoft社のSQL Sever以外のデータベース管理システムも操作できる。

IDCではIDC制御ファイルとHTXテンプレートを用いる。IDC制御ファイルでは、ODBCへの接続情報、SQLコマンド、テンプレートファイル名を記述する。テンプレート・ファイルでは、HTMLとIDCコマンドを組み合わせて、データベースから検索されたレコードに対するアプリケーション・ロジックをIF ELSE 文や制御ファイルへ渡す変数などを用いて記述できる。

IDC制御ファイルのURLを指定すると、IDCのDLLプログラムが起動され、SQLコマンドへのパラメータなどが渡される。次いで、このプログラムがデータベースとコネクションを張り、SQLコマンドを実行する。最後に、テンプレートファイルに基づいて、検索結果を編集してHTMLを生成する。

IDCは、ISAPIに基づいて設計されており、BYTE誌のWindowsNT上での評価報告によれば、WebServerやCold Fusionなどの製品に較べて応答速度が約2倍程度速いようである。

(b) WebServer

Oracle社のWebServerは、アーキテクチャ：独立型、アプリケーション記述：関数ライブラリ型、提供形

態：統合型のWWW－データベース連携製品である。

Oracle社のWebServer では、SpyGlass社の製品をライセンスしたWWWサーバであるWeb Listener にOracle7と連携するためのCGIスクリプトであるOracle Web Agentを組み合わせている。Oracle Web AgentからSQL Netインタフェースで Oracle7上のストアドプロシージャを呼び出す。データベース操作、アプリケーションロジック、HTML定義などはすべて、ストアドプロシージャで記述する。ストアドプロシージャでは、HTMLのタグを生成するHTF関数、HTMLの結果を出力するHTP関数を用いて、HTMLを作成するためのPL/SQLを記述する。

今後提供される予定のWebServer 2.0では、PL/SQLでOracleデータベースと接続するだけでなく、WRBというAPIを提供する。WRB　APIでは、サーバ側でJava、 C、 C++などのプログラミング言語で記述された外部プログラムを実行できるようになる。

(c) LiveWire

Netscape Communications社のLiveWireは、アーキテクチャ：独立型、アプリケーション記述：関数ライブラリ型、提供形態：統合型のWWW－データベース連携製品である。

LiveWireは、運用ツールSite Manager、JavaScriptの中間コードを生成するJavaScriptCompiler、WWWブラウザからデータベースにアクセスできるDatabase Connectivity Libralyからなる。Database Connectivity Libralyでは、ODBCとともにInformix、 Sybase、 Oracleなどのデータベースインタフェースを Javaの関数ライブラリで提供する。とくに、Suitespotに含まれるLiveWire Professionalでは、データベース連携アプリケーションを容易化するため、Informixを標準装備している。

(d) WebSite Professional

O'Reilly社のWebSite Professional は、アーキテクチャ：独立型、アプリケーション記述：テンプレート型、提供形態：統合型のWWW－データベース連携製品である。

O'Reilly社のWebSite Professional では、Allaire社のCold Fusion を、自社のWWWサーバであるWebSite Standardにバンドルした製品である。Cold Fusionは、単独に購入することもでき、そのときは、他のWWWサーバとの組み合わせでデータベース連携アプリケーションを作成することができる。

Cold Fusionでは、Microsoft社のIDCと同様に、ODBCインタフェースを用いてSQLコマンドをデータベースに送る。IDCとの違いは、HTML文、SQL文、制御構文を一つのテンプレートファイルで記述できること、IDCのテンプレートでは複数のSQL文を記述できることなどである。また、Cold Fusionでは、CGIを用いている。

(e) Web.sql

Sybase社のWeb.sqlは、アーキテクチャ：独立型、アプリケーション記述：テンプレート型、提供形態：

単独型のWWW－データベース連携製品である。

Web.sqlは、Perlインタプリタ部、データベース通信部、HTTP通信部から構成されている。Web.sqlでは、HTML文、SQL文、Perl文を一つのHyper Text Sybase (HTS) ファイルで記述できる。一つのHTSファイルの中に複数の処理を記述できる。HTML文の中に埋め込まれるSQL文やPerl文は、タグで識別される。この構文は、<syb type= perl> 埋め込み文並び </syb> というようになっている。SQL文については、この埋め込み部分の中で関数 ws_sql (データベース名、SQL文) を用いて記述する。

Web.sqlは、Sybaseのデータベース・エンジンに対してクライアント側のアプリケーション・プログラムとして動作する。したがって、Web.sqlとSybaseとのインタフェースは、Sybase社のデータベースアクセスインタフェースであるCT-Libインタフェースとなる。CT-Libを用いることにより、HTSファイルから、データベース側のストアドプロシージャを呼ぶこともできる。実際にアクセスするデータベースの定義情報を知るためにWeb.sqlでは、DBアクセスマップというファイルが必要である。

Web.sqlでは、CGIだけではなく、Netscape Communications社のWWWサーバが提供するNSAPIにも対応している。ただし、Netscape Communications社以外のWWWサーバとは、CGIで連携することになる。

(f) WebBASE

NTTソフトウェア研究所のWebBASEは、アーキテクチャ：独立型、アプリケーション記述：テンプレート型、提供形態：単独型のWWW－データベース連携製品である。

WebBASEは、NTTが開発したデータベース・アプリケーション開発用のミドルウェアVGUIDEをベースにして開発されている。WebBASEは、WebBASE Scriptインタプリタ部、VGUIDE通信部、HTTP通信部から構成されている。 WebBASE のシステム構成を図1.6-4に示す。

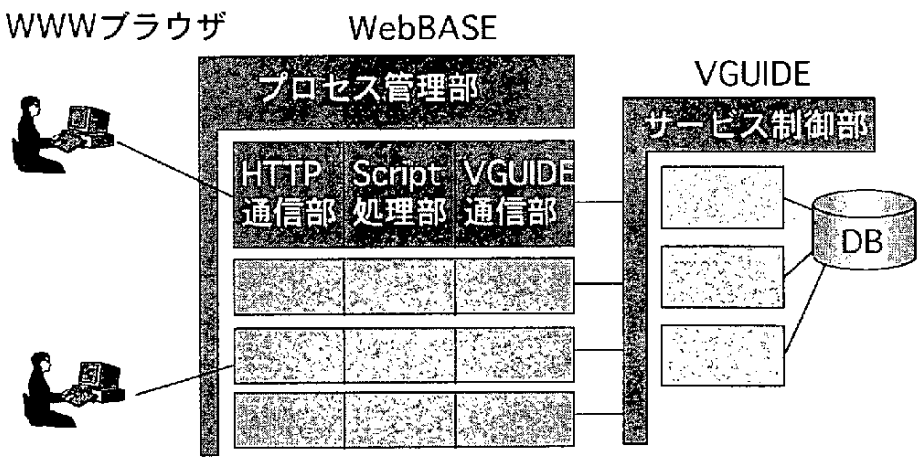


図1.6-4　WebBASEのシステム構成

この点では、Sybase社のWeb.sqlとシステム構成の点では、良く似ている。テンプレートに関する相違点は、HTML文にSQL文を混在させて記述する制御構文として、WebBASEでは、Visual BASICを基本にしていること。また、HTML文と他の文とを区別するのにWEbBASEでは、タグではなく、行の先頭のトークンに指定されたピリオドを用いたことなどである。また、WebBASEでは、VGUIDEにより、Informix、Sybase、 Oracleなどのデータベースと通信する。これに対してWeb.sqlでは、CT-Libを用いてSybaseと通信する。WebBASE Scriptでは、IF ELSE、LOOP、WHILEなどの制御構文があり、単純なHTML文に較べて少ない記述量で情報ページを作成できる。また、WebBASEでは、外部プログラムをDLLで呼び出すことができるのでデータベースだけでなく全文検索エンジンなどとの連携も容易である。

WebBASEの特徴は、WebBASEScriptによる簡潔なアプリケーション・ロジックの記述、複数ページに跨るセッション管理、WebBASEプロセスの常駐制御による高速な応答性能などである。

(g) WebQueen/MML

DEC社のWebQueen/MMLは、アーキテクチャ：独立型、アプリケーション記述：ビジュアルプログラミング型、提供形態：統合型のWWW－データベース連携製品である。

DEC社のWebQueen/MMLでは、一定の項目数のレコードからなるデータベースの検索／更新アプリケーションを対象としている。これにより、HTMLやSQLプログラミング経験のないエンドユーザでも、テーブル構成やページレイアウトをGUI上で簡単に定義するだけで、WWW－データベース連携アプリケーションを構築できるようにしている。MML自体は、WWWからCGIで起動されるゲイトウェイ・アプリケーションである。MMLが内部に持つテーブル構成やページレイアウト情報に基づいてデータベースを検索し、HTMLを生成する。

(h) WebGLOBE

NTT TE東京のWebGLOBEは、アーキテクチャ：埋め込み型、アプリケーション記述：ビジュアルプログラミング型、提供形態：統合型のWWW－データベース連携製品である。

NTT TE東京のWebGLOBEは、HTQLエディタとHTQLサーバからなる。HTQLエディタでは、GUIで編集した結果からWebBASEのスクリプト言語を生成する。HTQLサーバでは、このWebBASEのスクリプト言語に基づいてODBCによるデータベース連携ができ、WWWサーバ処理とスクリプトの実行管理処理を行う。HTQLエディタでは、検索ページやアプリケーション・ロジックを作成する。まず、表や文字、ボタン、入力フィールドなどのオブジェクトを画面に配置する。画面上のオブジェクトごとにアプリケーション・ロジックを設定できる。アプリケーション・ロジックの定義では、フローチャートを用いて処理コマンド間の制御フローを記述する。処理コマンドには、検索や条件分岐、検索結果表示などがあり、メニュから選択できる。また、処理コマンドのプロパティ画面により、処理コマンドが対象とするデータベースのテーブルや入力データを記述する。

1．6．4　WWWーデータベース連携アプリケーションの適用形態

(1)適用アーキテクチャ

WWWーデータベース連携システムのエンドユーザは、従来のクライアントサーバシステムのエンドユーザよりも拡大している。すなわち、基幹系エンドユーザ、イントラネット・エンドユーザ、インタネット・エンドユーザである。WWWーデータベース連携アプリケーションの適用アーキテクチャを図1.6-5に示す。

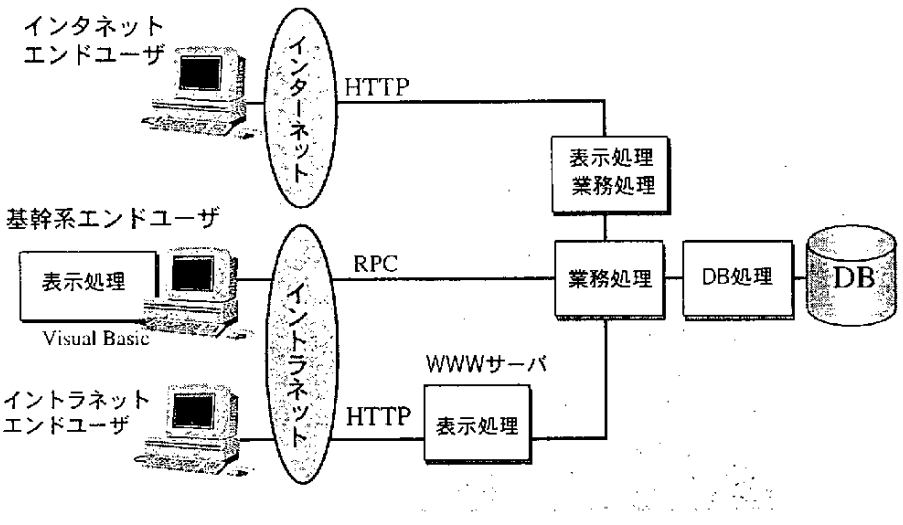


図1.6-5　WWWーデータベース・アプリケーションの適用アーキテクチャの例

(2)適用アプリケーションの分類

イントラネット上のアプリケーションを、コミュニケーションの範囲という視点と、コミュニケーションの形態という視点によって、分類してみよう。まず、コミュニケーションの範囲としては、イントラネットを組織内に閉じたコミュニケーション手段として利用するアプリケーションと、インタネットとイントラネットを接続して、組織の外部とのコミュニケーション手段として利用するアプリケーションがある。また、コミュニケーション形態としては、情報の共有手段としてイントラネットを利用するアプリケーションと協調作業手段としてイントラネットを利用するアプリケーションとがある。

WebBASEの適用事例をこの分類にあてはめると、表1.6-2のようになる。

表1.6-2 WWWデータベース・アプリケーションの事例

	組織内コミュニケーション	組織外とのコミュニケーション
情報共有	技術情報検索 経営支援 問合せ支援	NTTディレクトリ 物質スペクトル検索 ニュースオンデマンド フリーダイアル検索
コラボレーション	国際営業支援 ソフト開発支援 ソフト部品再利用支援 設備管理支援	ISDNオンライン申込み 宿泊施設案内 ゴルフ場予約

(3) WWW－データベース連携アプリケーションの適用効果

平成8年度当初から、急速にWWW－データベース連携アプリケーション開発が進展してきた。この背景には、インタネットやイントラネットが追い風になっていることは間違いない。ただ、それだけではなく、これまでクライアント／サーバ・システムの中で蓄積されてきたデータベース技術がWWWと組み合わされることにより大きく伸びたということができる。この流れは、当分、継続するだけではなく、むしろより大きな流れとなって発展していくと思われる。この理由は、この1年間で、WWW－データベース連携アプリケーションが単に見栄えがいいだけではなくて実際に役に立つということが実証されてきたからに他ならない。

ここでは、WWW－データベース連携アプリケーションの開発事例に基づいてその適用効果を分析する。

(a) エンドユーザ部門が直接開発できるようになった。

従来は、情報システム部門がエンドユーザ部門の要求を聞いて仕様を作成するところから始まった。しかし、WWWを用いたアプリケーション開発の場合は、プロトタイピングが簡単にできるだけではなく、エンドユーザ自身がHTMLを使うようになっており、情報システム開発の敷居が大幅に低くなった。このため、エンドユーザが自分でプロトタイプを作り、ユーザインタフェースについては、自ら要求を提示できるようになった。したがって、情報システム部門を介さずに、ソフトウェアハウスを使う場合が増加している。WWWが登場するまでは、クライアント／サーバ・システムであっても、このようなことは考えられなかった。

(b) アプリケーションの走行環境がWWWによって統合されたことにより、インタネットから一般のエンドユーザが直接利用できるようになった。

例えば、ISDNオンライン申し込みシステムでは、INS回線申し込みの約２０％が、インタネットから申し込まれている。インタネットやWWWがもしなければ、専用のクライアント・アプリケーションを一般のエンドユーザに配布することが最大の問題になっていたはずである。この事例は、データベース・アプリケーションの適用範囲を拡大するという点でWWWが大きく貢献していることを示している。

(c) チームによる協調作業を支援するアプリケーション開発へ適用範囲が拡大した。

インタネットやWWWにより、広域で利用できるデータベース・アプリケーションの可能性が広がった。例えば、会議文書をデータベース化しておき、電子会議で自由に検索しながら議論するなどグループウェアとの連携、国際的な営業情報の共有を支援するイントラネット、伝票決裁を行う電子稟議システムなど、これまでの狭い範囲でのデータベース・アプリケーションを越えるシステムの開発が進展している。

１．６．５　WWW－データベース連携の課題

WWW－データベース連携アプリケーションを構築していく上では、システムを効率よく短期間で開発すること、可能な限りオープンな環境でシステムを構築できること、オープンなネットワーク上で可能な限り安全で効率よく情報を流通できることが重要になる。

(1) WWW－データベース連携アプリケーション開発法

イントラネットのような時間が重要な因子となるシステム開発では、短期間にシステムを柔軟に変更していく必要がある。このため、従来のような、予め厳密に仕様を定義するライフサイクル型の開発方法論をそのまま、イントラネット開発に適用することが困難となる。したがって、WWWを前提とする新しいソフトウェア開発方法論が必要となる。このような迅速なアプリケーション開発を可能とするためには、WWW－データベース連携アプリケーションのアーキテクチャをいままで以上にコンポーネントを用いて標準化することと、広域で利用されるためアプリケーションの性能を最適化すること、WWWアプリケーション開発を可能な限りビジュアル化することなどが重要になる。

(a) コンポーネント化

例えば、ドメイン分析技術に基づいて、WWW－データベース連携アプリケーションをコンポーネントとして部品化しておき、類似するドメイン間で、このコンポーネントをカスタマイズして利用することにより短期間でWWW－データベース連携アプリケーションの開発を可能とするアプリケーション・アーキテクチャが考えられる。このようなアーキテクチャでは、ドメイン共通コンポーネント、ドメイン固有コンポーネント、アプリケーション個別コンポーネントにコンポーネント自身が階層化されていくと思われる。

(b) 最適化機能

WWW－データベース連携アプリケーションでは、インタネットと接続される場合には、予め、広域に散在する多数の端末から同時にアクセスされる可能性を考慮しておく必要がある。この場合、情報の負荷分散技術や情報ページの構成技術が重要になる。データベースの情報資源を効率的に流通させるための情報の負荷分散技術として、一度検索した情報をサーバ上に蓄積しておき、端末側からの情報の再要求に際して、冗長なデータベース検索を抑止する情報のキャッシング技術が必要になると考えられる。情報ページの構成技術としては、一度に大量の情報転送を必要とするような情報ページを作成しないことが基本となる。例えば、データベースから検索した情報をすべて端末側に転送するのではなく、情報ページへの転送件数の上限値を定めておき、継続要求ごとに、情報を転送する逐次的な方式により、通信トラヒックの増大を防ぐことができる。現状では、これらの性能向上対策については、アプリケーション・ロジックで実現することになる。広域ネットワーク上での利用を前提とすると、このような性能向上を実現するための最適化機能を具備したWWW－データベース連携製品の登場が望まれる。

(c) ビジュアルプログラミング

従来から、クライアント／サーバ・システム開発をビジュアル化する目的で開発されていた各種の製品が、WWWに対応するようになってきている。このような製品の例として、Borland International 社のDelPhi、Centura software 社のCENTURA、Power Soft 社のPowerBuilder、日立製作所のAPPGALLERY、富士通のIntelligentPadなどがある。これらのビジュアル開発ツールは、これまで情報システム開発に使われてきた実績があり、イントラネットへの適用が進むと思われる。また、先に紹介したWebGLOVEのように、最初からWWWのために開発されたツールも今後開発が進んでいくと思われる。例えば、Micrsoft社では、ActiveX Control Padを開発している。同様に、Java開発用のビジュアルツールの開発も進んでいる。

(2) オープンシステム構成法

サーバ側のソフトウェアとして、リレーショナルデータベースだけでなく、TPモニタならびに、CORBAやオブジェクト指向データベース、グループウェア、セキュリティ製品などさまざまなパッケージ製品との連携がこれから重要な課題となるだろう。WWW－データベース連携機構をこれらのパッケージ製品と連携させていく場合、WRBなどで提供されるようにできるだけオープンに外部コマンドを起動できるようなインタフェースが必要である。このような、外部プログラムとの接続インタフェースがあればデータベースだけでなく、全文検索エンジンなどともWWWを容易に連携できるのアプリケーションの対象が一気に拡大できる。ただし、さまざまな製品を組み合わせて一つのアプリケーションを構成した場合、それぞれの製品間でバージョンの食い違いや相性の問題が発生する。また、バグが発生した場合、アプリケーションを構成する製品数が多くなると不具合箇所の特定が困難になることが予想される。したがっ

て、オープンなシステムを構成する上で適切なシステム評価技術の確立が望まれる。

(3)ネットワーク構成法

イントラネット上のアプリケーションが走行するネットワークの設計では、セキュリティの水準や流通情報の制御方式に応じたサーバ構成が重要となる。セキュリティ技術では、暗号化技術やユーザ認証技術にはさまざまな方式があり、製品の選択や組み合わせ技術が必要となる。また、サーバ構成では、WWWとデータベースを独立に配置できるWWW－データベース連携製品の場合、WWWサーバをファイアウォール上に配備し、データベースを搭載するサーバをファイアウォールとは別に配備することにより、セキュリティを向上できる。この場合、ファイアウォール上のWWW－データベース連携機構からのデータベースへのアクセスについては、データベース側で操作を制限しておくことにより不当なデータベース操作を抑止できる。しかし、完全なセキュリティ方式はまだないのが現状であり、機密性の高い情報は、一般のインタネットを介してIP接続できるネットワークとは接続しないことや、不正ログがないか常にチェックするログ監視体制を確立することも重要となる。

情報の流通制御技術としては、情報の公開範囲に応じて適切にサーバを分散配置する必要がある。例えば、組織の外部に公開する情報、組織内で公開する情報、管理者以上にしか公開しない情報、あるいは、ある部の中だけでしか公開しない情報などとでは、情報の本質的な公開基準が違うと考えられる。このような互いに公開基準の異なる情報を同一のサーバに配置するのではなく、公開基準の同じ情報ごとにサーバを独立させることが、十分なセキュリティを確保する上で重要である。

したがって、ネットワーク上に配備された情報資源やサーバ設備を管理する機能や、セキュリティや情報資源の公開基準などを一貫して管理できる機能がWWW－データベース連携製品にとって重要な機能になっていくと思われる。

(4)WWW－データベース連携アプリケーションの発展

これまでのWWW－データベース連携アプリケーションを整理してみると、データベースに蓄積した情報に基づいてWWWコンテンツの作成を支援するコンテンツ型アプリケーションと、業務データベースをWWWブラウザを用いてイントラネットで利用する情報システム型アプリケーションに分類できる。今後は、複数のデータベースの情報をWWWブラウザから自由に参照したり、自分からデータベースにアクセスしていくプル型のアプリケーションだけではなく、必要な情報の種類を予め宣言しておくと、多数のデータベースの情報から必要なものだけをエンドユーザに配送してくれるようなプッシュ型のアプリケーションも開発されることだろう。このような新しいWWW－データベース連携アプリケーションでは、これまでの連携技術だけではなく、複数のデータベースの情報を管理するディレクトリ技術や必要な情報を選別するフィルタリング技術などが重要となる。

1．6．6 まとめ

WWW－データベース連携を実現するために必要となる基本技術について説明した。WWWとデータベースとを融合させるWWW－データベース連携技術により、ますます多様なアプリケーションを簡単に構築できるようになった。今後も、インターネットやイントラネットの世界で、ここで紹介したようなさまざまなWWW－データベース連携製品の開発競争が展開されていくと思われる。

【参考文献】

1) 日経コンピュータ, イントラネット構築のすべて, 日経ＢＰ社 (1996).

2) WWW－データベース連携システム構築法, 日経ＢＰ社 (1996).

3) 日経オープンシステム, イントラネット　WWWで文書共有, DB連携環境も整う, pp.240- 257, 5月号, no.38 (1996).

4) 日経オープンシステム, イントラネットかＣ／Ｓか, pp.236- 257, 9月号, no.42, (1996).

5) Hettler, M., Serving Up Data on the Web, BYTE, September, pp.112-116, (1996).

6) 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).

7) 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).

8) 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).

9) 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).

10) 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).

11) 河野, 長谷川: WWWデータ資源検索におけるデータマイニング手法, データベースシステム研究会, 情報処理学会, pp.33-40 (1996).

参考URL

1) WebBASE, http:// robin. sl. cae. ntt. jp/ vgindex.html
2) WebServer, http://www.oracle.com/
3) Web.sql, http://www.sybase.com/
4) WebSite, http://www.ora.com/
5) Live Wire, http://home.netscape.com
6) IIS, http://www.microsoft.com

2．アプリケーション統合環境

2．アプリケーション統合環境

2．1　エンドユーザ向けアプリケーション統合環境

ユーザニーズの多様化から、ユーザの個別ニーズに合った様々なアプリケーションを構築できる環境が求められている。ここでは、そうした環境の基礎となる部品（コンポーネント）を用いたアプリケーション開発、エンドユーザプログラミング、統合環境モデルについて考えてみる。

2．1．1　コンポーネントベース開発

複合文書（Compound document）技術をアプリケーションレベルの複合技術に発展させたコンポーネントソフトウェアに対する期待が高まっている[19]。文書をベースに様々なアプリケーションが統合される複合文書と同様の方法で、コンポーネントベース開発は、プラグ＆プレイの形式でコンポーネントソフトウェアを組み合わせて、エンドユーザの望むアプリケーションの構築を可能にさせる。ここでは、コンポーネントを用いたアプリケーション開発の具体的なイメージを得るために、マイクロソフト社のVisual Basic[7]を取り上げ、再利用可能なコンポーネントを用いた環境によってアプリケーションがいかに構築されるかを見る。ここで取り上げるVisual Basic（バージョン4.0）は次のような特徴を持っている。

(1) ビジュアルプログラミングツール

ビジュアルプログラミングには、様々な定義があるが、その最も広い範囲の定義は、ビジュアルなオブジェクトを用いたプログラミング及びその環境といえる。この定義においては、Visual Basicは、GUIベースのアプリケーションをテキストボックス、ボタンなどのビジュアルな部品を用いて作成できることからビジュアルプログラミングの範囲に入る。しかし、Prograph[2]のようにプログラミングそのものをビジュアルなオブジェクトを用いて行うものではない。

(2) GUI構築ツール

最近のユーザインタフェースは、テキストベースのインタフェースからWIMP（ウインドウ、アイコン、メニュー、ポインティングディバイス）を用いたインタフェースに変わってきており、こうしたインタフェースを容易に構築できる環境が求められている。従来、ウィンドウインタフェースのプログラミングはテキストベースのプログラミング言語を用いて開発していたため、極めて多くのプログラミングステップを要し、しかも変更が大変であった。こうした問題の解決を目指した製品としてNext社のInterfaceBuilder[16]が有名である。InterfaceBuilderでは、ユーザインタフェースをWYSIWYG（What You See Is What You Get）形式すなわち、最終的な結果をみながら作成できる。Visual BasicもInterfaceBuilderのように、WYSIWYG形式

でインタフェースの作成が行える。

(3) イベント駆動プログラミング

グラフィカルユーザインタフェースでは、ユーザのアクションシーケンスをアプリケーション側でコントロールするアプリケーション駆動プログラミングより、ユーザからのアクションによるイベントにコードを対応付けるイベント駆動プログラミングの方が好都合である。Visual Basicもイベント駆動型のプログラミングスタイルをとっている。

(4) プログラミング言語

コード記述言語としては、Apple社のHyperCardで使われているHyperTalk[17]のような簡易言語からParcPlace-Digitalk社のVisualSmalltalk[18]で使われているSmalltalkのようなオブジェクト指向言語まで様々なものがある。Visual Basicでは、その継続性から従来からのBasicをプログラミング言語としている。ただし、Visual Basicでは、クラス定義、オブジェクト生成などのオブジェクト指向拡張が行われている。

(5) GUIコンポーネント再利用環境

Visual Basicは、OLE（Object Linking and Embedding、ただし最近では機能の拡張から単にOLEと呼ぶようになっている）と呼ばれる技術を用いており、テキストボックス、ボタンなどのコントロール部品（これをOCXという）を使ってGUIアプリケーションを作成することができる。OLE [1] は、そもそも複合文書（Compound document）のために開発されたといわれているが、その後、拡張が続き、複合文書以外にもオブジェクトの連携技術として使われている。また、最近では分散オブジェクトの連携を可能にさせるActiveXへと発展している。OLEのオブジェクトモデルはCOM（Componet Object Model）と呼ばれ、OLEでのオブジェクトは、バイナリー形式で、インタフェースを介してのみアクセスできる実体を示している。この標準化により、いわゆるPlug-and-Playの形式でのコンポーネントの利用が可能となっている。コントロール部品は、Visual Basicが標準でもっているものの他にカスタムコントロールを利用することができる。カスタムコントロールは、Visual C++などで作成できることからソフトウェアベンダーなどが、様々なカスタムコントロールを開発販売しており、部品マーケットが形成されつつある。

(6) アプリケーションレベルコンポーネント再利用環境

マイクロソフト社では、同社の様々なビジネスアプリケーション（表計算ソフトのEXCELやデータベースソフトのACCESSなど）のOLE化を進めている。その結果、こうしたアプリケーションの様々なメソッドやプロパティに対するアクセスをVisual Basicで作成するアプリケーションから行える。例えば、作成しようとするアプリケーションで計算作業が必要なとき、アプリケーション側からEXCELにメソッドを送り、計算をEXCELで行って結果を得るというようなことができる。これをOLEオートメーションといい、Visual Basicで作成したアプリケーション側をOLEオートメーションコントローラ（あるいはクライアント）、

EXCEL側をOLEオートメーションサーバという。このように、OLEはアプリケーションを使ってアプリケーションを作る複合アプリケーション開発環境となっている。

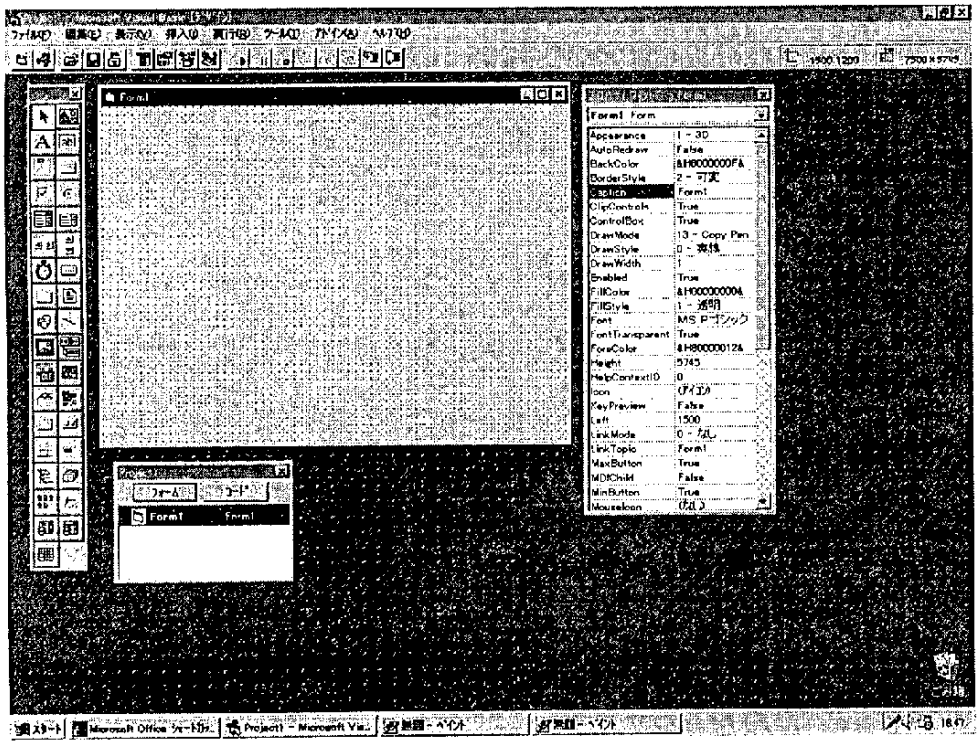


図2.1-1　Visual Basicの初期画面

以上Visual Basicの特徴について述べてきた。次にVisual Basicを用いてGUIベースのアプリケーションをいかに作るかを具体的にみてみる。Visual Basicをスタートさせると、メニューバー、ツールバー、ツールボックス、フォーム、プロパティウインドウ、プロジェクトウインドウから成る図2.1-1の初期画面が表示される。

①メニューバー

ファイル、編集、表示などのVisual Basicに対する操作メニューが表示される。

②ツールバー

Visual Basicを使う上で頻繁に使われるコマンドが表示される。コマンドには、新規フォームの作成、アプリケーション作成時でのプログラムの実行（プロトタイプ実行）などがある。

③ツールボックス

フォーム上に配置することのできるコントロール部品が表示される。コントロールの種類としては、リストボックス、ボタン、ラベル、タイマーなどがある。メニューバーのツールからカスタムコントロール

を選択することにより、コントロールの種類を追加することができる。

④フォーム

フォームは、作成するアプリケーションのメインウインドウとなるもので、この上にコントロールを配置する。ツールボックスウインドウ上で利用したいコントロールをダブルクリックすると、フォーム上にコントロール（例えばボタンなど）が表示され、プロパティウインドウにそのコントロールのプロパティが表示される。初期画面に表示されているフォームは何も行わなくてもすでにプログラムの機能をもっており、実行することができる。実行するとフォームがそのままウインドウとして表示され、最小化ボタン、最大化ボタン、クローズボタン等が使える。

⑤プロパティウインドウ

プロパティとは、オブジェクト（フォームやコントロール等）がもっている色、線の太さ、文字の大きさ等の特性で、プロパティウインドウには、プロパティの種類と既定値が表示される。

⑥プロジェクトウインドウ

プロジェクトは、アプリケーションの作成に必要なファイルの集合を示し、プロジェクトウインドウには、フォームモジュール、クラスモジュールなどの一覧が表示される。

さてここで、実際にアプリケーションを作成する手順を見てみる。Visual Basicでのアプリケーション作成手順は、次のようになる。

①コントロールの配置

②プロパティの設定

③コードの記述

図2.1-2に簡単なアプリケーションの構築画面を示す。

ここでは、まず、ツールボックスからテキストボックスを選択しフォームに配置する。同様に、コマンドボタンを２つフォーム上に配置する（コマンドボタン１とコマンドボタン２）。次に、プロパティウインドウを使ってフォーム及びコントロールのプロパティを変更する。

［フォームのプロパティ］

Caption（フォームのタイトルバーに表示されるフォーム名）を「Form1」から「Text Writing」に変更。

BackColorプロパティ（背景色）をカラーパレットを用いて変更（カラーパレットはBackColorプロパティの値をクリックすることにより表示される）。

［テキストボックのプロパティ］

Nameプロパティ（コントロールオブジェクトの名前）を「Text1」から「txtEnglish」に変更。

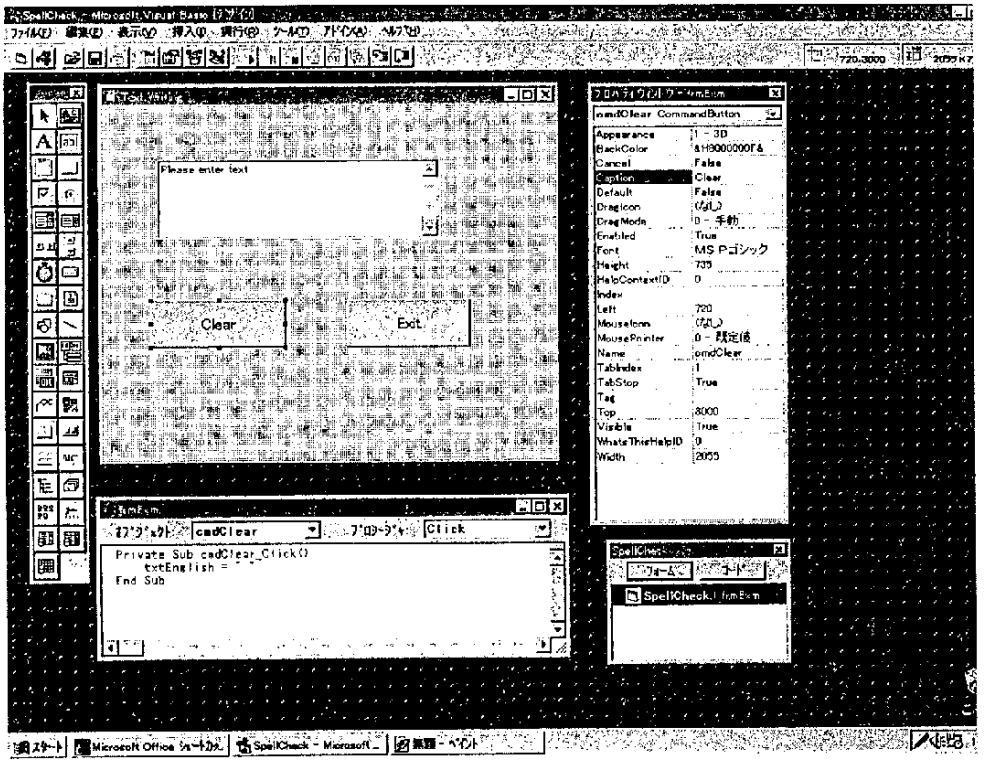


図2.1-2　簡単なアプリケーションの構築画面

MultiLineプロパティ（複数行の入力の可否）を「FalseからTrue」に変更。

ScrollBarsプロパティ（スクロールバーの有無）を「なし」から「垂直（垂直スクロールバー有り）」に変更。

Textプロパティ（テキストボックス内に表示されるテキスト）を「Text1」から「Please enter text.」に変更。

［コマンドボタン１のプロパティ］

Nameプロパティを「Command1」から「cmdClear」に変更。

Caption（コマンドボタン上に表示されるボタン名）を「Command1」から「Clear」に変更。

Fontプロパティ（フォント、サイズ等）のサイズを「9（初期値）」から「12」に変更。

［コマンドボタン２のプロパティ］

Nameプロパティを「Command2」から「cmdExit」に変更。

Caption（コマンドボタン上に表示されるボタン名）を「Command2」から「Exit」に変更。

Fontプロパティ（フォント、サイズ等）のサイズを「9（初期値）」から「12」に変更。

以上により、図2.1-2のフォームができ上がる。最後にコードを記述する。VisualBasicは、イベント駆動のプログラミングスタイルをとっているため、コントロールごとに必要なコードを作成する。ここでは、

Clearボタンに対応するコードとExitボタンに対応するコードを作成する。コード作成のためのウインドウはボタンをダブルクリックすることにより表示される。コードウインドウには、イベント（マウスクリック等）に対応したコードのひな型が表示されるので、必要なコードを付け加える。

［コマンドボタン１のコード］

```
Private Sub cmdClear_Click
  txtEnglish = " "　（テキストボックス内の文字をスペースにする）
End Sub
```

［コマンドボタン２のコード］

```
Private Sub cmdClear_Click
  end　（アプリケーションを終了させる）
End Sub
```

これまでの操作でアプリケーションが作成されたことになる。メニューバーの実行から開始を選ぶと今作ったアプリケーションの実行が始まる。テキストボックスに文字を入力した後、Clearボタンをマウスでクリックすると、テキストボックスの文字がクリアーされ、Endボタンをクリックするとアプリケーションが終了する。

次に、OLEオートメーション機能を用いて、テキストボックス内のテキストのスペルチェックを行うアプリケーションの作成を見てみる。ここでは、図2.1-2のアプリケーションの画面にSpell Checkのボタンを付け加え、EXCELのスペルチェック機能を使う。新たなアプリケーションの構築画面を図2.1-3に示す。

まず、先ほどと同様な手順で「Spell Check」というボタンを作成し、このボタンのコードを次のように記述する。

```
 1  Private Sub cmdCheck_Click ()
 2  Dim X As Object
 3  Set X = CreateObject("Excel.Sheet")
 4  X.Cells(1, 1).Value = txtEnglish.Text
 5  X.Visible = True
 6  X.CheckSpelling
 7  txtEnglish.Text = X.Cells(1, 1).Value
 8  X.Application.Quit
 9  Set X = Nothing
10  End Sub
```

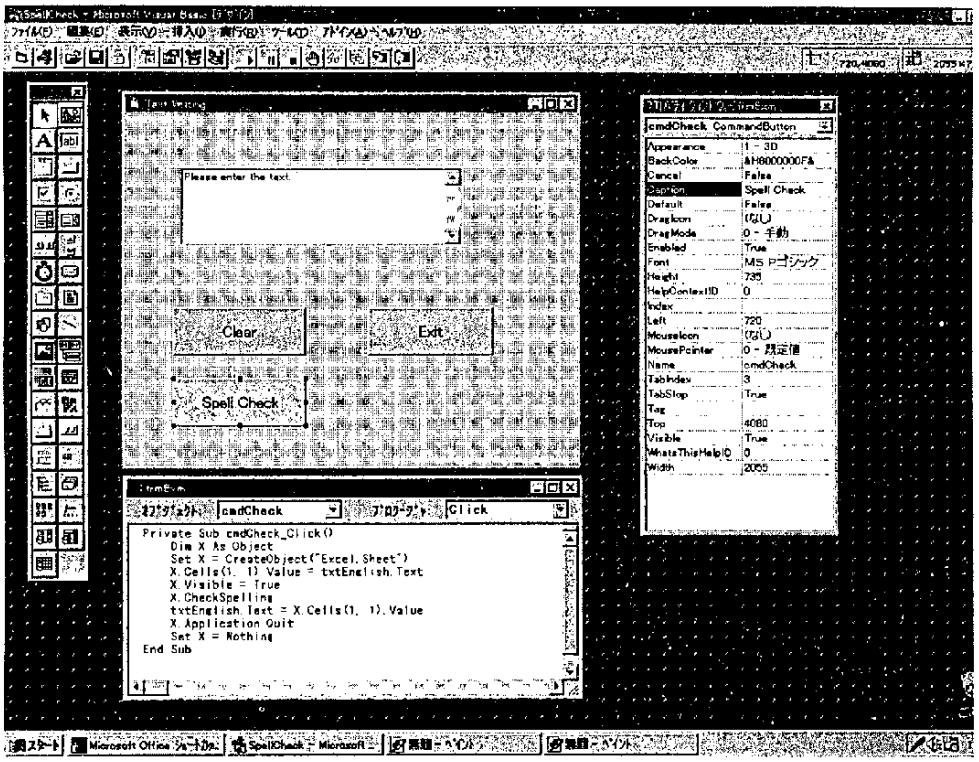


図2.1-3　新たなアプリケーションの構築画面

先に述べたように、Basicはオブジェクト指向プログラミング言語としての拡張が行われており、クラスやオブジェクト（インスタンス）の概念を使うことができる。次に各行の説明を行う。

①Spell Checkボタンのクリックプロシージャ（サブルーチン）をPrivateと宣言する。

②Xをオブジェクト変数として定義する。

③EXCELを起動し、オブジェクト変数（X）に割り当てる。

④テキストボックス内のテキストをセル（1，1）にコピーする。

⑤EXCELのダイアログボックス表示を可能にさせる。

⑥スペルチェックメソッドを呼び出す。

⑦セル（1，1）の内容をテキストボックス内のテキストにコピーする。

⑧EXCELを終了させる。

⑨オブジェクト変数を解放する。

⑩サブルーチンの終了を示す。

このアプリケーションを実行し、Spell Checkを押下すると、EXCELでテキストボックス内の文章のテストが行われ、スペルのエラーがあるとスペルチェックウインドウが表示（図2.1-4）される。また、すべてのチェックが終了すると、訂正された文章がテキストボックス内に表示される。

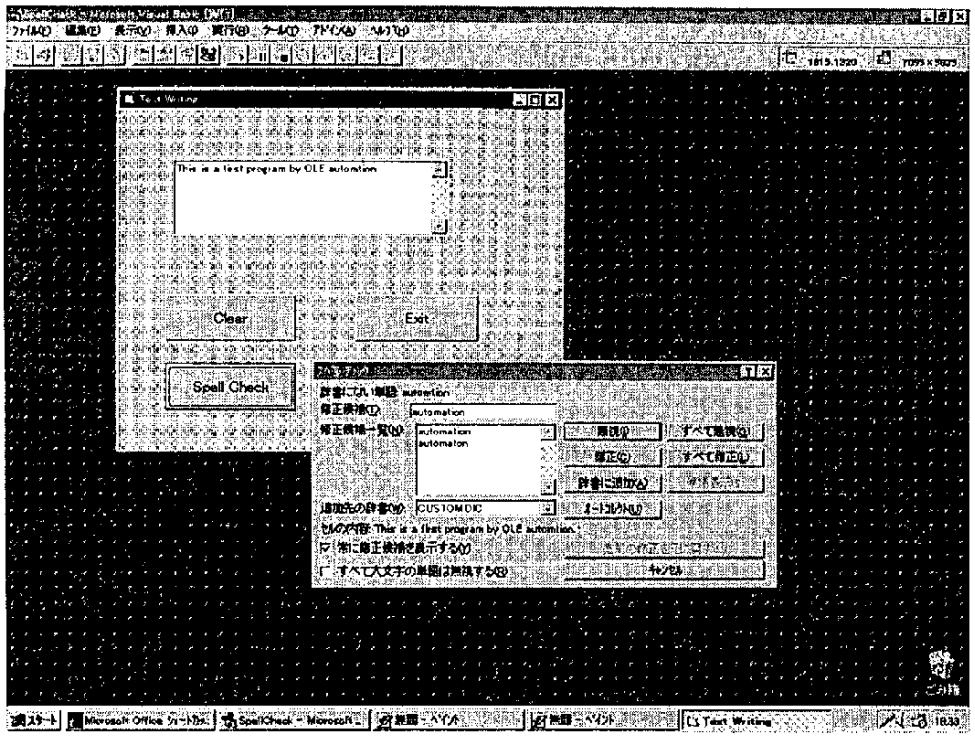


図2.1-4　スペルチェック実行中の画面

以上、Visual Basicを基に再利用コンポーネント（コントロールオブジェクトとサーバオブジェクト）をも用いたアプリケーションの作成について述べてきた。

次に、コンポーネントをベースとした開発の問題点や課題について述べる。

(1) コンポーネントのミスマッチの問題

Garlan [5]らはAesopと呼ばれるアーキテクチャ設計環境を生成するシステムを開発する過程において深刻なソフトウェアアーキテクチャ上のミスマッチに遭遇したと報告している。Aesop システムは、OBSTと呼ばれるオブジェクト指向データベース、InterViewと呼ばれるGUI生成ツールキット、SoftBenchと呼ばれるツール統合環境、Mach RPC Interface Generator（MIG）と呼ばれるRPCスタブジェネレータの４つのコンポーネントの組み合わせで開発したもので、開発過程で、全体のコードが膨大になった、パフォーマンスに問題があった、再利用可能パッケージと思われる部分においてもリバースエンジニアリングを含む改変が必要であった、パッケージへの機能の追加を必要とした、などの問題を発生させたと述べ、ミスマッチは、コンポーネント（基本的な生成、蓄積エンティティ）、コネクタ（コンポーネント間インタコネクション）、全体的なアーキテクチャ上の構造、構築プロセス、における仮定が相互に異なっていることに起因している述べている。ここで、コンポーネントの仮定とは、インフラストラクチャ、コントロールモ

デル、データモデル等の違いであり、コネクタの仮定とは、プロトコルや通信データモデルの違いである。ミスマッチを解消する方法として、暗黙の内に定義されているこれらコンポーネントアーキテクチャを明確に記述する方法やそのための言語の開発を提案している。

Garlanらの研究に対して、Sullivan[14]らは、OLEの環境下でVisioと呼ばれる描画ツール、ACCESSと呼ばれるデータベースとVisual Basicを組み合わせて欠陥木解析関係のシステムを構築したところ、ミスマッチの問題はそれほど発生せず、パフォーマンス的にも通常のシステムと同様であったと報告している。

また、ミスマッチを発生させなかった原因の多くは、コンポーネントインテグレーションアーキテクチャ（OLE）によるとし、こうしたアーキテクチャの重要性とアーキテクチャに基づいたコンポーネントの開発が重要であるとしている。

コンポーネントを用いた開発はまだ始まったばかりであるが、今後その普及を拡大させるためにはコンポーネントの性格を明確にする方法や統合化のための標準アーキテクチャの開発が必要とされる。

(2)コンポーネントの設計

Visual Basicの例で考えると2種類の粒度のコンポーネントを扱っていることが分る。一つは、テキストボックス、コマンドボタンなどのコントロール（入出力関係のインタラクションをコントロールという）部品の大きさであり、もう一つはExcel、Wardなどのアプリケーションの単位である。Excel、Wardなどのアプリケーションを利用しているユーザにとっては、スペルチェックや文章編集機能の呼び出し利用は馴染があって分り易いとも言えるが、コンポーネントとしては、単独のものがライブラリーとして提供された方が使い易い。今後、ユーザ独自のコンポーネントを開発していくためには、コンポーネント化のための分析、設計方法論等が必要とされる。

(3)コンポーネントの流通

コンポーネントベースの開発ツールは、Visual Basicの他にも、IBMのVisualAge、日本電気のHOLON/VPなど様々なものが開発されている。Visual Basicと日立のAPPGALLARYなどコンポーネント部品をお互いに共通に使えるものもあるが、多くは、部品の互換性がなく重複投資や優れた部品が使えないという状況が発生している。サンマイクロシステムズのJava Beansなどは、他の環境へのブリッジを考えているようであるが、今後更なる共有を進めていくためには、コンポーネントのミスマッチの所で述べた標準アーキテクチャを含めた何らかの標準化あるいは互換性の確保が必要とされる。

(4)さらに大きなコンポーネントの統合環境（メガプログラミング）

DARPAでは、メガモジュールと呼ばれるソフトウェアコンポーネントを用いて大規模システムを構築するメガプログラミングの研究を行っている[15]。関数やプロシージャが命令を隠蔽し、オブジェクトやクラスがデータとプロシジャを隠蔽しているという流れからすると、メガモジュールは振舞いや知識や並

列性やオントロジーを隠蔽しているといえる。こうした隠蔽により、メガモジュールの高度な自律性が確保でき並列処理が可能となる。Boehmらは、図2.1-5のメガプログラミングのためのエンタープライズモデルを提案している[6]。このモデルは、メガプログラミングのモデルとして提案されているが、コンポーネントを用いた開発の一般モデルとしても使えよう。

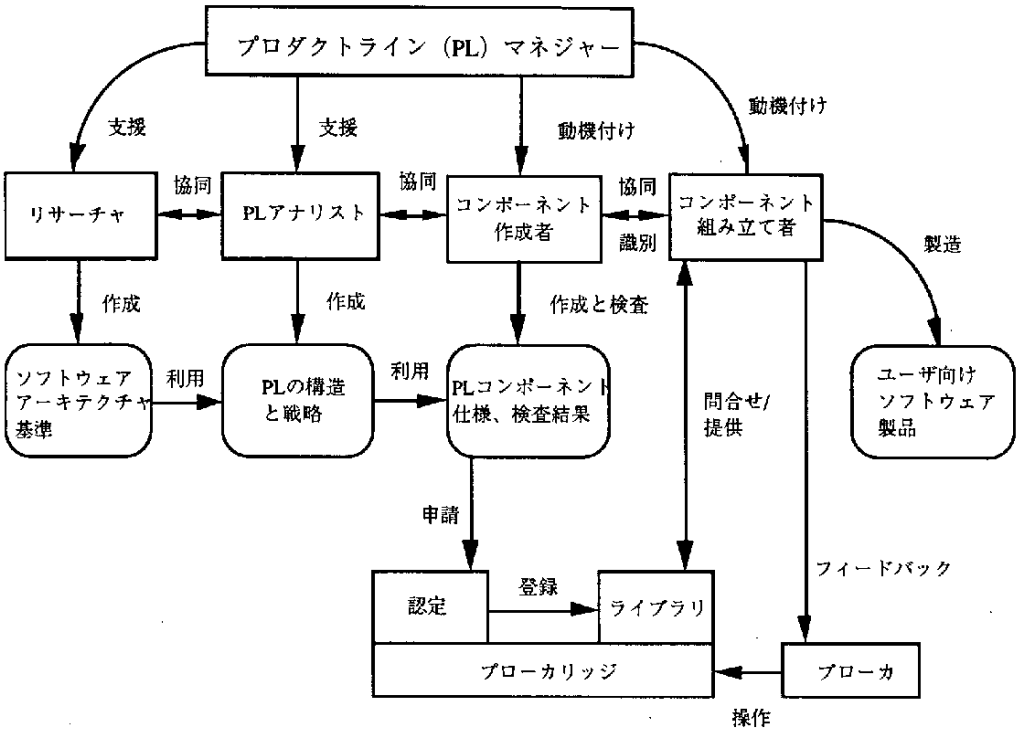


図2.1-5　メガプログラミングのためのエンタープライズモデル

2．1．2　エンドユーザプログラミング

Visual Basicのようなシステムによって、アプリケーションの作成が容易になったとはいえ、エンドユーザが自らアプリケーションを作成するとなると敷居が高い。そこで、もう少し敷居の低い、すなわち変数、繰り返し、分岐などのコンピュータサイエンスの概念を理解していなくてもプログラムを作成できるシステムへの期待が高まっている。そこで、ここではエンドユーザプログラミングとして注目されているビジュアルプログラミングについていくつか紹介し、その可能性や問題点について述べる。

(1)表計算ソフトウェア

エンドユーザが自らプログラムを作成することのできるシステムとして、最も実用的に広く普及しているシステムは、Lotus1-2-3やEXCELのような表計算ソフトウェアであろう。表計算ソフトウェアは、グラフィカルに表示される表という枠組みを用いて極めて簡単にプログラムを作成することができる。表計算ソフトウェアにおいてユーザが学習すべきことは、セルと、セル間の関係を示す関数のみである。関数と

言っても大抵は合計の計算や平均の計算などであり、これらも高レベルの記述法（例えばsumやaverageなど）が用意されている。すべての処理は、セルというローカルな場所のみで行われる。すなわち、すべてのことがセルに隠蔽され、表全体の制御を考える必要がない。プログラム作成という観点で見ると、プロトタイプ的な段階的開発が可能であるといえる。なかでも、入力に対する直接的なフィードバックはプログラムのデバックに大いに役立っている。例えば、あるセルに複数セルの合計を指定すると即座に計算が行われ値が表示される。また、もう一つ大切な点は、表計算の言語をすべて知っていなくても、とりあえず知っている範囲で使い始めることができることである。数時間の学習で玩具のプログラムでない実用プログラムを作成できるようになる。

表計算ソフトウェアをその構造から眺めると、グラフィカルなオブジェクト（表の枠組み）とシンボリックなオブジェクト（関数）がうまく組み合わさり、ハイブリッドなビジュアルアプリケーション構築環境を提供していることが分る。ビジュアルプログラミングと言うと勢いグラフィカルなオブジェクトに重点が置かれるが、両者は各々お互いを補完する特性をもっており、ビジュアルプログラミングシステムの開発に当たっては、両者をうまくミックスした方法が有効といえる。

表計算ソフトウェアをユーザの視点から見ると、表を作成するという仕事（タスク）に特化したプログラミング環境といえる。タスクに特化したビジュアルプログラミング環境としては、制御データの分析やシミュレーションに特化したLabVIEW[2]や科学情報の視覚化に特化したAVS[10]等がある。これらのツールを見ると、業務や業務知識に密着し業務に近い表現能力をもつツール（タスクに特化したプログラミング環境）を作成すればエンドユーザといえども使いこなすことができることが分る。

しかし、こうした環境を構築するに当たっては次のような問題も潜んでいる[9]。

・タスク毎に異なったプログラミング環境を用意するのは不経済である。

・タスク毎にプログラムを変えなくてはならなく、習得や整合性の問題がある。

・根本的な問題として、どのような対象、範囲をタスクとして切りだし、ツール化するかを決定するのが困難である。

この内、最初の問題は、本書で中心的に扱っている再利用コンポーネントの発展によって解決されるかもしれない。

(2) Pygmalion

Shu [12]は、ビジュアルプログラミングの分類を、視覚的環境と視覚的言語に分類している。視覚的環境は、データとデータ構造情報の視覚化、プログラムと実行の視覚化、ソフトウェアの視覚化、ビジュアルコーチングから成る。視覚的言語は、視覚情報を操作する言語、視覚的対話を支援する言語、視覚表現を用いたプログラミング言語から成る。エンドユーザプログラミングの視点からは、視覚表現を用いたプ

ログラミング言語とビジュアルコーチングが重要である。今迄述べてきたGUIベースのプログラミングツールや表計算ソフトは、視覚表現を用いたプログラミング言語の範疇に入もので、ここで述べるPygmalion（及び次に述べるEagerとPursuit）は、ビジュアルコーチングに入る。ところで、ビジュアルコーチングは、しばしばデモンストレーション（略してデモ）によるプログラミング（Programming by Demonstration）とか例示プログラミング（Programming by Example）と呼ばれる。ビジュアルコーチング、デモによるプログラミング、例示プログラミングは、本質的な違いがなく、またビジュアルコーチングがCAI（Computer-Assisted Instruction）の感じを与える、例示プログラミングよりデモによるプログラミングの方が直感的で分り易い等の理由により、これらまとめてデモによるプログラミングと称することにする。

Pygmalionは、今から20年以上前の1975年に研究開発されたデモによるプログラミングの最初のシステムであり、後続のシステムに大きな影響を与えたと言われている[4]。

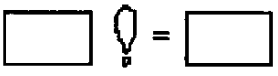
Pygmalionによっていかにプログラミングが作成されるかを例によって示す。例は、階乗計算で、C言語によるプログラムは次のようになる。

```
int factorial (int n)
{
    if (n==0)
    {
        return (1);
    }
    else
    {
        return (n * factorial (n - 1));
    {
  }
```

図2.1-6　C言語による階乗計算プログラム

Pygmalionのウィンドウの例を図2.1-7に示す（この画面は、Xerox Alto上に開発されたPygmalionをHyperCardでシミュレートしたものでオリジナルではない）。Pygmalionで6の階乗を計算するプログラムを作成する手順は次のようになる。

①まず階乗のアイコンを作成しfactorialと名付ける。



②アイコン内に6とタイプする。Pygmalionは、すぐに実行しようとするが、何を行うかまだ指定されていないので、ユーザに対して何をしたらよいか尋ねる。

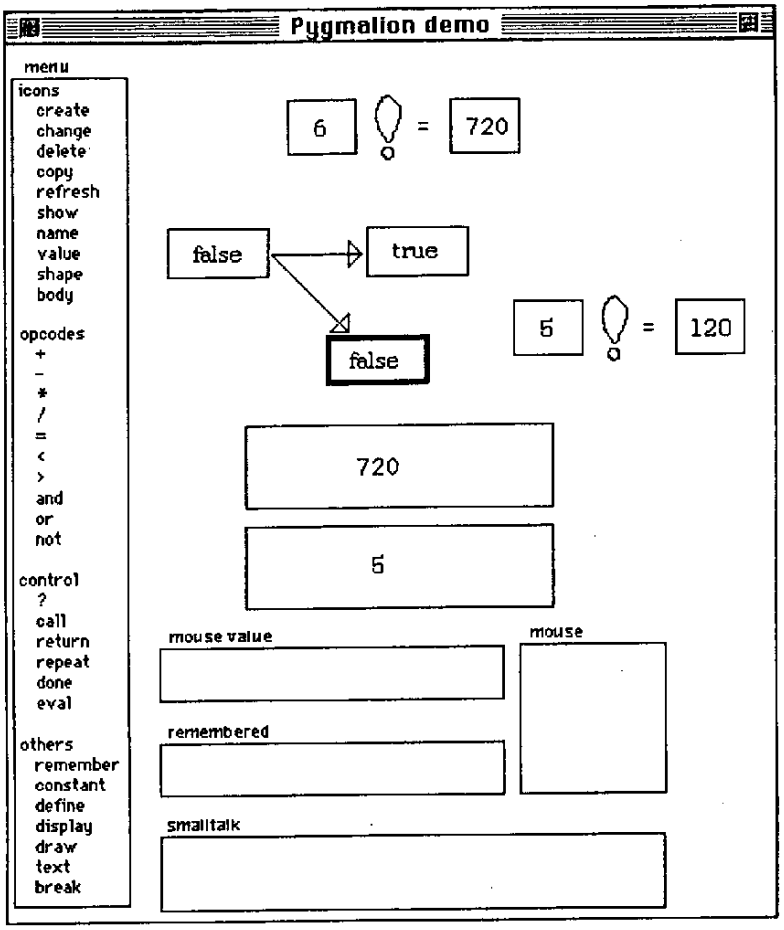
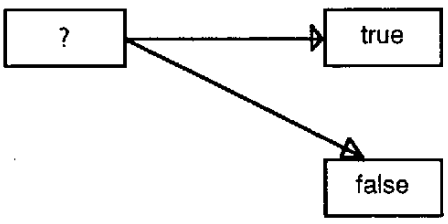


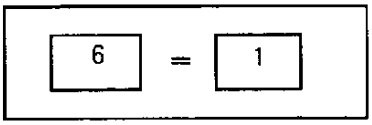
図2.1-7　Pygmalionのウィンドウの例

③そこで、ユーザはcontrolメニューから条件分岐（？）を選択する。画面には、



が表示される。

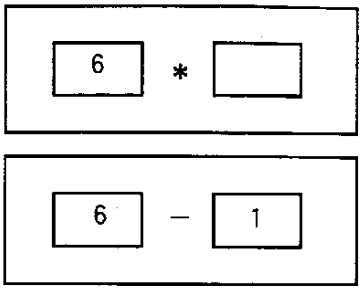
④次に、この条件分岐の条件を指定するために、controlメニューから＝比較を選択し、６と１の比較を指定する。



⑤値が指定されると、等式が評価され、結果がfalseになる。このfalseを③の［？］にドラッグ＆ドロップすると条件が評価され、falseに分岐する。

⑥falseに対して、Pygmalionは何をすべきか分らないので、Pygmalionはユーザに対して何をすべきかを尋ねる。

⑦そこで、乗算アイコン（＊）と減算アイコン（－）を用いて



を作成する。

⑧６－１の減算が即座に評価され、結果として５が得られる。

⑨factorialと名付けたアイコンを複製し、左辺に⑧の結果の５をドラッグ＆ドロップする。

⑩factorialのアイコンの実行が開始される。ここで、Pygmalionは、５に対する計算手順を今迄の経過で知っているので、③の分岐がtrueになるまで実行を継続する。

⑪trueに対して何を行ってよいかPygmalionは知らないので、ユーザに尋ねる。

⑫ユーザは、１の階乗が１であることを知っているので、上記の手順と同じような方法で１！＝１を指定し、その結果を２！の計算過程の２＊［　　］の四角の部分にドラッグ＆ドロップする。

⑬以上の操作でPygmalionは、すべての手順を理解し（すなわちプログラムが作成され）、６！の結果として７２０を得る。

Pygmalionは、そもそもプログラマーを対象として研究開発された。そのため、判断分岐で１と比較するというアルゴリズム的発想でプログラムが作成されている。しかし、大事な点は、変数などの抽象的なオブジェクトでなく、具体的なオブジェクトで指示を与えることによりプログラムができる点である。人間の知的発達が、身体的（enactive）な段階から、視覚的（iconic）な段階を経て記号的（symbolic）な段階に至ると言われているなかで、視覚的なものをマウスという身体的なもので操作してプログラムを作成するという考え方は、裾野の広いエンドユーザを対象としたプログラミングに適しているといえる。

(３) Eager

Eager [3]は、HyperCard環境におけるデモによるプログラミングシステムである。このシステムの最大の特徴は、プログラム作成過程におけるユーザインタラクションの最小化と推論機構である。まず、HyperCardで作成したメッセージカードスタック（図2.1-8）からsubject 欄内の文（サブジェクト）を抽出し、連番とサブジェクトからなるサブジェクト一覧カードを作成する例（サブジェクト一覧カードを作成するプログラムの作成）によりユーザインタラクションの最小化と推論がどのように行われるかを見る。

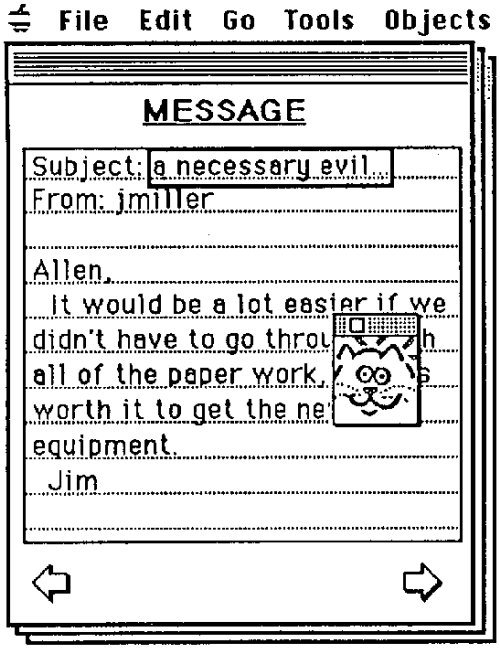


図2.1-8　メッセージカードスタック

①メッセージカード内のsubjectの文をマウスを使ったドラッグ操作により選択し、その内容をクリップボードにコピーする。

②サブジェクト一覧カードの一行目に１．とタイプし、次に①でコピーしたものをペーストとする。

③次のカードに移り、①と②の操作を行う。

④この時点で、Eagerは繰り返し作業が行われていると判断して、猫のマークを表示するとともに、次にユーザが行うであろう操作の対象（ボタンなど）の色をグリーンに変える。

⑤Eagerが先回りで示した操作（色で示しているだけで実際の操作は行っていない）と同じ事をユーザが行うと、更に次の操作を色で示す。ユーザが違う操作を行うと、Eagerは自分の推論が間違っていたと判断し、新たな推論に入る。

⑥ユーザは、Eagerが自分が行いたいことと同じことを推論していると確信し、猫のアイコンをクリックする。

⑦Eagerは、繰り返し作業のためのHyperTalk（HyperCardのスクリプト言語）によるプログラムを生成し、繰り返し作業を自動的に実行する。

Eagerと同じようなプログラム生成システムとしてマクロレコーダ（Macのスクリプト編集ツールなど）がある。マクロレコーダは、ユーザ操作のイベントを記録し、まったく同じ作業を繰り返し行うことはで

きるが、汎化が行えない。また、開始、終了などを陽に指定しなければならない。その点、Eagerはドメインの知識（この場合はHyperCardの知識）をもっており、一連の操作をスタックが終わるまでと判断し（汎化し）、プログラムを生成する。Eagerは、判断分岐や繰り返しの階層化ができないなど機能的に劣る面もあるが、実用的に使えるデモによるプログラミングとしてその意義は大きい。

(4) Pursuit

Xerox StarやMachitoshで取り入れられたデスクトップメタファは、UNIXのようなコマンドベースのシステムに比べて分り易い。その原因は、コンピュータ内の操作を、日常生活でよく知っているオブジェクトの操作に置き換えたことでる。例えば、ファイル管理システム内のカタログファイルに、データファイルを登録するというコンピュータ内の操作は、ドキュメントをフォルダーに入れるという日常的な操作に置き換えられることにより、コンピュータに馴染みのないユーザでも行うことができるようになる。このビジュアルなオブジェクトの直接操作とデモによるプログラミングの考えを使ったのがPursuit [8]である。図2.1-9にPursuitのプログラムを示す。

Pursuitでは、デスクトップ上でオブジェクトに対する操作が行われると、その模様がビジュアルなプログラムとして表示される。例えば、マウスによりファイルを選択し、そのファイルのコピー操作がコマンドメニュー操作などによって行われると、図2.1-9の右下のような図絵が表示される。すなわち、デスクトップ上でのコピー操作によりコピーというプログラムを作成したことになる。一見すると、マクロレコーダ

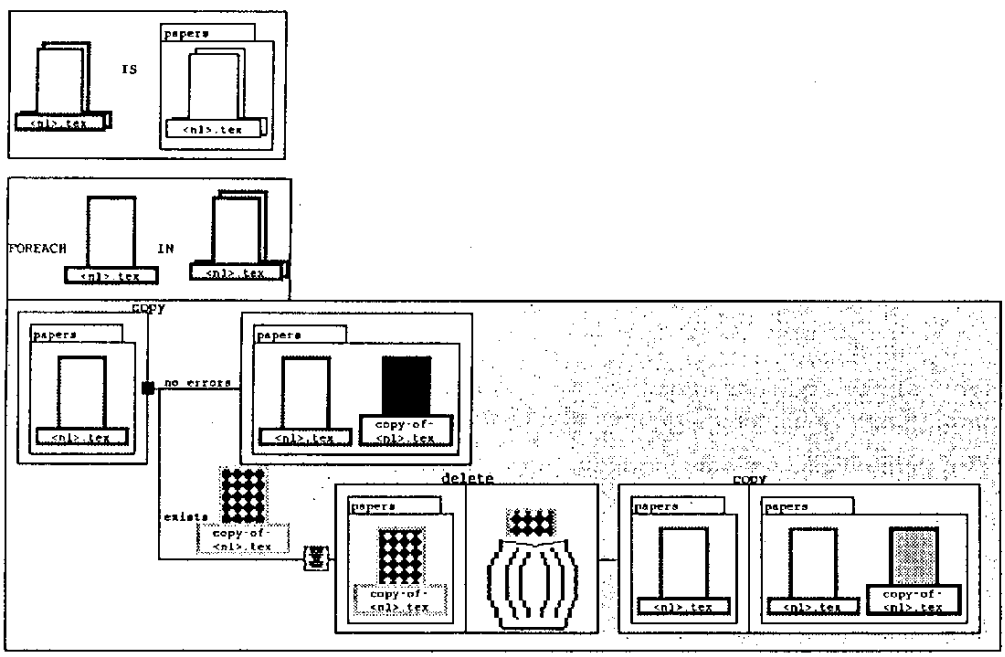


図2.1-9　Pursuitのプログラム

の記録内容をビジュアルに表示しているだけに見えるが、具体的操作の汎化が行われ、フォルダー内のすべてのファイルを対象とするプログラムになる。また、図2.1-9の左下にあるような判断分岐を行うこともできる。Pursuitでは、判断分岐のケースごとに、デモで操作（プログラム）を指定する。これは、抽象的な概念でプログラムを作成し、具体値（テストケース）でデバックを行う通常のプログラム作成のちょうど反対の手順でプログラムを作成していることになる。具体値でプログラムを作成する手順は分かり易いが、半面、操作が煩雑になるという欠点もある。例えば、図2.1-9は、エラー有り、無しの実行によりプログラムが作成されている。これを通常の手続き言語で行うと、エラー条件前までのコーディングは一回で、分岐条件後の操作を各々コーディングすれば済む。また、図絵による表示は、表示面積を多く必要とする。図2.1-9のようなプログラムをC言語などで書けば数行ですむ。ここら辺は、大きなサイズのプログラムをデモで作成するときの問題となろう。

(5) KidSim

KidSim（Kids' Simulations）[13]は、子供向けのシミュレーションプログラム作成ツールキットである。KidSimは、子供という非プログラマーが自らシミュレーションプログラムを作成できるように、プログラミング言語を用いずに、グラフィカルな書き換えルールとデモによるプログラミングを用いている。図2.1-10にKidSimのプログラム作成例を示す。

KidSimは、シミュレーションが実行されるゲームボード、実行を制御する時計（正順、逆順の実行、ステップごとの実行等）、シミュレーションオブジェクト、コピーボックス、ルールエディタなどから成っている。シミュレーションは、ゲームボードにシミュレーションオブジェクトを配置し、そのルールをグラフィカルな書き換えルールで記述する。ルールは、IF-THEN型のものであるが、記号を用いず、グラフィカルなオブジェクトをそのまま用いる。これにより、ユーザが考えている表現をそのまま用いることができ、コンピュータプログラムの世界でよく見られるコンピュータ表現（命令文）とメンタル表現（ユーザが描いているもの）のギャップを無くすことができる。しかし、グラフィカルなオブジェクトは、具体値（インスタンス）であり、ルールの汎化が困難という問題を起こす。KidSimでは、絵の抽象化（灰色の岩、すべての岩等）とプロパティの抽象化（岩の大きさ等）の手段を与えてこの問題に対処している。IF-THEN型ルールの実行時の振る舞いは、ルールエディタ内のオブジェクト（例では猿）をマウスで掴み、ドラッグ操作で岩の上をジャンプするという操作で与える（デモでプログラミングする）。

KidSimの特徴は、非手続き型のプログラミング（ルールベース）とデモによるプログラミングをうまく組み合わせてエンドユーザによるプログラムミングを可能にさせた点であると言える。

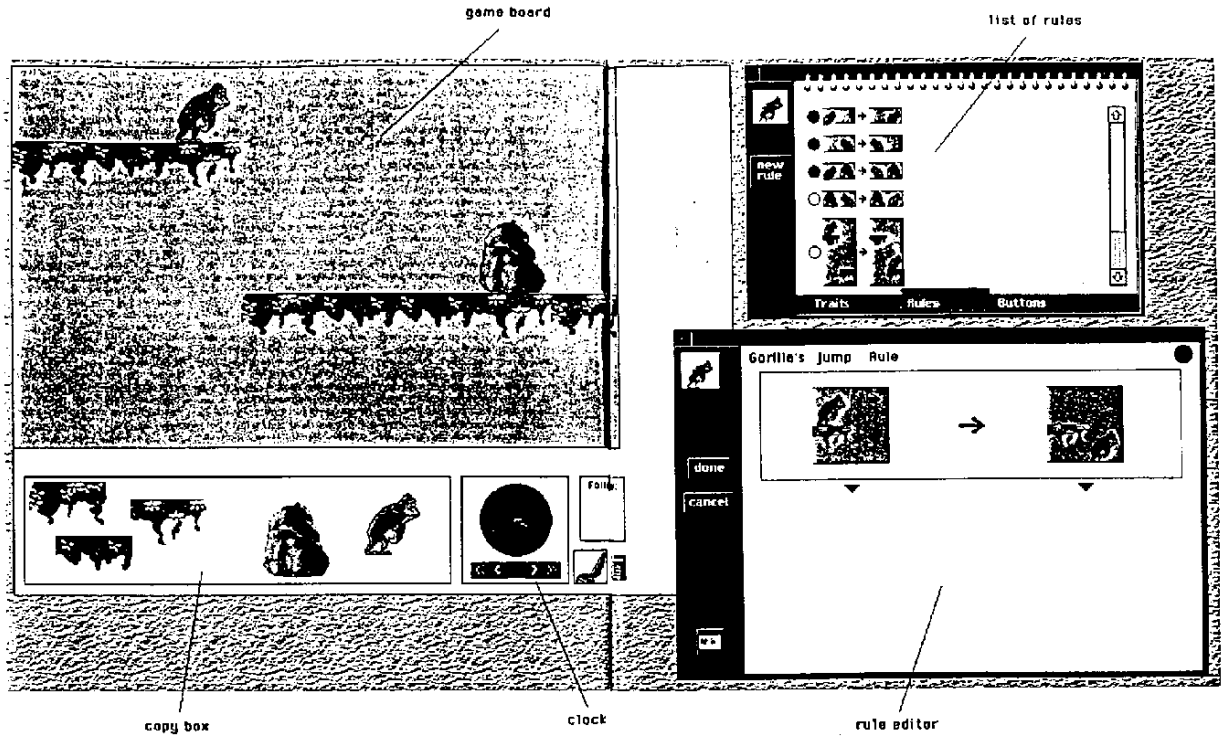


図2.1-10 KidSimのプログラム作成例

(6) Agentsheet

表計算ソフトウェアの所でも述べたように、エンドユーザプログラミングには、手続き言語のような低レベルの言語は不向きで、少数の高レベルコンポーネントからなるドメインに特化した環境が有効である。Agentsheet [11]は、ドメインに関する知識をもつエンドユーザとコンピュータの専門家であるシステムエンジニアが協同してドメインに特化したプログラミング環境（ビジュアル言語と処理系）を開発するためのツールである。Agentsheetによるドメインに特化したプログラミング環境構築の手順を表2.1-1に示す。

Agentsheetは、エンドユーザとシステムエンジニアが協同で作業を行うためのメディア環境ともいえる。図2.1-11は、構築ツールの観点からAgentsheetを見たものである。

エンドユーザは、Gallery(1)からコンポーネントを取り出し、ワークシート(2)内でそれらを組み合わせる（すなわちプログラムを作成する）。システムエンジニアは、ワークシートをエージェントシートの形で使用する。エージェントシートでは、ワークシートの内容をグリッドに配置された自立型エージェントとして参照できる。自立型エージェントは、エージェント間での通信を行う。システムエンジニアは、Depiction Editor (6)でアイコンを作成したり、拡張されたGallery(3)でそれらの関連付けを行う。また、Class Browser、AgenTalk Editor (5)、Tool Store (8)を用いて、コンポーネントの振舞いを定義する。

表2.1-1　ドメインに特化したプログラミング環境構築の手順

手順	エンドユーザ	システムエンジニア
1	ビジュアル言語を用いてプログラムを作成する。	ユーザの作成手順を観察する。
2	ビジュアル言語で表現できない事象に遭遇する。	ユーザの作成手順を観察する。
3	両者が協同でビジュアル言語の変更、拡張について検討する。	両者が協同でビジュアル言語の変更、拡張について検討する。
4	変更、拡張作業を観察する。	言語の変更、拡張作業を行う。
5	変更、拡張作業を観察し、1に戻る。	言語の変更、拡張作業後のテストを行い、1に戻る。

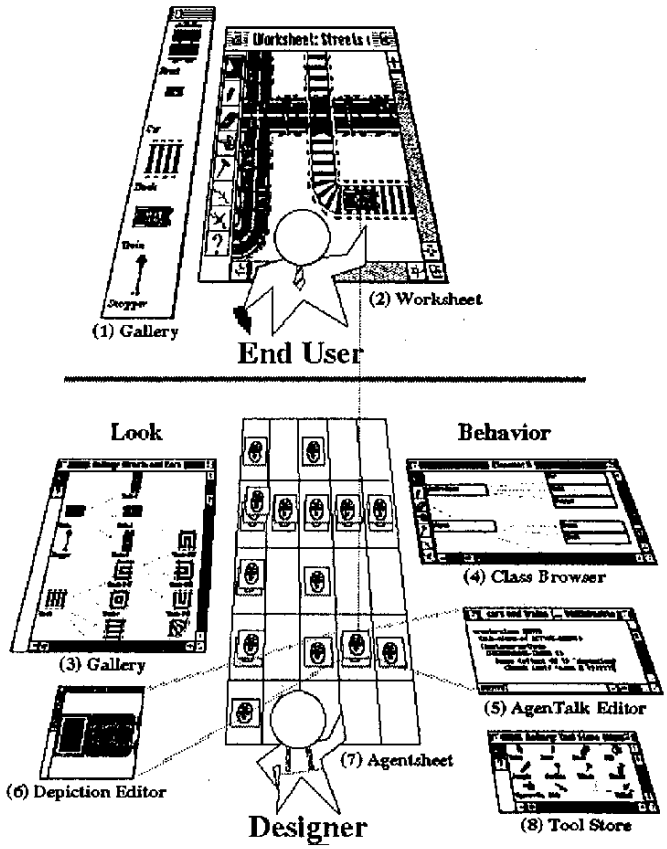


図2.1-11　Agentsheet構築ツール

Agentsheetは、ドメインに特化したプログラミング環境を構築できるのみならず、ユーザと協同でドメイン分析、ビジュアルプログラミング言語の設計、ユーザビリティのテストが一貫して行えるという利点がある。今後は、ドメイン分析ツール、タスク分析ツールとの連携などが期待される。

エンドユーザプログラミングは、コンピュータ利用の可能性を広げるものとして期待が高く、様々な研究が行われている。今後、更にエンドユーザプログラミング環境を前進させるためには、次のような研究が必要とされよう。

・新しいメタファ・・・デスクトップメタファがエンドユーザによるコンピュータ利用を広げたように、新しいプログラミングのためのメタファの研究。

・エージェント化・・・ユーザの意図を推測、理解し、細かい指示を与えなくてもユーザの望むことができるエージェントの研究。

・モデル変換・・・ユーザが抱いている高レベルの処理モデル（機能モデル）を計算機内の低レベルの処理モデル（手続きモデル）に自動変換する技術の研究。

・アダプティブシステム・・・ユーザの知識、経験やドメインの知識をユーザモデルやタスクモデルとしてもつシステムの研究。

・コンポーネント化・・・ソフトウェアをユーザが扱える高レベルのコンポーネントとして扱う技術の研究。

2．1．3　アプリケーション統合環境モデル

複合文書やコンポーネントベース開発を可能にさせるマルチベンダーでプラットフォーム・ニュートラルなアプリケーション統合環境として現在多くの支持を得ているのがOMG（Object Management Group）のOMA（Object Management Architecture）であり、エンドユーザ向けアプリケーション統合環境の有力な候補と考える。OMGは、７００社以上の企業が参加してオブジェクト指向技術の様々な標準化活動を行っている。OMGは、ISOやANSのような公的な標準化機関でないため、極めて短期間に様々な標準化（業界標準）を作成することができ、市場ニーズに基づいたタイムリーな標準化が可能となっている。図2.1-12にOMAを示す。OMAは、OMGの全体的な見直しの中で改定が計画されている（最新情報はOMGのホームページであるhttp://www.omg.org/を参照のこと）。OMGの活動は、当初ORB（Object Request Broker：ソフトウェアバス）やオブジクトサービス（オブジェクトの実行に必要となる基本的サービス）に重点が置かれていたが、これらの標準化が一応済んだことから、よりアプリケーション寄りの共通ファシリティ（Common Facility、OMGで採用された共通ファシリティは総称的にCORBAファシリティと呼ばれる）に中心が置かれ始めている。共通ファシリティは、アプリケーションが共通的に利用するサービスで、ORB

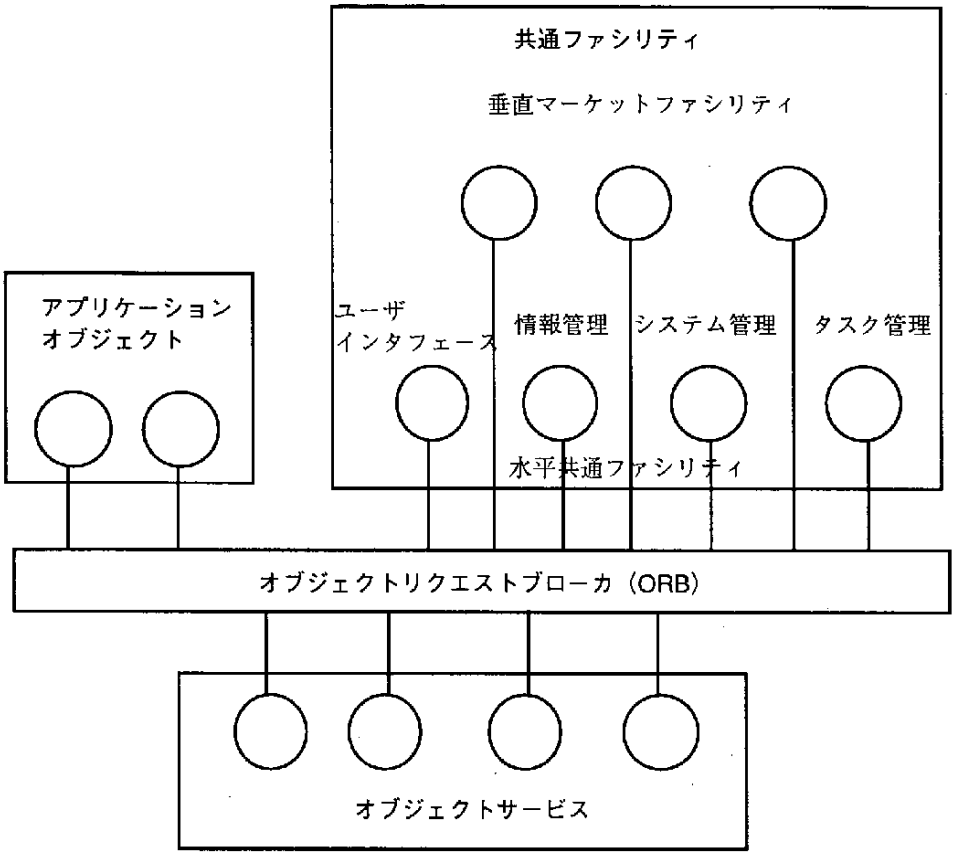


図2.1-12　OMA（Object Management Architecture）と共通ファシリティ

やオブジェクトサービスなどのインフラとアプリケーションオブジェクトの中間に位置し、印刷や複合文書を可能にさせるサービス等が入る。

共通ファシリティは、システムで共通的に使われる水平共通ファシリティとドメインに特化したタスクで使われる垂直マーケットファシリティがある。OMGでは、水平共通ファシリティと垂直マーケットファシリティをそれぞれ表2.1-2、表2.1-3のように分けている[20]。

この内、複合文書に関する水平共通ファシリティ（複合表示管理、複合交換）としてOpenDocが既に採用されている[21]。また、エージェント[22]やワークフローに関する議論も進んでいる。

こうした標準化作業や、2.1.2項で述べたビジュアルプログラミングの機能が追加されることにより、エンドユーザでも使いこなせるアプリケーション統合環境も近い将来可能となろう。

表2.1-2 水平共通ファシリティ

ユーザインタフェース	描画管理	印字や表示のようなオブジェクトの汎用表現支援
	複合表示管理	複合文書の印字や表示支援
	ユーザサポートファシリティ	アプリケーションのヘルプ情報の蓄積利用やテキストチェックのような共通的な機能
	デスクトップ管理	エンドユーザデスクトップに関する機能
	スクリプティング	オートメーションスクリプトのインタラクティブ作成の支援
情報管理	情報モデリング	情報モデルとスキーマの生成支援
	情報の蓄積と検索	情報の一貫性ある蓄積と検索支援
	複合交換	複合文書のデータ交換支援
	データ交換	データの一般的な交換支援
	情報交換	情報の交換支援
	データコード化と表現	データフォーマットのコード化と変換支援
	時間操作	カレンダーや時間の操作支援
システム管理	管理ツール	管理ツールのインタオペラビリティ支援
	コレクション管理	コレクション管理ファシリティの統合化支援
	コントロール	システムリソースのコントロール支援
タスク管理	ワークフロー	ワークプロセスを構成するオブジェクトのコーディネーション
	エージェント	静的、動的エージェントの支援
	ルール管理	ルールベースオブジェクトの支援
	オートメーション	オブジェクトの主要機能へのアクセス支援

表2.1-3 垂直マーケットファシリティ

画像	画像オブジェクト、画像情報、画像アプリケーション間のインタオペラビリティ
情報スーパハイウェイ	マルチユーザ情報サービスアプリケーションの支援
製造	製造オブジェクト間のインタオペラビリティ
分散シミュレーション	多重シミュレーションオブジェクトのインタラクション
石油、ガス産業	石油マーケットのインタオペラビリティ
会計	コマーシャルトランザクションの支援
アプリケーション開発	アプリケーション開発オブジェクト間のインタオペラビリティ
地図	地図オブジェクト間のインタオペラビリティ

【参考文献】

1) Brockschmidt, Kraig : INSIDE OLE Secon Edition, Microsoft Press (1995).

2) Burnett, Margaret : Visual Object-Oriented Programming, Manning Publications. (1995).

3) Cypher, Allen : EAGER : Programming Repetitive Tasks by Example, CHI91, pp.33-39 (1991).

4) Cyoher, Allen (ed) : Watch What I Do : Programming by Demonstration, MIT Press (1993).

5) Garlan,David et al. :Architectural Mismatch : Why Reuse Is So Hard, IEEE Software, pp.17-26 (1994).

6) Garlan, David et al. : Introduction to the Special Issue on Software Architecture, Trans. on SE, Vol.21, No.4, pp.269-274 (1995).

7) MicroSoft : Visual Basic プログラミングガイド (Version 4.0).

8) Modugno, Francesmary et al. : Visual Programming in a Visual Domain: A Case Study of Congnitive Dimensions, HCI'94, pp.91-108 (1994).

9) Nardi, Bonnie : A Small Matter of Programming : Perspectives on End User Computing, MIT Press (1993).

10) Rasure, Williams et al. : The State of the Art of Visual Languages for Visualization, Visualization'92, pp202-209 (1992).

11) Repenning Alexander et al. : Agentsheets : A Medium for Creating Domain-Oriented Visual Languages, IEEE Computer, March, pp.17-25 (1995).

12) Shu, Nan : Visual Programming Van Nostand Reinhold (1988) （西川博昭訳：ビジュアル・プログラミング、日経BP社 (1991)）

13) Smith, David et al. : KIDSIM : Programming Agents Without a Programming Language, CommACM, Vol.37, No.7,pp.55-67 (1994).

14) Sullivan, Kevin et al. : Experience Assessing an Architectural Approach to Latge-Scale Syatematic Reuse, ICSE-18, pp.220-229 (1996).

15) Wiederhold, Gio et al. : Toward Megaprogramming, CommACM, Vol.35, No.11, pp.89-99 (1992).

16) Nghiem, Alex : NeXTSTEP Programming Primer : Concepts and Applications, Prentice-Hall (1993)（生田りえ子訳：NEXTSTEP プログラミング, プレンティスホール, 1995）

17) Apple : HyperCard スクリプトランゲージガイド、アップルコンピュータ.

18) ParcPlace-Digitalk : VisualSmalltalk Language Reference (1995).

19) Adler, Richard : Emerging Standards for Component Software, IEEE Computer, March, pp.68-77 (1995).

20) OMG : Common Facilities Architecture, Revision 4.0, November (1995)

21) Apple et al. : OMG RFP Submission Compound Presentation and Compound Interchange Facilities, December (1995).

22) GMD et al. : Joint Submission Mobile Agent Facility Specification, OMG TC Document cf/96-12-01 (1996).

2．2　Composer/Arranger/WebCenter

元来CASEツールは、ビジネスの要件を的確に表現することをねらった上流CASEツールとシステムの製造工程での生産性向上をねらった下流CASEツールといった形で発展してきた。いいかえれば分析ツールと設計・構築ツールが別個に存在しており、それらのツール間の連携をいかにとるかが、ツールを使ったシステム開発の成否を決める最大の要因であった。この問題を解決すべく上流から下流までシステム開発の全工程をシームレスにサポートする統合CASEツールが８０年代後半から登場し発展してきたわけであり、Composerの前身であるIEFもそうしたツールの一つとして登場した。こういったツールの登場時期がメインフレームの全盛期であったということもあって、現在ではCASEツールとか統合CASEツールといった呼び方が古いイメージを与え、敬遠される向きもある。確かに、技術の変化に対応できず姿を消すツールもあるが、IEFは着実に進化を繰り返し、最先端の技術を身にまとったComposerへと生まれ変わっている。Composerは簡単に言ってしまえばクライアント・サーバ・アプリケーションに対応した、IEFの後継製品であるが、それにとどまらず、オブジェクト指向を取り入れてのアプリケーションの部品化への対応、そしてArranger、WebCenterといったComposerをサポートする他の製品と組み合わせて、エンドユーザ・コンピューティング、インターネット／イントラネットといったような最新の技術環境に対応するアプリケーション開発環境を整えている。

ここではまずモデルをベースに開発を進めるComposerの基本的な考え方を説明し、その後、オブジェクト指向を取り入れてその考えをさらに発展させた部品化の考え方、そしてできた部品の利用の一つの形態としてのArranger、WebCenterの使い方について述べる。

2．2．1　Composerとモデル・ベース開発

(1)モデルとは

先に述べたようにCASEツールはツール相互間をいかに連携させるかを大きな課題として持っていた。具体的には上流ツールから下流ツールへいかに情報を渡すかであり、分析フェーズから設計フェーズへの円滑な移行である。分析フェーズで入力した情報が設計フェーズで生かされず再入力の必要があるとすれば、そこには常に人為的なミスの可能性、正確な仕様がシステムに反映されない危険性がつきまとう。また一方で、下流で情報を修正した場合、その情報が上流へも反映されなければ実際のシステムとドキュメンテーションとしての上流工程での成果物に食い違いを生じてしまう。こういった問題を解決するには分析情報、設計情報が一元管理される必要があり、同一の情報についてはシステム開発のどのフェーズから入力あるいは変更されるとしても同一の実体として管理される必要がある。これを実現するための仕組みがリポジトリ（Composerではエンサイクロペディアと呼んでいる）であり、このリポジトリに格納され

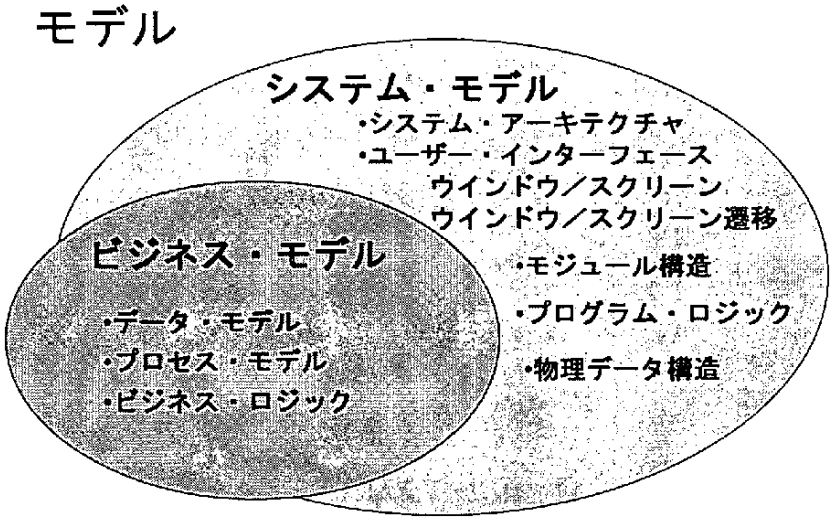


図2.2-1　モデルとは

た一群の分析情報、設計情報をモデルと呼ぶ。モデルの大きさ、カバーする領域（ビジネスの領域という意味で）については、論理的には制限はないが、通常は一つのビジネス・エリア、つまり経理システムとか人事システムといった単位で一つのモデルを構成することが多い。モデルの内容の実体は、開発フェーズの各工程に応じて分析情報、設計情報等が格納されて行くわけであるが、以下に、システム開発の工程とモデルに入力される情報という観点から、モデルというものの位置づけをまとめてみたい。

(2) Composerを使ってのシステム開発とモデル

システム開発の全工程の情報が一元管理されるということは、モデルというものがシステムに対してだけでなく、ビジネスの実現そのものに対して大きな役割を持ってくることになる。なぜなら、モデルというものが単に、アプリケーションの画面／ウインドウやモジュールの構造を表現するためのものではなく、ビジネスのアクティビィティそのものを表現することが可能なものだからである。ここで、Composerでのシステム開発の手順にしたがってモデルの役割を考えてみよう。まず上流のフェーズでは（計画のフェーズは終わっているものとして）ビジネスの中で扱うデータを捕捉することになる。それに並行してそのビジネスでのアクティビィティを洗いだし、アクティビィティの階層として整理する。この階層はビジネスとして意味のある最小ユニット（基本プロセスと呼ぶ）に分解されるまで行う。そして個々のプロセスに対して、そのプロセスで行うビジネスのロジックをプロセス・アクション図という形で記述して行くことになる。これらの分析のフェーズでの成果物が、ERD（エンティティ関係図）、AHD（アクティビィティ階層図）、PAD（プロセス・アクション図）等という形でモデルに情報として格納される。これらの上流フェーズでの作業はビジネスそのものを作業の対象物として捕えているので、基本的には実際のシステム

のハードウェアやソフトウェアのプラットフォームとは無関係に作業を進めることができる。つまりここまでの作業で出来るのはビジネスのモデルでありシステムのモデルではない。このビジネスのモデルに対してアプリケーション・システムとしてのアーキテクチャや画面／ウインドウといったプレセンテーションの方法を規定することにより実際のシステムに展開して行くのが下流の工程、すなわちシステム・モデルを作る工程となるわけである。このように、モデルはビジネス自体を表現するビジネス・モデルとそれの実現方法を規定するシステム・モデルから成立していることになるが、エンサイクロペディアにより情報が一元管理されるComposerの世界では、ビジネスのアクティビィティを忠実に実現するシステムの構築が可能となるのである。以下にモデルに格納される主な成果物をもう一度整理してみたい。

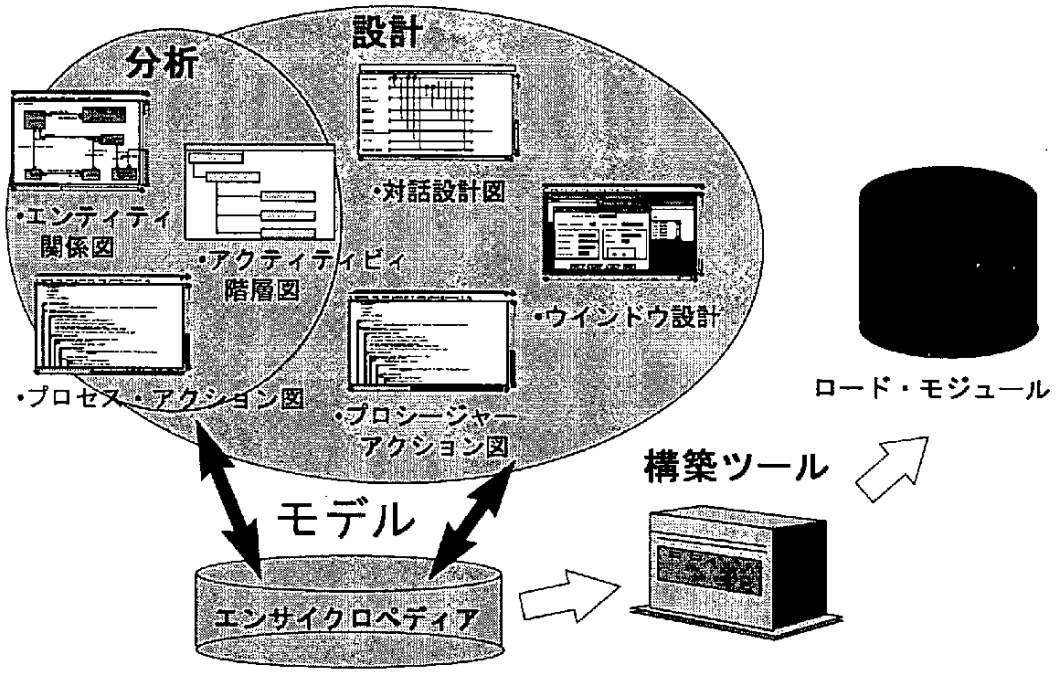


図2.2-2　システム開発：モデルから実システムへ

(a) 分析フェーズ

・ERD（エンティティ関係図）

いわゆるデータ・モデルである。当該ビジネスの中に存在するエンティティ・タイプとそれら相互間の関係を洗いだしたものである。各エンティティ・タイプにはそれの持つ属性、つまりデータ項目と識別子が定義される。

・AHD（アクティビィティ階層図）

ビジネスをそのアクティビィティの面から分析し、その最小構成要素である基本プロセスにまで分解して整理し、階層構造にまとめあげたもの。

・PAD（プロセス・アクション図）

アクティビィティ階層図で割り出された個々の基本プロセスに対して、その中で行われているビジネス・ロジック、データへの操作をアクション図という一種の疑似コードで記述したもの。

(b) 設計フェーズ

・対話設計図

ビジネスを表現する基本プロセスをシステムとして実行するための媒体としてプロシージャを作成する。プロシージャはスクリーンやウインドウを含みシステム設計を実現させるものとなる。基本プロセスはここから呼び出されて実行される形になる。対話設計図ではプロシージャを定義するとともに、スクリーンやウインドウのシステム上の遷移関係を表現する。

・スクリーン／ウインドウ設計

実際のスクリーンやウインドウの作成、フィールドの配置等。

・PrAD（プロシージャ・アクション図）

プロシージャを実際に実行するためのロジックの記述。ビジネス・プロセスの実行の他にシステムで実装したい機能のロジックを記述する。

(3) モデルから実システムへ

こうして出来上がったモデルに対し、アプリケーションのターゲット環境の指定、そして環境ごとに特有な情報を与えてやることにより、Composerの構築ツールはターゲット・アプリケーションに向けたソース・コードとデータベースの定義体を生成する。生成されたソース・コードはComposerを構成するツールの一つであるインプリメンテーション・ツールセットに渡され、ここでコンパイルおよびリンクされ、最終生成物である実行可能ロード・モジュールとなる。通信やデータベースのハンドリングのうち、アプリケーション・ロジックによらず共通に必要となる部分は、それぞれのターゲット環境に対応したランタイム・ルーチンとしてComposerにより提供され、ここの段階でリンクされる。ここでもうひとつ見落とせないことは、モデル・ベース開発においてはシステムのメインテナンスは全てモデルに対して行われるものであり、生成されたソース・コードやロード・モジュールに対してではないということである。つまり、システムに対するメインテナンスはモデル上のERDやアクション図に対して行われ、修正が行われた部分、またそのインパクトが及んだ部分はソース・コードの再生成、ロード・モジュールの再導入が行われることになるわけである。このときの影響度分析についても、Composerで行える。システム開発者はメインテナンスに際してソース・コードを一切見る必要はないわけであり、各種ダイアグラムを用いてのシステムの完全なメインテナンスが可能になる。

以上主要なダイヤグラムと作業の流れを述べたが、ツール上にはその他多くのダイヤグラムと機能が用

意されており開発作業を支援する。簡単に述べると、計画・分析時に使う各種マトリックス、アクティビティやプロセスの依存関係を表すアクティビィティ依存関係図、データ・モデリングを支援するエンティティ・ライフサイクル図やデータ・モデル・ブラウザ、データベースの物理設計やチューニング等のためのデータ構造図等がダイアグラムとしてサポートされている。またプロトタイピングの機能、テスト時のアクション図レベルでのトレース機能も備えている。さらに最新のツールでは、ウインドウの遷移関係をグラフィカルに表示するナビゲーション図、そして後ほど説明するコンポーネント・ベース開発を支援するためのコンポーネント・モデリングのためのダイヤグラムがサポートされている。

(4) Composerと開発方法論

ここまで説明してきたように、Composerではビジネスのアクティビティとその実現方法を記述したモデルから、ツールを一種のコンバーターとして実システムが生成されてくることになるわけである。ここでは話を簡単にするために全工程の流れを上から下へたどったが、これがいわゆるIE（インフォメーション・エンジニアリング）という方法論での作業の流れである。しかし今日ではこのように単純に上から下へ流れるやり方、クラシカルIEと呼ばれるやり方がそのまま使われるケースはほとんどないと言ってよい。Composerの前身であるIEFは忠実にIE、つまりトップダウンの方法を実践するツールと言われていたが、実際にはIEを基礎としながらも様々な手法を適用することが可能であり、システム開発の現場ではそれぞれのサイトに合うように柔軟に対応が図られてきた。こういったIEを発展させた方法論はいくつかあるが、これらの特徴はウォーターフォール型からスパイラル型への変化、ユーザの参加、そしてプロトタイピングの採用である。つまり、プロトタイピングを行い、ユーザの参加を得て、一定の範囲の工程を反復して行うことによりユーザの要求を正確に引出し、システムをそれに完全に合致するものとして作り上げて行く方法である。具体的な方法論としては、BSI（ビジネス・システム・インプリメンテーション）、RAD（ラピッド・アプリケーション開発）、MDR（開発リスク管理）等がある。ここでは方法論の内容を述べることが目的ではないので名前を挙げるにとどめておく。

(5) モデル・ベース開発のメリット

ここでComposerによるモデル・ベース開発の利点をまとめておきたい。

・リポジトリによるシステム開発全工程の情報の一元管理

情報の一元管理により全工程で整合性のとれたシステム開発が可能となる。またこれによりビジネスのモデル化が可能になり、それをスムーズにシステムとして実現できる。

・各種ダイアグラムの使用によるシステム開発の視覚化

ダイアグラムによる視覚化により仕様が理解しやすく明確に表現できる。またケアレスミスの排除により作業者の高効率化が図られる。

・各種要素技術、プラットフォームからの解放

ネットワーク、DBMS、OS等のプラットフォーム等のそれぞれに依存する技術から作業者が解放される。作業者は作業の重点を、システム側から、よりビジネス側へ移すことができる。

モデル・ベース開発では以上のような多くの利点を享受することができ、劇的な生産性、品質、の向上とコストの低減を図ることが可能になる。

(6) モデルとテンプレート

ここで話は少し外れるかもしれないが、モデルのシステム開発における重要性の認識を補強する意味合いで、少しテンプレートの説明をしておきたい。Composerのモデルがビジネスを表現出来るということは、いったんある分野のビジネスをモデル化できれば同種のビジネスを行う企業間でモデルの流通が可能であるということである。もちろん全く変更なしに使うケースは考えにくいが、出来上がっているモデルを雛形モデル、つまりテンプレートとして使いそれをカスタマイズして使うことが出来る。これは言ってみればパッケージと同じような考え方であるが、大きな違いがある。パッケージは基本的に実行可能モジュールで提供され、パラメーターでカスタマイズするのに比べ、テンプレートはモデルで提供され、ツールでカスタマイズを行う。つまりカスタマイズの自由度、容易さの点でテンプレート側に大きな利点があるわけである。欧米ではこのような形で多くのテンプレートが流通しており、これはモデル・ベース開発の利点を最大限に生かした方法と言えよう。

2．2．2　コンポーネント・ベース開発

ここからはこれまで説明してきたComposerでの考え方、モデル・ベース開発をさらに一歩進歩させた考え方、つまりComposerの目指している方向について述べる。これは将来の話ではなく、今現在技術的に可能なものであり、実際のシステム開発の現場で適用され始めているものである。

(1) コンポーネント・ベース開発の概要

コンポーネント・ベース開発とは非常に簡単に言ってしまえば、オブジェクト指向の考え方を取り入れて、再利用可能な部品を作り、部品によってシステムを構築して行こうというものである。ここでオブジェクト指向の考え方と言っているのはデータと手続きのカプセル化のことであり、Composerの言葉で言えば、エンティティ・タイプとプロセスをパッケージ化して部品として作りあげて行くことである。システム全体としては必要な部品を必要な場所に取り込んで使い、部品を有機的に結合させることにより完成させる。この場合、個々の部品はそのシステムに特化したものというよりは再利用可能なもの、つまり他のシステムの構築に際しても利用が可能な汎用性の高いものとして作りあげておく。こうすることによりシステム開発において以下のような様々なメリットが生まれてくる。

・生産性向上、コストの低減、サイクル・タイムの減少

システム開発においては既存の部品を最大限活用することにより、劇的な生産性の向上、開発コストの低減、システム・デリバリーのサイクル・タイムの減少が図られる。

・品質の向上

システムはすでに品質の検証された個々の部品の集合体として作ることができ、システム全体の品質も保証されたものが期待される。

・保守性の向上

システムに変更が加えられた場合、その影響範囲を部品内に限定することができるので、変更に迅速に対応できる。

・分散開発の促進部品化によりシステム開発を部品の単位で分割できる。平行開発が可能になり開発リスクを少なくできる。

・エンド・ユーザ・コンピューティングの促進

作った部品をエンド・ユーザに開放することにより、ユーザ自身が自分の求めるシステムを構築することができる。その一方、セキュリティの機能を部品に組み込んだり、部品の開放を管理することで、システム資源の濫用を防ぐこともできる。

すでにコンポーネントウェアという形でVBXやOCXの部品が流通し始めている現在、この話に新鮮味を感じないかもしれない。たしかにコンポーネントウェアを使ってGUI等のユーザ・インタフェースまわりは容易に構築可能になってきたが、ビジネス・プロセスのロジック部分になると話は簡単にはいかない。つまりここで言っている部品化は主にビジネス・プロセスの部品化、先にモデル・ベース開発で解説したモデル化されたビジネスのアクティビティーを部品化して行くことを言っているのである。別の見方をすれば、現在のコンポーネントウェアがクライアント・サーバ・システムにおけるクライアント側のアプリケーションを対象にしているのに対し、コンポーネント・ベース開発は（クライアント側を対象にしないというわけではないが）主にサーバ側のアプリケーションに焦点を置いたものと言うことができる。ここで、混乱を生じるといけないのでひとつ明確にしておきたいのは、コンポーネント・ベース開発とモデル・ベース開発との関係である。コンポーネント・ベース開発ではモデル・ベース開発をその基礎として使っており、取って代わるといった類のものではない。つまりコンポーネント・ベース開発における部品はモデル・ベースで作られるということであり、部品はComposerのモデルとして表現されるということである。ひとつのモデルでひとつの部品を表現する場合もあれば、複数の部品がひとつのモデルの中に含まれる場合もある。これは部品の大きさや用途によって決まってくることになる。

(2) 部品化の進展とその技術的背景

ソフトウェアの部品化という話は以前から繰り返されてきたテーマである。実際、システム開発の現場ではデータベースやネットワークのアクセス、その他アプリケーションで共通化できる部分をまず汎用サブルーチン化、つまり部品化して開発を進めるというのが典型的な手法であろう。しかし真に汎用的な部品、つまり一般に流通可能なような部品という意味では部品化はなかなか進展しておらず、先に述べたOCX等のGUIに関するものがようやく流通し始めた段階である。ここではComposerが実現するものをさらに明確にするために、いくつかの技術的観点から部品化への進展の背景とComposerでの考え方を述べてみたい。

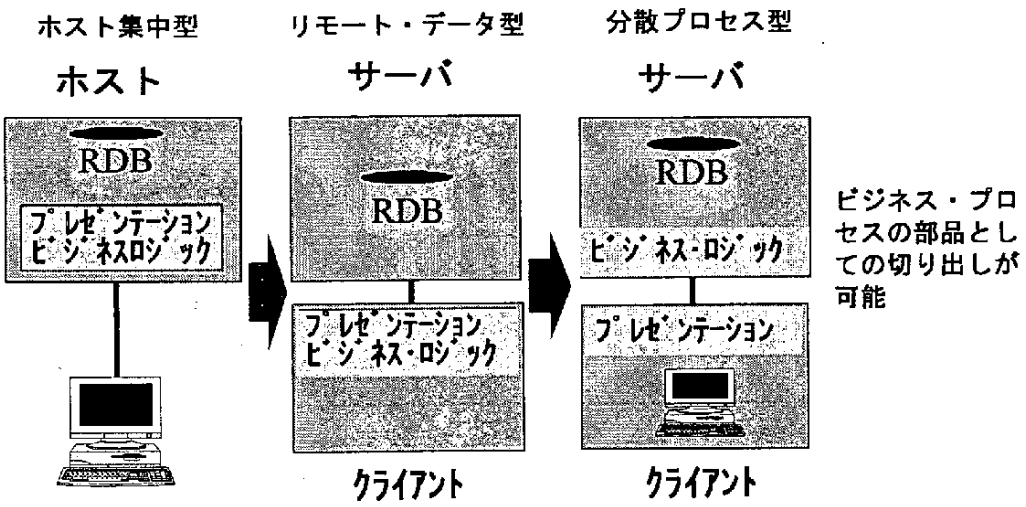


図2.2-3　クライアント・サーバ技術と部品化

(a) クライアント・サーバ技術

部品化、とくにビジネス・プロセスの部品化という意味では、近年のクライアント・サーバ技術の発展が大きな役割を果たしている。というのは、以前のホスト集中型のシステムにおいては、プログラミングにおいてユーザ・インタフェースとビジネス・プロセスのためのロジックを明確に区分けすることが難しく、したがってビジネス・プロセスとしての部品を切り出すことが困難であった。一方クライアント・サーバの技術の進化にともなって、クライアントとサーバの両方に処理ロジックを持たせること、つまり一般に言うアプリケーション・パーティショニングが可能になってきた。Composerではこの形を分散プロセス型と呼んでいるが、これによりクライアントはユーザ・インタフェースに特化しサーバはビジネス・プロセスの処理に特化するということが可能になってきた。特に大規模なクライアント・サーバ・システムではネットワーク上のトラフィックを減らし、システムのパフォーマンスを確保するために分散プロセス型のシステムが必須となる。Composerではリモート・データ型、つまりサーバにはデータベース・アクセス

のみが存在する形態のサポートもするが、今後のクライアント・サーバは分散プロセス型をその主流ととらえ、これをベースにシステムの部品化を図って行く。

(b) オブジェクト指向

データと手続きをカプセル化して再利用するというのはオブジェクト指向の大きな目的としてのひとつである。この観点からすでにC＋＋等のオブジェクト指向言語ではクラス・ライブラリーを作ることによってプログラミングでの生産性を上げることに成功している。くかし一方でこの恩恵を受けることのできる人々は一部のITの専門家に限られており、システム全体に対しての生産性と言う意味では大きく貢献できるところまではいたっていない。もっと一般に利用可能な部品、つまりユーザに見える形の部品を提供する必要があるという意味でOCX等のコンポーネントウェアが発展してきたわけである。確かにこういったコンポーネントウェアはユーザ・インタフェースの構築に威力を発揮するが、ビジネス・プロセスの部品を提供しているとは言い難い。なぜなら、コンポーネントウェア上でもデータベース・アクセスの手段等は部品として提供されるが、これは部品と呼ぶには粒が小さすぎ、プログラミング用の部品とは言えてもビジネス・プロセスの部品とは言えないからである。対して、コンポーネント・ベース開発で言っている部品はある一定の範囲のビジネス・プロセスを部品化したもの、別の言い方をすれば仕様のレベルでの部品と言うことができる。また、オブジェクト指向の言葉で言うならビジネス・プロセスのクラス・ライブラリー化を図ると言うことになる。この点から言えばコンポーネント・ベース開発が実現しているのは各種のオブジェクト指向分析／設計手法が目指していることと同じと言えるが、残念ながら現在のオブジェクト指向方法論はいまだ未成熟であり、一般のシステム開発に適用するのは難しいと言わざるをえない。コンポーネント・ベース開発は実証された技術をベースにしてオブジェクト指向の利点を最大限取り入れたものと言うことができる。

(c) リポジトリ技術

モデル・ベース開発の説明で述べたように、リポジトリ（エンサイクロペディア）はモデルをベースとして考える場合の技術の中核である。これは部品化においても同様で、特にビジネス・プロセスの部品化を考えたときに必須の技術である。GUIで使用する部品と異なり、ビジネス・プロセスの部品としての仕様をデータ・モデルやプロセス・モデルで表現しそれを管理する必要があるからである。さらに、部品化では出来た部品を管理する仕組み、つまり部品ライブラリを提供し部品の仕様とともにモジュールとしての実体のありかを示すということが必要になってくる。残念ながら今迄のリポジトリは分析・設計の情報、つまりシステムを作るための情報のみに主眼がおかれ、出来上がった実体の管理という点では注意が払われていなかった。Composerでは現在、後ほど説明するArrangerで部品の登録と管理の機能を提供しているが、今後はリポジトリ自体に部品の管理機能を持たせる方向で新しいリポジトリの開発を進めている。

(3) 部品の実体とコンポーネント・ベース・システム

ここまでの議論でコンポーネント・ベース開発で言っている部品というものがビジネス・プロセスを部品化たものであり、システム全体はライブラリー化された部品群を利用して組み立てられて行くものであることが分ったと思う。ここからは部品というものがどういったものなのか、またどのような形で利用されて行くものなのかを見てみたい。部品の実体を明らかにするには論理的な側面、つまり今迄述べてきた部品のビジネス・プロセスとしての表現をもう少し説明するとともに、物理的な側面、すなわち部品がComposer上のモデルでどう表現されるかを説明する必要がある。

(a) 部品とは－論理的側面－

ビジネス・プロセスを部品化することを考えた場合に最も難しいのはどういうくくり、どれぐらいの大きさで部品を作るかである。部品があまり小さくては単なるデータ検索や更新のためのSQL文の集合体になってしまいビジネスとして意味のあるプロセスを形成することが出来ない。ビジネス・プロセスの部品としては少なくともその部品の実行がビジネス上の行為を意味する一連のデータの操作とビジネス・ルールを含んでいる必要がある。一方、部品が大き過ぎる場合も問題が生じてしまう。特定のアプリケーションに特化したものが入り込んでしまうために、再利用することが困難になってしまうからである。では具体的にはどの程度の大きさにしたらよいのか。これはビジネスの性質や部品の用途に依存するので、それらを考慮せずに答えを出すことは難しいが、少なくとも一つのビジネス上の行為として完結したもの、もう少しシステム的に言えばデータベースのコミットの単位となるべきものと言えよう。

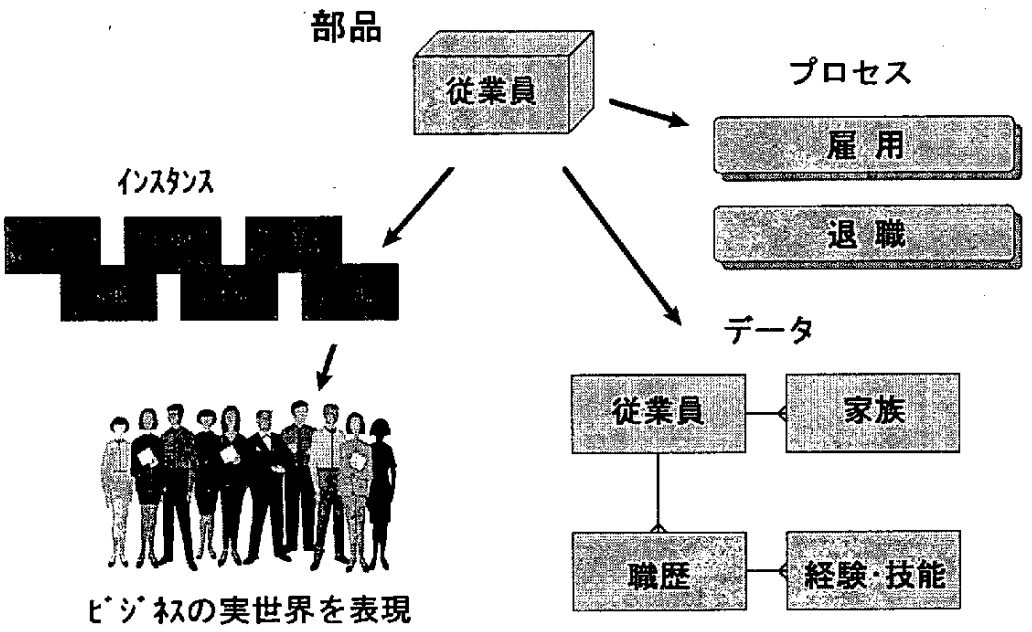


図2.2-4　部品とは （1）

例えば「従業員」という部品を考えてみよう。図2.2-4を参照していただきたい。この部品自体のビジネス上の意味としては各従業員の実体とそれに関係するビジネス上のアクティビィティを表現することになるわけであるが、モデルの概念から見るとエンティティ関係図で表現されるデータの部分と、このデータにアクセスしビジネス・ロジックを実行するいくつかのプロセスが存在することになる。データに関して言えば、従業員本人を表すエンティティ・タイプのほかに「家族」とか「職歴」などのエンティティ・タイプが従業員と関係を張ることになる。プロセスとしては「雇用」、「退職」などがまず考えられると思うが、例えば雇用が行われた場合、従業員本人のエンティティがCREATEされるだけでなく、その従業員に関して家族や職歴のエンティティもCREATEされてからコミットがかかることになる。実際のビジネス上はもっと多くのデータへのアクセスや複雑なロジックの実行が起こると思われるが、とにかく、こういったビジネス上意味のある一連の処理がプロセスとして部品の中に存在することになる。つまり、論理的な見方から部品をひと言でいうならば、部品とはビジネス上密接な関係を持った一連のエンティティ・タイプとプロセスをデータ・モデルとプロセス・モデルのレベルでカプセル化したものと言うことができる。

(b) 部品とは－物理的側面－

では次に部品をもう少し物理的な側面、つまりComposerに即した立場から部品がどう表現されるかを見てみたい。先に述べたように部品はデータ・モデルとプロセス・モデルをカプセル化したものなので、ツール上でも同じように表現される。つまりいくつかの密接な関係を持つエンティティ・タイプをピックアップしたデータ・モデルと、そのデータ・モデルにアクセスするプロセス群をカプセル化して部品として定義する機能がツールに備わっている。ただしこれだけでは部品としては十分ではない。部品が部品として広く再利用されて行くためには部品の仕様が的確に表現されていなければならないからである。これを可能にするためにComposerではスペシフィケーション・モデルと言う概念を導入しここで部品の仕様を記述する。このスペシフィケーション・モデルはスペシフィケーション・タイプ・モデルとパブリック・オペレーションから成っている。（図2.2-5参照）スペシフィケーション・タイプ・モデルは簡単に言ってしまえば部品を作るときに定義したデータ・モデルであるが、通常のデータ・モデルとの違いは、スペシフィケーション・タイプ・モデルは実際のデータベース等の導入には使われないということである。仕様を記述することだけを目的としたものだからである。パブリック・オペレーションは部品の入出力と処理内容を記述したものである。これも部品の中の実際の処理（プロセス）と異なりアクション図による実際の処理ロジックは含まれていない。使用者が部品の使い方とその振る舞いを理解できる必要十分な情報があれば実際の処理ロジックを知る必要はないからである。一方、部品を実際に実行可能モジュールやデータベースとして導入する際に使われるモデルをインプリメンテーション・モデルと呼ぶ。これは今迄我々が単にモデルとよんでいたものと同一であるが、スペシフィケーション・モデルと明確に区別するためにこう呼

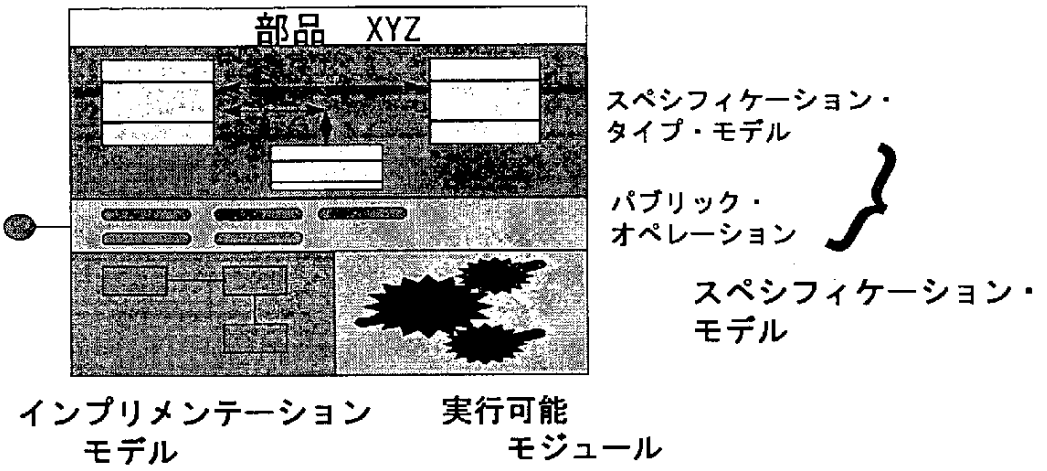


図2.2-5　部品とは（2）

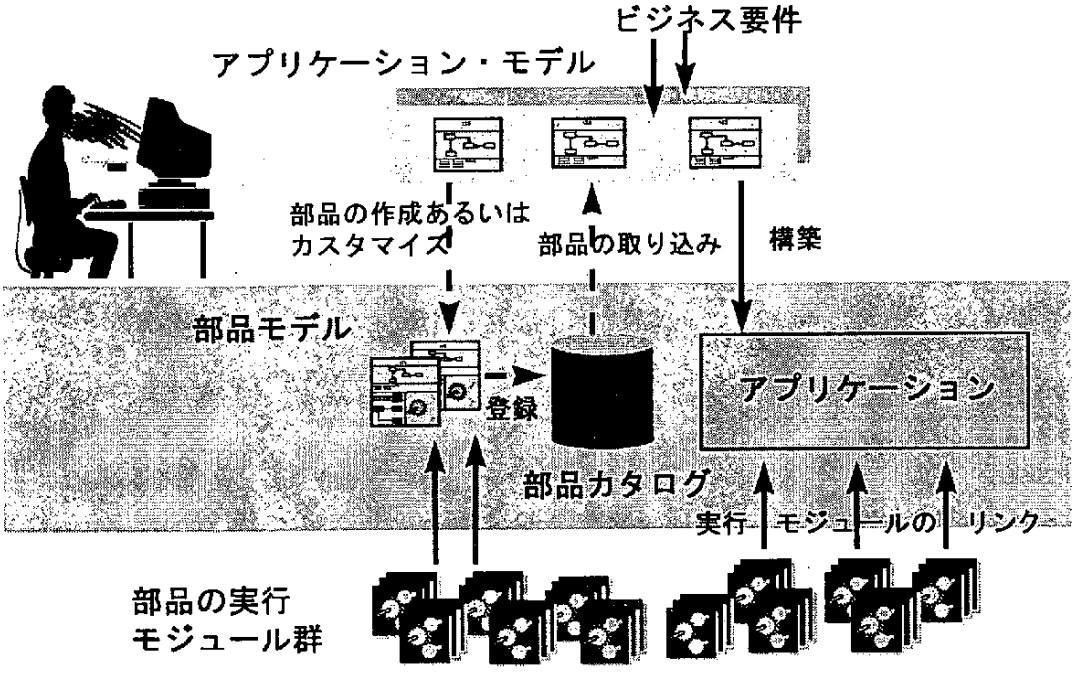


図2.2-6　アプリケーションの組み立て

ぶ。インプリメンテーション・モデルには処理ロジックの実体であるアクション図やデータベースの物理構造が含まれ、データベースの定義体やソース・コードを生成するための情報が含まれている。つまり部品の中には部品を実体として導入するためのインプリメンテーション・モデルとその仕様を明確にするためのスペシフィケーション・モデルが存在する。そして再利用可能な部品として部品を公開するには、インプリメンテーション・モデルだけでなくスペシフィケーション・モデルを記述しておく必要があるということである。部品の最後の構成要素である実行可能モジュールはインプリメンテーション・モデルを基

にして導入される。出来上がった部品は部品モデルとして公開され部品カタログに登録される。こうして実際に再利用可能な部品が出来てくることになる。

(c) 部品を使ってのアプリケーションの組み立て

では次に部品をどう開発中のアプリケーションに組み込んで行くかを考えたい。この手順をごく簡単に言うと開発中のモデル、つまりアプリケーション・モデルから部品を呼び出して使うということになる。出来上がっている部品モデルは部品カタログに登録されている。ユーザはビジネス要件に合致する部品モデルをそこから探し、アプリケーション・モデルに必要な部品のスペシフィケーション・モデルをコピーして取り込んで来る。そしてアプリケーション・モデルの中の必要な場所でパブリック・オペレーションを呼び出して使うということになる。部品のモジュールはアプリケーション・モジュールのリンク時に自動的に組み込まれることになる。このように部品をそのまま使う場合はスペシフィケーション・モデルと実行モジュールのみあればよい。一般に流通する部品は、スペシフィケーション・モデルと実行モジュールのみが提供され、インプリメンテーション・モデルは提供されない形態になる。部品とは通常、変形させて使うものではないからである。しかし一方で部品をカスタマイズして使いたい場合もある。このときはインプリメンテーション・モデルも提供される必要がある。これは単なる部品というよりは、テンプレート部品と呼ぶべきかもしれない。部品をカスタマイズして使う場合は、インプリメンテーション・モデルごとコピーして取り込んでカスタマイズして使うか、あるいは部品の段階で部品のモデル自体をカスタマイズしてからそのスペシフィケーション・モデルを取り込んで使うことになるが、部品化という観点からは後者のほうが望ましいであろう。また部品の取り込みとは逆に、作成しているアプリケーション・モデルの一部を切り出して部品を作って行くことも可能である。必要な部品がない場合にはこのような形になりえるが、こういった場合、たとえ再利用が期待出来ない場合であっても部品化しておくことが望ましい。保守性の向上、分散開発の促進等の面で多くのメリットが再利用以外にも期待できるからである。ここでもう一度部品の大きさについて言及しておきたい。今迄の議論では、部品の大きさについてあまり柔軟性がないような印象を与えたかもしれないが、実はかなり自由に設定する事ができる。というのは、部品自体をもっと小さな部品の集合体として作ったり、組み合わせて大きな部品を作ることができるからである。この場合、出来上がった部品の使用者は、最終部品の仕様だけが分かればよく、個々の構成部品について知っている必要はない。このように部品を入れ子の形態で構成することが可能なので、様々な大きさ、用途の部品を作り出すことができるのである。ここでもう一度コンポーネント・ベース開発で言う部品の概念を以下に整理しておきたいと思う。

・部品とはビジネス・プロセスを再利用可能な部品にしたものである。

・部品には部品の扱うビジネスを表現するデータ・モデルとプロセス・モデルがカプセル化されている。

・部品はスペシフィケーション・モデルとインプリメンテーション・モデル、そして実行可能モジュールからなる。スペシフィケーション・モデルは仕様の記述のために、インプリメンテーション・モデルは部品の実体の導入のために使われる。

・通常の部品はスペシフィケーション・モデルと実行可能モジュールで配布される。カスタマイズが必要な場合はインプリメンテーション・モデルも必要となる。

・部品は部品モデルの形でアプリケーション・モデルの中に取り込んで再利用する。

・部品はさらにいくつか組み合わせて新しい部品を作ることができる。

2．2．3 部品の利用

ここまでComposerで言う部品がどういうものなのか、どのようにアプリケーションに組み込んで行くものなのかを説明してきた。当然、部品の再利用の仕方で大多数を占めるのは、ここまで説明してきたようなやり方、つまり新規アプリケーション・システムの開発時に部品を組み込んで行くやり方であろう。しかし部品の利用の仕方はこれにとどまらず、異なった形態もある。Composerで作成したクライアント・サーバ・アプリケーションは、前にも説明した通り基本的には分散プロセス型である。この形態の場合、インタフェースを合わせればサーバのアプリケーションを別のクライアントから利用することが可能である。つまり、部品によって組み立てられたサーバ・アプリケーションのモジュール全体を部品とみなし、再利用することができる。これは今迄説明してきた部品の再利用とレベルが異なり、言ってみればシステム・アーキテクチャのレベルでの再利用と言える。同一サーバ・アプリケーションを利用しながらも、クライアントをデスクトップ・ツールにしたりWebブラウザーに替えたりと、システム・アーキテクチャを替えることが出来るからである。ここでは、部品の利用の応用形態として、Arranger、WebCenter等、Composerを支援する他のツールと組み合わた形について述べる。

(1) Arrangerとエンド・ユーザ・コンピューティング

企業システムのデータをエンド・ユーザに開放する際に最も気を配らなければならないのはセキュリティとシステム資源の浪費を管理することである。セキュリティは当然守られるとしても、悪意のないユーザが大量にシステム資源を浪費してしまうSQL文等を実行してしまうことを防ぐのは難しい。一方、デスクトップ上の部品という形で企業システムにアクセスする手段を提供すればシステム資源の浪費を防ぎつつ、ユーザ・フレンドリーなアプリケーション構築環境をデスクトップ上で提供することができる。Arrangerはこういった環境作りを目的として用意されたツールである。具体的には、ArrangerはComposerで作成したサーバ・アプリケーションをOLE対応のデスクトップ・ツール、例えばExcel、Word、Visual Basic等からアクセスする手段を提供する一種のミドルウェアと言える。これによって、全社規模のデータベースか

らデータを読みだし、Excel等自分のデスクトップ上のツールに展開し自由に加工することが可能になる。あるいはVisual Basicで自分の好みのクライアント・アプリケーションを作り、それをComposerで作成したクライアントの替わりに使うこともできる。ここで言うサーバ・アプリケーションは必ずしもコンポーネント・ベース開発に準拠したものである必要はなく、Composerで作成したものであればよい。

図2.2-7にArrangerを使う場合の作業手順の概略を示す。

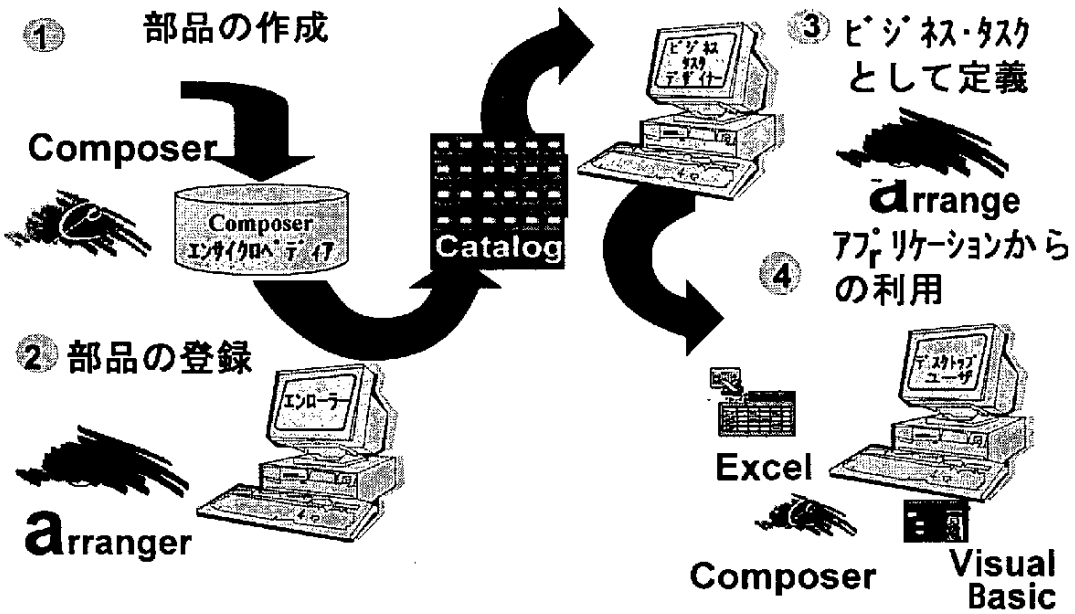


図2.2-7　Arrangerのしくみ

Arrangerで使うサーバ・モジュールの情報はComposerのエンサイクロペディアの中に格納されている。Arrangerの中のエンローラーというツールがエンサイクロペディアを読み出しその情報を表示する。ここでユーザに開放するサーバ・アプリケーションをArrangerのカタログに登録する。ここまではシステム管理者の役割となる。実際のエンド・ユーザはArrangerのツール上からカタログに登録されたサーバ・アプリケーションで自分が使うものをビジネス・タスクとして定義する。これによってビジネス・タスク・ファイルが作られる。ビジネス・タスク・ファイルはサーバ・アプリケーションの入出力情報が記述されたもので、エンド・ユーザのマシンに配布され、Arrangerのタイム・ルーチンがこのファイルを見てデスクトップ・ツールとOLEインタフェースによって連携をとる仕組みになっている。

(2) WebCenterとインターネット／イントラネット

WebCenterも製品の位置づけとしてはArrangerと同じであり、インターネットあるいはイントラネット上のWebブラウザーからComposerで作成したサーバ・アプリケーションを使うためのミドルウェアとしての機能を提供する。これによってComposerで作成された企業システムを簡単にインターネットやイントラネッ

トに展開することができるわけである。Arrangerの場合と同様で、サーバ・アプリケーションに手を加える必要はない。図2.2-8にWebCenterの仕組みの概略を示す。

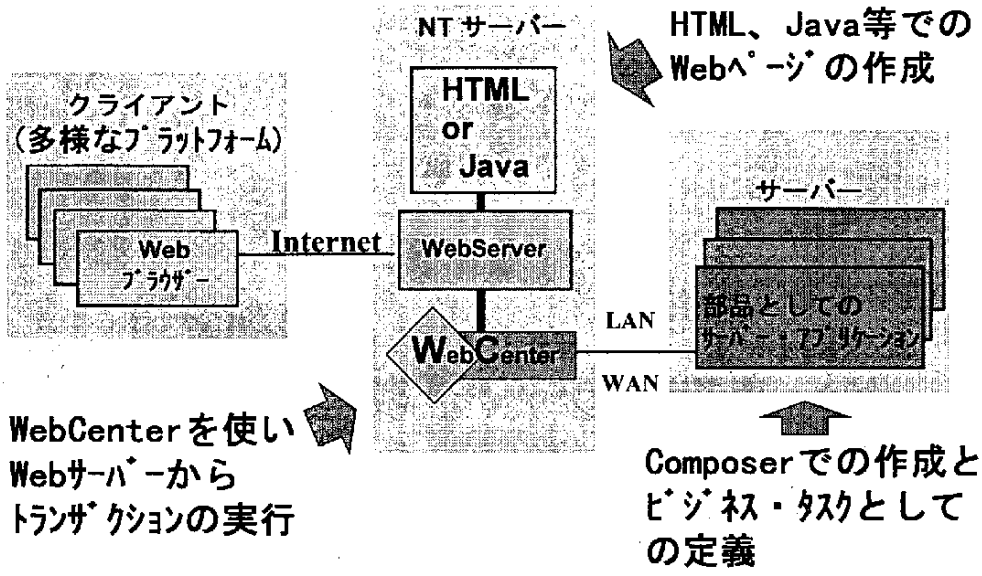


図2.2-8　WebCenterの仕組み

WebCenter自身はWebサーバ上で動くことになる。Webブラウザーから出されたサーバ・アプリケーション実行の要求はインターネットを介してWebサーバに渡り、WebサーバからCGIインタフェースを介してWebCenterに渡る。WebCenterは要求の内容を判断し、該当するサーバ・アプリケーションをLANあるいはWAN経由で実行し、実行結果を受け取る。Webサーバ管理者はCGIインタフェースを用い実行結果を新しいWebページに展開するプログラムを用意しておき、これがWebCenterによって起動されることによりWebブラウザーに実行結果が返ることになる。WebCenterとサーバ・アプリケーションの間の接続にはArrangerと同じインタフェースが使われるので、Arrangerの場合と同じ手順でつくられたビジネス・タスク・ファイルがWebサーバ上に配布されている必要がある。このようにComposerで作成されたサーバ・アプリケーションは全く内容を変更することなく、Composerで作成したクライアントのみならず、各種のデスクトップ・ツールやインターネット／イントラネットから利用することが可能になるわけである。どの形態にするのが最適かは、当然アプリケーションの要件によって異なってくるであろう。大量の入出力やパフォーマンスが要求される場合はComposerで作成したクライアントが最適となるであろうし、自由なデータの加工やレポーティング形式が必要ならArrangerを使ってのデスクトップ・ツールとの連携が最適であろう。また、全社的あるいは社外にもユーザが想定されるようなグローバルなものはWebCenterを使ったWebブ

ラウザーをクライアントにする形式が最もよいと考えられる。いずれにせよシステム開発を支援するツールにとって、このように柔軟にユーザの要求に答えられることは重要なポイントである。

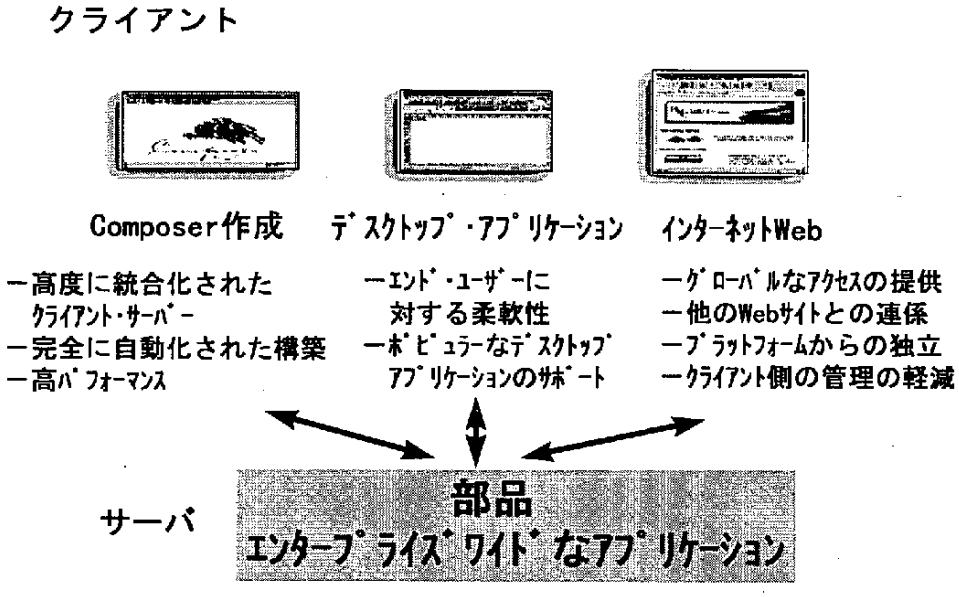


図2.2-9　アーキテクチャ・レベルでの部品の利用

2．2．4　まとめ

以上Composer/Arranger/WebCenterということで、Composerでのモデル・ベース開発の考え方、それを部品化という観点から発展させたコンポーネント・ベース開発、そして部品の利用の形態の例としてArranger、WebCenterによるデスクトップ、インターネット／イントラネットの企業システムとの統合方法を述べた。システム開発におけるモデル、部品化の重要性と有効性が理解できたものと思う。ここではComposerの機能面からの紹介が中心になったが、ツールにかかわらず一般のシステム開発という立場でも、共通することがらは多いであろう。

3．開発方法論

3．開発方法論

3．1 概 説

コンポーネントウェアによるアプリケーション開発がエンドユーザを中心に実践されつあるが、コンポーネントウェアが開発途上の技術であるため、その開発方法論もまだ確立されていない[1]。しかし、コンポーネントウェアが提示する様々な新しい概念や技術は、開発方法論においても新たな枠組みを必要としている。ここでは、コンポーネントウェアによる開発方法論のあり方とその現状を概観する。

(1) コンポーネントウェアの示唆するアプリケーション像

コンポーネントウェアの提示する技術をアプリケーション開発の視点から整理し、コンポーネントウェアによるアプリケーション像を示す[3]。

(a)文書指向／タスク指向：ユーザ指向

多くのコンポーネントウェアでは、複合文書アーキテクチャを採用し、アプリケーションは広義の文書として表わされる。例えば、Webのホームページは典型的な複合文書である。さらに、複合文書の意味として、業務などのタスクの記述、すなわち、なすべきこと(What)の記述として理解できる。これらのアーキテクチャは従来のアプリケーションアーキテクチャとは根本的に異なる。アプリケーションをユーザの視点から設計することを示唆している。これは、エンドユーザコンピューティングに適している。

(b) プラグ＆プレイ型部品の組立て型開発

コンパイルすることなくプラグ＆プレイできるコンポーネントの組立てによってアプリケーションの構築が可能となる。

(c) アーキテクチャ指向

多くのコンポーネントウェアでは、部品を組立てるための核となるアーキテクチャをフレームワークとして提供している。従って、コンポーネントウェアではあらかじめアーキテクチャを設計したり、コンポーネントウェアに付随するアーキテクチャの上でアプリケーションを構築するアーキテクチャ指向の開発となる。

(d) ネットワーク分散アーキテクチャ：分散コンポーネントウェア

WWW、分散オブジェクト環境、Javaなどのネットワーク上でアプリケーションを構築する基盤技術の発展により、ネットワーク上に分散したコンポーネントが連携してアプリケーションを実現できる。Webの文書にアプレットなどの部品を組み込む新しいアプリケーションアーキテクチャが実現する.

(e) ビジュアルプログラミング

部品単位で、操作、連結などを視覚的に行えるので、ハードウェア部品のようにソフトウェアが直接操作可能となる。

これらの点から、コンポーネントによりアプリケーション開発方法は次のように変わると考えられる。特に、開発の視点が開発者からユーザへ移ることに留意すべきである。

表3.1-1 コンポーネントによる開発パラダイム

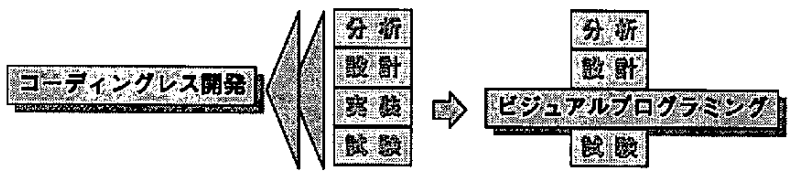
比較項目	従来型開発パラダイム（オブジェクト指向）	コンポーネントウェアによる開発パラダイム
開発の視点	開発者	ユーザ
構成要素	オブジェクト	コンポーネント
再利用の対象	実装	サービス（インタフェース）
アーキテクチャ	任意	フレームワークとして所与
開発プロセス	新規	組み立て型

(f) 部品開発と部品組立てへの開発組織の分離

コンポーネントにより部品開発と部品組立てが分離し、新たなソフトウェア開発形態が生まれようとしている。これは、ソフトウェア開発の組織や方法に大きな影響を及ぼす可能性がある。

(2) コンポーネントウェアによる開発形態

コンポーネントウェアによるアプリケーション開発の形態は、図3.1-1に示す2つに大別できる。



	エンドユーザ開発	組立て型アプリケーション開発
開発形態	エンドユーザ開発	組立て型アプリケーション開発
パラダイム	コーディングレス開発	ビジュアルプログラミング
開発者	エンドユーザ・SI・SE	プログラマ（専門家）
適用工程	全工程	設計ー実装（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	視覚的な開発支援環境上で部品やフレームワークを組み合せてアプリケーションを構築
部品	中ー大（多機能）	小ー中（単機能）
適用範囲	OAなど比較的限定	限定なし

図3.1-1 コンポーネントウェアによる開発形態

(3) エンドユーザ開発

コンポーネントウェアの典型的な適用領域はエンドユーザによる組立て型開発である。これを、エンド

ユーザ開発と呼ぶ。この開発形態の特徴は、プログラミングの非専門家であるエンドユーザがCOTS(Commercial Off-The-Shelf)ソフトウェアを組み合せてプリケーションを構築する点にある。ウィンドウズの上で、市販ソフトウェアを組み合せて中小規模の業務支援ソフトウェアを構築した例が少なくない。

エンドユーザ開発を実現するためには、次の点で、従来の専門家による開発とは異なる方法や支援が必要となる。

(a) プログラムミングレス／レスプログラミング：エンドユーザやシステムインテグレータなどプログラミングの非専門家が中心となる。

(b) ユーザ指向：エンドユーザの視点に立って考えるので、システムがユーザに提供すべき事項やシナリオなどのユーザとシステムの相互作用が重視される。

このような観点から、コンポーネントウェアをベースとするユーザ指向の開発方法が提案されている[2, 3, 8]。これらに共通する特徴は、ユーザから見た業務とアプリケーションモデルを一体化している点にある。[2]では、WYSIWYD(What You See Is What You Do)と呼び、視覚的な部品の組み合せが業務と対応づられるモデルを提示している。また、[8]では、業務モデル≡計算モデル（業務モデルと計算モデルの一致）、分析・設計・プログラミングの一体化を提案している。

さらに、HCI(Human-Computer Interaction)などの分野を中心にユーザ指向設計(User-Oriented Design)やシナリオベース設計(Scenario-Based Design)と呼ばれる、ユーザの視点からの開発方法論の研究が展開されている[7]。

(4) 組立て型アプリケーション開発

業務アプリケーションなどの比較的大規模なアプリケーション開発をコンポーネントウェアで開発する形態である。この形態の開発方法論のベースはオブジェクト指向開発方法論である。しかし、コンポーネントウェアによる開発に適用するためには、次の２点で問題があるため、そのまま適用することは適切ではない。

(i) 部品再利用を前提とする方法論となっていない.

(ii) 分析・設計の基礎単位がオブジェクトであり、コンポーネントの視点が提供されていない。

このため、コンポーネントウェアの特性を考慮してオブジェクト指向開発方法論を改良した方法論が提案されているが、必ずしも方法論としてまとまっているとは言えない。以下では、このような方法論の現状を、開発プロセス、分析・設計方法論の両面で紹介する。

(5) 開発プロセス

コンポーネントウェアにより業務アプリケーションなどのある程度の規模と複雑度をもつアプリケーションを開発するための開発プロセスが提案されている。これらの開発プロセスは、従来の開発プロセスをコ

ンポーネントウェアの視点から改良したものが中心である。

(a) CBSE(Component-Based Software Engineering)開発プロセス

アンダーセンコンサルティングの提案するコンポーネントウェアによる開発プロセスである。コンポーネントウェアの特性を考慮し、従来のウォータフォールモデルにコンポーネント分析、アーキテクチャ設計、コンポーネント製造・組立てなどのプロセスを付加したモデルである[13]。開発プロセスの概形を図3.1-2に示す。CORBAのIDLを拡張したADL(Architecture Description Language)とそれを支援する視覚的なアーキテクチャ設計支援環境を提案している。

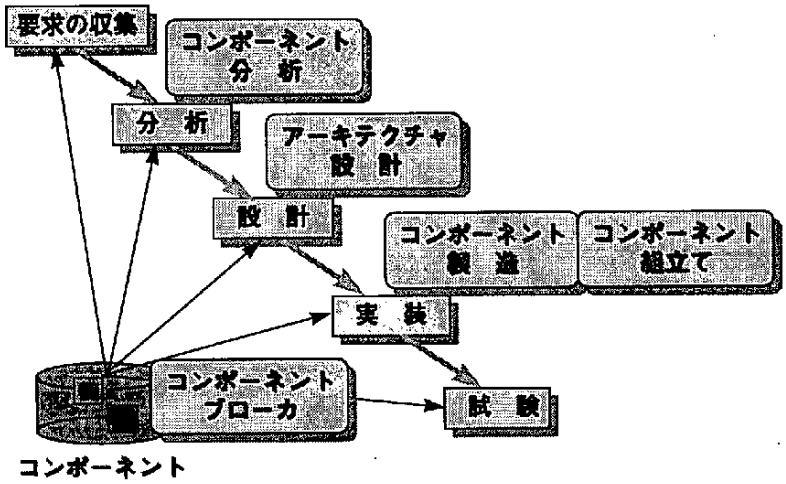


図3.1-2 CBSE開発プロセス

(b) APPGALLERY開発プロセス

ウィンドウズのOLE上で視覚的に部品を組立ててアプリケーションを開発するAPPGALLERYを利用する開発方法論である[10]。開発プロセスの概形を図3.1-3に示す。前半の画面を中心とする設計段階はスパイラルプロセスとしている。後半の既存部品の組立てと個別部品の開発組立てが並行するプロセスとし、コンポーネントウェアによる開発プロセスとして具体的な指針を与える。

(c) 部品選択プロセスモデル

部品の選択において考慮すべき主要な課題の一つがアーキテクチャミスマッチ[9, 16]である。これを考慮した部品組立て型開発のプロセスが提案されている。たとえば、図3.1-4に示すように、部品の選択と組み込みプロセスの参照モデルが提案されている[6]。

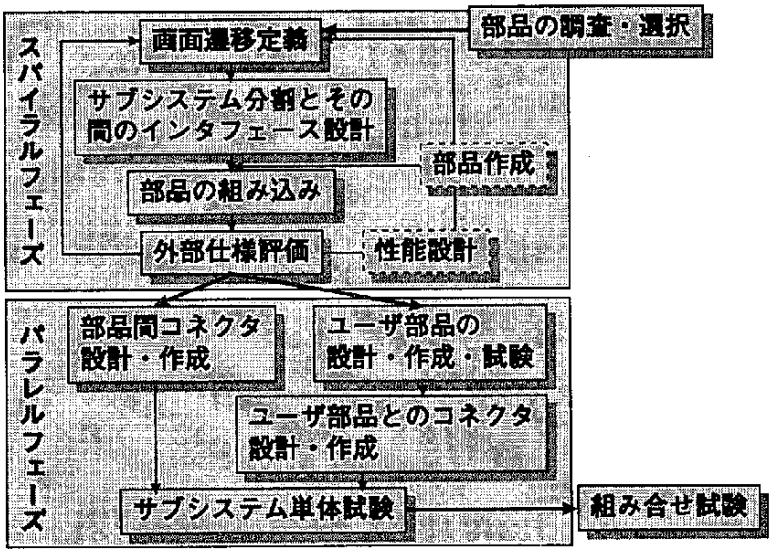


図3.1-3　APPGALLERY開発プロセス

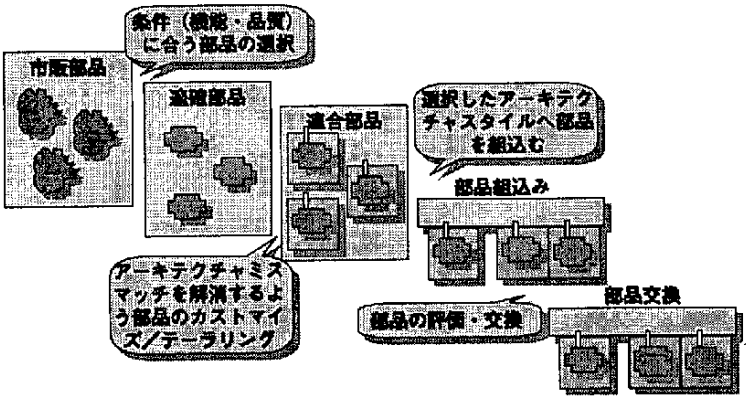


図3.1.4　部品選択プロセス

(d) 工程毎の工数配分

コンポーネントウェアによる開発事例における工程毎の工数配分を図3.1-5に示す。事例数が少ないので結論づけることはできないが、従来の開発と異なる２つの共通点がある。

(i) 部品開発（調達やカストマイズを含む）工程が新たに必要となる[11]。

(ii) 試験工数の比率が大幅に低下している。

(6) 分析・設計方法論

コンポーネントを組立ててアプリケーションを開発するためには、図3.1-6に示すように、分析・設計方法論も、部品開発と部品を組立てたアプリケーション開発に分けて考える必要がある。

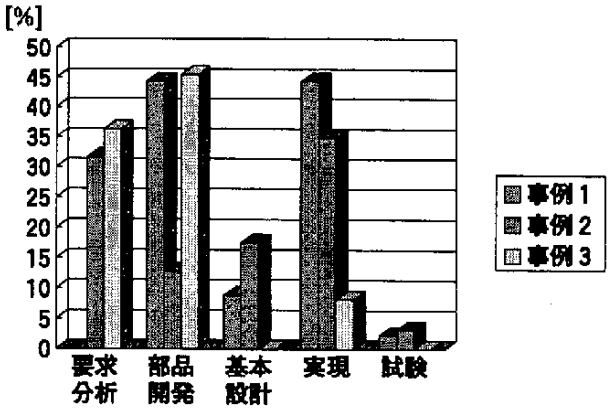


図3.1-5 事例に見る工程毎の工数比率

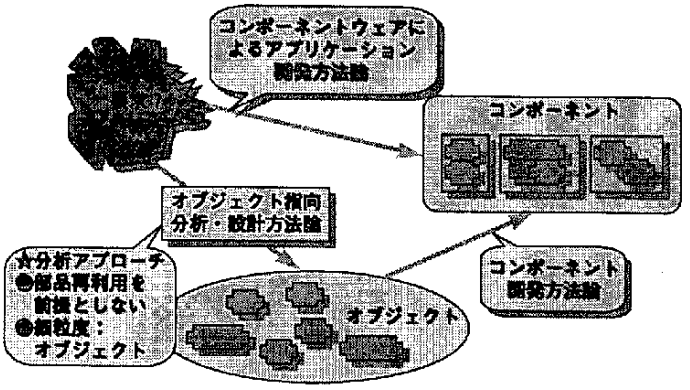


図3.1-6 コンポーネントウェアの開発方法論

(a) VMT(Visual Modeling Technique)

VisualAgeを用いたコンポーネントウェアの開発方法論として、VMT(Visual Modeling Technique)が提案されている[15]。方法論の全体像を図3.1-7に示す。

分析モデルは、OMT法をベースとしているがシナリオ抽出のためにユースケースを用いる。さらに、部品の組立てにおいてはクラス間の集団的な関係(Collaboration)を理解する必要があるのでRDD (Responsibility-Driven Design)[16]を導入している。これらの分析結果が、VisualAge上での開発に引き継げるようなプロセスをとる。

(7) コンポーネントウェアによる開発企業モデル

コンポーネントウェアにより、ソフトウェア開発が、部品開発と部品の組立てによるアプリケーション開発に分離すると期待されている。さらに、このような分離はこの間での部品の流通という新たなプロセスを創り出している。特に、コンポーネントウェアの場合、流通はネットワーク上で行える。また、流通

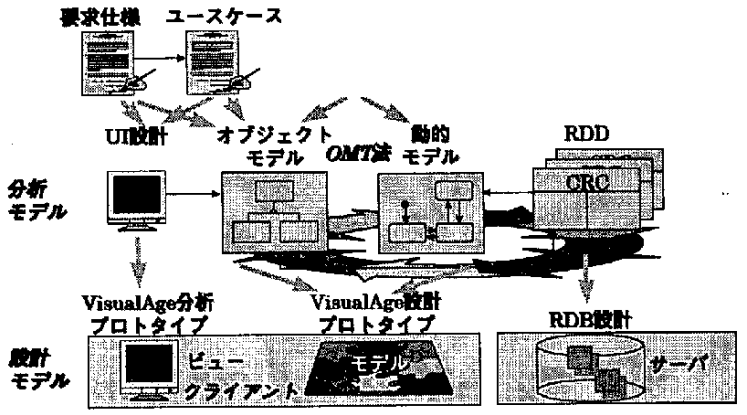


図3.1-7　VMTのプロセス

は単にソフトウェアの販売に留まらず、様々な情報提供の役割も期待される。この観点から、コンポーネントウェアによるソフトウェア開発組織あるいはソフトウェア産業モデルの共通モデルとして図3.1-8に示す「ベンダ・ブローカ・インテグレータモデル」が提案されている[14,4]。実際に、図3.1-9に示すように、Web上で部品の販売や部品情報の提供が行われている。

ソフトウェアCALSプロジェクト[12]の次世代開発環境の実証・実験ではこのようなネットワーク上での部品組立て型ソフトウェア開発を検討している。

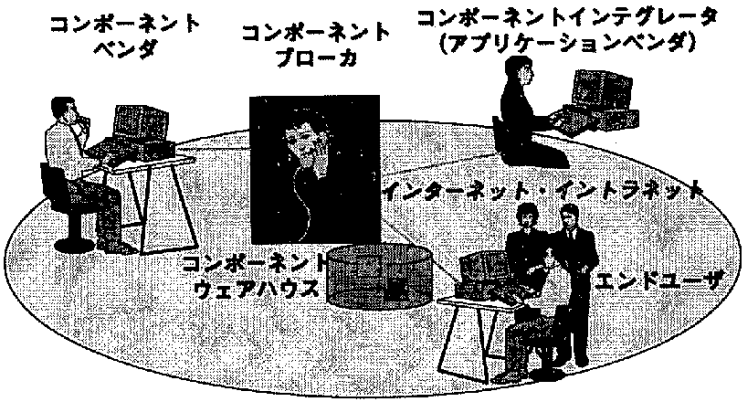


図3.1.8　コンポーネントウェアの企業モデル

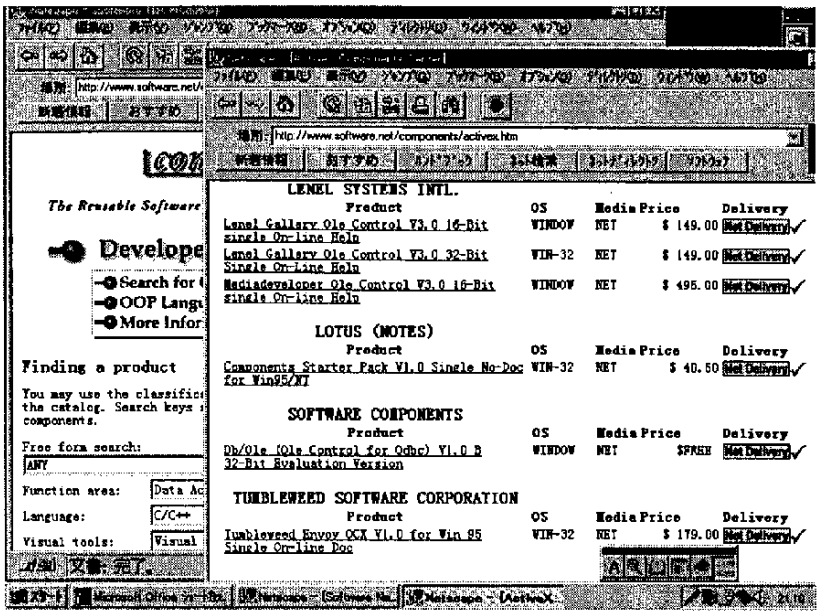


図3.1.9 Web上のコンポーネントブローカの例

【参考文献】

1) 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, Jan. 1996, pp. 71-79.

2) 青山幹雄：部品組立て型ソフトウェアプロセスモデル、日本ソフトウェア科学会ソフトウェアの新しい構成・統合技術ワークショップ、Mar. 1996, pp. 79-82.

3) 青山幹雄：コンポーネント指向ソフトウェア開発方法論、情報処理学会ソフトウェア工学研究会資料、No.111-5, Sep. 1996, pp. 33-40.

4)青山幹雄：コンポーネントウェア：ソフトウェアCALSのめざす次世代ソフトウェア開発像、JISA会報、No. 44, Dec. 1996, pp. 58-69.

5) G. Booch and J. Rumbaugh： Unified Method for Object-Oriented Development, Ver. 0.8, Rational Software, 1995.

6) A. W. Brown： Component-Based Software Engineering, IEEE CS Press, 1996.

7) J. M. Carroll： Scenario-Based Design, John Wiley & Sons, 1995.

8) 中所武司ほか：「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M-baseの開発技法、情報処理学会ソフトウェア工学研究会資料、No. 109-2、May 1996、pp. 9-16.

9) D. Garlan, et al.：Architecture Mismatch: Why Reuse is So Hard、IEEE Software, Vol. 12, No. 6, Nov. 1995, pp. 17-26.

10) 柿木 薫ほか：コンポーネントウェアを適用した開発事例と開発方法論、オブジェクト指向最前線－情報処理学会オブジェクト指向 '96シンポジウム論文集, 朝倉書店, Jul. 1996, pp. 135-138.

11) 真辺純裕, 藤田伸也：コンポーネントウェアのシステム開発への適用性、情報処理学会ソフトウェア工学研究会資料、No. 110-12, pp. 81-86, Jul. 1996.

12) 長野宏宣：ソフトウェアCALSの狙いと実証実験について、情報処理、Vol. 37, No. 12, Dec. 1996, pp. 1083-10 J. 88.

13) Q. Ning： A Component-Based Software Development Model, Proc. IEEE COMPSAC '96, Aug. 1996.

14) M. Shaw and D. Garlan： Software Architecture, Prentice Hall, 1996.

15) D. Tkach, W. Fang and A. So：Visual Modeling Technique, Addison Wesley, 1996.

16) R. Wirfs-Brock and B. Wikerson：Object-Oriented Design: A Responsibility-Driven Approach, Proc. ACM OOPSLA '89, pp. 71-75, 1989.

3．2 HIPACE for APPGALLERY

HIPACE for APPGALLERYは、システム開発方法論HIPACEのスパイラルアプローチ開発標準手順のメニューの一つとして位置付けられる。APPGALLERYを活用して効率的に業務プログラムを構築するための方法論である。

3．2．1 HIPACE for APPGALLERYの特徴

HIPACE for APPGALLERYは、部品組み立て型開発ツールであるAPPGALLERYの利点を最大限に利用して、アプリケーションプログラムを短期間、低コストで開発することを目的とする方法論である。この目的を実現するために、以下のような特徴がある。

(1)プロトタイピング技術

APPGALLERYでは、ドラッグ・アンド・ドロップ等の操作により簡単に3層モデルのプレゼンテーション層を作成できる。この機能を利用して、エンドユーザと開発者がその場で外部仕様を確定させていく「ラピッドプロトタイピング」が可能である。この方式には、開発の初期段階でシステムの最終イメージがつかめるというメリットがある。

しかし、この有効な技術もプロジェクトに適用するとさまざまな問題に直面する。開発者側から見ると、仕様が膨れ上がる、仕様が収束しない、いつでも仕様変更ができると誤解されやすい、仕様の承認があいまいになりがち、といった問題点である。いずれも納期遅延やコストの増大等、プロジェクトを混乱させる要因となる。HIPACE for APPGALLERYでは、解決策としてプロトタイピングをより効果的に行うための開発モデルを提供する（図3.2-1）。この開発モデルは、3つのフェーズから構成される。

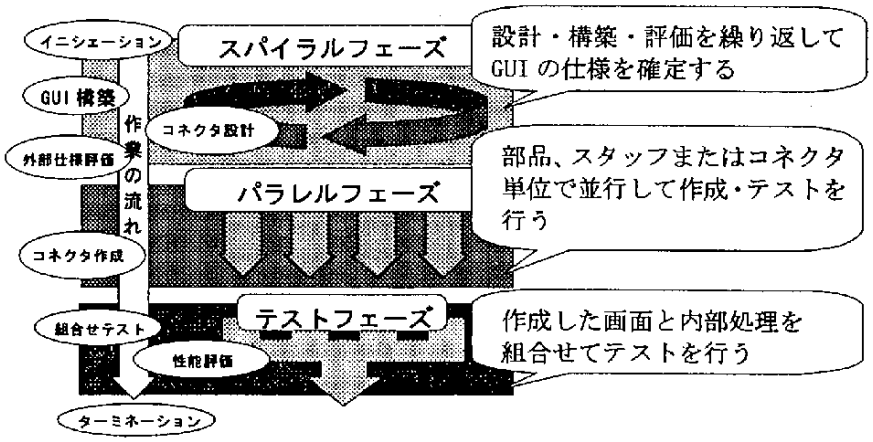


図3.2-1 HIPACE for APPGALLERYの開発モデル

(a) スパイラルフェーズ（外部仕様確定フェーズ）

GUIの構築と画面遷移を中心にプロトタイピング技術を使って、業務プログラムの外部仕様を確定させるフェーズである。

エンドユーザからの要求をもとに、仕様が明確になった部分から画面遷移の定義、キャンバス分割、キャンバスへの部品配置、フォーム及び画面遷移に必要なコネクタを作成する。キャンバスのGUI部分についてエンドユーザを含めたレビューを行い、結果をフィードバックしながら外部仕様を確定する。作成範囲を拡大し、システム全体の外部仕様が確定した時点で、チェックリストを作成し、内部処理構築フェーズに進む。

(b) パラレルフェーズ（内部処理構築フェーズ）

外部仕様を確定させた後、内部処理を構築していくフェーズである。

内部処理構築では、必要な部品やコネクタ、スタッフの設計と作成を行う。コネクタプロシジャには発生するイベントや業務に固有な処理を記述する。また、カスタマイズに必要なスタッフや部品も作成する。作成したものについて単体テストまで実施する。

このフェーズでは、業務単位やキャンバス等の分割した単位で並行して作業を進める。

(c) テストフェーズ

作成した画面、コネクタ、部品、スタッフなどを組み合せてテストを行う。

この開発モデルでは、各フェーズ内の作業が完了してから次のフェーズに進み、異なるフェーズにまたがる戻り作業は行わない。外部仕様確定フェーズとそれ以降のフェーズを切り離すことにより、仕様の承認と変更管理を行いやすくしている。また、プロジェクト管理の面からは、エンドユーザの参加など、流動的な要素の多いプロトタイピングの作業を局所化することによりプロジェクト全体を制御しやすくしている。

(2) EUCAP(End-User Customizable Application Program)への対応

EUCAPとは、エンドユーザが自分用にカスタマイズできるインタフェースをもつプログラムのことである。APPGALLERYは、業界標準のプログラム間連携技術に対応するソフトウェア部品とエンドユーザが修正できるビジュアルなインタフェースの提供によりEUCAPを実現する。EUCAPは、開発とカスタマイズを明確に分けるところに特徴がある。すなわち、情報システム部門はソフトウェア部品の評価、購入、開発などを行い、エンドユーザは情報システム部門から提供された部品を組み合わせて業務を構築する。また、基幹系業務システムの場合、エンドユーザはプレゼンテーション層のカスタマイズを行う。

HIPACE for APPGALLERYにおけるEUCAPの開発スタイルを図3.2-2に示す。エンドユーザはスパイラルフェーズに参加し、プロトタイプのレビューを通して要求仕様を決定する。パラレルフェーズ以降は、

情報システム部門は開発、エンドユーザ部門はカスタマイズの教育を受ける。こうした共同作業を通じて、開発システムに対するエンドユーザの知識を深めると同時にオーナーシップを醸成する。これがEUCを推進する動機となる。

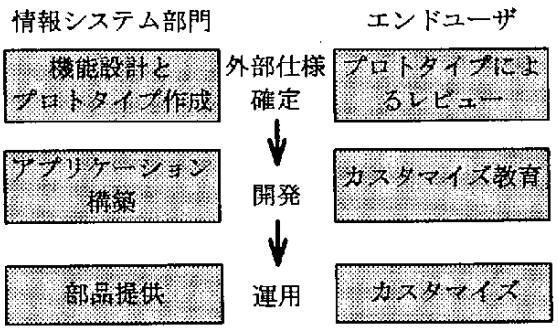


図3.2-2 EUCAPの開発スタイル

開発作業の結果、情報システム部門から提供されるのは、部品群、部品間の関連付けを支援する対話ツール（スタッフ）群、そしてカスタマイズを前提としたアプリケーションプログラムである。エンドユーザ部門は、このアプリケーションプログラムをAPPGALLERYが提供するビジュアルな開発環境を用いて、「絵を描くように」カスタマイズを行い業務システムを完成させる。

また、EUCAPの開発スタイルは、企業レベルで統制されたEUCを可能にする。APPGALLERYが扱うソフトウェア部品はバイナリで提供され、かつ利用できる機能はスタッフに制限されるため、エンドユーザがカスタマイズ可能な範囲を限定できる。データベースアクセス部品を統一したり、カスタマイズ範囲をプレゼンテーション層に制限することにより、開発したシステムが企業全体の情報の流れからの孤立やデータの重複によるコスト増加を防ぐことができる。

(3) 開発ツールを前提にした標準手順

HIPACEのスパイラルアプローチ開発では、大きく2種類の手順を提供している。1つは、特定のツールに特化しないCSS (Client Server System) 構築に必要な汎用的な手順である。もう1つは、HIPACE for APPGALLERYのように個々の開発ツールごとに用意された手順である。前者が、汎用性、網羅性を重視しているのに対し、後者は、開発ツールの機能に合わせて、汎用的な手順をツール専用にカスタマイズした手順である。CSS分野での、納期短縮、コストダウンの要求は厳しく、また開発ツールの機能拡張が極めて短サイクルで行われることから、特定開発ツールに最適化された手順を開発し世の中の激しい動きに追随できるようにしている。APPGALLERYについても、発表当初はEUCの高度化を主なターゲット領域

にしていたが、その後の機能拡張や性能改善により、現在では本格的なCSS開発に適用できる開発ツールへと発展してきている。

手順としては、開発ツールを特定することにより以下のようなメリットが生まれる。

(a) その開発ツール独自の機能を活かすことができる。

(b) 不必要な作業項目が明確になる。

(c) 効率的な作業の順序が明確になる。

(d) 作成するドキュメントを最小限にできる。

(e) ツールの用語を使って具体的な表現ができる。

(f) 名称基準等、標準化がしやすい。

(g) 成果物が明確になるので、プロジェクト管理の観点を示しやすい。

一口にコンポーネントウェアに対応する開発ツールと言っても、ツール毎に想定する開発スタイルは大きく異なる。共通するのは、JAD (Joint Application Design)形態でのラピッドプロトタイピング機能のサポートと、コンポーネント開発者とアプリケーション開発者の役割分担を意識した設計思想であるが、これについてもツールにより実現方式は大きく異なる。アプリケーション開発の現場に実戦的な手順を提供するためには、ツール対応にした方が効果的である。

また、開発ツール適用による生産性向上には、そのツール固有のノウハウをどれくらい蓄積しているかが大きく影響する。ツール対応に定義された作業項目は、成功事例や失敗事例から得られたノウハウの整理を容易にし、ノウハウの共有に寄与する。HIPACE for APPGALLERYでは、標準手順とともにプロジェクトの経験から得られた個別ノウハウも合わせて提供している。

3．2．2　HIPACE for APPGALLERYの開発プロセス

HIPACE for APPGALLERYが提供するAPPGALLERYの開発プロセスを図3.2-3に示す。先に示したように、開発プロセスは大きくスパイラル、パラレル、テストの各フェーズに分かれ、その中で必要な作業項目が定義されている。また、各作業項目に対し、ソフトウェア部品の調査や選択が情報システム部門の作業、エンドユーザは外部仕様の作成と評価に加わる等、情報システム部門とエンドユーザの役割分担を明確にしている。さらに、作業項目の名称は、「キャンバス分割」、「コネクタ設計・作成」、あるいは「スタッフ作成・設計」等、APPGALLERY固有の作業名称で記述している。

(1) プロジェクト開始の準備では、開発着手時に必要な上流の設計情報が揃っているかを確認する。上流の設計情報が詳細にドキュメント化されていなくとも、要求仕様を熟知した設計者が参画していれば開発に着手できる。このフェーズを開始するために必要なドキュメントの種別は厳密に規定しない。また、

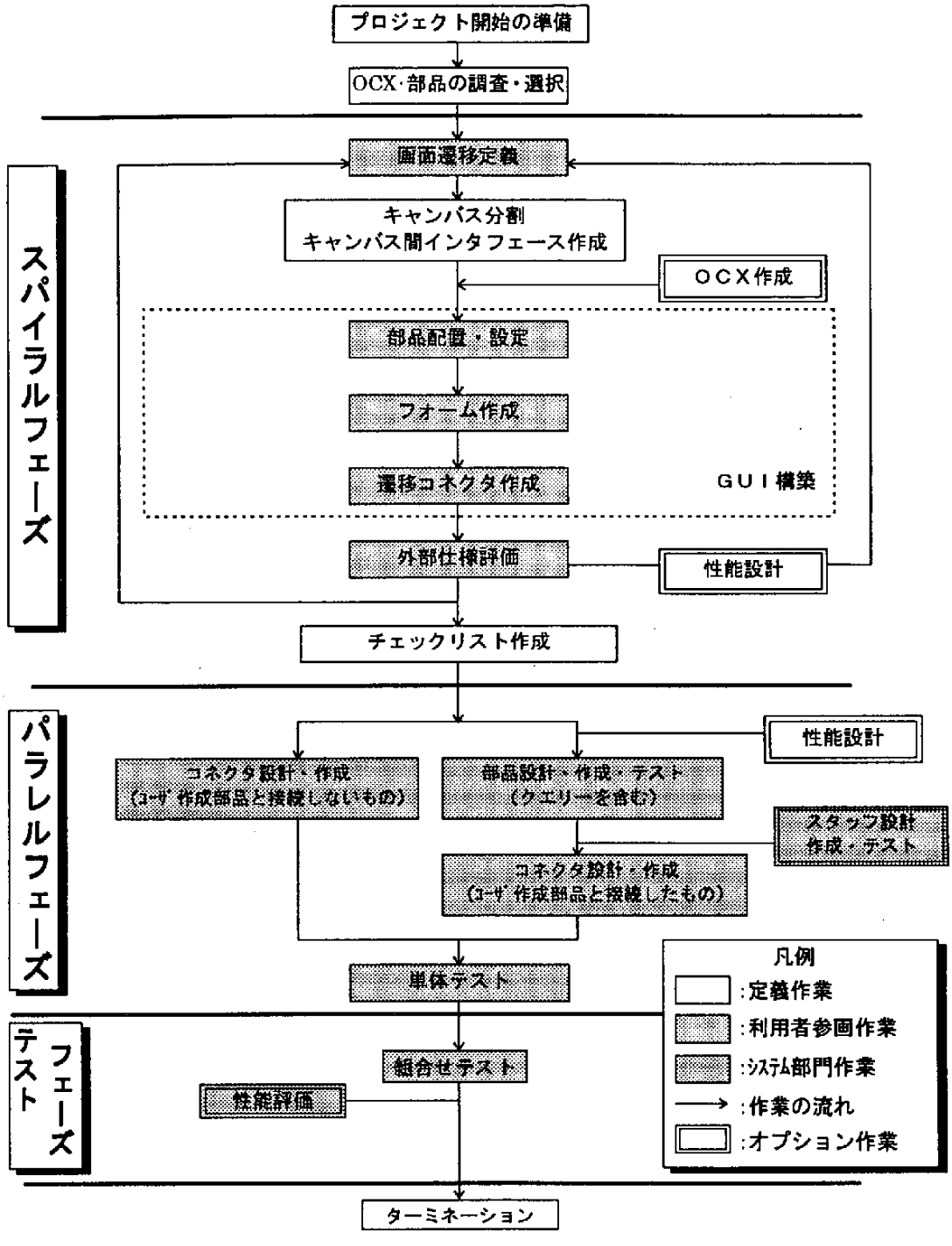


図3.2-3　HIPACE for APPGALLERYの開発プロセス

開発者の視点でAPPGALLERYで開発可能なプロジェクトかどうかを判断する。

さらに「プロジェクト開始の準備」では、図3.2-4に示すように開発プロジェクトの特性に合わせて、作業項目の取捨選択や作成ドキュメントの追加変更等、作業手順のカスタマイズを行う。

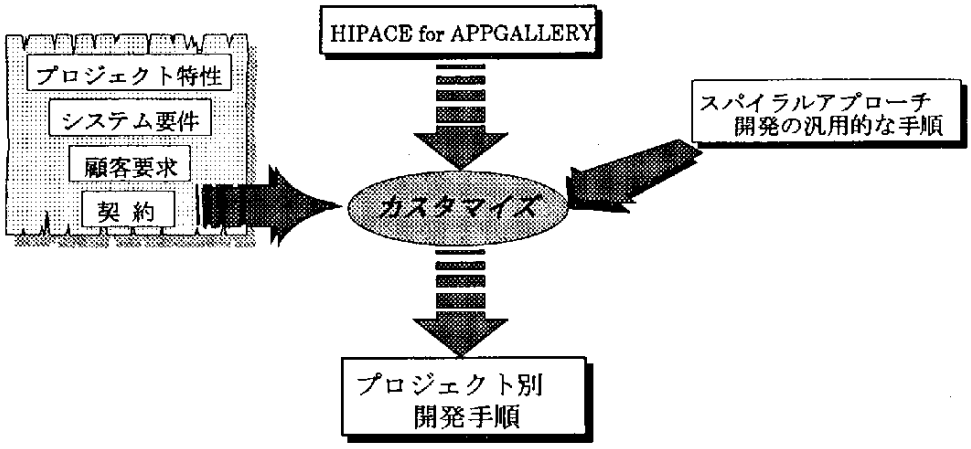


図3.2-4 手順のカスタマイズ

通常、手順のカスタマイズは、網羅された作業項目やドキュメントの中から不必要なものを削除する形で進められる。しかし、HIPACE for APPGALLERYは、先に述べたように短納期かつ低コストでアプリケーションを開発することを目指しており、作業項目とその成果物は最小限必要なものだけを設定している。このため、APPGALLERYの手順をカスタマイズする場合は、逆にスパイラルアプローチ開発の汎用的な手順の中からそのプロジェクトに不足しているドキュメントを選定し、その作成作業を手順の中に追加していく形になる。例えば、顧客との契約により納入物としてドキュメントが必要になるケースや、開発場所が地理的に分散している等の理由でコミュニケーションのためのドキュメントが必要になるようなケースである。

網羅されたものから削除していく場合、プロジェクトを安全に進めたいという心理から不必要な作業を念のために残してしまう可能性（落とすのに理由が必要）が高いが、特定ツール対応の手順のカスタマイズでは、追加作業を中心にさせる（入れるのに理由が必要）ことにより不必要な作業が入り込むのを防いでいる。

(2) スパイラルフェーズ

スパイラルフェーズでは、運用時の実行環境を考慮してアプリケーションやプレゼンテーション層を定義していく。エンドユーザを含めて画面遷移定義やGUI構築作業を行ない、外部仕様評価の結果をフィードバックしながら外部仕様が確定するまで繰り返す（図3.2-5）。GUI構築作業では、キャンバスの作成と、そのキャンバスに載る部品として、フォームの作成および画面遷移部分のコネクタ作成を行なう。

また、GUI構築作業を行いながら、標準部品として提供されていないソフトウェア部品を、APPGALEERYの部品ビルダを使って作成する。スパイラルフェーズで作成するソフトウェア部品は、プレゼンテーション層で利用する部品と切り出し可能なサブシステム間で共通する機能である。ソフトウェ

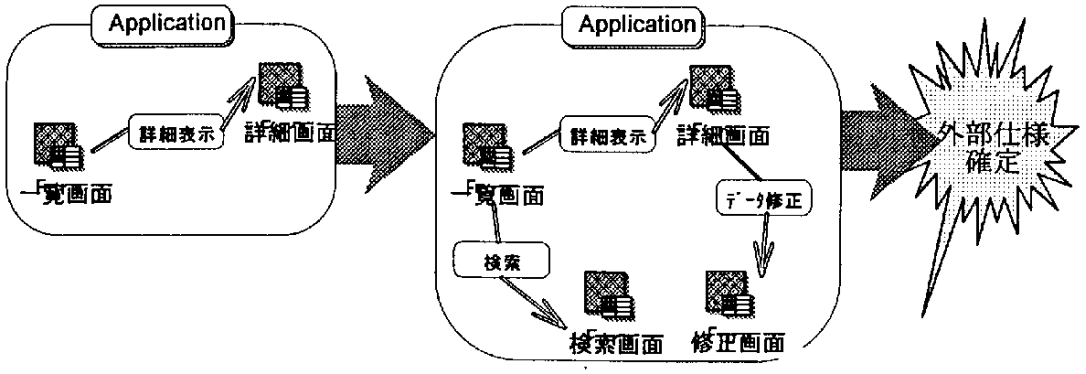


図3.2-5 外部仕様確定フェーズでの開発方法

ア部品は、情報システム部門が作成する。

外部仕様を徐々に確定させる過程で、必要に応じてキャンバスを分割する。キャンバス分割は、アプリケーションファイル内の部品数やキャンバスに表現される処理フローの複雑さ等を考慮して、業務上、意味のある単位に分割する。APPGALLERYのキャンバスは、そのまま設計仕様書あるいはエンドユーザに提供するカスタマイズ環境となるため、「見易さ」も重要な要素として考慮される。

外部仕様が確定したら、チェックリストを作成する。このチェックリストは、通常のプログラム作成で利用されているものより広い範囲のチェック項目を含む。エンドユーザとの共同作業を通じて出されるプロトタイプに反映しきれない要求項目（例えば、エラー処理の範囲や方式等）についても、チェックリストに反映する。

スパイラルフェーズは、ビジュアル化された仕様書としてのプロトタイプとチェックリストに対し、エンドユーザの承認を得て終了する。ここで作成するプロトタイプは使い捨てではなく、このフェーズで反復拡大され次フェーズに引き継がれる。

本開発手順では開発手順の簡易化を目的としているため、省略可能な作業はオプション作業として設定し、プロジェクト固有の要因により実施を決定する。性能に対して厳しい要求がある場合は、ボトルネックとなりそうな要因の洗い出しと、性能評価を目的としたプロトタイプによる実測を行い性能対策を検討する。

「プロジェクト開始の準備」とスパイラルフェーズで行う作業の概要を表3.2-1に示す。

表3.2-1 作業概要一覧（「プロジェクト開始の準備」とスパイラルフェーズ）

作業項目名			作 業
プロジェクト開始の準備			開発作業に着手できるかどうかを判断し、開発着手の条件を揃えるための調整を行なう。 作業方針と作業の成果物を確認し、作業日程を調整する。
OCX･部品の調査、選択			開発に着手する前に、開発に適するOCX･部品の調査を実施し、試用する部品を選択する。
スパイラルフェーズ	画面遷移定義		業務の流れを画面遷移図に表現する。 外部仕様評価後に、詳細な画面遷移図を作成する。
	キャンバス分割		アプリケーションの機能や処理の量に応じてキャンバスを分割する。
	GUI構築	部品配置・設定	キャンバスにフォーム部品などのGUI構築部品を配置する。 部品のプロパティなどの設定を行なう。
		フォーム作成	要求仕様に基いて、業務を実現するための画面の設計と作成を行なう。
		遷移コネクタ作成	画面遷移など外部仕様に関するイベントプロシージャのコーディングを行なう。
	外部仕様評価		フォームや遷移状況、キャンバスの内容などをユーザとレビューし、外部仕様の評価を行ない、この結果を画面遷移定義までフィードバックする。
	チェックリスト作成		機能仕様と画面遷移図をもとにして、チェックリストを作成する。 作成の際に、内部処理仕様を確認する。
	性能設計 （オプション）		性能上のネックになると想定される部分を洗い出し、該当部分の外部仕様を性能面から考慮して設計する。
	OCX作成 （オプション）		標準部品で機能を実現できない場合や、共通に利用する機能を切り出してOCXで実現する。

(3)パラレルフェーズ

パラレルフェーズでは、外部仕様の確定したアプリケーションの内部処理を業務単位に並行して構築する。内部処理としては、ソフトウェア部品、コネクタプロシジャおよびスタッフの設計と作成を行う。パラレルフェーズで行う作業の概要を表3.2-2に示す。

表3.2-2 作業概要一覧（パラレルフェーズ）

作業項目名		作業概要
パラレルフェーズ	コネクタ設計・作成 (ユーザ定義部品と 接続しないもの)	スパイラルフェーズで設計した内容を元にコネクタを設計する。チェックリストに基づき内部処理テストを行なう。
	部品設計・作成・テスト	ユーザ定義部品を設計し、作成する。また、作成した部品が正しい動きをするのかをテストする。チェックリストに基づき内部処理テストを行なう。
	コネクタ設計・作成 (ユーザ定義部品と 接続したもの)	作成したユーザ定義部品の内容を元に、ユーザ定義部品と接続するコネクタを設計する。チェックリストに基づき内部処理テストを行なう。
	スタッフの 設計・作成・テスト (オプション)	保守用またはエンドユーザのカスタマイズ要求を予測して、スパイラルフェーズで設計した内容を元にスタッフを作成する。チェックリストに基づき内部処理テストを行なう。
	単体テスト	キャンバス、アプリケーション単位にテストを行なう。

パラレルフェーズで行う単体テストは、以下のような観点で行う。

(a) コネクタの内部処理テスト

コネクタ単位にテストデータを設定し、正常値、異常値、境界値および限界値のテストを行う。

(b) 共通部品の内部処理テスト

共通部品は、コネクタの内部処理テストに関連付けてテストを行う。コネクタから部品内のメソッドを実行し、動作を確認する。

(c) フォームのテスト

フォーム部品で作成した画面を実際に表示し、設計時と同じ動作をしているか確認する。フォームのテストは、コネクタのテストおよび共通部品のテストが完了してから実施する。

(d) スタッフの内部処理テスト

スタッフが生成するコネクタスクリプトの動作をテストする。テスト方法は、コネクタの内部処理テストと同じである。スタッフが生成するコネクタスクリプトが複数種類ある場合は、すべてのケースをテストする。

(e) キャンバス単位のテスト

画面間のインタフェースが正しいか、データが正常に表示されているか等を確認する。このテストは、キャンバス内のすべてのコネクタ、共通部品およびフォームのテストが完了してから実施する。

(4) テストフェーズとターミネーション

テストフェーズでは、並行して開発したAPPGALLERYのアプリケーションファイルや他の連携したツールを統合して、アプリケーションとしてのテストを行う。

ターミネーションでは、開発したアプリケーションプログラムの品質が基準を満たしていることと所定の作業が完了していることを確認し、次工程であるシステムテストに入れるかどうか判断する。保守のために必要な資料や納入が必要なドキュメントは、APPGALLERYの印刷機能を活用して作成する。

テストフェーズとターミネーションで行う作業の概要を表3.2-3に示す。

表3.2-3 作業概要一覧（テストフェーズ／ターミネーション）

作業項目名		作業概要
テストフェーズ	組合せテスト	チェックリストをもとに組合せテストを実施する。
	性能評価（オプション）	要求される性能が実現されているか、実機による測定で確認する。
ターミネーション		作業の完了と品質を確認し、ドキュメントを作成する。ドキュメントは、APPGALLERY内に展開された設計情報から作成する。

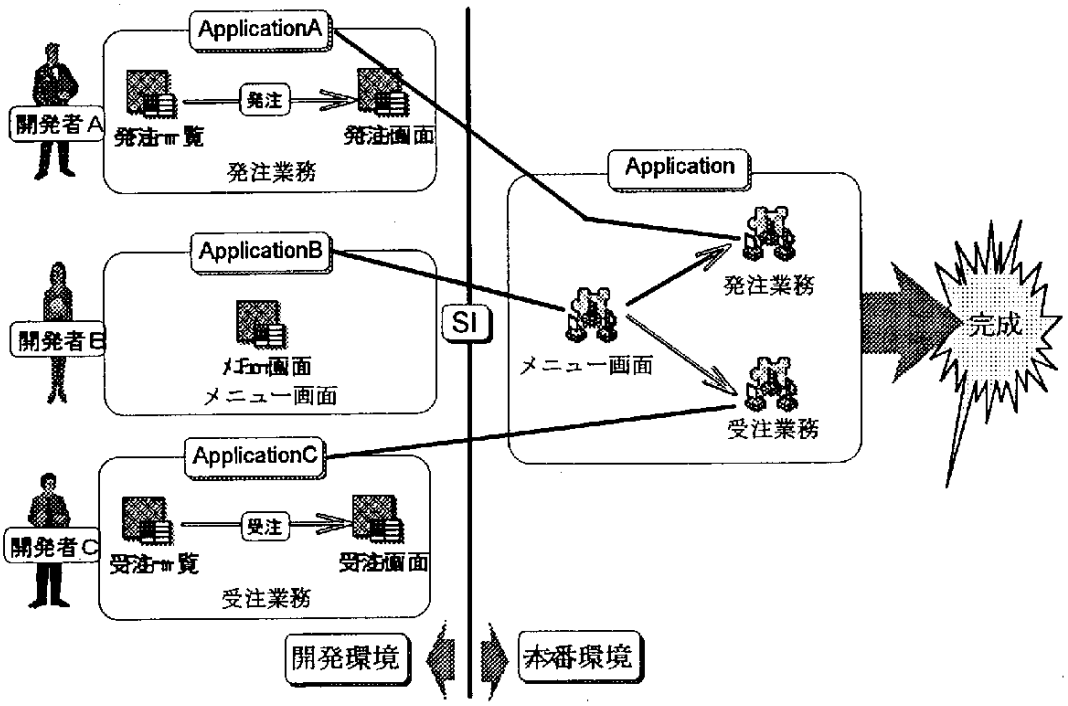


図3.2-6　テストフェーズでのシステムの統合およびテスト

テストフェーズでは、図3.2-6に示すように各開発者が作成し内部処理テストを完了したキャンバスを統合してテストを行う。テスト作業は、画面遷移図に記された単位で業務フローと機能仕様を確認する。チェックリストの設定件数とバグの摘出目標値は、ファンクションポイント法で計測したソフトウェアの規模をもとに基準を設定する。

3．2．3　まとめ

ここでは、HIPACE for APPGALLERYが提供するコンポーネントウェアを活用したアプリケーション構築技術について述べた。HIPACE for APPGALLERYは、既に数十のプロジェクトに適用され、生産性の向上と開発期間の短縮に貢献している。

開発方法論では考え方やコンセプトを確立していくことも重要であるが、HIPACEの特定ツールに対応した手順は、それにとどまらず実用的な構築技術の提供を目指している。コンセプトと現実のギャップを埋めるには、実践から得られたノウハウを反映することが重要と考えている。

EUCやコンポーネントウェアに関する理論や開発ツールはますます進化していくと思われるが、今後ともHIPACEを通じて利用技術を提供していく考えである。

【参考文献】

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

2) J. Martin：RAPID APPLICATION DEVELOPMENT, Macmillan Publishing (1991).
［邦訳：ラピッド・アプリケーション・デベロップメント, 芦沢訳, リックテレコム］

3) Udell：オブジェクト指向の新展開コンポーネントウェア, 日経バイト, pp.277-289 (1994-6).

4) 初田賢司, 他：CSS開発に対応した開発方法論HIPACE, 日立評論, Vol.78, No.6, pp.19-22 (1996-5).

5) 日立製作所：日立システム開発方法論HIPACE.

3．3　テンプレート型開発方法論

3．3．1　はじめに

ソフトウェアの再利用技術として、オブジェクト技術を用いたフレームワークが注目を浴びている。フレームワークは、ある問題領域におけるアプリケーションの基本的な枠組みを決めることにより、クラス群の再利用性を向上しようとするものである。フレームワークでは、アプリケーションを実現するために必要な“オブジェクト”群とオブジェクトを利用した“処理の流れ”のテンプレート（ひな形）が提供される。アプリケーションの開発者は、そのテンプレートをカスタマイズすることで目的のアプリケーションを構築でき、短工期、高品質なシステム開発が可能となる。

テンプレートを利用することで生産性を向上する試みは、従来からも行われてきた。例えば、上流における業務モデルの構築や、下流における汎用サブルーチンや汎用パターンの提供といった試みである。

しかし、これらの試みは、

・特定の開発フェーズのみを対象としている。

・構造化技法で構築されており、カスタマイズに柔軟に対応できない。

・問題領域の汎用部分とユーザ独自部分が明確にされていない。

などの理由から、効果的に利用されてこなかった。一方、「フレームワーク型開発＝テンプレート型開発」では、

・要件獲得から実装フェーズまでをサポートする。

・オブジェクト技術の適用により、カスタマイズに柔軟に対応できる。

・オブジェクト技術の適用により、問題領域の汎用部分とユーザ独自部分が明確にできる。

ことにより、再利用性の高いテンプレートの提供を目的としている。

本節は、次の構成になっている。

・テンプレート型開発の概要

・ビジネスオブジェクトと４層アーキテクチャー

・テンプレートの開発と利用方法

・おわりに

3．3．2　テンプレート型開発の概要

ここでは、テンプレートの概念、従来からの試みとの違い、全体構成などについて述べる。

(1) テンプレートの概念

図3.3-1は、フレームワークとテンプレートの関係を表わしたものである。フレームワークは、オブジェ

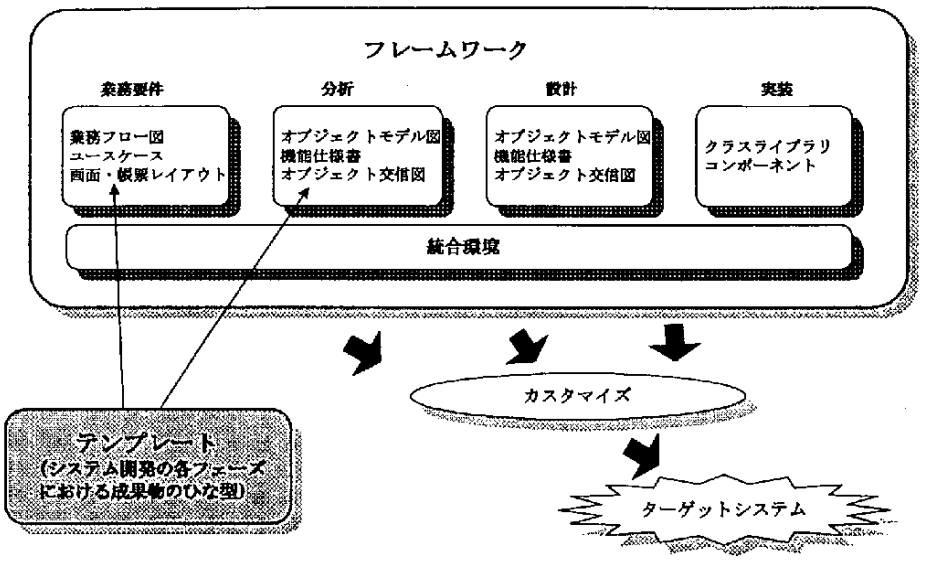


図3.3-1 テンプレートの構成概要

クト技術を利用したソフトウェアの再利用手法の一つである。ある問題領域におけるアプリケーションの基本的な枠組みを決め、その枠組みに従ったアプリケーション構成要素のひな形を提供することでソフトウェア開発の生産性を向上しようとするものである。アプリケーションの開発者は、フレームワークの提供するひな形をカスタマイズすることで目的のシステムを構築する。このフレームワークを用いた開発の各フェーズで提供される個々のひな形のことを「テンプレート」と呼ぶ。具体的には、要件獲得から設計にいたるフェーズでは、モデル図や仕様書などのドキュメントをテンプレートとして提供する。また、実装では、クラスライブラリやコンポーネントなどのソースコード／バイナリコードを提供する。

(2)従来法との違い

再利用技術の要件として次の様な項目があげられる。

・対象とする問題領域が明確である。

・システム開発の全工程をサポートしている。

・カスタマイズすべき部分が明確である。

・カスタマイズに柔軟に対応できる。

前述のように、従来から業務モデルや汎用サブルーチン等が再利用技術として提供されてきた。しかし、これらは、上記要件を十分に満たしておらず、システム開発全体を網羅した様な抜本的な再利用は困難であった。

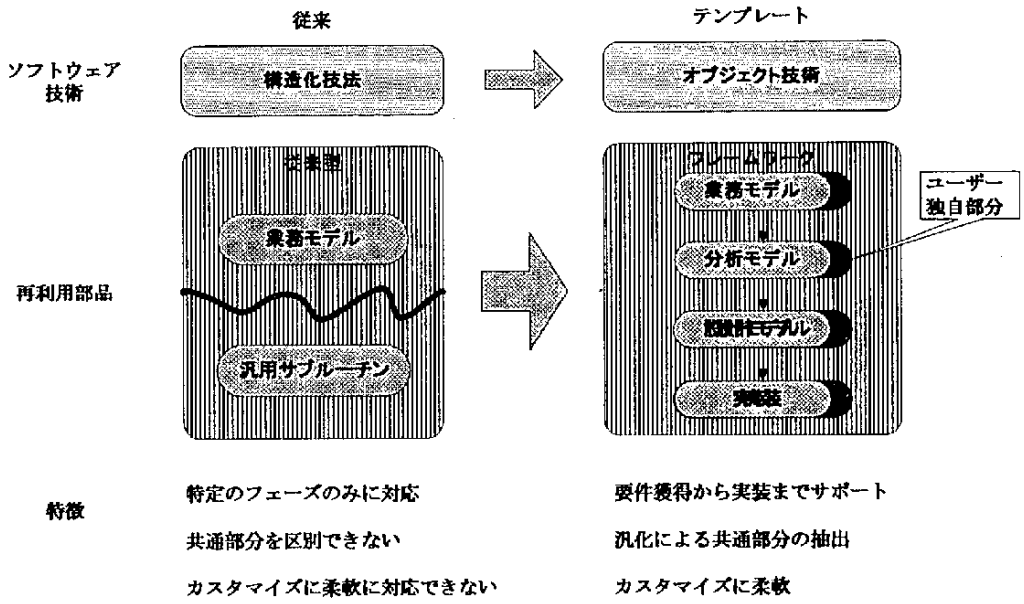


図3.3-2　従来の再利用方法との違い

図3.3-2は、従来の再利用技術と今回提案するテンプレート型開発の違いを表わす。従来方法は、基本的に構造化技法に基づいていた。構造化技法は、下位機能と上位機能が密接に関係しており、機能を変更しようとした場合、その影響範囲が広くなるため、仕様変更に対する柔軟性がない。また、ある問題領域において、複数ユーザ間での共通部分とユーザ独自の部分を明確にする方法を提供していない。これらのことから、再利用の対象は、構造化技法と関係のない業務モデルや、汎用サブルーチンなどの粒度の小さい実装部品にとどまっていた。

これに対し、テンプレート型開発ではオブジェクト技術を用いたフレームワークを採用しており、分析から実装まで同じオブジェクトを対象に作業を行うので、開発フェーズ間のギャップの少ないシームレスな開発が可能である。さらに、オブジェクト技術では、汎化／特化により問題領域の共通部分とユーザ独自部分を明確にすることができる。これにより、ユーザ独自部分の抽出を効果的に行え、抽出したユーザ独自部分の下流での影響個所が明確になる。これらのことから、テンプレート型開発では、要求仕様に応じたカスタマイズを効果的に行うことができる。

(3) 対象とする開発フェーズ

図3.3-3は、テンプレートの対象とする開発工程を示したものである。開発プロセスは、基本的にオブジェクト指向開発方法論の一つであるOMTをベースにしている。今回のテンプレート開発では、図中の網掛けの開発フェーズを対象としたテンプレートを提供する。図中右端に参考のために、テンプレートの各開発フェーズに対応するEFRAME（Extended FRAME：拡張共通フレーム)のフェーズを記しておく。

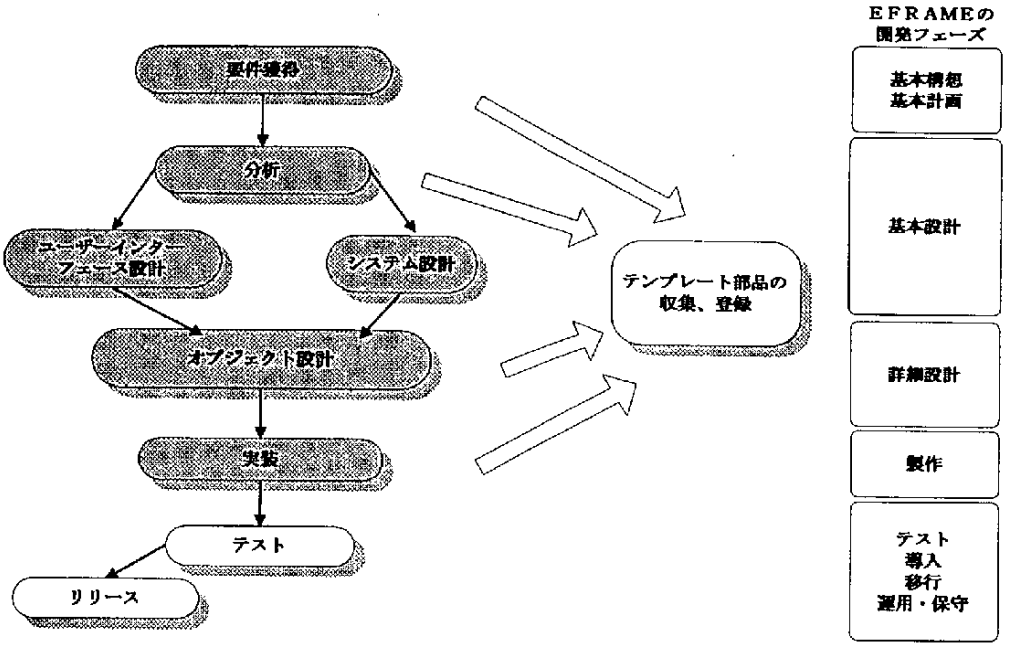


図3.3-3　テンプレートが提供される開発フェーズ

テンプレートは、現在のところ業務要件の獲得フェーズから分析、設計、実装を対象としている。企画書作成などの事前検討フェーズやテストなどのフェーズについては、今回は対象としていない。また、テンプレートはシステム開発の主要な成果物のひな形なので、問題領域の業界動向や、経験の浅いSE向けの説明などは含んでいない。したがって、テンプレートを利用するには、問題領域に出てくる単語の意味や業務概要などの一般的な知識は必須である。また、テンプレート型開発では、オブジェクト技術により再利用性向上を行うものであり、エンドユーザが利用する場合でも、オブジェクト指向方法論や利用しているオブジェクト指向言語について一通りの習得が必要である。

(4) 全体構成

図3.3-4は、フレームワーク（テンプレート）の全体構成図である。前述のように、フレームワークは、複数の種類のテンプレートから構成され、それらは開発フェーズごとに提供される。開発フェーズは、要件獲得、分析、システム設計、ユーザインタフェース設計、オブジェクト設計、実装に分類される。

要件獲得フェーズでは、ユーザがシステムに何を求めているかを定義する。ユーザ要件は、業務機能に関係する業務要件と、パフォーマンスや実装環境といった業務機能そのものでない非業務要件の２つに分けられる。業務要件の定義では、業務フロー図、ユースケース、画面レイアウトを提供する。ユースケースは、ユーザがシステムをどのように使うかを文章表現で定義したものであり、厳密な要件定義が可能である。ユースケースは、このフェーズの中心的な成果物である。非業務要件については、今回は、特に

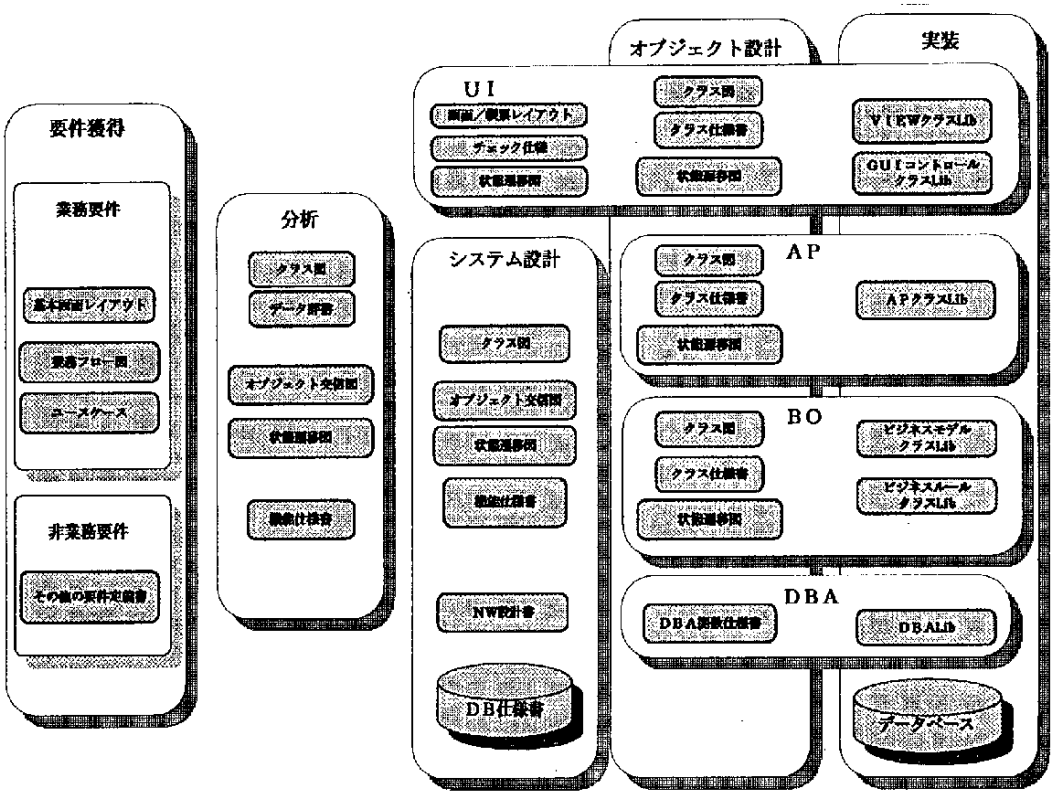


図3.3-4　フレームワーク全体構成

特定のドキュメントを規定していない。

分析フェーズでは、オブジェクトモデルとしてクラス図とデータ辞書、機能モデルとして機能仕様書、動的モデルとしてオブジェクト交信図と状態遷移図を提供する。これらは、問題領域をモデル化したものであり、実装環境とは独立したものである。それゆえ、システム環境が変わっても、分析フェーズのテンプレートには影響しない。

システム設計フェーズでは、アプリケーションの構造を記述したクラス図、機能仕様書、オブジクト交信図、状態遷移図と、物理設計を記述したネットワーク仕様書やデータベース仕様書を提供する。

ユーザインタフェース設計では、画面／帳票レイアウト、操作説明書、チェック仕様、状態遷移図を提供する。これらは、ユーザインタフェースの外部仕様のひな形であり、ユーザとの打合わせを効率化する。

オブジェクト設計フェーズでは、クラス図、オブジェクト交信図、状態遷移図、クラス仕様書を提供する。これらは、システム規模や実装環境に依存したものである。

実装フェーズでは、オブジェクト設計に基づいた成果物を、クラスライブラリ(ソースプログラム)またはコンポーネント(バイナリ部品)として提供する。

3．3．3　ビジネスオブジェクトと4層アーキテクチャ

(1) ビジネスオブジェクト

図3.3-5にビジネスオブジェクトの概念を示す。このシステムで利用されるオブジェクトを、次の3種類に分類する。

・ビジネスオブジェクト：あるビジネス(問題領域)に普遍的に存在する“もの”や事象を表わすオブジェクト。

・テクニカルオブジェクト：データを表示するビューオブジェクトやDBのハンドリング、帳票出力など、システム環境を制御するオブジェクト。

・アプリケーションオブジェクト：ビジネスオブジェクト、テクニカルオブジェクトを連携させ、業務機能を達成するオブジェクト。アプリケーションの主体となるオブジェクト。

テクニカルオブジェクトは、さらにユーザインタフェースを担当する部分とデータベースへのアクセスを担当する部分の2つに分類できる。

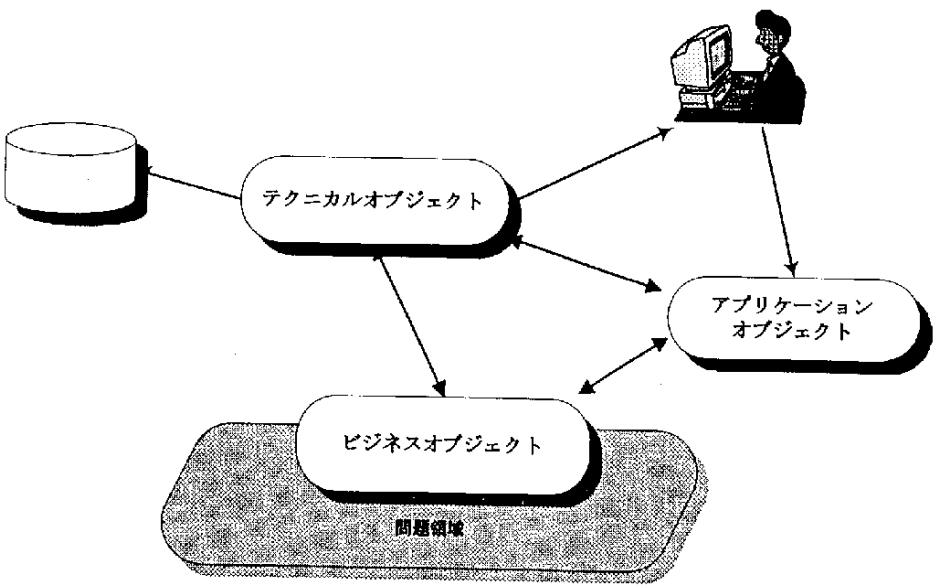


図3.3-5　ビジネスオブジェクトの概念

ビジネスオブジェクトとは、例えば、販売管理という問題領域では、取引や伝票、商品などがビジネスオブジェクトである。これらは対象のビジネス領域が同じであれば、複数のユーザ間でも変化の少ない本質的なものである。したがって、ある問題領域についてビジネスオブジェクトを抽出し、それを複数プロジェクトで再利用することで生産性向上が図れる。ビジネスオブジェクトは、実装から切離されたものであり、どちらかというと上流工程の生産性向上を主眼としている。ビジネスオブジェクトは、さらに

・ビジネスモデルオブジェクト：ビジネスで利用する“もの”。静的なデータオブジェクト。

・ビジネスルールオブジェクト：金額計算ロジックや決裁方法など、業務上のルール。

に分類できる。静的なビジネスモデルオブジェクトに比べ、ビジネスルールオブジェクトは、ユーザごとの変化が大きい。問題領域にある計算ロジックや制約をルールオブジェクトとして抽出し、ルールオブジェクトをビジネスモデルオブジェクトから独立させ共有化することで、ルールの変更による影響が局所化され、ビジネスモデルオブジェクトの再利用性が向上する。

(2)4層アーキテクチャー

図3.3-6は、ビジネスオブジェクトを用いた4層アーキテクチャーを表わしている。4層とは、ユーザインタフェース層(UI層)、アプリケーション層(AP層)、ビジネスオブジェクト層(BO層)、DBアクセス層(DBA層)から構成される。各層の機能概要は、次のようになる。

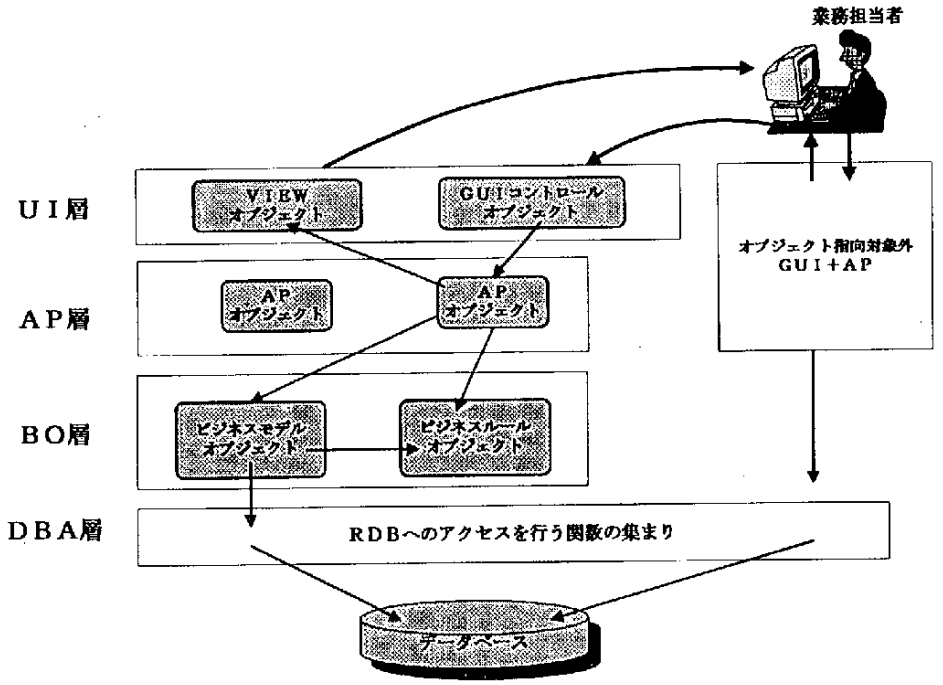


図3.3-6　ビジネスオブジェクトを用いた4層アーキテクチャー

・UI層

ユーザインタフェースを担当する層。データ表示を行うViewオブジェクト、ユーザ入力を受付けるGUIコントロールオブジェクト、帳票出力を受持つPrintオブジェクトなどがある。

・AP層

一つの業務機能を達成するために、システム内のオブジェクトを連携させる層。複数のアプリケーショ

ンオブジェクト(または関数)から構成される。

・BO層

問題領域をモデル化した層。ビジネスモデルオブジェクトとビジネスルールオブジェクトから構成される。

・DBA層

BOのDBへの格納／取出しを行う層。DBやファイルなどの実装方法の違いを隠蔽する。主に、RDBへのアクセスを行う関数の集まりである。

図3.3-7は、４層アーキテクチャーの具体例を示している（AP層がBOをコーディネートする手順は複数あるが、これはその１例にすぎない)。

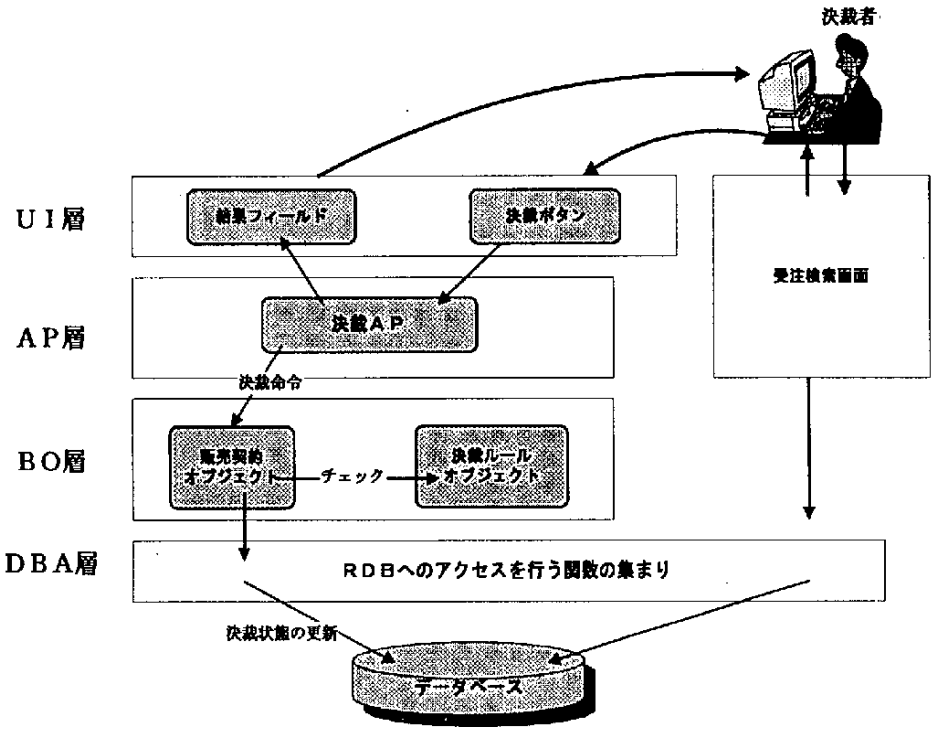


図3.3-7　４層アーキテクチャーの具体例

この例では、「決裁者がある販売契約について決裁を行う」という業務を表わしている。この処理は、以下のようなシーケンスで行われる。

・決裁者は、受注検索画面を使って、販売契約オブジェクトをDBから呼出してくる。

・決裁者は、決裁ボタン(GUIコントロール)を押し、決裁APに依頼する。

・決裁APは、対象の販売契約に対し、決裁命令をだす。

・販売契約は、決裁ルールを参照する(例：決裁ルールに決裁可能か聞く)。

・販売契約は、DBAに依頼して、決裁結果のアップデートを行う。

・決裁APは、決裁処理結果を結果フィールドに返す。

4層アーキテクチャーの利点として、次のようなことが挙げられる。

・UIの変更に柔軟に対応できる。

UIとアプリケーションを分離したことで、UIの変更がアプリケーションに依存しない。GUIの作成者は、業務のタイミングやロジックを気にしなくてよい。

・BOの再利用により、上流工程の生産性を向上できる。

分析モデルのうち、オブジェクトモデルはBO層に、機能モデルはAP層またはビジネスルールに、動的モデルはAP層にマップされる。特にBO層は、問題領域のモデルをそのまま反映でき、システム環境やユーザ独自業務(業務のタイミングやロジック)から独立しており、再利用性が高く、上流工程の生産性を向上する。

・DBAによりDBの違いを吸収している。

DBの違うプロジェクトでも、その差はDBAで隠蔽されており、BOやアプリケーションに影響しない。

一方、課題としては以下の２つがある。

・実装環境を考慮して、各層をどのプロセッサまたはプロセスに割付けるかを検討する必要がある。4層アーキテクチャーは、システムの論理的な分割(サブシステム化)しか表わしていない。AP層やBO層をどこに実装するかということについて検討する必要がある。それには、実装環境やコーディング負荷、開発体制、性能などを考慮しなくてはならない。このことについてはまだ標準的な構成やツールがないので、プロトタイプを作成するなどして繰返し検証することが必要である。

・性能の保証が難しい。

4層アーキテクチャーでは、従来のGUIとDBを組み合わせたアーキテクチャに比べ、構造的に冗長になっており、性能の劣化が予測されるので、AP/BO層をサーバーアプリケーション化するなど、柔軟に対応する必要がある。

3．3．4　テンプレートの開発と利用方法

前述のように、テンプレートは開発フェーズ単位に提供される。ここでは、各開発フェーズごとに、

・個々のテンプレートの概要

・新規テンプレートの開発方法

・テンプレートを利用した開発方法

について説明する。

(1) 要件獲得フェーズ

(a) 提供されるテンプレート

要件獲得フェーズでは、業務要件のテンプレートとして、業務フロー図、ユースケース、画面レイアウトを提供する。

(i) 業務フロー図

業務フロー図は、ユーザ業務の流れを業務ごと、業務担当者ごとに図に整理したものである。業務フロー図は、

・トップダウンでユーザ業務を表現できるため、煩雑な要件を整理しやすい。

・図表現なので、業務の大まかな流れを視覚的、直感的にすばやく把握できる。ユーザや経営者など情報システム部門以外の人にも理解しやすい。

という特徴がある。反面、業務内容の詳細を表現するには適さない。したがって、要件獲得フェーズの初期段階に、業務の大まかな流れを整理したり、新しい開発メンバーやユーザに業務を説明する場合に効果がある。

(ii) ユースケース

ユースケースは、ユーザがシステムをどのように使うかを文章表現で定義したものである(OMTではシナリオにあたる)。ユースケースには、ユーザ(アクター)、ユースケース名および概要、処理内容として基本／代替系列、制約(事前／事後条件)を記述する。ユースケースは、

・業務フロー図では表わしにくい詳細な業務ロジックを記述することができる。

・一定の形式で整理、記述されているので、複数の人の間で解釈の差が少ない。

という特徴がある。ユースケースは、要件獲得フェーズの最終的なアウトプットであり、以降のフェーズで利用する中心的な成果物である。ユースケースの作成には時間がかかるという問題があるが、テンプレートを用意することで負荷を軽減することができる。

(iii) 画面レイアウト

要件獲得フェーズでは、開発対象のシステムのイメージをつかみ、また、重要な情報の見落としを防ぐために画面レイアウトを利用する。したがって、ここでは画面の細かな制御や入力簡易化のための項目まで定義する必要はない。上記目的を実現できるのであれば、表現方法としては、実際の画面のハードコピーでも、単なる入出力項目一覧表でも良い。

(b) 新規テンプレート開発

要件獲得フェーズでは、ユーザがシステムに何を求めているかを定義する。業務要件の定義のために、業務フロー図とユースケース、画面レイアウトを作成する。分析以降のフェーズで中心的な成果物となるのはユースケースであり、このフェーズの主な作業は、ユースケースの作成である。しかし、まったく整理されていない要件から、直接ユースケースを作成するは難しい。したがって、業務フロー図を使って業務の大まかな流れを整理したり、画面レイアウトを作成してシステムの全体感を把握する。その後、ユースケースを用いてより詳細に業務要件を定義する。ユースケースは、次のような手順で作成する。

・ユーザの定義(アクターの定義)

・ユースケースの洗出し(ユーザから見た業務の洗出し)

・ユースケースの記述(ユーザが何をするかを記述する)

ユースケースをあまりに詳細に記述すると、膨大な量になり、工期を圧迫することになる。したがって、要件獲得フェーズでは、ユースケースを厳密に定義するよりも、要件や問題領域の境界を明確にすることに重点を置く。

(c) テンプレートの利用

業務フロー図や画面レイアウトのテンプレートが用意されているので、初期段階のかなりの作業を短縮できる。具体的な作業は、次のようになる。

・業務フロー図や画面レイアウトのテンプレートを使ってユーザとの打合わせを行う。その時、テンプレートとユーザ業務の差異をメモとして記述しておく。

・上記メモをインプットとして、ユースケースのテンプレートを改造し、ユーザオリジナルのユースケースを作成する。

(2) 分析フェーズ

(a) 提供されるテンプレート

OMTでは、システムをオブジェクトモデル、機能モデル、動的モデルの3つのモデルで定義する。それぞれのモデルに対して、以下のテンプレートを提供する。

・オブジェクトモデル：クラス図、データ辞書

・機能モデル：機能仕様書

・動的モデル：オブジェクト交信図、遷移図

(i) オブジェクトモデル

オブジェクトモデルは、問題領域内のオブジェクト、クラス、オブジェクト間(クラス間)の関係を示すことによって、問題領域の静的な構造を表わしている。オブジェクトモデルは、クラス図とデータ辞書か

ら成る。クラス図は、対象領域内に存在するクラスとそれらクラス間の関連を表現した図である。また、クラス図上の用語について説明したデータ辞書がある。

(ii) 機能モデル

機能モデルは、データ間の依存関係とそれを関係付ける機能を表わしている。機能モデルは、機能仕様書として定義される。機能仕様書には、機能概要、入出力データ、制約(事前／事後条件)、コラボレータ(利用するオブジェクト)などを記述する。機能仕様書は、ユーザに対してシステムが何をするかを記述したものであり、ユーザが起こすイベントに対応する。

(iii) 動的モデル

動的モデルは、システムへのイベントに対するオブジェクト間の相互作用やオブジェクト内の状態遷移を表現するものである。ユーザがシステムに与える(業務レベルの)イベントは、機能モデルの個々の機能に対応する。動的モデルは、オブジェクト交信図と状態遷移図で構成される。

オブジェクト交信図は、ある機能を実現するために必要なオブジェクトとそのオブジェクト間のメッセージ構造を記述したものである。メッセージ構造を記述する方法として、オブジェクト交信図とイベントトレース図があるが、表現力がある(柔軟に記述できる)との判断からオブジェクト交信図を採用している。状態遷移図は、あるオブジェクト内の状態の遷移とそれを引起こすイベントを対応させた図である。

作成手順は、まず、機能モデルで抽出した機能ごとにオブジェクト交信図を作成し、そのうち重要な振舞いのあるオブジェクトについて状態遷移図を作成する。

(b) 新規テンプレート開発

分析では、業務要件をもとに、システム化対象となる問題領域を明らかにし、その領域に存在するオブジェクトとそのオブジェクトの動きや役割を定義する。図3.3-8は、分析フェーズの作業を表わしている。ただし、前項（a）で述べた3つのモデルは、上記の順序のみで構築されるわけでなく、繰返しの中で作られていく。問題領域についてよく理解し、整合性のあるモデルを作成するには、分析を繰返す必要がある。また、新規開発の場合は、あるいは大きなシステムの場合、全体をいっきにモデル化するのは困難なので、業務単位に分けて分析し、それを統合、調整するという作業を繰返して、整合性のあるモデルを作成していく。

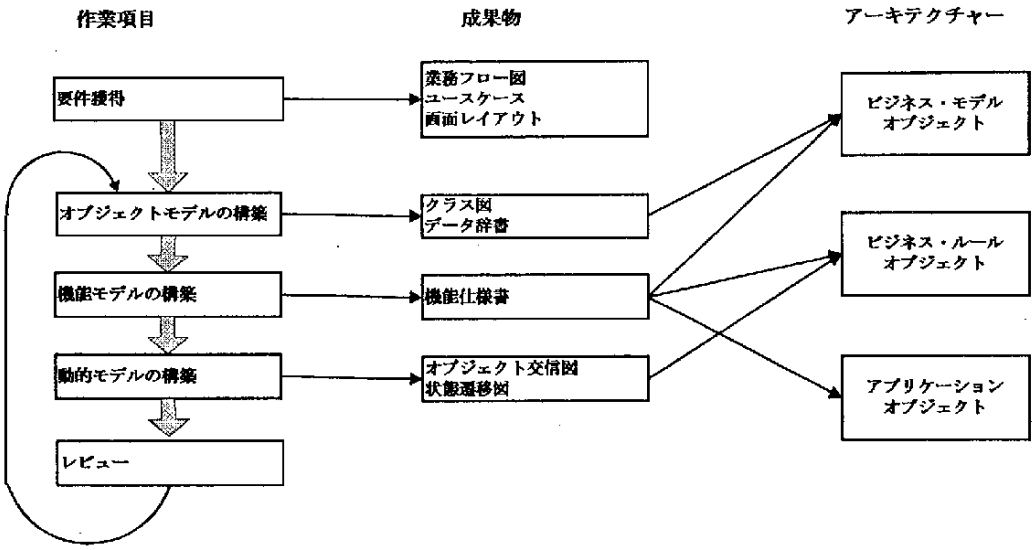


図3.3-8　分析フェーズの作業とモデル

(c) テンプレート利用

テンプレートとしてクラス図が用意されているので、クラスの抽出といった作業がかなり削減される。具体的な作業イメージは、次のようになる。

・ユースケースのユーザ独自部分に現れる新しいオブジェクトを抽出し、クラス図テンプレートへ追加する。

・ユースケースのユーザ独自部分から新しい業務機能を抽出し、機能仕様書を作成する。また、修正の必要な機能(特に事前／事後条件)について、機能仕様書を訂正する。

・新規機能仕様について、オブジェクト交信図を作成する。また、ユースケースを使って、オブジェクト交信図、状態遷移図の検証を行う。

・上記の結果をクラス図へ反映する。

テンプレートを用意することで、以下の効果が期待できる。

・分析初期に行う名詞やイベントの抽出からモデルを起こす作業を省略できる。

・分析フェーズ内での繰返し回数を減らすことができる。

・分析での詳細化レベルがテンプレートにより統一されるので、不用意に設計のレベルへ踏込むことがなくなる。

テンプレートを利用するときの問題の一つは、テンプレートが用いている用語と、ユーザの用いている用語の統一である。テンプレートは一般的な用語でモデルを作成しており、テンプレートをカスタマイズす

る場合、ユーザの用いている用語へ変換するべきかどうか明確にすべきである。あいまいなままだと、以降のフェーズでの不具合発生の元となってしまう。

(3) システム設計フェーズ

(a) 提供されるテンプレート

システム設計フェーズでは、以下のテンプレートを提供する。

・クラス図

・機能仕様書

・オブジェクト交信図

・状態遷移図

・DB物理設計書

・ネットワーク設計書

(b) 新規テンプレート開発

システム設計では、次のオブジェクト設計フェーズで分析モデルを実装モデルへと変換／詳細化していくための設計を行う。まず、システムをいくつかのサブシステムに分割し、アプリケーション・アーキテクチャーを決定する。その際、サブシステムへのプロセッサの割当てやRDBのアクセス方法などが主要な課題となる。各サブシステムまたは各層の関係、役割分担をクラス実装環境、データ量などがほぼ等しい場合、テンプレートのカスタマイズはあまり発生しない。実装環境やデータ量の違いが大きい場合、4層アーキテクチャの各層(UI,AP,BO,DBA)をどこへマッピングするかを設計しなければならない。

(4) UI設計フェーズ

(a) 提供されるテンプレート

UI設計フェーズでは、画面と帳票の外部設計を行う。ここでは、以下のテンプレートを用意する。

・画面／帳票レイアウト

・画面操作説明書

・入出力項目のチェック仕様書

・画面状態遷移図

(b) 新規テンプレート開発

ここでは、UIの外部仕様を定義する。要件獲得フェーズ作成した画面／帳票レイアウトに対し、コントロールの追加や入出力フィールドの定義といった肉付けを行っていく。また、ユースケースを使って、画面操作説明書や画面状態遷移図を作成する。

(c) テンプレートの利用

テンプレートをユーザとの打ち合わせに利用することで、作業の効率化が図れるとともに、レイアウトや操作の統一が図れる。

(5) オブジェクト設計フェーズ

(a) 提供されるテンプレート

オブジェクト設計フェーズでは、4層アーキテクチャーの各層ごとに、以下のテンプレートを提供する。

・クラス図

・機能仕様書

・オブジェクト交信図

・状態遷移図

・クラス仕様書（関数仕様書）

(b) 新規テンプレート開発

オブジェクト設計フェーズでは、システム設計の結果をもとに、分析モデルを実装するための仕様の詳細化や実装クラスの追加などを行う。オブジェクト設計では、4層アーキテクチャの各層ごとに、実装のためにクラスや関連、そのメソッドなどを完全に定義する。一般に、各層ごとに開発体制を分け、各層のノウハウの蓄積を行う。

(c) テンプレートの利用

実装環境やアーキテクチャーに大きな差がないとすると、各層のテンプレートに差分を追加することで、ユーザ用の仕様を作成する。実装環境が違う場合、分析から詳細化された業務ロジック(ビジネスルール)や制約などを再利用し、それらを適したサブシステムへ再配置する。実装に依存した実装クラスや関数は、見直す必要がある。

(6) 実装フェーズ

(a) 提供されるテンプレート

ここでは、以下のテンプレートを提供する。

・ソース、クラスライブラリ

・バイナリ部品(OCXなど)

(b) 新規テンプレート開発

このフェーズでは、オブジェクト設計のアウトプットを元にプログラミングを行う。オブジェクト設計と同様に、4層アーキテクチャの各層毎に分かれて作業を行う。ここでは、実装言語に対するかなりのノウハウが必要となる。

(c) テンプレートの利用

テンプレートとして提供されているクラスライブラリに、オブジェクト設計で定義した差分を追加して、ユーザクラスを作成する。

3．3．5 おわりに

テンプレート型開発は、ある問題領域において、アプリケーションの基本的枠組みを決め、その枠組みに沿ったひな形を提供することで、ソフトウェアの生産性向上を狙ったものである。従来から再利用による生産性向上施策はあったが、構造化技法をベースにしているため、仕様変更に柔軟に対応できない等の問題により、効果的に再利用されてこなかった。テンプレート型開発では、再利用性のキーとなる枠組みとして、ビジネスオブジェクトと4層アーキテクチャーを採用して、問題領域に普遍のオブジェクト(ビジネスオブジェクト)と他のオブジェクトを独立させることで、上流での再利用性向上が可能となる。

テンプレート型開発の適用は、現在、「販売管理業務」について開発が終わり、再利用の段階に入ってきた。今後、適用時のノウハウを蓄積してさらなる汎用化と利用方法の改善を継続していく予定である。

【参考文献】

1) J.ランボー他著, 羽生田栄一監訳：「オブジェクト指向方法論ＯＭＴ－モデル化と設計」, 株式会社トッパン(1992).

2) D.コールマン他著, 横河ヒューレット・パッカード株式会社カストマ教育センタ訳：「オブジェクトオリエンテッド開発設計論：The Fusion Method」, 株式会社トッパン(1994).

3) 奥井規晶, 本田直嗣, 田中正仁著：「オブジェクト指向システム開発－ビジネス・オブジェクトの設計と再利用」, 株式会社リックテレコム(1996).

4) 春木良且著：「オブジェクト指向実用講座」, 株式会社インプレス(1995)

5) 本位田真一, 青山幹雄, 深沢良彰, 中谷多哉子著：「オブジェクト指向分析・設計－開発現場に見る実践の秘訣」, 共立出版株式会社(1995).

3．4　M-base：「ドメインモデル≡計算モデル」を志向した開発技法

3．4．1　はじめに

近年、ワークステーションやパソコンの普及およびそれらをつなぐネットワークの普及と共に、業務の専門家が自ら情報システムを構築する必要性が高まっている。このような新しい動向に対応して、コンポーネントウェアやビジュアルプログラミングに代表されるような新しい開発技法が普及しはじめている。本技法では、業務の専門家が自ら作り、自ら使うようなエンドユーザコンピューティングの促進のためには、プログラミングの概念を排した新しいソフトウェアパラダイムが必要であるという認識から、基本的コンセプトとして、

A domain model ≡ a computation model （業務モデルと計算モデルの一致）

Analysis ≡ design ≡ programming（分析、設計、プログラミングの一体化）

を設定した。”モデリング＆シミュレーション”によってこれらのコンセプトを実現する分散オブジェクト指向設計技法を確立し、それに基づくアプリケーション開発環境M-baseを開発中である。マクロレベルではオブジェクト指向概念に基づく分散協調型問題解決モデルを用いて業務モデルの動的ふるまいを表現できるようなモデリングツールとそこで用いるスクリプト言語の基本機能を規定した。モデリングツールはアイコン操作を基本としたビジュアルな機能を実現した。本節では、本技法の目的と対象、モデリングプロセスの概要、モデリング＆シミュレーション、スクリプト言語の特徴、特定分野向きのコンポーネントの抽出方法などについて述べる。

3．4．2　目的と対象

(1)エンドユーザ

一般に、エンドユーザの範囲は広いが、本技法では、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基幹業務担当者および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業務の専門家を対象とし、特に今後急速に増大すると思われる後者の方により重点を置く。

(2)対象ソフトウエア

本技法では、「すべての日常的な業務をコンピュータ化する」、即ち、「日常的業務はマニュアル化でき、マニュアル化できればコンピュータ化できる」という観点から、オフィスでの業務用アプリケーションを中心にグループウェアやワークフローシステムなども対象とする。規模的には、中、小規模のアプリケーションソフトウエ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。

(3)開発・保守形態

本技法では、基本的コンセプトとして、

「ドメインモデル≡計算モデル」

「分析≡設計≡プログラミング」

をかかげた。これは、問題領域を分析してドメインモデルを構築した時点でソフトウエアの開発を完了させようというものである。即ち、

「ソフト開発＝モデリング＋シミュレーション」

という図式で表現されるように、問題領域のモデルを作成し、そのモデル上でシミュレーションして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実際のプログラムを自動生成するという方法である。この時、特定分野向きのコンポーネントウェアを導入して、プログラミングの複雑さを回避することも重要である。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エアを自ら開発し、自ら利用することを目標とするが、その実現に向けての技術課題は多い。そこで、研究の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を設定する。

3．4．3 モデリングプロセスの概要

(1)2階層モデル

今後増加していくと思われるエンドユーザコンピューティングの分野では、何をオブジェクトにするかを決定するためには、モデリングの初期の段階で、従来の技法であまり明確に示されていないオブジェクト群の動的なふるまいを記述することが重要である。ここでは、次に示すようなマクロモデルとミクロモデルの2階層モデルにより、基本的コンセプトを以下のように規定する。

(a)マクロモデルは、「ドメインモデル≡計算モデル」を実現するレベルであり、オブジェクト指向の分散協調型モデルとして位置づけられる。

(b)ミクロモデルは、「分析≡設計≡プログラミング」を実現するレベルであり、オブジェクト指向のクラス定義として位置づけられる。

(2)ドメインモデルの構築手順

本技法では、オブジェクト指向の概念の発生学的定義に基づいてモデリングプロセスの形式化[4]を行う。概要を図3.4-1に示す。分析フェーズで構築されるオブジェクトベースのドメインモデルを以下のようにOAM (Object-base Analysis Model) として表現する。

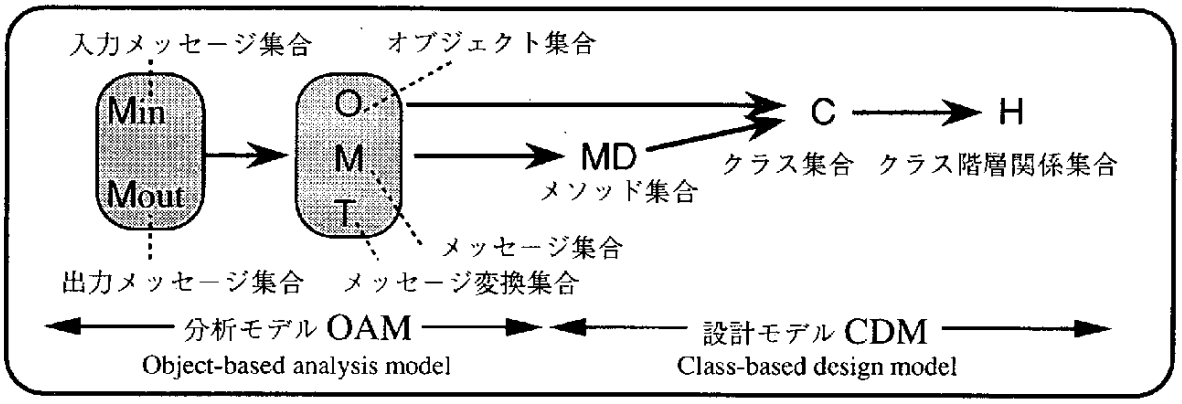


図3.4-1　M-Baseのモデリングプロセス

OAM = ｜O, M, T｜

O = {o[i]}

M = {m[i,j,n]}

T = {t[r]}

ドメインモデルOAMは、オブジェクトの集合O、メッセージの集合M、メッセージ変換の集合Tの３つ組で規定する。o[i]は、ドメインモデルに含まれるi番目のオブジェクトである。m[i,j,n] は、i番目のオブジェクトo[i]からj番目のオブジェクトo[j]へ送信されるn番目の種類のメッセージである。メッセージ変換の集合Tは、あるオブジェクトo[j]があるメッセージを受信した後、メッセージの列を送信するという関係を

t[r] : m[i,j,n] → {m[j,k1,n1], m[j,k2,n2], ...}

と表現したメッセージ変換の集合である。このドメインモデルは、２階層モデルの下位の層に対応する設計モデルに詳細化される。

設計モデルはクラスに基づいて構築されるので、以下のようにCDM (Class-based Design Model) として表現する。

CDM = ｜MD, C, H｜

設計モデルCDMは、メソッドの集合MD、クラスの集合C、クラスの階層関係の集合Hの３つ組で規定する。

3．4．4　開発環境の概要

M-baseの開発環境とアプリケーション・アーキテクチャの関係を図3.4-2に示す。開発環境は主に次のような構成要素からなる。

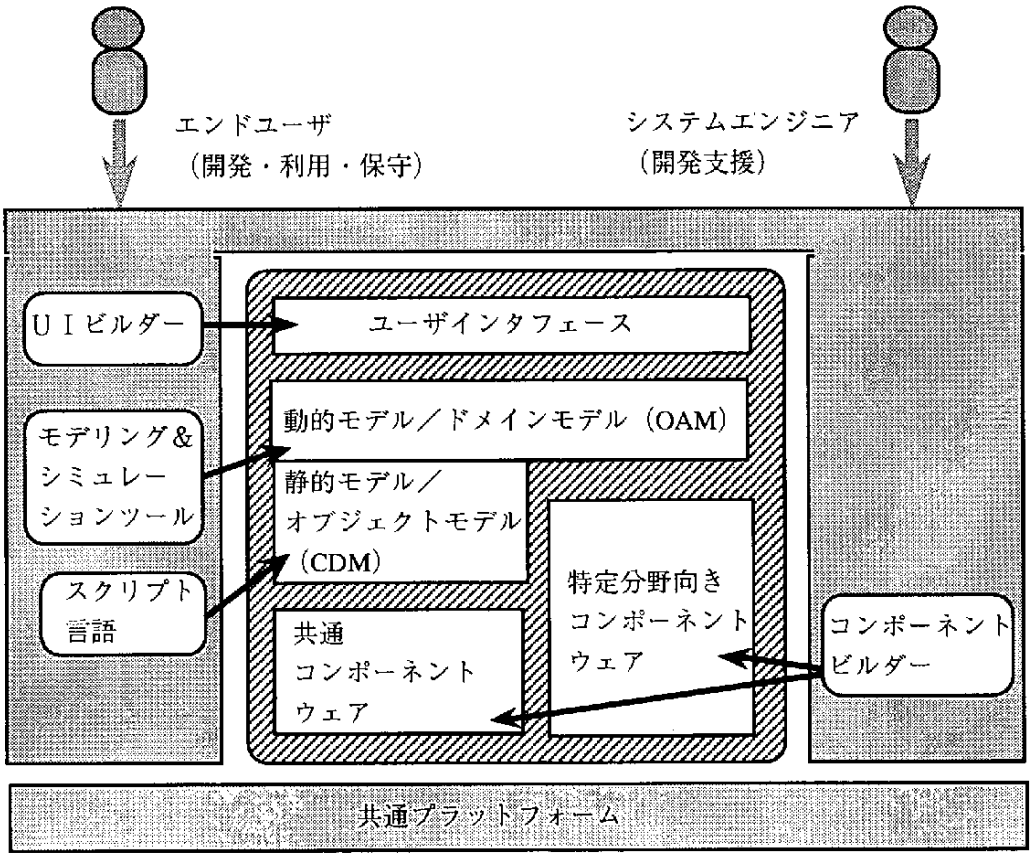


図3.4-2　M-baseの開発環境（外側）とアプリケーション・アーキテクチャ（内側）

・モデリング＆シミュレーションツール

・スクリプト言語

・UIビルダー

・コンポーネントビルダー

・共通プラットフォーム

これらのツールを用いて開発されるアプリケーションは大まかには次の３つの部分から構成される。

・ユーザインタフェース

・モデル

・コンポーネントウェア

モデルはアプリケーションに固有の処理を行う本体である。動的モデル（ドメインモデル）に対応するOAMの部分をモデリング＆シミュレーションツールを用いて作成する。静的モデル（オブジェクトモデル）に対応するCDMの部分をスクリプト言語を用いて作成する。ユーザインタフェースはそのアプリケーションのユーザとの対話処理部分であり、モデルと独立させることにより、クライアント／サーバシステムなどのシステム構成、あるいはクライアント端末のマルチプラットフォーム化が容易になる。その構築ツールとしてUIビルダーがある。コンポーネントウェアには分野に共通の基本部品と特定業務分野向きの部品がある。後者が充実してくるとエンドユーザによるアプリケーション構築が容易になる。共通プラットフォームはミドルウェアと基本ソフトウェアの部分で、オープンシステムや部品の流通の観点からは特に分散オブジェクト管理機能などが重要である。

3．4．5　モデリング＆シミュレーション

(1)「1オブジェクト1業務」の原則

オフィスの日常的業務（ルーチンワーク）は、メッセージ駆動型の分散協調型モデルを用いて次のように表現できる。

(a) ひとまとまりの日常的業務が割り当てられる一人または複数の人からなるグループを一つのオブジェクトに対応させる。

(b) 一人又はグループの間で相互の通信手段として用いられている書類、メモ、電話、郵便、口頭連絡などはすべてメッセージとみなす。

(c) 日常的業務における協調作業はメッセージの送受信によって遂行される。

本技法では、一つの業務を擬人化したオブジェクトに割り当てることにより、メタファーベースのモデル化を行う。実世界と同じように一つのオブジェクトに複数の業務を割り当てることも可能であるが、柔軟性と保守性を重視して「1オブジェクト1業務」の割り当てを原則とした。

(2) 基本機能

モデリング＆シミュレーションツールは、主にマウス操作により動的モデルを構築するツールである。動的モデルは、オブジェクトとメッセージから成り、メッセージパッシングで動作する。システムの動的モデルをアイコン表示などで視覚的に判りやすく表し、かつ、オブジェクト指向の概念を素直に表現することを目標としている。図3.4-3が画面の例である。この図は、左上の事務局オブジェクトが外部から会議開催準備のメッセージを受け取ると、右側の会議室オブジェクトやOHPオブジェクトに予約のメッセージを送り、左下の安部さん、馬場さん、千葉さんのオブジェクトに会議開催案内のメッセージを送るというモデルを作成しているところである。

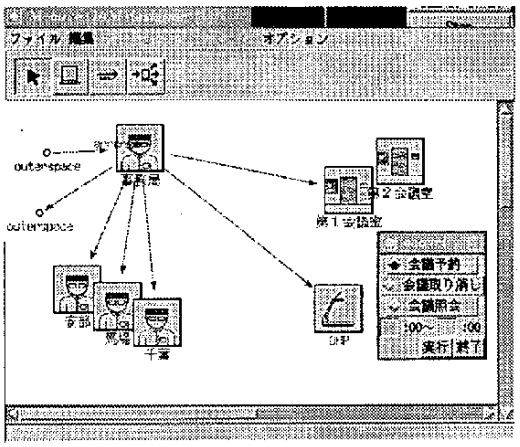


図3.4-3 M-baseによるモデリングの例

主な基本機能を以下に示す。

(a) オブジェクトの生成と表示

(b) メッセージの生成と表示

(c) メソッド定義

(d) メッセージ属性参照、変更

(e) メッセージ送信

(3) システム開発の手順

シミュレータでは、メッセージ変換の設定によって、SendMessageを含んだメソッドを自動的に作り出す。システム開発の手順は次のようになる。

(a) オブジェクトとメッセージを定義する。

(b) オブジェクトがあるメッセージを受け取り、その処理の中で幾つかのメッセージを送信するしくみをメッセージ変換として定義すると、SendMessageを含んだメソッドが自動生成される。

(c) 必要に応じてメソッド中に命令を追加する。

たとえば、図3.4-4に示すように、OfficeオブジェクトにArrangeメッセージを入力し、RoomオブジェクトにReserveメッセージ、AbeオブジェクトにAnnounceメッセージを出力するモデルを作成した場合、次のようなメソッド定義が自動生成される。

```
proc Office_Arrange { } {
    SendMessage Room Reserve
```

```
    SendMessage Abe Announce
  }
```

このメソッド定義に対し、会議室が予約できた場合だけ会議開催案内を出すようにユーザが変更を加えても良い。一例を以下に示す。

```
proc Office_Arrange {meeting day hour} {
  SendMessage Room Reserve $meeting $day $hour
    if {$result == "SUCCESS"}
      then {SendMessage Abe Announce $meeting $day $hour}
}
```

シミュレーションの実行例を図3.4-4と図3.4-5に示す。図3.4-4は、外部からの会議開催準備の指示メッセージを入力するために、吹きだしの形式でArrangeメソッドのパラメータ要求表示をして、会議名と日時を設定した画面である。この後、メッセージ送信をメニューで指示すると、図3.4-5の画面になる。この画面は、事務局オブジェクトが会議室予約管理オブジェクトに会議室予約のためのReserveメッセージを送信するところである。

開発済みのプロトプログラムは、実装はTcl/Tkを用いて行ない、作成したモデルから、Tcl/Tkファイルを出力するようにしている。現在、Java言語の版を開発中であり、スクリプト言語も次に述べるように、エンドユーザ向きのものを開発中である。

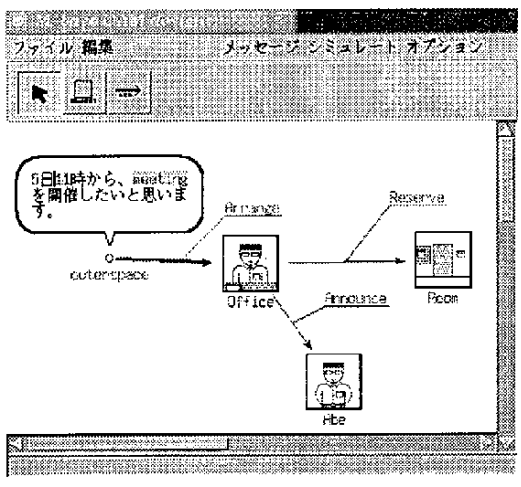


図3.4-4　M-baseによるシミュレーションの例（メッセージ送信前）

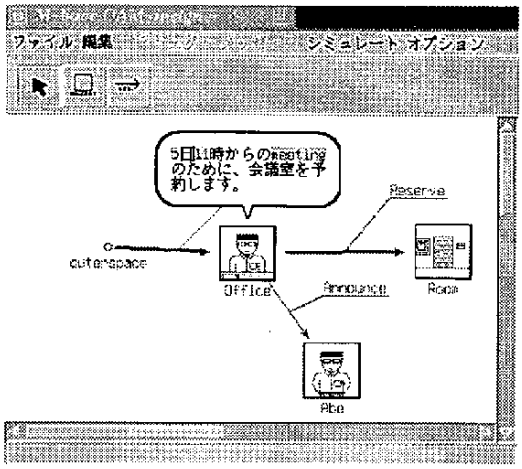


図3.4-5 M-baseによるシミュレーションの例（メッセージ送信後）

3．4．6 スクリプト言語

(1) 設計思想

グループウエアやワークフローシステムなどの分散協調型のアプリケーションソフトウェアを効率良く開発することを目的として、オブジェクト指向言語Hoopを開発中である。本言語は、分析・設計段階で使用することを目的としており、以下のような特徴を有する。

(a) オブジェクト間のメッセージの流れを記述するメッセージセット。

(b) ネットワーク上でのオブジェクトの柔軟な割り付けを可能とするオブジェクトのネスト構造。

(2) 分散協調型モデルの表現方法

(a) メッセージセット

Hoop のモデルでは、本来メッセージの流れを管理する立場のオブジェクト（業務担当者）がオブジェクト間に伝わるメッセージの流れを自然に記述するためにメッセージセットを導入した。例えば、ワークフローシステムにおいて全体のワークフローを管理するメタなシステムあるいはメタなオブジェクトが不要になり、純粋な分散協調型モデルを構築できる。このメッセージセットは、先に述べたOAMモデルにおけるメッセージ変換集合Mの各要素を時系列的に複数個まとめたものに対応する。

(b) メッセージセットの構文

メッセージセットの構文を以下に示すが、言語の詳細は文献[2]に詳しい。

M ::= M' || [cond] M'

M' ::= Ms || Mp || X

Ms ::= { M, M, ..., M }

X ::= (obj, msg, arg1, arg2, ... , argn) (n≧0)

objはオブジェクト、msgはメッセージ、condは条件、argnは引数、α ¦¦ β は α または β を意味する。

(c) 直列のメッセージセット

例えば、「ある書類を提出する時に、Aさん、Bさん、Cさんにハンコをもらわなければならず、かつAさん、Bさん、Cさんの順番でハンコをもらわなければならない」という場合を考える。このような逐次的に複数の処理が実行されるときは次のような直列のメッセージセットを記述する。

{ M1, M2, ... , Mn } (n≧1)

簡単な例として、{ (o1, m1), (o2, m2), (o3, m3) }というメッセージセットがo1送られたとする。オブジェクトo1はメッセージm1を実行後、オブジェクトo2に対して、{ (o2, m2), (o3, m3) }を送る。o2は同様にm2を実行後、o3に{ (o3, m3) }を送る。

(d) 並列のメッセージセット

次に、「会議を開催する時に、Aさん、Bさん、Cさんに出欠の問い合わせをする。ただし、Aさん、Bさん、Cさんのどの順番で返事がかえってきても良い」という場合を考える。このように複数の処理が並行して実行されるときは次のような並列のメッセージセットを記述する。

{ M1 ¦ M2 ¦ ... ¦ Mn } (n≧1)

簡単な例として、{ (o1, m1) ¦ (o2, m2) ¦ (o3, m3) }という並列メッセージの場合は、オブジェクトo1、o2、o3にそれぞれメッセージm1、m2、m3が送られる。

(3) 分散システムの実行方式

Hoopでは、分散協調すべきオブジェクト群をネットワーク上のノードへ割り当てる問題や個々のオブジェクトの機能が複雑になった場合のオブジェクト分割問題を解決するために以下の3種類のオブジェクトを導入する。その概要は図3.4-6に示す。

(a) ノードオブジェクト

ネットワーク上のノードに対応するオブジェクトであり、内部にプロセスオブジェクトを持つことができる。基本的にHoopではノードオブジェクトを用いてモデルを記述する。

(b) プロセスオブジェクト

解決すべき問題が複雑な場合に使用する。複数のプロセスの協調により、ノードオブジェクトが果たすべき機能を実現する。内部にサブオブジェクトを持つことができる。

(c) サブオブジェクト

補助的なオブジェクトである。内部にサブオブジェクトを持つことができる。ただし、ノードオブジェクトやプロセスオブジェクトと異なり、逐次処理に限られる。

オブジェクト単位に機能を分割していくときの一番大きな分割の単位がノードオブジェクト である。基本的には、ノードオブジェクト で問題を記述できる。しかし、ノードオブジェクト の機能が複雑な場合は、ノードオブジェクト 内をプロセスオブジェクトで分割することにより複雑さを解消する。サブオブジェクト については、ノードオブジェクト をプロセスオブジェクトで分割しても、まだ複雑である場合に使用する。

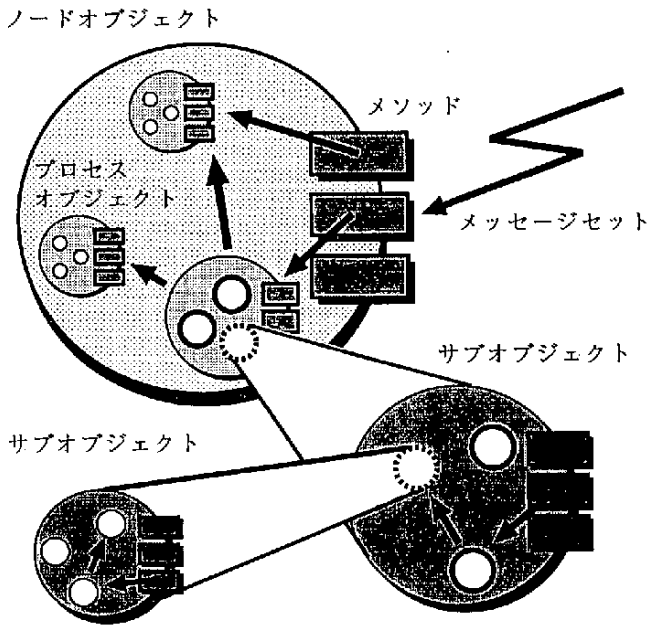


図3.4-6　3種類のオブジェクト

3．4．7　コンポーネントウェア抽出実験

(1) エンドユーザ主導の開発

エンドユーザ主導の開発における課題として、第1にコンピュータの専門的な知識の不足がある。そのため、エンドユーザにはコンピュータの専門的な知識を必要とする部分を見せないようにするシステムの支援が必要である。第2にはソフトウェア部品の粒度の問題がある。以下に示すような、アプリケーションを構成するソフトウェア部品の粒度と利用者との関係に注意しなければならない。

(a) 部品の粒度が小→利用者の負担増加

(b) 部品の粒度が大→利用者の自由度減少

(2) M-baseでのコンポーネント

現在、オブジェクト指向の世界では「ソフトウェア部品の粒度の問題」を解決するものとして、コンポー

ネント、フレームワーク、パターンといった研究が注目をあびており、各方面での研究が盛んである。コンポーネントとしては、基本コンポーネントと特定業務向けコンポーネントの２種類を考える。基本コンポーネントとは、情報システムを設計するうえで汎用的に利用できる部品である。例えば、GUI部品や基本データ型等がこれにあたる。基本コンポーネントだけでアプリケーションを開発する際に問題となるのが、業務モデルのオブジェクトに対応した適切なコンポーネントを選択することが難しいことである。そこで、特定業務領域に限定したコンポーネントを用意することが、これらの問題を解決するものとして有効であると考えている。

(３) 部品抽出の事例

この特定業務向けコンポーネントの抽出は、トップダウン的アプローチでは難しいので、業務の事例からボトムアップに行なう方法を支援する必要がある。部品抽出の事例として、次のような簡単な在庫管理を考える。利用者は、自分が食べたものを会計係にその都度申告する。会計係は、利用者からの申告を記録し、代金を集計し、月ごとに請求する。また、在庫が切れたり、利用者から欲しい商品のリクエストがあれば新しい商品を、買い出し係に発注する。買い出し係は、会計係から、購入依頼があったら買い出しに行く。また、会計係が管理しなければならないものとして、販売履歴表と商品管理表がある。

この業務内容のモデリングから、在庫管理システムに必要なコンポーネントとして次のようなものを抽出した。左側がモデルを構成するインスタンスオブジェクトであり、矢印の右側が抽出された部品である。

(a) 外部インタフェース→現実世界と計算機との橋渡しをする部品

(b) 販売履歴管理→名簿部品

(c) 在庫管理→商品管理表部品

(d) 会計係→受け付け部品

(４) 部品利用の事例

在庫管理システムで抽出したコンポーネントを会議開催事務処理システムに利用する場合を検討した。会議開催事務処理業務は、会議開催依頼を事務局が受け付け、必要な会議室や、OHP等の備品の予約管理を行ない、参加者に会議開催の通知をする、といったものである。　このモデリングを行い、先ほどの在庫管理で抽出されたコンポーネントの利用を検討した。左側がモデルを構成するインスタンスオブジェクトであり、矢印の右側が再利用される部品である。

(a) 外部インタフェース←現実世界と計算機との橋渡しをする部品

(b) 個人名簿管理←名簿部品

(c) 事務局←受け付け部品

会議室予約管理表と、備品予約管理表については利用できそうな部品が見当たらないので、新しく部品

を抽出する。これらは、管理する対象を時間で管理する部品であるので、スケジュール部品とする。

3．4．8 おわりに

業務の専門家が自ら作り、自ら使うようなエンドユーザコンピューティングの促進のために、プログラミングの概念を排した新しいソフトウェアパラダイムとして、業務モデルと計算モデルの一致により分析、設計、プログラミングの一体化を実現する開発方法論を提案した。これらのコンセプトを実現する分散オブジェクト指向設計技法を支援するアプリケーション開発環境M-baseおよびそのモデリング＆シミュレーションツール、スクリプト言語などのツールについて述べた。今後は、共通プラットフォーム上にこれらのツールを統合すると共に、具体的なアプリケーションの開発に適用し、実用性を評価する。

【参考文献】

1) 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、情報処理学会誌、Vol.37, No.1, pp.71-79, 1996.

2) 小西裕治、中所武司：分散協調型アプリケーションのためのオブジェクト指向分析・設計言語Hoopの設計とその記述実験、情報処理学会オブジェクト指向'96シンポジウム、pp.87-94, 1996.

3) 中所、小西、浜、吉岡：「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エア開発環境M-baseの開発技法、情報処理学会ソフトウェア工学研究会資料、96-SE-109, 109-2, pp.9-16, 1996.

4) T. Chusho, Y. Konishi and M. Yoshioka : M-base : An Application Development Environment for End-users Computing based on Message Flow, APSEC'96, 1996.

5) E. Gamma, R. Helm, R. Johnson and J. Vlissides : Design Patterns, Addison-Wesley, 1995.

6) D. E. Monarchi and G. I. Puhr : A research typology for object-oriented analysis and design, Comm. ACM, vol.35, no.9, pp.35-47, 1992.

3．5　Visual Modeling Technique（VMT）

現在すでに多くのOO方法論が紹介されている。しかしながらビジュアル・プログラミング、GUI開発あるいは開発プロセスにおいてプロトタイプと統合した方法を提示しているものはない。これらなしではOO開発でビジュアル・プログラミング・ツールの利点を十分生かしきることができない。VisualAgeを使ってOOアプリケーションを開発するシステム的な方法として、新しいアプローチ, Visual Modeling Technique(VMT)が1993年にIBM ITSO San Joseセンターで開発された。このVMTは既存のOO方法論として有効であると証明されているものを基礎としている。VMTはもとにしている方法から最良のアイデア、機能、ノーテーション、技法を取り出し、それらをOOシステム開発プロセスの中で統合し調和している。また、VisualAgeをOOA／OODプロトタイプからインプリメンテーション（実装）まで使うことにより分析・設計からコードへシームレスな開発ができる。詳しくは[2]を参照。

3．5．1　VMTのコンポーネント

VMTはそれぞれのOO方法論の良さを生かし弱さを埋め合わすように要素を組み合わせている。Object Modeling Technique(OMT)は包括的な方法論を提供し現在一番知られている方法論の一つである[3]。ゆえにOMTがバックボーンの方法論として選ばれVMTを通してそのノーテーションが使われている。

しかしながら、OMTは最初の段階でユーザ・インタラクションを取り込むメカニズムを提供していない。JacobsonのObject-Oriented Software Engineering(OOSE)[4]で示されているUse caseはこの問題のためのフォーマライズされた方法を与えている。VMTはユーザの要求、ユーザのアクティビティ、あるいはシステムがユーザに提供しなければならないサービスを見つけだし完全に記述するためにuse-case分析を適用している。このアプローチはユーザと開発されるシステムとのインタラクションの理解を助ける。加えて、OOSEのuse-case要求分析技術は特にプロトタイプによって得られた結果をフォーマライズするのに適している。

Rumbaughがクラスのアソシエーション（関係）をより強調しているのに対して、Wirfs-BrockのResposibility Driven Design(RDD)[5]方法論はオブジェクトの属性よりリスポンシビリティに重さをおいている。一旦クラスと関係が見つかれば、クラス間でのリスポンシビリティの配分を決める必要がある。このために、VMTはオブジェクトのリスポンシビリティ、コラボレーションを決定するのにRDDアプローチを使っている。CRCカードはクラスのデザイン、モデルでのクラスへのリスポンシビリティの配分のために用いている。

VMTはOMT、RDDあるいはOOSEで使われているsubsystemと同様の概念を使っている。オブジェクトのレスポンシビリティを分析した後、コントロールの流れ及びそのレスポンシビリティを満たすため要求

されたオブジェクトのメソッドを起動するメッセージパッシングを確定する。イベント・トレース・ダイアグラムと状態遷移図はオブジェクトのインタラクションとオブジェクトの状態の変化をモデルするのに使われる。イベント・トレースはオブジェクト間のメッセージ、すなわち起動されるメソッドを定義する。状態遷移図はメソッドによって影響される属性を理解するのに役に立つ。

3．5．2　VMTライフ・サイクル・モデル

VMT開発ライフ・サイクル（図3.5-1）はストラテジック・プランニングとビジネスエリア内あるいはそれをまたぐ組織のビジネス・プロセスのモデリングから始まる。ストラテジック・プランニングとビジネス・モデリングはアプリケーション開発プロジェクトの特定とプライオリティ付けを結果とする。しかしながら、時にはビジネス・モデリングの結果ではないビジネス上の問題を解決するためにアプリケーション開発を行うことが必要となる。ターゲット・アプリケーションのため高い要求ステートメントが起こり得ることは仮定される。このターゲット・アプリケーションを開発されるべきproductとする。productはビジネス・プランニング、開発、パッケーギングと配布という3つのステージからなるプロダクト・ライフ・サイクルをとおる。

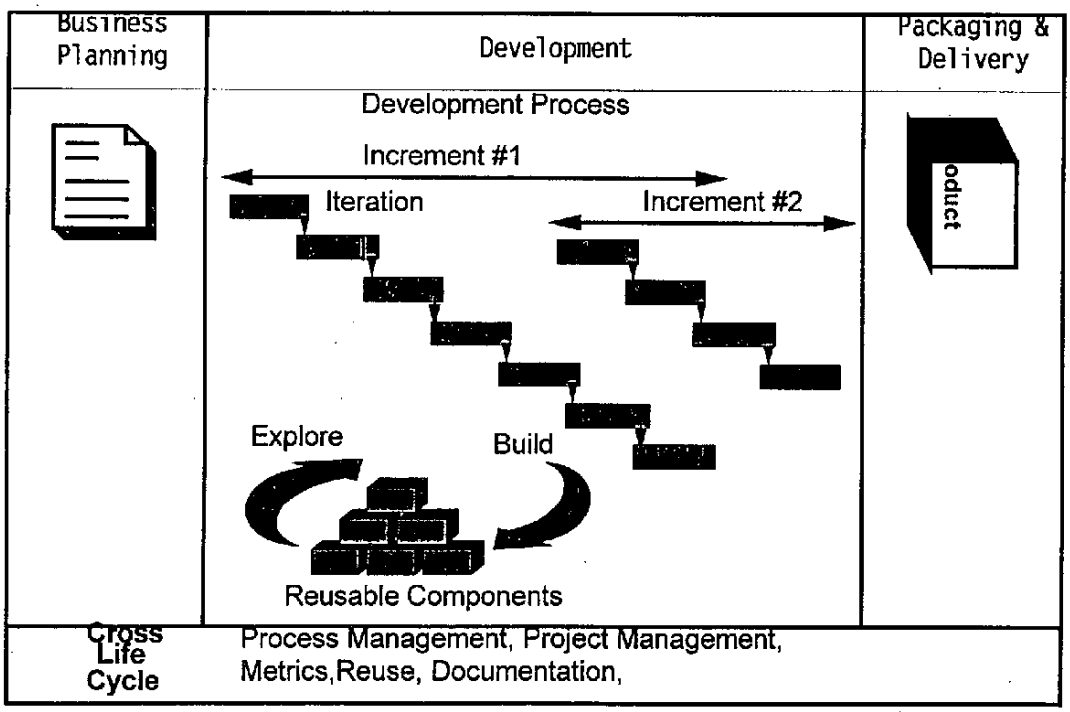


図3.5-1　VMT開発ライフ・サイクル

3．5．3　VMT開発プロセス

VMT開発プロセスはプロダクトの開発ステージに適応される。VMT開発プロセスはインタラクティブ、インクリメンタルの両方がある。

いくつかのイテレーションがあるプロジェクト・ライフ・サイクルの開発ステージで起こる。一つのイテレーションはそのイテレーションのプランニング期間、開発そしてアセスから成る（図3.5-2）。分析、設計、コーディング、テストは開発プロセスの開発の期間に行われる。

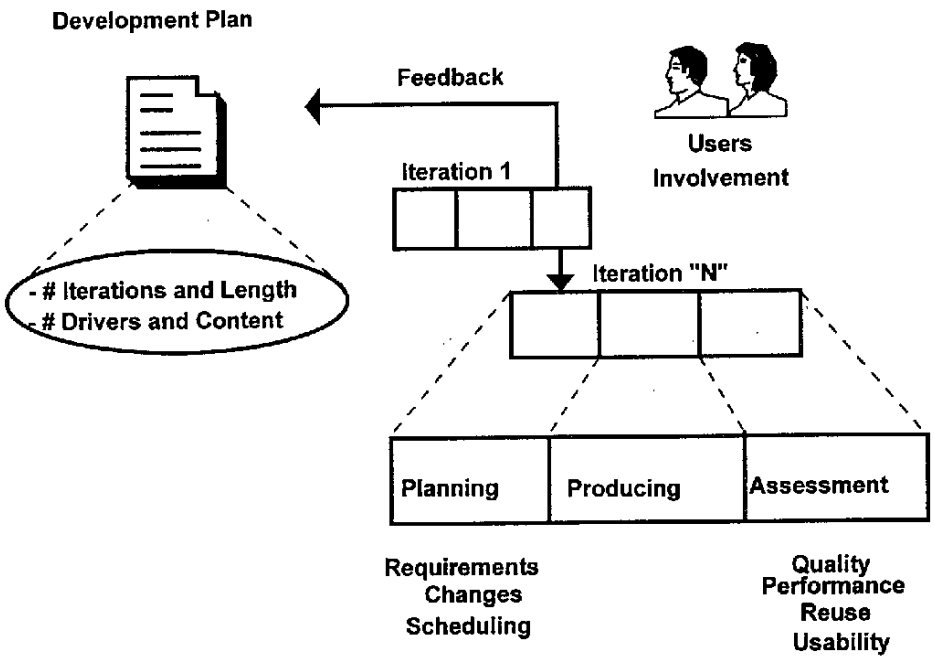


図3.5-2　VMT開発プロセス

プロトタイピングは開発プロセスの一部である。分析プロトタイピングはユーザのフィードバックとユーザの要求の確認のために使われる。分析者はハイレベルなユーザ・インタフェースのスケッチを開発し、VisualAgeのようなビジュアル・プログラミング・ツールでプロトタイプをし、アプリケーションの流れを示すために最小のコードを加える。これを用いながら分析者は視覚的にインタラクティブにアプリケーション・モデルを開発し、ユーザとアプリケーションの振る舞いを検証する。

設計プロトタイピングはアプリケーションの設計を検証するために使われ、その動くプロトタイプは最終プロダクトへと発展する。VisualAgeのようなビジュアル・プログラミング・ツールは理想の設計プロトタイプ環境を提供し“コンストラクション フロム パーツ（パーツからの構築）”パラダイムを助ける。オブジェクト・クラス、サブシステム、フレームワークといったいろいろなレベルの設計やコードをシームレスに発展するのを助ける。図3.5-3はVMT開発プロセスでどのようにプロトタイピングが使われるかを示す。

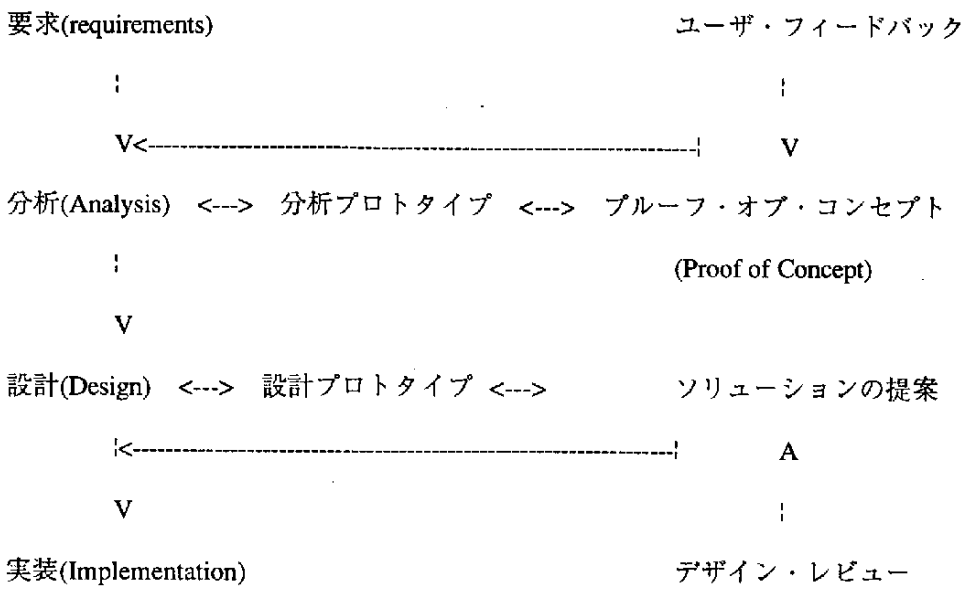


図3.5-3　VMT開発プロセスでのプロトタイピング

3．5．4　VMTモデリング・プロセス

図3.5-4はハイレベルなVMTのモデリング・プロセスを示している。

VMTモデリング・プロセスはOOAとOODを含んでいる。与えられた問題領域に対してOOAはそのシステムがその問題を解くために何をするかに注目するのに対してOODはそのシステムは実際にどのように実装されるかをみる。VMTのOOAは要求モデリングから始まりuse caseモデル、最初のオブジェクト・モデル、システムのユーザの概念モデルの評価のため分析プロトタイプをアウトプットとする。use caseモデルはシステムがアクターによって使われるかを定義しシステム・スコープを定める。オブジェクト・モデルはオブジェクト、その属性、サービス、クラスの関係をあらわす。そのオブジェクト・モデルでＣＲＣを使いながらクラスのリスポンシビリティを分析し定める。以下OOA・OODでの主なアクティビティとアウトプットを示す。ここではふれることができなかったがもう一つのVMTの強みであるOOA／OODからVisualAgeを用いた実装への流れは[2]を参照のこと。

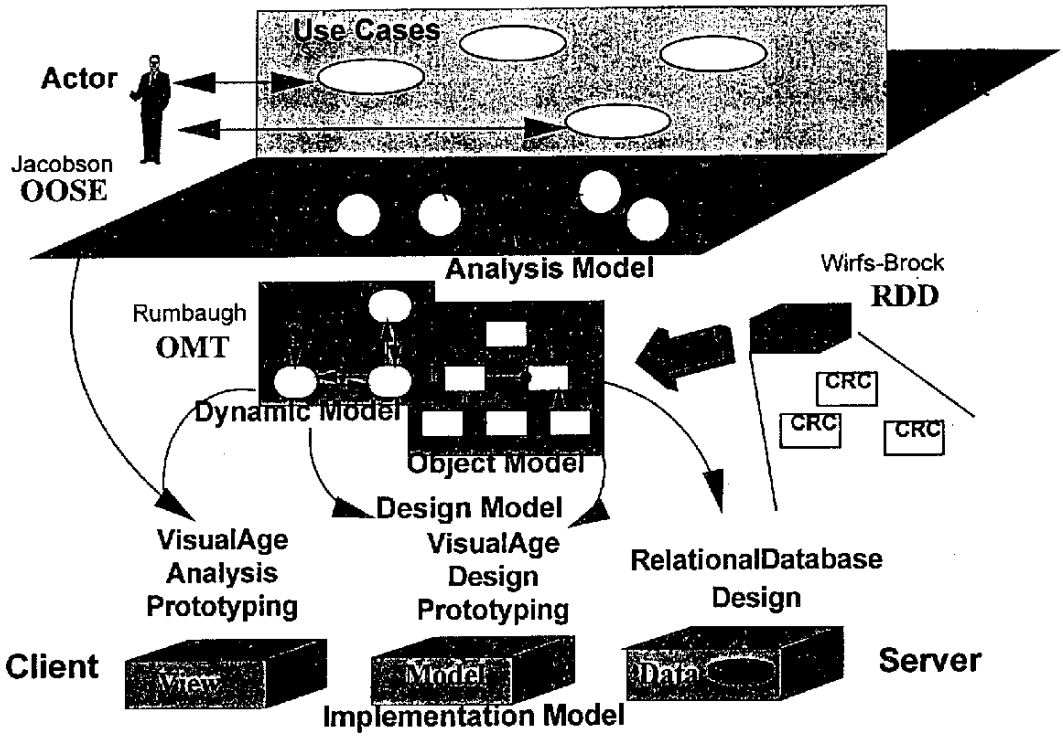


図3.5-4　ハイレベルなVMTのモデリング・プロセス

分析フェーズ

インプット	アクティビティ／テクニック	デリバラブル
アプリケーション要求 ユーザ・ロール ゴールとタスク ユーザ・インプット	use caseを集める	アクター ハイレベルuse case
ハイレベルuse case ユーザ・インプット	要求モデリング ユーザ・インタフェース 分析プロトタイプ	use case(refined) 要求モデル スケッチ GUIプロトタイプ
アプリケーション要求 ドメイン記述 要求モデル use case	要求分析	クラス，属性、関連， リスポンシビリティ
クラス，etc.	オブジェクト・モデリング	オブジェクト・モデル モデル辞書
オブジェクト・モデル use caseモデル モデル辞書	リスポンシビリティ分析	CRCカード オブジェクト・モデル

オブジェクト・モデル use caseモデル	ダイナミック・モデリング	イベント・トレース・ ダイアグラム 状態遷移図
オブジェクト・モデル ダイナミック・モデル CRCカード モデル辞書	確認，改良	オブジェクト・モデル ダイナミック・モデル CRCカード モデル辞書

設計フェーズ

インプット	アクティビティ／テクニック	デリバラブル
オブジェクト・モデル ダイナミック・モデル イベント・トレース・ ダイアグラム	システム分割	ハイレベル・システム・ アーキテクチャ ーサブシステム ーサブシステム・ インタフェース
オブジェクト・モデル サブシステム	マッピング	
サブシステム パフォーマンス考慮 既存システム考慮	end-to-endシステム設計	システム・ プラットフォーム選択

【参考文献】

1) W. Fang and A. So. "The visual modeling technique: An introduction and overview", ROAD, July-August, 1996.

2) D. Tkach, W. Fang and A. So. "Visual Modeling Technique: Object Technology with Visual Programming", Addison-Wesley, 1996 ISBN 0-8053-2574-3.

3) James Rumbaugh, et al. "Object-Oriented Modeling Technique", Prentice Hall, 1991. ISBN 0-13-629841-9.

4) Ivar Jacobson,et al. "Object-Oriented Software Engineering", Addison Wesley, 1992. ISBN 0-201-54435-0.

5) R. Wirfs-Brock and B. Wilkerson, "Designing Object-Oriented Software", Prentice Hall, 1990.

6) K. Deck, "CRC: Finding Objects the Easy Way", Object Magazine, November-December, 1993.

7) K. Beck and W.Cunningham, "A Laboratory for Teaching Object-Oriented Thinking", OOPSLA '89 Proceeding, 1989.

4．アプリケーション開発事例

4．アプリケーション開発事例

4．1　APPGALLERYによる開発事例

日本生命保険相互会社（以下、日本生命）では、APPGALLERYを適用した契約変更管理システムを構築した。ワークフローでシステム全体を構築し、個人作業部分を「デスクトップ・ワークフロー」としてAPPGALLERYで構築することで、システム全体をビジュアルに定義することができた。ここでは、コンポーネントウェアを意識したAPPGALLERYによる構築部分を中心に開発事例を示す。

4．1．1　日本生命の概要

日本生命では、個人保険だけで2700万件の契約数をもち、その規模も100万円の養老保険から25万人加入の企業年金までさまざまである。これらの多様な業務を支える組織および人員は、約140支社、2000支部、従業員数は約9万人であり、システムのユーザ層が広いのが特徴である。

4．1．2　契約管理業務の概要と問題点

日本生命・本部契約変更課では、保険契約での例外事務を集中して処理している。例外事務とは、通常のオンラインシステムで処理不可能な案件や、特殊処理に該当する案件を示す。これらの案件は通常、FAXや社内便などで本部に送付され処理されていた。しかし、トランザクションが月に13,000件の量で44種類、という少量かつ多品種の事務であるために、物流の時間的な問題、担当者間の連携困難、複雑な計算処理が原因で個人の知識習得が難しいなどの問題点を抱えていた。従来の事務処理フローを、図4.1-1に示す。

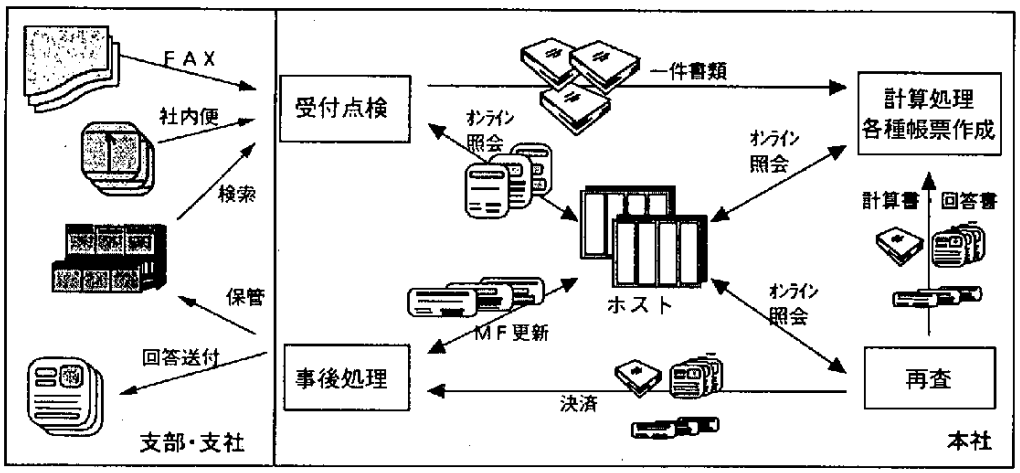


図4.1-1　事務処理フロー

事務処理の概要は以下のとおりである。

(a) 支部・支社から送付された処理依頼や他の必要な書類がFAXや社内便で送付される。

(b) 書類に不備がないかどうか、ホストの情報を参照しながら点検する。（受付点検）

(c) 書類1件毎に計算処理や各種帳票作成を行なう。担当者の持ち分の処理を完了すると、次の担当者に回覧する。業務内容によって再査者が回覧先を指定する。（計算処理・各種帳票作成）

(d) 作成された計算書や回答書は管理者がチェックし、決済を行なう。（再査）

(c)の処理では、案件毎に全く異なった計算を行い、その順序もさまざまである。これらの計算順序と結果の妥当性をチェックするのが再査者の役割であり、この人手への依存の高さがシステム化を阻害する大きな要因になっていた。

4．1．3　システム概要

本システムでは、契約変更課全体のワークフローにFlowmate、個人毎の作業フロー（デスクトップワークフロー）の開発ツールとして主にAPPGALLERYを使った。

図4.1-2にシステムイメージを示す。また、図4.1-3に機器構成を示す。

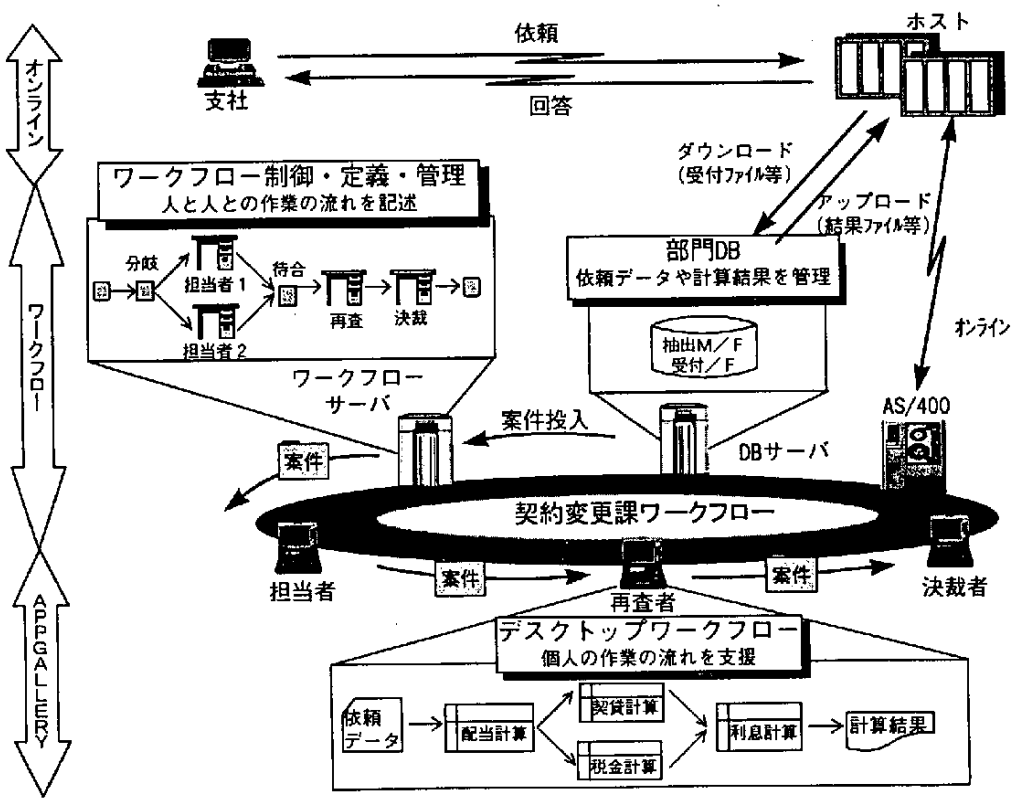


図4.1-2　システムイメージ

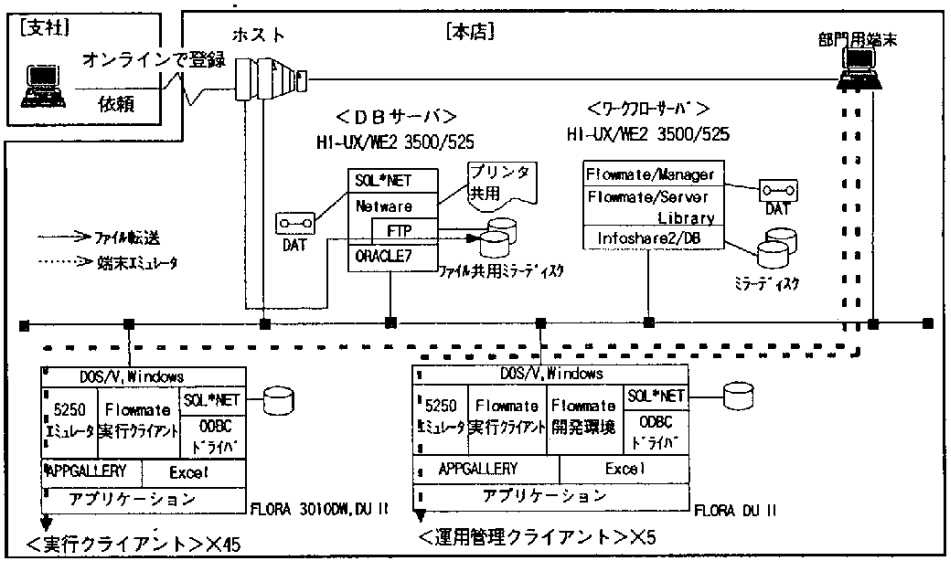


図4.1-3　機器構成

4．1．4　APPGALLERYの適用

(1) 開発規模

契約変更課で扱っているのは、名義変更、期間変更、貸付、解約など44種類の業務であり、そのうち一つの案件に複数の人間がかかわるワークフロー対象業務が26種類あった。この中から、主要7業務を取り出して業務の流れを検討し、金額通知、処分、査定の3つのカテゴリに分類し、コンポーネントの集約化を行なった。

APPGALLERYでの開発規模は、243キャンバス（APPGALLERYスクリプトで97Kstep）、ユーザ定義部品19アプリケーション（同4Kstep）、スタッフ40アプリケーション（同24Kstep）およびVisual Basicで32アプリケーションを作成した。Excelの計算シートはエンドユーザが作成した。

(2) 補完ツール

前にも示したとおり、本システムではクライアントのアプリケーションを構築するために、APPGALLERY以外のツールをコンポーネント（部品）と見なして補完的に使用した。これによって、ツールそれぞれの得意機能を引き出して統合する形で、システム開発することができた。例えばExcelは、エンドユーザが計算書を作成するために使われ、これをシステム部門がAPPGALLERYで組み合わせるといった使い方である。

使用したツールの切り分けを、図4.1-4に示す。

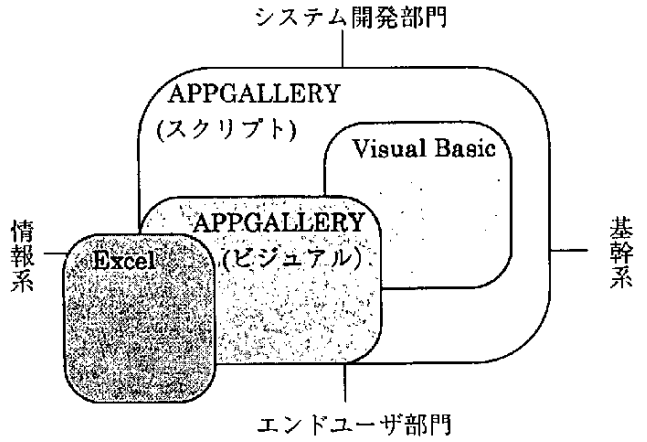


図4.1-4　ツールの適用範囲

システムの機能は、データや処理の制御といった情報系と、データベース検索や帳票出力といった基幹系の二つ大別できる。そこで、情報系をAPPGALLERYでビジュアルに構築し、定型的な処理や情報系との連携は、Visual BasicやAPPGALLERYのスクリプトでコーディングするという切り分けも行っている。

各ツールの実現機能を、表4.1-1に示す。

表4.1-1　ツールの実現機能

ツール名	実現機能	作成者
APPGALLERY	データ連携、処理制御	システム開発部門
Visual Basic	データベース検索・更新、帳票出力	システム開発部門
Excel	計算処理	エンドユーザ部門

(3) APPGALLERYの適用

エンドユーザ主導のインフラを整備したいという考えから、案件毎に処理手順の異なる点に着目すると、それぞれの担当者がPC上で行う作業は、人手に頼っている複雑な処理手順（＝作業フロー）があることがわかった。そして、この処理こそが、システム化の阻害要因になるということもわかった。そこで、個人の作業の流れをデスクトップワークフローに示し、この作業手順をビジュアルに定義するためにAPPGALLERYを採用した(図4.1-5)。さらに業務に必要なコンポーネントをビジュアルに並べてアプリケーション開発したい、というイメージが従来からあったので、処理フローを分析し、共通機能の部品化の検討に１年もの期間を費やした。

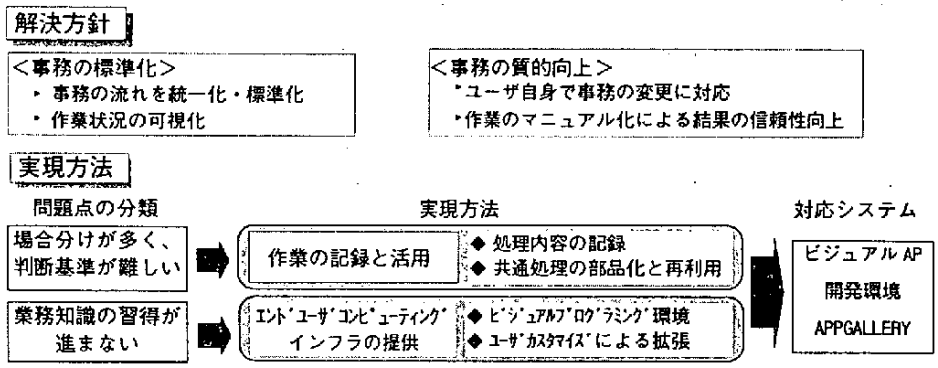


図4.1-5　解決方針と実現方法

まず、取扱いチェック、保険料計算、データの転記、帳票の作成などの細かい作業内容とその処理手順を整理した。計算、帳票作成などの個々の作業は「業務部品」と捉えてExcelやVisual Basicなどでそれぞれにアプリケーションを作成した。そしてこれら「業務部品」の組合せと実行順序を「作業フロー」としてAPPGALLERYで定義し、部品間のデータの流れを制御した。

APPGALLERYでの作業フロー定義例を、図4.1-6に示す。

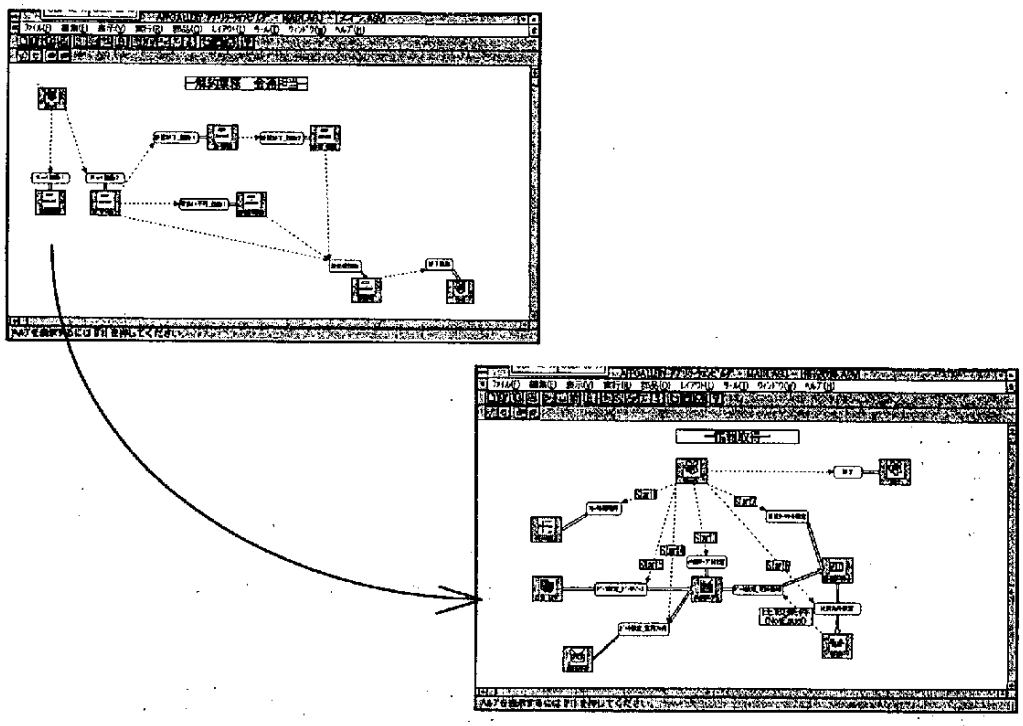


図4.1-6　APPGALLERYでの作業フロー定義例

この開発では、日立のシステム部門、ユーザ側システム部門およびエンドユーザと、システムに関わる立場によって、表4.1-2のように担当分野が明確に切り分けられた。

現段階では、EUC（End User Computing）の範囲を検討した場合、部品・スタッフをエンドユーザ自身に作成させることはまだ難しい。しかし、本システムのように必要な部品がインフラとして用意されていれば、契約管理業務のように常に変更（カスタマイズ）が発生する業務のシステム化が可能になり、保守面の問題も解決するため、ユーザの狙いと合致した。

表4.1-2　本システムでの役割分担

No.	対応部署	分担内容
1	日立	APPGALLERY共通部品・スタッフの提供とインフラ導入
2	顧客システム部門	業務部品（Excel, Visual Basic）及び業務アプリケーション開発
3	顧客エンドユーザ部門	スタッフを利用したアプリケーションのカスタマイズ

システム部門はプロトタイピングにより業務部品の組合せパターンを分析し、カスタマイズの発生しやすい「ファイル→Excelデータ読込み」「Excel→Excelデータ転記」などの機能をAPPGALLERY共通部品およびスタッフとした。共通的な機能を一まとめにする部品化と、それらの組み合わせ支援をするスタッフの作成で、作業フロー部分の開発は90%以上が部品とスタッフの組合せだけで実現でき、APPGALLERYのスクリプトコーディングはほとんど不要であった。その結果、開発工数の大幅削減に寄与した。

データ設定Excel－Excelスタッフ（Excelのセル単位で複写を行う）例を、図4.1-7に示す。

(a)複写元・先のファイルを指定する　　(b)複写元・先のセルを設定する

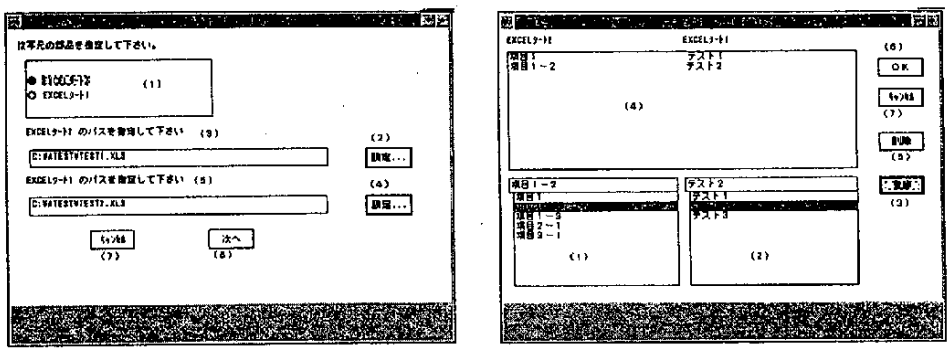


図4.1-7　共通スタッフ例

表4.1-3　共通部品・スタッフ一覧

項番	スタッフ名称	入力部品名称	出力部品名称	機　能
1	起動	AP起動部品		VBｱﾌﾟﾘｹｰｼｮﾝの起動を行い、終了を監視する。
2	データ設定	DBｱｸｾｽ部品	内部ﾃｰﾌﾞﾙ(Excel)	DBより検索キー(内部ﾃｰﾌﾞﾙに必要)に基づき情報を取得する。
3	Sheetコピー	Excelシート	Excelシート	Book内の全てのSheetを他のBookにコピーする。
4	データ設定	Excelシート	Excelシート	異なるBook間において、名称の付いているセル単位でデータを設定する。
5	条件式判定	Excelシート	イベント発生部品	条件式を作成し、条件が真ならば定義されたｲﾍﾞﾝﾄを発行する。
6	条件式判定	Excelシート×2	イベント発生部品	条件式を作成し、条件が真ならば定義されたｲﾍﾞﾝﾄを発行する。
7	データ設定	Excelシート	業務ファイル	名前の付いているセルの名称と値を、業務ﾌｧｲﾙ(CSV形式)に格納する。
8	データ設定	Excelシート	選択部品	選択部品にExcelよりデータを渡し、選択結果によりイベントを発生させる。
9	データ設定	Excelシート	内部ﾃｰﾌﾞﾙ(Excel)	Excelシートのデータを内部テーブルに設定する。
10	データ設定	Excelシート	比較部品（定数）	比較部品にExcelよりデータを渡し、選択結果によりイベントを発生させる。
11	データ設定	Excelシート×2	比較部品（項目間）	比較部品にExcelよりデータを渡し、選択結果によりイベントを発生させる。
12	情報設定	Excelシート		Excelシート部品のオブジェクト情報を解決する。
13	ｱﾄﾞｲﾝﾏｸﾛ組込み	Excelシート		Excelにｱﾄﾞｲﾝﾏｸﾛの組込み・解除を行う。
14	ＡＰ制御	Excelシート		Excelアプリケーションの表示・非表示・終了を行う
15	表示・非表示	Excelシート		Excelを表示・非表示状態にする。
16	閉じる	Excelシート		Excelアプリケーションを終了する（SAVE, NOSAVEを指定）
17	起動	Excel制御部品		Excel制御部品を起動する。
18	ﾌﾛｰ情報更新	Flowmate部品		案件中の文書を更新する。
19	ﾌﾛｰ情報取得	Flowmate部品		案件中の文書を取出す。
20	分岐情報設定	Flowmate部品		実項結果ファイルの生成。
21	公開ｲﾍﾞﾝﾄ設定	ｲﾍﾞﾝﾄ公開部品		ｷｬﾝﾊﾞｽ内で発生するｲﾍﾞﾝﾄ、部品化した時に外部に公開する。
22	表示	シート部品1		シート部品を表示する。（3種類のシート）
23	表示・非表示	シート部品2		シート部品を表示・非表示する。（データ設定中のシート）
24	表示	ﾅﾋﾞｹﾞｰｼｮﾝｼｰﾄ部品		ﾅﾋﾞｹﾞｰｼｮﾝｼｰﾄを表示する。
25	起動	階層部品		階層部品を起動する。
26	データ設定	業務ファイル	Excelシート	業務ﾌｧｲﾙ(CSV形式)のデータをExcelシートに設定する。
27	データ設定	業務ファイル	内部ﾃｰﾌﾞﾙ(Excel)	業務ﾌｧｲﾙ(CSV形式)のデータを内部テーブルに設定する。
28	ファイル削除	業務ファイル		業務ﾌｧｲﾙ(CSV形式)を削除する。
29	キャンバス終了	終了部品		キャンバスを終了するコネクタを生成する。
30	データ設定	内部ﾃｰﾌﾞﾙ(Excel)	Excelシート	内部テーブルのデータをExcelシートに設定する。
31	条件式判定	内部ﾃｰﾌﾞﾙ Excelシート	イベント発生部品	条件式を作成し、条件が真ならば定義されたｲﾍﾞﾝﾄを発行する。
32	条件式判定	内部ﾃｰﾌﾞﾙ(Excel)	イベント発生部品	条件式を作成し、条件が真ならば定義されたｲﾍﾞﾝﾄを発行する。
33	条件式判定	内部ﾃｰﾌﾞﾙ(Excel)×2	イベント発生部品	条件式を作成し、条件が真ならば定義されたｲﾍﾞﾝﾄを発行する。
34	データ設定	内部ﾃｰﾌﾞﾙ(Excel)	業務ファイル	内部テーブルの項目名称と内容を、業務ﾌｧｲﾙ(CSV形式)に格納する。
35	データ設定	内部ﾃｰﾌﾞﾙ(Excel)	選択部品	選択部品に内部ﾃｰﾌﾞﾙよりﾃﾞｰﾀを渡し、選択結果によりｲﾍﾞﾝﾄを発生させる。
36	データ設定	内部ﾃｰﾌﾞﾙ(Excel)	内部ﾃｰﾌﾞﾙ(Excel)	異なる内部ﾃｰﾌﾞﾙ間において、ﾃﾞｰﾀを設定する。
37	データ設定	内部ﾃｰﾌﾞﾙ(Excel)	比較部品（定数）	比較部品に内部テーブルよりデータを渡し、選択結果によりイベントを発生させる。
38	データ設定	内部ﾃｰﾌﾞﾙ(Excel)×2	比較部品（項目間）	比較部品に内部テーブルよりデータを渡し、選択結果によりイベントを発生させる。
39	情報設定	内部ﾃｰﾌﾞﾙ(Excel)		内部ﾃｰﾌﾞﾙ部品のオブジェクト情報を解決する。
40	表示・非表示・閉じる	内部ﾃｰﾌﾞﾙ(Excel)×２		比較部品に内部テーブルよりデータを渡し、選択結果によりイベントを発生させる。

（4）APPGALLERYとワークフローの連携

APPGALLERYで作成されたアプリケーションは、ワークフロー上での作業を自動化するために使われている。ワークフロー連携におけるそれぞれの役割は、下記のとおりである。

(a) ワークフロー：担当者間の文書の流れを制御する

(b) APPGALLERY：担当者（ノード）ごとの作業の流れを制御する（保険料計算等）

(c) 連携部分：サーバとユーザのインタフェースを提供する

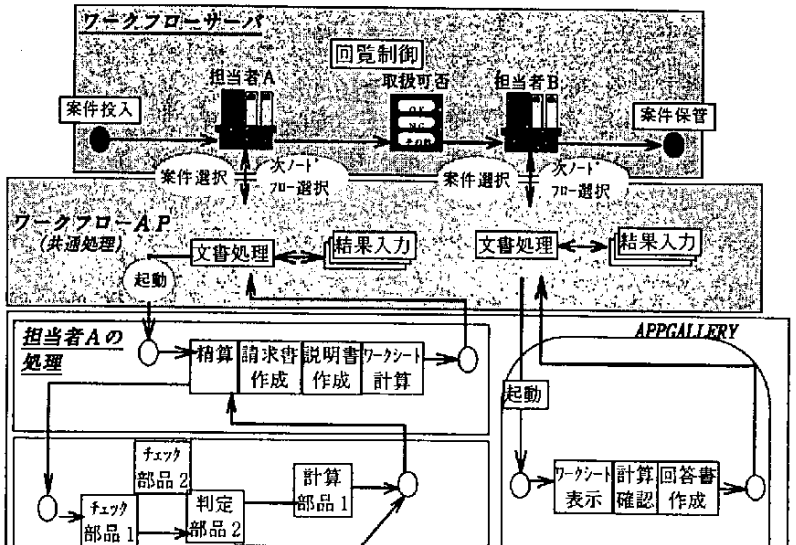


図4.1-8　APPGALLERYとワークフローの連携

連携によるメリットは、次の２点がある。まず、ユーザにとって、処理すべき案件の流れを意識せずに、担当処理をシームレスに行うことができる。また、案件毎に異なる計算順序も、連携されたアプリケーション内で表示されるので、ミスを減少できる。次に、システムの開発者にとっては、システム全体のフローと個々のPCで行われる作業が、階層的に表現され、しかもビジュアルに定義されているので、保守が容易になった。

（5）適用効果

図4.1-1に示したような従来の事務処理では、課員127名(パートスタッフ含む)、１件の処理にかかる最長日数16日、年間の紙の物流量が75万枚にもなっていた。しかし、このシステム導入後は、課員は84名、最長処理日数が7日、紙の物流は殆ど廃止した。図4.1-9に示したように、適用後の事務フローは、処理の流れを示す線の数が少なくなり、簡素化されて処理時間が短くなったことがわかる。

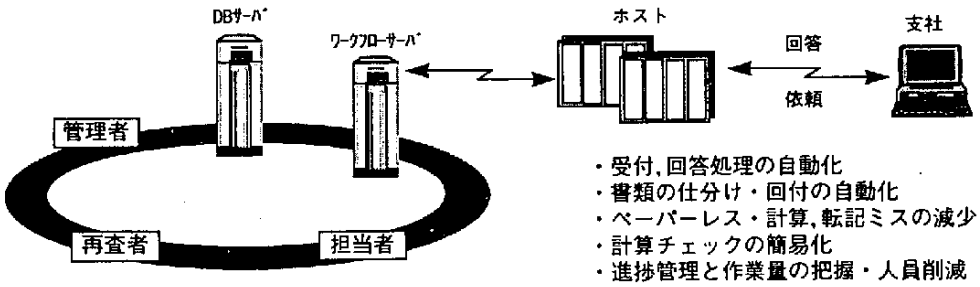


図4.1-9　システム適用後の事務フロー

保守面では、APPGALLERYの“キャンバス”を階層化することで、作成を容易にした。さらに、階層化とキャンバスでのビジュアル表現によって、これをプログラム仕様書と見立てることができ、ドキュメント作成の手間が省けた効果もある。その結果、システム完成期限の二週間前でも容易にユーザの要望をプログラムに取り込むことができた。

4．1．5　性能・運用上の留意点

本システムの性能面で最も問題となったのがクライアントPCのコンベンショナルメモリ等リソース容量であった（APPGALLERY 01-XXの場合）。この要因としては、APPGALLERYがOLE2.0を使用して複数アプリケーションを起動することや、他の前提ソフトがコンベンショナルメモリ上に常駐するなどが原因である。これらの問題は、運用上の環境で対策した。

信頼性ランクとしては、一部門に閉じたシステムであり紙での運用も可能なので、ランクCだが、ユーザの重要度から、以下のような方式で信頼性を確保している。

(a) システムリソースのミラーリングにより、ディスク障害の影響を回避する。

(b) 本番環境と同等環境を代替機に用意し、CPUダウン時に代替機での運用を可能にする。

(c) システム運用は、定時起動／停止パターンにより自動化し、オペレータミスを回避する。

(d) ホスト間のデータ転送を一括バッチでかつ自動化し、アクセス時間の短縮と、障害発生時の影響を少なくする。

(e) インテリジェントHUBを使用し、データが不必要に回線上に流れないようにする。

(f) 開発サーバ内にデータベースおよびワークフローサーバを同時に起動できる環境を新設し、障害サーバを開発サーバに切替えて業務を行う。

4．1．6 今後の課題

本システムは、平成8年3月から稼動している。今後の課題としては、以下のような点があげられる。

(1) 現状では、システム開発部門が作成したスタッフをエンドユーザが直接扱うことは少ないが、今後はエンドユーザ自身がスタッフを使ってアプリケーションをカスタマイズするなど、より高度なEUCを実践する。

(2) コンポーネントの適用範囲を広げ、他部署への適用拡大などのOAなどを含めた全社的な基幹系インフラとして展開する

(1)については、エンドユーザがAPPGALLERYのような開発ツールを使えるようにするためには、スタッフ（対話的に開発を支援する機能）が有効である。本システムでは、表4.1-3に示したような40種類の使用頻度が高いスタッフを開発した。エンドユーザに対するカスタマイズ教育を実施することでスタッフを使いこなせれば、エンドユーザおよびシステム部門両方の狙いであるEUCを実現できると考えている。

4．1．7 おわりに

コンポーネントウェアの開発事例について示した。本事例では、APPGALLERYとシステムのイメージが一致した点と、実開発に入る前に、コンポーネントウェアに適した業務モデリングを丁寧に行なった結果が、成功につながったと考える。また、本システムは、スクラップ&ビルドの考えを徹底させ、システムの開発投資を約3年の運用で回収できると見込まれている。

システム化を実現する特性にあったツールを選択できた点と、ユーザとシステム部門の新しい関係を築けた点、さらに償却期間を意識したスクラップ&ビルドの考え、これらの要因が、APPGALLERYでの開発には合致しているものと考えている。

【参考文献】

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

2) J. Martin：RAPID APPLICATION DEVELOPMENT, Macmillan Publishing (1991). [邦訳：ラピッド・アプリケーション・デベロップメント, 芦沢訳, リックテレコム]

3) Udell：オブジェクト指向の新展開コンポーネントウェア, 日経バイト, pp.277-289 (1994-6).

4) 初田賢司, 他：CSS開発に対応した開発方法論HIPACE, 日立評論, Vol.78, No.6, pp.19-22 (1996-5).

5) 日立製作所：日立システム開発方法論HIPACE.

4．2　HOLON/VPと適用事例

ネットワーク／マルチメディア時代を迎え、アプリケーションプログラムの開発と運用は変革を迫られている。巧みな部品化と、部品を作成、流用、再生産しながら自由に組合わせ、すばやく運用／変更していけるような環境が求められており、また、その基礎技術としてのオブジェクト指向技術が、ますます重要性を増しつつある。

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り、NEC-ORBは分散アプリケーションの基盤となるCORBA2.0準拠のミドルウェアである。これらは、今後のネットワークアプリケーションのための開発環境と分散基盤として、特に以下の要件を満たすことを目標として開発された。

・大規模アプリケーションへの対応

特に部品化・再利用の強化、分散システム構築基盤の提供、多人数共同開発支援等が重要である。

・部品のカスタム化機能

独自の部品を作成する機能を提供し、顧客ニーズへ肌理細かく対応する。

・マルチメディアの活用

高度で効果的なプレゼンテーション機能として必須になってきている。

・インターネット・イントラネット対応

Web、Java等のインターネット技術への対応も同様に最優先課題になっている。

4．2．1　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。

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4.2-1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動作確認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。

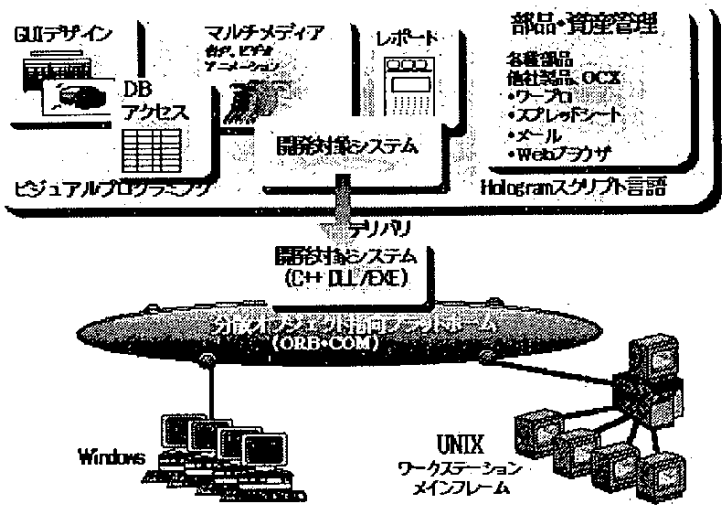


図4.2-1　HOLON/VPのシステム構成

4．2．2　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における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。

さらに、アプリケーションコンポーザで作成したビジュアルプログラム（APコンポーザ部品）を、APモジュールボックスとして部品化し、別のプログラムで再利用することが可能であり、これによりビジュアルプログラムの階層化も可能となり、大規模なアプリケーションの開発や保守への効果が期待される。

図4.2-2に簡単な例を示す。これは月次ファイルを読み込み、審査の上決済報告書を作成する処理を5個の部品のポート線でつないで組合せているものである。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表わす。

図4.2-2で部品間は線でつないでいるが、実線はコントロール、破線はデータを示し、例えばメニューの決済報告ボタンをクリックすると、コントロール線に従って決済報告部品の月次取得メソッドが呼び出され、この時画面上の月次を示すテキストが月次取得メソッドへの引数 month に設定される。

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。

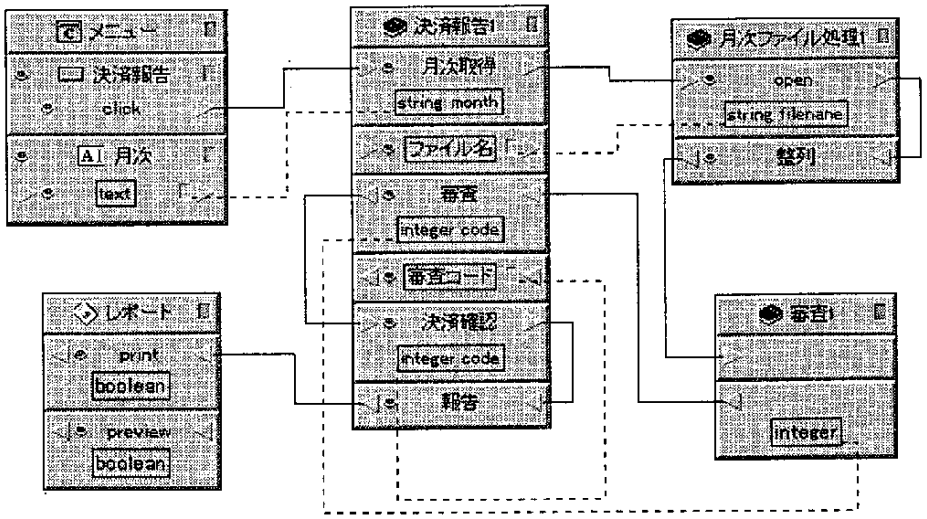


図4.2-2　ビジュアルプログラミングの例

(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HOLON/VPで提供されている部品を示す。

(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。

データベース関連の特徴的な機能としては、以下のものがある。

・ストアドプロシージャ他データベース機能の設計を支援するツール

ストアドプロシージャの作成からテスト実行に加え、トリガやインデックス作成など、関連する作業までを効率的に行える機能を提供している。ストアドプロシージャを利用することによって (ア) ネットワークトラフィックの減少および性能の向上、(イ) 生産性の向上、(ウ) 信頼性の向上と資源の節約、(エ) 安全性の向上、等の効果が得られる。

・コンパクトなリレーショナルデータベース(HOLON-DB)

HOLON-DBへのアクセスは、他のリレーショナルデータベースと同様にキュエリクラスを用いたり、言語を用いたりして操作することができる。また、リモートに存在するDBサーバーから、ローカルに存在するHOLON-DBへダウンロードしてからSQLで更に加工したりといった処理が可能となる。

・リモートサーバから大量のデータを高速にダウンロードするジェットストリーム機能

ネットワークの負荷の高い環境において検索処理を高速化する機能として、リモートに存在する

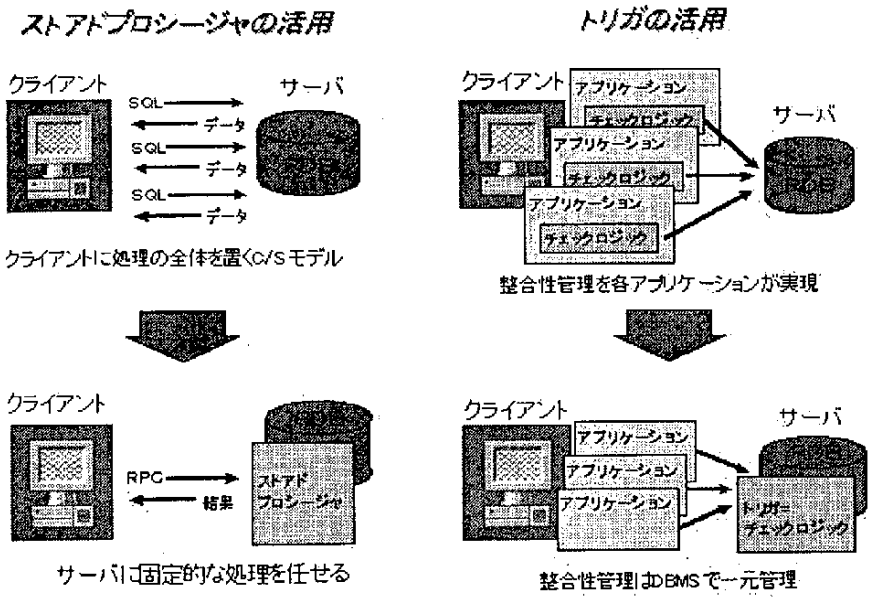


図4.2-3　ストアドプロシージャの効果

ORACLE中のデータを高速にダウンロードする機能を提供。

(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。

さらに、データと表示系が分離されているため、マルチビュー対応が可能であり、より分りやすい設計を行うことができる。

(c) 図形部品

図形部品は、数種類の図形を組合わせてGUIを定義するための部品である。図形部品には、図形部品エディタで作成した図形データの表示と、その図形データに対する各種図形 (基本図形やアニメーションの図形) の追加、プロパティの変更等の機能がある。

図形部品により、アニメーション図形、ビットマップの背景透過、構成部品単位のイベント処理等が可能になり、極めて魅力的な利用者インターフェースを実現することができる。

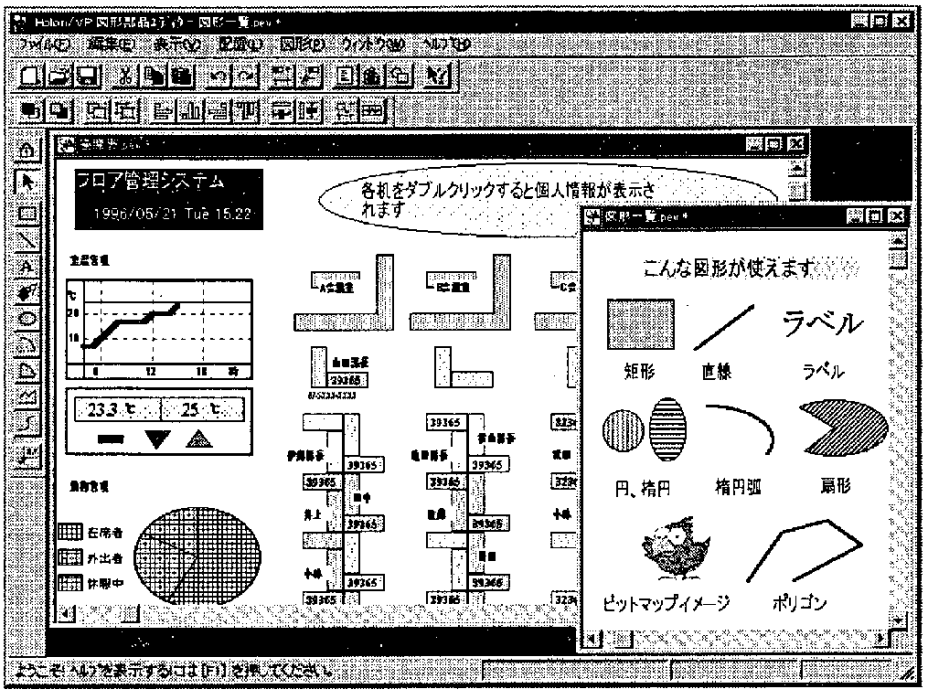


図4.2-4　図形部品エディタでの図形部品作成例

(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、ボタン(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、スピンボタン)、選択部品(リストボックス、コンボボックス)、スライダ、プログレスバー、テキスト部品(1行、複数行、リッチテキスト、ラベル)、スクロールバー、線分、図形、表、グラフ、フォーム、マルチメディア部品(後述)等が提供されている。

また、OLEサーバとOCXをHOLONのアプリケーション画面に配置することも可能である。一般に流通しているOLEサーバやOCX部品をHOLONの画面内に容易に取り込むことができ、さらにOLEオートメーション機能によって、Hologramの中で既存部品と違和感なくプログラミングすることができる。

これらの部品を組み合わせてGUI画面を構成するツールがプレゼンテーションデザイナである。プレゼンテーションデザイナでは、アプリケーション画面上の一部をGUIカスタム部品と呼ぶ独立した部品種別として定義・登録することができ、繰り返し利用される画面上の機能や、複数のアプリケーションの中で共通な処理画面（画面の一部も可能）を部品化することができる。

また、共通的な画面を定義し、これを継承して新しい画面を作成することができる画面クラス継承機能により、大規模な開発等において画面／処理設計の共通化と再利用を図ることが可能になる。

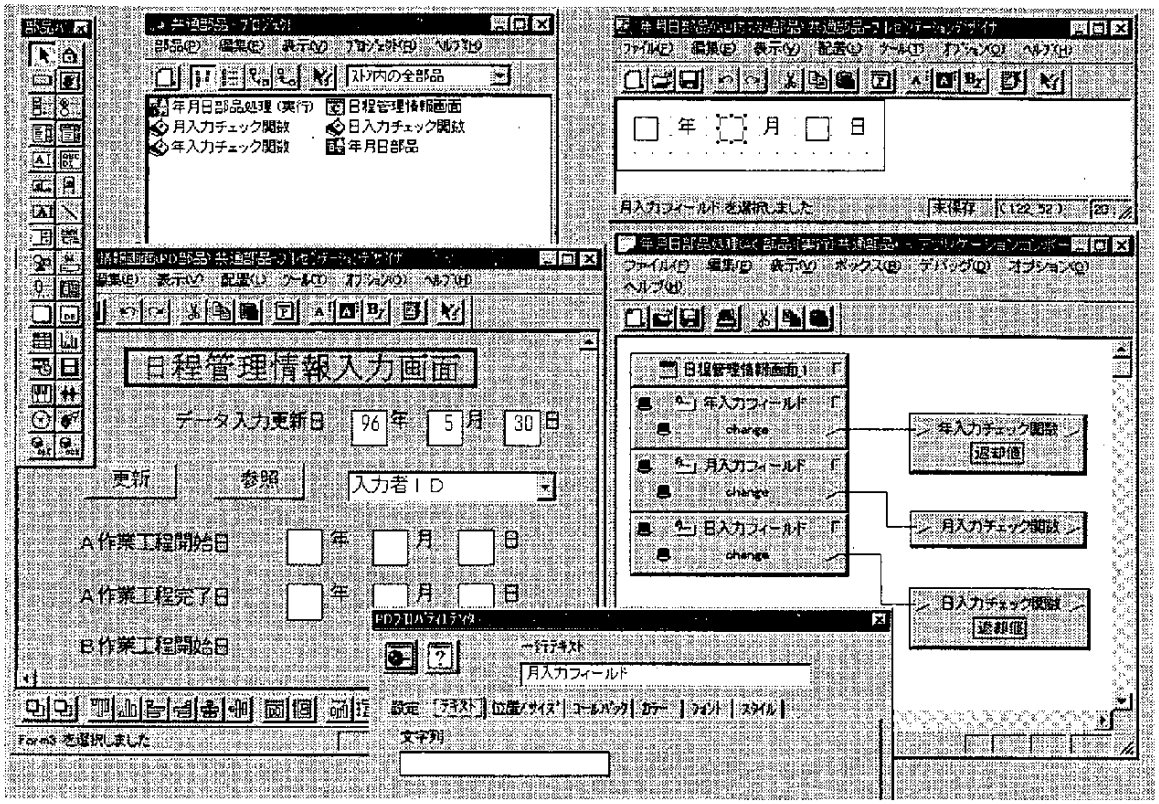


図4.2-5 プレゼンテーションデサイナによるGUIカスタム部品の定義例

(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/フッターと明細を含み、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。また、印刷・プレビュー時に呼び出されるコールバックメソッドが設定可能であり、印刷時に動的な計算結果等を確定させたい場合に有用である。

(f) 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、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を組み込むことができる。

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(別売)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本のタイムライン上にメディアとしてまとめたものである。さらに、タイムライン上でループを指定したり、マウスクリックによりタイムライン上の任意の位置に分岐を行なうロジック記述機能を提供しており、例えばメディアのオートリピートや、クリックする場所によって動作を変化させるといったインタラクティブな効果を持たせることができる。

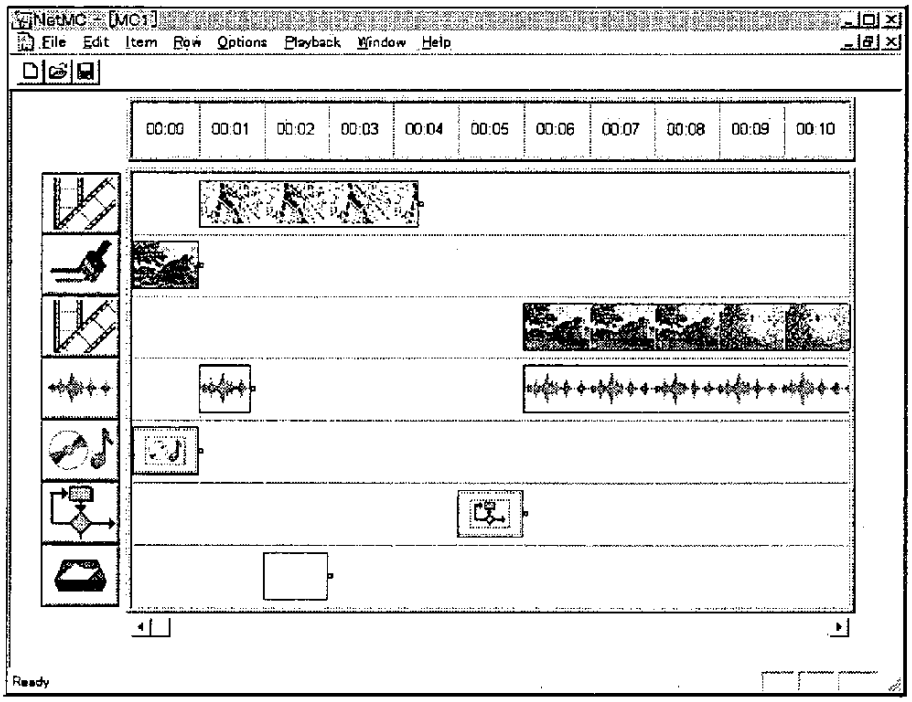


図4.2-6　タイムライン上の制御機能

(g) ネットワーク部品

各種部品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。

(i) CORBA2.0の利用

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NEC-ORB を提供しており(別売)、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、アプリケーションコンポーザを用いて、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。

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、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、送信の3種類に分類される。

図4.2-7にORBを用いた三層構造の分散アプリケーション例を示す。

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。

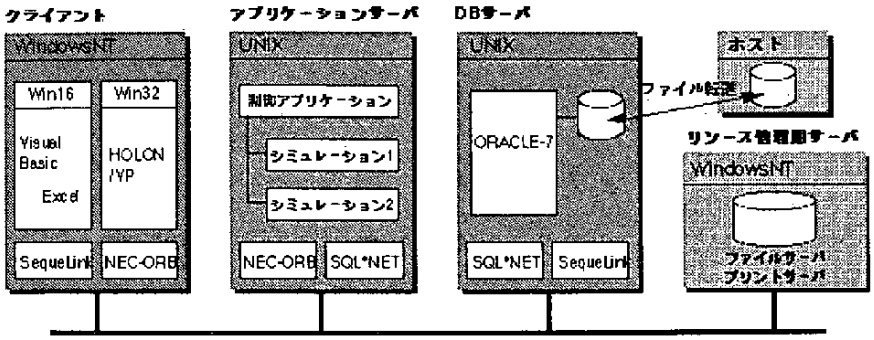


図4.2-7　ORBを用いた三層構造分散システム例

(ii) OLTPの利用

OLTP Tuxedo(TP-TUX、PCTUX)クライアントのインタフェースをサポートするOLTPクライアント部品を提供している。OLTPクライアント部品を利用してHologramプログラムを作成することにより、サーバ上のOLTPサービスアプリケーションとやりとりすることができる。

(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。

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。両部品は、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)における混在が可能で、スクリプト部品は検証が終わった段階でC++部品に変換され、高効率の部品として蓄積してゆくことができる。

(4) デバッグ

アプリケーションコンポーザ上でビジュアルプログラムをデバッグ実行する機能が提供されており、ビジュアルプログラムの1コントロール線を 1ステップとして、デバッグ実行の開始 (再開)／停止、ステップ単位の実行、ブレークポイントの設定などの実行制御が可能であり、変数やデータの値をモニタリング

するためのウォッチボックスなどのツールが提供されている。

これらの機能はスクリプトエディタのデバッグ機能と連動しており、ビジュアルプログラムのデバッグとHologramスクリプトのデバッグを一体となって行なうことができる。

(5)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。

また、ネットワーク上の共有ドライブ上に共有ライブラリを定義することで、共有部品の更新時には関連プロジェクトが矛盾なく管理できる、共同開発支援機構が提供されている。

(6)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。

(7)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、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、改変や発展は個々の部品の追加/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。

4．2．3　HOLON/VPの効果と今後の展開

最初に挙げた今後のネットワークアプリケーションの開発環境への要件に対する、現版のHOLON/VPの効果について述べる。

まず、大規模アプリケーションへの対応については、アプリケーションコンポーザのビジュアルプログラム階層化機能、図形部品エディタによる部品作成機能、プレゼンテーションデザイナによるGUIカスタム部品登録機能と既存画面の継承機能、共同開発支援機能、ORBによる分散オブジェクト基盤等が効果的である。

次に顧客ニーズへの肌理の細かい対応は、図形部品エディタによる部品のカスタム化が有用である。

さらに、高度で効果的なプレゼンテーション機能に関しては、簡易アニメーション等の他に、タイムライン上の分岐・ループ機能を備えたマルチメディア機能により補強される。

今後は、以下のような強化を予定している。

(1)インターネット／イントラネット対応

・CGI、Plug-in等 Web機構への対応

・StarEnterprise等グループウェアへの対応

・Java言語への対応など

(2)オブジェクト指向環境としてのブラッシュアップ

・フレームワークを始めとする分析・設計フェーズとの連動

・各部品のブラッシュアップ(マルチメディア、アニメーションなど)

・オープンなインテグレーションなど

(3)ネットワーク環境としてのブラッシュアップ

・ネットワークインテグレーション

・ネットワークデバッグなど

4．2．4　HOLON/VPの適用事例

HOLON/VPはいくつかのプロジェクトに適用されているが、特に、部品のカスタム化が要求されたり、大規模な開発に対して効果があり、比較的大きく継続的なプロジェクトに採用されている。それらの適用事例を以下に挙げておく。

(1)人事情報管理システム

静止画や動画を含む社員情報の検索表示

(2)ソフトウェア品質管理システム

テストの進捗や結果を集計・グラフ化して分析

(3)商品売上分析システム

売上実績を集計・グラフ化して比較分析、多種多様な情報の統合表示、分散／大規模DBの活用、高度なプレゼンテーション、誰でも使える操作性

(4)資金運用管理システム

預り資産の運用管理、実際の利回りと運用戦略の比較・分析、運用手法に応じた期待収益率とリスクの算出

(5)教育費用管理システム

社内教育の受講者は費用実績の管理集計

(6)海外旅行相談システム

観光地の情報をマルチメディアで紹介

商品売上分析システムや資金運用管理システムのような基幹的な業務を担うシステムがHOLON/VP適用の典型的な事例であり、いずれも、大規模アプリケーション対応、顧客ニーズに合ったプレゼンテーションのカスタム化、高信頼の分散オブジェクト基盤、等が要求されているものである。

その中でも特に、顧客独自のプレゼンテーション部品のカスタム化(個別対応)と部品の汎用化は、汎用コンポーネントウェアを提供する側としては、どこまでを汎用部品として開発・提供するかのバランスが難しい。現実問題として、業務に適用する場合には、画面スペースの問題や操作の一貫性の要求によって、GUIに限ってもカスタム化は必須である。特にGUI部品のカスタム化要求に対しては、HOLON/VPでは図形部品作成エディタを提供することによって応えている。

一方、業務ロジックに関しては、顧客毎の再利用はもちろん可能であるが、汎用化・再利用は本質的に難しいと言える。扱う製品のジャンル、客層分類や機会損失に対する評価基準等、顧客独自でかつ多くは非公開の基準と意志決定機構に基づいているためである。

4．2．5　コンポーネントウェアの可能性

コンポーネントウェアは従来のソフトウェア開発のあり方やソフトウェア産業構造そのものに変革を与え得るものとして期待されている。アプリケーションを構成する様々なソフトウェアモジュールをプラグ&プレイ型のソフトウェア部品として提供し、それを組み合せる開発環境を提供することで、エンドユーザが自らコンポーネントを調達し、アプリケーションを開発する場面も想定される。さらに、各コンポーネント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。

また、コンポーネントと従来のソフトウェア部品の違いとして、コンポーネントは、ブラックボックス利用、ソースではなくオブジェクトコード利用、クラスではなくインスタンス利用、実装ではなくインタフェース利用、であると言われる。

このような性質は、ソフトウェアの部品化、部品の反復利用による品質の安定、部品の流通、といった面を促進する。また、使用したい部品の仕様の検証の仕方、バグに対する責任の所在、流通(販売)の仕組みをどうするか、といった問題点も議論されている。

これらの問題点はいずれも部品側の問題であるが、一方、部品を組み合わせてアプリケーションを構築する開発環境の側の問題も無視できないと思われる。

既存の部品を組み合わせるだけでできる比較的小規模なアプリケーションにおいては、コンポーネントウェアのアプローチで開発効率や品質は劇的に改善されると期待されるが、多くの部品を組み合わせて作る高機能で大きな部品(モジュール)の再利用をいかに効率化するかについてはあまり言及されていないようで

ある。つまり、開発規模が比較的大きい場合に避けて通れないと思われる、このようなレベルの部品の再利用技術を提供することも考えなくてはならない。

HOLON/VPでは、部分的な解ではあるが、共通的な画面を継承して新しい画面を作成する画面クラス継承機能を提供している。これは従来のようなクラスレベルではなく、インスタンスレベルの継承であり、モデルや方法論としてもまだ十分に確立していない技術領域である。

いずれにしても、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に大きな影響を及ぼす可能性を秘めていることは確かであり、問題点を着実に解決してゆくことが肝要であろう。

４．３　ＩＰアプリケーション開発事例

４．３．１ はじめに

Intelligent Pad(以降、IPと略記する)を使ったアプリケーションシステム開発の事例として製薬分野向けの「GCP/PMS支援システム」を取り上げ、IPのアプリケーションへの適用の観点から述べる。

当アプリケーションの開発スタイルとしては、IPを使った「GCP/PMS支援システム構築部品」ライブラリ(ひな型アプリケーションを含む)を標準的に提供し、その部品ライブラリをそのまま使用あるいは修正することによって各顧客向けシステムを構築するものである。

４．３．２ 医薬品開発プロセスとGCP/PMS

医薬品の開発は1新薬10年～15年の期間と100億円以上のコストがかかるとも言われており、また新規物質の発見後の製品化は1万個に1個とその確率は極めて少ない。また、医薬品の有効性とともに安全性への認識もますます高まってきている。

医薬品の開発プロセスは図4.3-1にあるように、新規物質・薬理作用の発見、その医薬品の有効性・安全性に関する前臨床試験・臨床試験を行い、最終的に医薬品の申請・審査・承認という長いプロセスを経て市販される。さらに市販後も追跡調査がなされ、期間を区切って再審査・再評価が行われる。この医薬品の市販前の臨床試験(GCP:Good Clinical Practiceと呼ぶ)あるいは市販後の調査(PMS:Post Marketing Surveillanceと呼ぶ)は非常に重要である。なお、GCP及びPMSの実施にあたっては、各々厚生省により次のような実施基準が定められている。

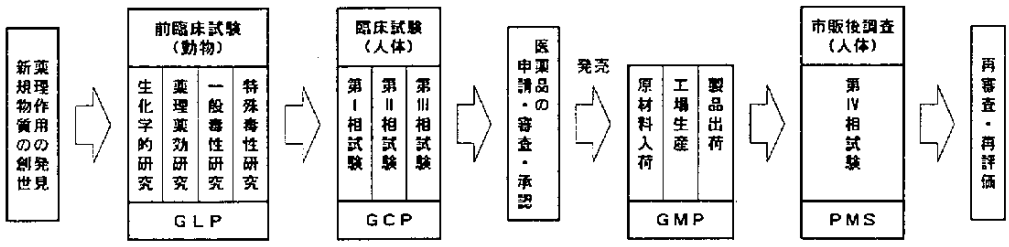


図4.3-1　医薬品開発プロセスとGCP/PMS

・GCP: Good Clinical Practice

医薬品の臨床試験の実施に関する基準(平成2年10月施行)

・GPMSP: Good Post Marketing Surveillance Practice

新医薬品等の再審査の申請のための市販後調査の実施に関する基準(平成6年4月施行)

このGCP/PMSでは、当該医薬品を実際に患者に投与した時の有効性・安全性を調査する。GCPが新医薬品の市販前の調査実施に対し、PMSが市販後の調査実施という違いはあるものの、業務内容及びシステム的な観点から類似性が高いため、ここでは以降、GCPに絞って説明することにする。

GCPにおける試験項目は一般に医薬品毎に異なっている。製薬会社では、臨床試験実施にあたってその試験項目を決め、調査票(CRF: Case Report Formあるいはケースカードと呼ばれる、ここでは以降CRFと呼ぶ)を作成し、臨床試験実施委託先の医療機関(病院)に試験を依頼する。各医療機関では、臨床試験対象の医薬品の投与結果をCRFに記入し、製薬会社に返す。この臨床試験結果データ(一般に症例データと呼ばれるので、以降、この用語を使用する)は試験項目の多さ、時間経過に応じた継続的データ収集、実施対象医療機関・患者数の多さのため、一般に膨大なデータ量になっている。製薬会社では、それらの症例データから有効性・安全性等の分析を行うため、症例データをデータベース化し、個々の症例データの信頼性チェックとともに各種統計解析処理を行う。その結果は最終的に厚生省への申請のための報告書の形にまとめられる。報告書には有効性・安全性等の観点からの製薬会社の申請内容とともに、前述の統計解析結果、および症例データ自身が含められ、この報告書は膨大な量のものになっている。

各製薬会社では、データ処理の効率化・信頼性の確保のため、症例データをコンピュータによるデータベースで管理し、後の統計解析処理、報告書作成処理へと電子的に一連の業務処理の流れで行う動きにある。このため、これらの一連の業務処理を支援するためにGCPのシステム化が盛んに行われている。ここで、GCPで使われるCRFは開発中の臨床試験対象医薬品の試験の数だけ存在し、また当業務が臨床試験担当部門というエンドユーザ部門で行われるため、CRFの簡単かつ効率的な作成、及び症例データ入力・管理の簡単さが望まれることになる。なお、近年、厚生省への報告書は症例データを含めて全て電子的な形態での申請に義務づけられる方向にあり、これからはますますこれらの臨床試験プロセスのコンピュータ処理化が必要になってくると思われる。

ここから後は、このGCP業務をコンピュータにより支援することを目的として開発した「GCP/PMS支援システム」について、具体的なIPアプリケーション事例の立場から説明する。

4．3．3 GCP/PMS支援システムとは

GCP/PMS支援システムは、図4.3-2に示すように、症例データの入力・管理を行う「CRFの設計と入力処理」、そのデータを対象として統計解析処理を行う「統計解析処理」、個々の症例データ自身の妥当性を検証する審査のための「症例一覧出力処理」の3つの処理から構成されている。当システムではこれらの処理毎に各々「CRFの設計と入力」、「統計解析」、「症例一覧出力」というサブシステムが受け持っている(その他のサブシステムもあるが、ここでは説明を割愛する)。

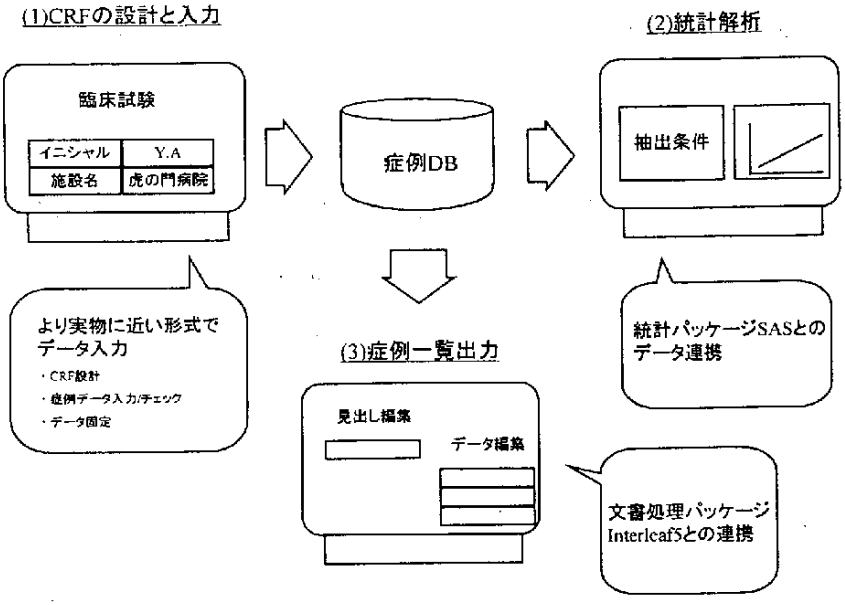


図4.3-2　システム構成

当GCP/PMS支援システムの動作環境を図4.3-3に示す。SUNワークステーションをサーバ、パソコンをクライアントとするクライアント・サーバシステムである。サーバではデータベース管理・統計解析・症例一覧出力を行っているが、各々、データベース管理システムORACLE、統計解析パッケージSAS、文書処理パッケージInterleaf5を使用している。クライアントではIPを使ったアプリケーション「GCP/PMS支援システム」が動いている。

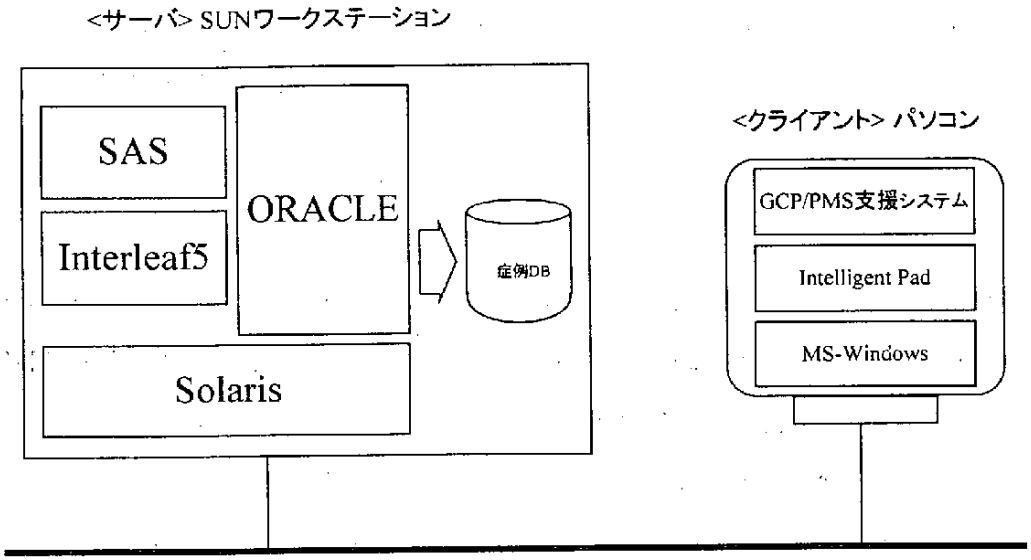


図4.3-3　動作環境

4．3．4 CRFの設計と入力

ここでは、GCP/PMS支援システムの中で、IPアプリケーション事例として特徴的な「CRFの設計と入力処理」部分を中心に述べることとする。

(1)CRF作成

まず、臨床試験対象の医薬品毎にCRFを作成する(図4.3-4aを参照)。ここで、CRFは症例データを入力・管理するためのコンピュータ画面上の画面帳票と、実際に医療機関に送付する紙帳票の2種類が存在し、両者は極力同一の形式であることが望まれる。さらに、紙帳票は画面帳票を作成することで自動的に印刷・作成することができる。

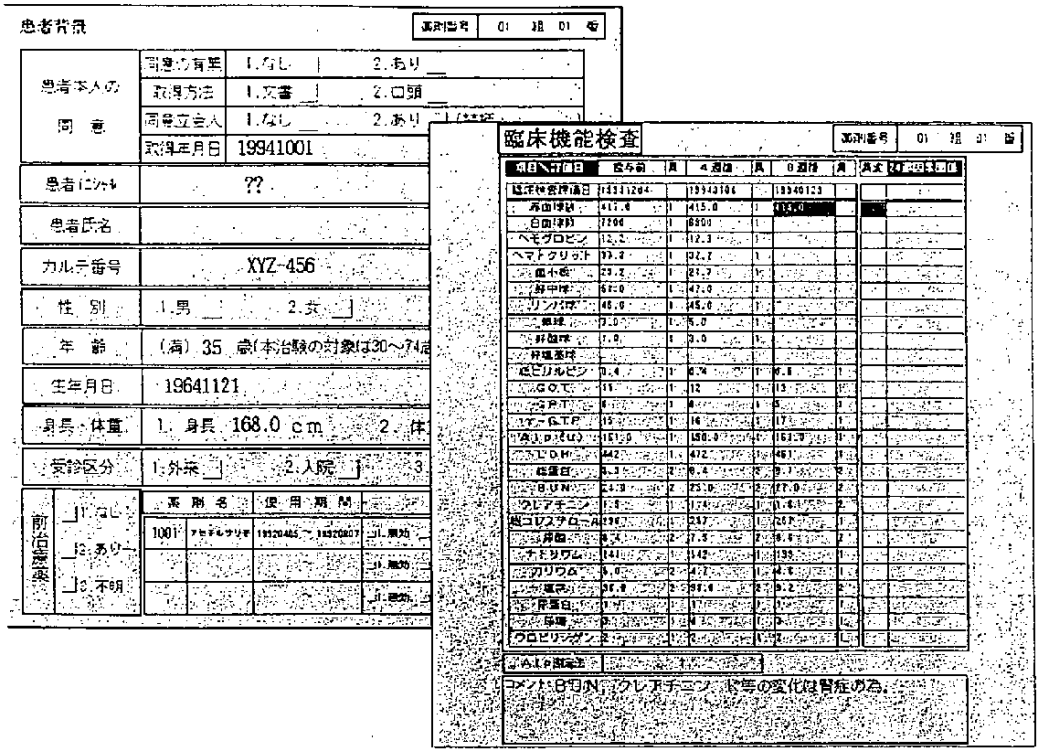


図4.3-4a　CRFの例（部分）

各医薬品毎のCRFは共通性があり、特に同一種類の医薬品では顕著である。そこで、CRF画面をいくつかのCRF画面部品で構成することにより、その部品がいろいろなCRFで共通に組み合わせて利用できるようになっている。さらに、各CRF部品は対象医薬品毎にカスタマイズが可能になっている。

CRF作成者は、各種の構成部品を組み合わせ(イメージ的には台紙に貼る)、場合によっては各部品をカスタマイズすることにより具体的なCRF画面を作成する(図4.3-4bを参照)。この部品には小さな単位の部品からそれらを組み合わせた大きい単位の部品があり、それらを最終的に組み合わせた結果がCRF画面となる。なお、同じ種類の医薬品では、CRF画面全体の類似性が高く、CRF画面全体を流用・カスタマイズ

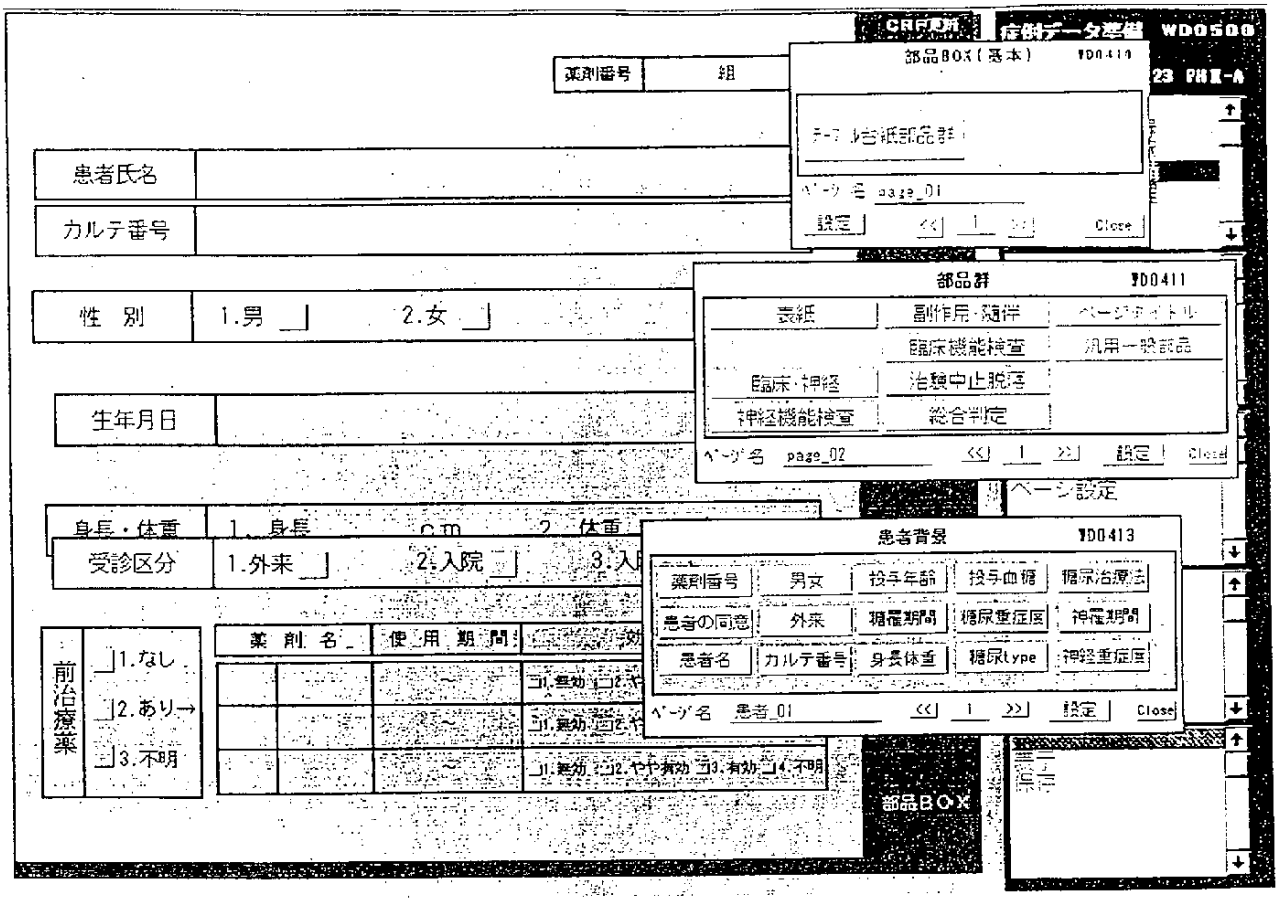


図4.3-4b　CRF部品とその利用例

することで作成される場合が多い。

このCRF画面を通して症例データを入力する場合、そのデータの範囲チェック等を定義することもでき、それらは実際にデータを入力するときに有効になる。また、内部コード化データと表示データを自動変換する機能等も部品として備えている(1/2と男/女の変換など)。

さらに、臨床試験期間中に試験項目が変更される場合があり(一般には項目の追加)、その場合は作成したCRF画面を修正することで可能になっている。

(2)　データベース作成

作成したCRFの試験項目に対応したデータベース、具体的にはRDBの表を自動生成する。一つのCRFに対して複数種類の表(一般には15－20表)が生成される。なお、医薬品にまたがったデータ処理を行うことは少ないので、医薬品毎に別々の表でデータを管理している(正確には、データベースファイルを分けて格納している)。

なお、前述の試験項目の変更によりCRF画面を修正した場合、自動的にデータベースの表の変更がなさ

れるようになっている。

(3) CRF印刷(医療機関送付用)

作成したCRF画面に対応するほぼ同一形式の紙によるCRF帳票を作成する。これは作成したCRF画面を印刷することでできるようになっている。

(4) CRF送付・回収

作成したCRF紙帳票は各臨床試験実施医療機関に送付され、記入された後、製薬会社で回収される。

(5) データ入力・チェック・データ固定

回収された記入済CRF紙帳票の症例データは、CRF画面を使って入力することによりデータベースに入力される。データ入力時にはCRF作成時に定義したデータチェックが行われる。さらに、データ入力の信頼性を確保するため、一般には二重入力が行われ、データ入力後に両データを自動比較処理することで単純入力ミスを防いでいる。

このようなCRFからの単純データ転記という意味での症例データ入力が終わった後、それらのデータは生データのまま印刷され(症例一覧出力)、製薬会社内の審査機関によって一件づつ妥当性・信憑性がチェックされ、問題のあるデータについては付帯コメントを付けた上で修正される。このような作業(審査)が終わるとこれらのデータは確定され(「データ固定」という)、以後変更はできなくなる。

その後、データ固定された症例データを対象に、医薬品の有効性・安全性の観点から各種の統計解析処理が行われ、結果が出力される(実際には、統計処理によるデータチェックが行われる場合もある)。

4．3．5 システムの特長

IPアプリケーションの観点からGCP/PMS支援システムの特長を列記する。

(1)　システム的にはGCPもPMSも同じであり(類似性あり)、共通に適用することができる。そのためシステム自身に柔軟なカスタマイズ性を持たせてある。なお、製薬会社毎/医薬品毎にCRFが異なり、カスタマイズ不要な完全なパッケージ化は困難であり、カスタマイズが必須となっている。

(2) 当システムはエンドユーザ部門(臨床試験業務実施部門)での利用を想定している(システム部門ではない)。そこでは、次の2つのシステム利用パターンがある。

・各医薬品毎のクライアント・サーバアプリケーションを作成する(言い換えるとCRF画面を作る)

・それを臨床試験業務で実際に利用する(症例データの入力等)

(3) データベース(DB)及びCRFに関する特性は以下の通りである。

・医薬品毎にCRFが存在し、それに対応するDBが存在する(医薬品が異なるとCRF及びDBが異なる)

・類似医薬品ではCRF及びDBともに共通性が高い。

・臨床試験中に項目が追加されることがある(CRF及びDBが変更される)。

・医薬品毎にDBの表は15～20種類、CRFを構成する画面(ページ)の種類は8～12種類程度である。

(4) CRF画面の設計・作成を行うと対応するデータベース(表)の自動生成を行う。そのため、利用者には具体的なデータベースは見えない(CRF画面のみ見える)。なお、このようなシステムを実現するには、一般に、まず最初にデータベースの設計・作成を行い、その後でそれに対応する画面の設計・作成を行うというアプローチも多いが、当システムで実現した方法が従来の紙ベースの運用の延長なので利用者にとって考えやすい。

(5) エンドユーザ部門ではCRF画面を作成することが重要な業務になるが、これが実はエンドユーザの意識なしに自動的に医薬品毎のクライアント・サーバアプリケーションシステムを作成していることになる。

(6) このようなシステム要件を実現するために、GCP/PMS支援システム構築向け共通部品パッケージを提供し(作るのは専門家)、顧客側(エンドユーザ部門)ではそれを使うことにより開発医薬品毎にアプリケーション画面・データベースの異なるクライアント・サーバシステムを簡単に作成するものである。

４．３．６ なぜIPか

システムを実現するために我々はアプリケーション作成ツールとしてIPを使った。ここでは、IPのどのような特長がその実現のために有効であるかを述べる。なお、完成されたアプリケーションにおいて、ツールは基本的に表に見えない(作ったアプリケーションの最終利用者にとってツールは何でも良い)。最近では、ActiveX、Javaに代表されるようなコンポーネントウェアを指向したツールが多く出てきているが、当GCP/PMS支援システムの開発をスタートさせた3～4年前はまだ部品化を指向したツールは多く出ておらず、試行錯誤的なアプローチであった。

なお、以降、「部品」という用語を基本パッド単体、あるいは基本パッドを組み合わせて作った合成パッドの両方の意味に使うことにする。

(1) IPはコンポーネントウェア(部品の提供とその利用)であり、部品を組み合わせていく、あるいはひな型アプリケーションを流用・カスタマイズすることで、最終アプリケーションを作成する。つまり、アプリケーションあるいは部品（業務部品及び汎用部品）レベルでの流用・カスタマイズが容易である。そのため、CRF画面の部品化とその利用が多様なレベルで可能であり、CRF作成者が画面を自分で作れる。カスタマイズには、レイアウトが自由、色・大きさの変更が簡単等の点が挙げられる。

(2) IPの基本パッドはデータ自身も保持できる。このことにより、CRF画面部品にデータベースのデータ項目に関する情報も含ませることができ、画面とデータ項目という関連する二つの情報をカプセル化させた部品が実現できる。

(3)基本パッドの単位を小さくできるので(IP製品提供の標準パッドで単位の小さいものも多いので)、きめ細かなCRF画面の作成が可能である。一般に、CRFは一瞥性を良くするため、一つのCRF画面(ページ)に載せるデータ項目数が多く、また種類の異なる関連データも一緒に載せるため、かなり複雑な画面となっている。

(4)基本パッドがあれば、プログラミングレスで高機能部品が作れる、あるいはカスタマイズできる。これは、普通のアプリケーションプログラマはC/C++での開発はできないというアプリケーション開発の現実に貢献する。なお、基本パッドが不足すれば、この基本パッドだけC/C++で作成すれば良く(独立性が高い)、この開発だけC/C++の得意な専門家に頼めば良い。

(5)IP製品の提供する開発環境自身をアプリケーション内で使用できる(通常は開発環境の利用と、できたアプリケーション自身の利用は別フェーズとなることが多い)。このため、CRF画面部品の貼り合わせ・カスタマイズ等がアプリケーションの中で行える。

4．3．7 IPでどう作るか

(1)アプリケーションの構成

(a) ソフトウェア構造

当GCP/PMS支援システムのソフトウェア構造を図4.3-5に示す。次のような3階層で構成される。

(i) 土台にクライアント・サーバアプリケーションを汎用的に構築するための部品がある。これは、IP製品が標準で提供するRDB連携パッド(SQL発行等の基本的なRDBアクセス機能を持つ)、それより高い操作レベルで汎用的な機能を持つRDBアクセス部品あるいは項目定義取得部品、画面(合成パッド)をメニュー形式で切り換えるメニュー制御部品(それら画面の間のデータの受渡しを行ったり、合成パッドのロードを制御する部品を含む)で構成されている。なお、基本的に、この部分の内部構造はユーザに見えない(ユーザは意識しない)。

(ii) クライアント・サーバ汎用部品の上にGCP/PMS支援システムを実現するGCP/PMS汎用部品がある。これは汎用的にCRF部品を作成したり、利用したりするための機能を提供し、サーバ接続、CRF作成、データベース作成、データ入力・チェック・固定等を行う部品群で構成される。これらの画面(合成パッド)は処理の遷移につれて、メニュー制御により動的に切り換えられる。

(iii) GCP/PMS汎用部品の上に、対象とする医薬品向けに作成されたCRF画面(合成パッド)を貼る。作成フェーズで作成されたCRF画面は、実際にデータ入力等の利用フェーズではメニュー制御により動的に接続が張り換えられる。

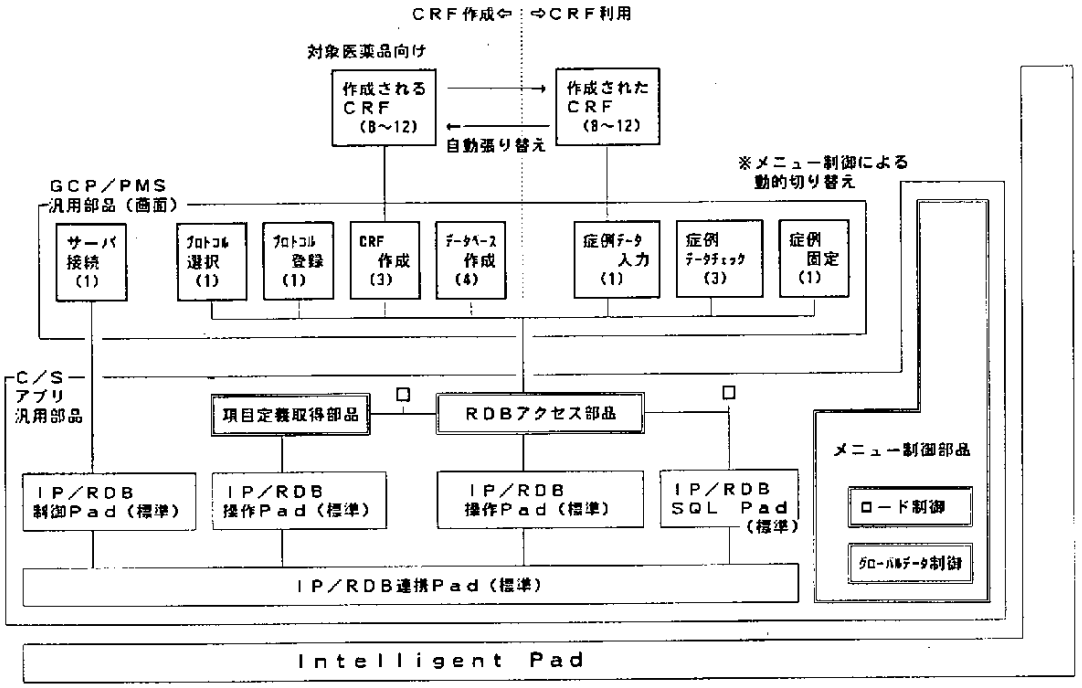


図4.3-5　ソフトウェア構造

(b) CRF構造

前述の対象医薬品向けのCRF画面の具体的な構造を、患者背景画面を例にとって図4.3-6に示す。土台にはCRF台紙部品あり、その上に患者の基本情報を扱う基本情報部品を貼る。この基本情報部品にはさらに、情報の種類毎にいくつかの部品が含まれており、それらの部品の間は階層関係で構成される。患者の過去の治療薬情報を扱う治療薬情報部品についても同様である。ここで、一般テーブル台紙部品、繰り返しテーブル台紙部品は、各々フォーム形式、テーブル形式でデータを表示・操作するための汎用部品である。

このように、CRF部品は機能の大きさの異なる部品で構成されており、新しく医薬品のCRF画面を作成する場合は、なるべく大きなレベルの部品を使うことが生産性の向上につながる。

(2) CRF共通システムをどう作るか

データベース及びCRFの基本構造は固定かつ共通にしておいて、各CRF毎に特化したアプリケーションの作成・カスタマイズが出来るような仕組みが必要とされる。このため、GCP/PMS支援システム用共通部品(機能部品及び画面部品)をどう作って提供するか、が最も重要なポイントになる。ここでは、GCP/PMS支援システムを開発した時の経験を踏まえて留意事項を述べるが、一般的なIPアプリケーションの開発時における留意事項とも捉えられる。

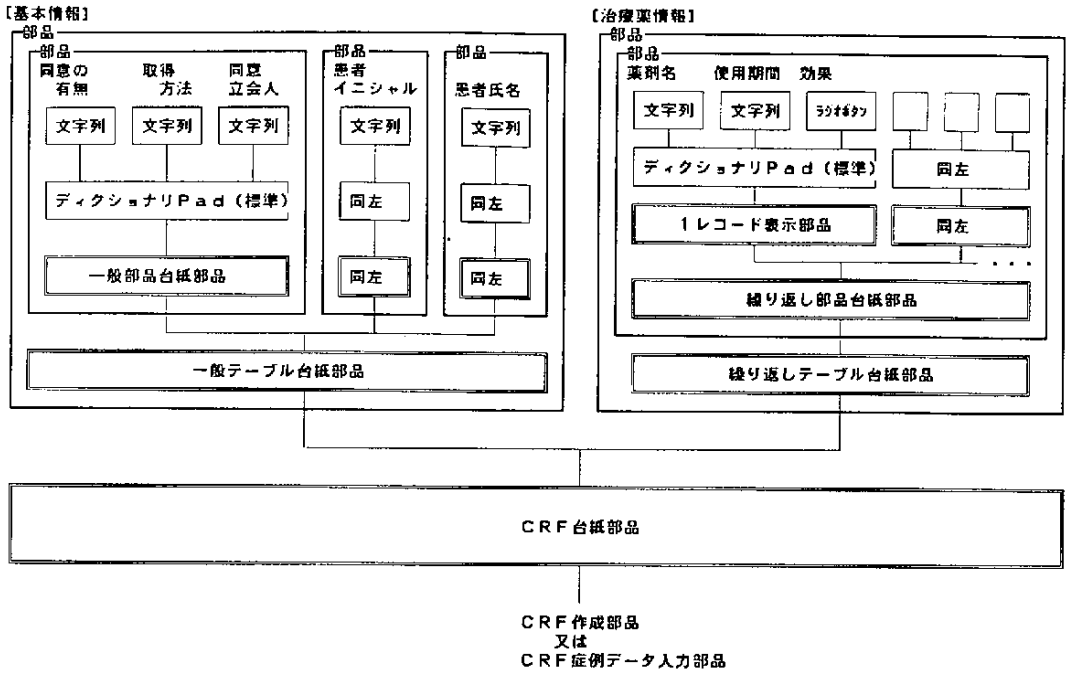


図4.3-6　CRF構造（患者背景より抜粋）

(a) 共通部品の整備

極力高い機能レベルでの業務依存部品あるいは業務に依存しない汎用部品を作成することが重要である。また、部品化にあたってはその部品と外とのインタフェースとして、パッドの親子関係の間でのスロット結合のみを使用し、パッド間のスロット結線は使用しない（結線は部品化を妨げるため)。

これにより、生産性の向上とともに、全体統一の取れた共通仕様のアプリケーション作成、メンテナンス性の向上、障害調査・修正の効率化が図れることにより、結果としてアプリケーション品質の向上といった効果も実現できる。

(b) 画面部品の作成

画面設計時においては入出力・表示を司る画面部品を抽出・作成する観点から、画面仕様の標準化（共通要素のまとめあげ）が必要である。なお、CRF画面の部品はここからデータベースを作成することになるため、データベースのデータ項目に関する情報も含んだ部品となることが、通常の機能部品と違って重要な意味を持つ。

(c) 不足部品への対応

アプリケーション開発時、特に新しいタイプの場合にはどうしても不足部品(基本パッド、あるいは合成パッド)が出てくる。この時、それを新たに開発するのではなく、いかに探してくるかが重要であり、そのためには、部品情報の入手のための社外へのアンテナを張っておくことやプロジェクト間共有のため

の情報提供も重要である。

(d) 基本パッドの新規開発

不足パッドが外部から調達できない場合、新規に開発する必要が出てくるが、その場合基本パッドの単位を極力小さくする。これは、基本パッドがあれば、それを合成した部品の作成は容易であり、部品の試行錯誤的な開発が柔軟にできるからである(C/C++で大きな機能単位の基本パッドを作ると変更が比較的困難である)。なお、ここで、一般にそのような合成パッドによる部品の機能性は性能とのトレードオフにあり、基本パッドの機能単位の大小が問題になる。性能が問題にならない範囲では極力小さな単位での開発(とそれを組み合わせた合成)を、性能が問題になる場合のみ、あるいは部品の仕様が確定した場合のみ、大きな単位とすることが望ましい。

(e) 自動化ツールの整備・利用

部品の設計・開発にあたっては試行錯誤的なアプローチになることも多い。その場合、設計ドキュメントをきっちり作ってから部品作成するより、部品開発後に設計・保守ドキュメントを自動的に作成するようなツールの整備・利用が望ましい。また、アプリケーション開発時には部品の変更に伴う部品の差替え等が発生するケースも多い。そのため、部品の追加・削除・差し替えが極力自動的にできるような仕掛けの組み込み、あるいはそのようなツールの整備・利用が望ましい。

(f) IP開発環境の利用

CRF画面部品の選択・貼り合わせを行うことにより、CRF画面、つまりアプリケーションを作成するが、そのために基本的なIPのパッド結合機能を使用する。また、カスタマイズ等でもIPの基本操作機能を使用する。つまり、IPの開発環境自身をアプリケーションの一つの機能として使う必要がある。

(3) CRF固有アプリケーションをどう作るか

ユーザ(CRF作成者)は、各医薬品に固有なCRF画面、つまりアプリケーションを作成することになるが、ユーザの意識はあくまでもCRF画面を作ることであり、その結果として自動的にデータベースも含めたアプリケーションが作成される。

このCRF画面の作成は、基本的には画面部品の組み合わせで行うが、効率のよい作成のためには極力高い機能レベル（範囲）での部品の利用・流用が重要である。また、類似医薬品の場合は、CRF画面全体を流用することができる。なお、部品の組み合わせ方を示すサンプルとして、かつアプリ全体を流用するためのひな型として、当システムではひな型アプリケーション(CRF画面)も合わせて提供している。

このように、CRF画面を作成するには、生産性の観点から、どのレベルの部品を利用・流用できるかの判断が重要であるが、現時点ではCRF作成者の経験に基づいた方法に依っていることが実際である。

4．3．8 おわりに

当システムの開発に携わりはじめたのは今から3～4年前であり、その初期段階では製品初版(富士通IP/V1.0)を使用したが、機能不足(部品不足)あるいは開発環境の不十分さ等で苦労した面が多かった。しかし、最近IP/V4.1が提供され、ここでは初版に比べて機能及び性能の両面において格段の改善・強化がなされていることから、今から初めてシステムを開発される利用者に取っては大きな相違が出てくると考える。

また、IPの新しい機能に次のようなものがあり、新しいタイプのアプリケーションの開発と利用に向けて期待できる。

(1) 他ツールとの連携機能が強化されていて、OLE連携、ActiveXコントロールの利用、合成パッドのActiveXコントロールへの変換ができる。

(2) インターネット環境での利用ができるようになり、Netscape NavigatorでのIP-plug-in機能及びMS-Internet Explorerでの上記ActiveX利用により、インターネットブラウザで基本パッド・合成パッドだけでなくＩＰアプリケーション自身も動作する。

最後に、前述したように、IP以外のツール利用の局面でもActiveX、Java等での部品化とその利用によるアプリケーション開発スタイルが進展しつつあり、それらの技術が当システム自体の進歩に貢献するとともに、逆に当システムの開発を通して培った技術が他ツール利用の局面でも共通に活かせるものと考えている。

【参考文献】

1) IntelligentPad V4.1 ユーザーズマニュアル, (株)富士通

2) GCPハンドブック, 薬業時報社

3) GPMSPのすすめ方とそのノウハウ, 薬業時報社

4) GCP/PMS支援システム構築部品(症例管理) 概説書, (株)富士通東京システムズ

4．4　VisualAgeの事例

VisualAgeは、オブジェクト指向テクノロジーに基づいたネットワークコンピューティング型のアプリケーションを構築する。優れたビジュアルプログラミング環境を提供し、豊富なパーツを取揃えていることから、RAD(Rapid ApplicationDevelopment)のツールとして使用されることもあるし、本格的なオブジェクト指向開発で適用される場合もある。ここでは主にVisualAge for Smalltalkの適用事例を様々なパターンごとに紹介する。

4．4．1　VisualAge for Smalltalk＆ネットワークコンピューティング

VisualAge for Smalltalkは再利用可能な豊富なパーツを提供し、多様なネットワークコンピューティングシステムを構築する際の生産性を向上させる。事例の紹介に入る前にネットワークコンピューティングを以下のタイプに分類して、VisualAge for Smalltalkがどのようなアプリケーションに適用されるかを説明する（図4.4-1参照）。

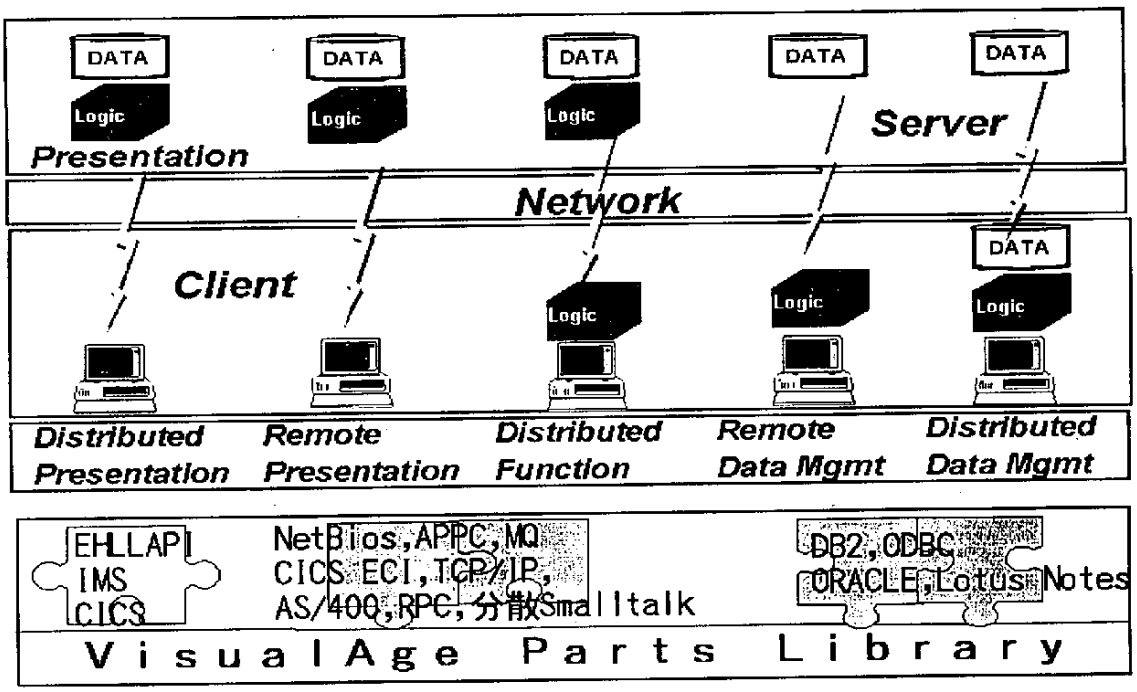


図4.4-1　VisualAge＆ネットワークコンピューティング

1．Distributed Presentation

プレゼンテーション部分をクライアントに持ってくる分散システム。ホストの端末エミュレーション画面をクライアント側で読み込み加工してGUIとして表示する。既存のホストシステムの変更をせずにユー

ザの操作環境の向上（GUI可）、機能追加（PCデータの蓄積や加工など）が可能となる。

2．Remote Presentation

プレゼンテーションはクライアント、処理やデータはサーバ。クライアントはデータ表示や入力チェックのみでクライアントの負荷を軽くする。

3．Distributed Function

プレゼンテーションはクライアント、処理はクライアントとサーバに分散する。2と3では図4.4-1にあるように様々なプロトコル（NetBios、TCP/IPなど）やトランザクションシステムをサポートする。また、分散Smalltalkにより、Smalltalkのオブジェクトをリモートでやりとりをする分散オブジェクトシステムを実装することが可能である。

4．Distributed Data Management

データを分散する。この場合、データ分散の仕組みはデータベース管理システムで実装し、アプリケーションプログラムは「そのデータはどこにあるのか」を意識しないような柔軟な作りにするのが一般的である。

4．4．2　適用事例1・・・A会社　コールセンター受付システム

A会社で既存のメインフレーム上で稼働しているアプリケーションをVisualAgeで分散クライアント／サーバシステムに再構築した事例を紹介する。構成を図4.4-2に示す。

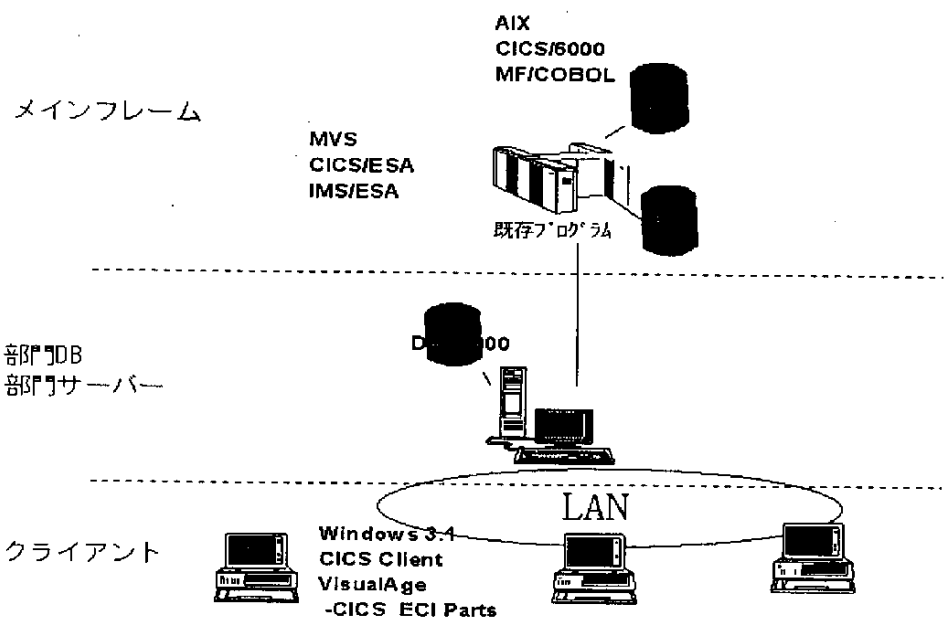


図4.4-2　Distributed Function型

［概要］同社はPCハード、ソフトに関する問い合わせをコールセンターの担当者が受付をし、履歴を検索して質問に回答、メーカーへの処理依頼、保守部門へ引き取りや修理の依頼をする業務がある。以前はメインフレームのシステムが稼働していたが、エンドユーザのGUIによる操作性の向上、メインフレームからのダウンサイジングを狙いとした。

［特徴］Distributed Function型で、クライアント、サーバ、メインフレームの３階層リアルタイム処理を実現。CICSというトランザクションシステムを採用し、VisualAgeが持つこのトランザクションシステムへのリンクのパーツを利用することによりプログラミングする部分が大幅に削減できた。また、VisualAgeのチーム開発機能によるバージョン管理やバックアップ管理が、複数人数による開発工程における運用面で生産を向上させた。パーツの利用とPC環境における開発の管理の機能をうまく使用することにより現行業務のダウンサイジングに成功したといえよう。また、エンドユーザからは内部の複雑なシステム体系とは別に、統一された簡単なGUI操作により業務の生産性が向上した（少人数で約５ケ月で開発）。

４．４．３　適用事例２・・・B会社　システムアドミニストレーションシステム

［概要］システムオペレーター（システム運用担当者）がシステムの運用・保守する業務。

［特徴］Distributed Function型で、クライアント、メインフレームの２階層システム。4.4.2項と同じでCICSトランザクションシステムをベースとし、VisualAgeのパーツを利用した。開発担当者は当初Smalltalk言語のスキルはなかったが、言語の簡便さとパーツの利用により短期間でシステム構築が実現できた（２人で３ケ月、４０クラス数を開発）。

４．４．４　適用事例３・・・C会社　ディレクトリーサービスシステム

［概要］情報系DBのデータの登録、更新、削除を行うシステムのプロトタイプ。各部門のデータ管理者が使用する。構成を図4.4-3に示す。

［特徴］Remote Data Management型でサーバ上のDBはSybaseを使用。アプリケーション開発の効率を向上するためにツールの選定を行い、クライアント／サーバ型システムの開発に適用可能であること、オブジェクト指向テクノロジーの採用、オープンな環境での動作、ビジュアルプログラミングが可能であることが条件された。オブジェクト指向技術を理解するための方法としてSmalltalkが有効、パーツ（他プログラムとの連携、DBアクセスなど）が豊富であることからVisualAgeが採用された。当システムではODBCのインタフェースを使用し、VisualAgeのODBCパーツを使用した。開発者の評価として、プログラミング生産性が高い、習得が容易、チームでの開発が容易であることがあがった。

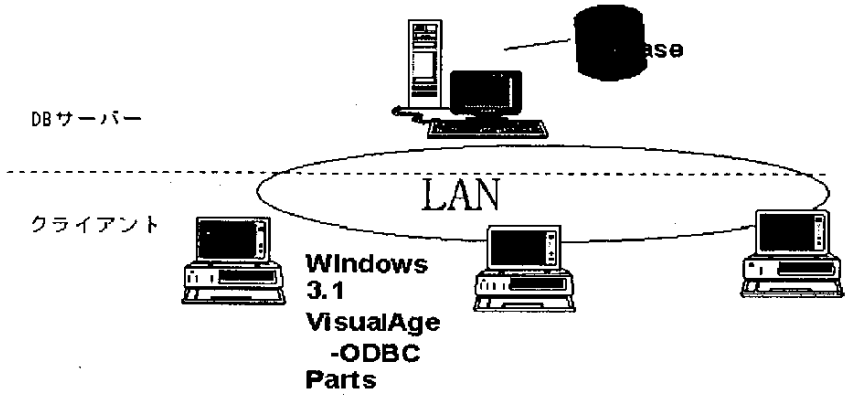


図4.4-3　Remote Data Management型

4．4．5 VisualAge Smalltalkとオブジェクト指向

ここまで、VisualAgeがもつコミュニケーションやＤＢのパーツを使用して生産性を上げた事例を紹介してきたが以下では上流からオブジェクト指向の開発メソドロジーを使用した例を紹介する。Smaltalkはピュアなオブジェクト指向言語かつ開発環境であるのでオブジェクト指向で分析／設計されたモデルを実装までスムースにつなげることができる。モデルと実際のコードが近いことにより、より部品化をしやすい面を持っている。

4．4．6　適用事例4・・・D会社　商品分析支援システム

［概要］D会社の扱う商品の分析支援を行うシステム。機能はデータ入力、分析・計算、グラフ表示、ファイル入出力、レポート作成支援がある。

［特徴］現行のLotus1-2-3とC言語のプログラムによるシステムに使いがってと拡張性に難があり再構築をした。システムを開発するにあたって、オブジェクト指向の適用（分析、設計、実装）、今後の分析対象の種類の増加への速やかな対応（拡張性）が目的とされた。開発方法論としてOMTとBoochを併用した。当商品は似た多種類の物が多い特徴を持っていたので、オブジェクト指向で設計したことにより、変更、拡張などの保守性が上がった。なお、グラフはパーツを利用した。

4．4．7　適用事例5・・・E会社

［概要］　同社はソフトウェアの開発を業務とし、ソフトウェアのモジュールの管理システムを構築した。企画からインベントリの制御、出荷までいわばソフトウェアのWarehouseシステムである。

［特徴］オブジェクト指向の手法を取り入れたことにより従来と比較して生産性が3～4倍向上した。プロ

グラマーのスキルはRPG、C++、PowerBuilderなど様々であったので教育から始めた。また、品質に関してファンクションポイントごとのバグを調査したところ、従来よりも3分の2へと減少した。システム環境は開発PCはOS/2、実行PCはWindows、さらに開発サーバをAIX(Unix)としマルチプラットフォーム対応により柔軟な構成が組めた。

4．4．8　適用事例6・・・地図パーツ

［概要］地図情報を操作するパーツをVisualAge for Smalltalk上で開発。パーツになっているので適用するアプリケーションによってカストマイズが容易にでき、利用できる。

［特徴］地図情報を扱うアプリケーションの構築を考えた場合、以下の点からオブジェクト指向の適用が向いており、Smalltalkが採用された。

－地理上の建物等を地理オブジェクトとしてモデル化すると、概念的にシンプルであり、アプリケーション特有の拡張が容易である。地図／図面に対する基本操作部分は一つのオブジェクトを共用して提供できるため、業務が異なっても、また図面が異なっても、統一した操作インタフェースを実現できる。

－地理対話処理をフレームワークとして設計することにより、標準操作部品（スクロール、ズームなど）の提供と、さらに部品の追加、変更が容易となる。当パーツはそれ自体が階層化されたオブジェクト・クラスとして定義されているのでアプリケーション開発をする時に、これらのクラスの使用および継承により生産性が向上するといえる。また、設備などのオブジェクトは必要になった時点でダイナミックに生成するのでシステムとしては不必要な資源を省いて効率よく稼働する。

以上、VisualAgeの事例を紹介してきたが、パーツの利用とその拡張のしやすさ、およびビジュアルプログラミング環境によるコーディングの少なさが開発の生産性を向上させることが共通して言える。今後はインターネット、イントラネットの普及によりWebサーバとエンタープライズサーバによるネットワークコンピューティングモデルが増えると思われるが、Web対応やJavaのパーツの充実が望まれ、ますますコンポーネント（パーツ）化が促進されることになるだろう。

平成９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

08－R003