PNC TN9440 96-025

分散オブジェクト技術を用いた複数解析コードの協調計算に関する研究

Coad Cooperation Analysis with Distributed Object Computing

（1）PARTS-FLOWとSuper-COPDの協調計算の原理確認

１９９６年７月

動力炉・核燃料開発事業団
大洗工学センター

本資料についての問合せは下記に願います。
〒311－13　茨城県東茨城郡大洗町成田町４００２
動力炉・核燃料開発事業団
大洗工学センター　システム開発推進部・技術管理室

Ⓐ

分散オブジェクト技術を用いた複数解析コードの協調計算に関する研究

Code Cooperation Analysis with Distributed Object Computing

(1) PARTS-FLOW と Super-COPD の協調計算の原理確認

細貝 広視*、笠原 直人**

要　旨

高速炉機器構造の支配荷重である熱応力を最小とする優れた構造形状および運転法を探索するため、熱・流体・構造の複合現象である熱過渡現象を、複数コードの協調により、統合解析する手法を開発している。

これまでに、コードごとに異なるインターフェースに依存せずに柔軟な組み合わせを可能とする解析システムとして、オブジェクト指向技術を応用した(1)熱・流体高速計算、(2)構造温度・応力高速計算、(3)ひずみ・強度高速計算 の協調解析コード、過渡熱応力リアルタイムシミュレータ PARTS (=Program for Arbitrary Real Time Simulation) のプロトタイプ設計と試作を行なった。

さらに本研究では、 PARTS コードの計算規模と速度の向上を目的として、計算処理の分散化と並列化を試みた。分散化と並列化を実現するための中核となる技術として、ネットワーク上に分散配置された計算機資源をオブジェクト指向技術を利用することにより物理的情報とは切り離して統合することができる分散オブジェクト技術を利用した。

検証例題として、 PC 上で稼動する PARTS の熱・流体高速計算サブシステムである PARTS-FLOW と、EWS 上で稼動する Super-COPD のＳＧモジュールのネットワークを介した協調計算を取り上げ、原理の確認を行った。

*　常陽産業株式会社

** 大洗工学センター 基盤技術開発部 構造・材料技術開発室

目　次

表　リ　ス　ト

図　リ　ス　ト

１．はじめに

熱過渡条件計算プログラム「PARTS-FLOW」と、水蒸気系計算機能を有する「SuperCopd」協調計算の可能性を確認するための原理確認システムを作成した。

本システムは、“リモート分散オブジェクト技術”に基づいて構築されている。この技術は「プログラムあるいは関数単位でネットワーク接続されたコンピュータに処理を割り振り、負荷を分散して全体の処理効率を向上させる。」というものである。
パソコン・ワークステーション・大型計算機という枠組みを超えて一元的に連携し、しかもネットワークにより遠隔地の計算機とも結び付く事が出来る。これによってかなり負荷の大きいプログラムでも効率的に処理することが可能になった。ユーザーインターフェースは、パソコンを使い操作性の良いものにし、実際の処理はワークステーションあるいは大型計算機に割り振る等の優れたシステムをデザインすることが出来るようになった。

分散オブジェクト技術はオブジェクト指向技術を中心にして、分散化と多様化に対応する技術成果を取り入れた複合技術である。したがってその技術構造も複数の構成要素から成り立っている。構成要素を以下に示す。

※・分散オブジェクトモデル（ＣＯＭ）
・オブジェクト・インターフェース
・オブジェクト・インプリメンテーション
・リクエスト転送

※ 分散オブジェクトモデル（ＣＯＭ）
分散オブジェクト技術はコンピュータシステムの基本設計に関わる技術であり、そのためにまずシステム構造の基盤となる概念的なモデルを提供する。分散オブジェクト技術では分散マルチベンダ環境に対応するためのモデルをクライアント／サーバー型のシステム形態に求めた。分散オブジェクト技術におけるクライアント／サーバーモデルでは、サービスを要求するユーザーやアプリケーションをクライアントと呼びサービスを提供するオブジェクトをサーバーと呼ぶ。サーバーとなるオブジェクトはインターフェースとインプリメンテーションから構成される。クライアントからオブジェクトへの要求はすべてインターフェースに基ずいたリクエスト転送によって行われる。

分散オブジェクトを実現する具体的アプリケーションとしてＤＥＣ社の「ＯｂｊｅｃｔＢｒｏｋｅｒ」という製品を使用している。
これはＤＥＣが開発したＣＯＲＢＡ準拠のミドルウェア製品である。

２．原理確認システム

２．１ 概要

本システムの目的は以下の３つである。

1. ２つの異なるコンピュータ間でクライアント・プログラムとサーバ・プログラムが正常に連携して動作することを確認する。
2. プログラム制御方式である(1)同期実行処理、(2)非同期実行処理の２つが本システムのソフトウェア環境で実現出来るかどうかを確認する。
3. 様々なタイプのパラメータがクライアント・サーバ間で正確に受け渡るかどうかも確認する。

異なる計算機、OS 間で「ObjectBroker」プログラムが正しく動作することを確認し、基本的な動き、I/O パラメータの取り扱い、制限事項等を踏まえて大規模システムへの適用へとつなげていく。その第一段階がこの原理確認システムである。

２．２ 動作環境

本システムは、ネットワークを介してクライアントマシンとサーバーマシンで構成されている。Table1　に具体的な動作環境を示す。

２．２．１ クライアント環境

機種： DOS/V 機　(例：GATEWAY2000 P5-133)
OS：WindowsNT 3.51
I/F ソフト：　(米) Digitalk 社 VisualSmalltalkEnterprise3.0.1 (VSEW301) ＋
　　PARTSWorkBench　(PWB)
ORB 製品： DEC ObjectBroker V2.5 for WindowsNT

２．２．２ サーバ環境

機種：SPARC Station
OS：SunOS V4.1.3
サーバプログラム本体：S-COPD (Fortran プログラム)
ORB 製品： DEC ObjectBroker V2.5 for SunOS

２．２．３ ネットワーク環境

トランスポート・プロトコル：ＴＣＰ／ＩＰ通信

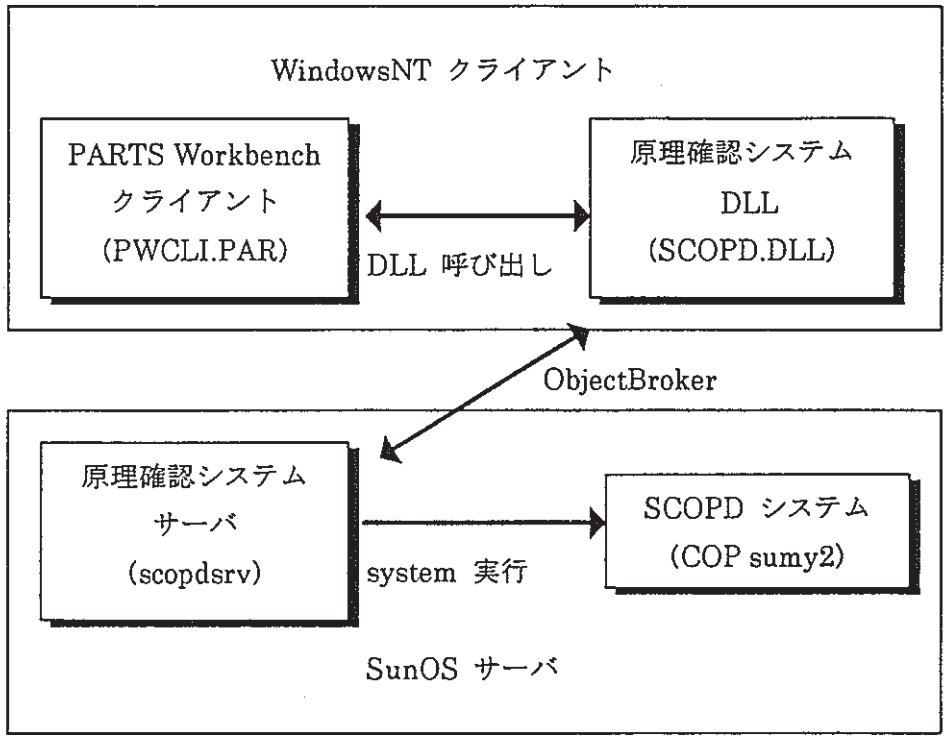
	機種	OS	トランスポートプロトコ	ORB環境	開発言語	IPアドレス	備考
サーバー環境	SunMicrosystems SparcStation10	SunOS Release4.1.3	TCP/IP	DEC ObjectBrokerV2.5 for SunOS	ANSI Cコンパイラ	133.188.87.23	login名は rootで行う
クライアント環境	GATEWAY2000 P5-133	Microsoft WindowsNT3.51	TCP/IP IPX/SPX	DEC ObjectBroker V2.5 for WindowsNT	Microsoft VC++ V2.0	133.188.87.47	

Table1　原理確認システム動作環境

※ 分散オブジェクト利用環境を実現している ObjectBroker は、TCP/IP の他さまざまな通信プロトコルをサポートしている。そのため大抵のネットワーク環境であればこの技術を実現することが出来る。Table2　に対応するネットワーク環境の一覧を示す。

2．2．4 原理確認システムの構成

原理確認システムの構成は以下の通りである。

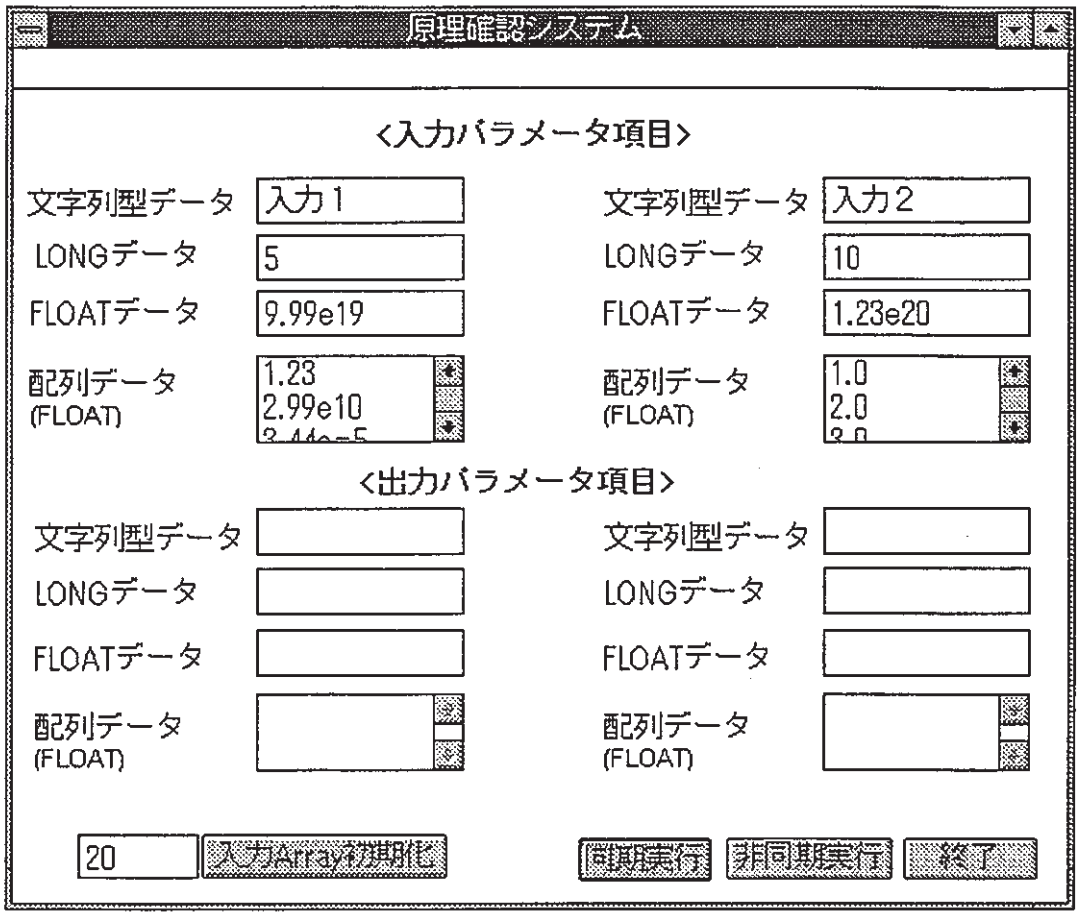


2．2．4．1 PARTS Workbench クライアント

原理確認システムのユーザインターフェースを提供するクライアントプログラムである。PARTS Workbench によって作成されている。PARTS Workbench クライアントは DLL 部品を用いて原理確認システム DLL を呼び出すことによって、サーバへアクセスする。以下に PARTS Workbench クライアントの画面イメージを示す。

ハードウェア	オペレーティングシステム	ネットワークソフトウェア
VAX	OpenVMS VAX V5.2以降	TCP/IP Services V3.0、DECnet V5.4-5.5
AlphaAXP	OpenVMS AXP V1.5以降	TCP/IP Services V3.0
AlphaAXP	OSF/1 V2.0	TCP/IP (OSにﾊﾞﾝﾄﾞﾙされるもの)
AlphaAXP	Microsoft WindowsNTV3.1以降	TCP/IP (OSにﾊﾞﾝﾄﾞﾙされるもの)、DECnet
ULTRIX RISC	ULTRIX V4.3以降	TCP/IP (OSにﾊﾞﾝﾄﾞﾙされるもの)、DECnet V4.2
HP9000-7xx,8xx	HP-UX Version8.07	TCP/IP (OSにﾊﾞﾝﾄﾞﾙされるもの)
IBM AIX	AIX V3.2	TCP/IP (OSにﾊﾞﾝﾄﾞﾙされるもの)
SunSparc	SunOS V4.1.1	TCP/IP (OSにﾊﾞﾝﾄﾞﾙされるもの)、DECnet
IBM-PC互換機	Microsoft WindowsNT V3.1以降	TCP/IP (OSにﾊﾞﾝﾄﾞﾙされるもの)、DECnet
IBM-PC互換機	Microsoft Windows V3.1以降	TCP/IP (WinSockV1.1対応ｿﾌﾄｳｪｱ)、DECnet
Macintosh	System7.0,7.1以降	MacTCPV1.1.1-V2.04,TSS-NetV2.4.1,DECnetなど

Table2 オブジェクトブローカが動作する環境



2．2．4．2 原理確認システム DLL

PARTS Workbench クライアントから呼び出される DLL である。原理確認システムサーバと ObjectBroker を使用して情報のやり取りを行う ObjectBroker クライアントである。原理確認システム DLL で提供される関数は以下の７つの関数である。

- SyncCopExec - SCOPD システムを同期実行する。
- AsyncCopExec - SCOPD システムを非同期実行する。
- PollMessage - 非同期実行の結果をポーリングする。
- CopGetResult - 非同期実行の結果を取得する。
- FreeOutSeq - 原理確認システムの SyncCopExec AsyncCopExec で確保した配列データを解放する。
- FreeData - 原理確認システムで確保した領域を解放する。
- InitDouble - Double 型の配列データを初期化（DBL_MAX）する。

2．2．4．3 原理確認システムサーバ

SCOPD システムの起動を行う ObjectBroker のサーバプログラムである。原理確認システムサーバは system コールによってSCOPD システムを実行し、結果を原理確認システムDLLに返す。

2．2．4．4 SCOPD システム

SCOPD システム本体である。原理確認システムサーバから起動される。

3．原理確認システム環境設定

原理確認システムのインストール方法と環境設定方法を示し、原理確認システムが使用可能な環境を構築する方法を示す。

3．1 サーバの環境設定

3．1．1 前提

サーバマシンに以下のソフトウェアがインストールされている事、 TCP/IP によってクライアント・サーバ間の通信が設定済みであることが前提となる。

- SunOS 4.1.3
- SCOPD システム
- DEC ObjectBroker V2.5 for SunOS
- Ansi C コンパイラ

またｌｏｇｉｎする時は、ＯｂｊｅｃｔＢｒｏｋｅｒの各種データファイルを編集するため「ｒｏｏｔ」でログインしなければならない 。
パスワードは、計算機係に問い合わせること。

- login は root で行う

クライアントとサーバーはそれぞれ固有の計算機名称で認識し合うため各自の計算機の「ｈｏｓｔｓ」ファイルに互いのＩＰアドレスと計算機名称を事前に登録しておかなければならない。
ｈｏｓｔｓファイルに関する詳細は、クライアント側を Table3 にサーバー側を Table4 に示す。

Table3　サーバー側 HOSTSファイルの内容

#	
# Sun Host Database	
#	
# If NIS is running, this file is only consulted when booting	
#	
127.0.0.1	localhost
#	
133.188.87.23	osessn01　　loghost
133.188.83.32	otmssn01
133.188.0.1	otmsm780
133.188.80.1	otmsms01
133.188.81.2	otmsfc02
133.188.83.29	otmsne01
133.188.83.2	otmsar01
133.188.86.9	oseshp01
133.188.86.10	oseshp02
133.188.87.24	osessn02
133.188.90.32	osessn03
133.188.95.15	osessn04
133.188.87.27	osessn05
133.188.87.15	osespc05
133.188.87.20	osespc12
133.188.87.2	osesib04
133.188.87.3	osesib05
133.188.87.4	osesib06
133.188.95.2	osesds01
133.188.87.22	osessg01
133.188.90.37	oscssn02
133.188.99.2	oirsar01
133.188.87.8	osesib10
－＞ 133.188.87.47	osesib37

クライアントとなるコンピュータのＩＰアドレスを事前に登録しておかなければならない。

機種：SunMicrosystems SparcStation10
OS：SunOS V4.1.3
格納位置：／etc／hosts

Table4　クライアント側 HOSTSファイルの内容

# Copyright (c) 1993-1995 Microsoft Corp.	
#	
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows NT.	
#	
# This file contains the mappings of IP addresses to host names. Each	
# entry should be localhost	
# be placed in the first column followed by the corresponding host name.	
# The IP address and the host name should be separated by at least one	
# space.	
#	
# Additionally, comments (such as these) may be inserted on individual	
# lines or following the machine name denoted by a '#' symbol.	
#	
# For example:	
#	
#　　102.54.94.97　　rhino.acme.com　　# source server	
#　　38.25.63.10　　x.acme.com　　# x client host	
133.188.87.47	osesib37　　localhost
133.188.83.32	otmssn01
133.188.0.1	otmsm780
133.188.80.1	otmsms01
133.188.81.2	otmsfc02
133.188.83.29	otmsne01
133.188.83.2	otmsar01
133.188.86.9	oseshp01
133.188.86.10	oseshp02
-> 133.188.87.23	osessn01
133.188.87.24	osessn02
133.188.90.32	osessn03
133.188.95.15	osessn04
133.188.87.27	osessn05
133.188.87.15	osespc05
133.188.87.20	osespc12
133.188.87.2	osesib04
133.188.87.3	osesib05
133.188.87.4	osesib06
133.188.95.2	osesds01
133.188.87.22	osessg01
133.188.90.37	oscssn02
133.188.99.2	oirsar01
133.188.87.8	osesib10

サーバーとなるコンピュータのＩＰアドレスを事前に
登録しておかなければならない。

機種：GATEWAY2000 P5-133
OS：WindowsNT3.51
格納位置：C:¥WINNT35¥SYSTEM32¥Drivers¥etc¥hosts

3．1．2 サーバ・キットのインストール

サーバ・キットは tar 形式の QIC テープで提供される。以下の手順でサーバ・キットをインストールする。(手順は /usr/usrs/user1/scopd にキットをインストールる。)

1. インストール先のディレクトリを作成する。
   %mkdir -p /home2/OB/scopd
   %cd /home2/OB/scopd
2. QIC テープからキットをコピーする。(例の QIC のテープデバイスは /dev/rst12 としている。)
   %tar -xvf /dev/rst12

以下に QIC テープに格納されているファイルの一覧を Table5 に示す。

Table5 ＱＩＣに格納されたファイルの一覧

```
osessn05# tar -tvf /dev/rst12
drwxr-xr-x622/15          0 Apr   2 16:23 1996 ./
-rw-r--r--   0/1      25292 Apr   2 16:19 1996 method.o
-rw-r--r--622/15       3270 Apr   1 18:09 1996 chdata.c
-rw-r--r--622/15       5433 Apr   1 18:09 1996 cli_main.c
-rw-r--r--622/15      21022 Apr   2 11:28 1996 disp.c
-rw-r--r--622/15       2393 Apr   1 18:09 1996 method.c
-rw-r--r--622/15        941 Apr   1 18:39 1996 method_tmp.c
-rw-r--r--622/15       4604 Apr   1 18:24 1996 srv_main.c
-rw-r--r--   0/1      34984 Apr   2 16:19 1996 disp.o
-rw-r--r--622/15       6877 Apr   2 11:28 1996 type.c
-rw-r--r--622/15       2061 Apr   2 11:28 1996 scopd.imh
-rw-r--r--622/15        835 Apr   1 18:14 1996 extern.h
-rw-r--r--622/15       1987 Apr   2 11:28 1996 scopd.h
-rw-r--r--622/15       1222 Apr   1 18:09 1996 scopd.idl
-rw-r--r--622/15        862 Apr   1 18:09 1996 scopd.iml
-rw-r--r--622/15        711 Apr   1 18:09 1996 scopd.mml
-rw-r--r--   0/1        203 Apr   2 16:23 1996 cop_obj.ref
-rw-r--r--   0/1      28432 Apr   2 16:19 1996 srv_main.o
-rw-r--r--622/15       1922 Apr   2 11:28 1996 scopd.tch
-rw-r--r--622/15       2259 Apr   2 16:22 1996 Makefile
-rw-r--r--   0/1      25964 Apr   2 16:19 1996 chdata.o
-rwxr-xr-x   0/1     819200 Apr   2 16:22 1996 scopdsrv
osessn05#
```

3．1．3 サーバ・プログラムのビルド

サーバプログラムをインストール環境で動作させるためにサーバプログラムのビルドを行う。以下の手順でビルドする。
ただし、コンパイラによっては Makefile_srv を修正する必要がある。

Makefile_srv の内容を添付資料 Table12 に示す。

1. **#make -f Makefile_srv clean**
2. **#make -f Makefile_srv**

サーバ実行イメージ scopsrv が作成される。

3．1．4 ObjectBroker サーバ環境設定

原理確認システムがクライアント・サーバ間でデータをやり取りするためには、サーバ側で以下の ObjectBroker 環境設定が必要となる。
システム全体のプログラム構造を Fig.1 に示す。

- クライアント・ホストに対するプロキシの設定
- SCOPD::COP オブジェクト・リファレンスの登録

細かい設定内容については ObjectBroker のマニュアルを参照すること。

3．1．4．1 クライアントホストに対するプロキシの設定

セキュリティー操作の１つで、クライアントアプリケーションとサーバーアプリケーションを異なるノードで実行する場合にはプロキシを設定する必要がある。
オブジェクトブローカでは２つのレベルでセキュリティーを管理している。ひとつめがノードレベルでのセキュリティーの管理で、もう１つはインプリメントレベルでの管理である。
ノードレベルでのセキュリティーの管理では、リモートノードでの特定のユーザーを自ノードのどのユーザーとして扱うかを指す。指定したノードのユーザーからリクエストが送られた結果としてサーバーを起動する場合には、設定してあるユーザーのプロセスとして起動する。
インプリメンテーションレベルでのセキュリティー管理では、各インプリメンテーションを実行可能であるユーザーを定義する。

ObjectBroker でクライアントからの要求をサーバが処理するためには、セキュリティ機能であるプロキシを設定することが必要である。以下にプロキシ設定の方法を示す。

1. root アカウントでアクセスする。
   %su
   passwd:
   #
   ※ ﾊﾟｽﾜｰﾄﾞは、計算機係に参照の事。

2. 以下のコマンドでプロキシを設定する。
   #obbadpxy -h クライアントノード名 -u ¥* サーバユーザ名
   例えば、クライアントホスト名 node1 から、サーバホストユーザ user1 でアクセスするには以下のように入力する。
   #obbadpxy -h osesib37 -u ¥* root
3. プロキシが正常に設定されていることを確認するためには、クライアント・ホストの System Administorator で以下のコマンドを実行する。
   obbmsho -A -n サーバホスト名
   プロキシが正常に設定されている場合、ObjectBroker Agent の情報が表示される。

※ この操作を行うには、あらかじめ Fig.2 に示す設定を行っておかなければならない。

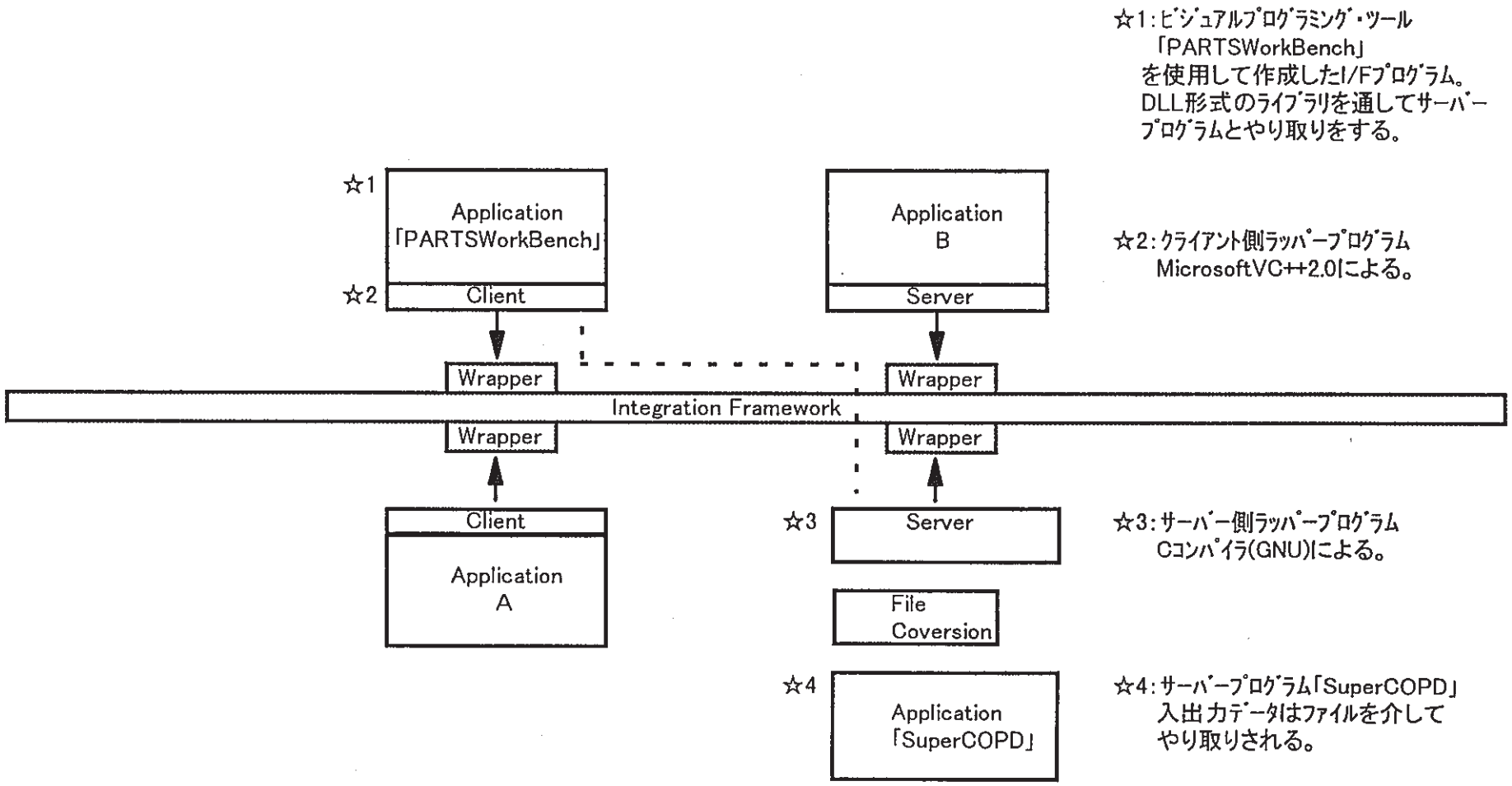


Fig.1　原理確認システムのプログラム構成

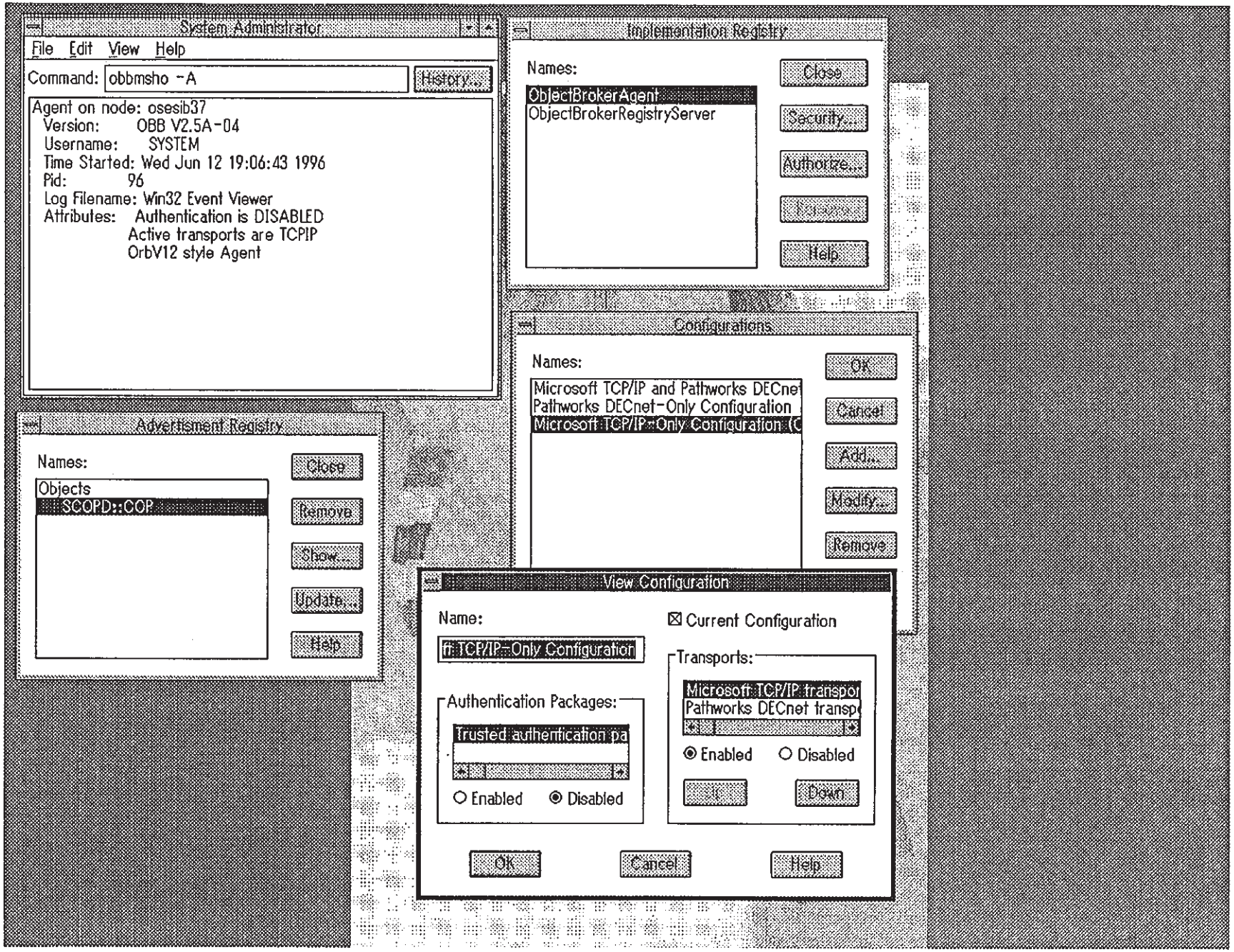


Fig.2 クライアントマシンNT／システムアドミストレータープログラムの設定情報

・コンテキスト・オブジェクトの修正
・ＳＣＯＰＤ：：ＣＯＰオブジェクト・リファレンスのマージ
詳細は、「3.2.3 ＯｂｊｅｃｔＢｒｏｋｅｒクライアントの環境設定」を参照の事。
このコマンドを実行すると Fig.2 の情報が表示される。

セキュリティーの設定には、以下の４つの操作がある。
（1）リモートノードのユーザーを登録する。

UNIX 及びＷｉｎｄｏｗｓＮＴでの操作：
obbadpxy -u ユーザー名 ノード名 ユーザー名

（2）登録したユーザーを削除する。

UNIX 及びＷｉｎｄｏｗｓＮＴでの操作：
obbrmpxy -u ユーザー名 ノード名 ユーザー名

（3）利用可能なインプリメンテーションやメソッドを指定する。

UNIX 及びＷｉｎｄｏｗｓＮＴでの操作：
obbgrath ユーザー名

	OpenVMS	UNIX+WindowsNT	
ｲﾝﾌﾟﾘﾒﾝﾃｰｼｮﾝ名	/IMPLEMENTATION	-I	ﾕｰｻﾞｰがｱｸｾｽできるｲﾝﾌﾟﾘﾒﾝﾃｰｼｮﾝを指定する。
ﾒｿｯﾄﾞ名	/METHOD	-m	ﾕｰｻﾞｰがｱｸｾｽできるﾒｿｯﾄﾞ名を指定する。
ｱｸｾｽﾚﾍﾞﾙ	/ADMIN_METHODS	-a	ﾕｰｻﾞｰがｱｸｾｽできるﾚﾍﾞﾙを指定する。

Table5 コンテキストオブジェクトの表示コマンドのオプション

（4）インプリメンテーションやメソッドを削除する。

UNIX 及びＷｉｎｄｏｗｓＮＴでの操作：
obbrvath ユーザー名

	OpenVMS	UNIX+WindowsNT	
ｲﾝﾌﾟﾘﾒﾝﾃｰｼｮﾝ名	/IMPLEMENTATION	-I	ｱｸｾｽを不可能にするｲﾝﾌﾟﾘﾒﾝﾃｰｼｮﾝを指定する。
ﾒｿｯﾄﾞ名	/METHOD	-m	ｱｸｾｽを不可能にするﾒｿｯﾄﾞ名を指定にする。

Table6 コンテキストオブジェクトの表示コマンドのオプション

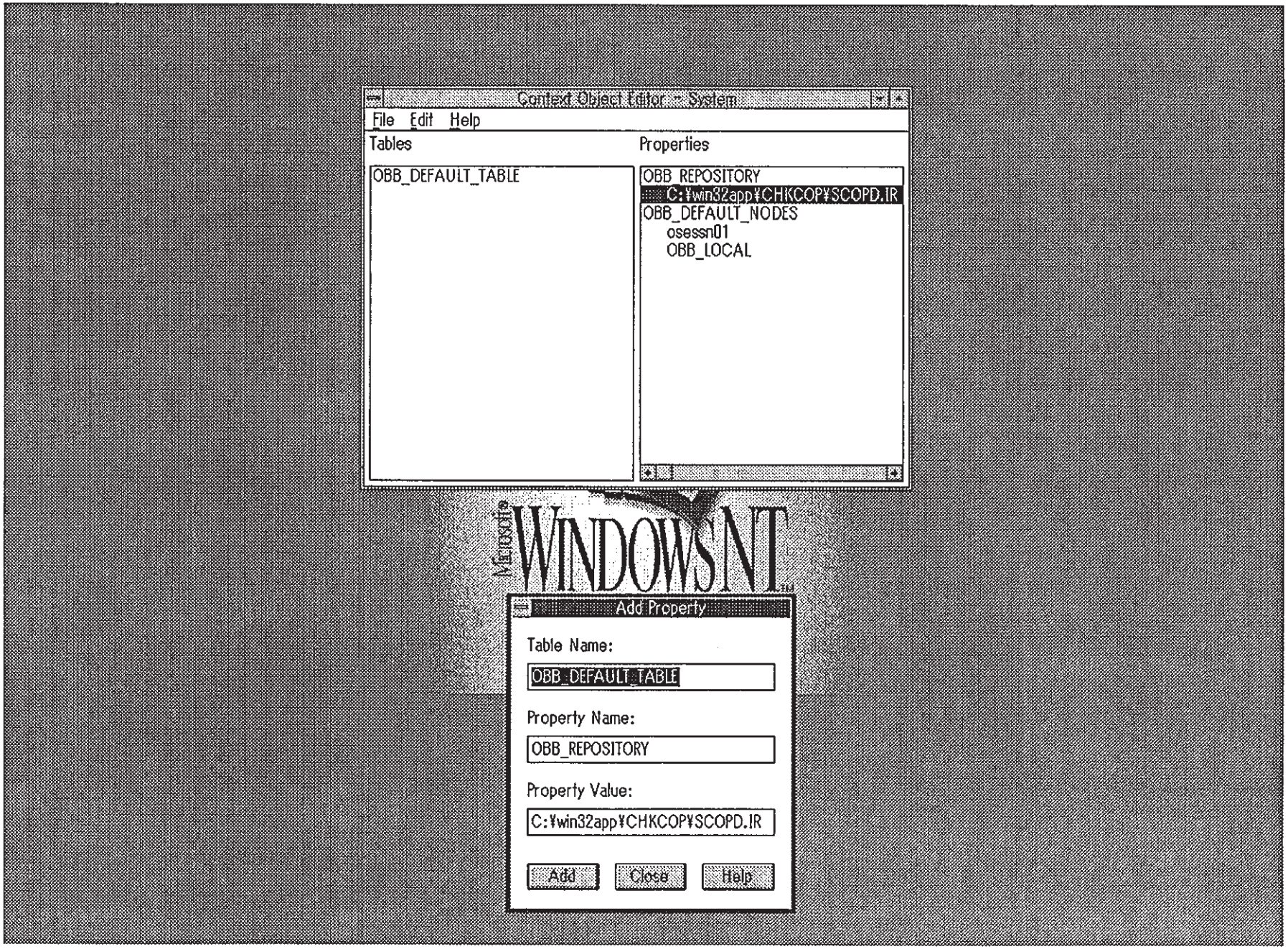


Fig.3　クライアントマシンNT／コンテキストオブジェクトエディターの設定情報

セキュリティーの参照設定には、以下の２つの操作がある。
（１）登録したリモートノードのユーザーを参照する。

UNIX及びＷｉｎｄｏｗｓＮＴでの操作：
obbshpxy -u ユーザー名 ノード名 ユーザー名

（２）ユーザーが利用可能なインプリメンテーションやメソッドを参照することができる。

UNIX及びＷｉｎｄｏｗｓＮＴでの操作：
obbshath ユーザー名

３．１．４．２ SCOPD::COP オブジェクト・リファレンスの登録

SCOPD::COP オブジェクト・リファレンスを ObjectBroker ネームサービスに登録する方法を以下に示す。ルート・アカウントで作業をすること。

1. 原理確認システムをインストールしたディレクトリに移動する。
   %cd /home2/OB/scopd
2. #obbreg -O -f cop_obj.ref SCOPD::COP
   コマンドを入力する。

オブジェクトリファレンス（オブジェクトの識別子）
クライアントがメッセージを送るために必要なものがオブジェクトリファレンスである。オブジェクトへの識別子のことをオブジェクト・リファレンスという。
しかしクライアントからのリクエストを発行した時点ではオブジェクト・リファレンスがオブジェクトと関連付けられているとは限らない。その場合にはオブジェクト・リクエスト・ボローカが実行時にインターフェース・リポジトリと呼ばれるデータベースにアクセスして、呼び出すオブジェクトを決める事になる。
オブジェクト・リファレンスは、クライアント側で作成することは出来ない。
したがってクライアントは以下のような方法によってオブジェクト・リファレンスを入手する。

★ ネームサービスからの入手
★ ファイル渡し
★ プログラム中へのハード・コーディング

クライアントからこのオブジェクト・リファレンスを使用してリクエストを発行するわけだが、これは定義されているクラスにメッセージを送ることになる。オブジェクトリクエストブローカは指定されたクラスとメッセージから、最適なサーバーアプリケーションを選択する。
選択されたサーバーアプリケーションが、クライアントから依頼された処理を実行することになる。

3．1．5 原理確認システム・サーバの環境設定

scopdsrv を実行する前に以下の環境変数を設定しておく必要がある。

- COP_PRG - SCOPD システムの COP コマンドの実行ファイル名をフルパスで指定
- SUMY_PRG - SCOPD システムの sumy2 コマンドの実行ファイル名をフルパスで指定

上記の環境変数が設定されていない場合、SCOPD システムの実行は行われず、データ転送のテスト（データの受け渡し）のみ行われる。

以下に設定例を示す。

```
osessn01#
osessn01# setenv COP_PRG /home2/OB/scopd/COP
osessn01# setenv SUMY_PRG /home2/OB/scopd/SUMY
osessn01#
```

3．2 クライアントの環境設定

3．2．1 前提

クライアントマシンに以下のソフトウェアがインストールされている事が前提となり得る。

- WindowsNT3.51
- VisualSmallTalk V3.0.1 + PARTSWorkBench
- DEC ObjectBroker V2.5 for WindowsNT

また TCP/IP によってクライアント・サーバ間の通信が設定済みであることが前提である。

Fig.4　に WindowsNT／コントロールパネル／ネットワークアイコンの設定情報を示す。

Windows ＮＴ／コントロールパネル／ネットワーク／

トランスポート・プロトコル：ＴＣＰ／ＩＰ

ＩＰアドレス：１３３．１８８．８７．４７
サブネットマスク：２５５．２５５．２５５．１２８
ゲートウェイ・アドレス：１３３．１８８．８７．１

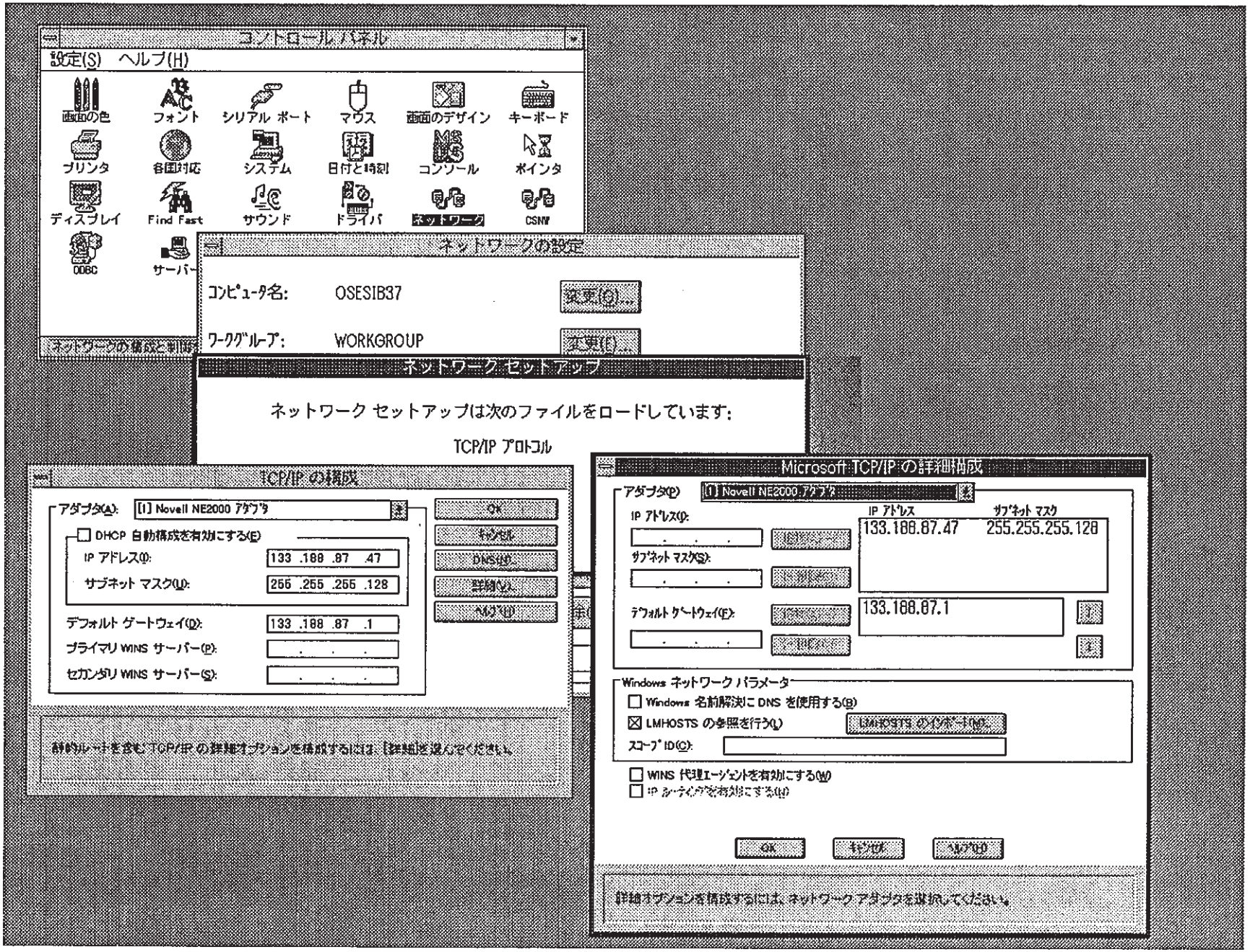


Fig.4 クライアントマシンNT／ネットワーク環境の設定情報

3．2．2 原理確認システム・クライアントのインストールと環境設定

原理確認システム・クライアントはフロッピー・ディスクによって提供され、以下のファイルが含まれる。

- scopdcli.par - 原理確認システム PARTS Workbench クライアント
- scopd.ir - 原理確認システム非同期実行用 ObjectBroker リポジトリ
- scopd.dll - 原理確認システム DLL
- extfloat.sll - （PARTS Workbench）double 配列拡張ファイル

以下の手順でインストールする。

1. extfloat.sll を Visual SmallTalk のトップ・ディレクトリ（例：¥VSEW301）にコピーする。

   NetWare サーバーディスク／R:¥Uty1¥VSEW303¥

2. Visual SmallTalk のトップ・ディレクトリにある、PARTSWBN.BND VDEVW.BND の２つのファイルの末尾に、それぞれ EXTFLOAT という１行を追加する。
   それぞれのファイルの内容を Table7・Table8 に示す。

3. 原理確認システム・クライアント用のディレクトリ（例：C:¥CHKCOP）を作成する。

   クライアント・マシン（ＮＴ）／Ｃ：¥win32app¥CHKCOP¥

4. scopdcli.par と scopd.ir を原理確認システム・クライアント用ディレクトリにコピーする。
5. scopd.dll を DLL を実行可能なディレクトリにコピーする。

   クライアント・マシン（ＮＴ）／Ｃ：¥WINNT35¥

Table7　「ＰＡＲＴＳＷＢＮ．ＢＮＤ」ファイルの内容

```
; PARTS Workbench 3.0 for Win32

VRDBG3A      ; Runtime Debugger Substitute

; enable the following line to prebind the optional libraries in the standard catalog
;@PWBOPRTW.BND

EXTFLOAT
```

Table8　「ＶＤＥＶＷ．ＢＮＤ」ファイルの内容

```
; install PARTS Workbench 3.0.1 for Win32 libraries
@PWBRUNW.BND
@PWBWRKW.BND

; enable the following line to prebind the optional libraries in the standard catalog
@PWBOPRTW.BND


@teamvstd.bnd; team bind file (tools hidden)

VRX303W                    ; Upgrade to the version 3.0.3

JP_FIX              ; Bug fixes for Japanese characters

@fbrsim.bnd         ; PNC's FBR Simulator libraries

EXTFLOAT
```

３．２．３ ObjectBroker クライアントの環境設定

原理確認システムがクライアント・サーバ間でデータをやり取りするためには、クライアント側で以下の ObjectBroker 環境設定が必要となる。

1. コンテキスト・オブジェクトの修正
2. SCOPD::COP オブジェクト・リファレンスのマージ

３．２．３．１ コンテキスト・オブジェクトの修正

システム・コンテキスト・オブジェクトに以下の項目を設定する。

- 原理確認システム・リポジトリの位置
- デフォルト・ノード

以下の手順でシステム・コンテキストの設定を行う。

1. Context Object Editor を起動する。
2. File→Open→System を選択する。
3. OBB_DEFALT_TABLE をダブル・クリック、OBB_DEFALT_NODES を選択した後、Edit→AddProperty... を選択してデフォルト・ノードにサーバ・ノードを追加する。
4. 同様に OBB_REPOSITORY を scopd.ir （フルパス指定）に設定する。

詳しい操作方法に関しては DEC ObjectBroker のマニュアルを参照する。

コンテキスト・オブジェクト

オブジェクトブローカでは、一般的な環境設定にはコンテキスト・オブジェクトを利用する。コンテキスト・オブジェクトを利用するには、コンテキストオブジェクト言語(ContextObjectLanguage／ＣＯＬ）を利用する。

コンテキストオブジェクトの構造

コンテキストオブジェクトは、複数のテーブルから構成される。それぞれのテーブルで任意の数のプロパティを定義することができる。テーブルは、参照するアプリケーションや設定するプロパティのカテゴリなどで分割することになる。プロパティは文字のみが設定できる。任意の数の値が設定できる。

```
CONTEXTOBJECT
        TABLETable_A
                attribute_A1 = value_A1X, value_A1Y;
                attribute_A2 = value_A2
         end Table;
         Table Table_B
                attribute_B1 = value_B1;
                attribute_B2 = value_B2
         end Table;
end contextobject
```

デフォルトで参照されるテーブルとして OBB_DEFAULT_TABLE が用意されている。このテーブルに設定することになるプロパティとして OBB_REPOSITORY と OBB_DEFAULT_NODS がある。
OBB_REPOSITORY にはインターフェース・リポジトリ、インプリメンテーション・リポジトリのファイル名を指定する。OBB_DEFAULT_NODES にはサーバーのデフォルトノードを指定する。

プロパティ値として OBB_ENVIRONMENT_VARIABLE を設定すると、プロパティ名と同じ名前の環境変数（UNIX の場合）あるいは論理名（ＶＭＳの場合）を参照することになる。

コンテキスト・オブジェクトのレベル

コンテキストオブジェクトには、ユーザー・グループ・システムの３レベルがある。それぞれのレベルのコンテキストオブジェクトには、ファイルのディレクトリおよびファイル名で参照する。

★ＵＮＩＸの場合
- ★ユーザーレベル　カレントディレクトリの OBB_USER_CONTEXT.CO
- ★グループレベル　カレントディレクトリの OBB_GROUP_CONTEXT.CO
- ★システムレベル　/var/adm/ObjectBroker/OBB_SYSTEM_CONTEXT.CO

★Windows の場合
- ★ユーザーレベル　カレントディレクトリの OBBUSR C.CO
- ★グループレベル　カレントディレクトリの OBBGRPC.CO
- ★システムレベル　C:¥OBROKER¥REPOS¥OBBSYSC.CO

★WindowsNT の場合
- ★ユーザーレベル　カレントディレクトリの OBBUSR C.CO
- ★グループレベル　カレントディレクトリの OBBGRPC.CO
- ★システムレベル　C:¥WIN32APPL¥OBROKER¥REPOS¥OBBSYSC.CO

ユーザーコンテキストオブジェクトとグループコンテキストオブジェクトを、環境変数あるいは論理名でデフォルトから変更することができる。

Table9　デフォルトコンテキストオブジェクト

ﾚﾍﾞﾙ	UNIX および WindowsNT の環境変数	VMS での論理名
ﾕｰｻﾞ	OBB_USER_CONTEXT	OBB_USER_CONTEXT
ｸﾞﾙｰﾌﾟ	OBB_GROUP_CONTEXT	OBB_GROUP_CONTEXT

コンテキストオブジェクトを参照する場合、ヘェイルオーバー機能を利用することができる。
ヘェイルオーバー機能とは、ユーザ・グループ・システムレベルの順に該当する属性が存在するレベルまで検索する機能である。

コンテキストオブジェクトのロード
コンテキストオブジェクトを利用するには、定義ファイルを次にのコマンドで変換する必要がある。

UNIX および WindowsNT
obbldctxｺﾝﾃｷｽﾄｵﾌﾞｼﾞｪｸﾄ定義ﾌｧｲﾙ

Table10　コンテキストオブジェクトのロードコマンドのオプション

	OpenVMS		UNIX および WindowsNT
ｺﾝﾃｷｽﾄｵﾌﾞｼﾞｪｸﾄ	/USER	-U	設定するｺﾝﾃｷｽﾄｵﾌﾞｼﾞｪｸﾄのﾚﾍﾞﾙを指定する
	/GROUP	-G	
	SYSTEM	-S	
ｺﾝﾃｷｽﾄｵﾌﾞｼﾞｪｸﾄﾌｧｲﾙ名	/CONTEXT_OBJECT	-U	変換するｺﾝﾃｷｽﾄｵﾌﾞｼﾞｪｸﾄﾌｧｲﾙを指定する
新規作成	/CREATE	-c	ｺﾝﾃｷｽﾄｵﾌﾞｼﾞｪｸﾄを新規に作成する

コンテキストオブジェクトの参照
コンテキストオブジェクトを次のコマンドで参照することが出来る。
UNIX および WindowsNT
obbshctx

Table11　　コンテキストオブジェクトの表示コマンドのオプション

	OpenVMS	UNIX および WindowsNT	
ｺﾝﾃｷｽﾄｵﾌﾞｼﾞｪｸﾄの定義	/USER /GROUP SYSTEM	-U -G -S	参照するｺﾝﾃｷｽﾄｵﾌﾞｼﾞｪｸﾄのﾚﾍﾞﾙを指定する
ｺﾝﾃｷｽﾄｵﾌﾞｼﾞｪｸﾄﾌｧｲﾙ名	/CONTEXT_OBJECT	-C	参照するｺﾝﾃｷｽﾄｵﾌﾞｼﾞｪｸﾄﾌｧｲﾙを指定する。
ﾃｰﾌﾞﾙ名	/TABLE	-T	参照するｺﾝﾃｷｽﾄｵﾌﾞｼﾞｪｸﾄのﾃｰﾌﾞﾙ名を指定する。
環境変数あるいは論理名の参照	/EXPAND_VARIABLES	-e	参照している環境変数あるいは論理名を表示する。

コンテキストオブジェクト設定例を以下に示す。
Fig.3　WindowsNT コンテキストオブジェクトの設定例

3．2．3．2 SCOPD::COP オブジェクト・リファレンスのマージ

原理確認システム DLL がサーバにアクセスするためには、クライアントのネームサービスに SCOPD::COP オブジェクト・リファレンスを登録することが必要である。サーバのネームサービスをマージすることによって設定される。ネームサービスのマージ方法を以下に示す。

1. System Administrator を起動する。
2. View→Advertisement Registry を選択し、Updata... を押す。
3. Computer... ボタンを押して、サーバノードを指定する。
4. OK ボタンを押してマージを行する。

詳しい操作方法に関しては DEC ObjectBroker のマニュアルを参照する。

４．原理確認システムの実行方法

概要

本システムは、２つの実行形態によって成り立っている。
（１）同期実行形態、（２）非同期実行形態 それぞれの詳細については以降に示す。
２つの処理（プログラム等）があり親プログラムと子プログラムに分かれる。親プログラムが子プログラムの終了を待つ方式（１）、待たずに処理を進める方式（２） という違いによって区別されている。
ネットワークを介して各計算機に処理を分散し並列で実行をしようとした場合、（２）は必ず使用する制御技術である。
こうした実行形態を実際に行いさまざまな型のパラメータを受け渡し、一連の流れが正常に行なわれるかどうかを確認している。
原理確認システムの実行方法を示す。

制御の流れ
ｸﾗｲｱﾝﾄ･ﾌﾟﾛｸﾞﾗﾑに入力ﾊﾟﾗﾒｰﾀ項目を与えそれをｻｰﾊﾞｰ･ﾌﾟﾛｸﾞﾗﾑに受け渡す
ｻｰﾊﾞｰ･ﾌﾟﾛｸﾞﾗﾑはﾃﾞｰﾀを受け取り「Ｓ－ＣＯＰＤ」を起動する。
Ｓ－ＣＯＰＤが終了したらﾊﾟﾗﾒｰﾀ項目を変更してｸﾗｲｱﾝﾄ･ﾌﾟﾛｸﾞﾗﾑに返す
そしてｸﾗｲｱﾝﾄ･ﾌﾟﾛｸﾞﾗﾑは受け取った出力ﾊﾟﾗﾒｰﾀ項目を表示する。

（１）同期実行形態
起動をかけると入力ﾊﾟﾗﾒｰﾀ項目をｻｰﾊﾞｰﾌﾟﾛｸﾞﾗﾑに渡し、ｻｰﾊﾞｰﾌﾟﾛｸﾞﾗﾑを実行させる。
そしてｻｰﾊﾞｰﾌﾟﾛｸﾞﾗﾑが終了し結果ﾃﾞｰﾀ(出力ﾊﾟﾗﾒｰﾀ項目)を返すまで wait 状態に入り
ﾃﾞｰﾀが戻った時点でそれを表示する。

（２）非同期実行形態
起動をかけると入力ﾊﾟﾗﾒｰﾀ項目をｻｰﾊﾞｰﾌﾟﾛｸﾞﾗﾑに渡し、ｻｰﾊﾞｰﾌﾟﾛｸﾞﾗﾑを実行させる。
そして起動を掛けたら直ぐにｸﾗｲｱﾝﾄﾌﾟﾛｸﾞﾗﾑに制御を戻す、これによりｸﾗｲｱﾝﾄ･ﾌﾟﾛｸﾞﾗﾑは別の処理を進める。
ｻｰﾊﾞｰﾌﾟﾛｸﾞﾗﾑが終了し結果ﾃﾞｰﾀ(出力ﾊﾟﾗﾒｰﾀ項目)を返す割り込みが来た時点で現在の処理を中断しそれを表示する。

４．１ サーバ・プログラムの起動

今回の原理確認システムは原理確認システム・サーバを予め起動しておく事が必要である。SunOS の端末上で以下の手順によって起動する。

1. 環境変数 COP_PRG を設定する。例えば COP コマンドが /usr/bin/COP で csh を使用している場合、
   %setenv COP_PRG /home2/OB/scopd/COP
   と入力する。

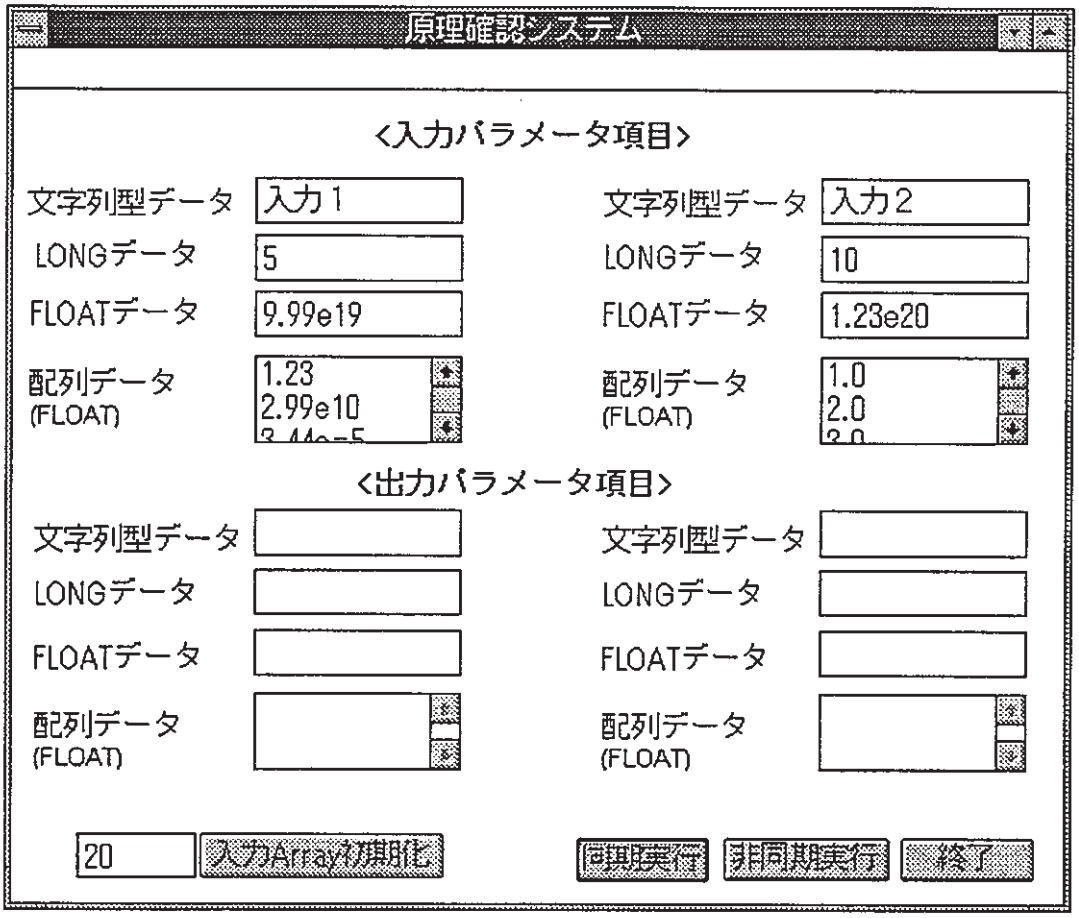
2. 環境変数 SUMY_PRG を設定する。例えば sumy2 コマンドが /usr/bin/sumy2 で csh を使用している場合、
   %setenv COP_PRG /home2/OB/scopd/SUMY
   と入力する。
3. %scopdsrv &
   を入力してサーバ・プログラムを起動する。

４．２ 原理確認システム・クライアントの実行

サーバプログラムの起動が終了した後、原理確認システム・クライアントを実行する。scopdcli.par を PARTS Workbench にロードして実行する。

４．３ 原理確認システム・クライアントの使用方法

scopdcli.par を launch すると、下に示すようなダイアログボックスが表示される。



実行は以下の手順で行う。

1. <入力パラメータ項目>の文字列型データ、LONG データ、FLOAT データのテキスト・ボックスに適当なデータを入力する。入力データの左半分の領域は各テキスト・ボックスごと単純なデータ型で、右半分の領域は構造体データである。配列データは変更することはできない。
   入力の際、入力データ部分は、各データ型に従った有効なデータを入力すること。データが有効でない場合、またはデータが入力されていない場合、VisualSmallTalk のデバッガが動作してしまう。
2. 「入力 Array 初期化ボタン」の左のテキスト・ボックスに配列データの配列数を入力し、「入力 Array 初期化ボタン」をクリックすると配列データにデータが設定される。例えば、テキスト・ボックスに 500 を入力すると 500 個の配列が作成される。
3. 同期実行ボタンをクリックすると、原理確認システム・サーバが実行され、<出力パラメータ項目の>各エントリに結果が出力される。文字列型データは入力がエコーバックされ、その他は入力データが -1 されて表示される。
4. 「終了ボタン」を押すと原理確認システム・クライアントは終了する。

※ パラメータについて...

あらゆる型のパラメータ・データが正しく受け渡るかどうかを確認している。

大量のデータを受け渡した場合どの程度時間がかかるか見るために、全てのパラメータのデータ容量の合計が１ＭＢになるように設定している。

データ容量は任意に設定出来るので、１ＭＢ以下でも１ＭＢ以上でもさまざまな容量のデータを受け渡すことができる。

１）文字列型データ：文字列１０文字
２）整数型：整数型で表される最大値
３）長整数型：長整数型で表される最大値
４）浮動小数点型：浮動小数点で表される最大値、10E32～10E-32
(Smalltalk では、単精度・倍精度の区別が無く FLOAT と宣言しておけば自動的に対応してくれる)
５）配列：FLOAT 型データの配列で、配列の数は１）～６）の全パラメータの総容量が１ＭＢになるように調整する。
６）構造体：メンバは、１）～５）に示すデータ型要素であること。
これを１セットのみ受け渡せばよい。
尚構造体の中の配列の数は１０個とする。

５．原理確認システムの注意点

原理確認システムを使用する上での注意点と今後のシステム作成において考慮すべき点を記述する。

５．１ 同一クライアント・ノードでの複数クライアントの実行

今回の原理確認システムにおいては､DLL 内のデータ管理を DLL の静的変数領域によって管理しているため、同一ノードで原理確認システムを複数インスタンス実行することはできない。特に非同期実行をおこなう場合は注意する。

５．２ 原理確認システム DLL を使用する場合の注意点

今回の原理確認システム･クライアント以外で原理確認システム DLL を使用する場合の注意点を示す。

- SyncCopExec、AsyncCopExec を実行した後、出力配列データ領域を解放するため、出力配列データが不要になった場合、FreeOutSeq を呼び出して配列メモリ領域を解放すること。
- 非同期実行（AsyncCopExec）を実行する場合、AsyncCopExec をコールした後、PollMessage を定期的に実行してサーバの処理の終了をチェックする。また、サーバの処理が終了した場合（PollMessage がの戻り値が TRUE の場合)、必ず GetResult を実行して結果を取得すること。結果を取得するまで、次の AsyncCopExec （または SyncCopExec）を呼び出す事はできない。

５．３ 膨大な配列サイズのデータ受け渡しに対するパフォーマンスの低下

原理確認システムクライアントの「入力 Array 初期化」ボタンの左のテキスト・ボックスに入力する配列数を非常に大きく（例：120000 約 1MB のデータ）した場合、原理確認システム・クライアントと原理確認システム DLL 間のデータ受け渡しがネックとなり著しいパフォーマンス低下が起こる。

- 原理確認システム・クライアントを使用せず、VC++ V2.0 (Optimize なし）で記述された同一のデータを処理させるクライアントの場合、128000 個の配列のデータ（約 1MB）転送に約 40 秒の時間を要す。(注：クライアントサーバ間の転送時間も含まれる)
- 原理確認システム・クライアントを使用した場合、データを ArrayHolder にセットする時間がネックとなる。メモリ中の配列データを ArrayHolder に格納するのに、約 6 時間の時間を要す。(注：これはクライアント・メモリ（DLL）上のデータを Array Holder に格納する時間のみの計算で、サーバとの通信は行っていない)

従って、膨大なサイズのデータを処理する場合、ArrayHolder の使用をさけ、高速にデータアクセス可能なクラス設計を行う必要がある。あるいは、DLL でデータのアクセス処理を行

い、PARTS Workbench で扱うデータ量の絞り込みを DLL 部品によって行うことも有効な方法になる。これらは実システムの個々のデータ処理の実装において考慮する必要がある。

※ 以上のような結果を踏まえ受け渡すパラメータ・データの容量＋期待するレスポンスタイムに応じて以下の中から受け渡す方式を決めることになる。

（1）直接 実データを受け渡す。
この方法が最も一般的であり、プログラム自体も分かりやすいものとなる。
特徴：
データ容量はそれほど多くない場合あるいは、受け渡す手順を簡略に済ませたい場合に使用する。

（2）データベースを介してＩＮＤＥＸデータのみ受け渡す。
会社の基幹システムとして稼動するような大規模システムでは、この方式を採用している場合が多い。
特徴：
データ容量が膨大な場合。レスポンスタイムを小さく抑えたい場合等はこの方式を採用する。大規模システムなどでしばしば見受けられる。
またシステムを長いサイクルで稼動させ、その間受け渡すデータの容量が増減しても充分に耐えられる能力を持っている事もこの方式の特徴である。
逆にデメリットとしては、データベースシステムを必要とするため、相応の初期投資が掛かり、また構築作業も行わなければならない。

Table12　Makefile_srv の内容

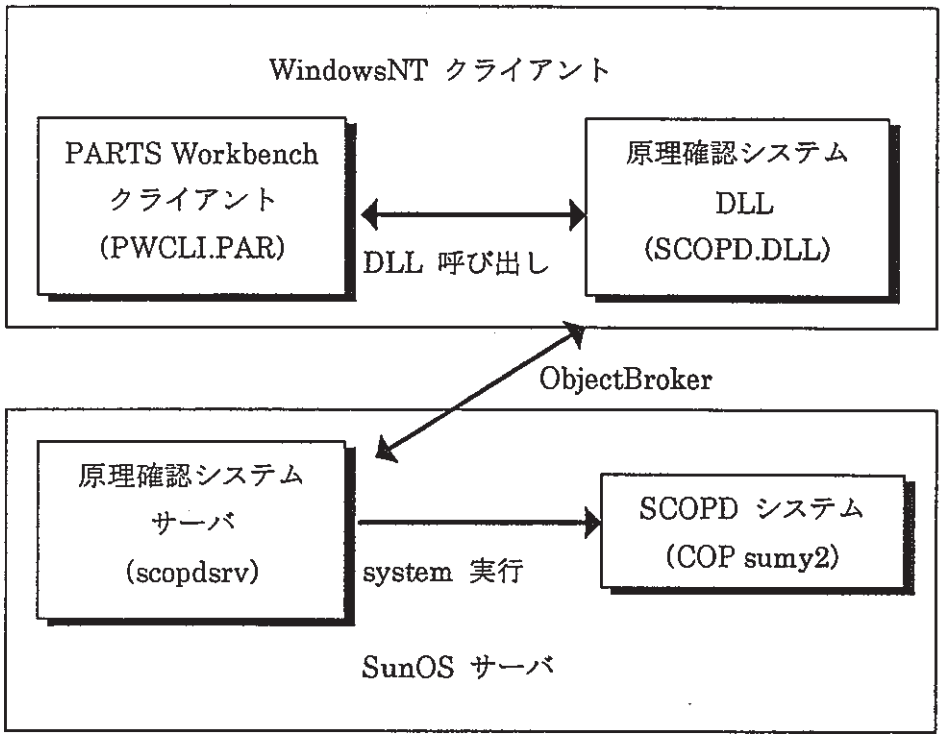
```
#
CC=/usr/local/bin/gcc

# LIBRARY NAMES GIVEN BELOW
LIBRARIES= -lobb
#LIBRARIES= -lobb
#/*   -lc_gnu   */

# LINKER GIVEN BELOW
LD=$(CC)
#LD=cc

# LINKER FLAGS GIVEN BELOW
LDFLAGS= -O

# DEFINES
AR=ar
ARFLAGS=rv
CP=cp
RM=rm -f

# COMPILER FLAGS GIVEN BELOW
CFLAGS= -g
PFLAGS=$(CFLAGS)
FFLAGS=$(CFLAGS)
COBFLAGS=$(CFLAGS) -xref


# INCLUDE SEARCH PATH GIVEN BELOW
INCLUDES=

.SUFFIXES: .uid .uil .o

# SOURCES GIVEN BELOW
SRCS= method.c¥
 srv_main.c¥
 disp.c¥
 chdata.c¥
 obbansi.c

# OBJECTS GIVEN BELOW
OBJS= method.o¥
 srv_main.o¥
 disp.o¥
 chdata.o¥
```

```
 obbansi.o

scopdsrv : $(OBJS)
        $(LD) -o $@ $(LDFLAGS) $(OBJS) $(LIBRARIES)

.c.o:
        $(CC) -c $(CFLAGS) $(INCLUDES) `pwd`/$<

.c.a:
        $(CC) -c $(CFLAGS) $(INCLUDES) `pwd`/$<
        $(AR) $(ARFLAGS) $@ $*.o
        $(RM) $*.o


IR: scopd.idl scopd.iml scopd.mml cstub method_tmp.c disp.c scopd.ir

#scopd.idl: scopd.idl
#       obbgen -u -f scopd.idl

#scopd.iml:
#       obbcomp -i scopd.iml scopd.idl

#scopd.mml: scopd.idl scopd.iml
#       obbcomp -m scopd.mml scopd.idl scopd.iml

method_tmp.c : scopd.idl scopd.iml scopd.mml
        obbcomp -t method_tmp.c scopd.idl scopd.iml scopd.mml

disp.c : scopd.idl scopd.iml scopd.mml
        obbcomp -d disp.c scopd.idl scopd.iml scopd.mml

cstub : scopd.idl scopd.iml scopd.mml
        obbcomp -T type.c -x -ps -c stub.c scopd.idl scopd.iml scopd.mml

scopd.ir : scopd.idl scopd.iml scopd.mml
        rm -f scopd.ir
        obbldrep scopd.idl scopd.iml scopd.mml
clean:
        $(RM) $(OBJS) core scopdsrv

touchsrcs:
        touch $(SRCS)

lint:
        lint $(INCLUDES) $(SRCS) $(LINTLIBS)

depend:
```

```
	@cat < /dev/null > makedep
	@for i in ${SRCS}; do ¥
		($(CC) -M $(INCLUDES) $$i >> makedep); done
#
# Remove the next two lines to keep dependencies on system include files
#
	@grep -v "¥.o:[ ]*/usr/include" makedep > makedep1
	@mv makedep1 makedep
	@echo '/^# DO NOT DELETE THIS LINE/+1,$$d' > eddep
	@echo '$$r makedep' >> eddep
	@echo 'w' >> eddep
	@${CP} Makefile_srv Makefile_srv.bak
	@ed - Makefile_srv < eddep
	@$(RM) eddep makedep
	@echo '# DO NOT EDIT THIS FILE HERE.' >> Makefile_srv
	@echo '# USER EDITS MUST PRECEDE THE COMMENT:' >> Makefile_srv
	@echo '# "# DO NOT DELETE THIS LINE".' >> Makefile_srv
	@echo '# see make depend above' >> Makefile_srv

# DO NOT DELETE THIS LINE -- make depend uses it
```

6．原理確認システムの構成

原理確認システムの構成は以下の通りである。



原理確認システム プログラム構成

6．1 PARTS Workbench クライアント

原理確認システムのユーザインターフェースを提供するクライアント・プログラムである。PARTS Workbench によって作成されており、PARTS Workbench クライアントは DLL 部品を用いて原理確認システム DLL を呼び出すことによって、サーバへアクセスする。

6．1．1 PARTS Workbench クライアント ファイル構成

PARTS Workbench クライアントの構成を以下に示す。

- scopdcli.par　　PARTS Workbench クライアント PAR ファイル
- extfloat.sll　　拡張インスタンス・メソッド追加ライブラリ

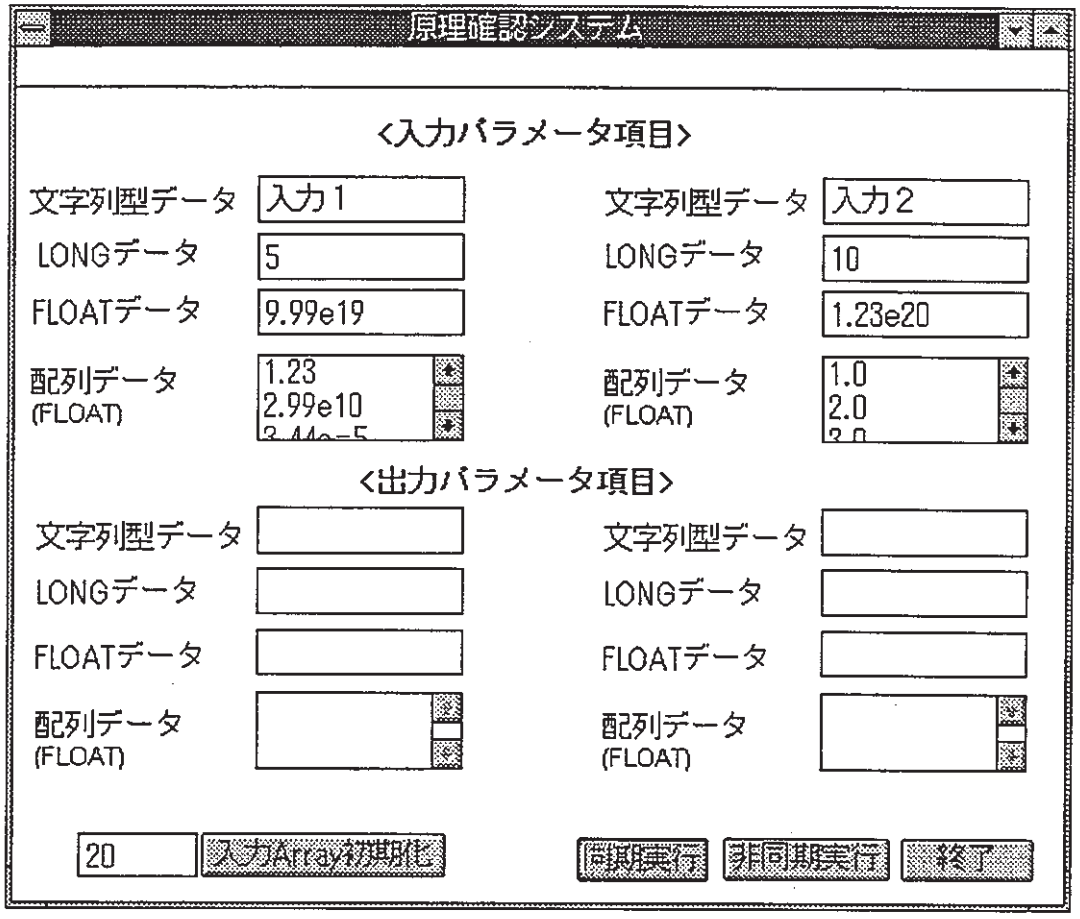
6．1．1．1 拡張モジュールの使用

通常の PARTS Workbench で提供されている External Buffer クラスでは float を扱うメソッドが提供されていない。そこで原理確認システムでは、External Buffer クラスを float 拡張するための、extfloat.sll を使用する。extfloat.sll では以下の２つのインスタンス・メソッドが追加されている。

- floatAtOffSet:
- floatAtOffSet:Put:

6．1．2 PARTS Workbench クライアント GUI

PARTS Workbench クライアントの画面イメージを示す。



6．2 原理確認システム DLL

PARTS Workbench クライアントから呼び出される DLL である。原理確認システムサーバと ObjectBroker を使用して情報のやり取りを行う ObjectBroker クライアントである。

6．2．1 ファイル構成

原理確認システム DLL は scopd.dll である。このファイルは以下の C モジュールからビルドされる。

- chkcop.cpp 原理確認システム DLL モジュール
- その他 MFC DLL ファイル (scopd.cpp 等)

6．2．2 DLL 関数

scopd.dll は以下の 7 つの関数を提供する。

- SyncCopExec
- AsyncCopExec
- PollMessage
- CopGetResult
- FreeOutSeq
- InitDouble
- FreeData

6．2．2．1 SyncCopExec

説明：
SCOPD システムを同期実行する。ObjectBroker の SCOPD::COP オブジェクトに対して Exec メッセージ送信を行う。Exec メッセージは、静的インタフェースでコールされる。

引き数：

型	名前	説明
char*	instring	入力文字列
long	inlong	入力整数
double	indouble	入力 Double
double*	indseq	入力 Double 配列
long	indseq_num	入力 Double 配列の配列数
DataStr	indstr	入力構造体
char*	outstring	出力文字列
long*	outlong	出力整数

double*	outdouble	出力 Double
double**	outdseq	出力 Double 配列
long*	outdseq_num	出力 Double 配列の配列数
DataStr*	outdstr	出力構造体

戻り値：int

SUC_EXECUTE	同期呼び出し成功
ERR_GETOBJ	オブジェクト・リファレンス取得エラー
ERR_EXECUTE	同期処理・実行エラー

6．2．2．2 AsyncCopExec

説明：
SCOPD システムを非同期実行する。ObjectBroker の SCOPD::COP オブジェクトに対して Exec メッセージ送信を行う。Exec メッセージは、動的インタフェースでコールされる。

引き数：

型	名前	説明
char*	instring	入力文字列
long	inlong	入力整数
double	indouble	入力 Double
double*	indseq	入力 Double 配列
long	indseq_num	入力 Double 配列の配列数
DataStr	indstr	入力構造体

戻り値：int

SUC_EXECUTE	非同期呼び出し成功
ERR_GETOBJ	オブジェクト・リファレンス取得エラー
ERR_GETCTX	デフォルト・コンテキスト取得エラー
ERR_CREATREQ	要求オブジェクト作成エラー
ERR_ADDARG	引き数リスト作成エラー
ERR_REQSEND	非同期実行・要求送信エラー

6．2．2．3 PollMessage

説明：
非同期実行をポーリングする。AsyncCopExec が実行された後は必ずこの関数を定期的にコールして、サーバの処理の終了をポーリングする必要がある。

引き数：
なし

戻り値：int

TRUE	サーバ処理終了
FALSE	サーバ処理中

6．2．2．4 CopGetResult

説明：
非同期実行の結果を取得する。PollMessage が TRUE （サーバの処理が終了したとき）で返ってきたときは、必ず CopGetResult をコールして結果を取得してください。

引き数：

型	名前	説明
char*	outstring	出力文字列
long*	outlong	出力整数
double*	outdouble	出力 Double
double**	outdseq	出力 Double 配列
long*	outdseq_num	出力 Double 配列の配列数
DataStr*	outdstr	出力構造体

戻り値：
なし

6．2．2．5 FreeOutSeq

説明：
SyncCopExec AsyncCopExec で確保した出力配列データ領域を解放する。このルーチンは、ObjectBroker ルーチンによって確保された、メモリ領域を解放するためのルーチンである。

引き数：
なし

戻り値：
なし

6．2．2．6 InitDouble

説明：
double 型の配列データの領域を確保して、double 型の最大値（DBL_MAX）で初期化する。

引き数：

型	名前	説明
unsigned long	dsize	データを確保する double 配列の配列数

戻り値：double*
確保した配列データへのポインタ

６．２．２．７ FreeData

説明：
原理確認システム DLL で確保した領域を解放する。この関数は DLL 内で malloc によって確保したデータの解放を行う。

引き数：

型	名前	説明
void*	ptr	解放するメモリ領域へのポインタ

戻り値：
なし

６．２．３ ObjectBroker インターフェース定義

原理確認システムで利用する、ObjectBroker インターフェース定義を示す。原理確認システム DLL はこのインターフェース定義に従ってオブジェクトに対してメッセージ送信を行う。

```
/*
   SCOPD 原理確認システム 1
   Interface Definition   V1.0
*/

module SCOPD {

  enum ERR_TYPES {
        ErrCopExec,
        ErrSumyExec,
        ErrCopRunning,
        ErrGetReslt
  };

  exception UserExcep {
    ERR_TYPES Reason;
  };

  typedef sequence<double> DoubleSeq;

  /*
   テスト用構造体データ
  */
  struct DataStr {
        string          stringdata;
        long            longdata;
```

```
        double              doubledata;
        DoubleSeq     dseqdata;
    };

    interface COP{

        void Exec(
            in   string      instring,
            in   long        inlong,
             in   double     indouble,
            in   DoubleSeq          indseq,
             in   DataStr    indstr,
            out string       outstring,
            out long         outlong,
             out double      outdouble,
            out DoubleSeq outdseq,
             out DataStr     outdstr)
        raises(UserExcep);

    };
};
```

6．3 原理確認システムサーバ

SCOPD システムの起動を行う ObjectBroker のサーバプログラムである。原理確認システム DLL から Exec メッセージが送信されたとき、原理確認システム・サーバの COPImpl_Exec メソッドがディスパッチされる。COPImpl_Exec メソッドは、system システムコールによって SCOPD システムを実行し、結果を原理確認システム DLL に返す。

6．3．1 ファイル構成

原理確認システムは以下のファイルから構成される。

- scopdsrv　　原理確認システム・サーバ

scopdsrv は以下のソースコードから生成される。

- Makefile_srv　　scopdsrv 生成用 make ファイル
- chdata.c　　テストデータ変換用ユーティリティ関数
- srv_main.c　　サーバ・メイン関数
- disp.c　　ObjectBroker ディスパッチ・ルーチン
- scopd.imh　　scopdsrv ヘッダ・ファイル
- scopd.h

6．4 SCOPD システム

SCOPD システムの本体である。原理確認システム・サーバから起動される。SCOPD システムは以下の２つのプログラムから構成される。

- COP コマンド
- sumy2 コマンド

これらのプログラムは原理確認システム・サーバを実行する際、COP_PRG、SUMY_PRG 環境変数によって指定されたコマンドを実行する。これらの環境変数が設定されていない場合、SCOPD システムは実行されないため COP sumy2 は不要となる。

ファイル一覧

原理確認システムで使用するプログラム・ファイル一覧を示する。

クライアント
- pwcli.par
- extfloat.sll
- scopd.dll

サーバ
- Makefile_srv
- chdata.c
- method.c
- srv_main.c
- disp.c
- scopd.h
- scopd.imh
- scopd.tch
- scopdsrv（上記ファイルからビルドされる）

7． 連携処理機能について

「過渡熱応力リアルタイムシミュレータ PARTS」は、熱、流体および構造の複合現象である高速炉の熱過渡現象に関し、熱過渡事象時の流体の熱流動計算、構造の応力計算および強度計算を統合して実施する環境である。その中で PARTS-FLOW は、評価対象のプラントをモデル化し、熱過渡事象時の熱流動計算を行い、シミュレーション全体を制御するプログラムである。

PARTS は異なるプログラムの連携によって機能するシミュレーション環境であり、解析に際しては PARTS-FLOW で計算タイムステップ毎に算出される流体の温度・流量を、構造の応力／強度計算プログラムへ引き渡す必要がある。また、それぞれのプログラム間の同期を取るなど、複数のプログラムの連携上の処理も必要である。

7．1 VisualSmalltalkの連携処理

7．2 DDE Client（DDEクライアント）

DDE クライアントは機能部品で、ダイナミック データ エクスチェンジ (DDE) を通して DDE サーバー アプリケーションにアクセスする。ダイナミック データ エクスチェンジは、サーバー・クライアント間の対話を支援する。クライアントはサーバーに何かを頼み、サーバーがそれに応える。DDE クライアント部品はこのクライアントのアプリケーションを作成するためのものである。サーバーにするのは、他の PARTS ワークベンチ アプリケーションでも、他の言語を用いて用意したアプリケーションでも、DDE サーバーとして動く商用アプリケーションでも、どれでも構わない。

① 操作

DDE クライアント部品の属性ダイアログで、サーバー アプリケーションの名前と通信トピックを指定する。サーバーはクライアントが通信を試みる前に起動されている必要がある。

クライアントはサーバーに命令やデータを送り、またサーバーから命令やデータを受け取ることができる。サーバーへの情報送達は、ポーク項目メッセージというものを送ることによって行なう。このメッセージを受け取るサーバーは、それによって発生するポーク項目イベントを備えている必要がある。

一方サーバーがクライアントに情報を送るには、ホットリンク項目メッセージを送り、受け取るクライアントでは、対応するホットリンク項目イベントが発生しなければならない。どちらの種類のメッセージも引数を取ることができ、同様にイベントは値を伝えることができる。これらのメッセージとイベントは、他の PARTS ワークベンチ部品で使われるメッセージとイベントに大変良く似ているが、いくつか重要な違いがある。

ポーク項目メッセージとホットリンク項目メッセージは、アプリケーションからアプリケーションへ、PARTS ワークベンチではなくオペレーティング システムによって伝達される。アプリケーション間に目に見える形で明示的にリンクを張ることはしない。これは、PARTS ワークベンチのアプリケーションが、他の PARTS ワークベンチのアプリケーションのみならず、DDE 通信を行えるどんなアプリケーションとでも、通信し合えることを意味している。ポーク項目とホットリンク項目のメッセージとイベントはまた、他のメッセージやイベントと異なり、部品ごとに定義を行う。

ポーク項目メッセージとホットリンク項目イベントは、DDE クライアント部品の属性ダイアログで定義する。ポーク項目メッセージを一つ定義するごとに、それと同じ名前を持つ（通常の）メッセージが DDE クライアント部品に登録される。このメッセージは引数を一個取ります。ホット

リンク項目イベントに対しても、値を一個持つ新たなイベントがそれと同じ名前で DDE クライアントに加えられる。これらのイベントは、サーバーが対応するメッセージを実行したときに発生するようになる。

② イベント

DDE クライアント部品は、属性ダイアログで定義した各ホットリンク項目イベントと同じ名前のイベントを持つ。各イベントともイベント値を一個持つ。ある項目が更新されると、対応するイベントが発生して、更新内容はイベント値として取り出すことができる。

- **initiateException: applicationString topic: topicString**

  サーバー・クライアント間の対話が開始できない時に発生する。属性ダイアログで設定されたアプリケーション名とトピック名がイベント値として与えられる。

  発生因子： initiate

- **hotLink: aValue**

  属性ダイアログで定義したホットリンク項目のどれかをサーバーが更新すると、このイベントが発生する。更新後の値が aValue によって与えられる。イベント名の hot-link の部分は、実際には対応する項目名になる。

  発生因子： サーバーの hot-link メッセージ

③ メッセージ

DDE クライアント部品は、属性ダイアログで定義した各ポーク項目メッセージと同じ名前のメッセージを持つ。各メッセージとも引数を一個取る。項目の更新は、更新値を引数にしたこれらのメッセージを実行して行う。

- **executeCommand: aString**

  サーバーに aString で表されるコマンドの実行を要求する。

  戻り値：-

  発生イベント： The server executeCommand: event

- **initiate**

  DDE の対話を開始する。他の全てのメッセージに先立って実行されなければならない。

  戻り値：-

  発生イベント： -

- **pokeItem: aString value: aValue**

  サーバーに、aString で表される名前の項目に対する更新値 aValue を送る。

  戻り値：-

  発生イベント： サーバーの pokeItem:value: イベント

- **pokeItem: aValue**

  サーバーに、メッセージ名と同じ名前の項目に対する更新値 aValue を送る。実際にはメッセージ名の pokeItem のところに項目名を入れて使う。

  戻り値：-

  発生イベント： サーバーの pokeItem: イベント

- **requestItem: aString**

  サーバーから、aString で表される名前の項目の値を取ってくる。

  戻り値：a Value

  発生イベント： -

- **terminate**

  DDE の対話を終了する。

  戻り値：-

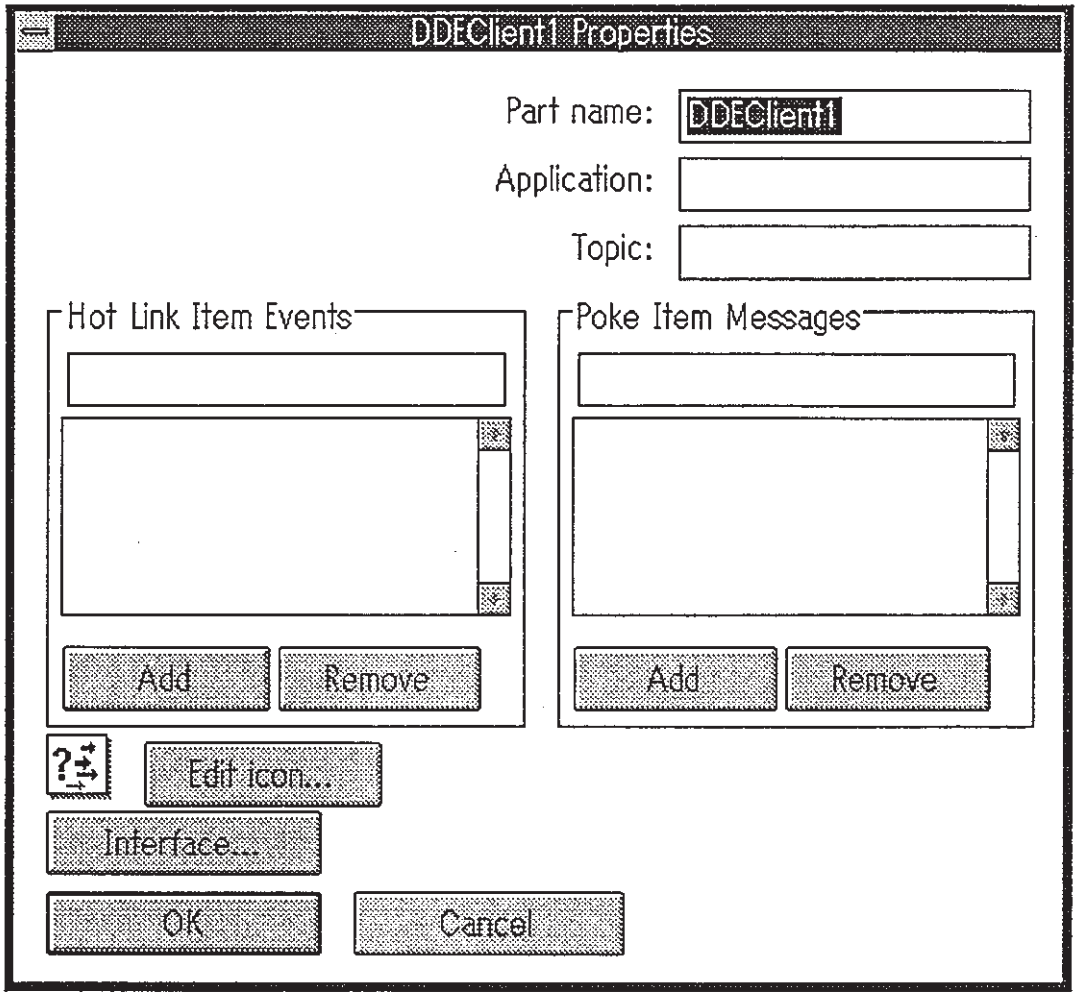
  発生イベント： -

④ 直接編集 (Direct-Edit)

DDE クライアントの直接編集は、作業台 (Workbench) 上のアイコンの下に現われる、部品の名前の編集になる。

⑤ 属性編集 (Property Edit)

この部品の属性ダイアログに特有なコントロールは以下の通りである。



- **Application（アプリケーション名）**

  サーバーのアプリケーション名を指定する。指定しないと DDE クライアント部品は使えない。

- **Topic（トピック名）**

  DDE 対話のトピック名を指定する。指定しないと DDE クライアント部品は使えない。

- **Hot link item events（ホットリンク項目イベント）**

  ホットリンク項目イベントは、サーバーからのホットリンク項目メッセージによって発生する。ホットリンク項目イベントを定義すると、イベント値一個を持つそれと同じ名前のイベントが、DDE クライアント部品に追加される。サーバーがホットリンク項目メッセージのどれかを実行すると、同じ名前を持つイベントがクライアントで発生し、ホットリンク項目メッセージの引数がイベント値として与えられる。

  ホットリンク項目イベントを新たに追加するには：
  1. 名前を入力フィールドに入力する。
  2. [Add] ボタンを押す。

  ホットリンク項目イベントを削除するには：
  1. リストの中から選択する。
  2. [Remove] ボタンを押す。

- **Poke item messages（ポーク項目メッセージ）**

  ポーク項目メッセージはサーバーに命令やデータを送るために使う。属性ダイアログでポーク項目メッセージを定義すると、引数一個を持つそれと同じ名前のメッセージが、DDE クライアント部品に追加される。サーバーに項目の更新値を送るには、項目と同じ名前のポーク項目メッセージに新しい値を引数として与えて実行する。

  ポーク項目メッセージを新たに追加するには：
  1. 名前を入力フィールドに入力する。
  2. [Add] ボタンを押す。

  ポーク項目メッセージを削除するには：
  1. リストの中から選択する。
  2. [Remove] ボタンを押す。

⑥ **関連部品**

- **DDE Server（DDE サーバー部品）DDEServer**

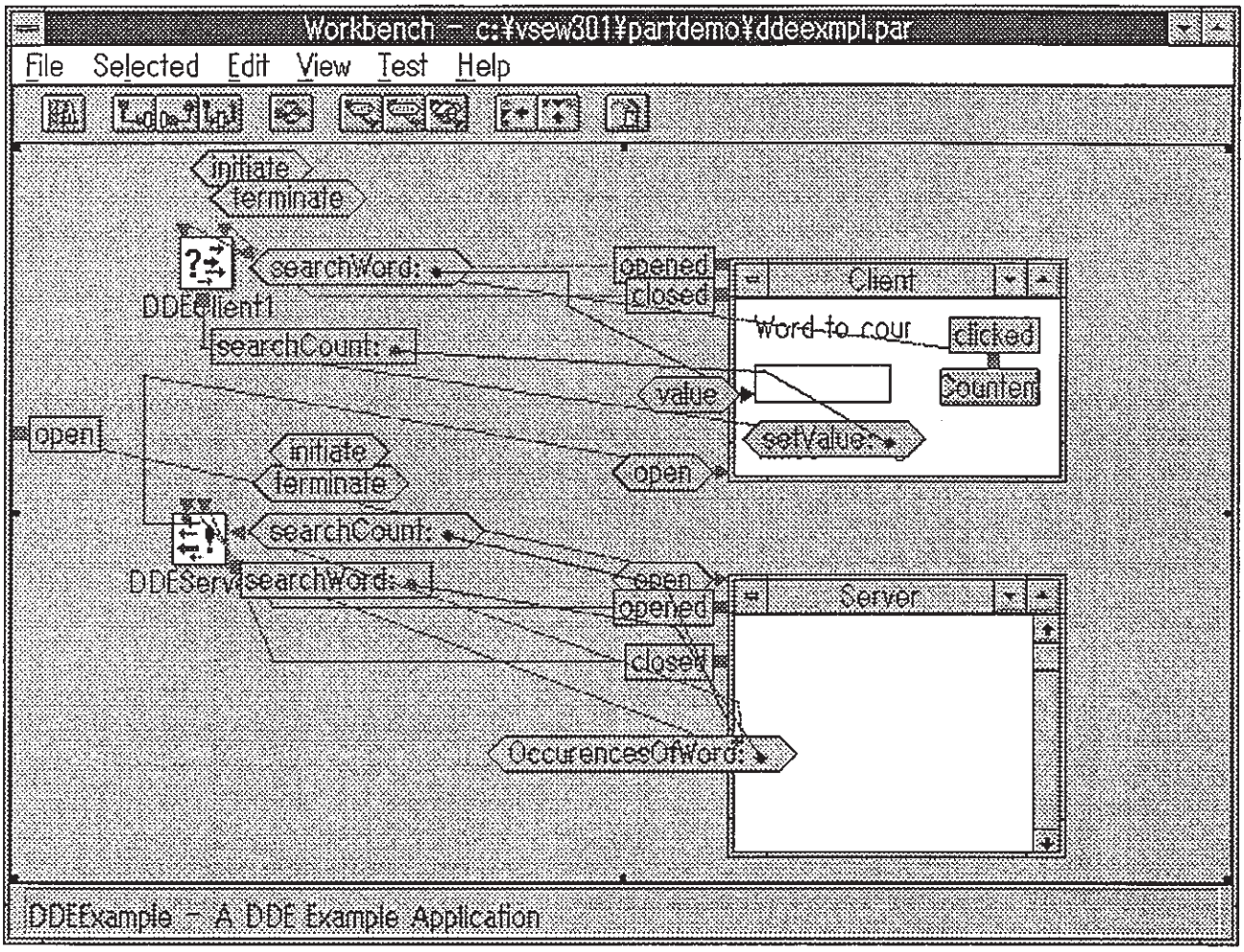
  PARTS ワークベンチのアプリケーションを DDE サーバーとして使うための部品である。

⑦ **備考**

クライアントが要求を出す前に、DDE 対話が開始されている必要がある。

ポーク項目やホットリンク項目を登録すると、対応して生成されるメッセージやイベントの名前の後ろには、引数ないしイベント値が来ることを示すコロン (:) が付く。このため、項目名にコロンを入れると、引数やイベント値の数が多く解釈されてしまうので、入れないように注意する必要がある。

⑧ 使用例



この部品の使用例は、PARTS ワークベンチをインストールしたディレクトリのサブディレクトリ PARTDEMO の中に、DDEXMPL.PAR のファイル名で収納されている。

この使用例では二つのウィンドウがそれぞれ模擬アプリケーションとして動き、互いに DDE を介して通信するようになっている。サーバーとクライアントが一つのアプリケーションの中にまとめられているのは、この二つが両方ともオープンされ、正しい順序（後述）で開始されて、デモが間違いなく行なえるように保証するためである。通常は、サーバーとクライアントは全く別のアプリケーションになる。

サーバーの方のウィンドウにはテキスト枠が一つ置いてあり、ワード プロセッサとしての役割をシミュレートしている。もう一つのウィンドウの方はクライアントになる。クライアントのウィンドウでは、ユーザーが入力フィールドに単語を入れて、[Count] ボタンを押す。そうすると、サーバーのウィンドウの中にその単語がいくつ含まれているかを、DDE を介してサーバーに質問が送られる。サーバーは答えを計算してクライアントに送り返し、その結果クライアントのウィンドウが更新されて答えを表示する。

サーバーは、通信に用いる自分の名前とトピック名をオペレーティング システムに登録するため、サーバーにそれらの情報を与えてやらなければならない。DDE クライアントの方でも、同じ名前とトピック名をオペレーティング システムに登録して、どのアプリケーションと通信するのかを宣言する必要がある。これらの名前はどんなものでも構わない。必要なのは、サーバーとクライアントの用いる名前とトピック名が同じになるということである。外部アプリケーション（PARTS でないアプリケーション）と通信するアプリケーションを構築するのであれば、その外部アプリケーションが使っている名前とトピック名を用いて、構築中のサーバーなりクライアントなりを設定しなければならない。アプリケーション名とトピック名は、DDE クライアント、サーバーどちらの場合とも、属性ダイアログで設定する。

使用例のアプリケーションでは、DDE クライアントには searchWord: というポーク項目メッセージがあり、これに対応して DDE サーバーの方には searchWord: というポーク項目イベントがある。またサーバーにあるホットリンク項目メッセージ searchCount: に対応して、クライアントには searchCount: というホットリンク項目イベントがある。（外部 DDE クライアントまたはサーバーと通信する PARTS ワークベンチ アプリケーションには、DDE 部品にその外部アプリケーションの DDE ポーク項目およびホットリンク項目の名前を設定して使う。）

アプリケーションが起動されると、以下の初期化が行なわれる。

- アプリケーションの open イベントによってサーバー ウィンドウが開く。
- サーバー ウィンドウの opened イベントによって DDEServer1 の initiate メッセージが実行される。
- その結果リンクによってクライアント ウィンドウの open メッセージが実行される。
- クライアント ウィンドウ opened イベントによって DDEClient1 の initiate メッセージが実行される。

注意： DDE サーバーは、DDE クライアントが接続を試みる前に起動され、オペレーティングシステムに登録されている必要がある。使用例で DDEServer1 の initiate メッセージの間接的な結果として DDEClient1 の initiate メッセージが実行されるようになっているのは、このことを保証するためである。

この時点で両方のウィンドウが開いており、DDEServer1 は（自分の initiate メッセージによって）オペレーティング システムへの登録を済ませて、DDEClient1 は（自分の initiate メッセージによって）DDEServer1 への接続を済ませています。次はユーザーがサーバー ウィンドウのテキスト枠に何かをタイプして、それからクライアント ウィンドウの入力フィールドに単語を入れて [Count] ボタンを押します。[Count] ボタンが押されると、以下のことが起こります。

- ボタンの clicked イベントによって DDEClient1 の searchWord: メッセージが実行される。
- EntryField1 の value メッセージの戻り値が searchWord: メッセージの引数になる。
- searchWord: メッセージの実行によってオペレーティング システムの DDE アクションが引き起こされる。つまり、searchWord という名前の項目が、EntryField1 の内容と共にサーバーに送られる。

実行はサーバー側で継続される。

- DDEServer1 の searchWord: イベントによって、テキスト枠に追加した OccurrencesOfWord: という一行のスクリプトが実行される。このメッセージには戻り値があり、引数の単語がテキスト枠の内容中に現れた回数を返す。このスクリプトは、テキスト枠の Interface を開いて読むことができる。
- OccurrencesOfWord: の戻り値が結果リンクによって DDEServer1 の searchCount: メッセージに与えられる。

ここで再びオペレーティング システムが実行を引き継ぎ、データ交換の部分を行なう。そして PARTS ワークベンチの実行が以下のように続く。

- DDEClient1 の searchCount: イベントによって StaticText2 の setValue: メッセージが実行される。

これで、ユーザーが [Count] ボタンを押してから実行される 1 サイクルが終わる。

使用例中の他のリンクは、二つのウィンドウが閉じた時に両方の DDE 部品を終了させるためのものである。

7．3 DDE Server（DDEサーバー）

DDE サーバーは機能部品で、ダイナミック データ エクスチェンジ (DDE) を通して DDE クライアント アプリケーションにアクセスする。ダイナミック データ エクスチェンジとは、サーバーとクライアントの対話である。クライアントはサーバーのアプリケーションを作成するためのものである。クライアントにするのは、他の PARTS ワークベンチ アプリケーションでも、他の言語を用いて用意したアプリケーションでも、DDE クライアントとして動く商用アプリケーションでも、どれでも構わない。

① 操作

DDE サーバー部品の属性ダイアログで、クライアントがサーバー アプリケーションを呼ぶ時の名前と通信トピックを指定する。サーバーはクライアントが通信を試みる前に起動されている必要がある。

DDE サーバーはクライアントに命令やデータを送り、またクライアントから命令やデータを受け取ることができる。クライアントへの情報送達は、ホットリンク項目メッセージというものを送ることによって行なう。このメッセージを受け取るクライアントは、それによって発生するホットリンク項目イベントを備えている必要がある。

一方クライアントがサーバーに情報を送るには、ポーク項目メッセージを送り、受け取るサーバーでは、対応するポーク項目イベントが発生しなければならない。どちらの種類のメッセージも引数を取ることができ、同様にイベントは値を伝えることができる。これらのメッセージとイベントは、他の PARTS ワークベンチ部品で使われるメッセージとイベントに大変良く似ていますが、いくつか重要な違いがある。

ポーク項目メッセージとホットリンク項目メッセージは、アプリケーションからアプリケーションへ、PARTS ワークベンチではなくオペレーティング システムによって伝達される。アプリケーション間に目に見える形で明示的にリンクを張ることはしない。これは、PARTS ワークベンチのアプリケーションが、他の PARTS ワークベンチのアプリケーションのみならず、DDE 通信を行えるどんなアプリケーションとでも、通信し合えることを意味している。ポーク項目とホットリンク項目のメッセージとイベントはまた、他のメッセージやイベントと異なり、部品ごとに定義を行う。

ホットリンク項目メッセージとポーク項目イベントは、DDE サーバー部品の属性ダイアログで定義する。ホットリンク項目メッセージを一つ定義するごとに、それと同じ名前を持つ（通常の）メッセージが DDE サーバー部品に登録される。このメッセージは引数を一個取る。ポーク項目イベントに対しても、値を一個持つ新たなイベントがそれと同じ名前で DDE サーバーに加えられ

る。これらのイベントは、クライアントが対応するメッセージを実行したときに発生するようになる。

② イベント

DDE サーバー部品は、属性ダイアログで定義した各ポーク項目イベントと同じ名前のイベントを持つようになる。各イベントともイベント値を一個持つ。何かの項目がクライアントから送られてくると、対応するイベントが発生して、更新内容はイベント値として取り出すことができる。

- **executeCommand: commandString**

  クライアントが executeCommand: メッセージを実行すると発生する。サーバーで実行すべきコマンドがイベント値で与えられる。

  発生因子： DDE クライアントの executeCommand: メッセージ

- **pokeItem: itemString value: valueString**

  クライアントが pokeItem:value: メッセージを実行すると発生する。項目名とその値はクライアントで引数として与えられたものになる。

  発生因子： DDE クライアントの pokeItem:value: メッセージ

- **pokeItem: aValue**

  クライアントが pokeItem: メッセージのどれかを実行すると、このイベントが発生する。更新値が aValue で与えられる。イベント名の pokeItem の部分は、実際には対応する項目名になる。

  発生因子： DDE クライアントの pokeItem: メッセージ

③ メッセージ

サーバー部品は、属性ダイアログで定義した各ホットリンク項目と同じ名前のメッセージを持つようになる。各メッセージとも引数を一個持つ。サーバーで更新した項目の値をクライアントに送るには、その値を引数として、対応するホットリンク メッセージを実行する。

- **executionFailed**

  executeCommand: イベントの結果として実行すると、コマンドの実行に失敗したことがオペレーティング システムに通達される。

戻り値：-

発生イベント：　-

- **hotLink: aValue**

  クライアントの、同じ名前のホットリンク項目イベントを、aValue の値で発生させる。

  戻り値：-

  発生イベント：　クライアントの hotLink: イベント

- **initiate**

  クライアント アプリケーションからの DDE 要求が受け付けられるよう、オペレーティング システムにサーバーを登録する。

  戻り値：-

  発生イベント：　-

- **pokeFailed**

  ポーク オペレーションが失敗したことをオペレーティング システムに通達する。

  戻り値：-

  発生イベント：　-

- **terminate**

  全てのクライアントとの DDE 対話を終了する。

  戻り値：-

  発生イベント：　-

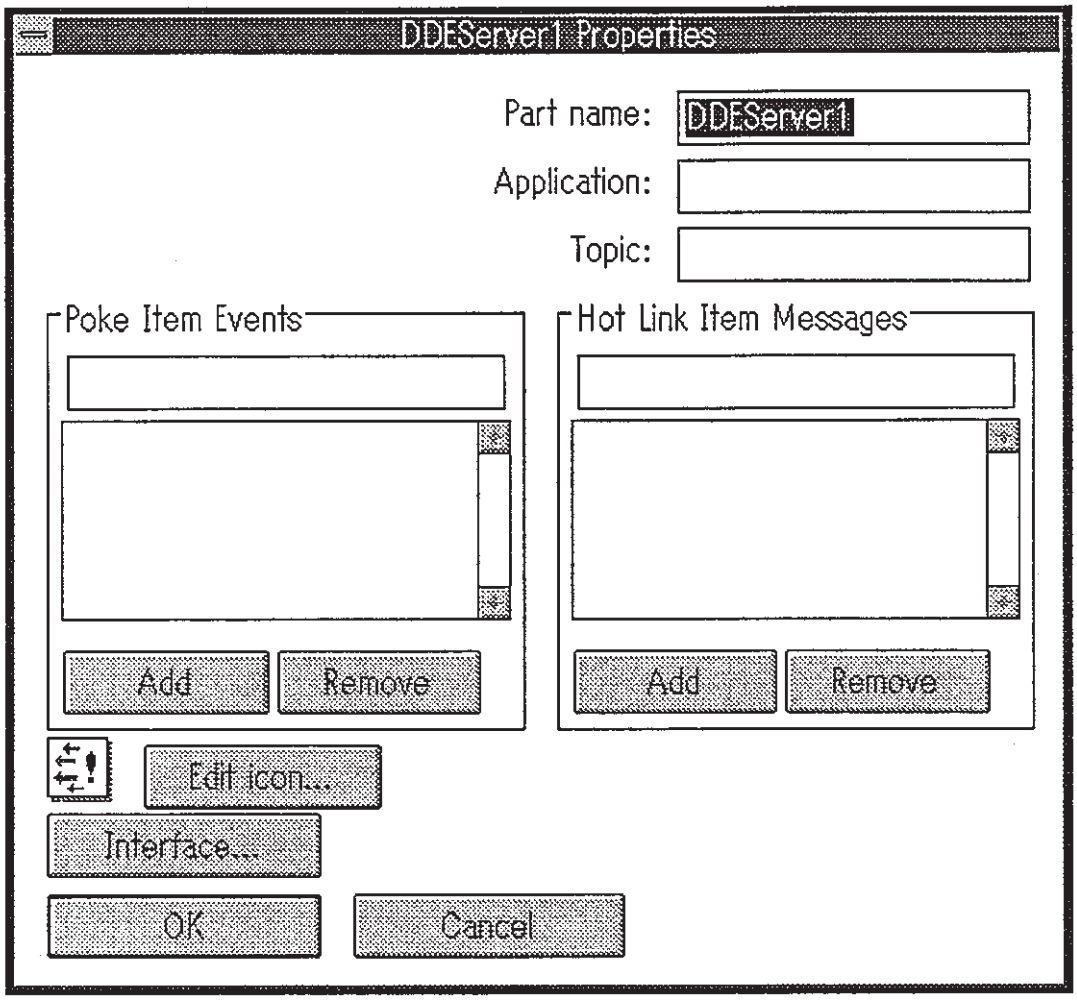
④　**直接編集 (Direct-Edit)**

DDE サーバーの直接編集は、作業台 (Workbench) 上のアイコンの下に現われる、部品の名前の

編集になる。

⑤ 属性編集 (Property Edit)

この部品に特有な属性は以下の通りである。



- **Application（アプリケーション名）**

  サーバーが属するアプリケーション名を指定する。指定しないと DDE サーバーは使えない。

- **Topic（トピック名）**

  サーバーが認識するトピック名を指定する。

- **Poke item events（ポーク項目イベント）**

  ポーク項目イベントは、対応するメッセージがクライアントによって実行されると発生する。

属性ダイアログでポーク項目イベントを定義すると、イベント値一個を持つそれと同じ名前のイベントが、DDE サーバー部品に追加される。

ポーク項目イベントを新たに追加するには：
1. 名前を入力フィールドに入力する。
2. [Add] ボタンを押す。

ポーク項目イベントを削除するには：
1 リストの中から選択する。
2. [Remove] ボタンを押す。

- **Hot link item messages（ホットリンク項目メッセージ）**

ホットリンク項目メッセージは、命令やデータをクライアントに送るために使う。属性ダイアログでホットリンク項目メッセージを定義すると、引数一個を持つそれと同じ名前のメッセージが、DDE サーバー部品に追加される。サーバーがそのメッセージを実行すると、同じ名前を持つイベントがクライアントで発生する。

ホットリンク項目を新たに追加するには：
1. 名前を入力フィールドに入力する。
2. [Add] ボタンを押す。

ホットリンク項目を削除するには：
1. リストの中から選択する。
2. [Remove] ボタンを押す。

⑥ 関連部品

- **DDE Client（DDE クライアント）DDEClient**

DDE クライアント部品は PARTS ワークベンチのアプリケーションを DDE クライアントとして使うための部品である。

⑦ 備考

ポーク項目やホットリンク項目を登録すると、対応して生成されるメッセージやイベントの名前の後ろには、引数ないしイベント値が来ることを示すコロン (:) が付く。このため、項目名にコロンを入れると、引数やイベント値の数が多く解釈されてしまうので、入れないように注意する必要がある。

クライアントが対話を開始する前に、DDE サーバーの開始とオペレーティング システムへの登録が（サーバー部品の initiate メッセージで）行なわれている必要がある。

⑧ 使用例

この「Reference」の 「DDE Client（DDE クライアント）DDEClient 」の項の使用例を参照。

7．4 DLL Accessor（DLLアクセス部品）

DLL アクセス部品は、ダイナミック リンク ライブラリ (DLL) の手続きへのアクセスを提供する機能部品である。ダイナミック リンクとは、アプリケーションの外部にある手続きや他の言語で書かれた手続きへアクセスする方法の一つである。

DLL の中の手続きにアクセスするために必要な情報は、属性ダイアログを用いて設定する。以下のような情報が必要である。

- 使用したい手続きを持っている DLL ファイルの名前
- アクセスしたい手続きの名前
- 手続きの呼び出し形式
- 手続きの引数の数と型
- 手続きの戻り値の型

これらの情報は、各 DLL のプログラマ レファレンスに載っている。

DLL ファイル中のアクセスする必要のある各手続きに対して、DLL アクセス部品に新しいメッセージを定義していく。メッセージ名には、DLL 手続きに渡す引数が一つ増えるごとに、コロン (:) を一つ加える。

DLL アクセス部品一個につき、一個の DLL ファイルへのアクセスが提供される。複数の DLL ファイルを扱う時には、同じ数だけ DLL アクセス部品を用意する。

DLL ファイル中の手続きにアクセスするには、まずそのファイルが開いている必要がある。これは open メッセージで行うこともできるし、手続きにアクセスする最初のメッセージが送られる時には、自動的に開かれる。DLL ファイルが何らかの理由で開けないと、openException: イベントが発生し、Windows のエラー コードがイベント値で与えられる。

手続き呼び出しが失敗すると、callException:arguments: イベントが発生する。これは、DLL ファイル中に存在しない手続きにアクセスした場合でも、引数の型が間違っていた場合でも、どんな場合でも同じである。

DLL ファイルを使用し終わったら、close メッセージを用いてファイルを閉じておくのが良いであろう。そうすれば、ライブラリの占めていたメモリ空間他、システムのリソースが解放されることになる。

① 操作

DLL ファイルの中のアクセスしたい手続き一つ一つに対し、対応するアクセス メッセージを DLL アクセス部品に定義する。このメッセージ名には、DLL 手続きの取る引数一つごとにコロン (:) が入る。

② イベント

- **openException: anInteger**

  DLL ファイルを開くのに失敗すると発生する。イベント値の anInteger には Windows のエラー コードが入る。

  発生因子： open

- **callException: aString arguments: aCollectionOfValues**

  DLL ファイル開くのに失敗する以外の理由で DLL 手続きの呼び出しが失敗すると、発生する。失敗したメッセージ名と失敗の説明が aString で与えられ、メッセージの引数が aCollectionOfValues で得られる。

  発生因子： procedureName

③ メッセージ

- **close**

  DLL ファイルを閉じる。

  戻り値：—

  発生イベント： —

- **open**

  DLL 手続きが使えるよう DLL ファイルを開く。

  戻り値：—

  発生イベント： openException:

- **procedureName**

  procedureName に対応させて属性ダイアログで定義した DLL 手続きを呼び出す。open メッセージがまだ実行されていなかった場合は、まず DLL ファイルを開いてから呼び出しを行なう。

  戻り値：属性ダイアログで定義された戻り値

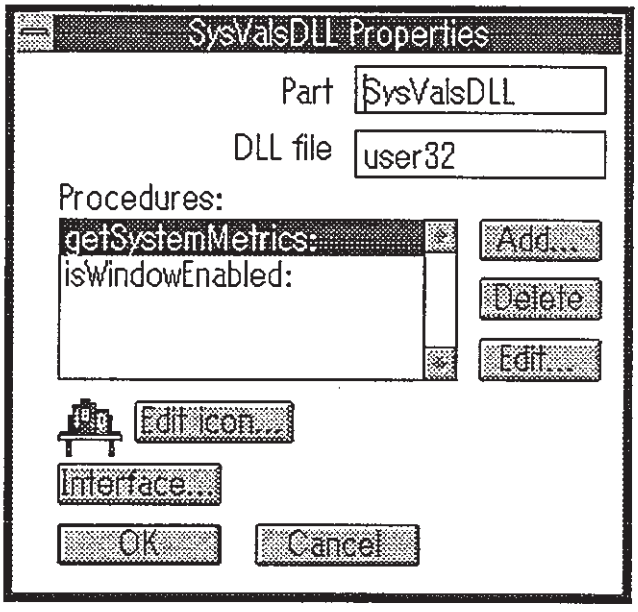
  発生イベント： callException:arguments:

④ **直接編集 (Direct-Edit)**

DLL アクセス部品の直接編集は、作業台 (Workbench) 上のアイコンの下に現われる、部品の名前の編集になる。

⑤ **属性編集 (Property Edit)**

DLL ファイルとのインターフェイスの定義は、属性ダイアログを使って、DLL の手続きに対応するメッセージを定義することによって行なう。この部品の属性ダイアログに特有なコントロールを以下に記述する。



- **DLL filename（DLL ファイル名）**

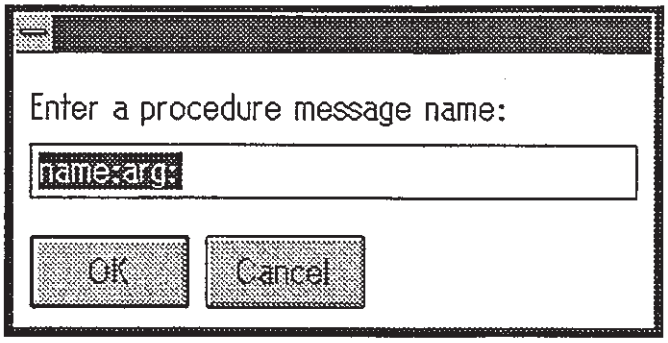
  このフィールドに、使用する DLL のファイル名（ディレクトリ パスを含まない）を入れる。DLL ファイルは PATH 環境変数に設定してあるディレクトリに置いてある必要がある。

- **Procedures（手続き）**

  ここに、現在定義されている DLL アクセス メッセージが列挙される。

- **Add...（追加...）**

  押すと、DLL アクセス部品にメッセージを追加するためのダイアログが開く。



  ここで DLL の個々の手続きにアクセスするメッセージの名前を入れる。DLL 手続きのパラメーター個につきコロン (:) を一個、対応するメッセージ名に加える。

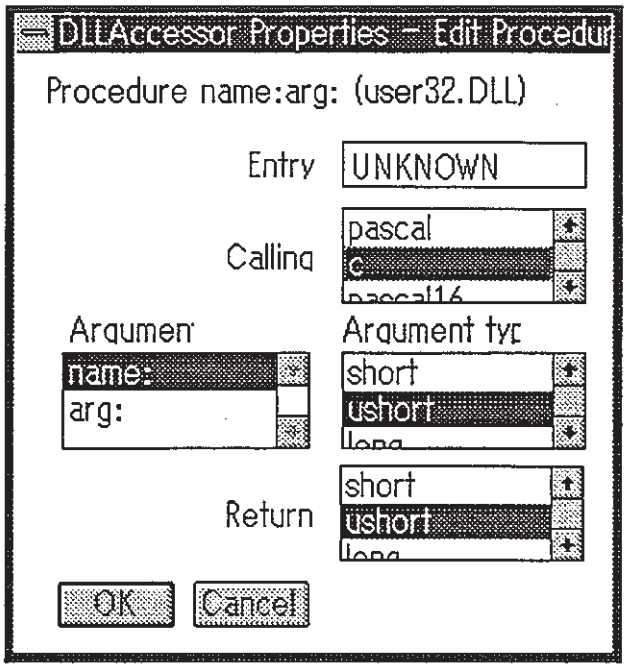
- **Delete（削除）**

  選択されたメッセージをリストから削除する。

- **Edit...（編集...）**

  選択されているメッセージに対して、以下の情報を指定するためのダイアログが開く。

  - 対応する DLL 手続きのエントリー ポイント
  - 使用する呼び出し形式
  - 手続きの各引数の型
  - 戻り値の型



- Entry point（エントリー ポイント）

  定義するメッセージが DLL アクセス部品によって実行された時に呼び出す手続きの、DLL ファイル中のエントリー ポイントを入れる。手続きのエントリー ポイント名、呼び出し形式、引数の型などは、使用する DLL ファイルのドキュメントに記述されているはずである。

- Calling convention（呼び出し形式）

  エントリー ポイントによって用いられる呼び出し形式を選ぶ。16 ビット C、16 ビット Pascal、32 ビット C、32 ビット Pascal がサポートされている。

- Arguments（引数）

  各引数の型を、リストの中から選ぶ。引数の名前は、メッセージ名の最初のコロンまでのテキストが第 1 引数の名前、次のコロンまでのテキストが第 2 引数の名前、というように決まる。作業台 (Workbench) 上でこのメッセージにリンクを張ると、引数一個に対して引数リンク一本が対応する。

- Argument type（引数の型）

  各引数には、次の内どれかの型を指定しなければならない。

  short　16 ビットの符号付き整数

  対応するイベントとメッセージには整数の引数を用いる。

  ushort　16 ビットの符号無し整数

  対応するイベントとメッセージには整数の引数を用いる。

  long　32 ビットの符号付き整数

  対応するイベントとメッセージには整数の引数を用いる。

ulong　32 ビットの符号無し整数

対応するイベントとメッセージには整数の引数を用いる。

boolean　8 ビットの値

対応するメッセージにはブーリアンの引数を用いる。

handle（ハンドル）　Windows のリソース ハンドル

対応するメッセージの引数に対しては、Pane 系の部品（Window、Group Pane など）からハンドルを得るためのスクリプトを用意する必要がある。

reference（参照）C 構造体へのポインタ

対応するメッセージの引数には、C 構造体アクセス部品の value メッセージの戻り値を使う。

文字列のパラメータを使う DLL 呼び出しが、その文字列に対する参照を返す場合には、文字列パラメータの供給にも C 構造体アクセス部品を使う必要がある。他の方法でこのパラメータを供給すると（例えば文字列保持部品で）、戻り値として nil が返ってきてしまう。

ただし例外があり、DLL が、返す値に自分自身のメモリを割り当てている場合には、参照型の手続き引数に文字列型の値を渡してやることができる。（C 構造体アクセス部品の参照型フィールドへは、C 構造体アクセス部品への参照でも文字列の値でも設定することができ、この場合とよく似ている。）文字列の値を持った参照型の引数は、DLL 手続き呼び出しが行なわれる時には、その文字列の中のキャラクタにヌルバイトを加えたものを指すポインタとして、手続きに渡される。返ってくる文字列には、この終端ヌルバイトは含まれない。どの場合にせよ、DLL で文字列パラメータが変更されるのであれば、その結果は DLL アクセス部品に渡した元の文字列にも反映される。

参照型が戻り値に適用される場合、なおかつ、戻り値が終端にヌルの入った文字列へのポインタである場合は、一旦文字列保持部品にそれを格納してから使う。これは、DLL 呼び出しメッセージから文字列保持部品の setFromReference: メッセージに結果リンクを張ることによって行なう。

参照値 (reference) の使い方の詳細は、「C Structure (C 構造体アクセス部品)」を参照。

double　float8 バイトの値

IEEE 倍精度浮動点小数に準拠。

- Return type（戻り値の型）

手続きの戻り値の型を、上と同様に選ぶ。void の型を持つ C の関数や Pacal の手続きに対しては none（なし）を選択する。

⑥ 関連部品

- C Structure（C 構造体アクセス部品）

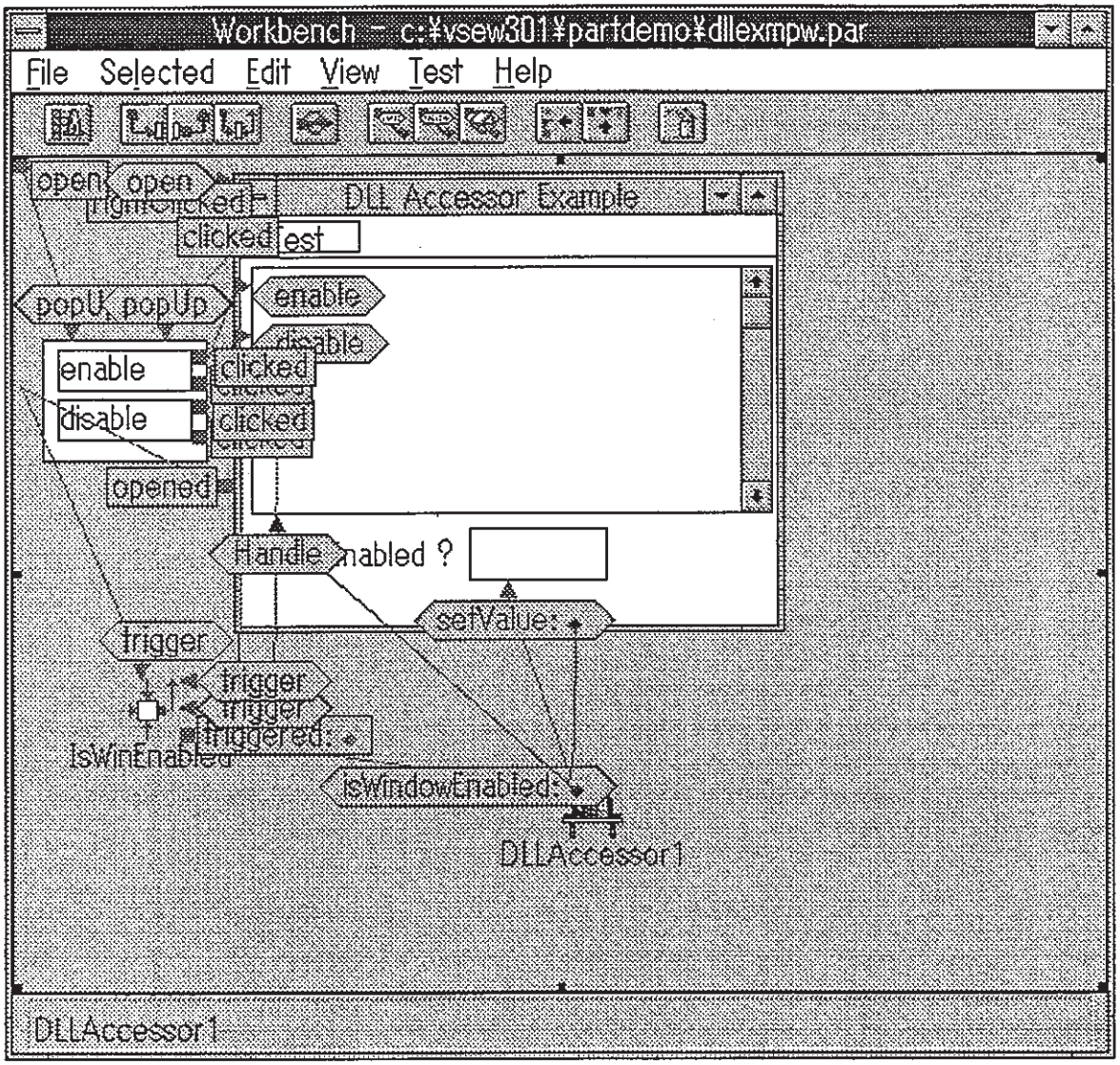
  C 構造体アクセス部品は、DLL 手続きの参照型引数として扱うことができるレコードの定義を行なう。

⑦ 備考

アクセスする DLL ファイルは、パス環境変数に指定されているディレクトリ内にある必要がある。

特殊なケースとして、DLL 呼び出しに文字列への参照が引数として渡され、同じ参照が戻り値として返る場合がある。このような場合、その文字列を表わすのに（文字列保持部品などでなく）C 構造体アクセス部品を用いる必要がある。ただし、DLL 呼び出しで自分自身の中に戻り値のメモリが割り当てられるのであれば、この制限はあてはまらない。

⑧ 使用例



この部品の使用例は、PARTS ワークベンチをインストールしたディレクトリのサブディレクトリ PARTDEMO の中に、DLLEXMPO.PAR のファイル名で収納されている。

この使用例は、DLL を通してシステム関数を呼び出す。メニュー Test には選択リストを活性化 (enable) および不活性化 (disable) するコマンドが入っている。コマンドがどちらか選ばれると、選択リストが活性化または不活性化され、システム関数 IsWindowEnabled が呼ばれる。この時選択リストのウィンドウ ハンドルが引数として渡され、返ってきたブーリアン値がウィンドウ下部の入力フィールドに表示される。なお、活性・不活性の状態が常に正しく表示されるように、このシステム関数は最初にウィンドウが開いたときにも呼ばれる。

メニュー [Test] はポップアップ メニューでもある。アプリケーション ウィンドウのどこででも右クリックを行なうと、カーソル位置にメニューが現れる。

7．5 C Structure（C構造体アクセス部品）

C 構造体アクセス部品は機能部品であり、C 構造体型の値（への参照）が引数として要求される C のダイナミック リンク ライブラリ（DLL）に手続き呼び出しを行なうために用いる。C 構造体アクセス部品の各フィールドは、DLL 内のフィールドに対応する。属性ダイアログを用いて構造体の各フィールドを宣言すると、C 構造体アクセス部品は自動的に、各フィールドの中身にアクセスするためのイベントとメッセージを生成する。

これらの自動生成されたイベントとメッセージの名前は、対応するフィールドと同じ名前が使われるが、以下の説明では、便宜上、field と field: という表記法を用いる。

① 操作

C 構造体アクセス部品を使う時には、部品の中のフィールドが、DLL の中と同じように配置されるよう、気をつける必要がある。C 構造体アクセス部品がフィールドを自動的にバイト境界やワード境界に合わせて整列したりすることはない。何故なら、DLL がどのコンパイラでどのようなオプションをもってコンパイルされるか、事前に知りようがなく、そのため、コンパイルされた DLL がどのように整列されているか、わからないからである。C 構造体アクセス部品中のフィールド配置が DLL の配置と異なっていると、エラーの原因となる。

16 ビット DLL を 32 ビット コンパイラで再コンパイルすると、このフィールド整列上の問題が起きる可能性がある。例えば以下のような C ヘッダー ファイルがあったとしよう。

```
typedef struct _cstruct {
    short   shortInt;
    char   *stringPointer;
    char   arrayOfChar[10];
    ulong   uLongInt;
} cstruct, *pcstruct;
```

これに対応する C 構造体アクセス部品の定義は、以下のようになります。

フィールド名	フィールド型	フィールド長
shortInt	unsigned short	2
stringPointer	reference	4
arrayOfChar	array of characters	10

```
uLongInt        unsigned long    4
```

この C ファイルが 16 ビット コンパイラで DLL にコンパイルされれば、C 構造体アクセス部品は正しく動く。これは、部品中の各フィールドは偶数バイトの長さを持ち、16 ビット コンパイラはフィールドを 2 バイトごとの境界に合わせて整列するからである。各フィールドの C 構造体アクセス部品内のオフセットは、DLL 内のフィールド オフセットと等しくなる。

ところが、同じ C ファイルを、データ構造のパック（後述）を行わずに 32 ビット コンパイラでコンパイルすると、コンパイラはパッディングを施して、フィールドを 4 バイトごとの境界に合わせて整列してしまう。C 構造体アクセス部品はフィールドの配置を自動的には整列しないため、これは正しくない配置になってしまう。

フィールド	C 構造体アクセス部品におけるオフセット	DLL におけるオフセット
shortInt0	0	
stringPointer	2	4
arrayOfChar	6	8
uLongInt	16	20

この例だと、最初のフィールド以外のフィールドへのアクセスは、C 構造体アクセス部品でのオフセットと実際のデータ オフセットが合っていないので、クラッシュを引き起こす可能性がある。

使用する C 構造体アクセス部品と DLL のオフセットが合っていない時には、以下のいずれかの方法で対処する。

- **DLL の再コンパイル**

  DLL が自分でコンパイルしたものであれば、コンパイルの仕方を変えることで、対処できる。大抵のコンパイラには、構造体をパックする、つまりパッディングを行わなずにコンパイルする方法が備わっている。これは普通は、コンパイル実行時のオプションか、または .H ファイル中の何らかのコンパイラ命令によって、指定することができる。例えば、#pragma pack 命令を受け付ける 32 ビット コンパイラを使って DLL のデータ構造をパックするには、以下のように構造体を定義する。

```
#pragma pack(1)      /* 強制的に 1 バイト境界整列にパック */

typedef struct _cstruct {
    short          shortInt;
```

```
    char        *stringPointer;
    char        arrayOfChar[10];
    ulong       uLongInt;
} cstruct, *pcstruct;

#pragma pack()      /* パックからデフォルトに戻す */
```

なお、普通デフォルトではデータ構造のパックは行われないので、パックする必要がある時には、コンパイラのドキュメントを読んで、どのように行えばよいのかを確かめる必要がある。

- **C 構造体アクセス部品にパッディングを施す**

DLL が再コンパイルできない、もしくはしたくない時には、PARTS ワークベンチの側からこの問題に対処することもできる。この場合、C 構造体アクセス部品のデータ フィールドが、コンパイラが DLL のデータ フィールドを整列したのと同じように配置されるよう、定義を行う。単にフィールドを並べ換えるだけで済むこともあるが、多くの場合、明示的にフィールドのパッディングを施してやる必要がある。上の例であれば、以下のように C 構造体アクセス部品を定義する。

フィールド名	フィールド型	フィールド長
shortInt	unsigned short	2
padding1	array of characters	2
stringPointer	reference	4
arrayOfChar	array of characters	10
padding2	array of characters	2
uLongInt	unsigned long	4

こうすると、各フィールドのオフセットは以下のようになります。

フィールド	C 構造体アクセス部品におけるオフセット	DLL におけるオフセット
shortInt	0	0
padding1	2	
stringPointer	4	4
arrayOfChar	8	8
padding2	18	
uLongInt	20	20

② イベント

各フィールドに対して、イベントが一つ自動生成される。イベント名はフィールド名と同じになり、フィールドの値をイベント値として提供するために、後ろにコロンが付く。

- **field: aValue**

  C 構造体アクセス部品への参照が、DLL アクセス部品のメッセージに引数として渡されると、そのメッセージが戻ってきた時に発生する。DLL の手続きで更新されるフィールドに対しては、このイベントからリンクを張って、更新後の値を設定するようにする。

  発生因子： DLL からの戻り（C 構造体アクセス部品にとっては外部要因）

③ メッセージ

C 構造体アクセス部品の各フィールドに対して、メッセージが二つ自動生成される。一つはフィールドに値を設定するためのものであり、もう一つはフィールドの値を取り出すためのものである。双方のメッセージとも、対応するフィールド名そのものを名前にしている。設定メッセージの方は、引数を要するために、最後にコロンが付く。

- **field**

  フィールドの現在値を返す。

  戻り値：a Value

  発生イベント： -

- **field: aValue**

  フィールドを aValue（何らかのオブジェクト）の値に設定する。

  戻り値：aValue

  発生イベント： -

- **free**

  C 構造体アクセス部品を用いて DLL アクセス部品に呼び出しが行なわれると、C 構造体アクセス部品のために広域メモリ領域が割り当てられる。このメモリ領域を使った後には、この

メッセージを C 構造体アクセス部品に送って、メモリ領域を解放しておくようにする。メモリ解放が明示的に行なわれなかった場合、その領域はイメージが落とされた時点で自動的に解放される。
イメージの中でアプリケーションを複数実行していて、その中のアプリケーションが C 構造体アクセス部品を使っている場合、そのアプリケーションは、上述のメモリ領域が解放されるまでイメージの中に残る。これは部品アクセス部品でロードした部品アプリケーションにも当てはまる。このアプリケーションも、たとえアンロードしたとしても、中の C 構造体アクセス部品を解放しない限り、メモリ上に残る。部品アクセス部品の場合には、そのaboutToUnload イベントを用いて C 構造体アクセス部品を解放することができる。

戻り値：-

発生イベント： 全ての field: イベント

- **setFromReference: aReference**

  DLL アクセス部品で C 構造体へのポインタを返すメッセージを使う場合には、このメッセージを使ってその C 構造体の中身を取り出す。DLL アクセス部品のメッセージから C 構造体アクセス部品の setFromReference: に結果リンクを張ることによって、DLL 呼び出しの度に、C 構造体アクセス部品の中身が更新され、フィールドのイベントも全て発生するようになる。

  戻り値：-

  発生イベント： 全ての field: イベント

- **value**

  C 構造体への参照を返す。この参照値は、DLL アクセス部品の、参照型の引数を取る DLL 手続きを呼び出すメッセージへの引数として使われる。

  戻り値：a reference

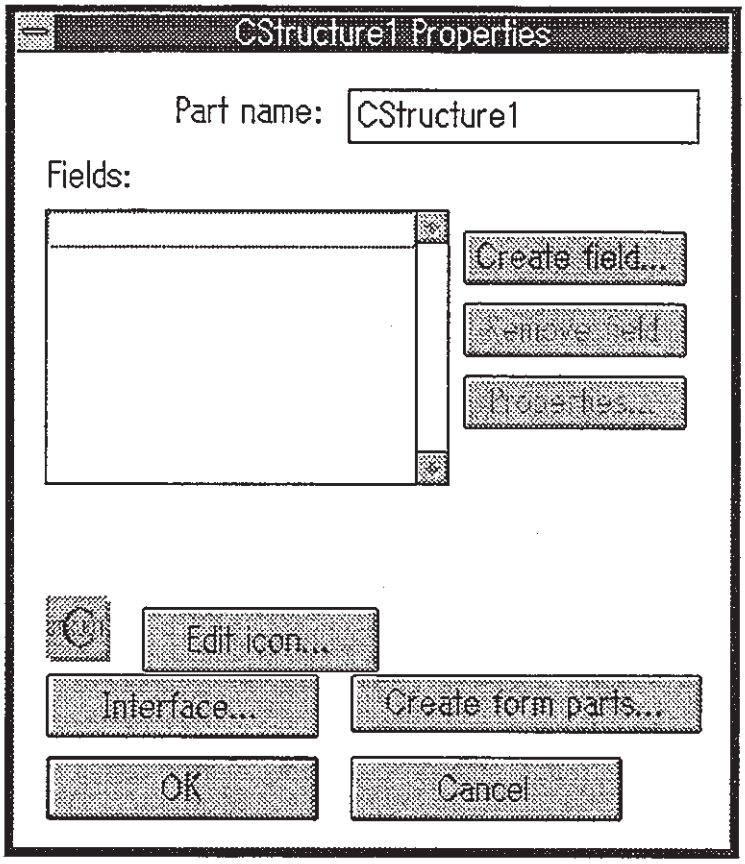
  発生イベント： -

④ **直接編集 (Direct-Edit)**

C 構造体アクセス部品の直接編集は、作業台 (Workbench) 上のアイコンの下に現われる、部品の名前の編集になる。

⑤ 属性編集 (Property Edit)

C 構造体アクセス部品に特有なコントロールは以下の通りである。

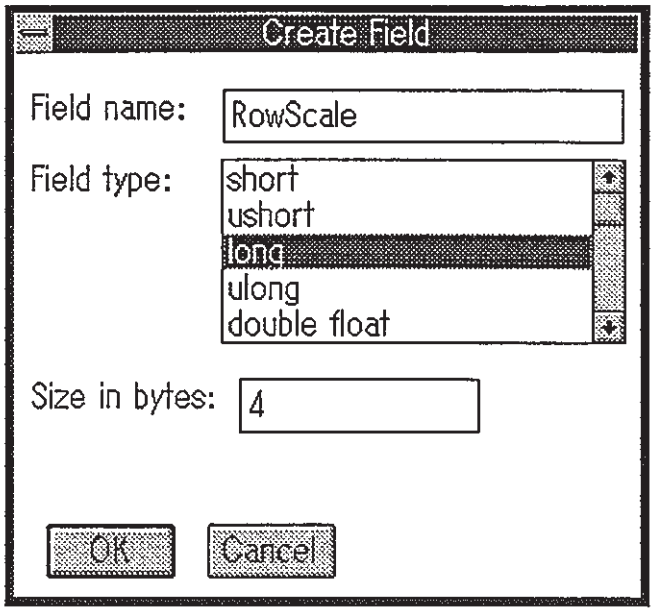


- **Fields（フィールド）**

  このリスト枠の中に、定義されているフィールドとその順序が表示される。

- **Create field...（フィールドの作成...）**

  このボタンによって、構造体を定義するための Create Field（フィールド作成）ダイアログが開く。このダイアログは、[Properties...（属性...）] ボタンで既に定義してあるフィールドの属性を変更するときにも使われる。



- **Field name（フィールド名）**

  ここには新たに作るフィールドの名前を入れる。

- **Field type（フィールドの型）**

  フィールドに対する C のデータ型を選択する。可能な型は以下の通り。

  short　　16 ビットの符号付き整数

  対応するイベントとメッセージには整数の引数を用いる。

  ushort　　16 ビットの符号無し整数

  対応するイベントとメッセージには整数の引数を用いる。

  long　　32 ビットの符号付き整数

  対応するイベントとメッセージには整数の引数を用いる。

  ulong　　32 ビットの符号無し整数

  対応するイベントとメッセージには整数の引数を用いる。

  double float　　8 バイトの値

  IEEE 倍精度浮動点小数に準拠。

  array of char　　固定長キャラクタ配列

  対応するイベントとメッセージには文字列の引数を用いる。配列のバイト長を Size in bytes フィールドに入れることが必要。

  reference　reference（参照）型フィールド

  他の C 構造体へのポインタやキャラクタ配列へのポインタを扱うときに使う。このフィールドには４バイト使用される。

  フィールド値が他の C 構造体へのポインタである場合には、対応する C 構造

体アクセス部品への value メッセージからこのフィールドへ結果リンクを張る。フィールド値がキャラクタ配列へのポインタである場合には、文字列保持部品への value メッセージからこのフィールドへ結果リンクを張る。後者の場合、メッセージを実行する前に、文字列保持部品の文字列の長さが適切であることを確認する必要がある。

- **Size in bytes（バイト サイズ）**

  このフィールドには、選択されたフィールド型のサイズがバイト長で表示される。上で array of char を選択した時には、[OK] ボタンを押す前にここにその配列のサイズを入力しておく必要がある。

- **Remove Field（フィールドの削除）**

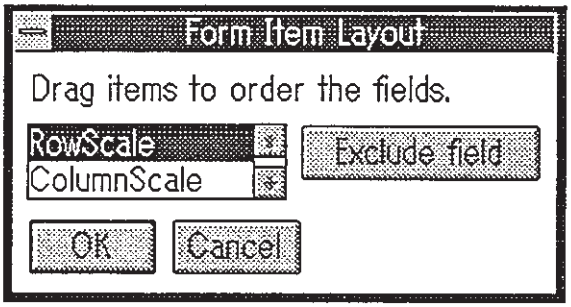
  このボタンで属性ダイアログの Fields リスト枠で選択されたフィールドを削除する。

- **Properties...（属性...）**

  このボタンによって、現在選択されているフィールドのフィールド属性ダイアログが開く。上述の「フィールドの作成... 」の項で説明した属性を変更するときに使う。

- **Create form parts...（フォーム部品の作成...）**

  C 構造体アクセス部品の各フィールドに対応させて、フィールド値の入出力用視覚部品を自動生成することができる。各フィールドのイベント、メッセージとも、これらの視覚部品の value および setValue: メッセージに自動的につなげられる。
  [Create from parts] のボタンを押すと、C 構造体のフィールドのリストが入ったフォーム項目レイアウト ダイアログ (Form Item Layout dialog) が表示される。



このリストから、視覚部品を割り当てたくないフィールドを選び、[Exclude filed（除外）] ボタンをクリックする。フィールドの除外が終わったら、[OK] ボタンをクリックする。ここで作業台ウィンドウが最前面に出て、カーソルの形が部品アイコンのようになる。このカーソルで、生成される部品を置きたいウィンドウをクリックすると、その中に視覚部品が配置され、

それらと C 構造体アクセス部品のフィールドとの間のリンクも自動生成される。

⑥ 関連部品

- DLL Accessor（DLL アクセス部品）DLLAccessor

  C 構造体アクセス部品は DLL アクセス部品と一緒に使う。特に、DLL アクセス部品で reference（参照）型として宣言されている引数のフィールドにアクセスする際に必要。

⑦ 備考

フィールド名を変更したい時には、そのフィールドを一旦削除して作り直す必要がある。また、新たなフィールドを C 構造体アクセス部品の既存のフィールドの間に入れることはできない。定義したフィールドは必ず C 構造体アクセス部品の最後に付け加えられる。

７．６ ＯＬＥによる連携処理

OLE による通信は、Visual Smalltalk の次期バージョン（3.1）からサポートされる機能となる。したがって、今回の連携処理機能の実現には利用できない。

8. ＯｂｊｅｃｔＢｒｏｋｅｒの連携処理機能

8．1 ＣＯＲＢＡ

Object Broker は OMG によって制定された CORBA 準拠のミドルウェアであり、Miscrosoft Windows や Apple Macintosh、各種 UNIX 等、幅広いプラットフォーム上で動作する。

① OMG（Object Management Group）とは

オブジェクト指向技術の普及と標準化のために 1989 年 4 月設立された非営利団体で、DEC、Hewlett-Packard、IBM、SunSoft など世界各国の主要な計算機のハードウェア／ソフトウェアのベンダを中心に、X/Open などの他の標準化団体と連携して、分散オブジェクト技術に関する標準化作業を行っている。

OMG における標準化は、会員企業に対して標準化案の提示を求め、応募された案を技術委員会で比較・検討した上で整合性のある最終案に一本化し、全会員企業による投票によって採否を決定するというものである。CORBA もこのような手続きを経て、採択された。

② ORB（Object Request Broker）とは

オブジェクトが処理要求を発行し、その応答を受け取ることができる透過的なメカニズムで、クライアントから ORB を介してオブジェクトインプリメンテーションに要求が送られる。

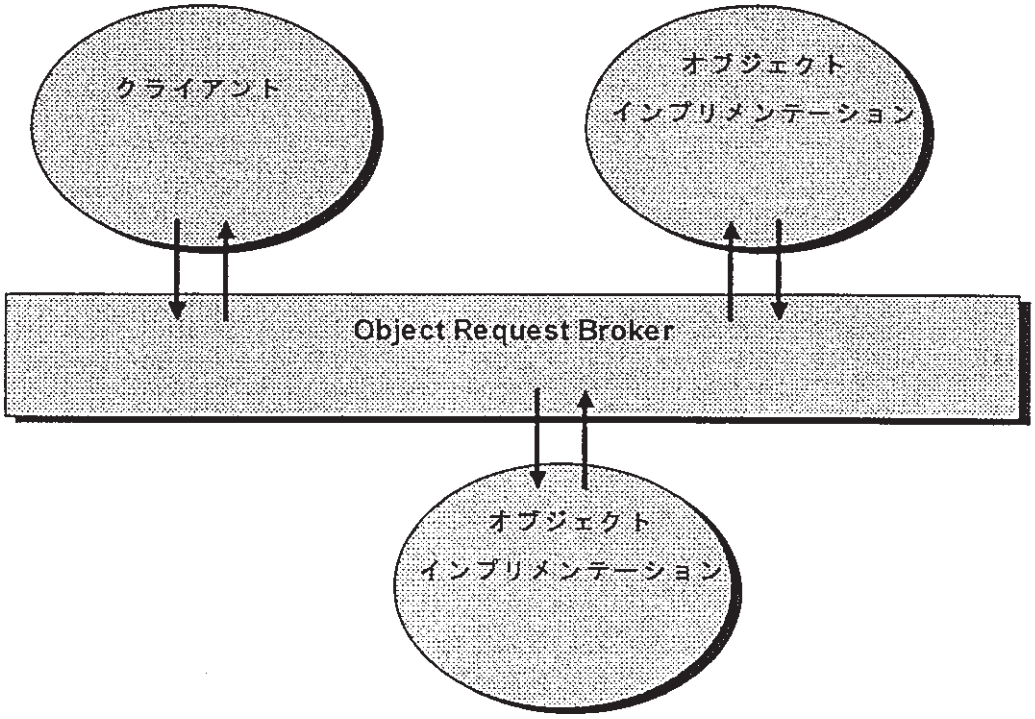


図 1. ORB 概念図

クライアントは、オブジェクトにおけるオペレーションの実行を要求する実体である。オブジェクトインプリメンテーションは、オブジェクトを実装するためのコードとデータである。ORB は、クライアントから送られた要求に対するオブジェクトインプリメンテーションを探し、オブジェクトインプリメンテーションに要求を受け取る準備をさせ、要求を構成するデータのやり取りに必要なすべての機構について責任を持つ。

クライアントにとって、オブジェクトインプリメンテーションがどこに存在するか、どのプログラム言語で実装されているか、などについては一切無関係である。

③ CORBA（Common Object Request Broker Architecture）とは

異機種分散環境における任意のアプリケーションがネットワーク上の資源を自由に使えることを目的として OMG が制定した規約である。分散オブジェクトモデルを規定し、前述の ORB を中心に分散オブジェクトの実現と管理を行うための枠組みである。

8．2 VisualSmalltalkとのインターフェース

CORBA の仕様の範囲内で、様々な種類の ORB の実装が存在する。Object Broker はその様な ORB の実装の１つであり、CORBA V1.2 に準拠している。

Windows における Object Broker のインタフェースには、DDE、OLE があるので、実現可能な Visual Smalltalk とのインタフェースとしては以下の２つが挙げられる。

- DDE によって直接 Visual Smalltalk と Object Broker が通信する。
- クライアントを DLL として作り、Visual Smalltalk はその DLL を介して Object Broker と通信する。

このうち、DDE による通信はオーバーヘッドも大きいので現実的ではない。

それに対して DLL を用いた方法であれば、Visual Smalltalk は Object Broker をまったく意識せず、あたかもローカルマシン上の DLL を呼び出すのと同じやり方で分散環境を利用できる。逆にワークステーションに計算を委譲するのではなく、ローカルマシン上で行う場合であっても、その計算ライブラリを同じインタフェースを持つ DLL として作る限り Visual Smalltalk 側には何の変更も必要ない。

以上のことから、Object Broker と Visual Smalltalk の連携には、ローカルマシン上に DLL によるクライアントを作成するのがよいと思われる。

９．結 言

本システムは、分散オブジェクト技術を基にして構築されたシステムである。

ここ数年クライアント／サーバー・コンピューティングが一種のトレンドとなっている。最近のコンピュータ・システムは従来のように１台の大型コンピュータを大勢で利用する集中形態から、一人または小人数で使用するコンピュータをネットワークで接続する分散形態へ変化してきている。このような分散形態のコンピュータ・システムに対応するためにソフトウェア・システムも複数のコンピュータ上のアプリケーションが相互に通信しあいながら協調して業務を遂行する形態、つまりクライアント／サーバーコンピューテを実現することが求められている。

こうした背景からリモート分散環境は、徐々にその必要性を高めている。

コンピュータ本体の機種及びＯＳの壁を超えて連携処理が実現出来るため、Ｉ／Ｆ処理は操作性の高いＰＣで、負荷の大きな計算処理はワークステーションでというようなシステムをデザインすることが可能になる。

原理動作の確認を目的とした本システムの次のシステムでは、「並列処理」を行う。

その際「同期制御」をしなければならない。

並列処理は、複数の処理（サブルーチン）を同時に実行する形態の手法であるが、次システムでは並列処理された結果データを蓄えておき横一線にして参照する場面が発生する。こうした場合勝手にそれぞれが処理を進めるのでは無く、全てのサブルーチンが終了するのを待って全結果データを参照しなければならない。ここで全てのサブルーチンを待たせる制御方法が「同期制御」である。

ＯｂｊｅｃｔＢｒｏｋｅｒは、同期制御を関数レベルでサポートしていないためプログラム言語の作り込みで同期制御を実現しなければならない。本システムではプログラム制御に関する動作確認もしているため参考にして次システムを構築することになる。

本システムを構築して動作させてみたところ、大量のデータ受け渡しにはかなりの時間が掛かることが分かった。実行する計算機のグレード及びデータの容量によって時間の増減はあるが、ＯｂｊｅｃｔＢｒｏｋｅｒプログラムの処理時間＋通信により発生する転送時間は見逃せないものがある。

本システムでは、（１）直接データを受け渡す方法で実行したが、それ以外の方法として（２）データベースを介してデータを受け渡す方法がある。

この方法はクイックレスポンスが期待出来るが、相応の初期投資＋ＤＢ構築作業を必要とする。それぞれメリット／デメリットはあるが一般的に以下の項目を検討してどちらにするか判断することになる。

- データの容量
- レスポンス・タイム
- 拡張性
- 開発予算

謝　辞

本システムの構築を進めるにあたり 日本デジタルイクイップメント(株) 久保田氏、高祖氏、松本氏および(株)ＳＲＡ 西中氏には有益なご意見並びに多大なる技術的支援を頂きました。
ここに感謝の意を表します。

参 考 文 献

（1） アスキー出版局，‘オブジェクト指向入門’ (1991)

（2） 日本デジタルイクイップメント（株），
‘ＯｂｊｅｃｔＢｒｏｋｅｒＶ２．５ for ＳｕｎＯＳ’ (1995)

（3） 日本デジタルイクイップメント（株），
‘ＯｂｊｅｃｔＢｒｏｋｅｒＶ２．５ for ＷｉｎｄｏｗｓＮＴ’
(1995)

（4） Ｄｉｇｉｔａｌｋ，
『ＶｉｓｕａｌＳｍａｌｌｔａｌｋＥｎｔｅｒｐｒｉｓｅＶ３．
０．３』 (1995)

（5） Ｄｉｇｉｔａｌｋ，
‘３２ＢｉｔＯｂｊｅｃｔＯｒｉｅｎｔｅｄＰｒｏｇｒａｍｉｎｇ
ＳｙｓｔｅｍｆｏｒＷｉｎ３２’ (1995)

（6） Ｄｉｇｉｔａｌｋ，
‘ＰＡＲＴＳＷｏｒｋＢｅｎｃｈ Ｓｃｒｉｐｔ Ｌａｎｇｕａｇｅ
Ｇｕｉｄｅ’ (1995)

（7） 日本デジタルイクイップメント（株），
‘分散オブジェクト環境の構築とアプリケーション統合’ (1995)

（8） 日本デジタルイクイップメント（株），
‘ＯｂｊｅｃｔＢｒｏｋｅｒ入門’ (1995)

（9） 日本デジタルイクイップメント（株），
‘分散オブジェクト環境の統合’ (1995)

（１０） 日本デジタルイクイップメント（株），
‘ＦＢＥ概要’ (1995)

（１１） アスキー出版局，‘ＩｎｓｉｄｅＷｉｎｄｏｗｓＮＴ’ (1993)

付録A　ソースコード Ｎｏ．１ ／ クライアント側プログラム（Ｃ＋＋言語）

MAIN.CPP

```
#include <windows.h>
BOOL WINAPI DllMain (HINSTANCE hDLL, DWORD dwReason, LPVOID lpReserved)
{
return TRUE;
}
```

ASYNC. CPP

```
#include "scopd.h"
#include <stdio.h>
#include <stdlib.h>
#include <float.h>
#include <values.h>


/* for dynamic invokation */
static  CORBA_Context ctx;
static  CORBA_NVList arglist;
static  CORBA_Request request;


int  Finished;

/* function prototypes */
int InitObject();
int   ReportExcep();
void TestExec();
void cleanup();

void SendExecMessage(
    CORBA_Object COPObj,
    CORBA_Environment *ev,
    CORBA_string instring,
    CORBA_short inshort,
    CORBA_double indouble,
    SCOPD_DoubleSeq * indseq,
    SCOPD_DataStr * indstr,
    CORBA_string * outstring,
    CORBA_short * outshort,
    CORBA_double * outdouble,
    SCOPD_DoubleSeq * outdseq,
    SCOPD_DataStr * outdstr);


char gsum[512];
```

```
/* The main routine */
main ()
{
  CORBA_Environment Ev;
  int Finished = FALSE;
  int status;
  char Input[512];
  int Option;
  CORBA_Object COPObj;

  gsum[0]='¥0';

  status = InitObject(&Ev, &COPObj);
  if (status==TRUE)
  {
    {
      while (Finished==FALSE)
      {
        printf("¥nSelect a Transaction:¥n");
        printf("  1 - Execution. ¥n");
        printf("  5 - Exit¥n");
        printf("Transaction: ");
        gets(Input);
        Option = atoi(Input);
        switch (Option)
        {
           case  1: TestExec(&Ev, &COPObj); break;
           case  5: Finished=TRUE; break;
           default: break;
        }
      }
    }
  }


  CORBA_Object_release(COPObj,
                       &Ev);
  if (Ev._major!=CORBA_NO_EXCEPTION)
    ReportExcep(&Ev);

  OBB_ORB_rundown(CORBA_DEC_ORB_OBJECT,
                  &Ev,
                  (CORBA_Flags)NULL);
  if (Ev._major!=CORBA_NO_EXCEPTION)
    ReportExcep(&Ev);
}  /* end of routine main */

int InitObject(
  CORBA_Environment *ev,
  CORBA_Object *COPObj)
{
  FILE *fp;
  char FileRec[1024];

  fp = fopen("cop_obj.ref","r");
  if (fp == NULL){
    printf("Could not find cop_obj.ref.¥n");
    exit(0);
  }
  fgets(FileRec, 1024, fp);
  FileRec[strlen(FileRec)-1] = '¥0';
  *COPObj = CORBA_ORB_string_to_object(
```

```
                CORBA_DEC_ORB_OBJECT,
                ev,
                FileRec);
  if (ev->_major != CORBA_NO_EXCEPTION) ReportExcep(ev);
  return TRUE;
}


int ReportExcep(
  CORBA_Environment *Ev)  /* in,  CORBA environment structure */
{
  int status;
  SCOPD_UserExcep *UserExcep;

  if (Ev->_major==CORBA_SYSTEM_EXCEPTION)
  {
    printf("¥nSystem exception encountered: %s¥n",
           (char *)CORBA_exception_id(Ev));
    printf("  %s¥n¥n¥n",
           (char *)OBB_exception_errortext(Ev,(CORBA_Flags)OBB_FORMAT_80));
    status = FALSE; /* flag a system exception */
  }
  else
  {
    UserExcep = (SCOPD_UserExcep *)CORBA_exception_value(Ev);
    switch (UserExcep->Reason)
    {
       case SCOPD_ErrCopExec:
         printf("¥nCOP command Execution Error.¥n");
         break;
       case SCOPD_ErrSumyExec:
         printf("¥nSUMY command Execution Error.¥n");
         break;
       case SCOPD_ErrCopRunning:
         printf("¥nCOP command is running now.¥n");
         break;
       case SCOPD_ErrGetReslt:
         printf("¥nGet Result error.¥n");
         break;
       default:
         printf("¥nAn unrecognized problem has occurred; problem code %u.¥n",
                UserExcep->Reason);
         break;
    }
    status = TRUE; /* not a system exception */
  }
  CORBA_exception_free(Ev);
  return(status);
}  /* end of routine ReportExcep */

void TestExec(CORBA_Environment *ev,CORBA_Object *COPObj)
{
  CORBA_double indouble;
  CORBA_double outdouble;
  CORBA_short  inshort;
  CORBA_short  outshort;
  CORBA_long   inlong;
  CORBA_long   outlong;
  CORBA_string instring;
  CORBA_string outstring;
  SCOPD_DataStr instruct;
  SCOPD_DataStr outstruct;
  SCOPD_DoubleSeq indseq;
```

```
  SCOPD_DoubleSeq outdseq;

  unsigned long i;

  /* Creating String Data */
  printf("Creating String Data ...¥n ");
  instring = (char *)malloc(11);
  strcpy(instring, "abcdefghij");

  /* Creating Double Data */
  printf("Creating Double Data ...¥n ");
  indouble = DBL_MAX;

  /* Creating short Data */
  printf("Creating Short Data ...¥n ");
  inshort = MAXSHORT;

  /* Creating Structure Data */
  printf("Creating Structure Data ...¥n ");
  instruct.stringdata = malloc(11);
  strcpy(instruct.stringdata, "abcdefghij");
  instruct.shortdata = MAXSHORT;
  instruct.doubledata = DBL_MAX;
  instruct.dseqdata._length = 10;
  instruct.dseqdata._maximum = 10;
  instruct.dseqdata._buffer = malloc(sizeof(CORBA_double)*10);
  for(i=0; i<10; i++){
    instruct.dseqdata._buffer[i] = DBL_MAX;
  }

  /* Creating Array Data */
  printf("Creating Double Seqence Data ... ¥n");
  indseq._length = 1024*1024;
  indseq._maximum = 1024*1024;
  indseq._buffer=(double *)malloc(sizeof(CORBA_double)*1024*1024);
  if(indseq._buffer == NULL){
    fprintf(stderr,"sequence data malloc error.¥n");
    exit(0);
  }
  for(i=0; i<1024*1024; i++){
    indseq._buffer[i]=DBL_MAX;
  }

/* Executing COP  */
printf("¥n¥n COP command Executing now ....¥n");
/*
SCOPD_COP_Exec (*COPObj, ev,
   instring,
   inshort,
   indouble,
   &indseq,
   &instruct,
   &outstring,
   &outshort,
   &outdouble,
   &outdseq,
   &outstruct);
 */
SendExecMessage (*COPObj, ev,
   instring,
   inshort,
   indouble,
   &indseq,
```

```
        &instruct,
        &outstring,
        &outshort,
        &outdouble,
        &outdseq,
        &outstruct);

      if(ev->_major == CORBA_NO_EXCEPTION){

        while(PollMessage()==FALSE){
          printf("Polling messages now.¥n");
          sleep(1);
        }

        printf("¥n¥n ### Normal Data. ####¥n");
        printf("out double = %e¥n",outdouble);
        printf("out string = %s¥n",outstring);
        printf("out short = %d¥n",outshort);

        printf("¥n ### Struct Data ###.¥n");
        printf("stringdata = %s¥n",outstruct.stringdata);
        printf(" shortdata = %d¥n",outstruct.shortdata);
        printf("doubledata = %e¥n",outstruct.doubledata);
        for(i=0;i<outstruct.dseqdata._length;i++)
          printf("douleseq[%d] = %e¥n",i,outstruct.dseqdata._buffer[i]);

        printf("¥n ### Sequence Data ###.¥n");
        printf("First Sequence = %e¥n",outdseq._buffer[0]);
        printf("Last  Sequence = %e¥n",outdseq._buffer[outdseq._length-1]);
        printf("Sequence Num = %d¥n",outdseq._length);

      }


      /* free routine */
      CORBA_free(outstring);
      CORBA_free(outdseq._buffer);
      CORBA_free(outstruct.stringdata);
      CORBA_free(outstruct.dseqdata._buffer);

}



/*****************************************************************/
/*
  This is dynamic invokation routine for async invokation.
*/

void SendExecMessage(
    CORBA_Object COPObj,
    CORBA_Environment *ev,
    CORBA_string instring,
    CORBA_short inshort,
    CORBA_double indouble,
    SCOPD_DoubleSeq * indseq,
    SCOPD_DataStr * indstr,
    CORBA_string * outstring,
    CORBA_short * outshort,
    CORBA_double * outdouble,
    SCOPD_DoubleSeq * outdseq,
    SCOPD_DataStr * outdstr)
```

```
{

  /* Get default context */
  CORBA_ORB_get_default_context(CORBA_DEC_ORB_OBJECT,
                                ev,
                                &ctx);
  if (ev->_major!=CORBA_NO_EXCEPTION) {
    ReportExcep(ev);
    cleanup(); return;
  }

  /* create the request object */
  CORBA_Object_create_request(COPObj,  /* implementation object */
                              ev,        /* ev */
                              ctx,       /* context object */
                              "Exec", /* operation name */
                              NULL,    /* operation arg list */
                              (CORBA_NamedValue *)NULL,  /* operation result */
                              &request,  /* created request object */
                              (CORBA_Flags)0); /* create_request flags */
  if (ev->_major!=CORBA_NO_EXCEPTION) { ReportExcep(ev); cleanup(); }

  /*
  OBB_ORB_register_tc_lookup_rtn(CORBA_DEC_ORB_OBJECT,
                                 ev,
                                 _TC_scopd_LookupByID);
  if(ev->_major != CORBA_NO_EXCEPTION) { ReportExcep(ev); cleanup(); return;}
  */

  CORBA_Request_add_arg(request,
                        ev,
                        "instring",
                        TC_string,
                        instring,
                        sizeof(CORBA_string),
                        (CORBA_Flags)CORBA_ARG_IN);
  if (ev->_major!=CORBA_NO_EXCEPTION) { ReportExcep(ev); cleanup(); return; }

  CORBA_Request_add_arg(request,
                        ev,
                        "inshort",
                        TC_short,
                        &inshort,
                        sizeof(CORBA_short),
                        (CORBA_Flags)CORBA_ARG_IN);
  if (ev->_major!=CORBA_NO_EXCEPTION) { ReportExcep(ev); cleanup(); return; }

  CORBA_Request_add_arg(request,
                        ev,
                        "indouble",
                        TC_double,
                        &indouble,
                        sizeof(CORBA_double),
                        (CORBA_Flags)CORBA_ARG_IN);
  if (ev->_major!=CORBA_NO_EXCEPTION) { ReportExcep(ev); cleanup(); return; }

  CORBA_Request_add_arg(request,
                        ev,
                        "indseq",
                        TC_SCOPD_DoubleSeq,
                        indseq,
                        sizeof(SCOPD_DoubleSeq),
                        (CORBA_Flags)CORBA_ARG_IN);
```

```
if (ev->_major!=CORBA_NO_EXCEPTION) { ReportExcep(ev); cleanup(); return; }

CORBA_Request_add_arg(request,
                      ev,
                      "indstr",
                      TC_SCOPD_DataStr,
                      indstr,
                      sizeof(SCOPD_DataStr),
                      (CORBA_Flags)CORBA_ARG_IN);
if (ev->_major!=CORBA_NO_EXCEPTION) { ReportExcep(ev); cleanup(); return; }

CORBA_Request_add_arg(request,
                      ev,
                      "outstring",
                      TC_string,
                      outstring,
                      sizeof(CORBA_string),
                      (CORBA_Flags)CORBA_ARG_OUT);
if (ev->_major!=CORBA_NO_EXCEPTION) { ReportExcep(ev); cleanup(); return; }

CORBA_Request_add_arg(request,
                      ev,
                      "outshort",
                      TC_short,
                      outshort,
                      sizeof(CORBA_short),
                      (CORBA_Flags)CORBA_ARG_OUT);
if (ev->_major!=CORBA_NO_EXCEPTION) { ReportExcep(ev); cleanup(); return; }

CORBA_Request_add_arg(request,
                      ev,
                      "outdouble",
                      TC_double,
                      outdouble,
                      sizeof(CORBA_double),
                      (CORBA_Flags)CORBA_ARG_OUT);
if (ev->_major!=CORBA_NO_EXCEPTION) { ReportExcep(ev); cleanup(); return; }

CORBA_Request_add_arg(request,
                      ev,
                      "outdseq",
                      TC_SCOPD_DoubleSeq,
                      outdseq,
                      sizeof(SCOPD_DoubleSeq),
                      (CORBA_Flags)CORBA_ARG_OUT);
if (ev->_major!=CORBA_NO_EXCEPTION) { ReportExcep(ev); cleanup(); return; }

CORBA_Request_add_arg(request,
                      ev,
                      "outdstr",
                      TC_SCOPD_DataStr,
                      outdstr,
                      sizeof(SCOPD_DataStr),
                      (CORBA_Flags)CORBA_ARG_OUT);
if (ev->_major!=CORBA_NO_EXCEPTION) { ReportExcep(ev); cleanup(); return; }



/* send the request */
```

ASYNC.CPP

```
  printf("(sending request)¥n");
  CORBA_Request_send(request,      /* request object */
                     ev,
                     (CORBA_Flags)0);  /* request_send flags */
  if (ev->_major!=CORBA_NO_EXCEPTION) { ReportExcep(ev); cleanup(); return; }


}

int PollMessage()
{
  CORBA_Environment ev;
  int status;

  printf("(request sent, waiting for response)¥n");
  status = CORBA_Request_get_response(request, &ev,
                                      (CORBA_Flags)CORBA_RESP_NO_WAIT);
  if(status == OBB_INV_NORESP)
     return FALSE;
  else
     return TRUE;
}

void cleanup()
{
  CORBA_Environment ev;

  CORBA_Request_delete(request,&ev);
  if (ev._major!=CORBA_NO_EXCEPTION) { ReportExcep(&ev); }
  CORBA_NVList_free(arglist,&ev);
  if (ev._major!=CORBA_NO_EXCEPTION) { ReportExcep(&ev); }
  CORBA_Context_delete(ctx,&ev,(CORBA_Flags)0);
  if (ev._major!=CORBA_NO_EXCEPTION) { ReportExcep(&ev); }
  exit(1);
  return;
}
```

CHKCOP.CPP

```
//#include "stdafx.h"

#include <stdlib.h>
#include <namesvc.h>
#include <float.h>
#include "scopd.h"

#define TRUE 1
#define FALSE 0

#define OBJECT_NAME     "SCOPD::COP"
#define SEQ_SIZ                 5
#define SEQ_SIZ2                1024*1024/8
#define STR_SIZ                 80

#define SUC_EXECUTE     0
#define ERR_GETOBJ              1
#define ERR_EXECUTE     2
#define ERR_GETCTX              3
#define ERR_CREATREQ    4
#define ERR_ADDARG              5
#define ERR_REQSEND             6


typedef struct _DataStr {
    double      doubledata;
    double      dseqdata[SEQ_SIZ];
    long        longdata;
        char    stringdata[STR_SIZ];
} DataStr;

int SendExecMessage(
    CORBA_Object COPObj,
    CORBA_Environment *ev,
    CORBA_string instring,
    CORBA_long inlong,
    CORBA_double indouble,
    SCOPD_DoubleSeq * indseq,
    SCOPD_DataStr * indstr);
void cleanup();

/* Async External output value */
static CORBA_string g_outstring;
static CORBA_long g_outlong;
static CORBA_double g_outdouble;
static SCOPD_DoubleSeq g_outdseq;
static SCOPD_DataStr g_outdstr;
static  CORBA_Context ctx;
static  CORBA_NVList arglist;
static  CORBA_Request request;
static  CORBA_Object object;


/*
        DLL function
*/
__declspec(dllexport) int SyncCopExec (
    char        *instring,
    long        inlong,
    double      indouble,
    double      *indseq,
        long    indseq_num,             /* input indseq number */
    DataStr *indstr,
```

```
    char        *outstring,
    long        *outlong,
    double      *outdouble,
    double      **outdseq,
        long    *outdseq_num,    /* return outdseq number */
    DataStr *outdstr)
{
  CORBA_Environment ev;
/*  CORBA_Object object; */
  CORBA_sequence_CosNaming_NameComponent n;

  CORBA_string _outstring;
  SCOPD_DataStr _instruct;
  SCOPD_DataStr _outstruct;
  SCOPD_DoubleSeq _indseq;
  /* SCOPD_DoubleSeq _outdseq;          g_outdseq を使用する */

  unsigned long i;

  /* 初期化 */
  g_outdseq._buffer = NULL;

  /* ネームサービスからオブジェクトを取得する。*/
  n._maximum = 1;
  n._length = 1;
  n._buffer = (struct CosNaming_NameComponent *)malloc(sizeof(CosNaming_NameComponent))
;
  n._buffer->id = OBJECT_NAME;
  n._buffer->kind = (char *)1;
  object = CosNaming_NamingContext_resolve(COSNAMING_OBJECT_NAMECONTEXT, &ev, &n);
  if(ev._major != CORBA_NO_EXCEPTION)
        return ERR_GETOBJ;

  /* 入力データの作成 */
  _indseq._length =  indseq_num;
  _indseq._maximum =  indseq_num;
  _indseq._buffer = indseq;   /* 領域は DLL 外で確保 */

  _instruct.stringdata = indstr->stringdata; /* 領域は DLL 外で確保 */
  _instruct.longdata =  indstr->longdata;
  _instruct.doubledata = indstr->doubledata;
  _instruct.dseqdata._length  = SEQ_SIZ;
  _instruct.dseqdata._maximum = SEQ_SIZ;
  _instruct.dseqdata._buffer = indstr->dseqdata; /* 領域はDLL 外で確保 */

  /* メッセージ送信 */
  SCOPD_COP_Exec (object, &ev,
    instring,
    inlong,
    indouble,
    &_indseq,
    &_instruct,
    &_outstring,
    outlong,
    outdouble,
    &g_outdseq, /* &_outdseq, */
    &_outstruct);
  if(ev._major != CORBA_NO_EXCEPTION){
        return ERR_EXECUTE;
  }

  /*
        出力データの作成
```

```
    */

    /* 文字列 */
    outstring[STR_SIZ-1]=NULL;
    strncpy(outstring,_outstring,STR_SIZ-1);
    CORBA_free(_outstring);

    /* double sequence */
    *outdseq_num = g_outdseq._length;   /* _outdseq._length; */
    *outdseq = g_outdseq._buffer;           /* _outdseq._buffer; */

    /* 構造体 */

    outdstr->stringdata[STR_SIZ-1]=NULL;
    strncpy(outdstr->stringdata, _outstruct.stringdata,STR_SIZ-1);
    outdstr->longdata = _outstruct.longdata;
    outdstr->doubledata = _outstruct.doubledata;
    for(i=0; i<SEQ_SIZ; i++){
          outdstr->dseqdata[i] = _outstruct.dseqdata._buffer[i];
    }
    CORBA_free(_outstruct.dseqdata._buffer);
    CORBA_free(_outstruct.stringdata);

    /* オブジェクトの開放 */
    CORBA_Object_release(object,
                         &ev);

    return SUC_EXECUTE;
}

__declspec(dllexport) int AsyncCopExec (
      char          *instring,
      long          inlong,
      double        indouble,
      double        *indseq,
          long      indseq_num,             /* input indseq number */
      DataStr *indstr)
{
    CORBA_Environment ev;
    CORBA_Object object;
    CORBA_sequence_CosNaming_NameComponent n;

    SCOPD_DataStr _instruct;
    SCOPD_DoubleSeq _indseq;
    int status;

    /* 初期化 */
    g_outdseq._buffer = NULL;

    /* ネームサービスからオブジェクトを取得する。*/
    n._maximum = 1;
    n._length = 1;
    n._buffer = (struct CosNaming_NameComponent *)malloc(sizeof(CosNaming_NameComponent))
;
    n._buffer->id = OBJECT_NAME;
    n._buffer->kind = (char *)1;
    object = CosNaming_NamingContext_resolve(COSNAMING_OBJECT_NAMECONTEXT, &ev, &n);
    if(ev._major != CORBA_NO_EXCEPTION)
          return ERR_GETOBJ;

    /* 入力データの作成 */
    _indseq._length =  indseq_num;
    _indseq._maximum =  indseq_num;
```

```
  _indseq._buffer = indseq;   /* 領域は DLL 外で確保 */

  _instruct.stringdata = indstr->stringdata; /* 領域は DLL 外で確保 */
  _instruct.longdata =  indstr->longdata;
  _instruct.doubledata = indstr->doubledata;
  _instruct.dseqdata._length  = SEQ_SIZ;
  _instruct.dseqdata._maximum = SEQ_SIZ;
  _instruct.dseqdata._buffer = indstr->dseqdata; /* 領域はDLL 外で確保 */


  status = SendExecMessage(
    object,
    &ev,
    instring,
    inlong,
    indouble,
    &_indseq,
    &_instruct);

  return status;
}

/*
        サーバにメッセージ送信を行う
        (非同期通信用)
*/
int SendExecMessage(
    CORBA_Object COPObj,
    CORBA_Environment *ev,
    CORBA_string instring,
    CORBA_long inlong,
    CORBA_double indouble,
    SCOPD_DoubleSeq * indseq,
    SCOPD_DataStr * indstr)
{

  /* Get default context */
  CORBA_ORB_get_default_context(CORBA_DEC_ORB_OBJECT,
                                ev,
                                &ctx);
  if (ev->_major!=CORBA_NO_EXCEPTION) {
    cleanup();
    return ERR_GETCTX;
  }

  /* create the request object */
  CORBA_Object_create_request(COPObj,  /* implementation object */
                              ev,        /* ev */
                              ctx,       /* context object */
                              "Exec", /* operation name */
                              NULL,    /* operation arg list */
                              (CORBA_NamedValue *)NULL,  /* operation result */
                              &request,  /* created request object */
                              (CORBA_Flags)0); /* create_request flags */
  if (ev->_major!=CORBA_NO_EXCEPTION) { cleanup(); return ERR_CREATREQ;}

  CORBA_Request_add_arg(request,
                        ev,
                        "instring",
                        TC_string,
                        instring,
                        sizeof(CORBA_string),
                        (CORBA_Flags)CORBA_ARG_IN);
```

```
if (ev->_major!=CORBA_NO_EXCEPTION) { cleanup(); return ERR_ADDARG; }

CORBA_Request_add_arg(request,
                      ev,
                      "inlong",
                      TC_long,
                      &inlong,
                      sizeof(CORBA_long),
                      (CORBA_Flags)CORBA_ARG_IN);
if (ev->_major!=CORBA_NO_EXCEPTION) { cleanup(); return ERR_ADDARG; }

CORBA_Request_add_arg(request,
                      ev,
                      "indouble",
                      TC_double,
                      &indouble,
                      sizeof(CORBA_double),
                      (CORBA_Flags)CORBA_ARG_IN);
if (ev->_major!=CORBA_NO_EXCEPTION) { cleanup(); return ERR_ADDARG;}

CORBA_Request_add_arg(request,
                                          ev,
                      "indseq",
                      TC_SCOPD_DoubleSeq,
                      indseq,
                      sizeof(SCOPD_DoubleSeq),
                      (CORBA_Flags)CORBA_ARG_IN);
if (ev->_major!=CORBA_NO_EXCEPTION) { cleanup(); return ERR_ADDARG; }


CORBA_Request_add_arg(request,
                      ev,
                      "indstr",
                      TC_SCOPD_DataStr,
                      indstr,
                      sizeof(SCOPD_DataStr),
                      (CORBA_Flags)CORBA_ARG_IN);
if (ev->_major!=CORBA_NO_EXCEPTION) { cleanup(); return ERR_ADDARG; }


CORBA_Request_add_arg(request,
                      ev,
                      "outstring",
                      TC_string,
                      &g_outstring,
                      sizeof(CORBA_string),
                      (CORBA_Flags)CORBA_ARG_OUT);
if (ev->_major!=CORBA_NO_EXCEPTION) { cleanup(); return ERR_ADDARG; }

CORBA_Request_add_arg(request,
                      ev,
                      "outlong",
                      TC_long,
                      &g_outlong,
                      sizeof(CORBA_long),
                      (CORBA_Flags)CORBA_ARG_OUT);
if (ev->_major!=CORBA_NO_EXCEPTION) { cleanup(); return ERR_ADDARG; }

CORBA_Request_add_arg(request,
                      ev,
                      "outdouble",
                      TC_double,
                      &g_outdouble,
```

```
                        sizeof(CORBA_double),
                        (CORBA_Flags)CORBA_ARG_OUT);
  if (ev->_major!=CORBA_NO_EXCEPTION) { cleanup(); return ERR_ADDARG; }

  CORBA_Request_add_arg(request,
                        ev,
                        "outdseq",
                        TC_SCOPD_DoubleSeq,
                        &g_outdseq,
                        sizeof(SCOPD_DoubleSeq),
                        (CORBA_Flags)CORBA_ARG_OUT);
  if (ev->_major!=CORBA_NO_EXCEPTION) { cleanup(); return ERR_ADDARG; }


  CORBA_Request_add_arg(request,
                                              ev,
                        "outdstr",
                        TC_SCOPD_DataStr,
                        &g_outdstr,
                        sizeof(SCOPD_DataStr),
                        (CORBA_Flags)CORBA_ARG_OUT);
  if (ev->_major!=CORBA_NO_EXCEPTION) { cleanup(); return ERR_ADDARG; }

  /* send the request */
  printf("(sending request)¥n");
  CORBA_Request_send(request,      /* request object */
                     ev,
                     (CORBA_Flags)0);  /* request_send flags */
  if (ev->_major!=CORBA_NO_EXCEPTION) { cleanup(); return ERR_REQSEND; }

  return SUC_EXECUTE;
}

/*
        メッセージのポーリングをおこなう
*/
__declspec(dllexport)  int PollMessage()
{
  CORBA_Environment ev;
  int status;

  /* printf("(request sent, waiting for response)¥n"); */
  status = CORBA_Request_get_response(request, &ev,
                                      (CORBA_Flags)CORBA_RESP_NO_WAIT);
  if(status == OBB_INV_NORESP)
     return FALSE;
  else
     return TRUE;
}

/*
        結果の取得
*/
__declspec(dllexport) void CopGetResult(
    char        *outstring,
    long        *outlong,
    double      *outdouble,
    double      **outdseq,
        long    *outdseq_num,    /* return outdseq number */
    DataStr *outdstr)
{
    int i;
        CORBA_Environment ev;
```

```
        outstring[STR_SIZ-1]=NULL;
        strncpy(outstring,g_outstring,STR_SIZ-1);
        CORBA_free(g_outstring);
        *outlong = g_outlong;
        *outdouble = g_outdouble;
        *outdseq = g_outdseq._buffer;
        *outdseq_num = g_outdseq._length;

        outdstr->stringdata[STR_SIZ-1]=NULL;
        strncpy(outdstr->stringdata,g_outdstr.stringdata,STR_SIZ-1);
        outdstr->longdata = g_outdstr.longdata;
        outdstr->doubledata = g_outdstr.doubledata;
        for(i=0; i<SEQ_SIZ; i++){
                outdstr->dseqdata[i] = g_outdstr.dseqdata._buffer[i];
        }
        CORBA_free(g_outdstr.dseqdata._buffer);
        CORBA_free(g_outdstr.stringdata);

        CORBA_Object_release(object, &ev);
}

void cleanup()
{
  CORBA_Environment ev;

  CORBA_Request_delete(request,&ev);
  CORBA_NVList_free(arglist,&ev);
  CORBA_Context_delete(ctx,&ev,(CORBA_Flags)0);
  CORBA_Object_release(object, &ev);

  return;
}

/*
    DLL で確保した領域の開放
    (now not implemented)
*/
__declspec(dllexport) void FreeOutSeq()
{
        if(g_outdseq._buffer != NULL){
                CORBA_free(g_outdseq._buffer);
                g_outdseq._buffer = NULL;
        }
}



__declspec(dllexport) int SyncCopExecTest (
    char        *instring,
    long        inlong,
    double      indouble,
    double      *indseq,
        long    indseq_num,             /* input indseq number */
    DataStr *indstr,
    char        *outstring,
    long        *outlong,
    double      *outdouble,
    double      **outdseq,
        long    *outdseq_num,   /* return outdseq number */
    DataStr *outdstr)
{
long i;
```

```
*outlong = inlong;
*outdouble = indouble;
strcpy(outstring, instring);
*outdseq_num = indseq_num;
*outdseq = (double *)malloc(sizeof(double)*indseq_num);
for(i=0;i<indseq_num;i++)
        *(*outdseq + i) = *(indseq + i);


strcpy(outdstr->stringdata, indstr->stringdata);
outdstr->longdata = indstr->longdata;
outdstr->doubledata = indstr->doubledata;
for(i=0;i<5;i++)
        outdstr->dseqdata[i] = indstr->dseqdata[i];

return 1;

#if 0
char buf[1024];
char buf2[256];
long i;
DataStr *sp = (DataStr *)indstr;

        sprintf(buf, "str=%s l=%ld d=%e¥n", instring, inlong, indouble);
        strcat(buf, "array¥n");
        for(i=0;i<indseq_num;i++)
                {
                sprintf(buf2, "¥t %e ¥n", indseq[i]);
                strcat(buf, buf2);
                }

        sprintf(buf2, "str=%s l=%ld d=%e ¥n seqd=(%e %e %e %e %e)¥n",
                sp->stringdata,
                sp->longdata,
                sp->doubledata,
                sp->dseqdata[0],
                sp->dseqdata[1],
                sp->dseqdata[2],
                sp->dseqdata[3],
                sp->dseqdata[4]);

        strcat(buf, buf2);

        MessageBox(0, buf, 0, 0);
        return 1;
#endif
}

__declspec(dllexport) double *InitDouble(unsigned long dsize)
{
    double *buf;
        unsigned long i;

        /* buf = (double *)malloc(sizeof(double)*SEQ_SIZ2); */
        buf = (double *)malloc(sizeof(double)*dsize);
        if(buf == NULL)
                return NULL;
        /* for(i=0; i<SEQ_SIZ2; i++) */
        for(i=0; i<dsize; i++)
                buf[i] =DBL_MAX;
        return buf;
```

```
}

/*
    DLL で確保した領域の開放
    (now not implemented)
*/
__declspec(dllexport) void FreeData(void *ptr)
{
        if(ptr!=NULL)
                free(ptr);
}
```

DISP.CPP

```
/*****************************************************************
 * Created Tue Mar 26 15:23:49 1996 by OBB V2.5A-06 (COMPILE/GENERATE)
 *****************************************************************
 */

/*
 * OBB Dispatcher/Registration routines
 * ------------------------------------
 *
 * This module contains dispatcher/registration routines for the following
 * Implementations:
 *
 *    COPImpl
 *
 * WARNING: The source code in this module was generated by Digital
 * Equipment Corporation's(DEC) ObjectBroker. Digital will not support any
 * changes to the source code in this module.
 *
 * How This Module Works
 * ---------------------
 *
 * For a detailed description of the files generated by ObjectBroker, refer
 * to the gen_info.txt file in the ObjectBroker example directory.
 */
#include <stdio.h>
#include <string.h>
#ifndef ORB_MAKEPROC
#if ((defined(MSDOS) || defined(__MSDOS__)) && !(defined(_WINDLL) || defined(__DLL__)))

#include <windows.h>
/* If Windows, you must declare a global variable ORB__hInstance and
   initialize it to the HANDLE passed into WinMain. This is used in the
   call to MakeProcInstance. */
extern HANDLE ORB__hInstance;
#define ORB_MAKEPROC(rtn)  MakeProcInstance ((FARPROC) (rtn), ORB__hInstance)
#else
/* Stubs will reside in DLL, or not under windows */
#define ORB_MAKEPROC(rtn)  (rtn)
#endif
#endif
#include <orb.h>

/*
 * The following macro definitions are provided in order to allow the
 * tables and global variables to be included as SHARED READONLY in a
 * shareable image. To enable this, define ORB_SHARED_READONLY. A PSECT
 * name may be specified by defining RO_PSECT
 */

#ifndef GLOBAL_READONLY
#if !defined(ORB_SHARED_READONLY) || !defined(vms)
#define GLOBAL_READONLY
#else
#ifndef RO_PSECT
#define RO_PSECT
#endif
#define GLOBAL_READONLY globaldef RO_PSECT readonly
#endif
#endif

#ifndef STATIC_READONLY
#if !defined(ORB_SHARED_READONLY) || !defined(vms)
#define STATIC_READONLY static
```

```
#else
#define STATIC_READONLY static readonly
#endif
#endif


#ifndef GLOBAL_VARIABLE
#if !defined(ORB_SHARED_READONLY) || !defined(vms)
#define GLOBAL_VARIABLE
#else
#ifndef RW_PSECT
#define RW_PSECT
#endif
#define GLOBAL_VARIABLE globaldef RW_PSECT
#endif
#endif

#include "scopd.imh"

#define TC_SCOPD_ERR_TYPES TC_31578d75_0
#define TC_SCOPD_UserExcep TC_31578d75_1
#define TC_SCOPD_DoubleSeq TC_sequence_double
#define TC_SCOPD_COP TC_Object
#define TC_SCOPD_DataStr TC_31578d75_2

#define _TC_HASH_SCOPD_ERR_TYPES 0
#define _TC_HASH_SCOPD_UserExcep 1
#define _TC_HASH_SCOPD_DoubleSeq -38
#define _TC_HASH_SCOPD_COP -14
#define _TC_HASH_SCOPD_DataStr 2

STATIC_READONLY CORBA_unsigned_long TC_31578d75_0 [] = {
        OBBLONG(0x0d, 0x11, 0x09, 0xbb),
        OBBLONG(0x11,0x8d,0x52,0x74),OBBLONG(0x80,0x06,0x00,0x00),
        OBBLONG(0x02,0x10,0xa1,0x40),OBBLONG(0x59,0x00,0x00,0x00),
        OBBLONG(0x68,0x00,0x00,0x00),OBBLONG(0xff,0xff,0xff,0xff),
        OBBLONG(0x01,0x00,0xff,0xff),OBBLONG(0x03,0x00,0x00,0x00),
        OBBLONG(0x03,0x00,0x00,0x00),OBBLONG(0x08,0x83,0xaf,0xe1),
        OBBLONG(0x1f,0x5d,0xc9,0x11),OBBLONG(0x91,0xa4,0x08,0x00),
        OBBLONG(0x2b,0x14,0xa0,0xfa),OBBLONG(0x00,0x00,0x00,0x00),
        OBBLONG(0x04,0x00,0x00,0x00),OBBLONG(0xcc,0x00,0x00,0x00),
        OBBLONG(0xff,0xff,0xff,0xff),OBBLONG(0xff,0xff,0xff,0xff),
        OBBLONG(0xff,0xff,0xff,0xff),OBBLONG(0xff,0xff,0xff,0xff),
        OBBLONG(0x01,0x00,0x00,0x00),OBBLONG(0x00,0x00,0x00,0x00),
        OBBLONG(0x02,0x00,0x00,0x00),OBBLONG(0x01,0x00,0x00,0x00),
        OBBLONG(0x58,0x00,0x00,0x00),OBBLONG(0x00,0x00,0x00,0x00),
        OBBLONG(0xff,0xff,0xff,0xff),OBBLONG(0x01,0x00,0x00,0x00),
        OBBLONG(0x05,0xfe,0x00,0x00),OBBLONG(0x00,0x00,0x00,0x00) };

STATIC_READONLY CORBA_unsigned_long TC_31578d75_1 [] = {
        OBBLONG(0x0d, 0x0f, 0x19, 0xbb),
        OBBLONG(0x11,0x8d,0x52,0x74),OBBLONG(0x78,0x1b,0x00,0x00),
        OBBLONG(0x02,0x10,0xa1,0x40),OBBLONG(0x59,0x00,0x00,0x00),
        OBBLONG(0x78,0x00,0x00,0x00),OBBLONG(0xff,0xff,0xff,0xff),
        OBBLONG(0x01,0x00,0xff,0xff),OBBLONG(0x03,0x00,0x00,0x00),
        OBBLONG(0x03,0x00,0x00,0x00),OBBLONG(0x08,0x83,0xaf,0xe1),
        OBBLONG(0x1f,0x5d,0xc9,0x11),OBBLONG(0x91,0xa4,0x08,0x00),
        OBBLONG(0x2b,0x14,0xa0,0xfa),OBBLONG(0x00,0x00,0x00,0x00),
        OBBLONG(0x04,0x00,0x00,0x00),OBBLONG(0xcc,0x00,0x00,0x00),
        OBBLONG(0xff,0xff,0xff,0xff),OBBLONG(0xff,0xff,0xff,0xff),
        OBBLONG(0xff,0xff,0xff,0xff),OBBLONG(0xff,0xff,0xff,0xff),
        OBBLONG(0x01,0x00,0x00,0x00),OBBLONG(0x00,0x00,0x00,0x00),
        OBBLONG(0x02,0x00,0x00,0x00),OBBLONG(0x01,0x00,0x00,0x00),
```

```
        OBBLONG(0x58, 0x00, 0x00, 0x00), OBBLONG(0x00, 0x00, 0x00, 0x00),
        OBBLONG(0xff, 0xff, 0xff, 0xff), OBBLONG(0x01, 0x00, 0x00, 0x00),
        OBBLONG(0x0f, 0x07, 0xff, 0xff), OBBLONG(0x68, 0x00, 0x00, 0x00),
        OBBLONG(0xfe, 0xff, 0xff, 0xff), OBBLONG(0x01, 0x00, 0x00, 0x00),
        OBBLONG(0x05, 0xfe, 0x00, 0x00), OBBLONG(0x00, 0x00, 0x00, 0x00),
        OBBLONG(0x14, 0x00, 0x00, 0x00),  0x00000000,  sizeof(SCOPD_UserExcep)
        , (CORBA_unsigned_long) (((CORBA_octet*)&((SCOPD_UserExcep * ) NULL)->Reason-
(CORBA_octet*)0))
        , 0x00000000 };

STATIC_READONLY CORBA_unsigned_long TC_31578d75_2 [] = {
        OBBLONG(0x0d,  0x0f,  0x19,  0xbb),
        OBBLONG(0x11, 0x8d, 0x52, 0x74), OBBLONG(0x54, 0x1e, 0x00, 0x00),
        OBBLONG(0x02, 0x10, 0xa1, 0x40), OBBLONG(0x59, 0x00, 0x00, 0x00),
        OBBLONG(0x8c, 0x00, 0x00, 0x00), OBBLONG(0xff, 0xff, 0xff, 0xff),
        OBBLONG(0x01, 0x00, 0xff, 0xff), OBBLONG(0x03, 0x00, 0x00, 0x00),
        OBBLONG(0x03, 0x00, 0x00, 0x00), OBBLONG(0x08, 0x83, 0xaf, 0xe1),
        OBBLONG(0x1f, 0x5d, 0xc9, 0x11), OBBLONG(0x91, 0xa4, 0x08, 0x00),
        OBBLONG(0x2b, 0x14, 0xa0, 0xfa), OBBLONG(0x00, 0x00, 0x00, 0x00),
        OBBLONG(0x04, 0x00, 0x00, 0x00), OBBLONG(0xcc, 0x00, 0x00, 0x00),
        OBBLONG(0xff, 0xff, 0xff, 0xff), OBBLONG(0xff, 0xff, 0xff, 0xff),
        OBBLONG(0xff, 0xff, 0xff, 0xff), OBBLONG(0xff, 0xff, 0xff, 0xff),
        OBBLONG(0x01, 0x00, 0x00, 0x00), OBBLONG(0x00, 0x00, 0x00, 0x00),
        OBBLONG(0x02, 0x00, 0x00, 0x00), OBBLONG(0x01, 0x00, 0x00, 0x00),
        OBBLONG(0x58, 0x00, 0x00, 0x00), OBBLONG(0x00, 0x00, 0x00, 0x00),
        OBBLONG(0xff, 0xff, 0xff, 0xff), OBBLONG(0x01, 0x00, 0x00, 0x00),
        OBBLONG(0x0f, 0x07, 0xff, 0xff), OBBLONG(0x68, 0x00, 0x00, 0x00),
        OBBLONG(0xfe, 0xff, 0xff, 0xff), OBBLONG(0x01, 0x00, 0x00, 0x00),
        OBBLONG(0x28, 0x56, 0x09, 0x04), OBBLONG(0x58, 0xff, 0xff, 0xff),
        OBBLONG(0x7c, 0x00, 0x00, 0x00), OBBLONG(0xfe, 0xff, 0xff, 0xff),
        OBBLONG(0x06, 0x00, 0x00, 0x00), OBBLONG(0x04, 0x00, 0x00, 0x00),
        OBBLONG(0x00, 0x00, 0x00, 0x00),
        OBBLONG(0x24, 0x00, 0x00, 0x00),  0x00000000,  sizeof(SCOPD_DataStr)
        , (CORBA_unsigned_long) (((CORBA_octet*)&((SCOPD_DataStr * ) NULL)->stringdat
a-(CORBA_octet*)0))
        , (CORBA_unsigned_long) (((CORBA_octet*)&((SCOPD_DataStr * ) NULL)->longdata-
(CORBA_octet*)0))
        , (CORBA_unsigned_long) (((CORBA_octet*)&((SCOPD_DataStr * ) NULL)->doubledat
a-(CORBA_octet*)0))
        , (CORBA_unsigned_long) (((CORBA_octet*)&((SCOPD_DataStr * ) NULL)->dseqdata-
(CORBA_octet*)0))
        , 0x00000000
        , 0x00000000 };


GLOBAL_READONLY struct {
        CORBA_char    Bucket0 [75];     CORBA_char    Bucket1 [38];
    } _TCS_scopd_BT = {
        "74528d110680.02.10.a1.40.59.00.00.00¥00074528d111b78.02.10.a1.40.59.00.00.00
¥000",
        "74528d111e54.02.10.a1.40.59.00.00.00¥000"
    };

GLOBAL_READONLY OBBHashEntry      _TCS_scopd_HT [3] = {
    {   0,      0 },     {   2,     75 },     {  -1, 65535 }
    };

CORBA_TypeCode _TCS_scopd_HashLookup( CORBA_short HashIndex)
 {

 if (HashIndex < 0)
    return(OBB__ConvertKindToTypeCode(
                (CORBA_Environment *) NULL,
```

```
                (CORBA_short) (- HashIndex)));

 switch (HashIndex)
        {
         case _TC_HASH_SCOPD_ERR_TYPES :
                return(TC_31578d75_0);

         case _TC_HASH_SCOPD_UserExcep :
                return(TC_31578d75_1);

         case _TC_HASH_SCOPD_DataStr :
                return(TC_31578d75_2);

         default:
                break;
        }
 return ((CORBA_TypeCode) CORBA_OBJECT_NIL);
 }

CORBA_TypeCode OBB_EXPORT _TCS_scopd_LookupByID( CORBA_string TypeCodeID)
 {
 CORBA_short hash_index = -1;

 hash_index = OBB__MethodHash(
        TypeCodeID, 3,
        _TCS_scopd_HT,
        (CORBA_string) &_TCS_scopd_BT);
 if (hash_index < 0)
        return((CORBA_TypeCode) CORBA_OBJECT_NIL);

 return ( _TCS_scopd_HashLookup( hash_index ) );

 }

/*
 * The following are Repository Constants for the implementation and
 * any interfaces that it implements. The implementation constant
 * can be used to pass to the generated registration routine. The interface
 * constants can be used to create objects with the default binding.
 */

GLOBAL_READONLY OBB_ReposConst _OBB_IMPLDEF_31578d75 =
        {
        10, 18, 1, 187, 114, 31, 124, 116, 0, 98, 0, 0, 2, 16, 161, 64,
        89, 0, 0, 0, 24, 0, 0, 0
        };

GLOBAL_READONLY CORBA_ImplementationDef COPImpl__OBJ =
        (CORBA_ImplementationDef) _OBB_IMPLDEF_31578d75;

GLOBAL_READONLY OBB_ReposConst _OBB_INTFDEF_31578d75 =
        {
        10, 20, 1, 187, 17, 141, 82, 116, 72, 31, 0, 0, 2, 16, 161, 64,
        89, 0, 0, 0, 24, 0, 1, 0
        };

GLOBAL_READONLY CORBA_InterfaceDef SCOPD_COP__OBJ =
        (CORBA_InterfaceDef) _OBB_INTFDEF_31578d75;

/*
 * The following globals are the UUID and name for each of the implementations
 * for which a dispatch routine is generated. These globals are used in the
 * generated registration routine used for registering the corresponding
```

```
 * implementation.
 */

GLOBAL_READONLY CORBA_char     COPIMPL_ID [] = "74528d111f48.02.10.a1.40.59.00.00.00"
;
GLOBAL_READONLY CORBA_char     COPIMPL_NAM [] = "COPImpl";
GLOBAL_READONLY CORBA_octet     COPIMPL_I0[16] = {
        17,141,82,116,72,31,0,0,2,16,161,64,89,0,0,0
        };

GLOBAL_READONLY CORBA_octet     COPIMPL_M0[3] = {
        1,0,128
        };

GLOBAL_READONLY OBBImplMapEntry  COPIMPL_IM[1] = {
        { COPIMPL_I0,  COPIMPL_M0 }
    };
GLOBAL_READONLY CORBA_unsigned_short    COPIMPL_NI = 1;
GLOBAL_READONLY struct {
        CORBA_char    Bucket0 [38];    } COPIMPL_BT = {
        "74528d11203c.02.10.a1.40.59.00.00.00¥000"
    };
GLOBAL_READONLY OBBHashEntry     COPIMPL_HT [1] = {
    {   0,      0 }
    };
GLOBAL_READONLY CORBA_unsigned_long     COPIMPL_NO = 1;
GLOBAL_READONLY CORBA_unsigned_short COPIMPL_OM[1] = {
        0
    };
GLOBAL_READONLY CORBA_char      * COPIMPL_NT[1] = {
        "Exec"
    };
GLOBAL_READONLY CORBA_unsigned_long     COPIMPL_NM = 1;
GLOBAL_READONLY OBBMethodRtnContextEntry  COPIMPL_MC[1] = {
        { 0,  0 }
    };

GLOBAL_VARIABLE OBBImplementationData COPIMPL_IPD;

/* Get method args routine prototype. */

CORBA_NVList OBB_EXPORT
COPImpl__getmthargs (

OBB_BinaryID            ImplId,
CORBA_Environment       * Ev,
OBB_OpInfo              * OpInfo,
CORBA_NamedValue        * Result,
CORBA_Flags             Flags);

GLOBAL_READONLY CORBA_short COPIMPL_AO[1] = {
        0
    };
GLOBAL_READONLY CORBA_short COPIMPL_AH[12] = {
        10,_TC_HASH_void,_TC_HASH_string,_TC_HASH_long,_TC_HASH_double,
        _TC_HASH_SCOPD_DoubleSeq,_TC_HASH_SCOPD_DataStr,_TC_HASH_string,
        _TC_HASH_long,_TC_HASH_double,_TC_HASH_SCOPD_DoubleSeq,
        _TC_HASH_SCOPD_DataStr,
    };
```

DISP.CPP

```
/*
 *
 *  ROUTINE NAME:       COPImpl__dispatch
 *
 *  FUNCTIONAL DESCRIPTION:
 *
 *      This is the method dispatch routine for the implementation
 *          COPImpl.

 */

CORBA_Status OBB_EXPORT
COPImpl__dispatch (

OBBInvocationContext * invctx,
CORBA_string         method,
OBBInt               argcount,
OBBArgList           args,
CORBA_Environment    * ev)

{
 OBBInt              hashvalue;
 CORBA_unsigned_short   method_index;
 OBBArg              * arg_ptr = args;
 CORBA_Object        obj_ref;

 /*.....................................................*/

 obj_ref = invctx->ObjRef;

 if ( (argcount != 0) && (arg_ptr == NULL) )
    return(OBB_INV_BADARGLST);

 hashvalue = OBB__MethodHash (method,
             (OBBInt) COPIMPL_IPD.NumOperations,
             COPIMPL_IPD.HashTable, (CORBA_string) COPIMPL_IPD.BucketTable );

 method_index = COPIMPL_IPD.OperToMthTable [ hashvalue ];
 switch ( method_index )
    {
    case 0 : /* Exec */
         {

         if ( argcount < 11 )
             return(OBB_INV_ARGMISMATCH);

         if ( (argcount > 11) &&
              !(OBB__IsAllOptional(args, argcount, 11)))
             return(OBB_INV_ARGMISMATCH);

         COPImpl_Exec (
             obj_ref,
             ev,
             (CORBA_string) args[0] . Value,
             * ((CORBA_long *) args[1] . Value),
             * ((CORBA_double *) args[2] . Value),
             (SCOPD_DoubleSeq *) args[3] . Value,
             (SCOPD_DataStr *) args[4] . Value,
             (CORBA_string *) args[5] . Value,
             (CORBA_long *) args[6] . Value,
             (CORBA_double *) args[7] . Value,
             (SCOPD_DoubleSeq *) args[8] . Value,
```

```
            (SCOPD_DataStr *) args[9] . Value);

        break;
        }

    default :
        return(OBB_INV_MTHDNOTFND);
    }

 switch (ev->_major)
    {
     case CORBA_NO_EXCEPTION:     return (OBB_INV_METHODSUC);
     case CORBA_USER_EXCEPTION:   return (OBB_INV_METHODFAIL);
     case CORBA_SYSTEM_EXCEPTION: return (OBB_INV_METHODFAIL);
     default:                    return (OBB_INV_METHODFAIL);
    }

}
```

```
/*
 *
 *  ROUTINE NAME:       COPImpl__register
 *
 *  FUNCTIONAL DESCRIPTION:
 *
 *      This is the registration routine for the method
 *      server COPImpl.
 *
 *      This routine registers and creates the ImplementationDef.
 *
 *
 *  FORMAL PARAMETERS:
 *
 *      parameter name          [In|Out|In/Out]         [Reference|Value]
 *      --------------          ---------------         -----------------
 *      ImplConst               In                      Reference
 *                              The implementation constant
 *
 *      Ev                      In/Out                  Reference
 *                              The environment. This will contain any
 *                              exceptions from the ORB or BOA routines.
 *
 *      RegServerAttrList       In              Reference
 *                              A list of additional registration attributes.
 *
 *      SelServerAttrList       In              Reference
 *                              A list of additional selection attributes.
 *
 *      Flags                   In                      Value
 *                              The Flags for the registration routine.
 *.                             Possible values are TBS.
 *
 *  FUNCTION VALUE:
 *
 *      CORBA_ImplementationDef - the ImplementationDef returned from
 *              CORBA_BOA_create_implementation.
 *      NULL - if an error occurs in the ORB or BOA routines.
 *
 */

CORBA_ImplementationDef
COPImpl__register (

CORBA_ImplementationDef ImplConst,
CORBA_Environment       * Ev,
CORBA_NVList            RegServerAttrList,
CORBA_NVList            SelServerAttrList,
CORBA_Flags             Flags)

{
 OBB_procedure dispatch_rtn = (OBB_procedure) ORB_MAKEPROC(COPImpl__dispatch);
 OBB_procedure get_mthd_args_rtn = (OBB_procedure) ORB_MAKEPROC(COPImpl__getmthargs);
 OBB_procedure lookup_tc_by_id_rtn = (OBB_procedure) ORB_MAKEPROC(_TCS_scopd_LookupBy
ID);
 CORBA_Flags            reg_flags =  Flags;
 CORBA_NVList           attr_list;
 CORBA_NVList           reg_attr_list;
 CORBA_NVList           sel_attr_list = NULL;
 CORBA_NVList           merge_list;
 CORBA_ImplementationDef  impl;
 CORBA_Status           status;
```

```
CORBA_long              reg_attr_count = 4;
CORBA_long              sel_attr_count = 0;
/*.....................................................*/


/* Initialize the implementation data */

COPIMPL_IPD . Name = COPIMPL_NAM;
COPIMPL_IPD . Id = COPIMPL_ID;
COPIMPL_IPD . NumMethods = COPIMPL_NM;
COPIMPL_IPD . NumOperations = COPIMPL_NO;
COPIMPL_IPD . BucketTable = (OBB_VoidPtr) & COPIMPL_BT;
COPIMPL_IPD . HashTable = COPIMPL_HT;
COPIMPL_IPD . NameTable = COPIMPL_NT;
COPIMPL_IPD . MethodRtnContextTable = COPIMPL_MC;
COPIMPL_IPD . OperToMthTable = COPIMPL_OM;
COPIMPL_IPD . NumActivationContext = 0;
COPIMPL_IPD . ActivationContextTable = NULL;
COPIMPL_IPD . ActivationPolicy = 1;
COPIMPL_IPD . ActivationType = 1;
COPIMPL_IPD . NumImplMap = COPIMPL_NI;
COPIMPL_IPD . ImplMapTable = COPIMPL_IM;

if (Ev != NULL)
   Ev->_major = CORBA_NO_EXCEPTION;

if (sel_attr_count > 0 || SelServerAttrList != NULL)
   reg_attr_count++;

/* Create registration and selection attribute lists. */

status = CORBA_ORB_create_list (CORBA_DEC_ORB_OBJECT, Ev, reg_attr_count,
           & reg_attr_list);
attr_list = reg_attr_list;
if (status != OBB_SUCCESS)
   return (NULL);

/* OBB_IMPL_DATA */

status = CORBA_NVList_add_item (attr_list, Ev,
   (CORBA_Identifier)OBB_IMPL_DATA,
   (CORBA_TypeCode)TC_VoidPtr,
   (CORBA_void *)& COPIMPL_IPD,
   (CORBA_long)sizeof(COPIMPL_IPD),
   (CORBA_Flags)0);
if (status != OBB_SUCCESS)
   {
   CORBA_NVList_free (attr_list, NULL);
   return (NULL);
   }

/* OBB_IMPLEMENTATION_NAME */

status = CORBA_NVList_add_item (attr_list, Ev,
   (CORBA_Identifier)OBB_IMPLEMENTATION_NAME,
   (CORBA_TypeCode)TC_string,
   (CORBA_void *)COPIMPL_NAM,
   (CORBA_long)strlen(COPIMPL_NAM),
   (CORBA_Flags)0);
if (status != OBB_SUCCESS)
   {
   CORBA_NVList_free (attr_list, NULL);
   return (NULL);
```

```
    }

status = CORBA_NVList_add_item (attr_list, Ev,
    (CORBA_Identifier)OBB_GET_MTHD_ARGS_RTN,
    (CORBA_TypeCode)TC_procedure,
    (CORBA_void *) & get_mthd_args_rtn,
    (CORBA_long) sizeof (OBB_procedure),
    (CORBA_Flags)0 );
if (status != OBB_SUCCESS)
    {
    CORBA_NVList_free (attr_list, NULL);
    return (NULL);
    }

status = CORBA_NVList_add_item (attr_list, Ev,
    (CORBA_Identifier)OBB_LOOKUP_TC_BY_ID_RTN,
    (CORBA_TypeCode)TC_procedure,
    (CORBA_void *) & lookup_tc_by_id_rtn,
    (CORBA_long) sizeof (OBB_procedure),
    (CORBA_Flags)0 );
if (status != OBB_SUCCESS)
    {
    CORBA_NVList_free (attr_list, NULL);
    return (NULL);
    }

/* Merge registration overrides. */

if (RegServerAttrList != NULL && reg_attr_list != NULL)
    {
    /* Create new merged list. */

    status = OBB__MergeLists((OBBList) RegServerAttrList, (OBBList)reg_attr_list,
        (OBBList *) & merge_list);
    if (status != OBB_SUCCESS)
        {
        CORBA_NVList_free (reg_attr_list, NULL);
        return (NULL);
        }
    CORBA_NVList_free (reg_attr_list, NULL);
    attr_list = reg_attr_list = merge_list;
    }

if (sel_attr_count > 0)
    {
    status = CORBA_ORB_create_list (CORBA_DEC_ORB_OBJECT, Ev, sel_attr_count,
        & sel_attr_list);
    if (status != OBB_SUCCESS)
        {
        CORBA_NVList_free (reg_attr_list, NULL);
        return (NULL);

        }
    }

attr_list = sel_attr_list;

/* Merge selection overrides. */

if (SelServerAttrList != NULL && sel_attr_list != NULL)
    {
    /* Create new merged list. */
```

```
        status = OBB__MergeLists((OBBList)SelServerAttrList, (OBBList)sel_attr_list,
            (OBBList *) & merge_list);
        if (status != OBB_SUCCESS)
            {
            CORBA_NVList_free (reg_attr_list, NULL);
            CORBA_NVList_free (sel_attr_list, NULL);
            return (NULL);
            }
        CORBA_NVList_free (sel_attr_list, NULL);
        attr_list = sel_attr_list = merge_list;
        }
    else if (SelServerAttrList != NULL)
        attr_list = SelServerAttrList;
    else attr_list = sel_attr_list;

    if (attr_list != NULL)
        {
        status = OBB_NVList_get_utilized (attr_list, Ev, & sel_attr_count);
        if (status == OBB_SUCCESS)
            status = CORBA_NVList_add_item (reg_attr_list, Ev,
                    (CORBA_Identifier)OBB_SELATTR_LIST,
                    (CORBA_TypeCode)TC_reserved_nvl,
                    (CORBA_void *) attr_list,
                    (CORBA_long) sel_attr_count,
                    (CORBA_Flags)0 );
        }
    if (status != OBB_SUCCESS)
        {
        CORBA_NVList_free (sel_attr_list, NULL);
        CORBA_NVList_free (reg_attr_list, NULL);
        return (NULL);
        }

    /* Create an implementation object. */

    impl = OBB__CreateImplementationDef( Ev,
        ImplConst,
        reg_attr_list,
        reg_flags,
        dispatch_rtn);

    CORBA_NVList_free( reg_attr_list, NULL);
    if (sel_attr_list != NULL)
        CORBA_NVList_free( sel_attr_list, NULL);
    return (impl);
}
```

```
/*
 *
 *  ROUTINE NAME:       COPImpl__getmthargs
 *
 *  FUNCTIONAL DESCRIPTION:
 *
 *      This is the get method argsroutine for the implementation
 *          COPImpl.
 *
 *      This routine returns the list of arguments for the specified method.
 *
 */

CORBA_NVList OBB_EXPORT
COPImpl__getmthargs (

OBB_BinaryID            ImplId,
CORBA_Environment       * Ev,
OBB_OpInfo              * OpInfo,
CORBA_NamedValue        * Result,
CORBA_Flags             Flags)

{
 OBBInt         hashvalue;
 CORBA_unsigned_short   method_index;
 CORBA_short            arg_offset;
 CORBA_long             arg_count;
 CORBA_NVList           arg_list;
 CORBA_short            * index_ptr;
 CORBA_short            i;
 CORBA_NamedValue               * arg_ptr;
 CORBA_Status           status;

 /*.....................................................................*/

 hashvalue = OBB__MethodHash (OpInfo -> OperationID,
        (OBBInt) COPIMPL_IPD.NumOperations,
        COPIMPL_IPD.HashTable, (CORBA_string) COPIMPL_IPD.BucketTable);
 if (hashvalue < 0)
    {
    if (Ev != NULL)
       {
       Ev -> _major = CORBA_SYSTEM_EXCEPTION;
       Ev -> status = OBB_INV_MTHDNOTFND;
       }
    return(NULL);
    }

 method_index = COPIMPL_IPD.OperToMthTable [ hashvalue ];
 arg_offset = COPIMPL_AO [method_index];
 index_ptr = & COPIMPL_AH [arg_offset];
 arg_count = (* index_ptr); index_ptr++;

 status = CORBA_ORB_create_list (CORBA_DEC_ORB_OBJECT, Ev, arg_count+1,
           & arg_list);
 if (status != OBB_SUCCESS)
    return (NULL);

 arg_list[arg_count].argument._type = _TCS_scopd_HashLookup (*index_ptr);
 index_ptr++;
 for (i = 0, arg_ptr = arg_list; i < arg_count; i++, index_ptr++, arg_ptr++)
    {
```

```
    arg_ptr -> argument._type = _TCS_scopd_HashLookup (*index_ptr);
    }

 return (arg_list);
}
```

SCOPD.CPP

```
// scopd.cpp : DLL 用の初期化処理の定義を行います。
//

#include <afxdllx.h>

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

static AFX_EXTENSION_MODULE scopdDLL = { NULL, NULL };

extern "C" int APIENTRY
DllMain(HINSTANCE hInstance, DWORD dwReason, LPVOID lpReserved)
{
        if (dwReason == DLL_PROCESS_ATTACH)
        {
                TRACE0("SCOPD.DLL 初期化処理中。¥n");

                // 拡張 DLL を 1 回だけ初期化します
                AfxInitExtensionModule(scopdDLL, hInstance);

                // この DLL をリソース チェインへ挿入します
                new CDynLinkLibrary(scopdDLL);
        }
        else if (dwReason == DLL_PROCESS_DETACH)
        {
                TRACE0("SCOPD.DLL 終了処理中。¥n");
        }
        return 1;   // ok
}
```

STDAFX.CPP

```
// stdafx.cpp : 標準インクルードファイルを含むソースファイル
//       scopd.pch : 生成されるプリコンパイル済ヘッダー
//       stdafx.obj : 生成されるプリコンパイル済タイプ情報

#include "stdafx.h"
```

```
/*****************************************************************
 * Created Wed Apr 17 09:58:23 1996 by OBB V2.5A-04 (COMPILE/GENERATE)
 *****************************************************************
 */

/*
 * OBB Client Stubs routines
 * -------------------------
 *
 * This module contains client stub routines for the following
 * Interfaces:
 *
 *    COP
 *
 * WARNING: The source code in this module was generated by Digital
 * Equipment Corporation's(DEC) ObjectBroker. Digital will not support any
 * changes to the source code in this module.
 *
 * How This Module Works
 * ---------------------
 *
 * For a detailed description of the files generated by ObjectBroker, refer
 * to the gen_info.txt file in the ObjectBroker example directory.
 */
//#include "stdafx.h"
#include "scopd.h"
#ifndef ORB_MAKEPROC
#if ((defined(MSDOS) || defined(__MSDOS__)) && !(defined(_WINDLL) || defined(__DLL__)))

#include <windows.h>
/* If Windows, you must declare a global variable ORB__hInstance and
   initialize it to the HANDLE passed into WinMain. This is used in the
   call to MakeProcInstance. */
extern HANDLE ORB__hInstance;
#define ORB_MAKEPROC(rtn)  MakeProcInstance ((FARPROC) (rtn), ORB__hInstance)
#else
/* Stubs will reside in DLL, or not under windows */
#define ORB_MAKEPROC(rtn)  (rtn)
#endif
#endif
/*
 * The following macro definitions are provided in order to allow the
 * tables and global variables to be included as SHARED READONLY in a
 * shareable image. To enable this, define ORB_SHARED_READONLY. A PSECT
 * name may be specified by defining RO_PSECT
 */

#ifndef GLOBAL_READONLY
#if !defined(ORB_SHARED_READONLY) || !defined(vms)
#define GLOBAL_READONLY
#else
#ifndef RO_PSECT
#define RO_PSECT
#endif
#define GLOBAL_READONLY globaldef RO_PSECT readonly
#endif
#endif

#ifndef STATIC_READONLY
#if !defined(ORB_SHARED_READONLY) || !defined(vms)
#define STATIC_READONLY static
#else
#define STATIC_READONLY static readonly
```

```
#endif
#endif


#ifndef GLOBAL_VARIABLE
#if !defined(ORB_SHARED_READONLY) || !defined(vms)
#define GLOBAL_VARIABLE
#else
#ifndef RW_PSECT
#define RW_PSECT
#endif
#define GLOBAL_VARIABLE globaldef RW_PSECT
#endif
#endif
```

```
/*
 *
 *  ROUTINE NAME:       COPMap__resolve
 *
 *  FUNCTIONAL DESCRIPTION:
 *
 *      This is the resolve routine for the method map
 *          COPMap.
 *
 *      This routine is a compiled version of the method map and allows
 *      the clients stubs to not require a repository.
 */


void OBB_EXPORT
COPMap__resolve (

CORBA_Request           Req,
CORBA_Environment       * Ev,
CORBA_Context           Ctx,
CORBA_string            InterfaceID,
CORBA_string            OperationName)

{
 OBB_SelMthdDescription mthd_dsc;
 OBB_SelSvrDescription svr_dsc;
 CORBA_Identifier search_scope = NULL;
 CORBA_Flags req_flags = 0;

 /*.....................................................*/

 memset ((void *) & mthd_dsc, 0, (size_t) sizeof(mthd_dsc));
 memset ((void *) & svr_dsc, 0, (size_t) sizeof(svr_dsc));


 if (strcmp (OperationName, "Exec") == 0)
    {
    if (Ev -> _major == CORBA_NO_EXCEPTION)
        {
        mthd_dsc.imp_id = "74528d111f48.02.10.a1.40.59.00.00.00";
        }
    }

 if (Ev->_major == CORBA_NO_EXCEPTION)
    OBB__Request_set_ctx_search (Req, Ev, search_scope, req_flags);
 if (Ev->_major == CORBA_NO_EXCEPTION)
    OBB__Request_set_select_method (Req, Ev, & mthd_dsc, req_flags);
 if (Ev->_major == CORBA_NO_EXCEPTION)
    OBB__Request_set_select_server (Req, Ev, & svr_dsc, req_flags);
 return;
}

static OBB_TypeCodeProc _lookup_tc_by_id_rtn = (OBB_TypeCodeProc) 0;

static OBB_procedure _COP_resolve_rtn = (OBB_procedure) 0;

STATIC_READONLY
    CORBA_char  _SCOPD_COP_ID [] = "74528d111f48.02.10.a1.40.59.00.00.00";
```

```
/*
 *
 *  ROUTINE NAME:       SCOPD_COP_Exec
 *
 *  FUNCTIONAL DESCRIPTION:
 *
 *      Client stub routine for operation Exec.
 *       (Interface : COP)
 *
 */

 void SCOPD_COP_Exec (SCOPD_COP object,  CORBA_Environment * ev,
    CORBA_string instring,
    CORBA_long inlong,
    CORBA_double indouble,
    SCOPD_DoubleSeq * indseq,
    SCOPD_DataStr * indstr,
    CORBA_string * outstring,
    CORBA_long * outlong,
    CORBA_double * outdouble,
    SCOPD_DoubleSeq * outdseq,
    SCOPD_DataStr * outdstr)

{
 CORBA_Request          _request;
 CORBA_Context          _ctx;
 CORBA_Identifier       _opname;
 CORBA_Flags            _invoke_flags, _request_flags;
 CORBA_Status           _status;
 CORBA_Flags            _opflags = 0;
 OBB_OpInfo             _op_info;
 CORBA_Identifier       _argname;
 CORBA_void             * _argval;
 CORBA_long             _arglen;
 CORBA_Flags            _argflags;
 CORBA_TypeCode         _argtype;
 /*.....................................................*/

 _opname = "Exec";
 _request_flags = 0;
 _invoke_flags = OBB_NO_REP_OPERATION | OBB_NO_REP_METHOD_MAP;

 /* Get default context object */

 _status = CORBA_ORB_get_default_context(
    CORBA_DEC_ORB_OBJECT,       /* object reference */
    ev,                         /* environment object */
    & _ctx);                    /* (out) context object */

 if (_status != OBB_SUCCESS)
    {
    return;
    }

 /* Register the lookup routine if not done already */

 if (!_lookup_tc_by_id_rtn)
  {
     _lookup_tc_by_id_rtn = (OBB_TypeCodeProc) ORB_MAKEPROC(_TC_scopd_LookupByID);
     _status = OBB_ORB_register_tc_lookup_rtn (
               CORBA_DEC_ORB_OBJECT,
               ev,
```

```
                _lookup_tc_by_id_rtn);

     if (_status != OBB_SUCCESS)
       {
         CORBA_Context_delete( _ctx, NULL,
                               (CORBA_Flags) 0);
    return;
    }
}
 /* Create request */

 _status = CORBA_Object_create_request (
    object,                      /* object reference */
    ev,                          /* environment object */
    _ctx,                        /* context object */
    _opname,                     /* operation name */
    NULL,                        /* argument list */
    NULL,                        /* (out) result */
    & _request,                  /* (out) request object */
    _request_flags);             /* flags */

 if (_status != OBB_SUCCESS)
    {

    CORBA_Context_delete( _ctx, NULL, (CORBA_Flags)0 );
    return;
    }

 if (!_COP_resolve_rtn)
    _COP_resolve_rtn = (OBB_procedure) ORB_MAKEPROC(COPMap__resolve);

 /* Set operation information. */

 if (_status == OBB_SUCCESS)
    {
    _op_info.InterfaceID = _SCOPD_COP_ID;
    _op_info.OperationID = "74528d11203c.02.10.a1.40.59.00.00.00";
    _op_info.OperationIndex = 1;
    _op_info.ResolveRtn = _COP_resolve_rtn;

    _status = OBB__Request_set_opinfo (_request, ev, & _op_info, _opflags);
    }

 if (_status == OBB_SUCCESS)
    {
    _argname = "instring";
    _argflags = CORBA_ARG_IN;
    _argval = (CORBA_void *) instring;
    _arglen = (instring == NULL) ? 0 : strlen(instring) + 1;
    _argtype = TC_string;

    _status = CORBA_Request_add_arg (
        _request,                /* request object */
        ev,                      /* environment object */
        _argname,                /* argument name */
        _argtype,                /* argument type */
        _argval,                 /* argument value */
        _arglen,                 /* argument length */
        _argflags);              /* argument flags */
    }

 if (_status == OBB_SUCCESS)
    {
```

```
    _argname = "inlong";
    _argflags = CORBA_ARG_IN;
    _argval = (CORBA_void *) & inlong;
    _arglen = sizeof (CORBA_long);
    _argtype = TC_long;

    _status = CORBA_Request_add_arg (
        _request,                /* request object */
        ev,                      /* environment object */
        _argname,                /* argument name */
        _argtype,                /* argument type */
        _argval,                 /* argument value */
        _arglen,                 /* argument length */
        _argflags);              /* argument flags */
    }

if (_status == OBB_SUCCESS)
    {
    _argname = "indouble";
    _argflags = CORBA_ARG_IN;
    _argval = (CORBA_void *) & indouble;
    _arglen = sizeof (CORBA_double);
    _argtype = TC_double;

    _status = CORBA_Request_add_arg (
        _request,                /* request object */
        ev,                      /* environment object */
        _argname,                /* argument name */
        _argtype,                /* argument type */
        _argval,                 /* argument value */
        _arglen,                 /* argument length */
        _argflags);              /* argument flags */
    }

if (_status == OBB_SUCCESS)
    {
    _argname = "indseq";
    _argflags = CORBA_ARG_IN;
    _argval = (CORBA_void *) indseq;
    _arglen = sizeof (SCOPD_DoubleSeq);
    _argtype = TC_SCOPD_DoubleSeq;

    _status = CORBA_Request_add_arg (
        _request,                /* request object */
        ev,                      /* environment object */
        _argname,                /* argument name */
        _argtype,                /* argument type */
        _argval,                 /* argument value */
        _arglen,                 /* argument length */
        _argflags);              /* argument flags */
    }

if (_status == OBB_SUCCESS)
    {
    _argname = "indstr";
    _argflags = CORBA_ARG_IN;
    _argval = (CORBA_void *) indstr;
    _arglen = sizeof (SCOPD_DataStr);
    _argtype = TC_SCOPD_DataStr;

    _status = CORBA_Request_add_arg (
        _request,                /* request object */
        ev,                      /* environment object */
```

```
        _argname,                /* argument name */
        _argtype,                /* argument type */
        _argval,                 /* argument value */
        _arglen,                 /* argument length */
        _argflags);              /* argument flags */
    }

if (_status == OBB_SUCCESS)
    {
    _argname = "outstring";
    _argflags = CORBA_ARG_OUT;
    _argval = (CORBA_void *)  outstring;
    _arglen = 0;
    _argtype = TC_string;

    _status = CORBA_Request_add_arg (
        _request,                /* request object */
        ev,                      /* environment object */
        _argname,                /* argument name */
        _argtype,                /* argument type */
        _argval,                 /* argument value */
        _arglen,                 /* argument length */
        _argflags);              /* argument flags */
    }

if (_status == OBB_SUCCESS)
    {
    _argname = "outlong";
    _argflags = CORBA_ARG_OUT;
    _argval = (CORBA_void *) outlong;
    _arglen = sizeof (CORBA_long);
    _argtype = TC_long;

    _status = CORBA_Request_add_arg (
        _request,                /* request object */
        ev,                      /* environment object */
        _argname,                /* argument name */
        _argtype,                /* argument type */
        _argval,                 /* argument value */
        _arglen,                 /* argument length */
        _argflags);              /* argument flags */
    }

if (_status == OBB_SUCCESS)
    {
    _argname = "outdouble";
    _argflags = CORBA_ARG_OUT;
    _argval = (CORBA_void *) outdouble;
    _arglen = sizeof (CORBA_double);
    _argtype = TC_double;

    _status = CORBA_Request_add_arg (
        _request,                /* request object */
        ev,                      /* environment object */
        _argname,                /* argument name */
        _argtype,                /* argument type */
        _argval,                 /* argument value */
        _arglen,                 /* argument length */
        _argflags);              /* argument flags */
    }

if (_status == OBB_SUCCESS)
    {
```

```
    _argname = "outdseq";
    _argflags = CORBA_ARG_OUT;
    _argval = (CORBA_void *) outdseq;
    _arglen = sizeof (SCOPD_DoubleSeq);
    _argtype = TC_SCOPD_DoubleSeq;

    _status = CORBA_Request_add_arg (
        _request,               /* request object */
        ev,                     /* environment object */
        _argname,               /* argument name */
        _argtype,               /* argument type */
        _argval,                /* argument value */
        _arglen,                /* argument length */
        _argflags);             /* argument flags */
    }

  if (_status == OBB_SUCCESS)
    {
    _argname = "outdstr";
    _argflags = CORBA_ARG_OUT;
    _argval = (CORBA_void *) outdstr;
    _arglen = sizeof (SCOPD_DataStr);
    _argtype = TC_SCOPD_DataStr;

    _status = CORBA_Request_add_arg (
        _request,               /* request object */
        ev,                     /* environment object */
        _argname,               /* argument name */
        _argtype,               /* argument type */
        _argval,                /* argument value */
        _arglen,                /* argument length */
        _argflags);             /* argument flags */
    }

  /* Invoke method, if the list was successfully created. */

  if (_status == OBB_SUCCESS)
    {
    _status = CORBA_Request_invoke (
        _request,               /* request object */
        ev,                     /* environment struct */
        _invoke_flags);         /* invoke flags */
    }


  /* Free the context object. */

  CORBA_Context_delete( _ctx, NULL, (CORBA_Flags)0 );
  /* Free the request object. */

  CORBA_Request_delete ( _request, NULL );

  return;
}
```

TYPE.CPP

```
/*****************************************************************
 * Created Wed Apr 17 09:58:43 1996 by OBB V2.5A-04 (COMPILE/GENERATE)
 *****************************************************************
 */

/*
 * OBB Typecodes and Typecode routines
 * -----------------------------------
 *
 * This module contains Typecodes and Typecode routines for the following
 * Interfaces:
 *
 *    COP
 *
 * WARNING: The source code in this module was generated by Digital
 * Equipment Corporation's(DEC) ObjectBroker. Digital will not support any
 * changes to the source code in this module.
 *
 * How This Module Works
 * ---------------------
 *
 * For a detailed description of the files generated by ObjectBroker, refer
 * to the gen_info.txt file in the ObjectBroker example directory.
 */
//#include "stdafx.h"
#include "scopd.h"
/*
 * The following macro definitions are provided in order to allow the
 * tables and global variables to be included as SHARED READONLY in a
 * shareable image. To enable this, define ORB_SHARED_READONLY. A PSECT
 * name may be specified by defining RO_PSECT
 */

#ifndef GLOBAL_READONLY
#if !defined(ORB_SHARED_READONLY) || !defined(vms)
#define GLOBAL_READONLY
#else
#ifndef RO_PSECT
#define RO_PSECT
#endif
#define GLOBAL_READONLY globaldef RO_PSECT readonly
#endif
#endif

#ifndef STATIC_READONLY
#if !defined(ORB_SHARED_READONLY) || !defined(vms)
#define STATIC_READONLY static
#else
#define STATIC_READONLY static readonly
#endif
#endif


#ifndef GLOBAL_VARIABLE
#if !defined(ORB_SHARED_READONLY) || !defined(vms)
#define GLOBAL_VARIABLE
#else
#ifndef RW_PSECT
#define RW_PSECT
#endif
#define GLOBAL_VARIABLE globaldef RW_PSECT
#endif
#endif
```

```
GLOBAL_READONLY CORBA_unsigned_long TC_31744243_0 [] = {
        OBBLONG(0x0d, 0x11, 0x09, 0xbb),
        OBBLONG(0x11, 0x8d, 0x52, 0x74), OBBLONG(0x80, 0x06, 0x00, 0x00),
        OBBLONG(0x02, 0x10, 0xa1, 0x40), OBBLONG(0x59, 0x00, 0x00, 0x00),
        OBBLONG(0x68, 0x00, 0x00, 0x00), OBBLONG(0xff, 0xff, 0xff, 0xff),
        OBBLONG(0x01, 0x00, 0xff, 0xff), OBBLONG(0x03, 0x00, 0x00, 0x00),
        OBBLONG(0x03, 0x00, 0x00, 0x00), OBBLONG(0x08, 0x83, 0xaf, 0xe1),
        OBBLONG(0x1f, 0x5d, 0xc9, 0x11), OBBLONG(0x91, 0xa4, 0x08, 0x00),
        OBBLONG(0x2b, 0x14, 0xa0, 0xfa), OBBLONG(0x00, 0x00, 0x00, 0x00),
        OBBLONG(0x04, 0x00, 0x00, 0x00), OBBLONG(0xcc, 0x00, 0x00, 0x00),
        OBBLONG(0xff, 0xff, 0xff, 0xff), OBBLONG(0xff, 0xff, 0xff, 0xff),
        OBBLONG(0xff, 0xff, 0xff, 0xff), OBBLONG(0xff, 0xff, 0xff, 0xff),
        OBBLONG(0x01, 0x00, 0x00, 0x00), OBBLONG(0x00, 0x00, 0x00, 0x00),
        OBBLONG(0x02, 0x00, 0x00, 0x00), OBBLONG(0x01, 0x00, 0x00, 0x00),
        OBBLONG(0x58, 0x00, 0x00, 0x00), OBBLONG(0x00, 0x00, 0x00, 0x00),
        OBBLONG(0xff, 0xff, 0xff, 0xff), OBBLONG(0x01, 0x00, 0x00, 0x00),
        OBBLONG(0x05, 0xfe, 0x00, 0x00), OBBLONG(0x00, 0x00, 0x00, 0x00) };

GLOBAL_READONLY CORBA_unsigned_long TC_31744243_1 [] = {
        OBBLONG(0x0d, 0x0f, 0x19, 0xbb),
        OBBLONG(0x11, 0x8d, 0x52, 0x74), OBBLONG(0x78, 0x1b, 0x00, 0x00),
        OBBLONG(0x02, 0x10, 0xa1, 0x40), OBBLONG(0x59, 0x00, 0x00, 0x00),
        OBBLONG(0x78, 0x00, 0x00, 0x00), OBBLONG(0xff, 0xff, 0xff, 0xff),
        OBBLONG(0x01, 0x00, 0xff, 0xff), OBBLONG(0x03, 0x00, 0x00, 0x00),
        OBBLONG(0x03, 0x00, 0x00, 0x00), OBBLONG(0x08, 0x83, 0xaf, 0xe1),
        OBBLONG(0x1f, 0x5d, 0xc9, 0x11), OBBLONG(0x91, 0xa4, 0x08, 0x00),
        OBBLONG(0x2b, 0x14, 0xa0, 0xfa), OBBLONG(0x00, 0x00, 0x00, 0x00),
        OBBLONG(0x04, 0x00, 0x00, 0x00), OBBLONG(0xcc, 0x00, 0x00, 0x00),
        OBBLONG(0xff, 0xff, 0xff, 0xff), OBBLONG(0xff, 0xff, 0xff, 0xff),
        OBBLONG(0xff, 0xff, 0xff, 0xff), OBBLONG(0xff, 0xff, 0xff, 0xff),
        OBBLONG(0x01, 0x00, 0x00, 0x00), OBBLONG(0x00, 0x00, 0x00, 0x00),
        OBBLONG(0x02, 0x00, 0x00, 0x00), OBBLONG(0x01, 0x00, 0x00, 0x00),
        OBBLONG(0x58, 0x00, 0x00, 0x00), OBBLONG(0x00, 0x00, 0x00, 0x00),
        OBBLONG(0xff, 0xff, 0xff, 0xff), OBBLONG(0x01, 0x00, 0x00, 0x00),
        OBBLONG(0x0f, 0x07, 0xff, 0xff), OBBLONG(0x68, 0x00, 0x00, 0x00),
        OBBLONG(0xfe, 0xff, 0xff, 0xff), OBBLONG(0x01, 0x00, 0x00, 0x00),
        OBBLONG(0x05, 0xfe, 0x00, 0x00), OBBLONG(0x00, 0x00, 0x00, 0x00),
        OBBLONG(0x14, 0x00, 0x00, 0x00), 0x00000000, sizeof(SCOPD_UserExcep)
        , (CORBA_unsigned_long) (((CORBA_octet*)&((SCOPD_UserExcep * ) NULL)->Reason-
(CORBA_octet*)0))
        , 0x00000000 };

GLOBAL_READONLY CORBA_unsigned_long TC_31744243_2 [] = {
        OBBLONG(0x0d, 0x0f, 0x19, 0xbb),
        OBBLONG(0x11, 0x8d, 0x52, 0x74), OBBLONG(0x54, 0x1e, 0x00, 0x00),
        OBBLONG(0x02, 0x10, 0xa1, 0x40), OBBLONG(0x59, 0x00, 0x00, 0x00),
        OBBLONG(0x8c, 0x00, 0x00, 0x00), OBBLONG(0xff, 0xff, 0xff, 0xff),
        OBBLONG(0x01, 0x00, 0xff, 0xff), OBBLONG(0x03, 0x00, 0x00, 0x00),
        OBBLONG(0x03, 0x00, 0x00, 0x00), OBBLONG(0x08, 0x83, 0xaf, 0xe1),
        OBBLONG(0x1f, 0x5d, 0xc9, 0x11), OBBLONG(0x91, 0xa4, 0x08, 0x00),
        OBBLONG(0x2b, 0x14, 0xa0, 0xfa), OBBLONG(0x00, 0x00, 0x00, 0x00),
        OBBLONG(0x04, 0x00, 0x00, 0x00), OBBLONG(0xcc, 0x00, 0x00, 0x00),
        OBBLONG(0xff, 0xff, 0xff, 0xff), OBBLONG(0xff, 0xff, 0xff, 0xff),
        OBBLONG(0xff, 0xff, 0xff, 0xff), OBBLONG(0xff, 0xff, 0xff, 0xff),
        OBBLONG(0x01, 0x00, 0x00, 0x00), OBBLONG(0x00, 0x00, 0x00, 0x00),
        OBBLONG(0x02, 0x00, 0x00, 0x00), OBBLONG(0x01, 0x00, 0x00, 0x00),
        OBBLONG(0x58, 0x00, 0x00, 0x00), OBBLONG(0x00, 0x00, 0x00, 0x00),
        OBBLONG(0xff, 0xff, 0xff, 0xff), OBBLONG(0x01, 0x00, 0x00, 0x00),
        OBBLONG(0x0f, 0x07, 0xff, 0xff), OBBLONG(0x68, 0x00, 0x00, 0x00),
        OBBLONG(0xfe, 0xff, 0xff, 0xff), OBBLONG(0x01, 0x00, 0x00, 0x00),
        OBBLONG(0x28, 0x56, 0x09, 0x04), OBBLONG(0x58, 0xff, 0xff, 0xff),
        OBBLONG(0x7c, 0x00, 0x00, 0x00), OBBLONG(0xfe, 0xff, 0xff, 0xff),
```

```
        OBBLONG(0x06, 0x00, 0x00, 0x00), OBBLONG(0x04, 0x00, 0x00, 0x00),
        OBBLONG(0x00, 0x00, 0x00, 0x00),
        OBBLONG(0x24, 0x00, 0x00, 0x00),  0x00000000,  sizeof(SCOPD_DataStr)
        , (CORBA_unsigned_long) (((CORBA_octet*)&((SCOPD_DataStr * ) NULL)->stringdat
a-(CORBA_octet*)0))
        , (CORBA_unsigned_long) (((CORBA_octet*)&((SCOPD_DataStr * ) NULL)->longdata-
(CORBA_octet*)0))
        , (CORBA_unsigned_long) (((CORBA_octet*)&((SCOPD_DataStr * ) NULL)->doubledat
a-(CORBA_octet*)0))
        , (CORBA_unsigned_long) (((CORBA_octet*)&((SCOPD_DataStr * ) NULL)->dseqdata-
(CORBA_octet*)0))
        , 0x00000000
        , 0x00000000 };


GLOBAL_READONLY struct {
        CORBA_char    Bucket0 [75];     CORBA_char    Bucket1 [38];
    } _TC_scopd_BT = {
        "74528d110680.02.10.a1.40.59.00.00.00¥00074528d111b78.02.10.a1.40.59.00.00.00
¥000",
        "74528d111e54.02.10.a1.40.59.00.00.00¥000"
    };

GLOBAL_READONLY OBBHashEntry      _TC_scopd_HT [3] = {
    {   0,      0 },     {   2,     75 },     {  -1,  65535 }
    };

CORBA_TypeCode _TC_scopd_HashLookup( CORBA_short HashIndex)
 {

 if (HashIndex < 0)
    return(OBB__ConvertKindToTypeCode(
                (CORBA_Environment *) NULL,
                (CORBA_short) (- HashIndex)));

 switch (HashIndex)
        {
         case _TC_HASH_SCOPD_ERR_TYPES :
                return(TC_31744243_0);

         case _TC_HASH_SCOPD_UserExcep :
                return(TC_31744243_1);

         case _TC_HASH_SCOPD_DataStr :
                return(TC_31744243_2);

         default:
                break;
        }
 return ((CORBA_TypeCode) CORBA_OBJECT_NIL);
 }

CORBA_TypeCode OBB_EXPORT _TC_scopd_LookupByID( CORBA_string TypeCodeID)
 {
 CORBA_short hash_index = -1;

 hash_index = OBB__MethodHash(
        TypeCodeID, 3,
        _TC_scopd_HT,
        (CORBA_string) &_TC_scopd_BT);

 return ( _TC_scopd_HashLookup( hash_index ) );
```

TYPE.CPP

```
}
```

```
/*****************************************************************
 * Created Wed Apr 17 09:58:43 1996 by OBB V2.5A-04 (COMPILE/GENERATE)
 *****************************************************************
 */

/*
 * OBB Typecodes and Typecode routines
 * -----------------------------------
 *
 * This module contains Typecodes and Typecode routines for the following
 * Interfaces:
 *
 *    COP
 *
 * WARNING: The source code in this module was generated by Digital
 * Equipment Corporation's(DEC) ObjectBroker. Digital will not support any
 * changes to the source code in this module.
 *
 * How This Module Works
 * ---------------------
 *
 * For a detailed description of the files generated by ObjectBroker, refer
 * to the gen_info.txt file in the ObjectBroker example directory.
 */
#ifndef _scopd_TCH
#define _scopd_TCH
#ifdef __cplusplus
extern "C" {
#endif
#include <stdio.h>
#include <string.h>
#include <orb.h>

/*
 * The following macro definitions are provided in order to allow the
 * tables and global variables to be available to other modules.
 */

#ifndef ORB__EXTERN
#if !defined(ORB_SHARED_READONLY) || !defined(vms)
#define ORB__EXTERN extern
#else
#define ORB__EXTERN globalref
#endif
#endif


#define TC_SCOPD_ERR_TYPES TC_31744243_0
#define TC_SCOPD_UserExcep TC_31744243_1
#define TC_SCOPD_DoubleSeq TC_sequence_double
#define TC_SCOPD_COP TC_Object
#define TC_SCOPD_DataStr TC_31744243_2

#define _TC_HASH_SCOPD_ERR_TYPES 0
#define _TC_HASH_SCOPD_UserExcep 1
#define _TC_HASH_SCOPD_DoubleSeq -38
#define _TC_HASH_SCOPD_COP -14
#define _TC_HASH_SCOPD_DataStr 2

ORB__EXTERN CORBA_unsigned_long TC_31744243_0 [];
ORB__EXTERN CORBA_unsigned_long TC_31744243_1 [];
ORB__EXTERN CORBA_unsigned_long TC_31744243_2 [];
```

```
CORBA_TypeCode _TC_scopd_HashLookup( CORBA_short HashIndex);

CORBA_TypeCode OBB_EXPORT _TC_scopd_LookupByID( CORBA_string TypeCodeID);

#ifdef __cplusplus
}
#endif
#endif
```

RESOURCE.H

```
//{{NO_DEPENDENCIES}}
// Microsoft Visual C++ generated include file.
// Used by SCOPD.RC
//

// 次に示されているデフォルトの値は新しいオブジェクトのためのものです
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS

#define _APS_NEXT_RESOURCE_VALUE        129
#define _APS_NEXT_COMMAND_VALUE         32771
#define _APS_NEXT_CONTROL_VALUE         1000
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

SCOPD.H

```
/*****************************************************************
 * Created Wed Apr 17 09:58:23 1996 by OBB V2.5A-04 (COMPILE/GENERATE)
 *****************************************************************
 */

/*
 * OBB Client Stubs include file
 * -----------------------------
 *
 * This module contains definitions and prototypes for the client stub
 * routines for the following Interfaces:
 *
 *    COP
 *
 * WARNING: The source code in this module was generated by Digital
 * Equipment Corporation's(DEC) ObjectBroker. Digital will not support any
 * changes to the source code in this module.
 *
 * How This Module Works
 * ---------------------
 *
 * For a detailed description of the files generated by ObjectBroker, refer
 * to the gen_info.txt file in the ObjectBroker example directory.
 */
#ifndef _scopd_H
#define _scopd_H
#ifdef __cplusplus
extern "C" {
#endif
#include <stdio.h>
#include <string.h>
#include <orb.h>

#include "scopd.tch"

typedef CORBA_Object SCOPD_COP;
typedef CORBA_unsigned_long SCOPD_ERR_TYPES;
#define SCOPD_ErrCopExec 1
#define SCOPD_ErrSumyExec 2
#define SCOPD_ErrCopRunning 3
#define SCOPD_ErrGetReslt 4
typedef struct SCOPD_UserExcep {
        SCOPD_ERR_TYPES Reason;
        } SCOPD_UserExcep;
#define ex_SCOPD_UserExcep "74528d111b78.02.10.a1.40.59.00.00.00"
typedef CORBA_sequence_double SCOPD_DoubleSeq;
typedef struct SCOPD_DataStr {
        CORBA_string stringdata;
        CORBA_long longdata;
        CORBA_double doubledata;
        SCOPD_DoubleSeq dseqdata;
        } SCOPD_DataStr;
 void SCOPD_COP_Exec (SCOPD_COP object, CORBA_Environment * ev,
    CORBA_string instring,
    CORBA_long inlong,
    CORBA_double indouble,
    SCOPD_DoubleSeq * indseq,
    SCOPD_DataStr * indstr,
    CORBA_string * outstring,
    CORBA_long * outlong,
    CORBA_double * outdouble,
    SCOPD_DoubleSeq * outdseq,
    SCOPD_DataStr * outdstr);
```

SCOPD.H

```
#ifdef __cplusplus
}
#endif
#endif
```

SCOPD.MAK

```
# Microsoft Visual C++ Generated NMAKE File, Format Version 2.00
# ** DO NOT EDIT **

# TARGTYPE "Win32 (x86) Dynamic-Link Library" 0x0102

!IF "$(CFG)" == ""
CFG=Win32 Debug
!MESSAGE ｺﾝﾌｨｸﾞﾚｰｼｮﾝが指定されていません。ﾃﾞﾌｫﾙﾄの Win32 Debug を設定します。
!ENDIF

!IF "$(CFG)" != "Win32 Release" && "$(CFG)" != "Win32 Debug"
!MESSAGE 指定された ﾋﾞﾙﾄﾞ ﾓｰﾄﾞ "$(CFG)" は正しくありません。
!MESSAGE ｺﾏﾝﾄﾞ ﾗｲﾝ上でﾏｸﾛ 'CFG' を定義することによって
!MESSAGE NMAKE 実行時にﾋﾞﾙﾄﾞ ﾓｰﾄﾞを指定できます。例えば:
!MESSAGE
!MESSAGE NMAKE /f "scopd.mak" CFG="Win32 Debug"
!MESSAGE
!MESSAGE 選択可能なﾋﾞﾙﾄﾞ ﾓｰﾄﾞ:
!MESSAGE
!MESSAGE "Win32 Release" ("Win32 (x86) Dynamic-Link Library" 用)
!MESSAGE "Win32 Debug" ("Win32 (x86) Dynamic-Link Library" 用)
!MESSAGE
!ERROR 無効なｺﾝﾌｨｸﾞﾚｰｼｮﾝが指定されています。
!ENDIF

################################################################################
# Begin Project
# PROP Target_Last_Scanned "Win32 Debug"
MTL=MkTypLib.exe
CPP=cl.exe
RSC=rc.exe

!IF  "$(CFG)" == "Win32 Release"

# PROP BASE Use_MFC 0
# PROP BASE Use_Debug_Libraries 0
# PROP BASE Output_Dir "WinRel"
# PROP BASE Intermediate_Dir "WinRel"
# PROP Use_MFC 0
# PROP Use_Debug_Libraries 0
# PROP Output_Dir "WinRel"
# PROP Intermediate_Dir "WinRel"
OUTDIR=.¥WinRel
INTDIR=.¥WinRel

ALL : $(OUTDIR)/scopd.dll $(OUTDIR)/scopd.bsc

$(OUTDIR) :
    if not exist $(OUTDIR)/nul mkdir $(OUTDIR)

# ADD BASE MTL /nologo /D "NDEBUG" /win32
# ADD MTL /nologo /D "NDEBUG" /win32
MTL_PROJ=/nologo /D "NDEBUG" /win32
# ADD BASE CPP /nologo /MT /W3 /GX /YX /O2 /D "WIN32" /D "NDEBUG" /D "_WINDOWS" /FR /
c
# ADD CPP /nologo /MT /W3 /GX /YX /O2 /D "WIN32" /D "NDEBUG" /D "_WINDOWS" /FR /c
CPP_PROJ=/nologo /MT /W3 /GX /YX /O2 /D "WIN32" /D "NDEBUG" /D "_WINDOWS"¥
 /FR$(INTDIR)/ /Fp$(OUTDIR)/"scopd.pch" /Fo$(INTDIR)/ /c
CPP_OBJS=.¥WinRel/
# ADD BASE RSC /l 0x411 /d "NDEBUG"
# ADD RSC /l 0x411 /d "NDEBUG"
BSC32=bscmake.exe
# ADD BASE BSC32 /nologo
```

```
# ADD BSC32 /nologo
BSC32_FLAGS=/nologo /o$(OUTDIR)/"scopd.bsc"
BSC32_SBRS= ¥
        $(INTDIR)/stub.sbr ¥
        $(INTDIR)/type.sbr ¥
        $(INTDIR)/chkcop.sbr ¥
        $(INTDIR)/main.sbr

$(OUTDIR)/scopd.bsc : $(OUTDIR)  $(BSC32_SBRS)
    $(BSC32) @<<
  $(BSC32_FLAGS) $(BSC32_SBRS)
<<

LINK32=link.exe
# ADD BASE LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi3
2.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib /NOLOGO /SU
BSYSTEM:windows /DLL /MACHINE:I386
# ADD LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib
 shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib obb.lib obbcos.l
ib /NOLOGO /SUBSYSTEM:windows /DLL /MACHINE:I386
LINK32_FLAGS=kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib¥
 advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib¥
 odbccp32.lib obb.lib obbcos.lib /NOLOGO /SUBSYSTEM:windows /DLL /INCREMENTAL:no¥
 /PDB:$(OUTDIR)/"scopd.pdb" /MACHINE:I386 /DEF:".¥scopd.def"¥
 /OUT:$(OUTDIR)/"scopd.dll" /IMPLIB:$(OUTDIR)/"scopd.lib"
DEF_FILE=.¥scopd.def
LINK32_OBJS= ¥
        $(INTDIR)/stub.obj ¥
        $(INTDIR)/type.obj ¥
        $(INTDIR)/chkcop.obj ¥
        $(INTDIR)/main.obj

$(OUTDIR)/scopd.dll : $(OUTDIR)  $(DEF_FILE) $(LINK32_OBJS)
    $(LINK32) @<<
  $(LINK32_FLAGS) $(LINK32_OBJS)
<<

!ELSEIF  "$(CFG)" == "Win32 Debug"

# PROP BASE Use_MFC 0
# PROP BASE Use_Debug_Libraries 1
# PROP BASE Output_Dir "WinDebug"
# PROP BASE Intermediate_Dir "WinDebug"
# PROP Use_MFC 0
# PROP Use_Debug_Libraries 1
# PROP Output_Dir "WinDebug"
# PROP Intermediate_Dir "WinDebug"
OUTDIR=.¥WinDebug
INTDIR=.¥WinDebug

ALL : $(OUTDIR)/scopd.dll $(OUTDIR)/scopd.bsc

$(OUTDIR) :
    if not exist $(OUTDIR)/nul mkdir $(OUTDIR)

# ADD BASE MTL /nologo /D "_DEBUG" /win32
# ADD MTL /nologo /D "_DEBUG" /win32
MTL_PROJ=/nologo /D "_DEBUG" /win32
# ADD BASE CPP /nologo /MT /W3 /GX /Zi /YX /Od /D "WIN32" /D "_DEBUG" /D "_WINDOWS" /
FR /c
# ADD CPP /nologo /MT /W3 /GX /Zi /YX /Od /D "WIN32" /D "_DEBUG" /D "_WINDOWS" /FR /c
CPP_PROJ=/nologo /MT /W3 /GX /Zi /YX /Od /D "WIN32" /D "_DEBUG" /D "_WINDOWS"¥
 /FR$(INTDIR)/ /Fp$(OUTDIR)/"scopd.pch" /Fo$(INTDIR)/ /Fd$(OUTDIR)/"scopd.pdb"¥
```

```
 /c
CPP_OBJS=.¥WinDebug/
# ADD BASE RSC /l 0x411 /d "_DEBUG"
# ADD RSC /l 0x411 /d "_DEBUG"
BSC32=bscmake.exe
# ADD BASE BSC32 /nologo
# ADD BSC32 /nologo
BSC32_FLAGS=/nologo /o$(OUTDIR)/"scopd.bsc"
BSC32_SBRS= ¥
        $(INTDIR)/stub.sbr ¥
        $(INTDIR)/type.sbr ¥
        $(INTDIR)/chkcop.sbr ¥
        $(INTDIR)/main.sbr

$(OUTDIR)/scopd.bsc : $(OUTDIR)  $(BSC32_SBRS)
    $(BSC32) @<<
  $(BSC32_FLAGS) $(BSC32_SBRS)
<<

LINK32=link.exe
# ADD BASE LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi3
2.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib /NOLOGO /SU
BSYSTEM:windows /DLL /DEBUG /MACHINE:I386
# ADD LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib
 shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib obb.lib obbcos.l
ib /NOLOGO /SUBSYSTEM:windows /DLL /DEBUG /MACHINE:I386
LINK32_FLAGS=kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib¥
 advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib¥
 odbccp32.lib obb.lib obbcos.lib /NOLOGO /SUBSYSTEM:windows /DLL¥
 /INCREMENTAL:yes /PDB:$(OUTDIR)/"scopd.pdb" /DEBUG /MACHINE:I386¥
 /DEF:".¥scopd.def" /OUT:$(OUTDIR)/"scopd.dll" /IMPLIB:$(OUTDIR)/"scopd.lib"
DEF_FILE=.¥scopd.def
LINK32_OBJS= ¥
        $(INTDIR)/stub.obj ¥
        $(INTDIR)/type.obj ¥
        $(INTDIR)/chkcop.obj ¥
        $(INTDIR)/main.obj

$(OUTDIR)/scopd.dll : $(OUTDIR)  $(DEF_FILE) $(LINK32_OBJS)
    $(LINK32) @<<
  $(LINK32_FLAGS) $(LINK32_OBJS)
<<

!ENDIF

.c{$(CPP_OBJS)}.obj:
   $(CPP) $(CPP_PROJ) $<

.cpp{$(CPP_OBJS)}.obj:
   $(CPP) $(CPP_PROJ) $<

.cxx{$(CPP_OBJS)}.obj:
   $(CPP) $(CPP_PROJ) $<

################################################################################
# Begin Group "ソース ファイル"

################################################################################
# Begin Source File

SOURCE=.¥stub.cpp
DEP_STUB_=¥
        .¥scopd.h¥
```

```
        C:¥win32app¥obroker¥include¥orb.h

$(INTDIR)/stub.obj :  $(SOURCE)  $(DEP_STUB_) $(INTDIR)

# End Source File
################################################################################
# Begin Source File

SOURCE=.¥type.cpp
DEP_TYPE_=¥
        .¥scopd.h¥
        C:¥win32app¥obroker¥include¥orb.h

$(INTDIR)/type.obj :  $(SOURCE)  $(DEP_TYPE_) $(INTDIR)

# End Source File
################################################################################
# Begin Source File

SOURCE=.¥chkcop.cpp
DEP_CHKCO=¥
        C:¥win32app¥obroker¥include¥namesvc.h¥
        .¥scopd.h¥
        C:¥win32app¥obroker¥include¥orb.h

$(INTDIR)/chkcop.obj :  $(SOURCE)  $(DEP_CHKCO) $(INTDIR)

# End Source File
################################################################################
# Begin Source File

SOURCE=.¥scopd.def
# End Source File
################################################################################
# Begin Source File

SOURCE=.¥main.cpp

$(INTDIR)/main.obj :  $(SOURCE)  $(INTDIR)

# End Source File
# End Group
# End Project
################################################################################
```

SCOPD.DEF

```
; scopd.def : DLL 用のモジュール パラメータ宣言

LIBRARY      SCOPD
DESCRIPTION  'SCOPD Windows Dynamic Link Library'

EXPORTS
    ; 明示的なエクスポートはここへ記述できます
        SyncCopExec
        AsyncCopExec
        PollMessage
        CopGetResult
        FreeOutSeq
        SyncCopExecTest
        InitDouble
        FreeData
```

付録B　ソースコード No. 2 ／ サーバー側プログラム（FORTRAN言語）

SRV_MAIN.C

```
#include "scopd.imh"
#include <stdio.h>
#include <stdlib.h>
/* #include <ctype.h> */
#include <string.h>

#define MAXLINELEN 1024
#define STR_OBJ_FILNAM "./cop_obj.ref"
#define COP_TOP "COP_TOP"
#define COP_PRG "COP_PRG"
#define SUMY_PRG "SUMY_PRG"

#ifndef TRUE
#define TRUE 1
#endif

#ifndef FALSE
#define FALSE 0
#endif

char *cop_top_dir=NULL;
char *g_sumy_prg=NULL;
char *g_cop_prg=NULL;
```

SRV_MAIN.C

```
/* function prototypes */

/* server startup routines */
void RegisterImpls          (CORBA_Environment *ev,
                             CORBA_ImplementationDef *COPImpl);

int CreateObjRef           (CORBA_Environment *ev,
                            CORBA_ImplementationDef *COPImpl,
                            CORBA_Object *COPObjRef);

void StopServer();

/* ObjectBroker error handler */
void ReportOBError          (CORBA_Environment *ev,
                             char *CallName,
                             char *RoutineName);
```

SRV_MAIN.C

```
main ()
{
  CORBA_Environment ev;
  CORBA_ImplementationDef COPImpl;
  CORBA_Object COPObjRef;


  /*
  cop_top_dir = getenv(COP_TOP);
  if(cop_top_dir == NULL){
    fprintf(stderr,"COP_TOP is not defined.¥n");
  }
  */

  g_cop_prg = getenv(COP_PRG);
  if(g_cop_prg == NULL){
    fprintf(stderr,"COP_PRG (COP program file) is not defined.¥n");
    fprintf(stderr,"Data Test Only.¥n");
  }
  g_sumy_prg = getenv(SUMY_PRG);
  if(g_sumy_prg == NULL){
    fprintf(stderr,"SUMY_PRG (SUMY program file) is not defined.¥n");
    fprintf(stderr,"Data Test Only.¥n");
  }
  printf("¥n¥nSCOPD server starting¥n");

  /* Register the implementations that this server provides */
  RegisterImpls(&ev,
                &COPImpl);

  /* */
  CreateObjRef(&ev, &COPImpl,&COPObjRef);

  CORBA_BOA_impl_is_ready(
    CORBA_DEC_BOA_OBJECT,
    &ev,
    COPImpl);
  if (ev._major!=CORBA_NO_EXCEPTION)
    ReportOBError(&ev,"CORBA_BOA_impl_is_ready [COPImpl]","main");

  printf("¥nEntering loop to wait for service requests...¥n¥n");
  OBB_BOA_main_loop(
    CORBA_DEC_ORB_OBJECT,
    &ev,
    (CORBA_Flags)NULL);
  if (ev._major!=CORBA_NO_EXCEPTION)
    ReportOBError(&ev,"CORBA_BOA_main_loop","main");


  CORBA_BOA_dispose(CORBA_DEC_BOA_OBJECT,
                    &ev,
                    COPObjRef);
  OBB_BOA_impl_unregister(CORBA_DEC_BOA_OBJECT,
                          &ev,
                          COPImpl,
                          (CORBA_Flags)NULL);

   free(cop_top_dir);

}  /* end of routine main */
```

SRV_MAIN.C

```
void RegisterImpls(
  CORBA_Environment *ev,
  CORBA_ImplementationDef *COPImpl)
{
  *COPImpl =
    (CORBA_ImplementationDef)COPImpl__register(
            COPImpl__OBJ,
            ev,
            (CORBA_NVList)NULL,
            (CORBA_NVList)NULL,
            (CORBA_Flags)NULL);
  if (ev->_major!=CORBA_NO_EXCEPTION)
    ReportOBError(ev,"COPImpl__register","RegisterImpls");
  return;

}  /* end of routine RegisterImpls */
```

SRV_MAIN.C

```
int CreateObjRef            (CORBA_Environment *ev,
                               CORBA_ImplementationDef *COPImpl,
                              CORBA_Object *COPObjRef)
{
  CORBA_string COPObjString = NULL;
  CORBA_ReferenceData RefData;
  FILE *fp;
  char FileRec[MAXLINELEN];

  RefData._maximum = 0;
  RefData._length = 0;
  RefData._buffer = NULL;
  fp = fopen(STR_OBJ_FILNAM,"w");
  if (fp == NULL){
    printf("¥n Could not Create %s¥n",STR_OBJ_FILNAM);
    return FALSE;
  }

  printf("Creating initial Object Reference file %s¥n",STR_OBJ_FILNAM);

  COPObjRef = CORBA_BOA_create(
                CORBA_DEC_BOA_OBJECT,
                ev,
                &RefData,
                SCOPD_COP__OBJ,
                *COPImpl);
  if (ev->_major!=CORBA_NO_EXCEPTION)
    ReportOBError(ev,"CORBA_BOA_create [COPObjRef]","CreateObjRef");


  COPObjString = CORBA_ORB_object_to_string (
                    CORBA_DEC_ORB_OBJECT,
                    ev,
                    COPObjRef);

  fprintf(fp,"%s¥n",COPObjString);
  fclose(fp);
  CORBA_free(COPObjString);


  return TRUE;
}
```

```
/* ReportOBError

   Report an error from an ObjectBroker routine call, and exit.
*/
void ReportOBError(
  CORBA_Environment *Ev,
  char *CallName,
  char *RoutineName)
{

  printf("¥n¥nERROR return from call %s, routine %s",CallName,RoutineName);
  if (Ev->_major==CORBA_SYSTEM_EXCEPTION)
  {
    printf("¥nSystem exception encountered: %s¥n¥n¥n",
           (char *)CORBA_exception_id(Ev));
  }
  exit(0);
}  /* end of routine ReportOBError */


void StopServer()
{
  CORBA_Environment Ev;

  OBB_BOA_exit_main_loop(CORBA_DEC_ORB_OBJECT,
                         &Ev,
                         (CORBA_Flags)NULL);
  if (Ev._major!=CORBA_NO_EXCEPTION)
    ReportOBError(&Ev,"OBB_BOA_exit_main_loop","ServerStop");
  return;

}
```

CLI_MAIN.C

```
#include "scopd.h"
#include <stdio.h>
#include <stdlib.h>
#include <float.h>
#include <values.h>


int  Finished;

/* function prototypes */
int InitObject();
int   ReportExcep();
void TestExec();

char gsum[512];
```

CLI_MAIN.C

```
/* The main routine */
main ()
{
  CORBA_Environment Ev;
  int Finished = FALSE;
  int status;
  char Input[512];
  int Option;
  CORBA_Object COPObj;

  gsum[0]='¥0';

  status = InitObject(&Ev,&COPObj);
  if (status==TRUE)
  {
    {
      while (Finished==FALSE)
      {
        printf("¥nSelect a Transaction:¥n");
        printf("  1 - Execution. ¥n");
        printf("  5 - Exit¥n");
        printf("Transaction: ");
        gets(Input);
        Option = atoi(Input);
        switch (Option)
        {
           case  1: TestExec(&Ev,&COPObj); break;
           case  5: Finished=TRUE; break;
           default: break;
        }
      }
    }
  }

  CORBA_Object_release(COPObj,
                       &Ev);
  if (Ev._major!=CORBA_NO_EXCEPTION)
    ReportExcep(&Ev);

  OBB_ORB_rundown(CORBA_DEC_ORB_OBJECT,
                  &Ev,
                  (CORBA_Flags)NULL);
  if (Ev._major!=CORBA_NO_EXCEPTION)
    ReportExcep(&Ev);
}  /* end of routine main */

int InitObject(
  CORBA_Environment *ev,
  CORBA_Object *COPObj)
{
  FILE *fp;
  char FileRec[1024];

  fp = fopen("cop_obj.ref","r");
  if (fp == NULL){
    printf("Could not find cop_obj.ref.¥n");
    exit(0);
  }
  fgets(FileRec,1024,fp);
  FileRec[strlen(FileRec)-1] = '¥0';
  *COPObj = CORBA_ORB_string_to_object(
               CORBA_DEC_ORB_OBJECT,
```

```
                ev,
                FileRec);
  if (ev->_major != CORBA_NO_EXCEPTION) ReportExcep(ev);
  return TRUE;
}


int ReportExcep(
  CORBA_Environment *Ev)  /* in,  CORBA environment structure */
{
  int status;
  SCOPD_UserExcep *UserExcep;

  if (Ev->_major==CORBA_SYSTEM_EXCEPTION)
  {
    printf("¥nSystem exception encountered: %s¥n",
           (char *)CORBA_exception_id(Ev));
    printf("  %s¥n¥n¥n",
           (char *)OBB_exception_errortext(Ev,(CORBA_Flags)OBB_FORMAT_80));
    status = FALSE; /* flag a system exception */
  }
  else
  {
    UserExcep = (SCOPD_UserExcep *)CORBA_exception_value(Ev);
    switch (UserExcep->Reason)
    {
       case SCOPD_ErrCopExec:
         printf("¥nCOP command Execution Error.¥n");
         break;
       case SCOPD_ErrSumyExec:
         printf("¥nSUMY command Execution Error.¥n");
         break;
       case SCOPD_ErrCopRunning:
         printf("¥nCOP command is running now.¥n");
         break;
       case SCOPD_ErrGetReslt:
         printf("¥nGet Result error.¥n");
         break;
       default:
         printf("¥nAn unrecognized problem has occurred; problem code %u.¥n",
                UserExcep->Reason);
         break;
    }
    status = TRUE; /* not a system exception */
  }
  CORBA_exception_free(Ev);
  return(status);
}  /* end of routine ReportExcep */

void TestExec(CORBA_Environment *ev,CORBA_Object *COPObj)
{
  CORBA_double indouble;
  CORBA_double outdouble;
  CORBA_long  inlong;
  CORBA_long  outlong;
  CORBA_string instring;
  CORBA_string outstring;
  SCOPD_DataStr instruct;
  SCOPD_DataStr outstruct;
  SCOPD_DoubleSeq indseq;
  SCOPD_DoubleSeq outdseq;

  unsigned long i;
```

```
  /* Creating String Data */
  printf("Creating String Data ...¥n ");
  instring = (char *)malloc(11);
  strcpy(instring, "abcdefghij");

  /* Creating Double Data */
  printf("Creating Double Data ...¥n ");
  indouble = DBL_MAX;

  /* Creating long Data */
  printf("Creating Long Data ...¥n ");
  inlong = MAXLONG;

  /* Creating Structure Data */
  printf("Creating Structure Data ...¥n ");
  instruct.stringdata = malloc(11);
  strcpy(instruct.stringdata, "abcdefghij");
  instruct.longdata = MAXSHORT;
  instruct.doubledata = DBL_MAX;
  instruct.dseqdata._length = 10;
  instruct.dseqdata._maximum = 10;
  instruct.dseqdata._buffer = malloc(sizeof(CORBA_double)*10);
  for(i=0; i<10; i++){
    instruct.dseqdata._buffer[i] = DBL_MAX;
  }

  /* Creating Array Data */
  printf("Creating Double Seqence Data ... ¥n");
  indseq._length = 1024*1024;
  indseq._maximum = 1024*1024;
  indseq._buffer=(double *)malloc(sizeof(CORBA_double)*1024*1024);
  if(indseq._buffer == NULL){
    fprintf(stderr,"sequence data malloc error.¥n");
    exit(0);
  }
  for(i=0; i<1024*1024; i++){
    indseq._buffer[i]=DBL_MAX;
  }

/* Executing COP  */
printf("¥n¥n COP command Executing now ....¥n");
SCOPD_COP_Exec (*COPObj, ev,
    instring,
    inlong,
    indouble,
    &indseq,
    &instruct,
    &outstring,
    &outlong,
    &outdouble,
    &outdseq,
    &outstruct);

  if(ev->_major == CORBA_NO_EXCEPTION){
    printf("¥n¥n ### Normal Data. ####¥n");
    printf("out double = %e¥n",outdouble);
    printf("out string = %s¥n",outstring);
    printf("out long = %d¥n",outlong);

    printf("¥n ### Struct Data ###.¥n");
    printf("stringdata = %s¥n",outstruct.stringdata);
    printf(" longdata = %d¥n",outstruct.longdata);
```

CLI_MAIN.C

```
    printf("doubledata = %e¥n",outstruct.doubledata);
    for(i=0;i<outstruct.dseqdata._length;i++)
      printf("douleseq[%d] = %e¥n",i,outstruct.dseqdata._buffer[i]);

    printf("¥n ### Sequence Data ###.¥n");
    printf("First Sequence = %e¥n",outdseq._buffer[0]);
    printf("Last  Sequence = %e¥n",outdseq._buffer[outdseq._length-1]);
    printf("Sequence Num = %d¥n",outdseq._length);

  }


  /* free routine */
  CORBA_free(outstring);
  CORBA_free(outdseq._buffer);
  CORBA_free(outstruct.stringdata);
  CORBA_free(outstruct.dseqdata._buffer);

}
```

OBBANSI.C

```
#include <stdio.h>
#include <errno.h>

unsigned int strtoul(char* str, char**ptr, int base)
{
        unsigned int result;
        sscanf(str, "%ul", &result);
        return(result);
}

char* strerror(int errnum)
{
extern char* sys_errlist[];
extern sys_nerr;
extern int errno;

        return(sys_errlist[errnum]);
}
```

METHOD.C

```
/*****************************************************************
 * Created Mon Mar 18 14:00:52 1996 by OBB V2.5A-06 (COMPILE/GENERATE)
 *****************************************************************
 */

/*
 * Method Routines
 * ---------------
 *
 * This module contains method routine templates for the following
 * Implementations:
 *
 *    COPImpl
 *
 */

#include "scopd.imh"
#include "extern.h"

extern char * g_cop_prg;
extern char * g_sumy_prg;
```

METHOD.C

```
/*
 *
 *  ROUTINE NAME:       COPImpl_Exec
 *
 *  FUNCTIONAL DESCRIPTION:
 *
 *      Method routine for Exec.
 *       (Implementation : COPImpl)
 *
 */

 void COPImpl_Exec (CORBA_Object object, CORBA_Environment * ev,
    CORBA_string instring,
    CORBA_long inlong,
    CORBA_double indouble,
    SCOPD_DoubleSeq * indseq,
    SCOPD_DataStr * indstr,
    CORBA_string * outstring,
    CORBA_long * outlong,
    CORBA_double * outdouble,
    SCOPD_DoubleSeq * doutdseq,
    SCOPD_DataStr * outdstr)

{
  OBB_memptr ArgMem = NULL;
  int status;
  SCOPD_UserExcep *ExcepParam;

  change_data(ev, &ArgMem, instring, inlong, indouble, indseq,
              indstr, outstring, outlong, outdouble, doutdseq,
              outdstr);

   /* invoke COP command */
   if(g_cop_prg != NULL && g_cop_prg != NULL){
        status = system(g_cop_prg);
        if(status != 0){
          fprintf(stderr,"COP command execution status error.¥n");
          ExcepParam = (SCOPD_UserExcep *)OBBarg_malloc(
                                          ev,
                                          &ArgMem,
                                          sizeof(SCOPD_UserExcep));
          if(ExcepParam == NULL){
                fprintf(stderr,"Exception area malloc error.¥n");
                StopServer();
          }
          ExcepParam->Reason = SCOPD_ErrCopExec;
          CORBA_BOA_set_exception(
                CORBA_DEC_BOA_OBJECT,
                ev,
                CORBA_USER_EXCEPTION,
                ex_SCOPD_UserExcep,
                (void *)ExcepParam);
        }

        status = system(g_sumy_prg);
        if(status != 0){
          fprintf(stderr,"SUMY command execution status error.¥n");
          ExcepParam = (SCOPD_UserExcep *)OBBarg_malloc(
                                          ev,
                                          &ArgMem,
                                          sizeof(SCOPD_UserExcep));
          if(ExcepParam == NULL){
```

```
                fprintf(stderr,"Exception area malloc error.¥n");
                StopServer();
            }
            ExcepParam->Reason = SCOPD_ErrCopExec;
            CORBA_BOA_set_exception(
                CORBA_DEC_BOA_OBJECT,
                ev,
                CORBA_USER_EXCEPTION,
                ex_SCOPD_UserExcep,
                (void *)ExcepParam);
        }

    }

return;
}
```

CHDATA.C

```
#include "scopd.imh"
/* #include "extern.h" */
/*
  Prototype Definition
*/
void change_data(
    CORBA_Environment *,
    OBB_memptr *,
    CORBA_string,
    CORBA_long,
    CORBA_double,
    SCOPD_DoubleSeq *,
    SCOPD_DataStr *,
    CORBA_string *,
    CORBA_long *,
    CORBA_double *,
    SCOPD_DoubleSeq *,
    SCOPD_DataStr *);

void change_string_data(
  CORBA_Environment *,
  OBB_memptr *,
  CORBA_string,
  CORBA_string *);

void change_long_data(
  CORBA_Environment *,
  OBB_memptr *,
  CORBA_long,
  CORBA_long *);


void change_double_data(
  CORBA_Environment *,
  OBB_memptr *,
  CORBA_double,
  CORBA_double *);

void change_dseq_data(
  CORBA_Environment *,
  OBB_memptr *,
  SCOPD_DoubleSeq *,
  SCOPD_DoubleSeq *);

void change_datastr_data(
  CORBA_Environment *,
  OBB_memptr *,
  SCOPD_DataStr *,
  SCOPD_DataStr *);


/*
  ﾆﾎﾏｰ　・・ﾐﾎﾏｰ　・ﾋﾊﾑｴｹ､ｹ､
*/
void change_data(
    CORBA_Environment *ev,
    OBB_memptr *ArgMem,
    CORBA_string instring,
    CORBA_long inlong,
    CORBA_double indouble,
    SCOPD_DoubleSeq * indseq,
    SCOPD_DataStr * indstr,
    CORBA_string * outstring,
```

```
    CORBA_long * outlong,
    CORBA_double * outdouble,
    SCOPD_DoubleSeq * outdseq,
    SCOPD_DataStr * outdstr)
{
    change_string_data(ev, ArgMem,instring, outstring);
    change_long_data(ev, ArgMem, inlong, outlong);
    change_double_data(ev, ArgMem, indouble, outdouble);
    change_dseq_data(ev, ArgMem, indseq, outdseq);
    change_datastr_data(ev, ArgMem, indstr, outdstr);
}

void change_string_data(
  CORBA_Environment *ev,
  OBB_memptr *ArgMem,
  CORBA_string instring,
  CORBA_string *outstring)
{
  unsigned long i;

  /* Allocate outstring memory */
  *outstring=(CORBA_string)OBBarg_malloc(ev,ArgMem, strlen(instring)+1);
  if(*outstring == NULL){
    fprintf(stderr,"OBBarg_malloc error.¥n");
    return;
  }
  /* copy data */
  for(i=0; i<strlen(instring); i++){
    (*outstring)[i] = instring[i];
  }
  (*outstring)[strlen(instring)]='¥0';

  /*
  for(i=0; i<strlen(instring); i++){
    (*outstring)[i] = instring[i] + 1;
  }
  (*outstring)[strlen(instring)]='¥0';
  */
}

void change_long_data(
  CORBA_Environment *ev,
  OBB_memptr *ArgMem,
  CORBA_long inlong,
  CORBA_long *outlong)
{
   *outlong=inlong-1;
}

void change_double_data(
  CORBA_Environment *ev,
  OBB_memptr *ArgMem,
  CORBA_double indouble,
  CORBA_double *outdouble)
{
  *outdouble = indouble - 1.0;
}

void change_dseq_data(
  CORBA_Environment *ev,
  OBB_memptr *ArgMem,
  SCOPD_DoubleSeq *indseq,
  SCOPD_DoubleSeq *outdseq)
```

```
{
  unsigned long i;

  outdseq->_length = indseq->_length;
  outdseq->_maximum = indseq->_maximum;

  outdseq->_buffer=(CORBA_double *)OBBarg_malloc(ev, ArgMem,
                                  sizeof(CORBA_double)*indseq->_length);
  for(i=0; i<indseq->_length; i++){
    outdseq->_buffer[i]=indseq->_buffer[i]-1.0;
  }
}


void change_datastr_data(
  CORBA_Environment *ev,
  OBB_memptr *ArgMem,
  SCOPD_DataStr *indstr,
  SCOPD_DataStr *outdstr)
{
  change_string_data(ev, ArgMem, indstr->stringdata, &outdstr->stringdata);
  change_long_data(ev, ArgMem, indstr->longdata, &outdstr->longdata);
  change_double_data(ev, ArgMem, indstr->doubledata,
                     &outdstr->doubledata);
  change_dseq_data(ev, ArgMem, &indstr->dseqdata, &outdstr->dseqdata);
}
```

DISP.C

```
/*****************************************************************
 * Created Thu Apr 25 09:51:24 1996 by OBB V2.5A-04 (COMPILE/GENERATE)
 *****************************************************************
 */

/*
 * OBB Dispatcher/Registration routines
 * ------------------------------------
 *
 * This module contains dispatcher/registration routines for the following
 * Implementations:
 *
 *    COPImpl
 *
 * WARNING: The source code in this module was generated by Digital
 * Equipment Corporation's(DEC) ObjectBroker. Digital will not support any
 * changes to the source code in this module.
 *
 * How This Module Works
 * ---------------------
 *
 * For a detailed description of the files generated by ObjectBroker, refer
 * to the gen_info.txt file in the ObjectBroker example directory.
 */
#include <stdio.h>
#include <string.h>
#ifndef ORB_MAKEPROC
#if ((defined(MSDOS) || defined(__MSDOS__)) && !(defined(_WINDLL) || defined(__DLL__)))

#include <windows.h>
/* If Windows, you must declare a global variable ORB__hInstance and
   initialize it to the HANDLE passed into WinMain. This is used in the
   call to MakeProcInstance. */
extern HANDLE ORB__hInstance;
#define ORB_MAKEPROC(rtn)  MakeProcInstance ((FARPROC) (rtn), ORB__hInstance)
#else
/* Stubs will reside in DLL, or not under windows */
#define ORB_MAKEPROC(rtn)  (rtn)
#endif
#endif
#include <orb.h>

/*
 * The following macro definitions are provided in order to allow the
 * tables and global variables to be included as SHARED READONLY in a
 * shareable image. To enable this, define ORB_SHARED_READONLY. A PSECT
 * name may be specified by defining RO_PSECT
 */

#ifndef GLOBAL_READONLY
#if !defined(ORB_SHARED_READONLY) || !defined(vms)
#define GLOBAL_READONLY
#else
#ifndef RO_PSECT
#define RO_PSECT
#endif
#define GLOBAL_READONLY globaldef RO_PSECT readonly
#endif
#endif

#ifndef STATIC_READONLY
#if !defined(ORB_SHARED_READONLY) || !defined(vms)
#define STATIC_READONLY static
```

```
#else
#define STATIC_READONLY static readonly
#endif
#endif


#ifndef GLOBAL_VARIABLE
#if !defined(ORB_SHARED_READONLY) || !defined(vms)
#define GLOBAL_VARIABLE
#else
#ifndef RW_PSECT
#define RW_PSECT
#endif
#define GLOBAL_VARIABLE globaldef RW_PSECT
#endif
#endif

#include "scopd.imh"

#define TC_SCOPD_ERR_TYPES TC_317ecc8c_0
#define TC_SCOPD_UserExcep TC_317ecc8c_1
#define TC_SCOPD_DoubleSeq TC_sequence_double
#define TC_SCOPD_COP TC_Object
#define TC_SCOPD_DataStr TC_317ecc8c_2

#define _TC_HASH_SCOPD_ERR_TYPES 0
#define _TC_HASH_SCOPD_UserExcep 1
#define _TC_HASH_SCOPD_DoubleSeq -38
#define _TC_HASH_SCOPD_COP -14
#define _TC_HASH_SCOPD_DataStr 2

STATIC_READONLY CORBA_unsigned_long TC_317ecc8c_0 [] = {
        OBBLONG(0x0d, 0x11, 0x09, 0xbb),
        OBBLONG(0x11,0x8d,0x52,0x74),OBBLONG(0x80,0x06,0x00,0x00),
        OBBLONG(0x02,0x10,0xa1,0x40),OBBLONG(0x59,0x00,0x00,0x00),
        OBBLONG(0x68,0x00,0x00,0x00),OBBLONG(0xff,0xff,0xff,0xff),
        OBBLONG(0x01,0x00,0xff,0xff),OBBLONG(0x03,0x00,0x00,0x00),
        OBBLONG(0x03,0x00,0x00,0x00),OBBLONG(0x08,0x83,0xaf,0xe1),
        OBBLONG(0x1f,0x5d,0xc9,0x11),OBBLONG(0x91,0xa4,0x08,0x00),
        OBBLONG(0x2b,0x14,0xa0,0xfa),OBBLONG(0x00,0x00,0x00,0x00),
        OBBLONG(0x04,0x00,0x00,0x00),OBBLONG(0xcc,0x00,0x00,0x00),
        OBBLONG(0xff,0xff,0xff,0xff),OBBLONG(0xff,0xff,0xff,0xff),
        OBBLONG(0xff,0xff,0xff,0xff),OBBLONG(0xff,0xff,0xff,0xff),
        OBBLONG(0x01,0x00,0x00,0x00),OBBLONG(0x00,0x00,0x00,0x00),
        OBBLONG(0x02,0x00,0x00,0x00),OBBLONG(0x01,0x00,0x00,0x00),
        OBBLONG(0x58,0x00,0x00,0x00),OBBLONG(0x00,0x00,0x00,0x00),
        OBBLONG(0xff,0xff,0xff,0xff),OBBLONG(0x01,0x00,0x00,0x00),
        OBBLONG(0x05,0xfe,0x00,0x00),OBBLONG(0x00,0x00,0x00,0x00) };

STATIC_READONLY CORBA_unsigned_long TC_317ecc8c_1 [] = {
        OBBLONG(0x0d, 0x0f, 0x19, 0xbb),
        OBBLONG(0x11,0x8d,0x52,0x74),OBBLONG(0x78,0x1b,0x00,0x00),
        OBBLONG(0x02,0x10,0xa1,0x40),OBBLONG(0x59,0x00,0x00,0x00),
        OBBLONG(0x78,0x00,0x00,0x00),OBBLONG(0xff,0xff,0xff,0xff),
        OBBLONG(0x01,0x00,0xff,0xff),OBBLONG(0x03,0x00,0x00,0x00),
        OBBLONG(0x03,0x00,0x00,0x00),OBBLONG(0x08,0x83,0xaf,0xe1),
        OBBLONG(0x1f,0x5d,0xc9,0x11),OBBLONG(0x91,0xa4,0x08,0x00),
        OBBLONG(0x2b,0x14,0xa0,0xfa),OBBLONG(0x00,0x00,0x00,0x00),
        OBBLONG(0x04,0x00,0x00,0x00),OBBLONG(0xcc,0x00,0x00,0x00),
        OBBLONG(0xff,0xff,0xff,0xff),OBBLONG(0xff,0xff,0xff,0xff),
        OBBLONG(0xff,0xff,0xff,0xff),OBBLONG(0xff,0xff,0xff,0xff),
        OBBLONG(0x01,0x00,0x00,0x00),OBBLONG(0x00,0x00,0x00,0x00),
        OBBLONG(0x02,0x00,0x00,0x00),OBBLONG(0x01,0x00,0x00,0x00),
```

```
        OBBLONG(0x58, 0x00, 0x00, 0x00), OBBLONG(0x00, 0x00, 0x00, 0x00),
        OBBLONG(0xff, 0xff, 0xff, 0xff), OBBLONG(0x01, 0x00, 0x00, 0x00),
        OBBLONG(0x0f, 0x07, 0xff, 0xff), OBBLONG(0x68, 0x00, 0x00, 0x00),
        OBBLONG(0xfe, 0xff, 0xff, 0xff), OBBLONG(0x01, 0x00, 0x00, 0x00),
        OBBLONG(0x05, 0xfe, 0x00, 0x00), OBBLONG(0x00, 0x00, 0x00, 0x00),
        OBBLONG(0x00, 0x00, 0x00, 0x14),  0x00000000,  sizeof(SCOPD_UserExcep)
        , (CORBA_unsigned_long) (((CORBA_octet*)&((SCOPD_UserExcep * ) NULL)->Reason-
(CORBA_octet*)0))
        , 0x00000000 };

STATIC_READONLY CORBA_unsigned_long TC_317ecc8c_2 [] = {
        OBBLONG(0x0d,  0x0f,  0x19,  0xbb),
        OBBLONG(0x11, 0x8d, 0x52, 0x74), OBBLONG(0x54, 0x1e, 0x00, 0x00),
        OBBLONG(0x02, 0x10, 0xa1, 0x40), OBBLONG(0x59, 0x00, 0x00, 0x00),
        OBBLONG(0x8c, 0x00, 0x00, 0x00), OBBLONG(0xff, 0xff, 0xff, 0xff),
        OBBLONG(0x01, 0x00, 0xff, 0xff), OBBLONG(0x03, 0x00, 0x00, 0x00),
        OBBLONG(0x03, 0x00, 0x00, 0x00), OBBLONG(0x08, 0x83, 0xaf, 0xe1),
        OBBLONG(0x1f, 0x5d, 0xc9, 0x11), OBBLONG(0x91, 0xa4, 0x08, 0x00),
        OBBLONG(0x2b, 0x14, 0xa0, 0xfa), OBBLONG(0x00, 0x00, 0x00, 0x00),
        OBBLONG(0x04, 0x00, 0x00, 0x00), OBBLONG(0xcc, 0x00, 0x00, 0x00),
        OBBLONG(0xff, 0xff, 0xff, 0xff), OBBLONG(0xff, 0xff, 0xff, 0xff),
        OBBLONG(0xff, 0xff, 0xff, 0xff), OBBLONG(0xff, 0xff, 0xff, 0xff),
        OBBLONG(0x01, 0x00, 0x00, 0x00), OBBLONG(0x00, 0x00, 0x00, 0x00),
        OBBLONG(0x02, 0x00, 0x00, 0x00), OBBLONG(0x01, 0x00, 0x00, 0x00),
        OBBLONG(0x58, 0x00, 0x00, 0x00), OBBLONG(0x00, 0x00, 0x00, 0x00),
        OBBLONG(0xff, 0xff, 0xff, 0xff), OBBLONG(0x01, 0x00, 0x00, 0x00),
        OBBLONG(0x0f, 0x07, 0xff, 0xff), OBBLONG(0x68, 0x00, 0x00, 0x00),
        OBBLONG(0xfe, 0xff, 0xff, 0xff), OBBLONG(0x01, 0x00, 0x00, 0x00),
        OBBLONG(0x28, 0x56, 0x09, 0x04), OBBLONG(0x58, 0xff, 0xff, 0xff),
        OBBLONG(0x7c, 0x00, 0x00, 0x00), OBBLONG(0xfe, 0xff, 0xff, 0xff),
        OBBLONG(0x06, 0x00, 0x00, 0x00), OBBLONG(0x04, 0x00, 0x00, 0x00),
        OBBLONG(0x00, 0x00, 0x00, 0x00),
        OBBLONG(0x00, 0x00, 0x00, 0x24),  0x00000000,  sizeof(SCOPD_DataStr)
        , (CORBA_unsigned_long) (((CORBA_octet*)&((SCOPD_DataStr * ) NULL)->stringdat
a-(CORBA_octet*)0))
        , (CORBA_unsigned_long) (((CORBA_octet*)&((SCOPD_DataStr * ) NULL)->longdata-
(CORBA_octet*)0))
        , (CORBA_unsigned_long) (((CORBA_octet*)&((SCOPD_DataStr * ) NULL)->doubledat
a-(CORBA_octet*)0))
        , (CORBA_unsigned_long) (((CORBA_octet*)&((SCOPD_DataStr * ) NULL)->dseqdata-
(CORBA_octet*)0))
        , 0x00000000
        , 0x00000000 };


GLOBAL_READONLY struct {
        CORBA_char    Bucket0 [75];     CORBA_char    Bucket1 [38];
    } _TCS_scopd_BT = {
        "74528d110680.02.10.a1.40.59.00.00.00¥00074528d111b78.02.10.a1.40.59.00.00.00
¥000",
        "74528d111e54.02.10.a1.40.59.00.00.00¥000"
    };

GLOBAL_READONLY OBBHashEntry      _TCS_scopd_HT [3] = {
    {   0,      0 },     {   2,     75 },     {  -1, 65535 }
    };

CORBA_TypeCode _TCS_scopd_HashLookup( CORBA_short HashIndex)
 {

 if (HashIndex < 0)
    return(OBB__ConvertKindToTypeCode(
                (CORBA_Environment *) NULL,
```

```
                (CORBA_short) (- HashIndex)));

 switch (HashIndex)
        {
         case _TC_HASH_SCOPD_ERR_TYPES :
                return(TC_317ecc8c_0);

         case _TC_HASH_SCOPD_UserExcep :
                return(TC_317ecc8c_1);

         case _TC_HASH_SCOPD_DataStr :
                return(TC_317ecc8c_2);

         default:
                break;
        }
 return ((CORBA_TypeCode) CORBA_OBJECT_NIL);
 }

CORBA_TypeCode OBB_EXPORT _TCS_scopd_LookupByID( CORBA_string TypeCodeID)
 {
 CORBA_short hash_index = -1;

 hash_index = OBB__MethodHash(
        TypeCodeID, 3,
        _TCS_scopd_HT,
        (CORBA_string) &_TCS_scopd_BT);

 return ( _TCS_scopd_HashLookup( hash_index ) );

 }

/*
 * The following are Repository Constants for the implementation and
 * any interfaces that it implements. The implementation constant
 * can be used to pass to the generated registration routine. The interface
 * constants can be used to create objects with the default binding.
 */

GLOBAL_READONLY OBB_ReposConst _OBB_IMPLDEF_8ccc7e31 =
        {
        10, 18, 1, 187, 13, 214, 16, 117, 126, 190, 0, 0, 2, 133, 188, 87,
        23, 0, 0, 0, 24, 0, 0, 0
        };

GLOBAL_READONLY CORBA_ImplementationDef COPImpl__OBJ =
        (CORBA_ImplementationDef) _OBB_IMPLDEF_8ccc7e31;

GLOBAL_READONLY OBB_ReposConst _OBB_INTFDEF_8ccc7e31 =
        {
        10, 20, 1, 187, 17, 141, 82, 116, 72, 31, 0, 0, 2, 16, 161, 64,
        89, 0, 0, 0, 24, 0, 1, 0
        };

GLOBAL_READONLY CORBA_InterfaceDef SCOPD_COP__OBJ =
        (CORBA_InterfaceDef) _OBB_INTFDEF_8ccc7e31;

/*
 * The following globals are the UUID and name for each of the implementations
 * for which a dispatch routine is generated. These globals are used in the
 * generated registration routine used for registering the corresponding
 * implementation.
 */
```

DISP.C

```
GLOBAL_READONLY CORBA_char      COPIMPL_ID [] = "74528d111f48.02.10.a1.40.59.00.00.00"
;
GLOBAL_READONLY CORBA_char      COPIMPL_NAM [] = "COPImpl";
GLOBAL_READONLY CORBA_octet      COPIMPL_IO[16] = {
        17,141,82,116,72,31,0,0,2,16,161,64,89,0,0,0
        };

GLOBAL_READONLY CORBA_octet     COPIMPL_MO[3] = {
        1,0,128
        };

GLOBAL_READONLY OBBImplMapEntry  COPIMPL_IM[1] = {
        { COPIMPL_IO,  COPIMPL_MO }
    };
GLOBAL_READONLY CORBA_unsigned_short    COPIMPL_NI = 1;
GLOBAL_READONLY struct {
        CORBA_char    Bucket0 [38];    } COPIMPL_BT = {
        "74528d11203c.02.10.a1.40.59.00.00.00¥000"
    };
GLOBAL_READONLY OBBHashEntry     COPIMPL_HT [1] = {
    {   0,      0 }
    };
GLOBAL_READONLY CORBA_unsigned_long     COPIMPL_NO = 1;
GLOBAL_READONLY CORBA_unsigned_short COPIMPL_OM[1] = {
        0
    };
GLOBAL_READONLY CORBA_char      * COPIMPL_NT[1] = {
        "Exec"
    };
GLOBAL_READONLY CORBA_unsigned_long     COPIMPL_NM = 1;
GLOBAL_READONLY OBBMethodRtnContextEntry  COPIMPL_MC[1] = {
        { 0,  0 }
    };

GLOBAL_VARIABLE OBBImplementationData COPIMPL_IPD;

/* Get method args routine prototype. */

CORBA_NVList OBB_EXPORT
COPImpl__getmthargs (

OBB_BinaryID            ImplId,
CORBA_Environment       * Ev,
OBB_OpInfo              * OpInfo,
CORBA_NamedValue        * Result,
CORBA_Flags             Flags);

GLOBAL_READONLY CORBA_short COPIMPL_AO[1] = {
        0
    };
GLOBAL_READONLY CORBA_short COPIMPL_AH[12] = {
        10, _TC_HASH_void, _TC_HASH_string, _TC_HASH_long, _TC_HASH_double,
        _TC_HASH_SCOPD_DoubleSeq, _TC_HASH_SCOPD_DataStr, _TC_HASH_string,
        _TC_HASH_long, _TC_HASH_double, _TC_HASH_SCOPD_DoubleSeq,
        _TC_HASH_SCOPD_DataStr,
    };
```

DISP.C

```
/*
 *
 *  ROUTINE NAME:       COPImpl__dispatch
 *
 *  FUNCTIONAL DESCRIPTION:
 *
 *      This is the method dispatch routine for the implementation
 *          COPImpl.

 */

CORBA_Status OBB_EXPORT
COPImpl__dispatch (

OBBInvocationContext * invctx,
CORBA_string         method,
OBBInt               argcount,
OBBArgList           args,
CORBA_Environment    * ev)

{
 OBBInt              hashvalue;
 CORBA_unsigned_short   method_index;
 OBBArg              * arg_ptr = args;
 CORBA_Object        obj_ref;

 /*.....................................................*/

 obj_ref = invctx->ObjRef;

 if ( (argcount != 0) && (arg_ptr == NULL) )
    return(OBB_INV_BADARGLST);

 hashvalue = OBB__MethodHash (method,
             (OBBInt) COPIMPL_IPD.NumOperations,
             COPIMPL_IPD.HashTable, (CORBA_string) COPIMPL_IPD.BucketTable );

 method_index = COPIMPL_IPD.OperToMthTable [ hashvalue ];
 switch ( method_index )
    {
    case 0 : /* Exec */
         {

         if ( argcount < 11 )
             return(OBB_INV_ARGMISMATCH);

         if ( (argcount > 11) &&
              !(OBB__IsAllOptional(args, argcount, 11)))
             return(OBB_INV_ARGMISMATCH);

         COPImpl_Exec (
             obj_ref,
             ev,
             (CORBA_string) args[0] . Value,
             * ((CORBA_long *) args[1] . Value),
             * ((CORBA_double *) args[2] . Value),
             (SCOPD_DoubleSeq *) args[3] . Value,
             (SCOPD_DataStr *) args[4] . Value,
             (CORBA_string *) args[5] . Value,
             (CORBA_long *) args[6] . Value,
             (CORBA_double *) args[7] . Value,
             (SCOPD_DoubleSeq *) args[8] . Value,
```

```
            (SCOPD_DataStr *) args[9] . Value);

        break;
        }

    default :
        return(OBB_INV_MTHDNOTFND);
    }

 switch (ev->_major)
    {
     case CORBA_NO_EXCEPTION:     return (OBB_INV_METHODSUC);
     case CORBA_USER_EXCEPTION:   return (OBB_INV_METHODFAIL);
     case CORBA_SYSTEM_EXCEPTION: return (OBB_INV_METHODFAIL);
     default:                    return (OBB_INV_METHODFAIL);
    }

}
```

```
/*
 *
 *  ROUTINE NAME:       COPImpl__register
 *
 *  FUNCTIONAL DESCRIPTION:
 *
 *      This is the registration routine for the method
 *      server COPImpl.
 *
 *      This routine registers and creates the ImplementationDef.
 *
 *
 *  FORMAL PARAMETERS:
 *
 *      parameter name          [In|Out|In/Out]         [Reference|Value]
 *      --------------          ---------------         -----------------
 *      ImplConst               In                      Reference
 *                              The implementation constant
 *
 *      Ev                      In/Out                  Reference
 *                              The environment. This will contain any
 *                              exceptions from the ORB or BOA routines.
 *
 *      RegServerAttrList       In              Reference
 *                              A list of additional registration attributes.
 *
 *      SelServerAttrList       In              Reference
 *                              A list of additional selection attributes.
 *
 *      Flags                   In                      Value
 *                              The Flags for the registration routine.
 *                              Possible values are TBS.
 *
 *  FUNCTION VALUE:
 *
 *      CORBA_ImplementationDef - the ImplementationDef returned from
 *              CORBA_BOA_create_implementation.
 *      NULL - if an error occurs in the ORB or BOA routines.
 *
 */

CORBA_ImplementationDef
COPImpl__register (

CORBA_ImplementationDef ImplConst,
CORBA_Environment       * Ev,
CORBA_NVList            RegServerAttrList,
CORBA_NVList            SelServerAttrList,
CORBA_Flags             Flags)

{
 OBB_procedure dispatch_rtn = (OBB_procedure) ORB_MAKEPROC(COPImpl__dispatch);
 OBB_procedure get_mthd_args_rtn = (OBB_procedure) ORB_MAKEPROC(COPImpl__getmthargs);
 OBB_procedure lookup_tc_by_id_rtn = (OBB_procedure) ORB_MAKEPROC(_TCS_scopd_LookupBy
ID);
 CORBA_Status OBB_EXPORT StopServer (CORBA_Object Obj,
    CORBA_Environment OBB_FAR * Ev,
    CORBA_Context Ctx, CORBA_short Reason, CORBA_short Subreason,
    void OBB_FAR * Data);
 OBB_procedure deact_impl_rtn = (OBB_procedure) ORB_MAKEPROC(StopServer);
 CORBA_Flags            reg_flags =  Flags;
 CORBA_NVList           attr_list;
```

```
CORBA_NVList            reg_attr_list;
CORBA_NVList            sel_attr_list = NULL;
CORBA_NVList            merge_list;
CORBA_ImplementationDef  impl;
CORBA_Status            status;
CORBA_long              reg_attr_count = 5;
CORBA_long              sel_attr_count = 0;
/*.....................................................*/


/* Initialize the implementation data */

COPIMPL_IPD . Name = COPIMPL_NAM;
COPIMPL_IPD . Id = COPIMPL_ID;
COPIMPL_IPD . NumMethods = COPIMPL_NM;
COPIMPL_IPD . NumOperations = COPIMPL_NO;
COPIMPL_IPD . BucketTable = (OBB_VoidPtr) & COPIMPL_BT;
COPIMPL_IPD . HashTable = COPIMPL_HT;
COPIMPL_IPD . NameTable = COPIMPL_NT;
COPIMPL_IPD . MethodRtnContextTable = COPIMPL_MC;
COPIMPL_IPD . OperToMthTable = COPIMPL_OM;
COPIMPL_IPD . NumActivationContext = 0;
COPIMPL_IPD . ActivationContextTable = NULL;
COPIMPL_IPD . ActivationPolicy = 1;
COPIMPL_IPD . ActivationType = 1;
COPIMPL_IPD . NumImplMap = COPIMPL_NI;
COPIMPL_IPD . ImplMapTable = COPIMPL_IM;

if (Ev != NULL)
   Ev->_major = CORBA_NO_EXCEPTION;

if (sel_attr_count > 0 || SelServerAttrList != NULL)
   reg_attr_count++;

/* Create registration and selection attribute lists. */

status = CORBA_ORB_create_list (CORBA_DEC_ORB_OBJECT, Ev, reg_attr_count,
           & reg_attr_list);
attr_list = reg_attr_list;
if (status != OBB_SUCCESS)
   return (NULL);

/* OBB_IMPL_DATA */

status = CORBA_NVList_add_item (attr_list, Ev,
   (CORBA_Identifier)OBB_IMPL_DATA,
   (CORBA_TypeCode)TC_VoidPtr,
   (CORBA_void *)& COPIMPL_IPD,
   (CORBA_long)sizeof(COPIMPL_IPD),
   (CORBA_Flags)0);
if (status != OBB_SUCCESS)
   {
   CORBA_NVList_free (attr_list, NULL);
   return (NULL);
   }

/* OBB_IMPLEMENTATION_NAME */

status = CORBA_NVList_add_item (attr_list, Ev,
   (CORBA_Identifier)OBB_IMPLEMENTATION_NAME,
   (CORBA_TypeCode)TC_string,
   (CORBA_void *)COPIMPL_NAM,
   (CORBA_long)strlen(COPIMPL_NAM),
```

```
    (CORBA_Flags)0);
if (status != OBB_SUCCESS)
    {
    CORBA_NVList_free (attr_list, NULL);
    return (NULL);
    }

status = CORBA_NVList_add_item (attr_list, Ev,
    (CORBA_Identifier)OBB_GET_MTHD_ARGS_RTN,
    (CORBA_TypeCode)TC_procedure,
    (CORBA_void *) & get_mthd_args_rtn,
    (CORBA_long) sizeof (OBB_procedure),
    (CORBA_Flags)0 );
if (status != OBB_SUCCESS)
    {
    CORBA_NVList_free (attr_list, NULL);
    return (NULL);
    }

status = CORBA_NVList_add_item (attr_list, Ev,
    (CORBA_Identifier)OBB_LOOKUP_TC_BY_ID_RTN,
    (CORBA_TypeCode)TC_procedure,
    (CORBA_void *) & lookup_tc_by_id_rtn,
    (CORBA_long) sizeof (OBB_procedure),
    (CORBA_Flags)0 );
if (status != OBB_SUCCESS)
    {
    CORBA_NVList_free (attr_list, NULL);
    return (NULL);
    }

status = CORBA_NVList_add_item (attr_list, Ev,
    (CORBA_Identifier)OBB_DEACTIVATE_IMPL_RTN,
    (CORBA_TypeCode)TC_procedure,
    (CORBA_void *) & deact_impl_rtn,
    (CORBA_long) sizeof (OBB_procedure),
    (CORBA_Flags)0 );
if (status != OBB_SUCCESS)
    {
    CORBA_NVList_free (attr_list, NULL);
    return (NULL);
    }

/* Merge registration overrides. */

if (RegServerAttrList != NULL && reg_attr_list != NULL)
    {
    /* Create new merged list. */

    status = OBB__MergeLists((OBBList) RegServerAttrList, (OBBList)reg_attr_list,
        (OBBList *) & merge_list);
    if (status != OBB_SUCCESS)
        {
        CORBA_NVList_free (reg_attr_list, NULL);
        return (NULL);
        }
    CORBA_NVList_free (reg_attr_list, NULL);
    attr_list = reg_attr_list = merge_list;
    }

if (sel_attr_count > 0)
    {
    status = CORBA_ORB_create_list (CORBA_DEC_ORB_OBJECT, Ev, sel_attr_count,
```

```
        & sel_attr_list);
    if (status != OBB_SUCCESS)
        {
        CORBA_NVList_free (reg_attr_list, NULL);
        return (NULL);

        }
    }

attr_list = sel_attr_list;

/* Merge selection overrides. */

if (SelServerAttrList != NULL && sel_attr_list != NULL)
    {
    /* Create new merged list. */

    status = OBB__MergeLists((OBBList)SelServerAttrList, (OBBList)sel_attr_list,
        (OBBList *) & merge_list);
    if (status != OBB_SUCCESS)
        {
        CORBA_NVList_free (reg_attr_list, NULL);
        CORBA_NVList_free (sel_attr_list, NULL);
        return (NULL);
        }
    CORBA_NVList_free (sel_attr_list, NULL);
    attr_list = sel_attr_list = merge_list;
    }
else if (SelServerAttrList != NULL)
    attr_list = SelServerAttrList;
else attr_list = sel_attr_list;

if (attr_list != NULL)
    {
    status = OBB_NVList_get_utilized (attr_list, Ev, & sel_attr_count);
    if (status == OBB_SUCCESS)
        status = CORBA_NVList_add_item (reg_attr_list, Ev,
                (CORBA_Identifier)OBB_SELATTR_LIST,
                (CORBA_TypeCode)TC_reserved_nvl,
                (CORBA_void *) attr_list,
                (CORBA_long) sel_attr_count,
                (CORBA_Flags)0 );
    }
if (status != OBB_SUCCESS)
    {
    CORBA_NVList_free (sel_attr_list, NULL);
    CORBA_NVList_free (reg_attr_list, NULL);
    return (NULL);
    }

/* Create an implementation object. */

impl = OBB__CreateImplementationDef( Ev,
    ImplConst,
    reg_attr_list,
    reg_flags,
    dispatch_rtn);

CORBA_NVList_free( reg_attr_list, NULL);
if (sel_attr_list != NULL)
    CORBA_NVList_free( sel_attr_list, NULL);
return (impl);
}
```

DISP.C

```
/*
 *
 *  ROUTINE NAME:       COPImpl__getmthargs
 *
 *  FUNCTIONAL DESCRIPTION:
 *
 *      This is the get method argsroutine for the implementation
 *          COPImpl.
 *
 *      This routine returns the list of arguments for the specified method.
 *
 */

CORBA_NVList OBB_EXPORT
COPImpl__getmthargs (

OBB_BinaryID            ImplId,
CORBA_Environment       * Ev,
OBB_OpInfo              * OpInfo,
CORBA_NamedValue        * Result,
CORBA_Flags             Flags)

{
 OBBInt         hashvalue;
 CORBA_unsigned_short   method_index;
 CORBA_short            arg_offset;
 CORBA_long             arg_count;
 CORBA_NVList           arg_list;
 CORBA_short            * index_ptr;
 CORBA_short            i;
 CORBA_NamedValue               * arg_ptr;
 CORBA_Status           status;

 /*.....................................................*/

 hashvalue = OBB__MethodHash (OpInfo -> OperationID,
        (OBBInt) COPIMPL_IPD.NumOperations,
        COPIMPL_IPD.HashTable, (CORBA_string) COPIMPL_IPD.BucketTable);
 if (hashvalue < 0)
    {
    if (Ev != NULL)
       {
       Ev -> _major = CORBA_SYSTEM_EXCEPTION;
       Ev -> status = OBB_INV_MTHDNOTFND;
       }
    return(NULL);
    }

 method_index = COPIMPL_IPD.OperToMthTable [ hashvalue ];
 arg_offset = COPIMPL_AO [method_index];
 index_ptr = & COPIMPL_AH [arg_offset];
 arg_count = (* index_ptr); index_ptr++;

 status = CORBA_ORB_create_list (CORBA_DEC_ORB_OBJECT, Ev, arg_count+1,
           & arg_list);
 if (status != OBB_SUCCESS)
    return (NULL);

 arg_list[arg_count].argument._type = _TCS_scopd_HashLookup (*index_ptr);
 index_ptr++;
 for (i = 0, arg_ptr = arg_list; i < arg_count; i++, index_ptr++, arg_ptr++)
    {
```

```
    arg_ptr -> argument._type = _TCS_scopd_HashLookup (*index_ptr);
    }

 return (arg_list);
}
```

TYPE.C

```
/*****************************************************************
 * Created Tue Apr  2 11:28:44 1996 by OBB V2.5B-08 (COMPILE/GENERATE)
 *****************************************************************
 */

/*
 * OBB Typecodes and Typecode routines
 * -----------------------------------
 *
 * This module contains Typecodes and Typecode routines for the following
 * Interfaces:
 *
 *    COP
 *
 * WARNING: The source code in this module was generated by Digital
 * Equipment Corporation's(DEC) ObjectBroker. Digital will not support any
 * changes to the source code in this module.
 *
 * How This Module Works
 * ---------------------
 *
 * For a detailed description of the files generated by ObjectBroker, refer
 * to the gen_info.txt file in the ObjectBroker example directory.
 */
#include "scopd.h"
/*
 * The following macro definitions are provided in order to allow the
 * tables and global variables to be included as SHARED READONLY in a
 * shareable image. To enable this, define ORB_SHARED_READONLY. A PSECT
 * name may be specified by defining RO_PSECT
 */

#ifndef GLOBAL_READONLY
#if !defined(ORB_SHARED_READONLY) || !defined(vms)
#define GLOBAL_READONLY
#else
#ifndef RO_PSECT
#define RO_PSECT
#endif
#define GLOBAL_READONLY globaldef RO_PSECT readonly
#endif
#endif

#ifndef STATIC_READONLY
#if !defined(ORB_SHARED_READONLY) || !defined(vms)
#define STATIC_READONLY static
#else
#define STATIC_READONLY static readonly
#endif
#endif


#ifndef GLOBAL_VARIABLE
#if !defined(ORB_SHARED_READONLY) || !defined(vms)
#define GLOBAL_VARIABLE
#else
#ifndef RW_PSECT
#define RW_PSECT
#endif
#define GLOBAL_VARIABLE globaldef RW_PSECT
#endif
#endif
```

TYPE. C

```
GLOBAL_READONLY CORBA_unsigned_long TC_316090dc_0 [] = {
        OBBLONG(0x0d, 0x11, 0x09, 0xbb),
        OBBLONG(0x11, 0x8d, 0x52, 0x74), OBBLONG(0x80, 0x06, 0x00, 0x00),
        OBBLONG(0x02, 0x10, 0xa1, 0x40), OBBLONG(0x59, 0x00, 0x00, 0x00),
        OBBLONG(0x68, 0x00, 0x00, 0x00), OBBLONG(0xff, 0xff, 0xff, 0xff),
        OBBLONG(0x01, 0x00, 0xff, 0xff), OBBLONG(0x03, 0x00, 0x00, 0x00),
        OBBLONG(0x03, 0x00, 0x00, 0x00), OBBLONG(0x08, 0x83, 0xaf, 0xe1),
        OBBLONG(0x1f, 0x5d, 0xc9, 0x11), OBBLONG(0x91, 0xa4, 0x08, 0x00),
        OBBLONG(0x2b, 0x14, 0xa0, 0xfa), OBBLONG(0x00, 0x00, 0x00, 0x00),
        OBBLONG(0x04, 0x00, 0x00, 0x00), OBBLONG(0xcc, 0x00, 0x00, 0x00),
        OBBLONG(0xff, 0xff, 0xff, 0xff), OBBLONG(0xff, 0xff, 0xff, 0xff),
        OBBLONG(0xff, 0xff, 0xff, 0xff), OBBLONG(0xff, 0xff, 0xff, 0xff),
        OBBLONG(0x01, 0x00, 0x00, 0x00), OBBLONG(0x00, 0x00, 0x00, 0x00),
        OBBLONG(0x02, 0x00, 0x00, 0x00), OBBLONG(0x01, 0x00, 0x00, 0x00),
        OBBLONG(0x58, 0x00, 0x00, 0x00), OBBLONG(0x00, 0x00, 0x00, 0x00),
        OBBLONG(0xff, 0xff, 0xff, 0xff), OBBLONG(0x01, 0x00, 0x00, 0x00),
        OBBLONG(0x05, 0xfe, 0x00, 0x00), OBBLONG(0x00, 0x00, 0x00, 0x00) };

GLOBAL_READONLY CORBA_unsigned_long TC_316090dc_1 [] = {
        OBBLONG(0x0d, 0x0f, 0x19, 0xbb),
        OBBLONG(0x11, 0x8d, 0x52, 0x74), OBBLONG(0x78, 0x1b, 0x00, 0x00),
        OBBLONG(0x02, 0x10, 0xa1, 0x40), OBBLONG(0x59, 0x00, 0x00, 0x00),
        OBBLONG(0x78, 0x00, 0x00, 0x00), OBBLONG(0xff, 0xff, 0xff, 0xff),
        OBBLONG(0x01, 0x00, 0xff, 0xff), OBBLONG(0x03, 0x00, 0x00, 0x00),
        OBBLONG(0x03, 0x00, 0x00, 0x00), OBBLONG(0x08, 0x83, 0xaf, 0xe1),
        OBBLONG(0x1f, 0x5d, 0xc9, 0x11), OBBLONG(0x91, 0xa4, 0x08, 0x00),
        OBBLONG(0x2b, 0x14, 0xa0, 0xfa), OBBLONG(0x00, 0x00, 0x00, 0x00),
        OBBLONG(0x04, 0x00, 0x00, 0x00), OBBLONG(0xcc, 0x00, 0x00, 0x00),
        OBBLONG(0xff, 0xff, 0xff, 0xff), OBBLONG(0xff, 0xff, 0xff, 0xff),
        OBBLONG(0xff, 0xff, 0xff, 0xff), OBBLONG(0xff, 0xff, 0xff, 0xff),
        OBBLONG(0x01, 0x00, 0x00, 0x00), OBBLONG(0x00, 0x00, 0x00, 0x00),
        OBBLONG(0x02, 0x00, 0x00, 0x00), OBBLONG(0x01, 0x00, 0x00, 0x00),
        OBBLONG(0x58, 0x00, 0x00, 0x00), OBBLONG(0x00, 0x00, 0x00, 0x00),
        OBBLONG(0xff, 0xff, 0xff, 0xff), OBBLONG(0x01, 0x00, 0x00, 0x00),
        OBBLONG(0x0f, 0x07, 0xff, 0xff), OBBLONG(0x68, 0x00, 0x00, 0x00),
        OBBLONG(0xfe, 0xff, 0xff, 0xff), OBBLONG(0x01, 0x00, 0x00, 0x00),
        OBBLONG(0x05, 0xfe, 0x00, 0x00), OBBLONG(0x00, 0x00, 0x00, 0x00),
        OBBLONG(0x00, 0x00, 0x00, 0x14), 0x00000000, sizeof(SCOPD_UserExcep)
        , (CORBA_unsigned_long) (((CORBA_octet*)&((SCOPD_UserExcep * ) NULL)->Reason-
(CORBA_octet*)0))
        , 0x00000000 };

GLOBAL_READONLY CORBA_unsigned_long TC_316090dc_2 [] = {
        OBBLONG(0x0d, 0x0f, 0x19, 0xbb),
        OBBLONG(0x11, 0x8d, 0x52, 0x74), OBBLONG(0x54, 0x1e, 0x00, 0x00),
        OBBLONG(0x02, 0x10, 0xa1, 0x40), OBBLONG(0x59, 0x00, 0x00, 0x00),
        OBBLONG(0x8c, 0x00, 0x00, 0x00), OBBLONG(0xff, 0xff, 0xff, 0xff),
        OBBLONG(0x01, 0x00, 0xff, 0xff), OBBLONG(0x03, 0x00, 0x00, 0x00),
        OBBLONG(0x03, 0x00, 0x00, 0x00), OBBLONG(0x08, 0x83, 0xaf, 0xe1),
        OBBLONG(0x1f, 0x5d, 0xc9, 0x11), OBBLONG(0x91, 0xa4, 0x08, 0x00),
        OBBLONG(0x2b, 0x14, 0xa0, 0xfa), OBBLONG(0x00, 0x00, 0x00, 0x00),
        OBBLONG(0x04, 0x00, 0x00, 0x00), OBBLONG(0xcc, 0x00, 0x00, 0x00),
        OBBLONG(0xff, 0xff, 0xff, 0xff), OBBLONG(0xff, 0xff, 0xff, 0xff),
        OBBLONG(0xff, 0xff, 0xff, 0xff), OBBLONG(0xff, 0xff, 0xff, 0xff),
        OBBLONG(0x01, 0x00, 0x00, 0x00), OBBLONG(0x00, 0x00, 0x00, 0x00),
        OBBLONG(0x02, 0x00, 0x00, 0x00), OBBLONG(0x01, 0x00, 0x00, 0x00),
        OBBLONG(0x58, 0x00, 0x00, 0x00), OBBLONG(0x00, 0x00, 0x00, 0x00),
        OBBLONG(0xff, 0xff, 0xff, 0xff), OBBLONG(0x01, 0x00, 0x00, 0x00),
        OBBLONG(0x0f, 0x07, 0xff, 0xff), OBBLONG(0x68, 0x00, 0x00, 0x00),
        OBBLONG(0xfe, 0xff, 0xff, 0xff), OBBLONG(0x01, 0x00, 0x00, 0x00),
        OBBLONG(0x28, 0x56, 0x09, 0x04), OBBLONG(0x58, 0xff, 0xff, 0xff),
        OBBLONG(0x7c, 0x00, 0x00, 0x00), OBBLONG(0xfe, 0xff, 0xff, 0xff),
        OBBLONG(0x06, 0x00, 0x00, 0x00), OBBLONG(0x04, 0x00, 0x00, 0x00),
```

TYPE.C

```
        OBBLONG(0x00,0x00,0x00,0x00),
        OBBLONG(0x00,0x00,0x00,0x24), 0x00000000, sizeof(SCOPD_DataStr)
        , (CORBA_unsigned_long) (((CORBA_octet*)&((SCOPD_DataStr * ) NULL)->stringdat
a-(CORBA_octet*)0))
        , (CORBA_unsigned_long) (((CORBA_octet*)&((SCOPD_DataStr * ) NULL)->longdata-
(CORBA_octet*)0))
        , (CORBA_unsigned_long) (((CORBA_octet*)&((SCOPD_DataStr * ) NULL)->doubledat
a-(CORBA_octet*)0))
        , (CORBA_unsigned_long) (((CORBA_octet*)&((SCOPD_DataStr * ) NULL)->dseqdata-
(CORBA_octet*)0))
        , 0x00000000
        , 0x00000000 };


GLOBAL_READONLY struct {
        CORBA_char    Bucket0 [75];     CORBA_char    Bucket1 [38];
    } _TC_scopd_BT = {
        "74528d110680.02.10.a1.40.59.00.00.00¥00074528d111b78.02.10.a1.40.59.00.00.00
¥000",
        "74528d111e54.02.10.a1.40.59.00.00.00¥000"
    };

GLOBAL_READONLY OBBHashEntry      _TC_scopd_HT [3] = {
    {   0,      0 },    {   2,     75 },    {  -1, 65535 }
    };

CORBA_TypeCode _TC_scopd_HashLookup( CORBA_short HashIndex)
 {

 if (HashIndex < 0)
    return(OBB__ConvertKindToTypeCode(
                (CORBA_Environment *) NULL,
                (CORBA_short) (- HashIndex)));

 switch (HashIndex)
        {
         case _TC_HASH_SCOPD_ERR_TYPES :
                return(TC_316090dc_0);

         case _TC_HASH_SCOPD_UserExcep :
                return(TC_316090dc_1);

         case _TC_HASH_SCOPD_DataStr :
                return(TC_316090dc_2);

         default:
                break;
        }
 return ((CORBA_TypeCode) CORBA_OBJECT_NIL);
 }

CORBA_TypeCode OBB_EXPORT _TC_scopd_LookupByID( CORBA_string TypeCodeID)
 {
 CORBA_short hash_index = -1;

 hash_index = OBB__MethodHash(
        TypeCodeID, 3,
        _TC_scopd_HT,
        (CORBA_string) &_TC_scopd_BT);
 if (hash_index < 0)
        return((CORBA_TypeCode) CORBA_OBJECT_NIL);

 return ( _TC_scopd_HashLookup( hash_index ) );
```

TYPE. C

}

EXTERN. H

```
/*
  Prototype Definition
*/
extern void change_data(
    CORBA_Environment *,
    OBB_memptr *,
    CORBA_string,
    CORBA_long,
    CORBA_double,
    SCOPD_DoubleSeq *,
    SCOPD_DataStr *,
    CORBA_string *,
    CORBA_long *,
    CORBA_double *,
    SCOPD_DoubleSeq *,
    SCOPD_DataStr *);

void change_string_data(
  CORBA_Environment *,
  OBB_memptr *,
  CORBA_string,
  CORBA_string *);

extern void change_long_data(
  CORBA_Environment *,
  OBB_memptr *,
  CORBA_long,
  CORBA_long *);


extern void change_double_data(
  CORBA_Environment *,
  OBB_memptr *,
  CORBA_double,
  CORBA_double *);

extern void change_dseq_data(
  CORBA_Environment *,
  OBB_memptr *,
  SCOPD_DoubleSeq *,
  SCOPD_DoubleSeq *);

extern void change_datastr_data(
  CORBA_Environment *,
  OBB_memptr *,
  SCOPD_DataStr *,
  SCOPD_DataStr *);
```

SCOPD.H

```
/*****************************************************************
 * Created Tue Apr  2 11:28:44 1996 by OBB V2.5B-08 (COMPILE/GENERATE)
 *****************************************************************
 */

/*
 * OBB Client Stubs include file
 * -----------------------------
 *
 * This module contains definitions and prototypes for the client stub
 * routines for the following Interfaces:
 *
 *    COP
 *
 * WARNING: The source code in this module was generated by Digital
 * Equipment Corporation's(DEC) ObjectBroker. Digital will not support any
 * changes to the source code in this module.
 *
 * How This Module Works
 * ---------------------
 *
 * For a detailed description of the files generated by ObjectBroker, refer
 * to the gen_info.txt file in the ObjectBroker example directory.
 */
#ifndef _scopd_H
#define _scopd_H
#ifdef __cplusplus
extern "C" {
#endif
#include <stdio.h>
#include <string.h>
#include <orb.h>

#include "scopd.tch"

typedef CORBA_Object SCOPD_COP;

#ifndef _SCOPD_definitions
#define _SCOPD_definitions
typedef CORBA_unsigned_long SCOPD_ERR_TYPES;
#define SCOPD_ErrCopExec 1
#define SCOPD_ErrSumyExec 2
#define SCOPD_ErrCopRunning 3
#define SCOPD_ErrGetReslt 4
typedef struct SCOPD_UserExcep {
        SCOPD_ERR_TYPES Reason;
        } SCOPD_UserExcep;
#define ex_SCOPD_UserExcep "74528d111b78.02.10.a1.40.59.00.00.00"
typedef CORBA_sequence_double SCOPD_DoubleSeq;
typedef struct SCOPD_DataStr {
        CORBA_string stringdata;
        CORBA_long longdata;
        CORBA_double doubledata;
        SCOPD_DoubleSeq dseqdata;
        } SCOPD_DataStr;
#endif
 void SCOPD_COP_Exec (SCOPD_COP object, CORBA_Environment * ev,
    CORBA_string instring,
    CORBA_long inlong,
    CORBA_double indouble,
    SCOPD_DoubleSeq * indseq,
    SCOPD_DataStr * indstr,
    CORBA_string * outstring,
```

```
    CORBA_long * outlong,
    CORBA_double * outdouble,
    SCOPD_DoubleSeq * outdseq,
    SCOPD_DataStr * outdstr);
#ifdef __cplusplus
}
#endif
#endif
```

SCOPD.IDL

```
/*

   SCOPD ｸｶﾍ   Interface Definition  V1.0
*/

module SCOPD {

  enum ERR_TYPES {
        ErrCopExec,
        ErrSumyExec,
        ErrCopRunning,
        ErrGetReslt
  };

  exception UserExcep {
    ERR_TYPES Reason;
  };

  typedef sequence<double> DoubleSeq;

  /*
   ･ﾆ･ｹ･ﾈﾍﾑｹｽﾂ､ﾂﾎ･ﾇ｡ｼ･ｿ
  */
  struct DataStr {
        string          stringdata;
        long            longdata;
        double          doubledata;
        DoubleSeq       dseqdata;
  };


  interface COP{

    void Exec(
        in  string      instring,
        in  long        inlong,
        in  double      indouble,
        in  DoubleSeq   indseq,
        in  DataStr     indstr,
        out string      outstring,
        out long        outlong,
        out double      outdouble,
        out DoubleSeq   outdseq,
        out DataStr     outdstr)
    raises(UserExcep);

  };
};
#pragma repository_id( "SCOPD::ERR_TYPES", "74528d110680.02.10.a1.40.59.00.00.00")
#pragma repository_id( "SCOPD::UserExcep", "74528d111b78.02.10.a1.40.59.00.00.00")
#pragma repository_id( "SCOPD::DoubleSeq", "74528d111d60.02.10.a1.40.59.00.00.00")
#pragma repository_id( "SCOPD::DataStr", "74528d111e54.02.10.a1.40.59.00.00.00")
#pragma interface_id( "SCOPD::COP", "74528d111f48.02.10.a1.40.59.00.00.00")
#pragma operation_id( "SCOPD::COP::Exec", "74528d11203c.02.10.a1.40.59.00.00.00", 1)
```

SCOPD.IMH

```
/*****************************************************************
 * Created Thu Apr 25 09:51:24 1996 by OBB V2.5A-04 (COMPILE/GENERATE)
 *****************************************************************
 */

/*
 * OBB Method routine include file
 * -------------------------------
 *
 * This module contains definitions and prototypes for the method
 * routines for the following Implementations:
 *
 *    COPImpl
 *
 * WARNING: The source code in this module was generated by Digital
 * Equipment Corporation's(DEC) ObjectBroker. Digital will not support any
 * changes to the source code in this module.
 *
 * How This Module Works
 * ---------------------
 *
 * For a detailed description of the files generated by ObjectBroker, refer
 * to the gen_info.txt file in the ObjectBroker example directory.
 */
#ifndef _scopd_IMH
#define _scopd_IMH
#ifdef __cplusplus
extern "C" {
#endif
#include <stdio.h>
#include <string.h>
#include <orb.h>

/*
 * The following macro definitions are provided in order to allow the
 * tables and global variables to be available to other modules.
 */

#ifndef ORB__EXTERN
#if !defined(ORB_SHARED_READONLY) || !defined(vms)
#define ORB__EXTERN extern
#else
#define ORB__EXTERN globalref
#endif
#endif

#include "scopd.h"
 void COPImpl_Exec (CORBA_Object object, CORBA_Environment * ev,
    CORBA_string instring,
    CORBA_long inlong,
    CORBA_double indouble,
    SCOPD_DoubleSeq * indseq,
    SCOPD_DataStr * indstr,
    CORBA_string * outstring,
    CORBA_long * outlong,
    CORBA_double * outdouble,
    SCOPD_DoubleSeq * outdseq,
    SCOPD_DataStr * outdstr);

/* Registration routine prototype. */
CORBA_ImplementationDef
COPImpl__register (
CORBA_ImplementationDef ImplConst,
```

```
        CORBA_Environment       * Ev,
        CORBA_NVList            RegServerAttrList,
        CORBA_NVList            SelServerAttrList,
        CORBA_Flags             Flags);

#define COPImpl__OBJ _OBB_IMPL_8dcc7e31
ORB__EXTERN CORBA_ImplementationDef COPImpl__OBJ;

#define SCOPD_COP__OBJ _OBB_INTF_8ecc7e31
ORB__EXTERN CORBA_InterfaceDef SCOPD_COP__OBJ;
#ifdef __cplusplus
}
#endif
#endif
```

SCOPD.IML

```
// ********************************************************************
// Created Tue Mar 26 16:11:34 1996 by OBB V2.5A-06 (COMPILE/GENERATE)
// ********************************************************************
//
// OBB Default IML file
// --------------------
//
// Use this file as a base for your additional IML definitions.
//

implementation COPImpl
    {
    activation_type ( program );
    activation_string ( "<tba> - command to run implementation" );
    implementation_identifier ( "74528d111f48.02.10.a1.40.59.00.00.00" );
    registration_attribute string ImplementationName = "COPImpl";
    method_dispatcher_routine COPImpl_dispatch ();
    registration_routine COPImpl_register ();
    deactivate_impl_routine StopServer();

    Exec ( )
        implements ( SCOPD::COP::Exec )
        invoke_builtin ( "COPImpl_Exec" )
        ;

    };
```

SCOPD.MML

```
// ********************************************************************
// Created Tue Mar 26 16:11:39 1996 by OBB V2.5A-06 (COMPILE/GENERATE)
// ********************************************************************
//
// OBB Default MML file
// --------------------
//
// Use this file as a base for your additional MML definitions.
//

method_map COPMap interface SCOPD::COP
    {

    Exec :
        {
        default :
            select_implementation
                by_name ( COPImpl )
                by_identifier ( "74528d111f48.02.10.a1.40.59.00.00.00" )
                ;
            select_server
                selection_policy ( default_nodes )
                startup_policy ( default_nodes )
                ;
        };

    };
```

```
/*****************************************************************
 * Created Tue Apr  2 11:28:44 1996 by OBB V2.5B-08 (COMPILE/GENERATE)
 *****************************************************************
 */

/*
 * OBB Typecodes and Typecode routines
 * -----------------------------------
 *
 * This module contains Typecodes and Typecode routines for the following
 * Interfaces:
 *
 *    COP
 *
 * WARNING: The source code in this module was generated by Digital
 * Equipment Corporation's(DEC) ObjectBroker. Digital will not support any
 * changes to the source code in this module.
 *
 * How This Module Works
 * ---------------------
 *
 * For a detailed description of the files generated by ObjectBroker, refer
 * to the gen_info.txt file in the ObjectBroker example directory.
 */
#ifndef _scopd_TCH
#define _scopd_TCH
#ifdef __cplusplus
extern "C" {
#endif
#include <stdio.h>
#include <string.h>
#include <orb.h>

/*
 * The following macro definitions are provided in order to allow the
 * tables and global variables to be available to other modules.
 */

#ifndef ORB__EXTERN
#if !defined(ORB_SHARED_READONLY) || !defined(vms)
#define ORB__EXTERN extern
#else
#define ORB__EXTERN globalref
#endif
#endif


#define TC_SCOPD_ERR_TYPES TC_316090dc_0
#define TC_SCOPD_UserExcep TC_316090dc_1
#define TC_SCOPD_DoubleSeq TC_sequence_double
#define TC_SCOPD_COP TC_Object
#define TC_SCOPD_DataStr TC_316090dc_2

#define _TC_HASH_SCOPD_ERR_TYPES 0
#define _TC_HASH_SCOPD_UserExcep 1
#define _TC_HASH_SCOPD_DoubleSeq -38
#define _TC_HASH_SCOPD_COP -14
#define _TC_HASH_SCOPD_DataStr 2

ORB__EXTERN CORBA_unsigned_long TC_316090dc_0 [];
ORB__EXTERN CORBA_unsigned_long TC_316090dc_1 [];
ORB__EXTERN CORBA_unsigned_long TC_316090dc_2 [];
```

SCOPD.TCH

```
CORBA_TypeCode _TC_scopd_HashLookup( CORBA_short HashIndex);

CORBA_TypeCode OBB_EXPORT _TC_scopd_LookupByID( CORBA_string TypeCodeID);

#ifdef __cplusplus
}
#endif
#endif
```

MAKEFILE

```
#
CC=/usr/local/bin/gcc

# LIBRARY NAMES GIVEN BELOW
LIBRARIES= -lobb
#LIBRARIES= -lobb
#/*  -lc_gnu  */

# LINKER GIVEN BELOW
LD=$(CC)
#LD=cc

# LINKER FLAGS GIVEN BELOW
LDFLAGS= -O

# DEFINES
AR=ar
ARFLAGS=rv
CP=cp
RM=rm -f

# COMPILER FLAGS GIVEN BELOW
CFLAGS= -g
PFLAGS=$(CFLAGS)
FFLAGS=$(CFLAGS)
COBFLAGS=$(CFLAGS) -xref


# INCLUDE SEARCH PATH GIVEN BELOW
INCLUDES=

.SUFFIXES: .uid .uil .o

# SOURCES GIVEN BELOW
SRCS= method.c¥
 srv_main.c¥
 disp.c¥
 chdata.c¥
 obbansi.c

# OBJECTS GIVEN BELOW
OBJS= method.o¥
 srv_main.o¥
 disp.o¥
 chdata.o¥
 obbansi.o

scopdsrv : $(OBJS)
        $(LD) -o $@ $(LDFLAGS) $(OBJS) $(LIBRARIES)

.c.o:
        $(CC) -c $(CFLAGS) $(INCLUDES) `pwd`/$<

.c.a:
        $(CC) -c $(CFLAGS) $(INCLUDES) `pwd`/$<
        $(AR) $(ARFLAGS) $@ $*.o
        $(RM) $*.o


IR: scopd.idl scopd.iml scopd.mml cstub method_tmp.c disp.c scopd.ir

#scopd.idl: scopd.idl
#       obbgen -u -f scopd.idl
```

MAKEFILE

```
#scopd.iml:
#       obbcomp -i scopd.iml scopd.idl

#scopd.mml: scopd.idl scopd.iml
#       obbcomp -m scopd.mml scopd.idl scopd.iml

method_tmp.c : scopd.idl scopd.iml scopd.mml
        obbcomp -t method_tmp.c scopd.idl scopd.iml scopd.mml

disp.c : scopd.idl scopd.iml scopd.mml
        obbcomp -d disp.c scopd.idl scopd.iml scopd.mml

cstub : scopd.idl scopd.iml scopd.mml
        obbcomp -T type.c -x -ps -c stub.c scopd.idl scopd.iml scopd.mml

scopd.ir : scopd.idl scopd.iml scopd.mml
        rm -f scopd.ir
        obbldrep scopd.idl scopd.iml scopd.mml
clean:
        $(RM) $(OBJS) core scopdsrv

touchsrcs:
        touch $(SRCS)

lint:
        lint $(INCLUDES) $(SRCS) $(LINTLIBS)

depend:
        @cat < /dev/null > makedep
        @for i in ${SRCS}; do ¥
                ($(CC) -M $(INCLUDES) $$i >> makedep); done
#
# Remove the next two lines to keep dependencies on system include files
#
        @grep -v "¥.o:[  ]*/usr/include" makedep > makedep1
        @mv makedep1 makedep
        @echo '/^# DO NOT DELETE THIS LINE/+1,$$d' > eddep
        @echo '$$r makedep' >> eddep
        @echo 'w' >> eddep
        @${CP} Makefile_srv Makefile_srv.bak
        @ed - Makefile_srv < eddep
        @$(RM) eddep makedep
        @echo '# DO NOT EDIT THIS FILE HERE.' >> Makefile_srv
        @echo '# USER EDITS MUST PRECEDE THE COMMENT:' >> Makefile_srv
        @echo '# "# DO NOT DELETE THIS LINE".' >> Makefile_srv
        @echo '# see make depend above' >> Makefile_srv

# DO NOT DELETE THIS LINE -- make depend uses it
```

付録C　　オブジェクトブローカ スタートアップ 実習テキスト

付録D　　分散オブジェクト環境の構築とアプリケーション統合

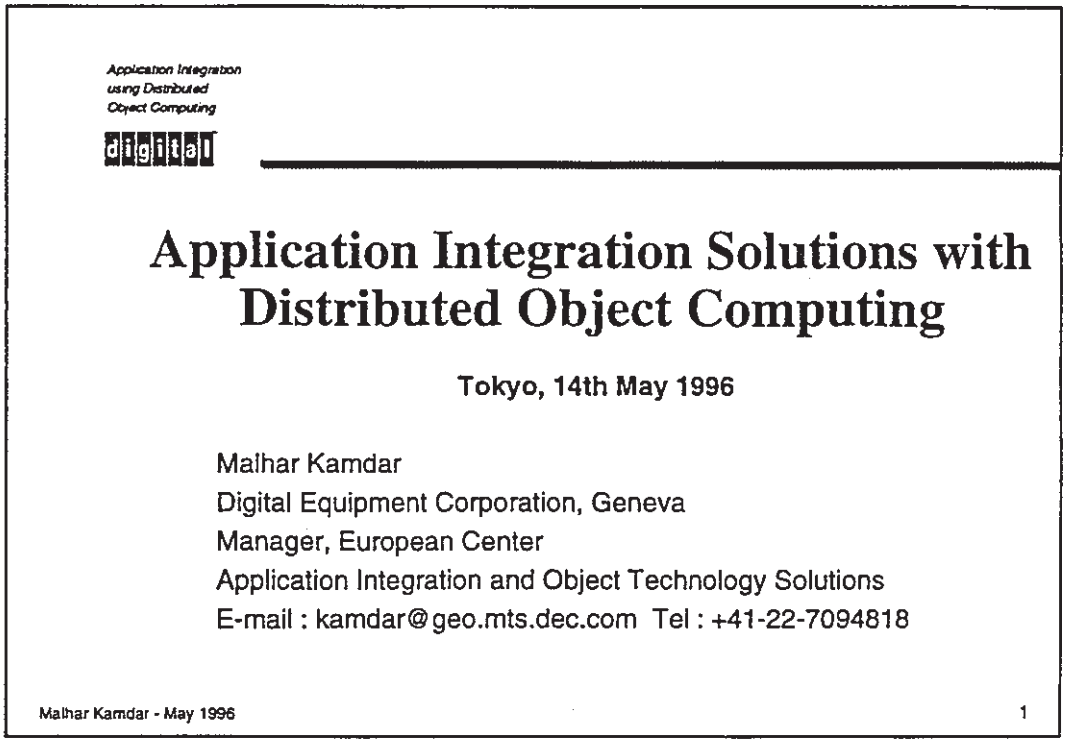
Application Integration using Distributed Object Computing

digital

分散オブジェクト環境の構築とアプリケーション統合

CORBAによるクライアント／サーバ・アプリケーション統合技術のご紹介

Malhar Kamdar - May 1996



Application Integration using Distributed Object Computing

digital

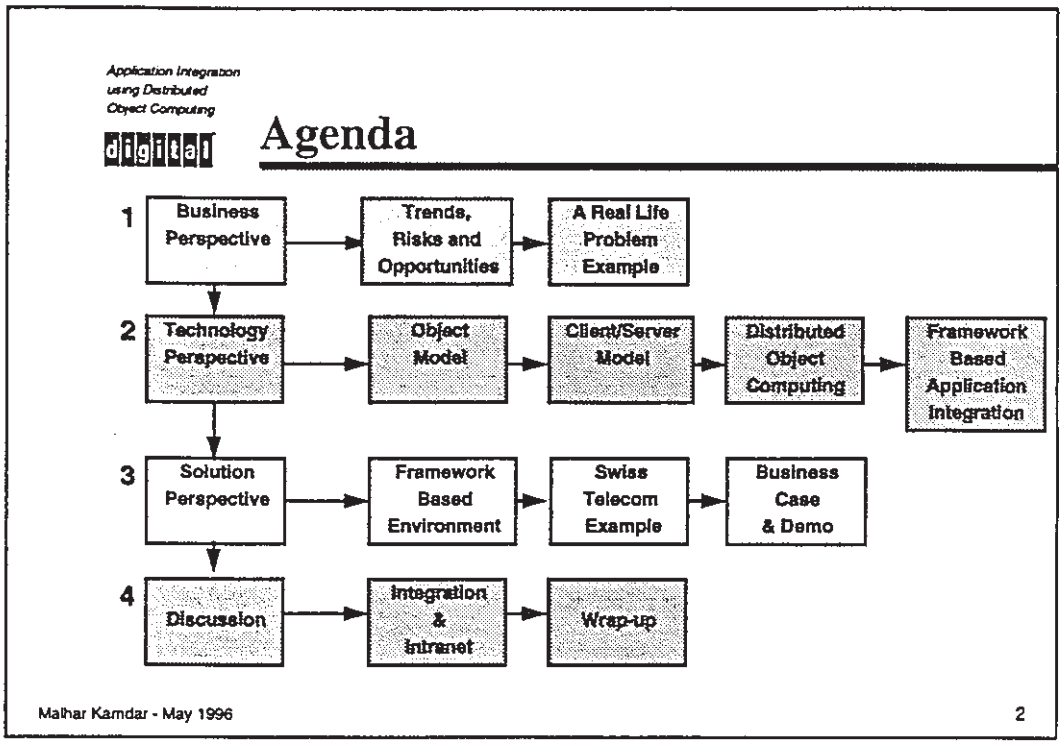
Application Integration Solutions with Distributed Object Computing

Tokyo, 14th May 1996

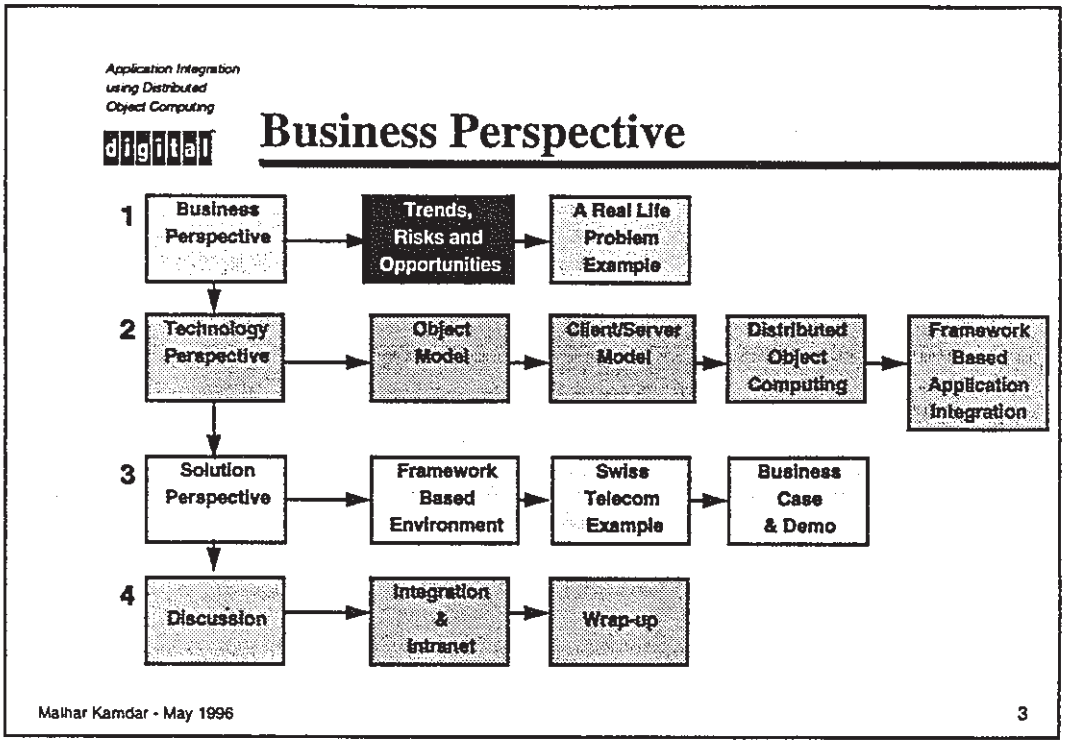
Malhar Kamdar
Digital Equipment Corporation, Geneva
Manager, European Center
Application Integration and Object Technology Solutions
E-mail : kamdar@geo.mts.dec.com Tel : +41-22-7094818

Malhar Kamdar - May 1996 1

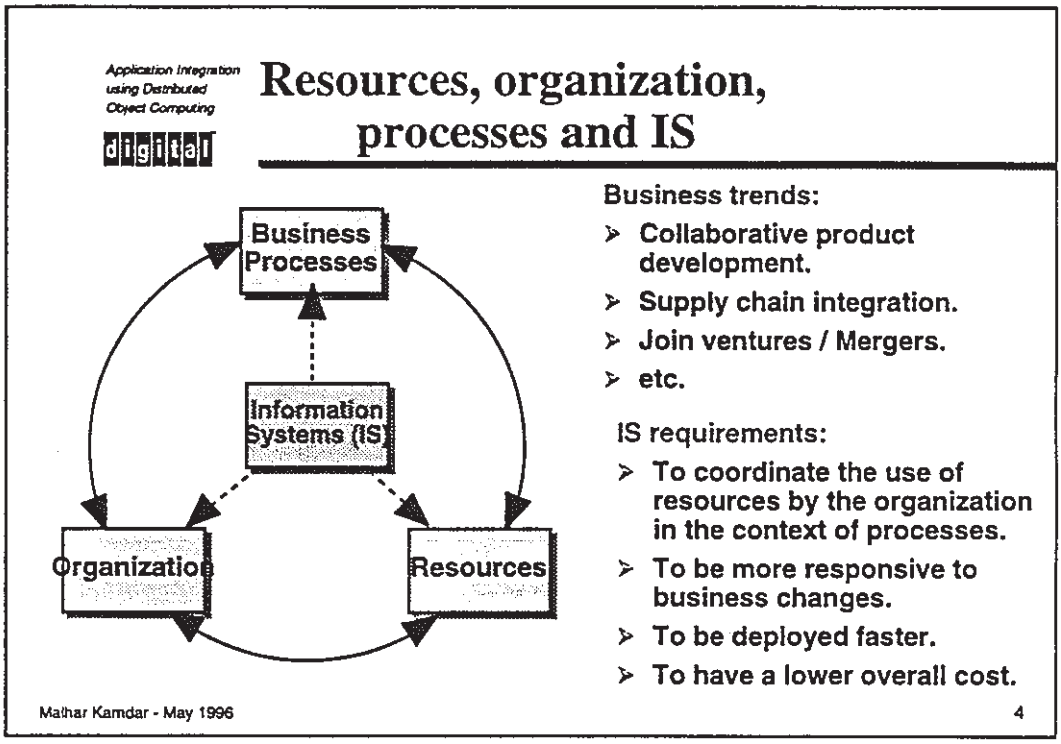
分散オブジェクト・コンピューティングによるアプリケーション統合ソルーション


Application Integration using Distributed Object Computing
digital
Agenda
1
Business Perspective
Trends, Risks and Opportunities
A Real Life Problem Example
2
Technology Perspective
Object Model
Client/Server Model
Distributed Object Computing
Framework Based Application Integration
3
Solution Perspective
Framework Based Environment
Swiss Telecom Example
Business Case & Demo
4
Discussion
Integration & Intranet
Wrap-up
Malhar Kamdar - May 1996
2

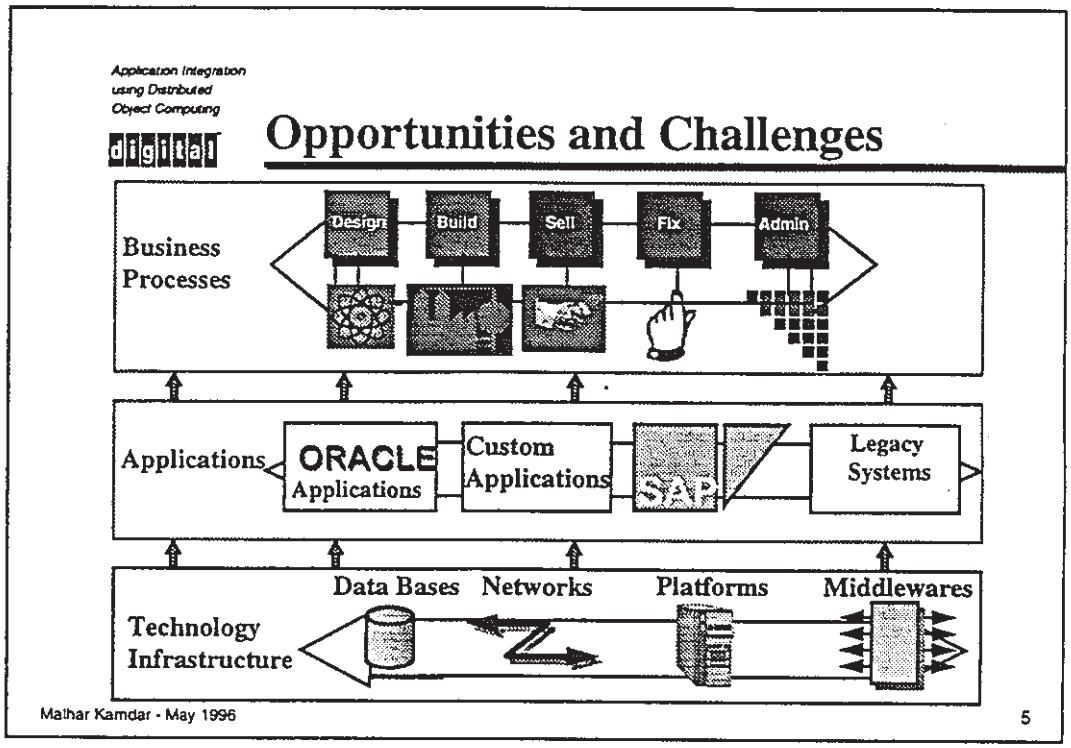
アジェンダ



ビジネス：トレンド、リスクそして機会



リソース、組織、プロセスそしてＩＳ
ビジネス・トレンド
－ プロダクトの協調開発
－ サプライ・チェーンの統合
－ 投機と合併
ＩＳへの要求
－ 作業プロセスの資源の利用を組織間で整合させる
－ ビジネスの変動により対応できるように
－ より早く展開できるように
－ よりコストを押さえて


Application Integration
using Distributed
Object Computing
digital
Opportunities and Challenges
Business
Processes
Design
Build
Sell
Fix
Admin
Applications
ORACLE
Applications
Custom
Applications
SAP
Legacy
Systems
Data Bases
Networks
Platforms
Middlewares
Technology
Infrastructure
Malhar Kamdar - May 1996
5

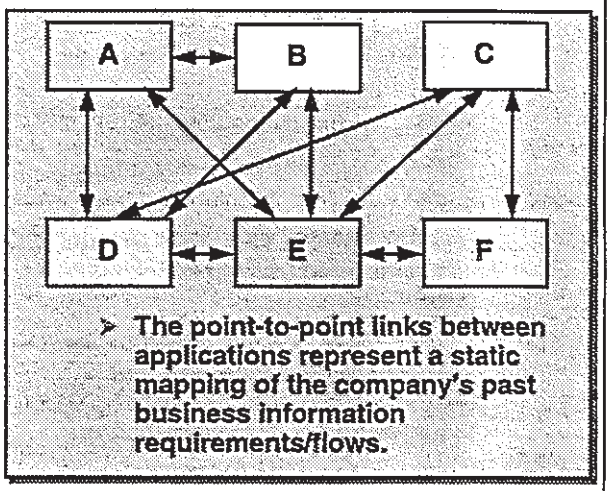
機会と変革

Application Integration using Distributed Object Computing

digital

Today's typical IS environment: Integration Gridlock

- The response to these trends was/is to integrate by means of a point-to-point communication links.
- The result is an increasingly rigid IT application infrastructure.



Malhar Kamdar - May 1996 6

典型的なＩＳ環境の統合の症状

ポイント・ツー・ポイントのアプリケーション・リンクを統合とした／する結果

ＩＴのアプリケーションのインフラの硬直化が増す

ポイント・ツー・ポイントのアプリケーション統合は、会社のこれまでのビジネス要求／フローを表している

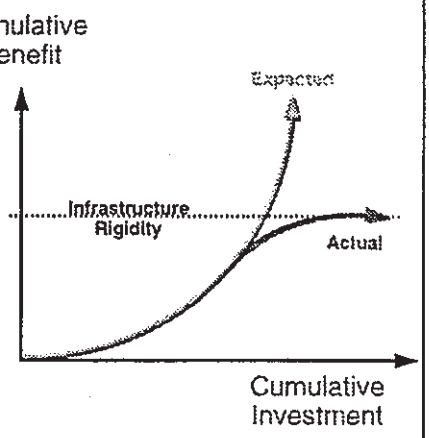
Application Integration using Distributed Object Computing

digital

IT crisis

The IT cannot support the business:

- **Old and inflexible applications which are difficult to change.**
- **Maintenance of existing systems absorbs most of the IT budget and resources.**
- **Traditional methodology and development tools cannot support the speed of business changes.**



Mathar Kamdar - May 1996 7

ＩＴの危機 － ＩＴはビジネスをサポートできない
硬直した旧来のアプリケーションは変更が困難
既存のシステムのメンテナンスでＩＴ予算のほとんどを費やす
伝統的な方法論と開発ツールはビジネスの変動についていけない

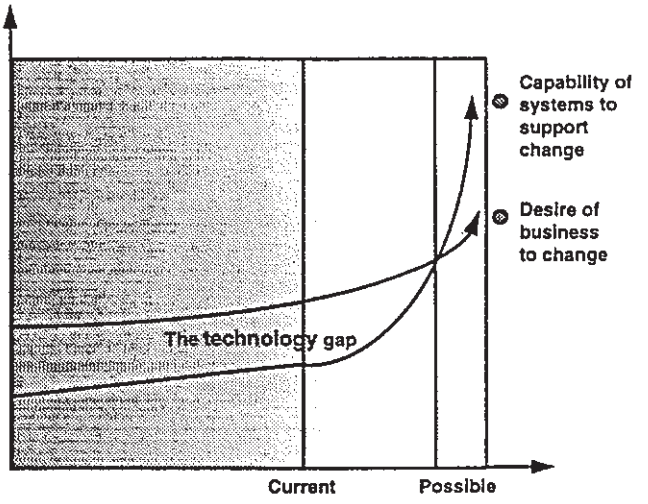
Application Integration using Distributed Object Computing

digital

The Technology Gap

Paradox:

→ **At the same time, the new technologies start to be able to provide more capabilities to support changes than what is desired by the business.**



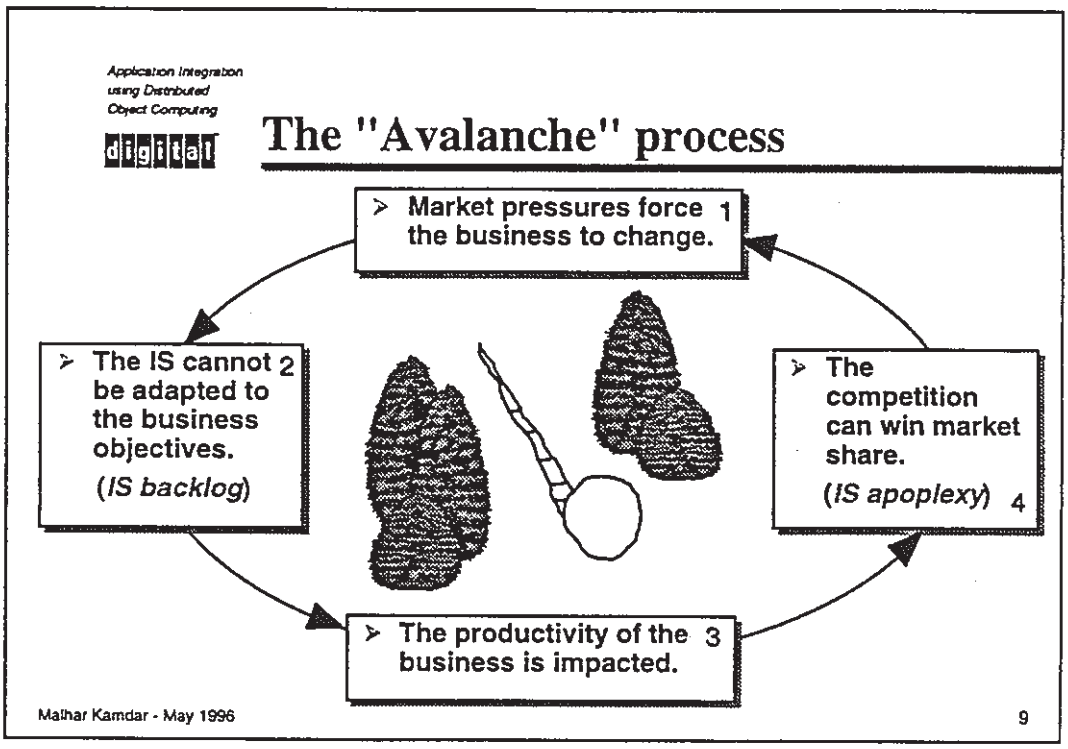
Malhar Kamdar - May 1996

8

テクノロジのギャップ

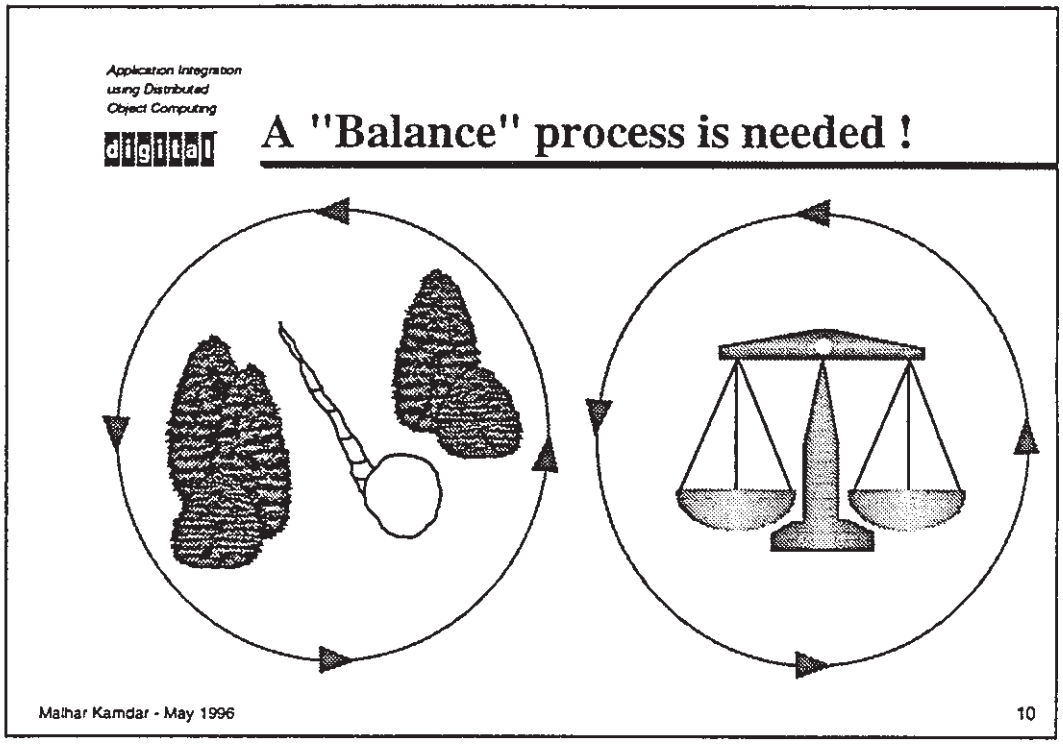
矛盾

－ ビジネスから望まれる変更より、変更をサポートするより高い能力をシステムが提供できるように、新しい技術は始まる

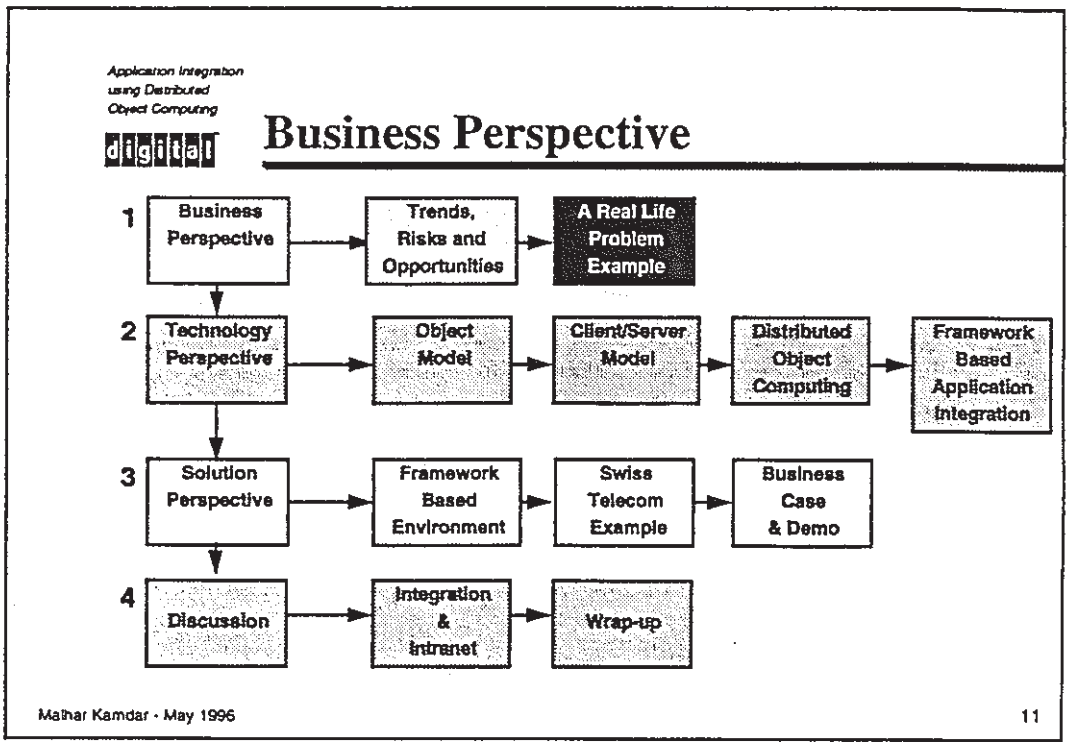


雪崩現象

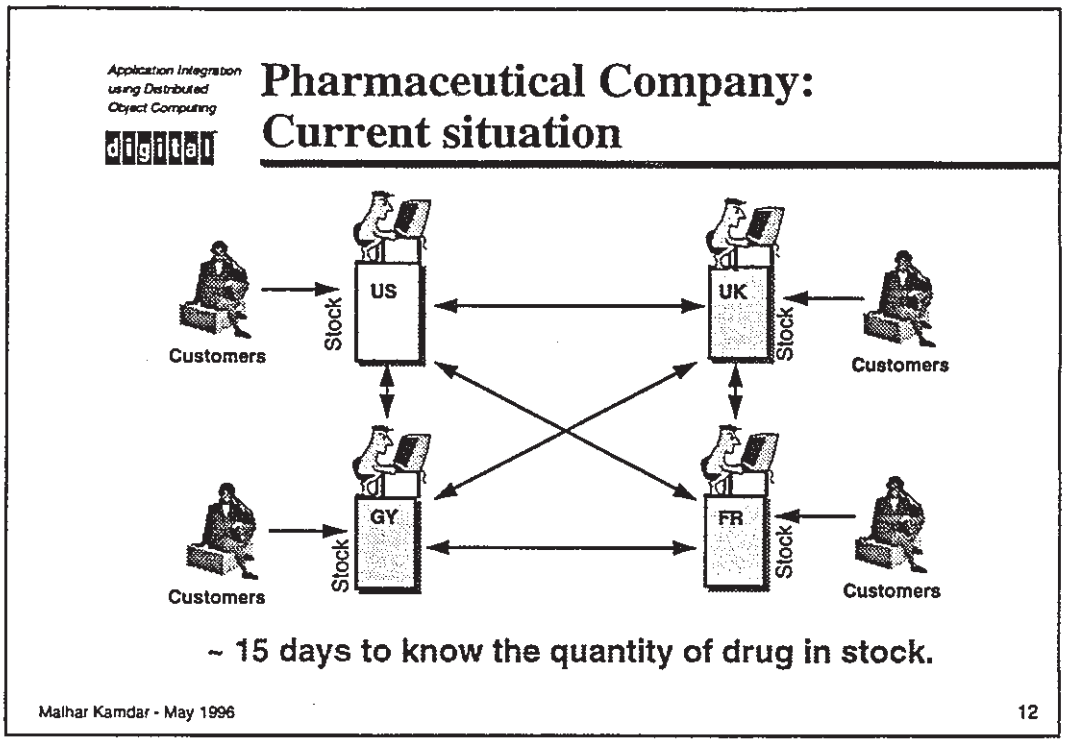
1. ビジネスの変革を強いる市場の圧力
2. ＩＳはビジネスに対応できない（ＩＳのバックログ）
3. ビジネスの生産性にインパクトを与える
4. 競合他社が市場のシェアを得る



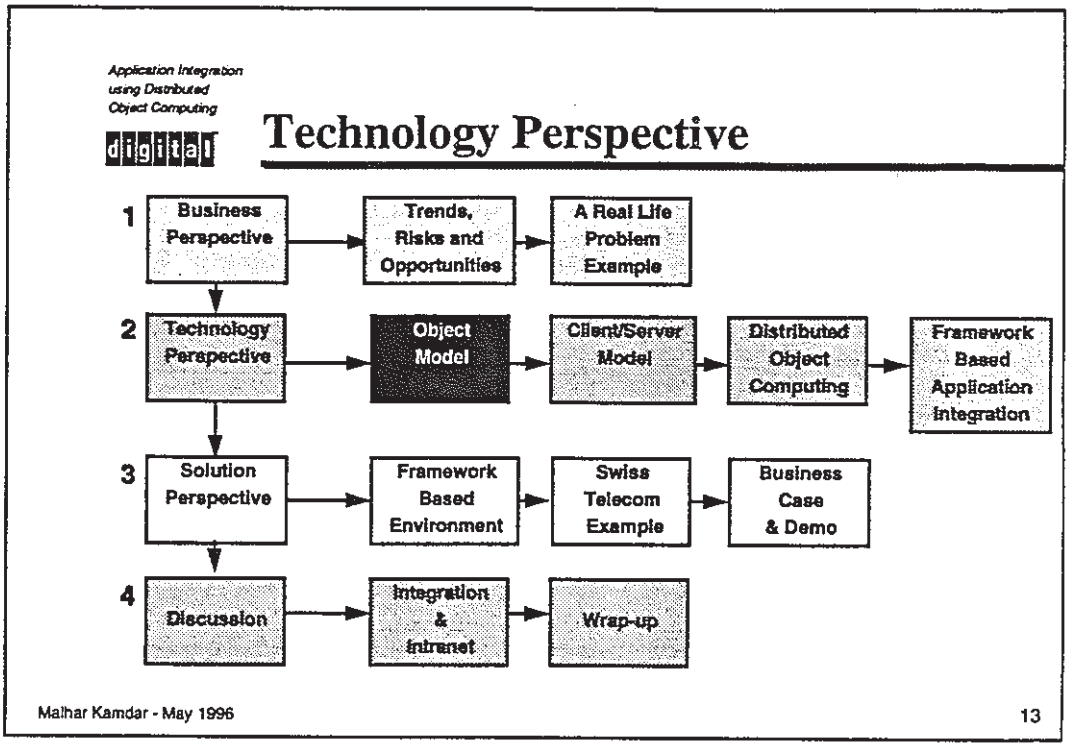
釣り合いのとれたプロセスが必要



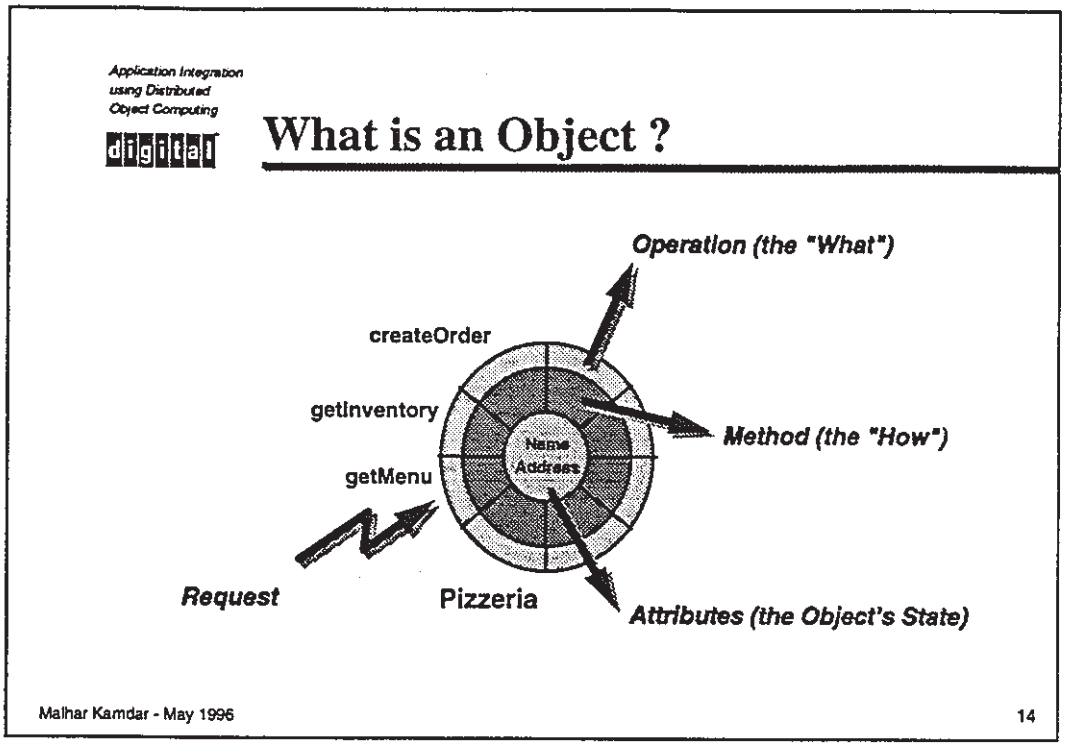
ビジネスの展望 － 問題の例



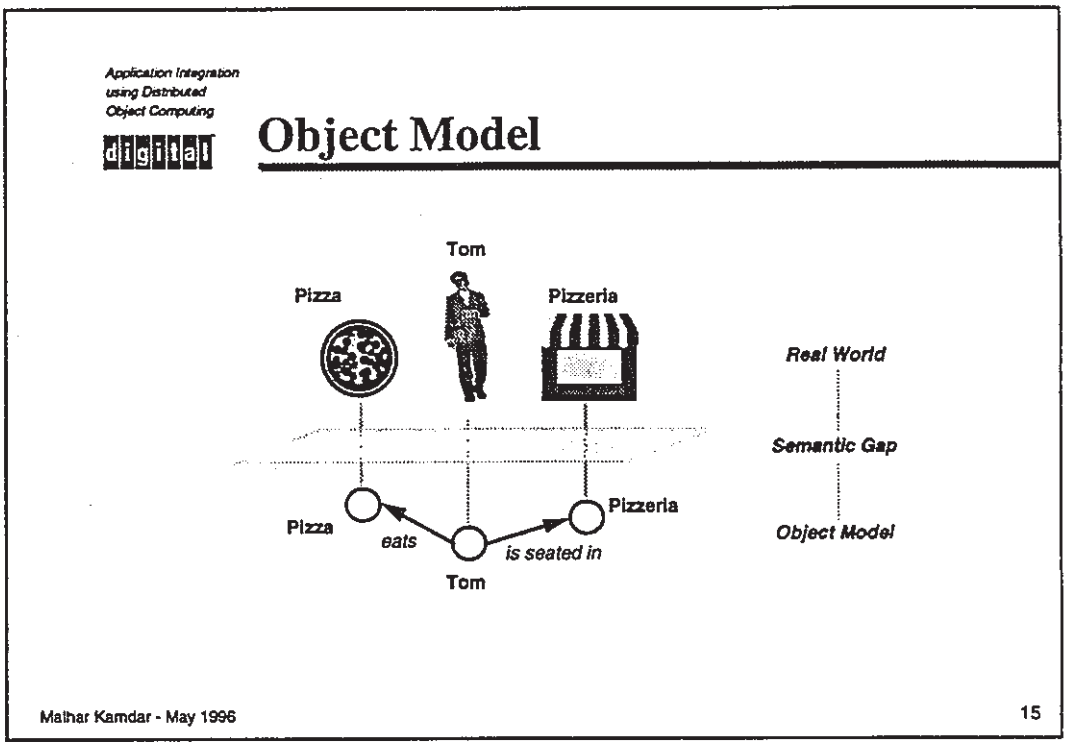
製薬会社の現状 － 薬品の在庫を知るのに１５日かかる



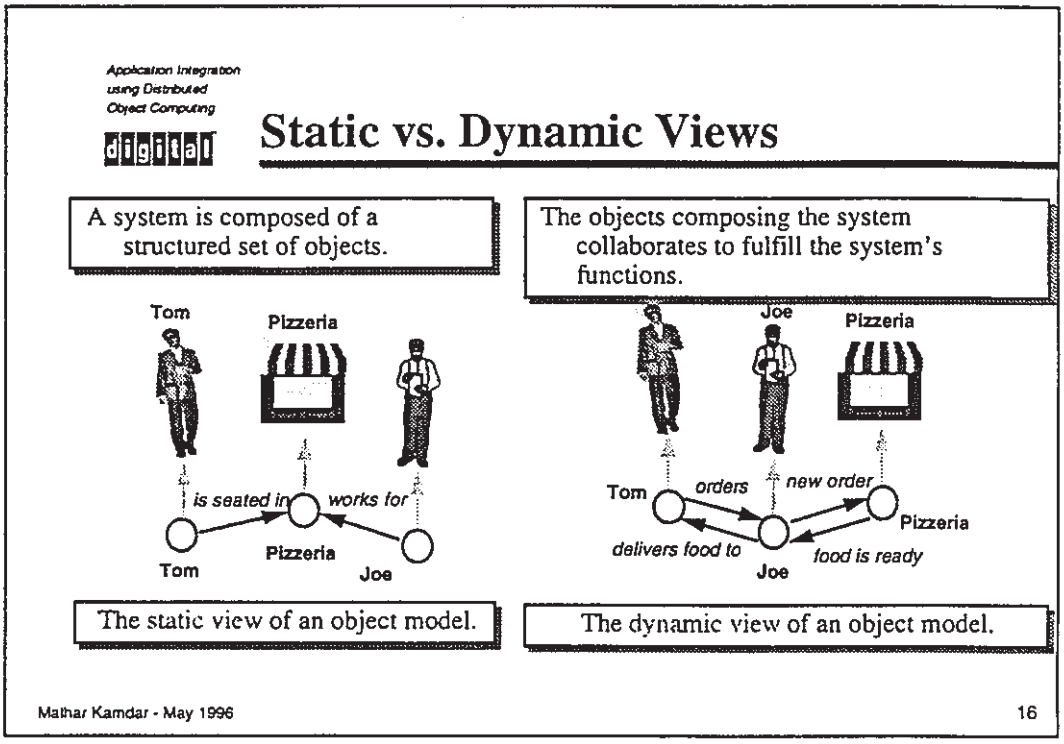
テクノロジの展望 － オブジェクト・モデル


Application Integration
using Distributed
Object Computing
digital
What is an Object ?
Operation (the "What")
createOrder
getInventory
getMenu
Name
Address
Method (the "How")
Request
Pizzeria
Attributes (the Object's State)
Malhar Kamdar - May 1996
14

オブジェクトとは？


Application Integration
using Distributed
Object Computing
digital
Object Model
Tom
Pizza
Pizzeria
Real World
Semantic Gap
Object Model
Pizza
eats
Tom
is seated in
Pizzeria
Mathar Kamdar - May 1996
15

オブジェクト・モデル



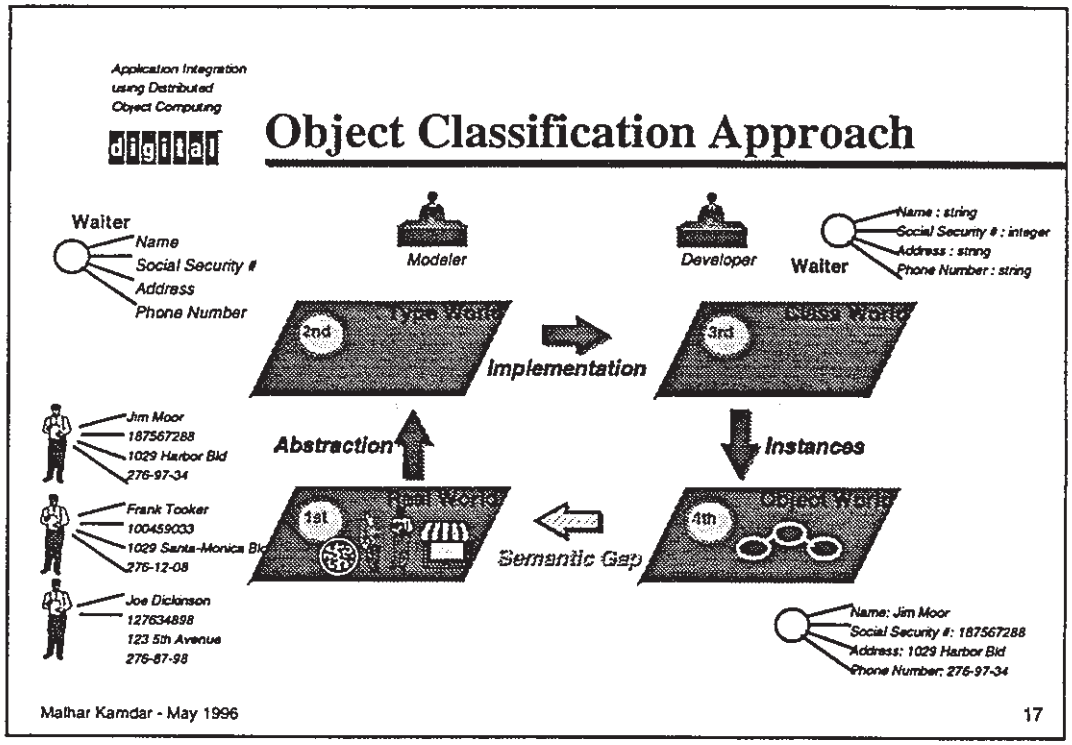
静的ビューｖｓ動的ビュー

オブジェクトの静的なビュー

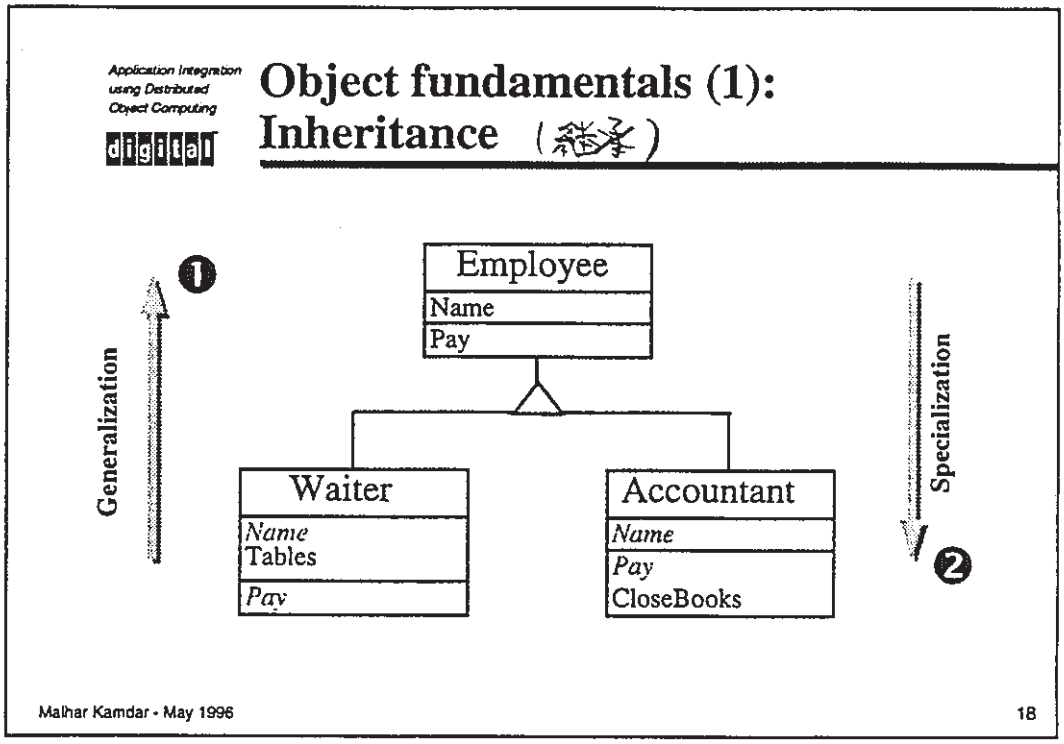
－ 一つのシステムはオブジェクトの構造的な集まりから構成される

オブジェクトの動的なビュー

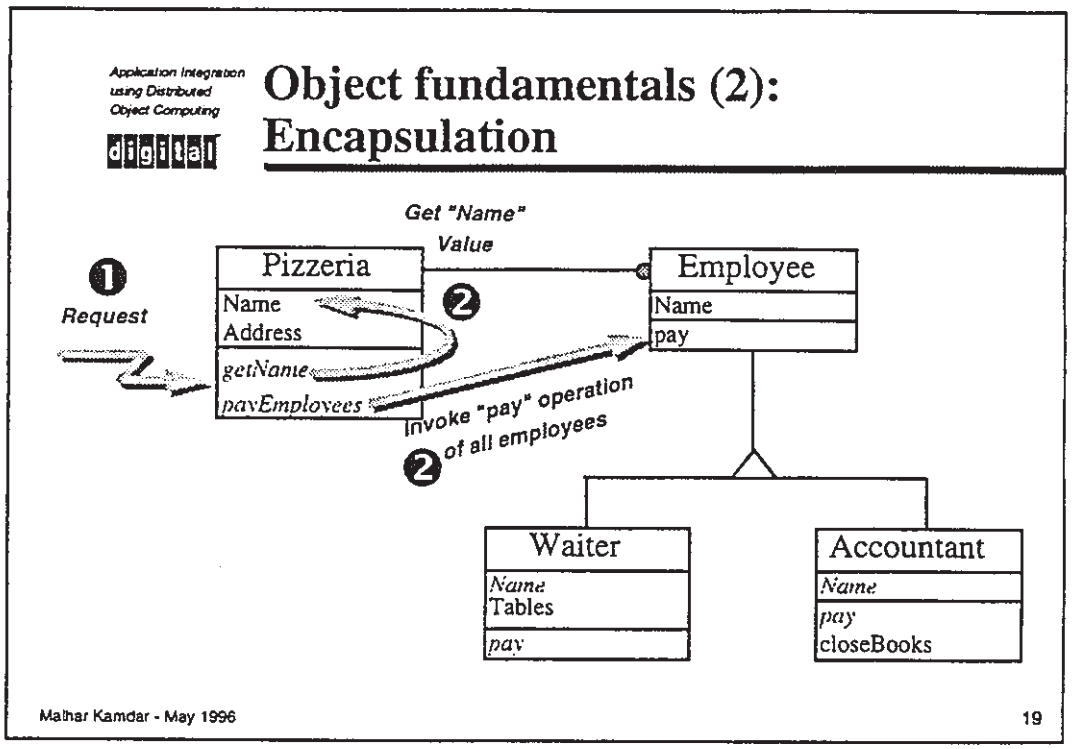
－ システムを構成するオブジェクトは、システムの機能を遂行するために協調する



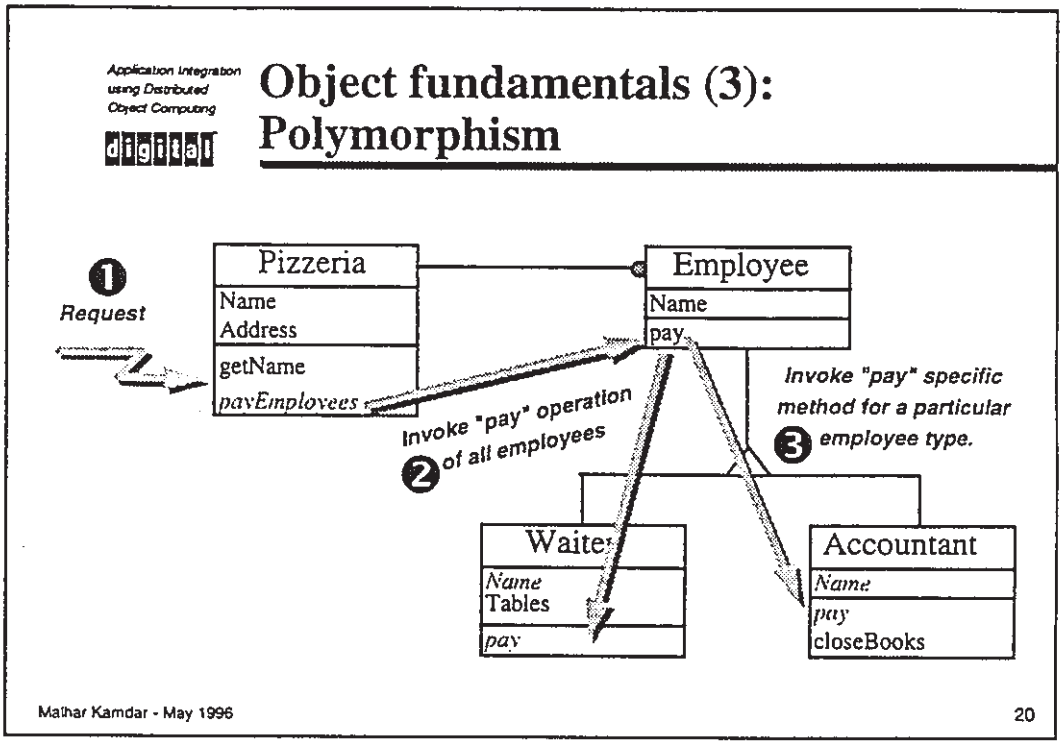
オブジェクトのクラス化によるアプローチ



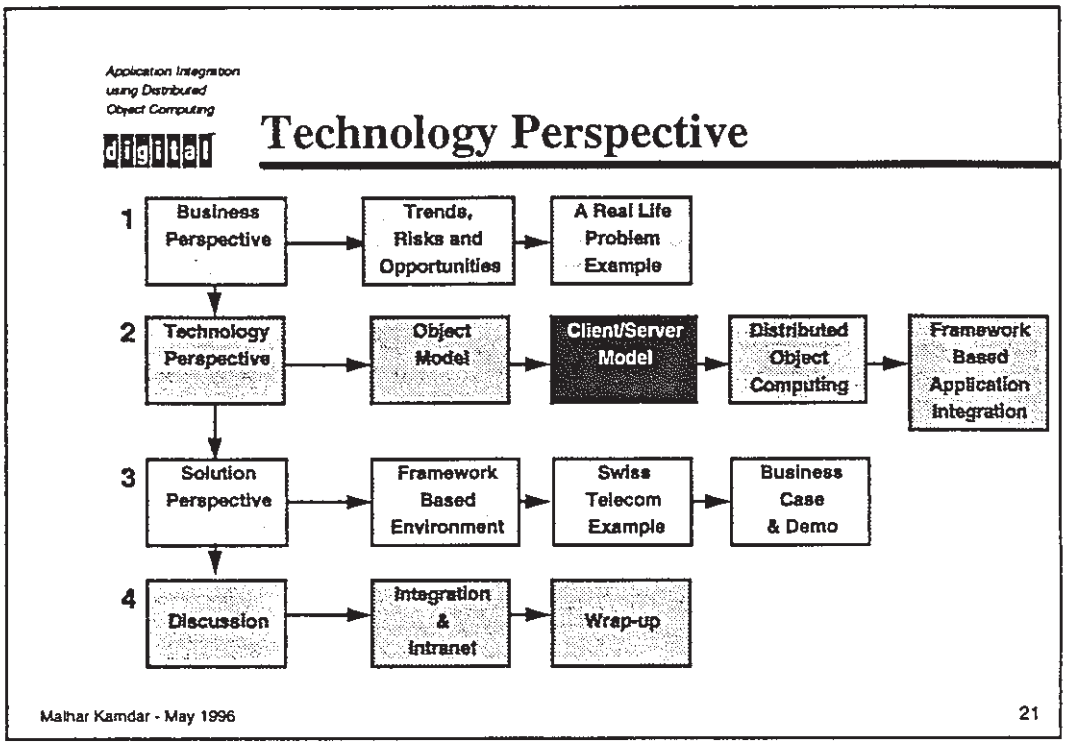
オブジェクト指向の基礎（1） － 継承
特化と汎化



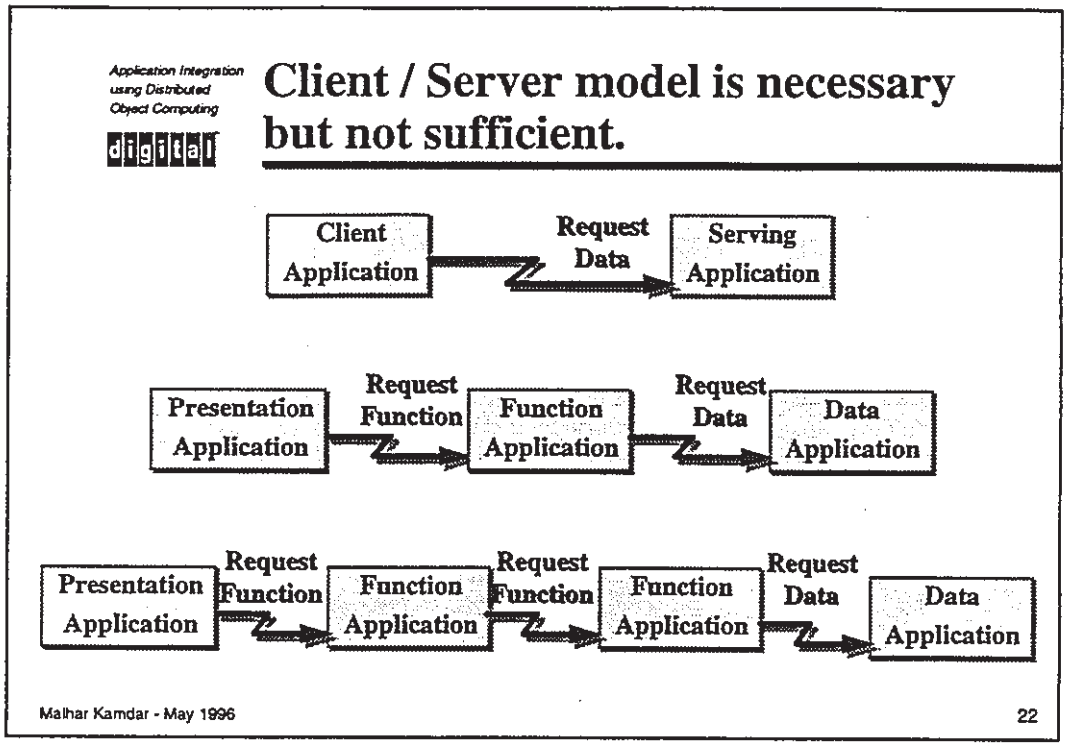
オブジェクト指向の基礎（2） － カプセル化



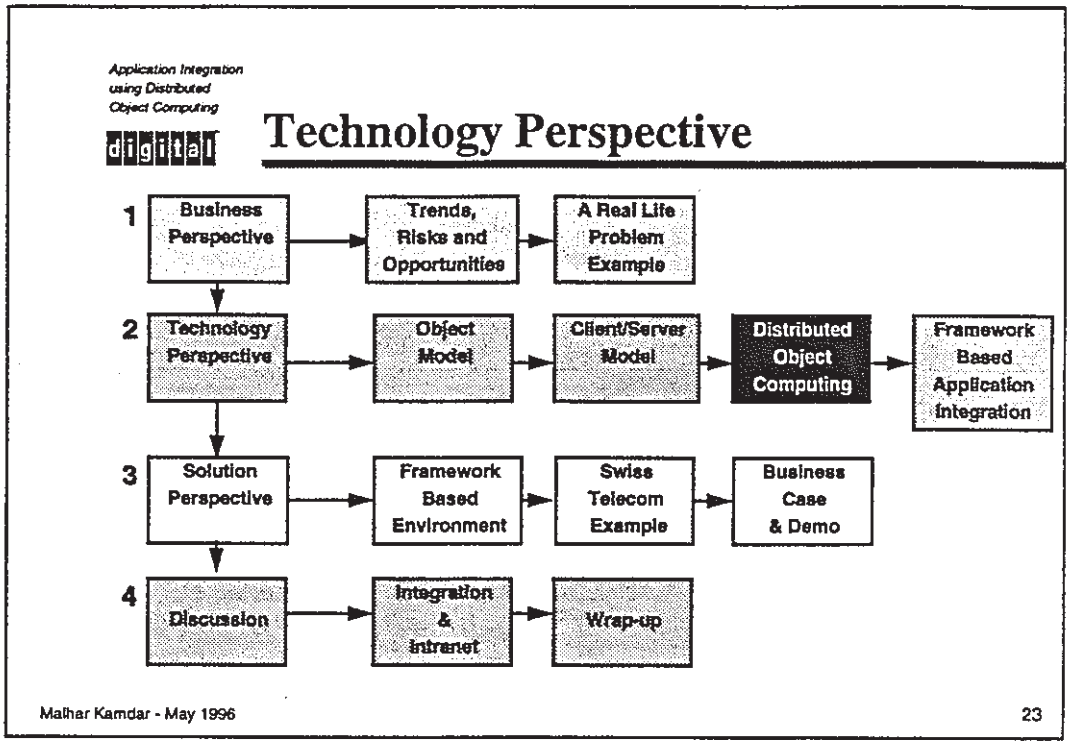
オブジェクト指向の基礎（３） ー 多態性



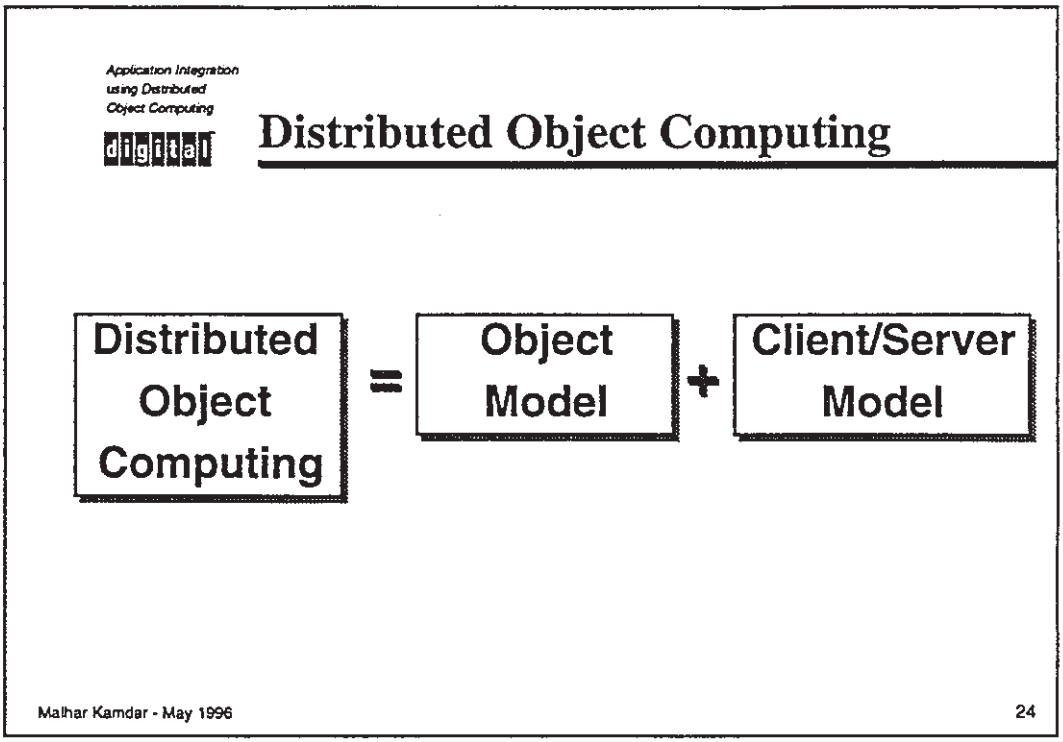
テクノロジの展望 ー クライアント／サーバ・モデル



クライアント／サーバ・システムは必要、でもそれだけでは十分ではない

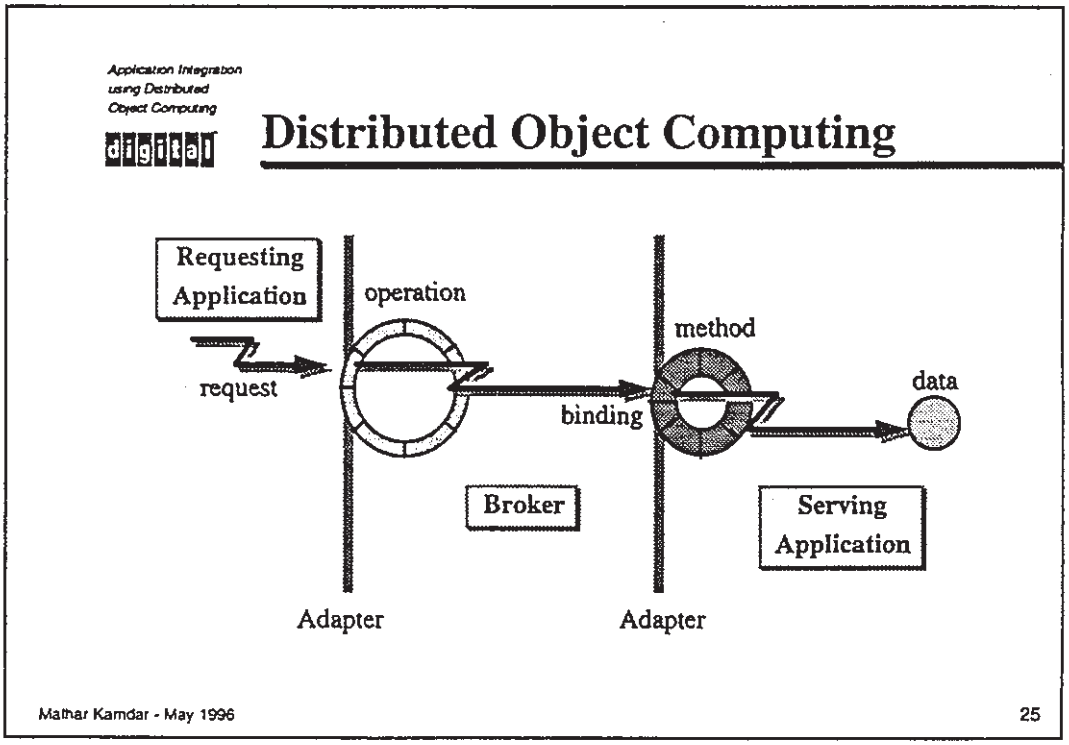


テクノロジ展望 － 分散オブジェクト・コンピューティング

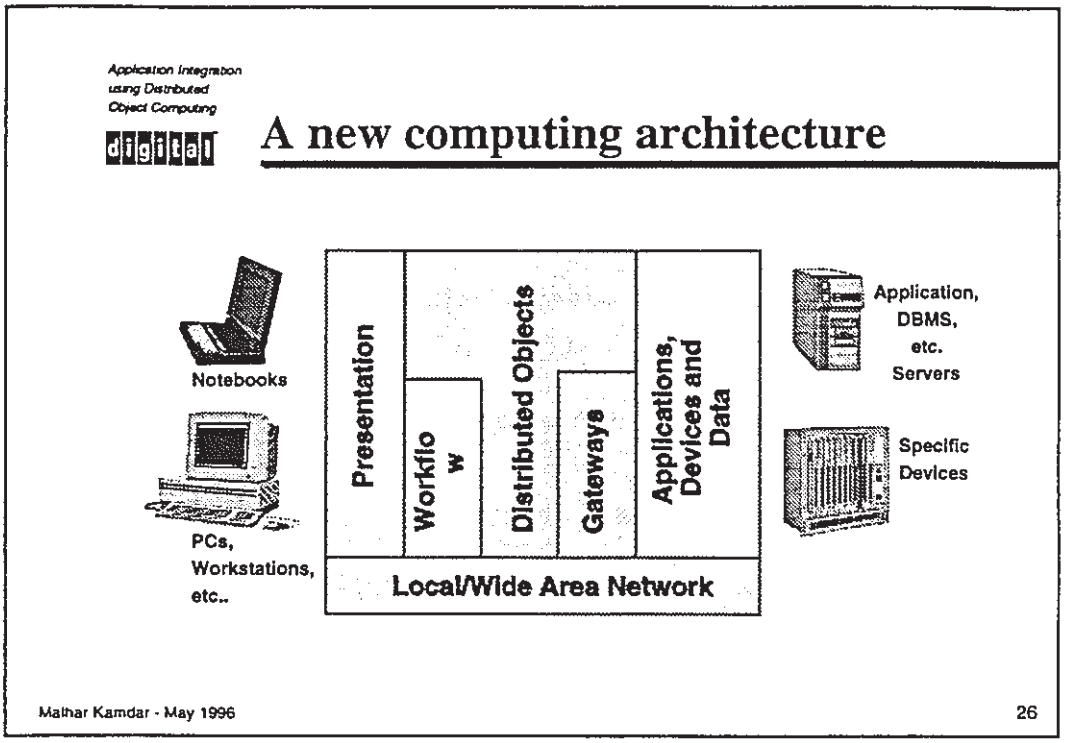


分散オブジェクト・コンピューティング

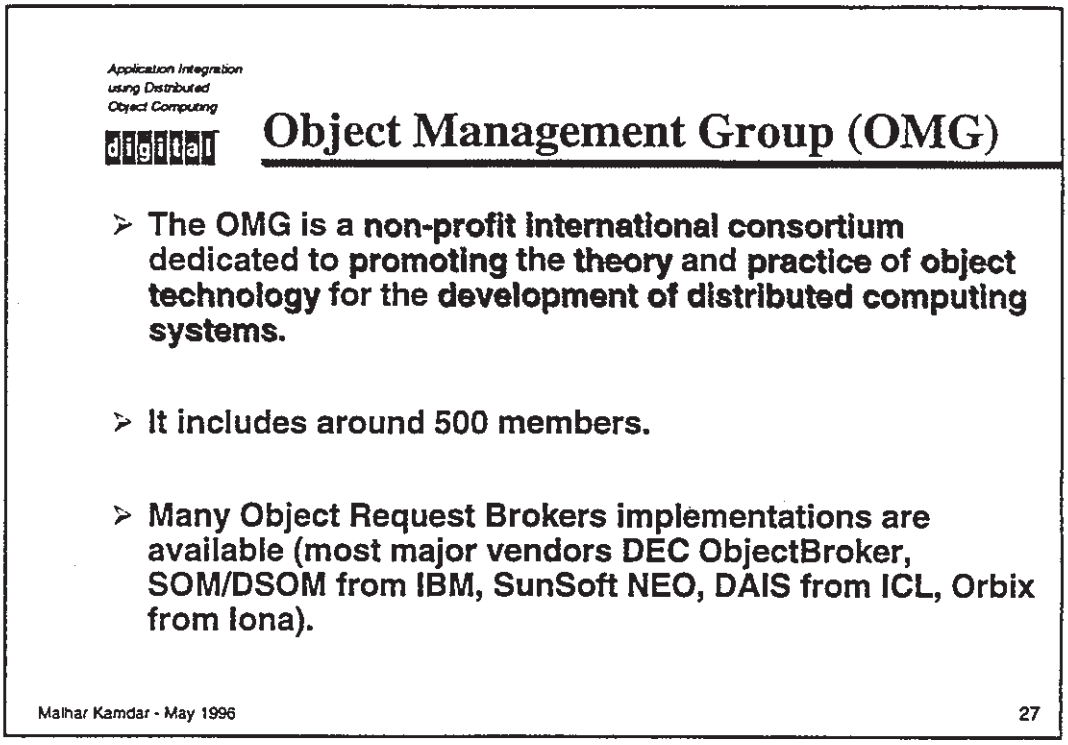
分散オブジェクト・コンピューティング ＝
オブジェクト・モデル ＋ クライアント／サーバ・モデル



分散オブジェクト・コンピューティング



新しいコンピューティング・アーキテクチャ

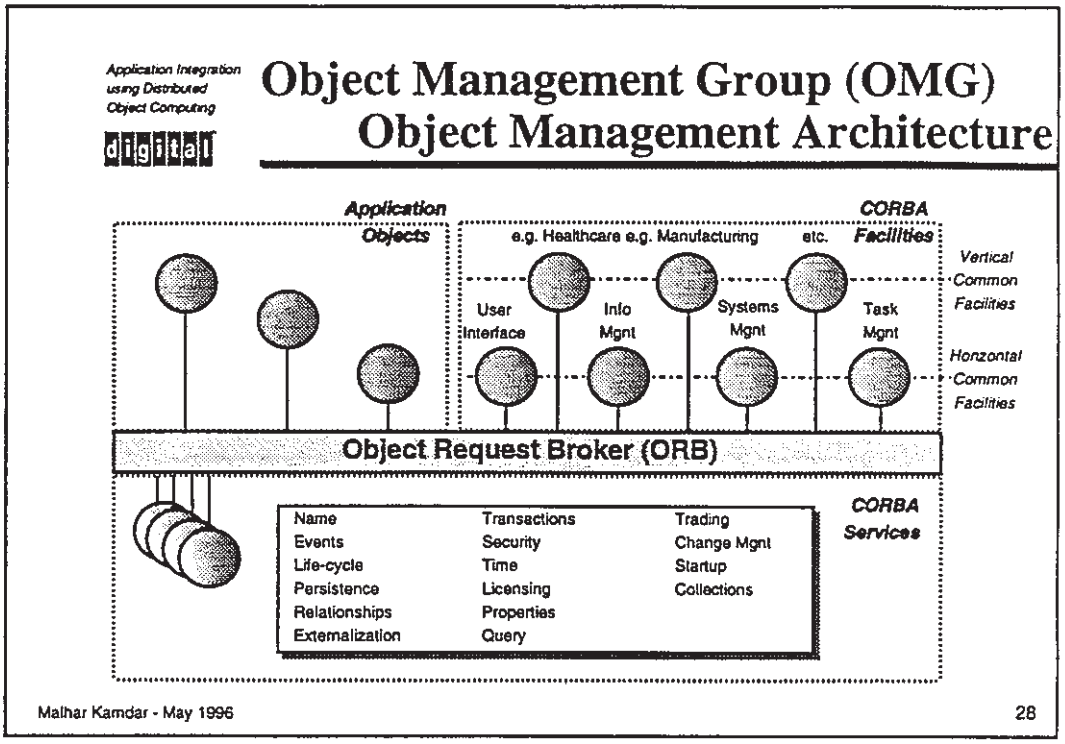


オブジェクト・マネージメント・グループ（ＯＭＧ）

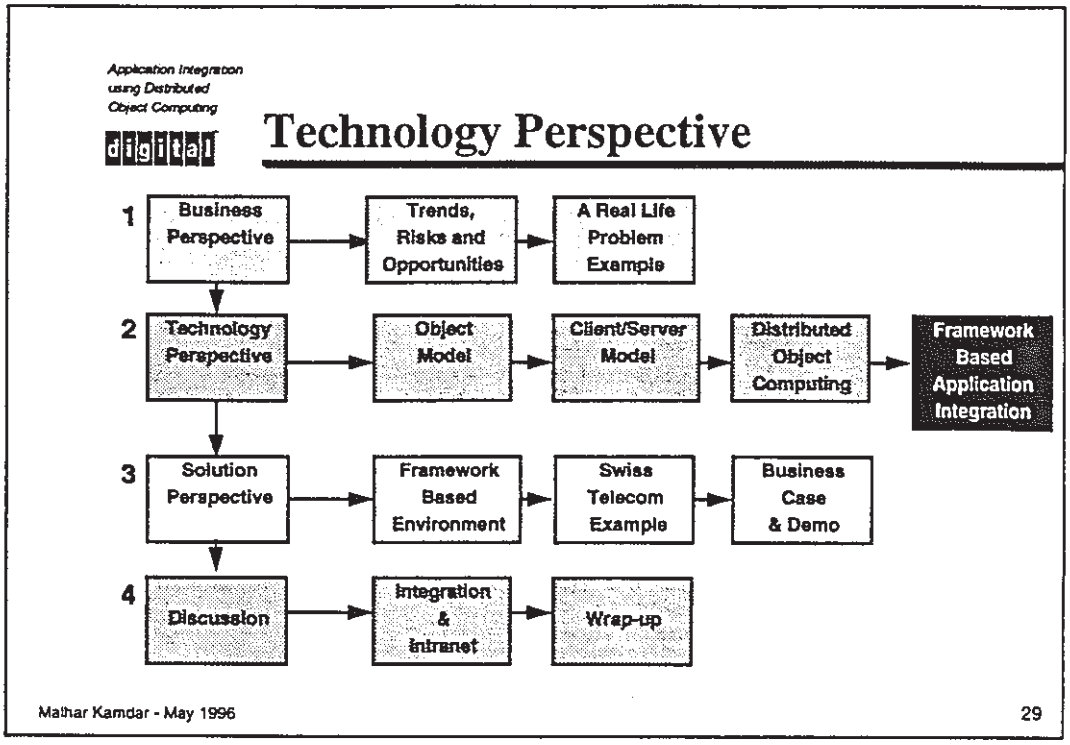
ＯＭＧは、分散コンピューティング・システムの開発のための理論の深化と、オブジェクト技術の実践を目的とした国際的な非営利団体

約５００社が参加している

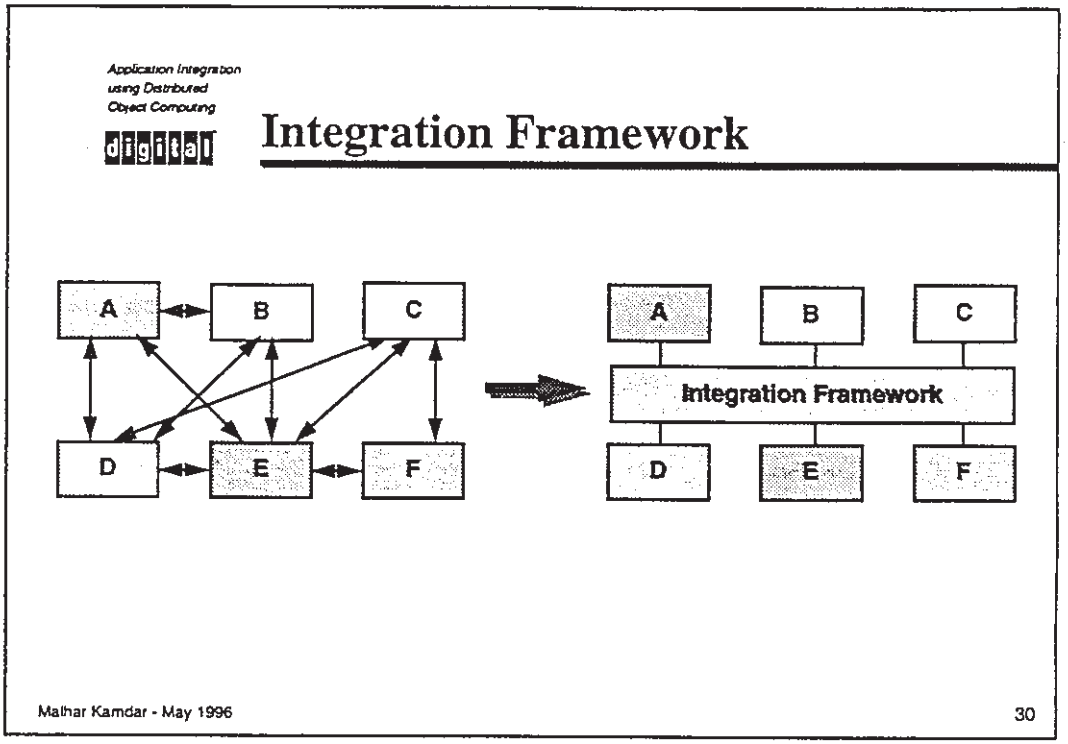
多くのオブジェクト・リクエスト・ブローカが製品化されている（DEC ObjectBroker, IBM SOM/DSOM, SunSoft NEO, ICL DAIS, Iona Orbix）



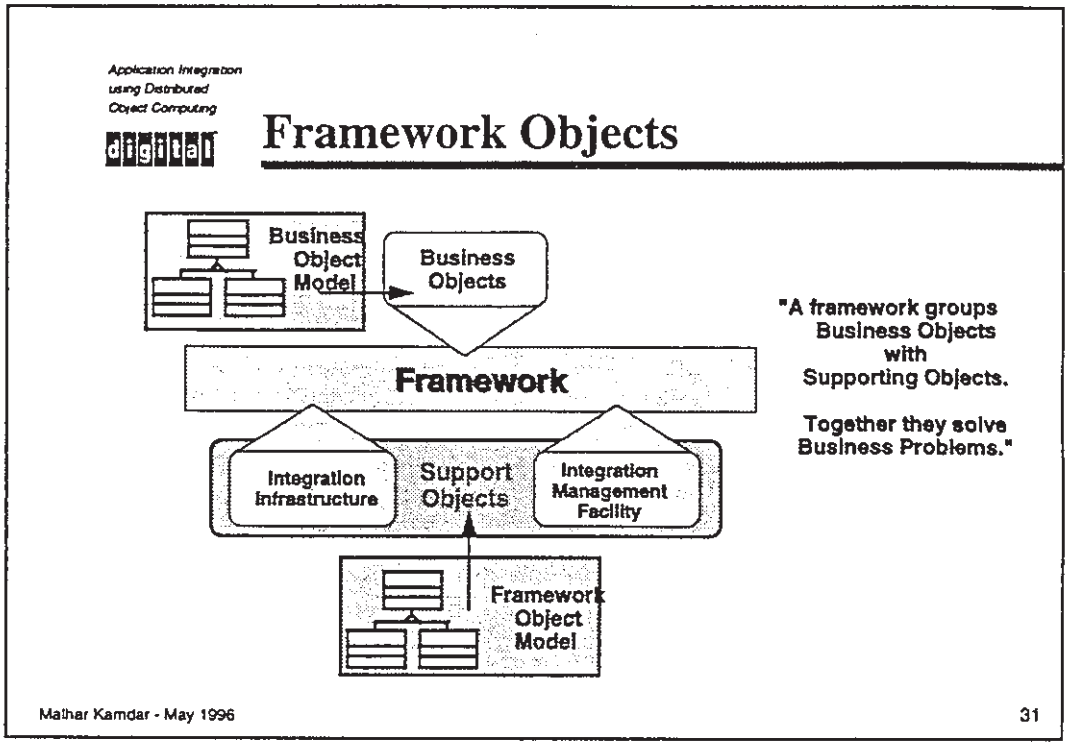
オブジェクト・マネージメント・アーキテクチャ



テクノロジの展望 － フレームワーク・ベースの統合

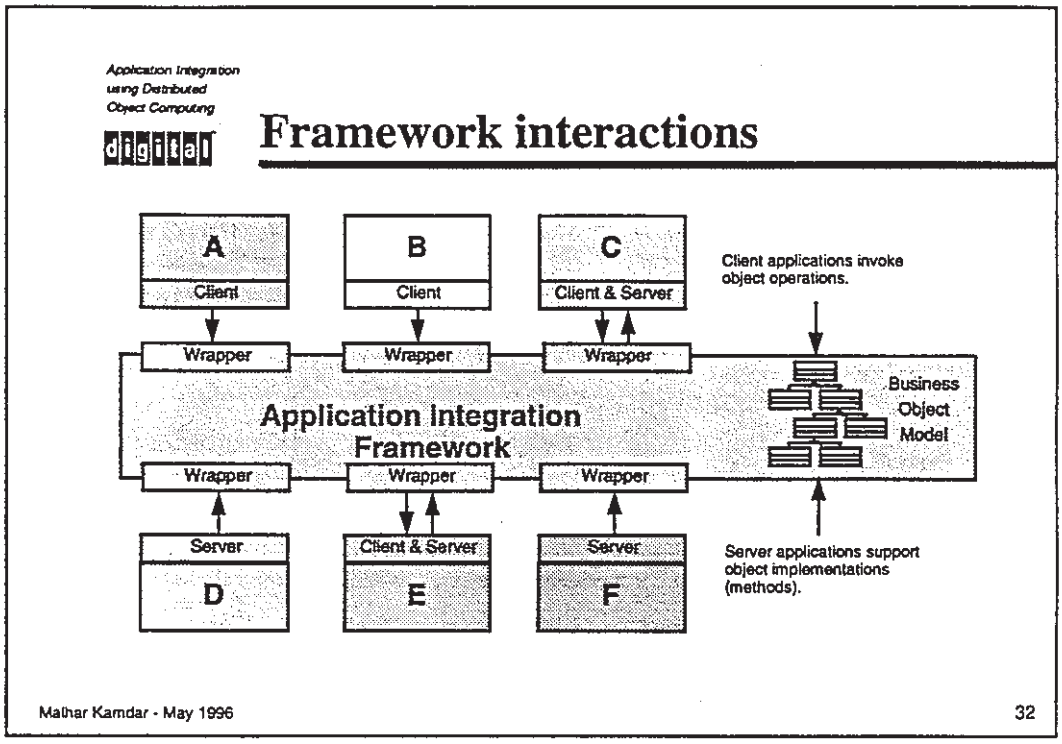

Application Integration
using Distributed
Object Computing
digital
Integration Framework
A
B
C
D
E
F
A
B
C
Integration Framework
D
E
F
Malhar Kamdar - May 1996
30

統合フレームワーク

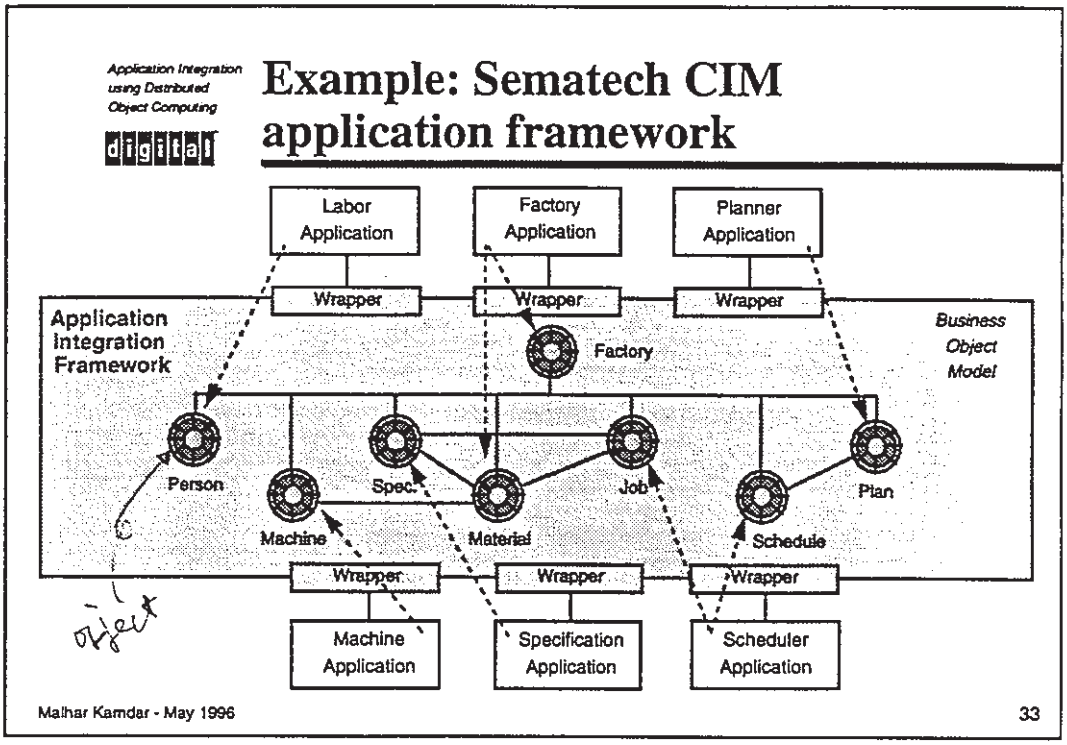


フレームワーク・オブジェクト

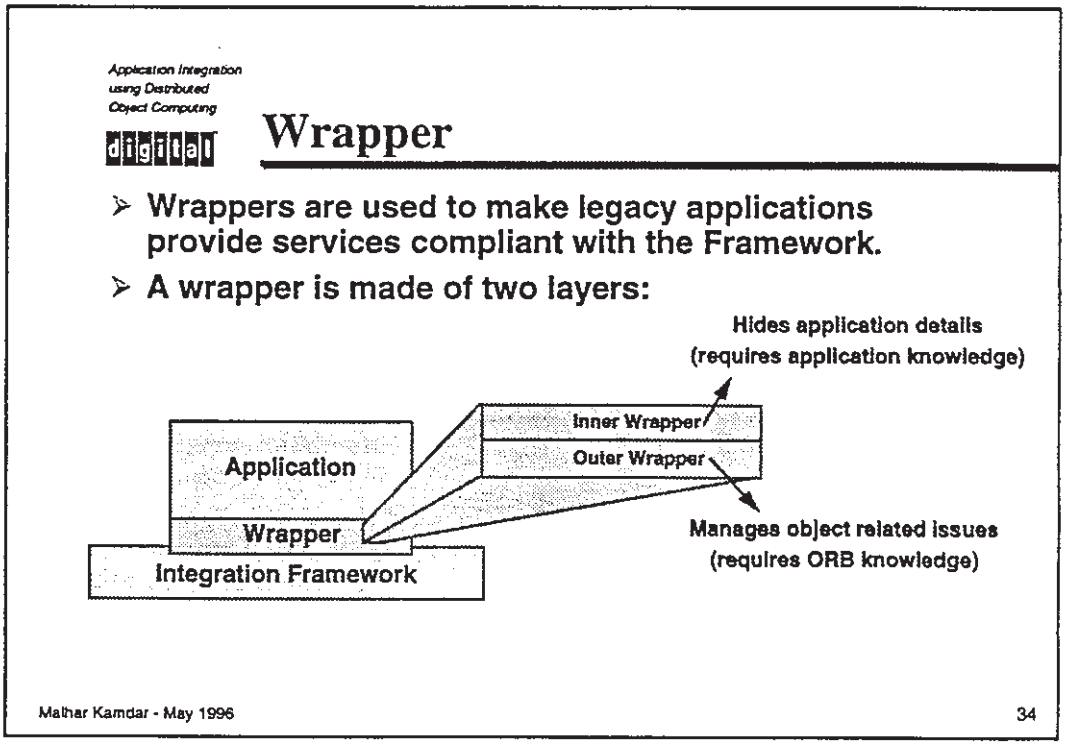
フレームワークは、ビジネス・オブジェクトにサポート・オブジェクトから成り、ビジネスの問題を解決する



フレームワークの相互作用
クライアント・アプリケーションはオブジェクトのオペレーションを発行する
サーバ・アプリケーションはオブジェクトのインプリメンテーションをサポートする



事例：半導体ＣＩＭアプリケーション・フレームワーク

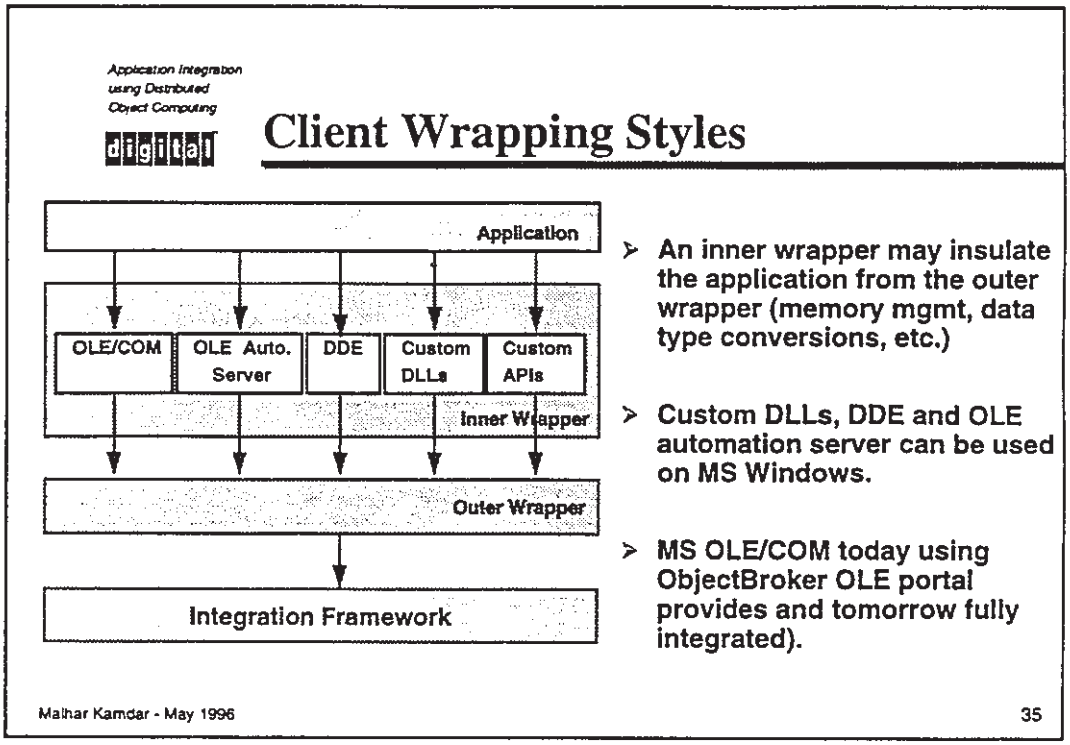


ラッパー

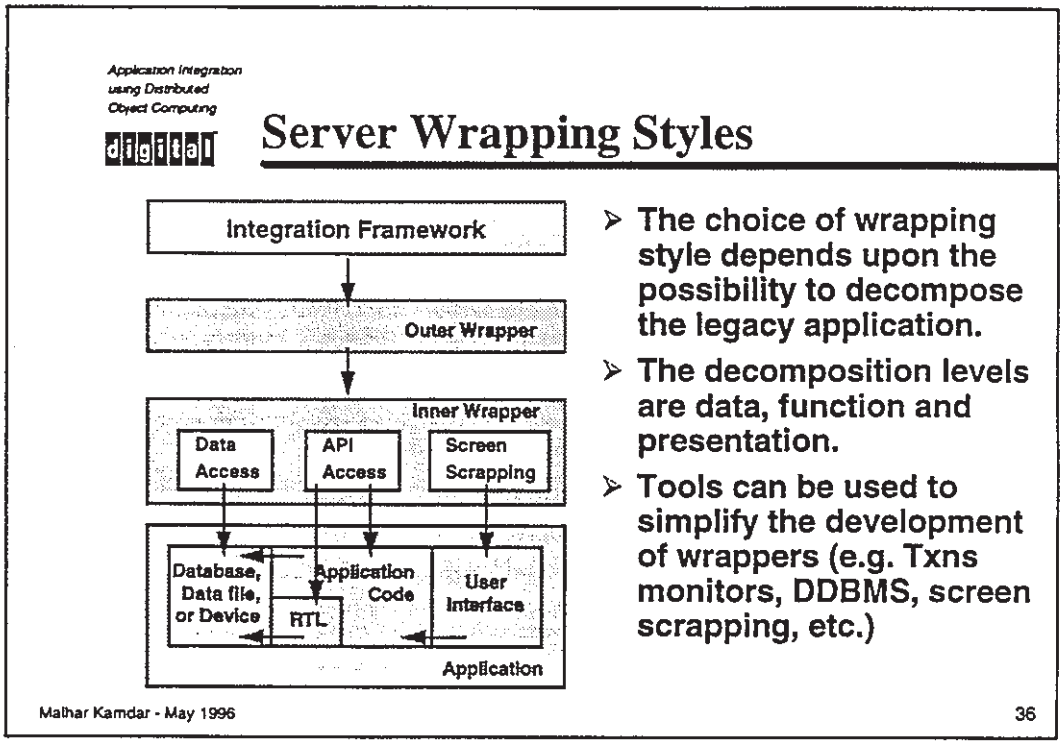
ラッパーはレガシー・アプリケーションが提供するサービスをフレームワークに適合させる

ラッパーは2層からなる：

- インナー・ラッパー：アプリケーションの詳細を隠蔽する
- アウター・ラッパー：オブジェクトに関する事柄を管理する



クライアントのラッピング・スタイル
インナー・ラッパーはアプリケーションをアウター・ラッパーから隔離する
カスタム DLL,DDE,OLE オートメーション・サーバは MS Windows から使用できる
現在 OLE/COM は ObjectBroker の OLE Portal を使用する

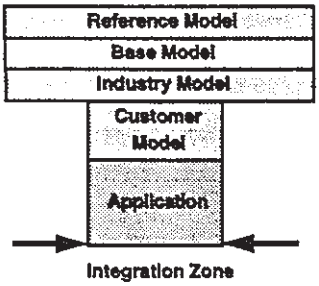


サーバのラッピング・スタイル
ラッピング・スタイルの選択は、レガシー・アプリケーションがどれだけ分解できるかによる
分解のレベルはデータ、ファンクション、プレゼンテーションである
ツールはラッパーの開発を容易にする

Application Integration using Distributed Object Computing

digital

Reference Models



- ***Reference Model*** - Describes system objects and object primitives.
- ***BASE Model*** - Describes predominant types utilized across different models; e.g. naming services.
- ***Industry Model*** - Extensions to the domain model that defines specific industry types; e.g. pharmaceutical.
- ***Customer Model*** - Extensions to the domain that defines specific customer types; e.g. Ciba, Roche.
- ***Application*** - Uses a particular version of the Customer Model.
- ***Integration Zone*** - Delimits the scope around the applications which, when integrated, carry the most strategic value to the customer.

Malhar Kamdar - May 1996 37

リファレンス・モデル

リファレンス・モデル ー システム・オブジェクトとオブジェクトのプリミティブ

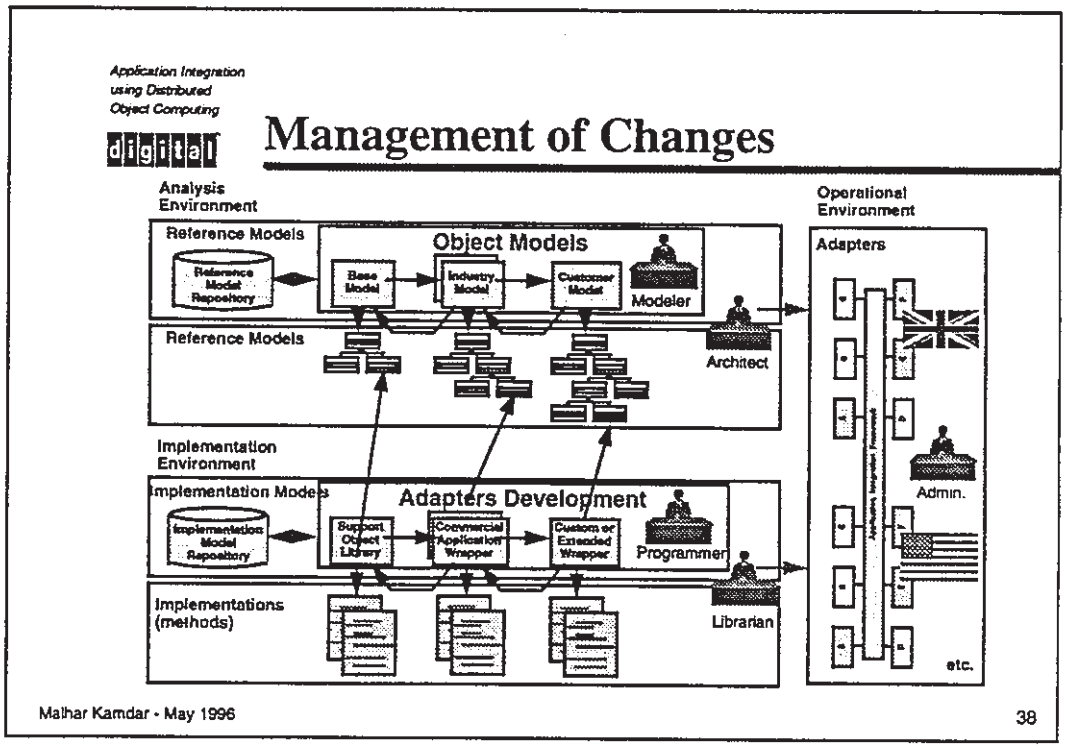
ベース・モデル ー 異なるモデル間で利用できる主なタイプ

インダストリ・モデル ー 特定の業界のタイプを定義したドメイン・モデルの拡張

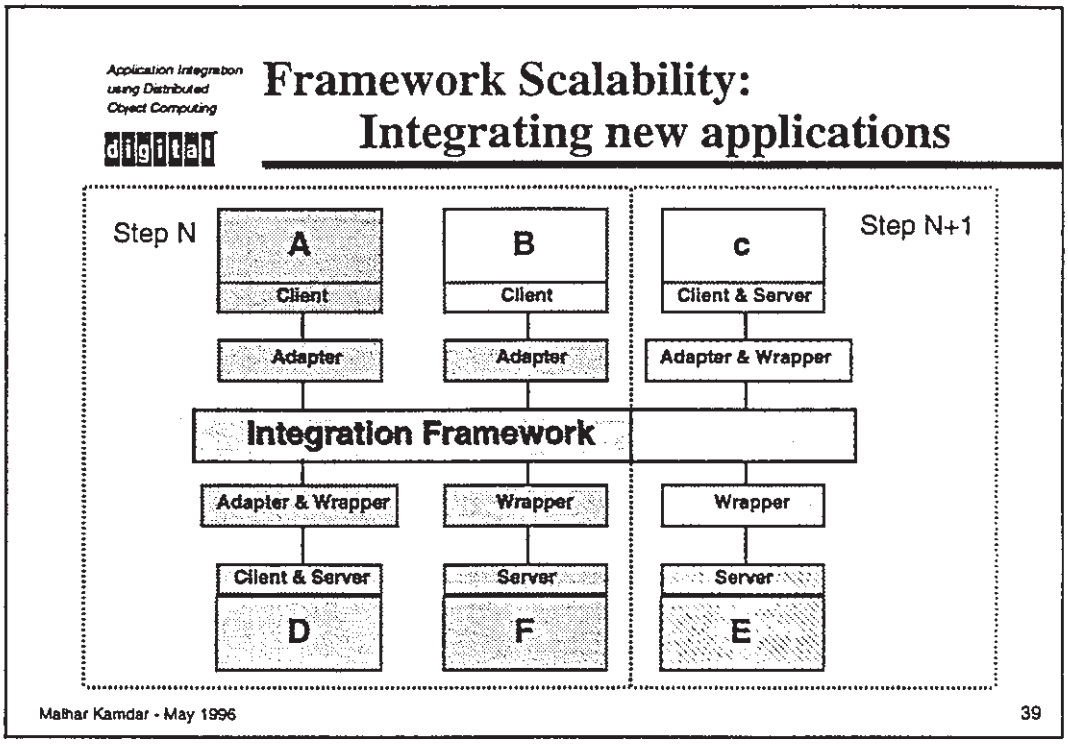
カスタマ・モデル ー 特定のカスタマのタイプを定義したドメイン・モデルの拡張

アプリケーション ー カスタマ・モデルの特定のバージョン

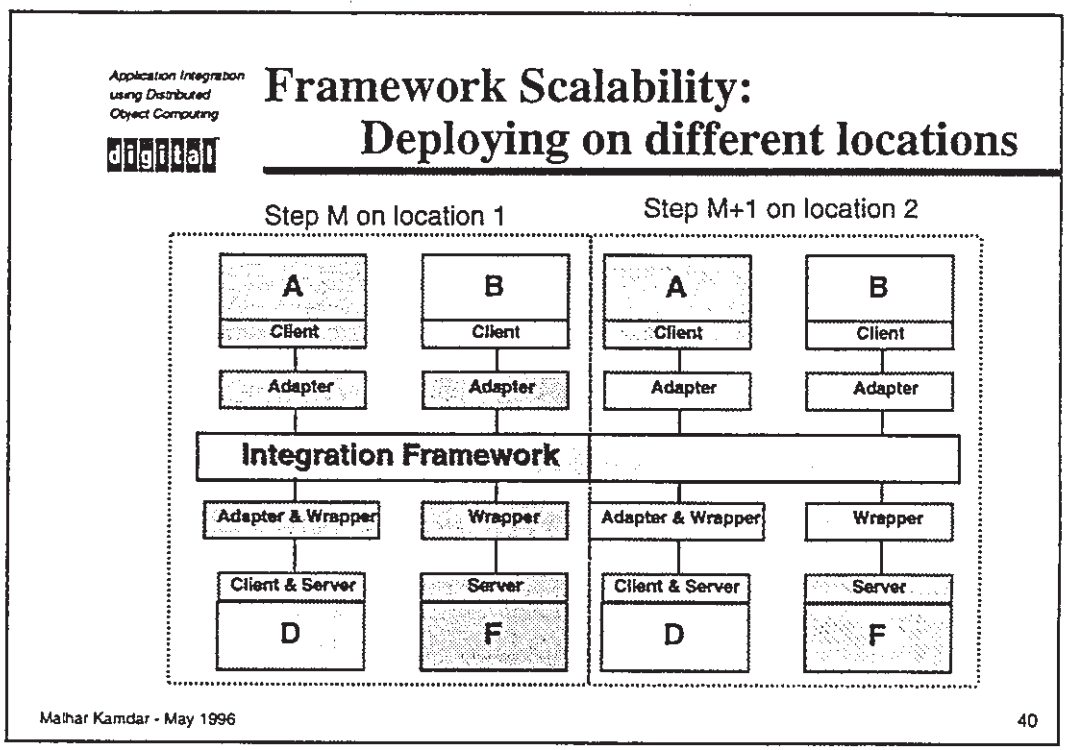
インテグレーション・ゾーン ー アプリケーション統合により、カスタマにとって最も戦略的に価値がある範囲


Application Integration using Distributed Object Computing
digital
Management of Changes
Analysis Environment
Reference Models
Reference Model Repository
Object Models
Base Model
Industry Model
Customer Model
Modeler
Reference Models
Architect
Operational Environment
Adapters
Admin.
etc.
Implementation Environment
Implementation Models
Implementation Model Repository
Adapters Development
Support Object Library
Commercial Application Wrapper
Custom or Extended Wrapper
Programmer
Implementations (methods)
Librarian
Malhar Kamdar - May 1996
38

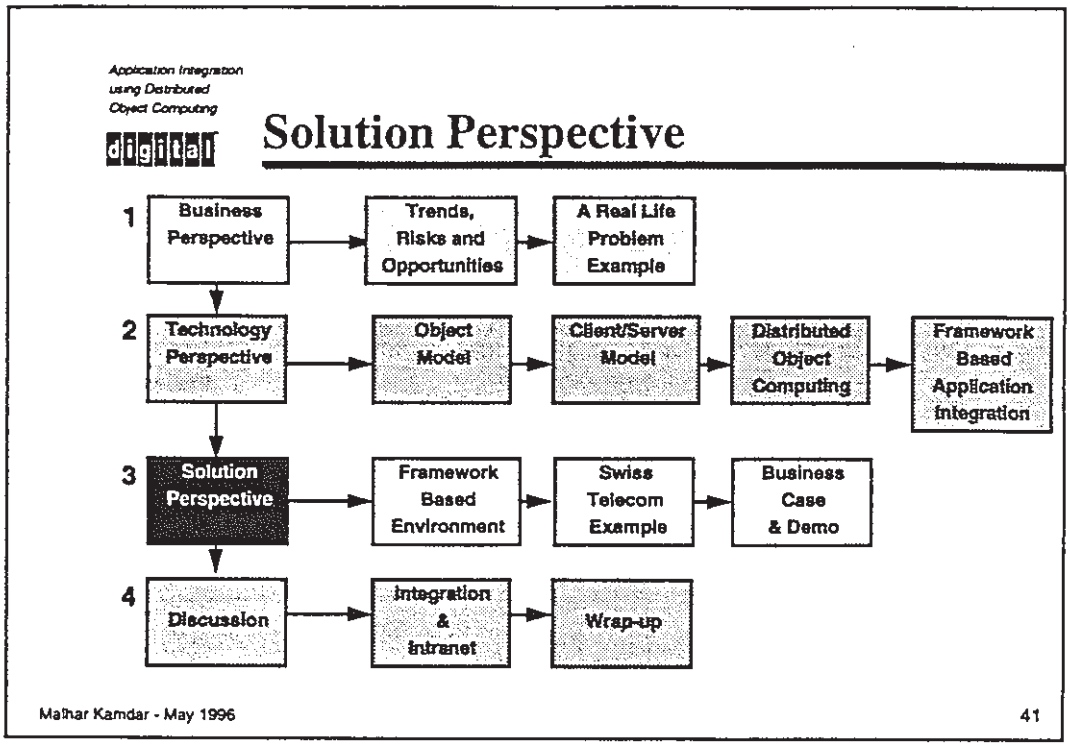
変更の管理



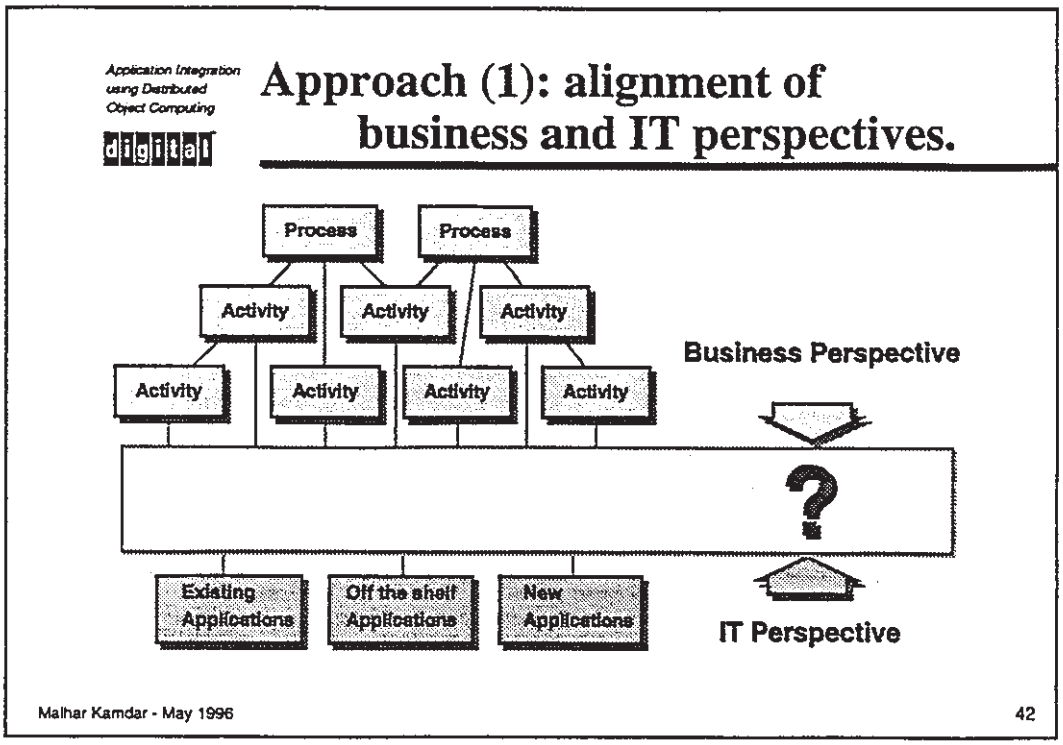
フレームワークの拡張性 － 新たにアプリケーションを統合する



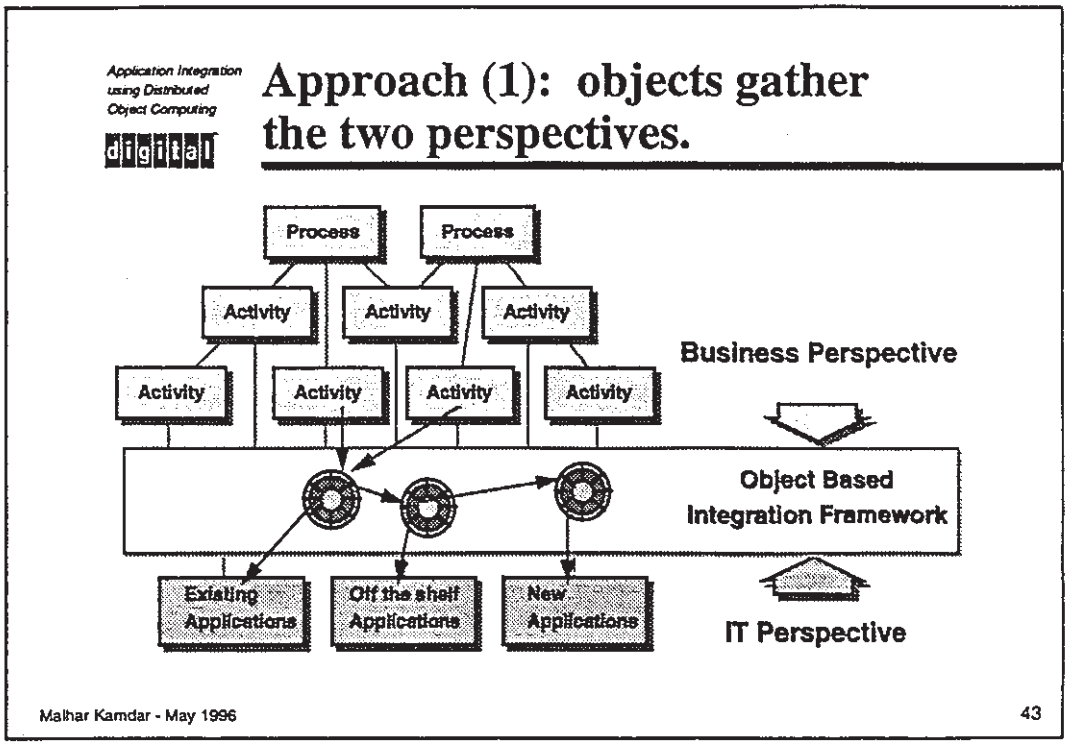
フレームワークの拡張性 － 異なる場所へ展開する


Application Integration using Distributed Object Computing
digital
Solution Perspective
1
Business Perspective
Trends, Risks and Opportunities
A Real Life Problem Example
2
Technology Perspective
Object Model
Client/Server Model
Distributed Object Computing
Framework Based Application Integration
3
Solution Perspective
Framework Based Environment
Swiss Telecom Example
Business Case & Demo
4
Discussion
Integration & Intranet
Wrap-up
Malhar Kamdar - May 1996
41

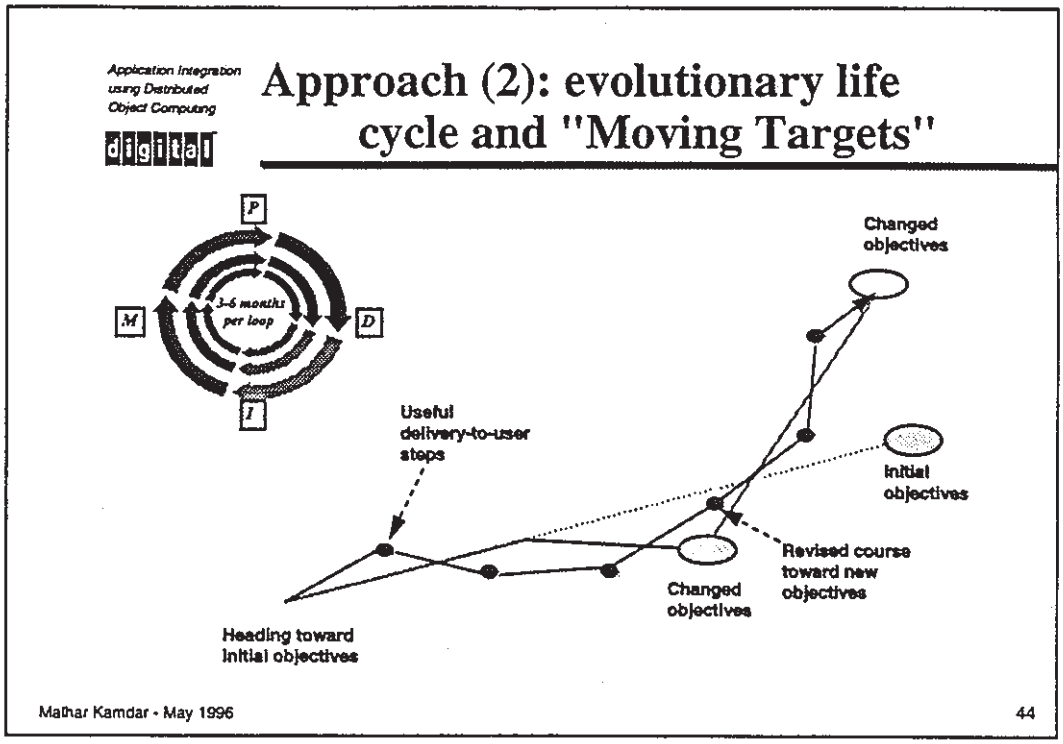
ソルーションの展望



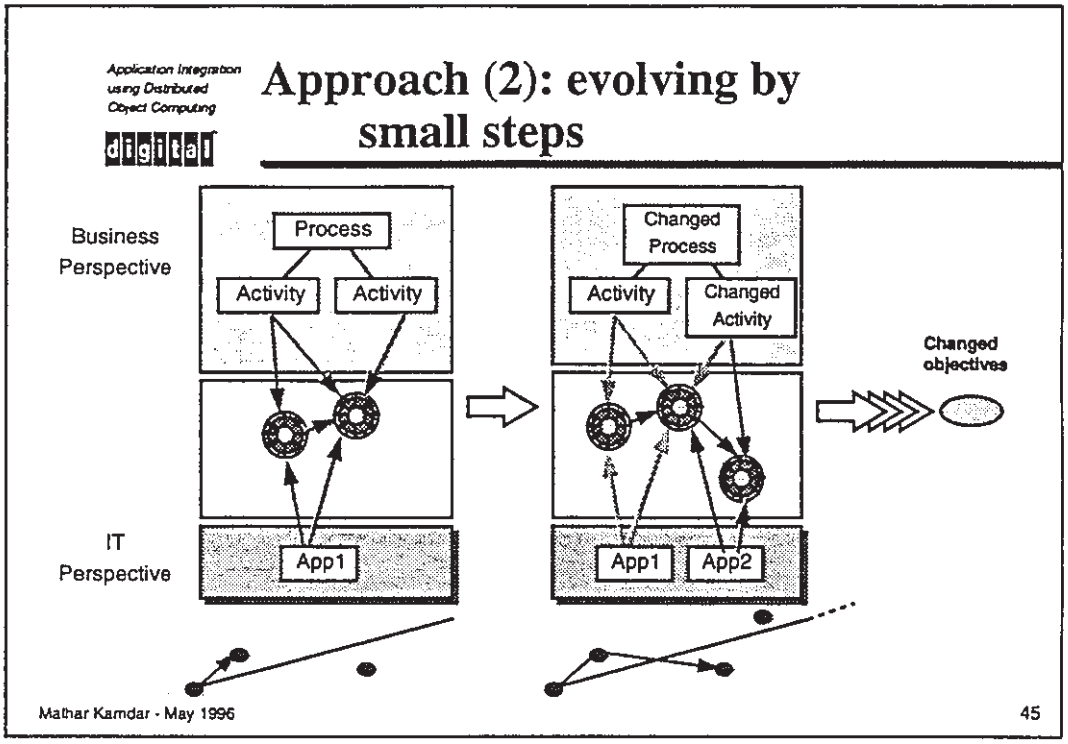
アプローチ（１）：ビジネスとＩＴの観点を提携させる



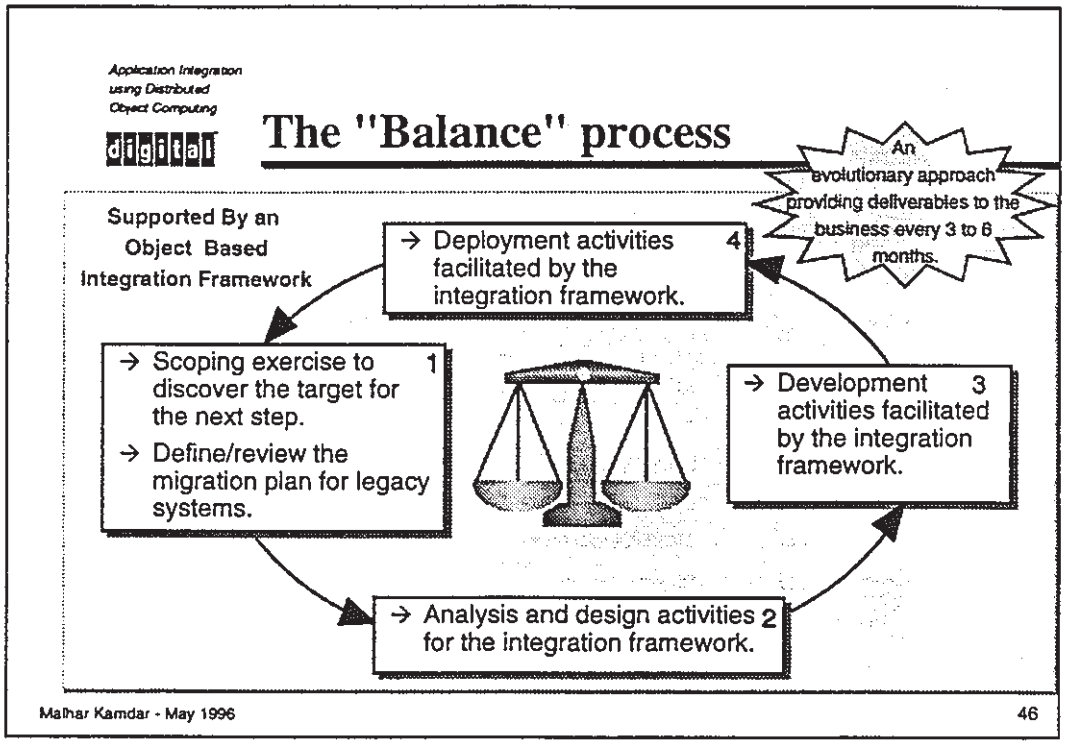
アプローチ（1）：オブジェクトは二つの観点を一つにする



アプローチ（2）：漸進的なライフ・サイクルと”動く標的”



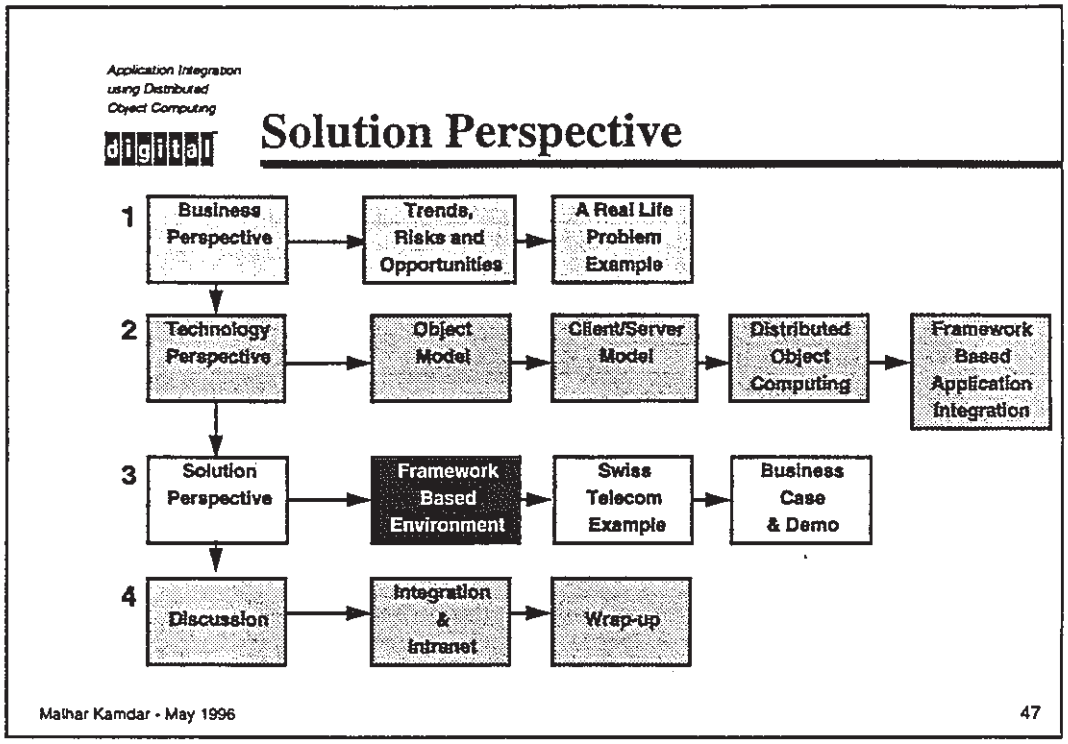
アプローチ（２）：わずかな歩みにより漸進



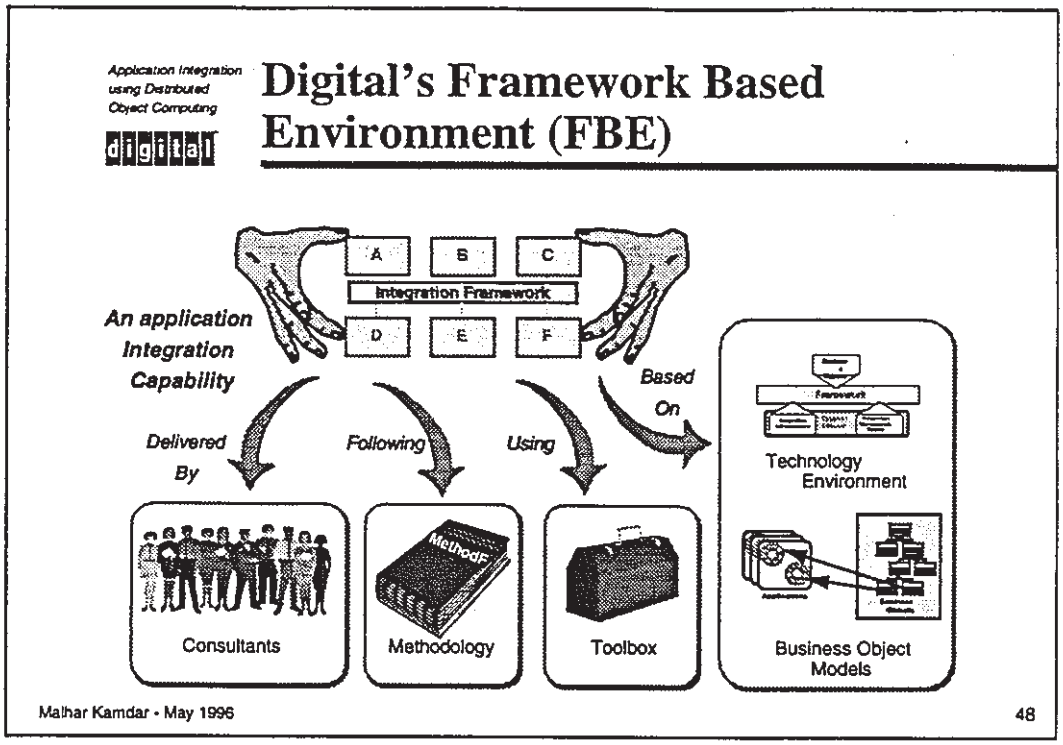
"バランス"プロセス

1. 次のステップのターゲットを見つけだし範囲を定める
   レガシー・システムの移行計画を定義／レビューする
2. 統合フレームワークの分析／設計作業
3. 統合フレームワークの開発作業
4. 統合フレームワークの展開作業

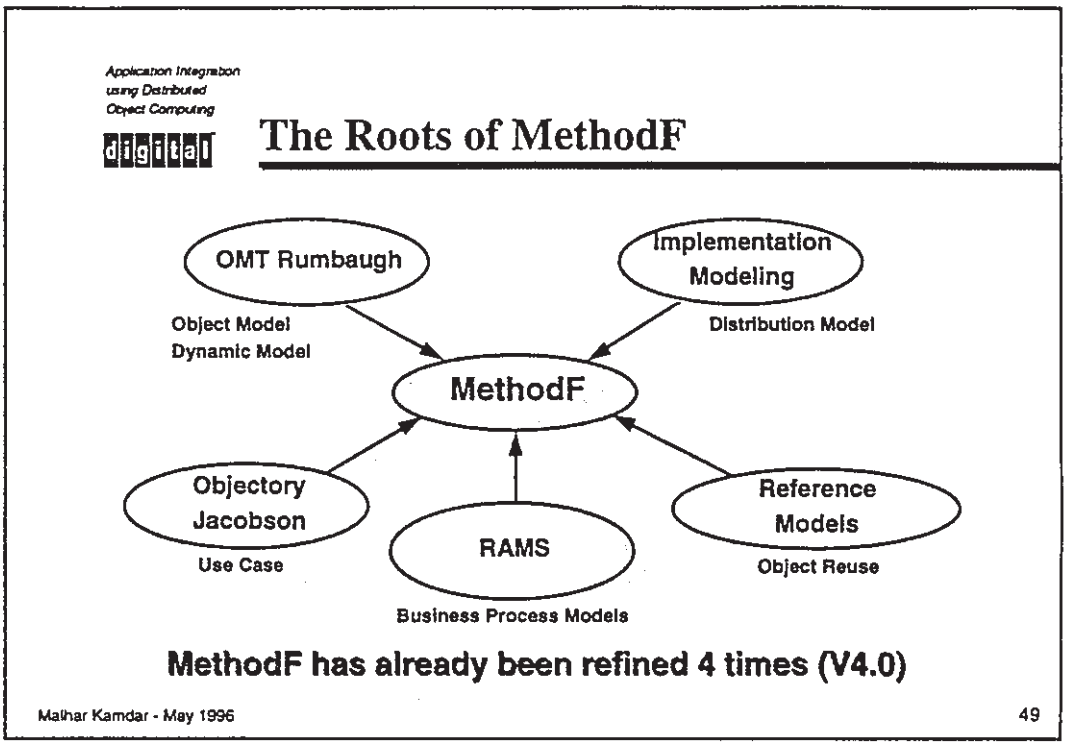
漸進的アプローチでは3－6ヶ月ごとに成果を提供



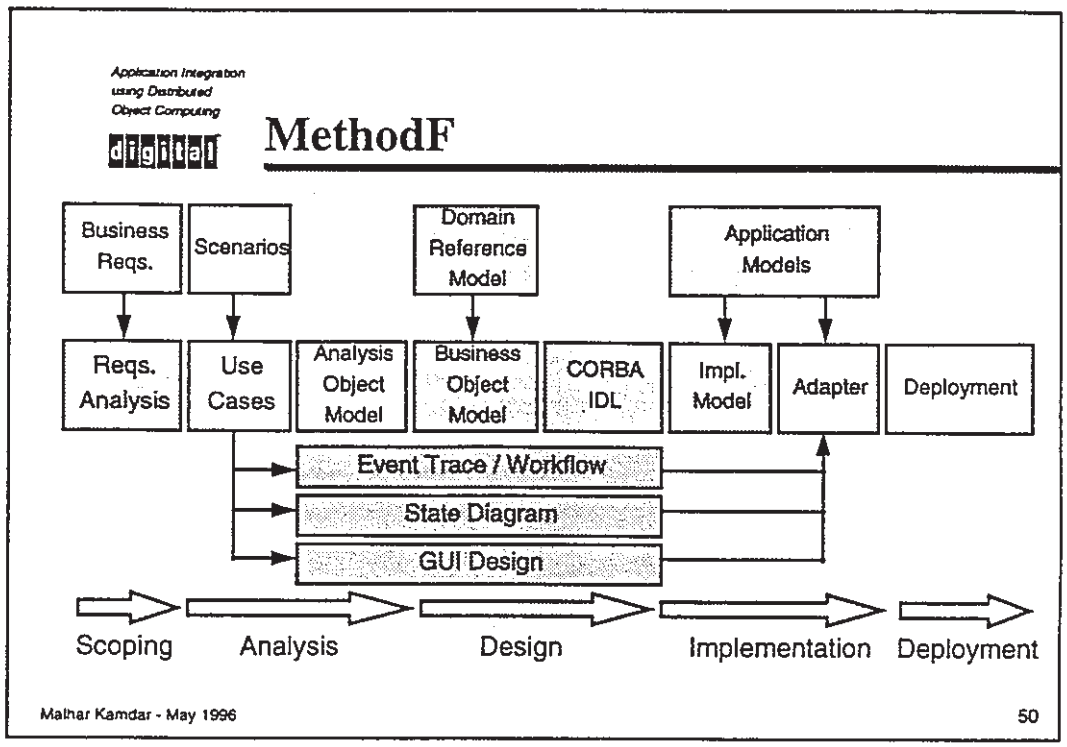
ソルーションの展望 － フレームワーク・ベースの環境



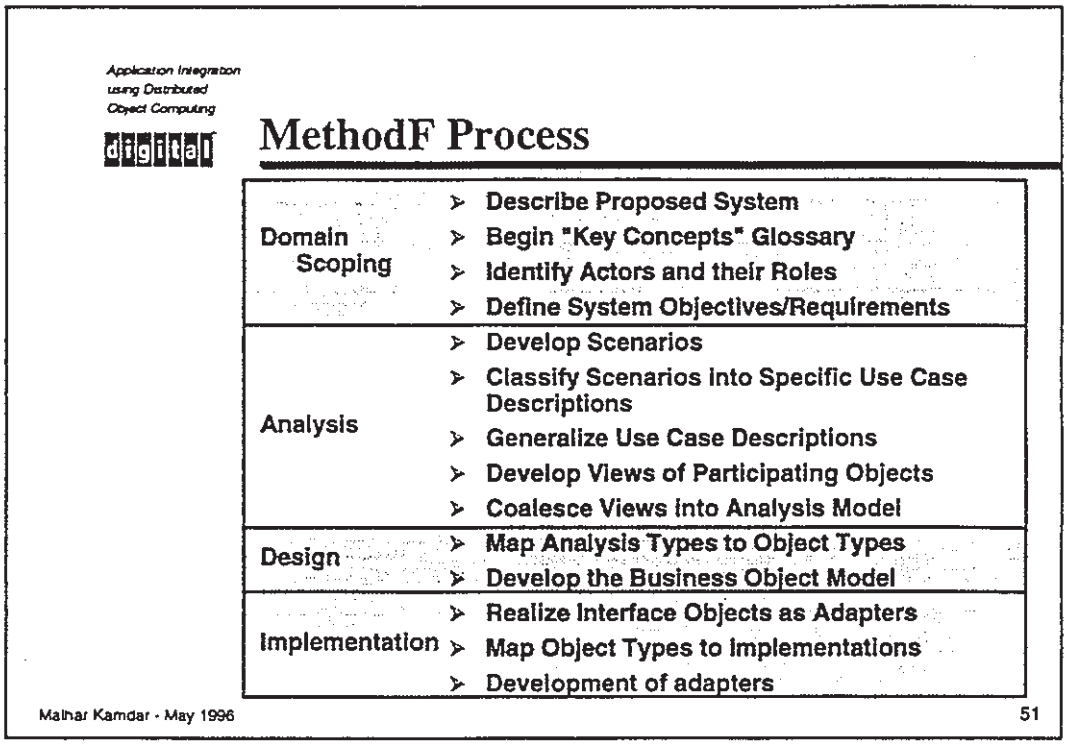
ＤＥＣの提供するフレームワーク・ベースの環境（ＦＢＥ）
アプリケーション統合力



ＭｅｔｈｏｄＦのルーツ
MethodFはすでに４度のバージョン・アップがなされている


Application Integration using Distributed Object Computing
digital
MethodF
Business Reqs.
Scenarios
Domain Reference Model
Application Models
Reqs. Analysis
Use Cases
Analysis Object Model
Business Object Model
CORBA IDL
Impl. Model
Adapter
Deployment
Event Trace / Workflow
State Diagram
GUI Design
Scoping
Analysis
Design
Implementation
Deployment
Malhar Kamdar - May 1996
50

M e t h o d F



ＭｅｔｈｏｄＦのプロセス

範囲の設定

- 対象のシステムを記述する
- キー・コンセプトの用語集から始める
- アクタと役割を見極める
- システムの目的と要求を定義する

分析

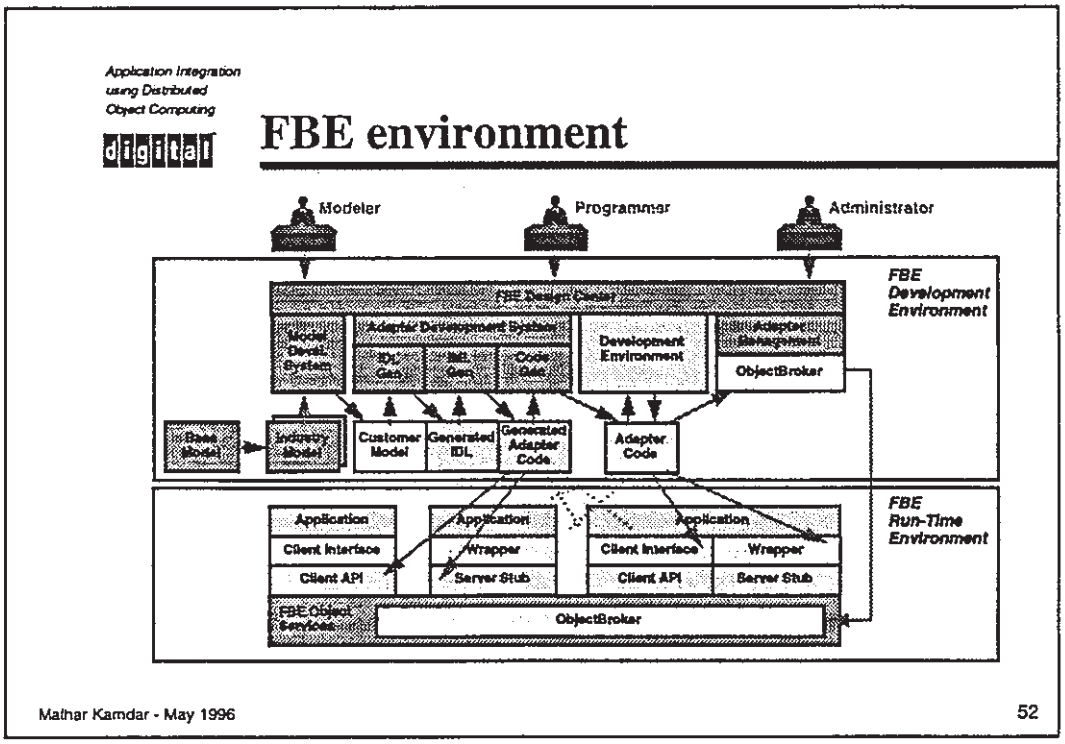
- シナリオの作成
- シナリオを特定のUse Caseの説明に分類する
- Use Caseの説明の汎化を行う
- 関係するオブジェクトのビューを作成する
- 分析モデルにビューを融合する

設計

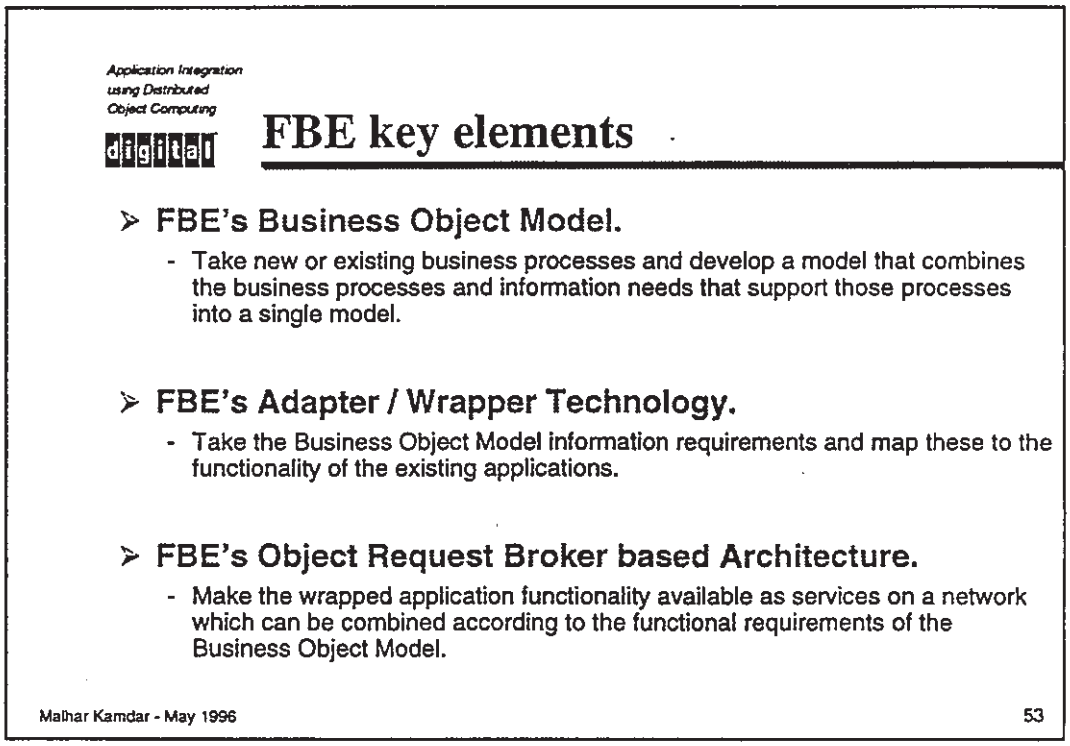
- オブジェクト・タイプに分析タイプをマップする
- ビジネス・オブジェクト・モデルを作成する

実装

- インタフェース・オブジェクトはアダプタとして実現する
- オブジェクト・タイプの実装
- アダプタの作成


Application Integration using Distributed Object Computing
digital
FBE environment
Modeler
Programmer
Administrator
FBE Development Environment
FBE Design Center
Adapter Development System
Development Environment
Adapter Management
ObjectBroker
Base Model
Industry Model
Customer Model
Generated IDL
Generated Adapter Code
Adapter Code
FBE Run-Time Environment
Application
Client Interface
Client API
Wrapper
Server Stub
FBE Object Services
ObjectBroker
Mathar Kamdar - May 1996
52

ＦＢＥの環境



ＦＢＥのキー・エレメント

・ＦＢＥのビジネス・オブジェクト・モデル

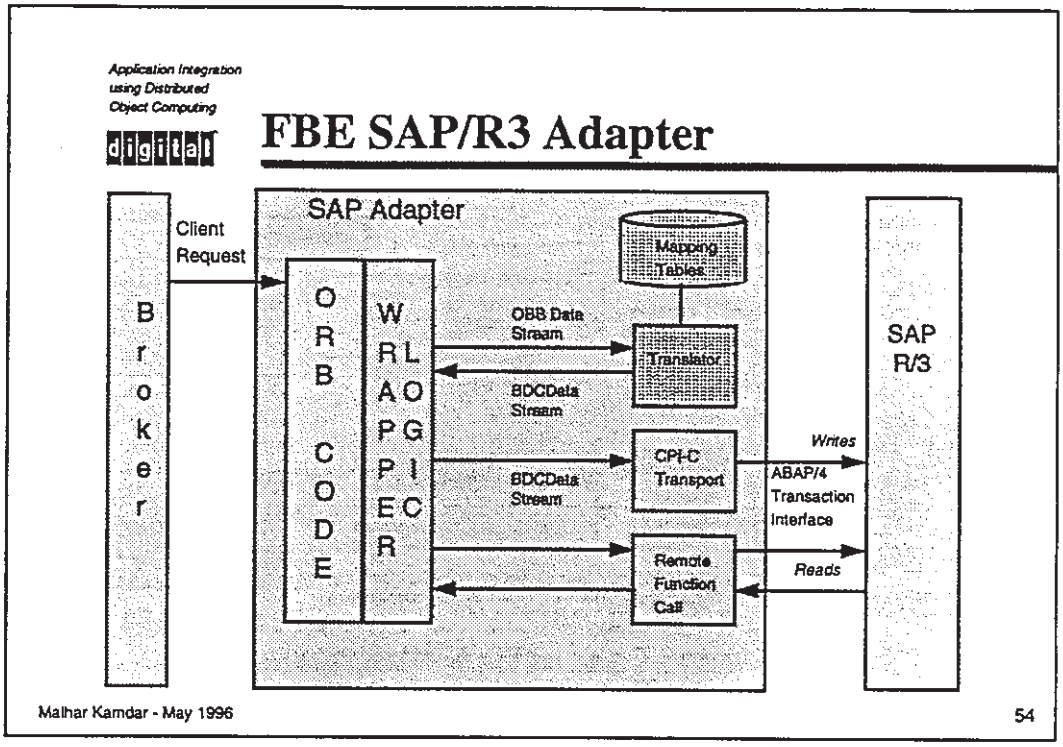
新規あるいは既存のビジネス・プロセスから、ビジネス・プロセスと、プロセスが提供するインフォメーション・ニーズを一つのモデルに結合することで、モデルを作成する

ＦＢＥのアダプタ／ラッパー・テクノロジー

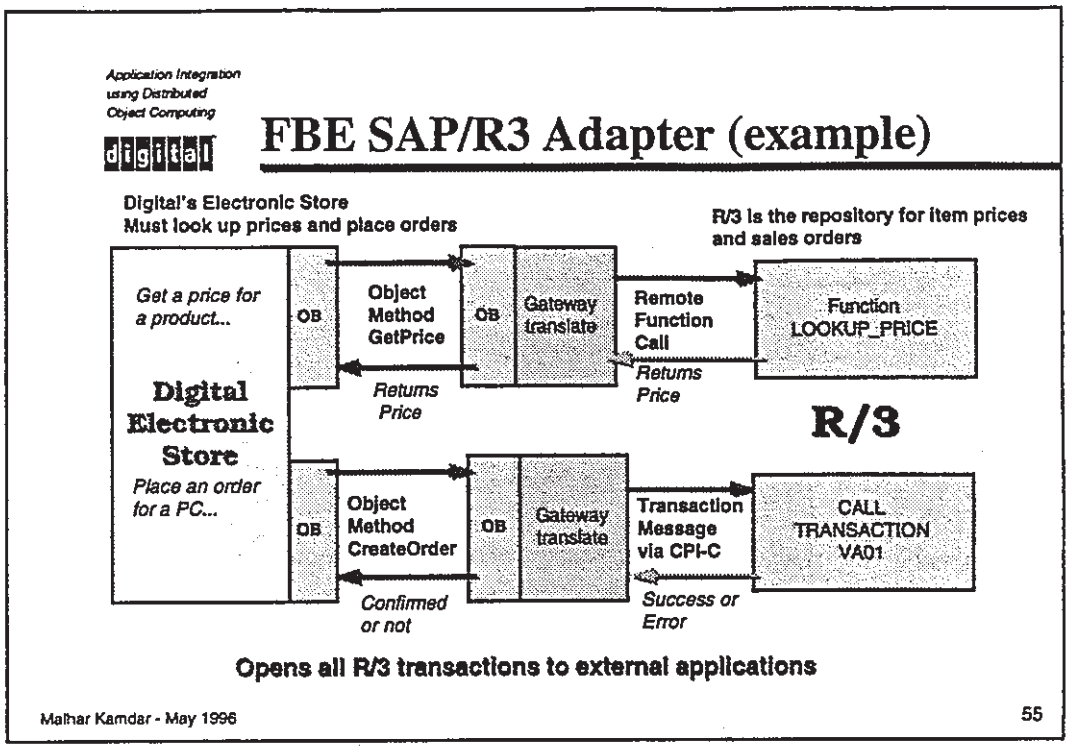
ビジネス・オブジェクト・モデルに、既存のアプリケーションの機能をマップする

ＦＢＥのＯＲＢベースのアーキテクチャ

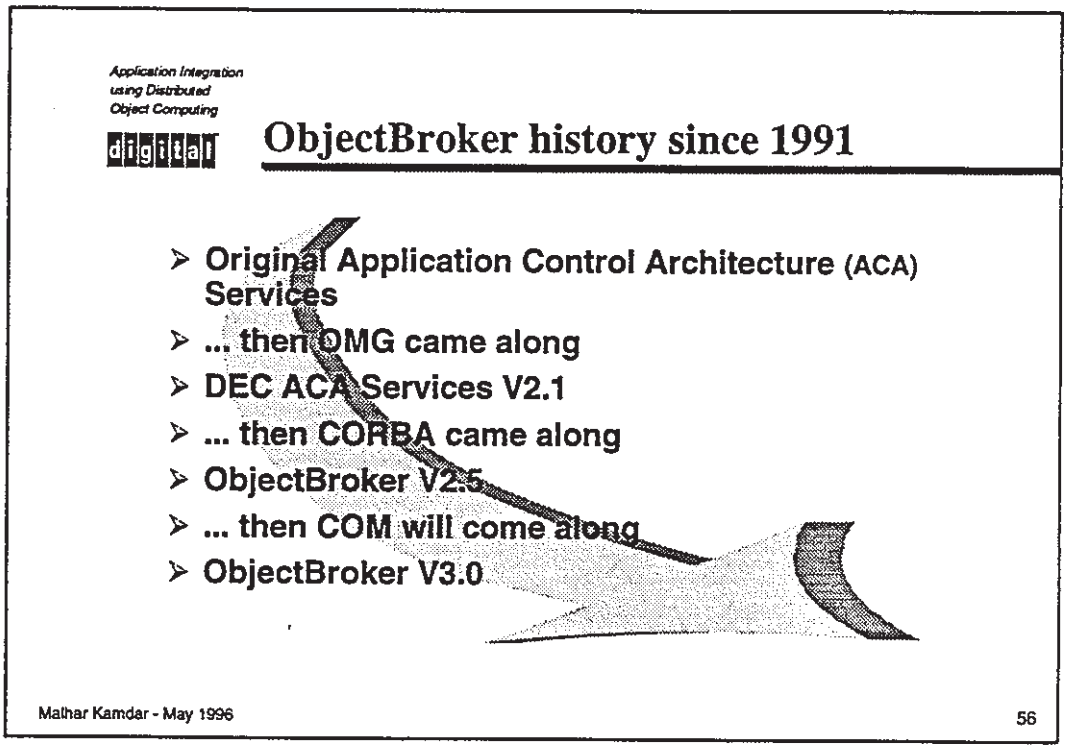
ビジネス・オブジェクト・モデルに従い結合されたアプリケーションの機能を、ラッピングしてサービスとしてネットワーク上に提供する



ＦＢＥのＳＡＰ／Ｒ３アダプター



ＦＢＥのＳＡＰ／Ｒ３アダプター（例）
すべてのＲ／３トランザクションを外部アプリケーションとしてオープンする



１９９１年からのObjectBrokerの歴史
ACA(Application Control Architecture)Servicesのオリジナル
. . . OMG の検討がが進められ
DEC ACA Services V2.1 リリース
. . . CORBA の検討が進められ
ObjectBroker V2.5 リリース
. . . COM の検討が進むと
ObjectBroker V3.0

Application Integration using Distributed Object Computing

digital

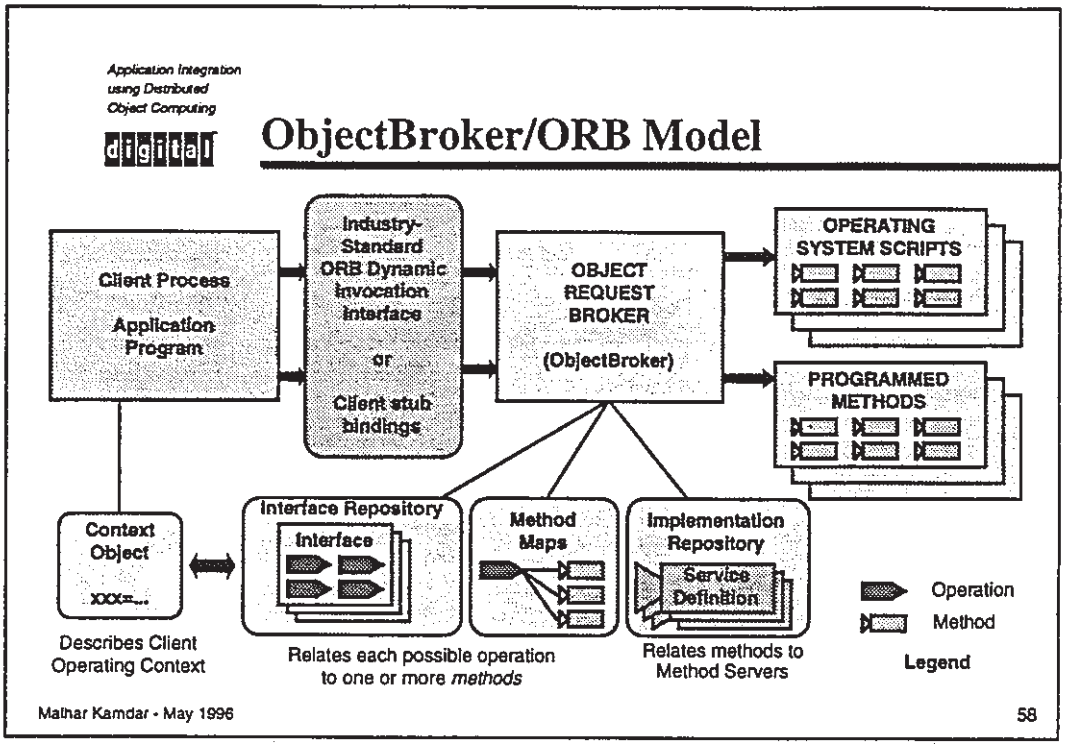
Platforms Supported by ObjectBroker

System	Local Agent	TCP/IP	DECnet
Mac. System 7	No	Yes	Yes
OpenVMS AXP	Yes	Yes	Yes
OpenVMS VAX	Yes	Yes	Yes
OS/2	Yes	Yes	No
OSF/1 AXP	Yes	Yes	No
Ultrix RISC	Yes	Yes	Yes
HP-UX	Yes	Yes	No
IBM AIX	Yes	Yes	No
MS Windows	No	Yes	Yes
MS NT AXP	Yes	Yes	Yes
MS NT Intel	Yes	Yes	Yes
SunOs	Yes	Yes	No
Sun Solaris	Yes	Yes	No
Tandem	Yes	Yes	No
MVS	Yes	Yes	No
AS/400	Yes	Yes	No
Windows 95	Yes	Yes	Yes

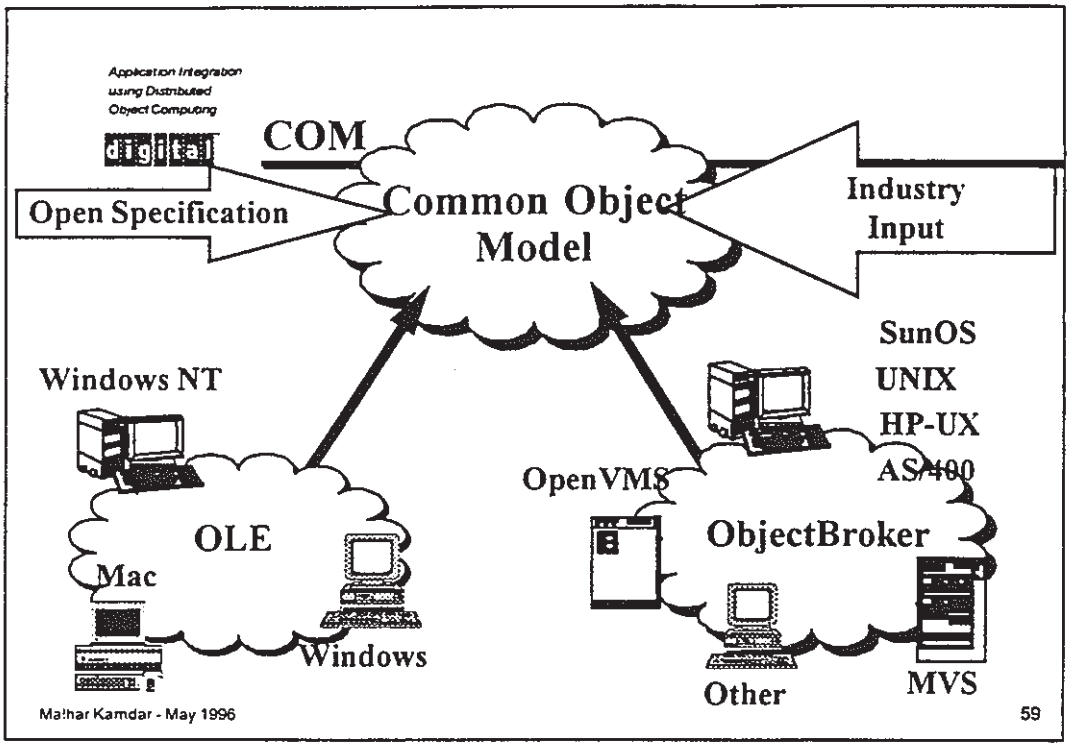
Malhar Kamdar - May 1996

57

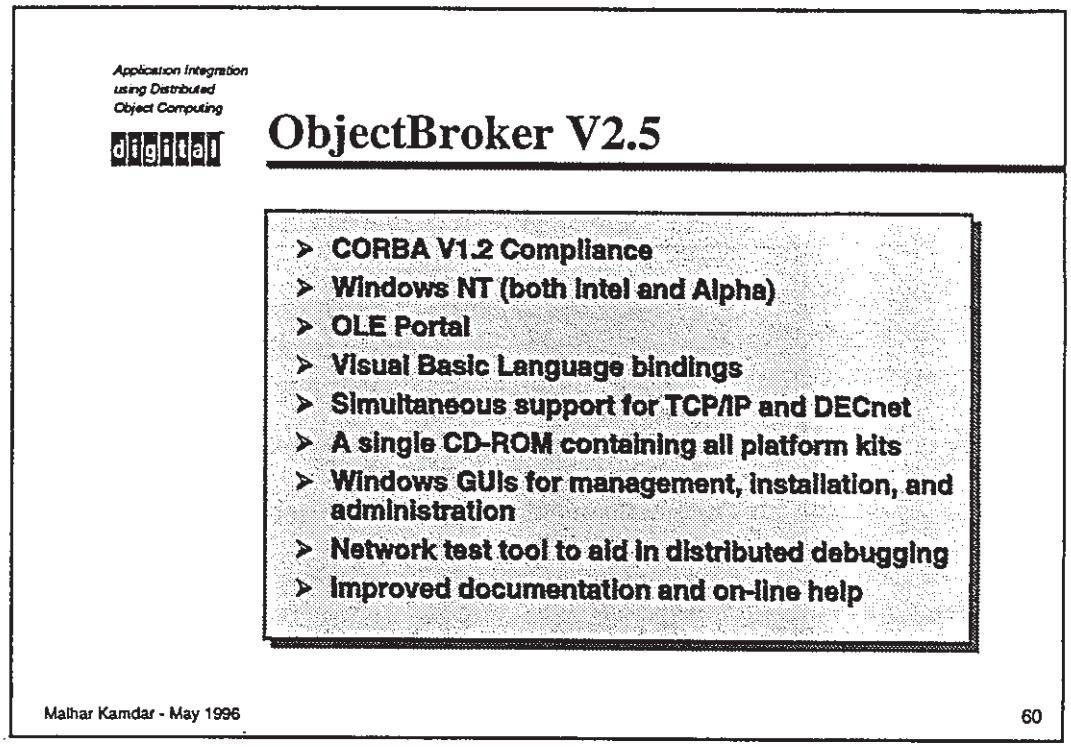
ＯｂｊｅｃｔＢｒｏｋｅｒでサポートされるプラットフォーム



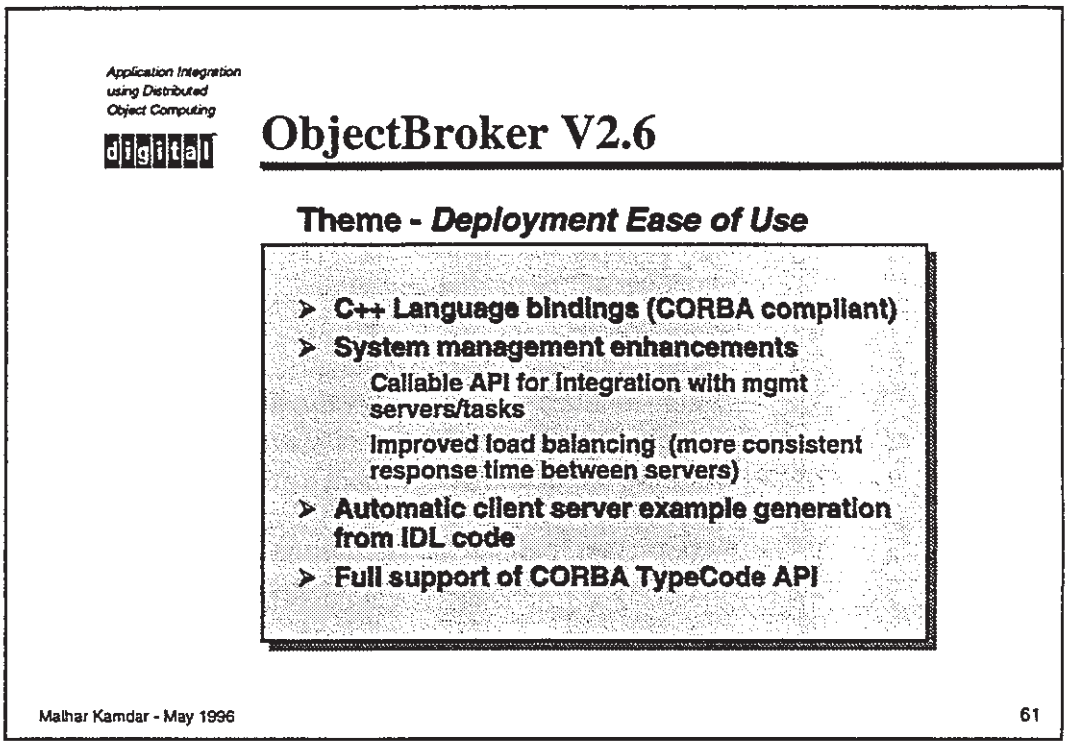
ＯｂｊｅｃｔＢｒｏｋｅｒ／ＯＲＢモデル


Application Integration
using Distributed
Object Computing
digital
COM
Open Specification
Common Object
Model
Industry
Input
SunOS
UNIX
HP-UX
AS/400
Windows NT
OpenVMS
OLE
Mac
Windows
ObjectBroker
Other
MVS
Malhar Kamdar - May 1996
59

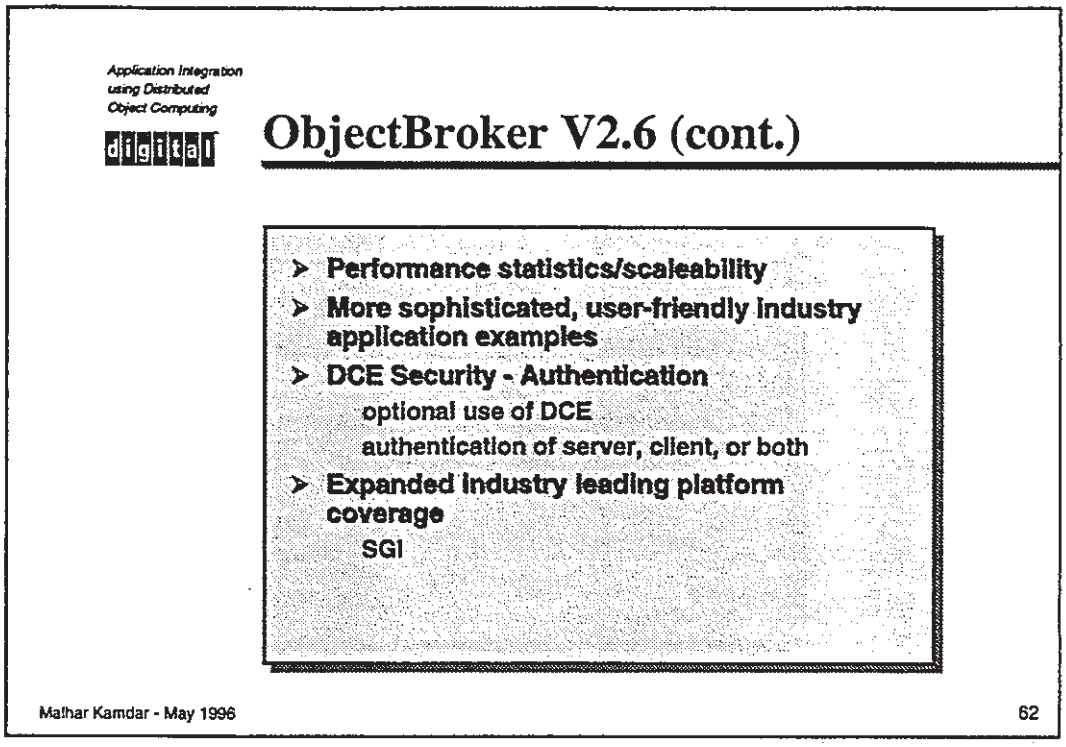
COM



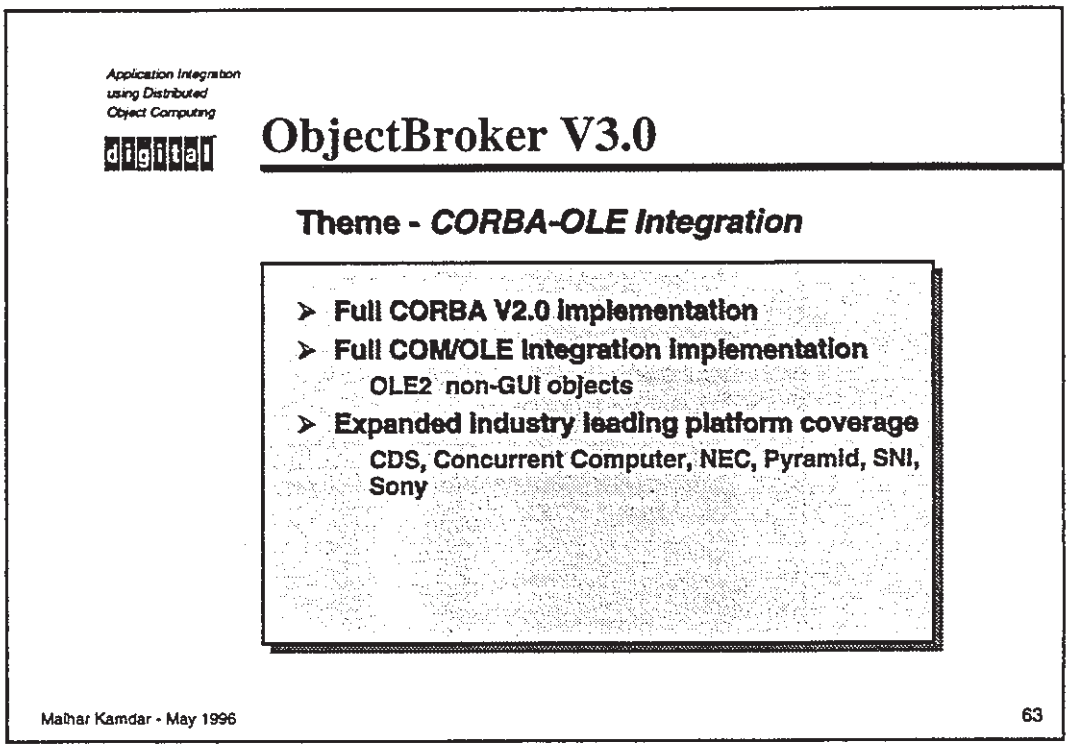
ＯｂｊｅｃｔＢｒｏｋｅｒ Ｖ２．５
CORBA V1.2 準拠
Windows NT(Intel,Alpha) サポート
OLE Portal サポート
Visual Basic 言語バインディング
TCP/IP と DECnet をサポート
一枚の CD-ROM にすべてのプラットフォームのキット
管理、インストール、運用のための Windows GUI ツール
分散デバッグを助けるネットワーク・テストのツール
改良が加えられたドキュメントとオンライン・ヘルプ



ＯｂｊｅｃｔＢｒｏｋｅｒ Ｖ２．６ － より使い易く
C++ 言語バインディング（CORBA 準拠）
システム管理機能の改善
　　管理サーバ／タスクを統合するコーラブルＡＰＩ
　　改良された負荷分散
ＩＤＬからクライアントとサーバのサンプルを自動生成
CORBA TypeCode APIをフル・サポート



ＯｂｊｅｃｔＢｒｏｋｅｒ Ｖ２．６（続き）
パフォーマンス統計と拡張性
より洗練された、使い易い業界アプリケーションのサンプル
ＤＣＥセキュリティ － 認証
　　ＤＣＥのオプション
　　サーバ、クライアント、あるいは両方の認証
サポート・プラットフォームの拡大
　　ＳＧＩ



ＯｂｊｅｃｔＢｒｏｋｅｒ Ｖ３．０ － ＣＯＲＢＡ－ＯＬＥ統合

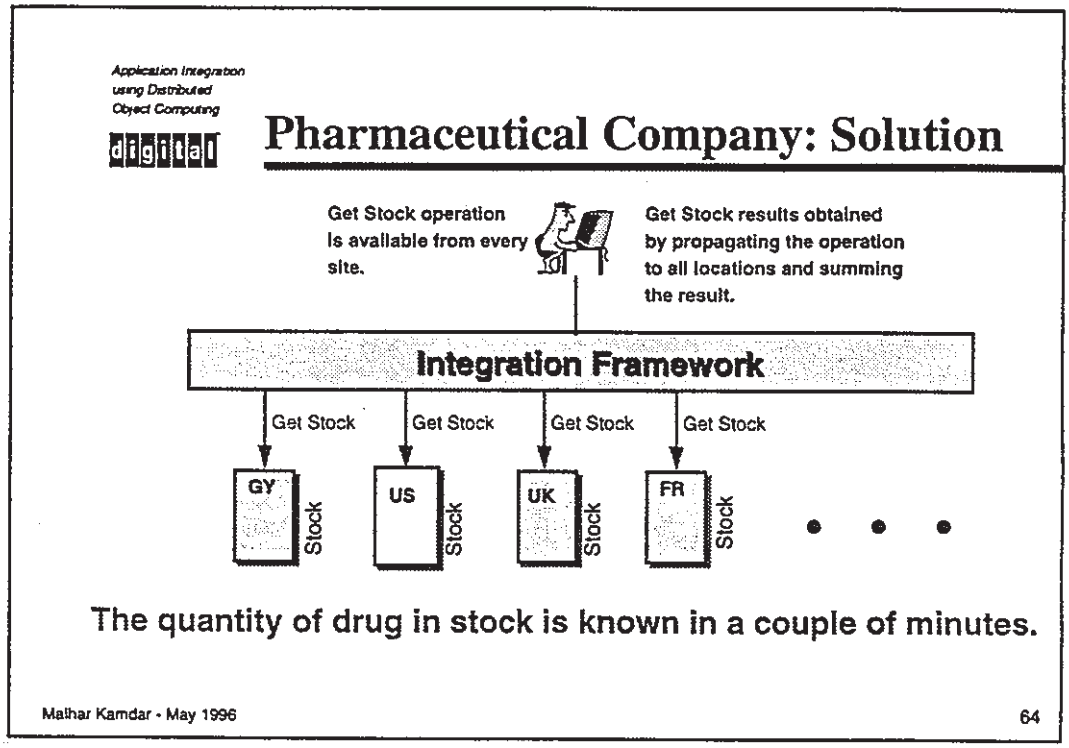
CORBA V2.0 完全準拠

COM/OLE 統合の完全実装

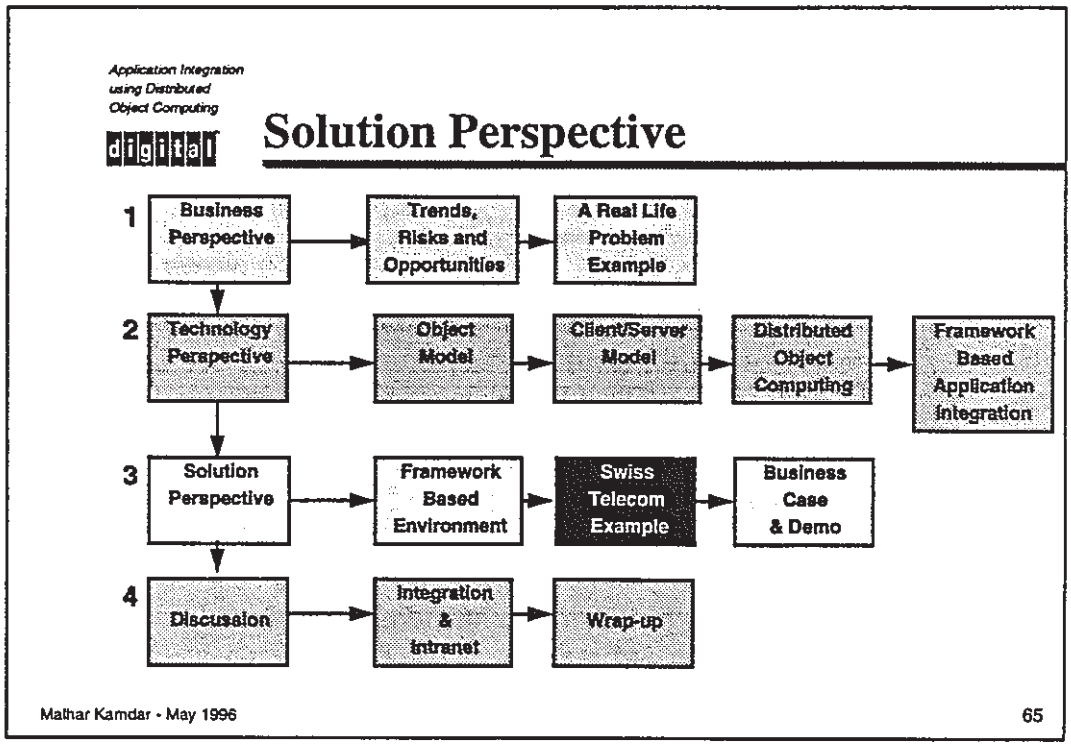
　　OLE2 non-GUIオブジェクト

サポート・プラットフォームの拡大

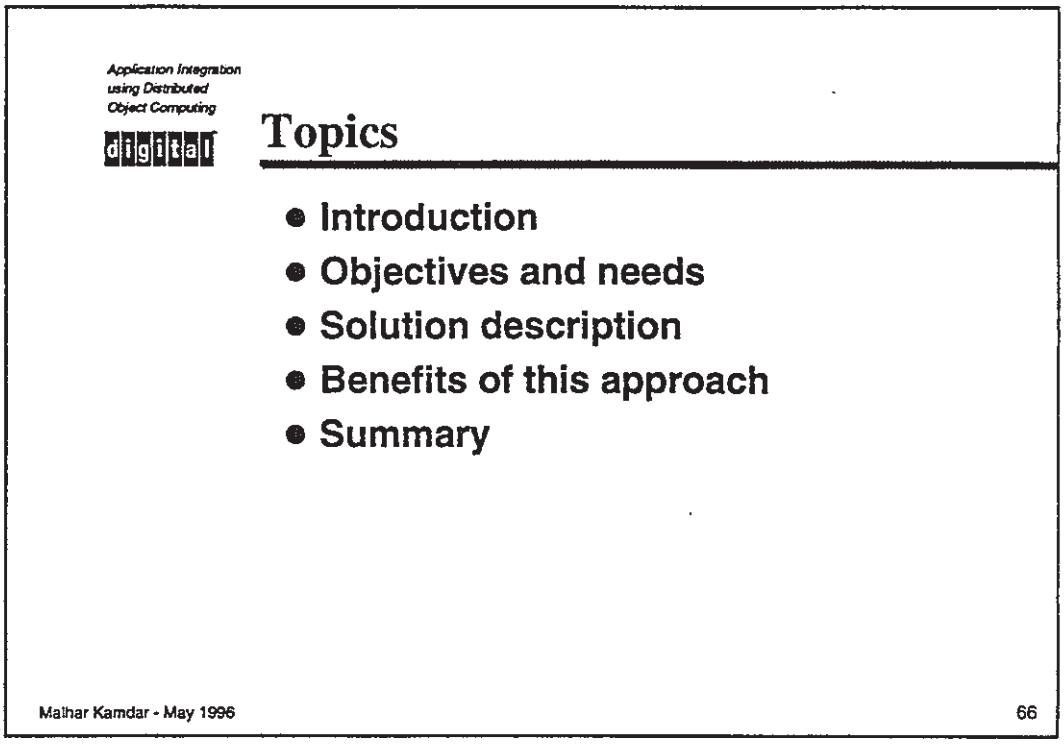
　　CDS, Concurrent Computer, NEC, Pyramid, SNI, Sony



製薬会社：ソルーション
すべてのサイトから在庫確認の操作ができる
在庫確認の操作はすべての支店に通知され結果が集計される
在庫の薬品の量は瞬時にわかる



ソルーションの展望：スイス・テレコムの事例



トピック
はじめに
目的とニーズ
ソルーションの説明
この方法の利点
まとめ

Application Integration using Distributed Object Computing

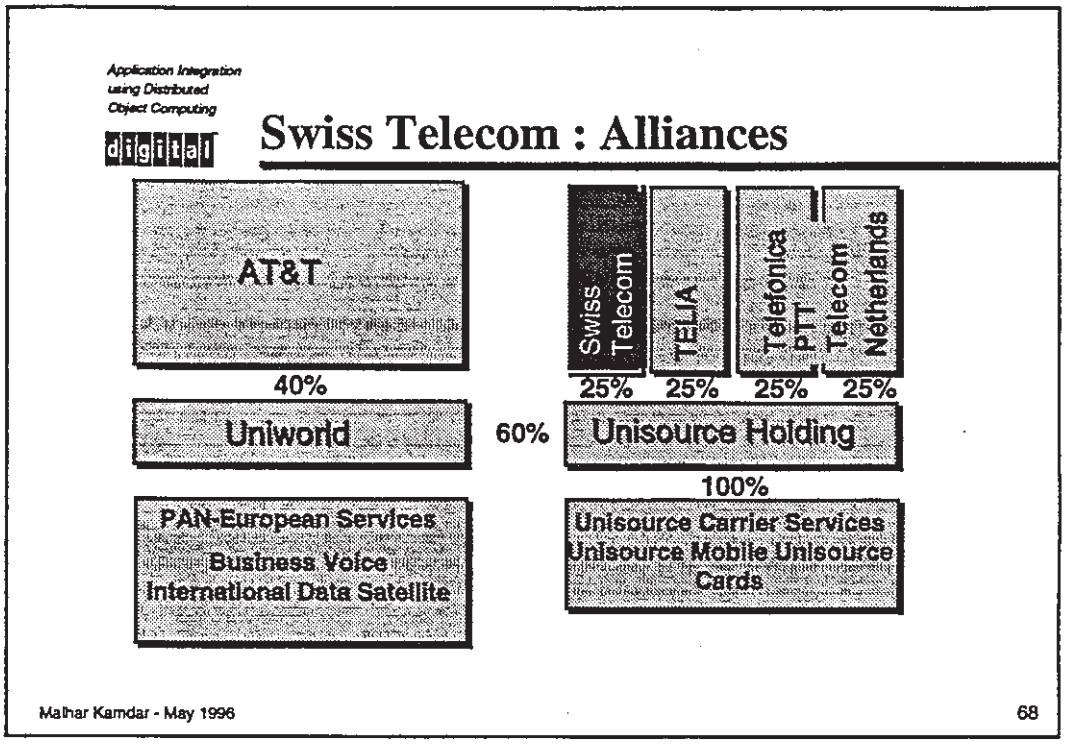
digital

Background

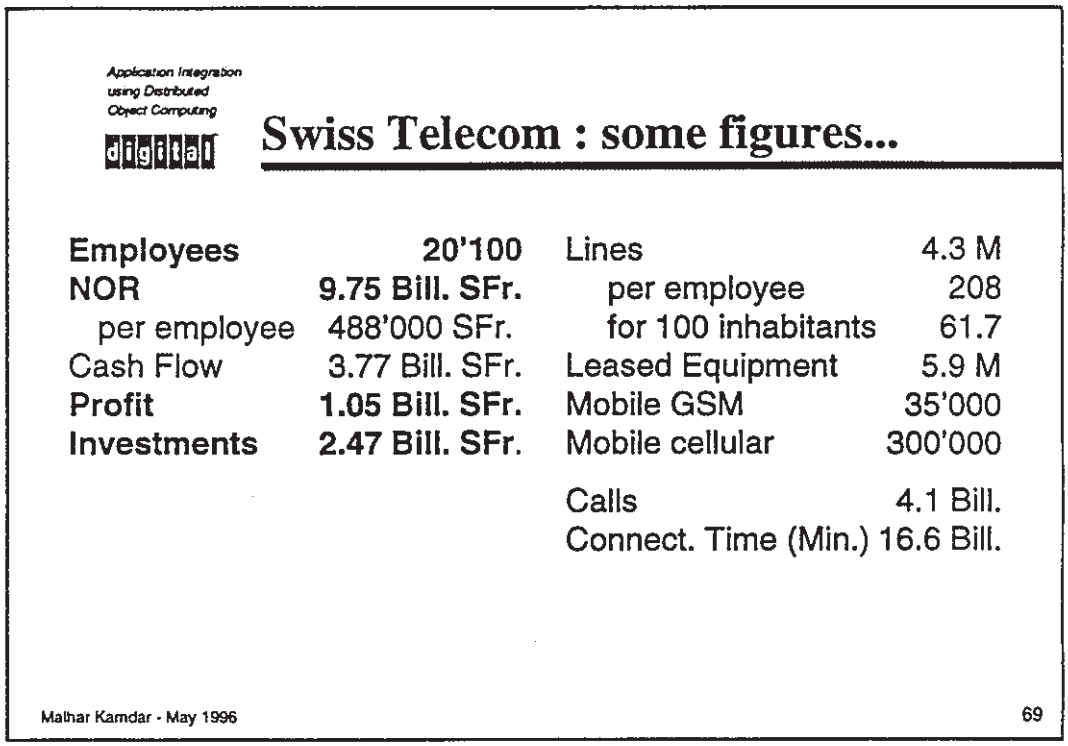
- Swiss PTT split
  - Two main businesses: Telecom and Postal
- Changing environment:
  - Liberalization of the Telecom market in Switzerland
  - Large business program: "Privatizing Telecom"
  - Competitive environment Vs. monopoly status

Malhar Kamdar - May 1996 67

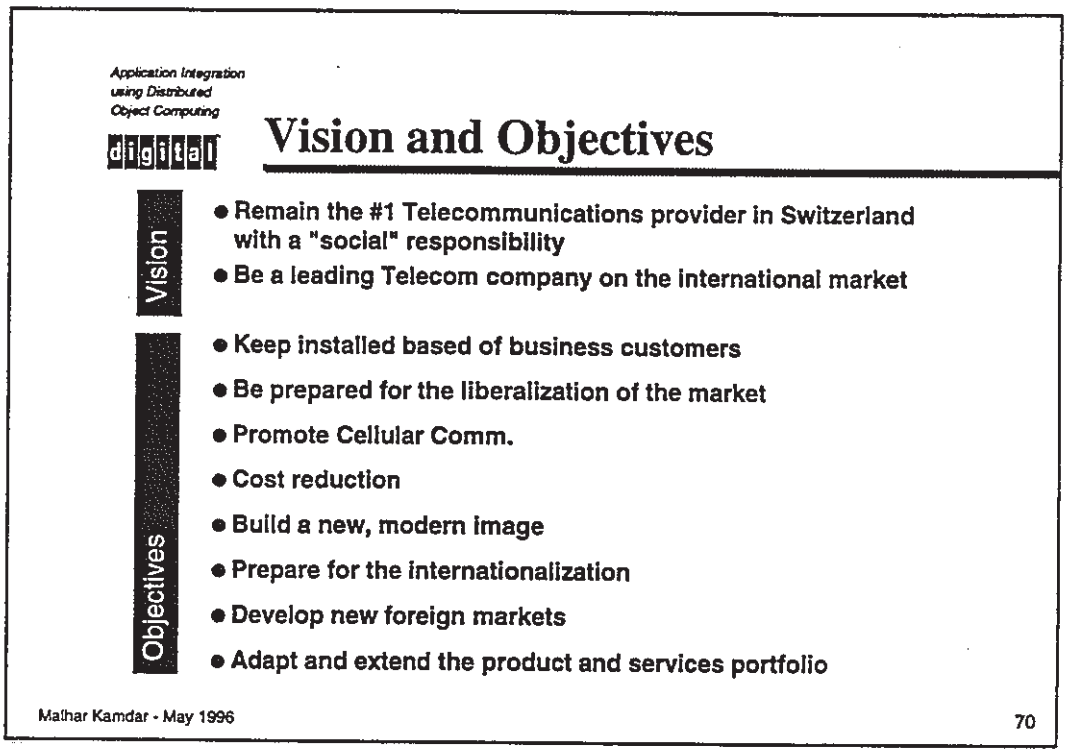
背景
スイスＰＴＴの分割
－ 二つの主な事業：テレコムと郵便
環境の変化
－ スイステレコム市場の自由化
－ 巨大なビジネス・プログラム
－ 競合環境 ｖｓ 独占状態



スイス・テレコム：相互関係



スイス・テレコム：会社概要



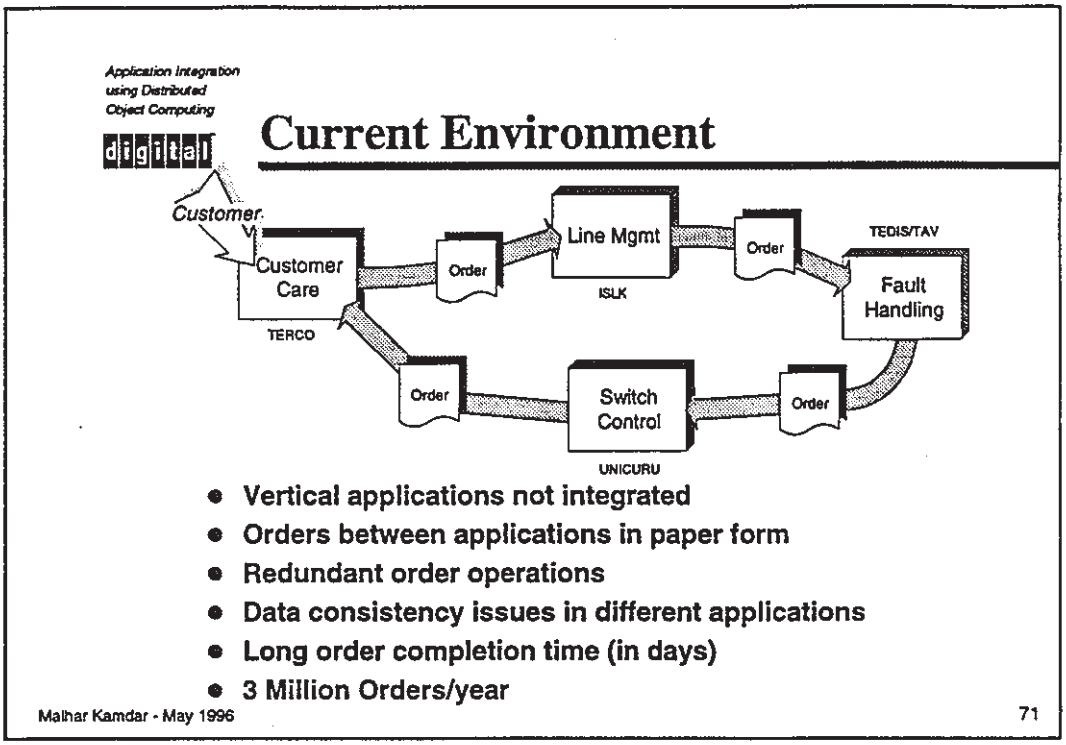
ビジョンと目的

ビジョン

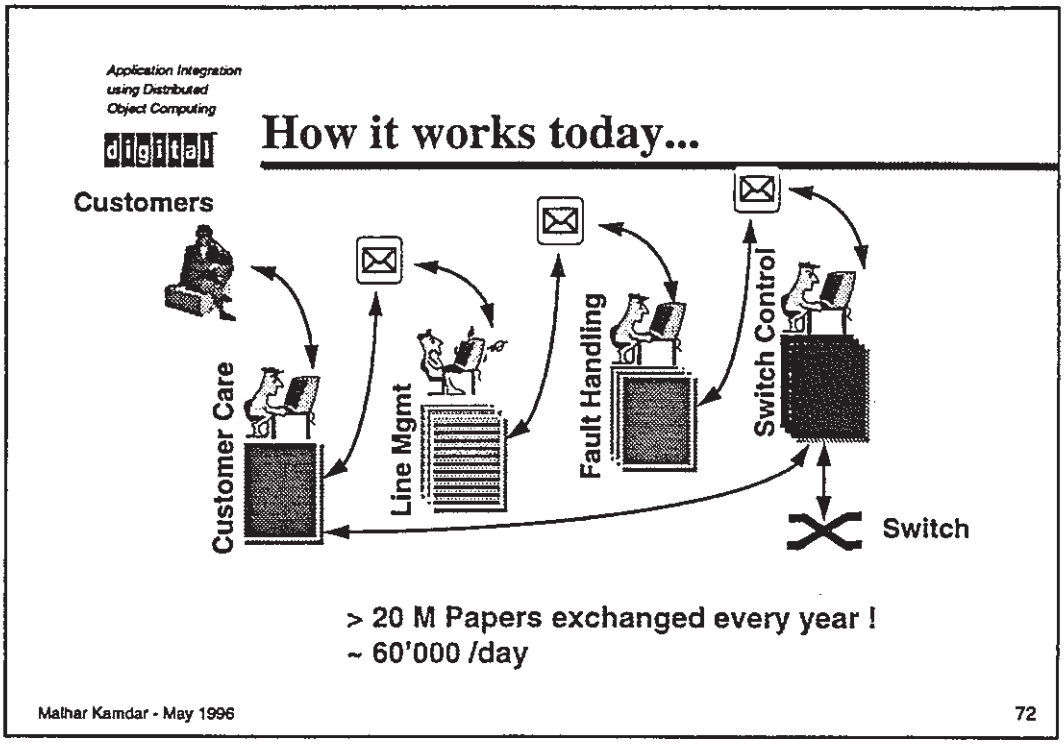
- 社会的責任をもってスイスでNo.1の通信プロバイダであり続ける
- 世界市場でテレコム会社のリーダとなる

目的

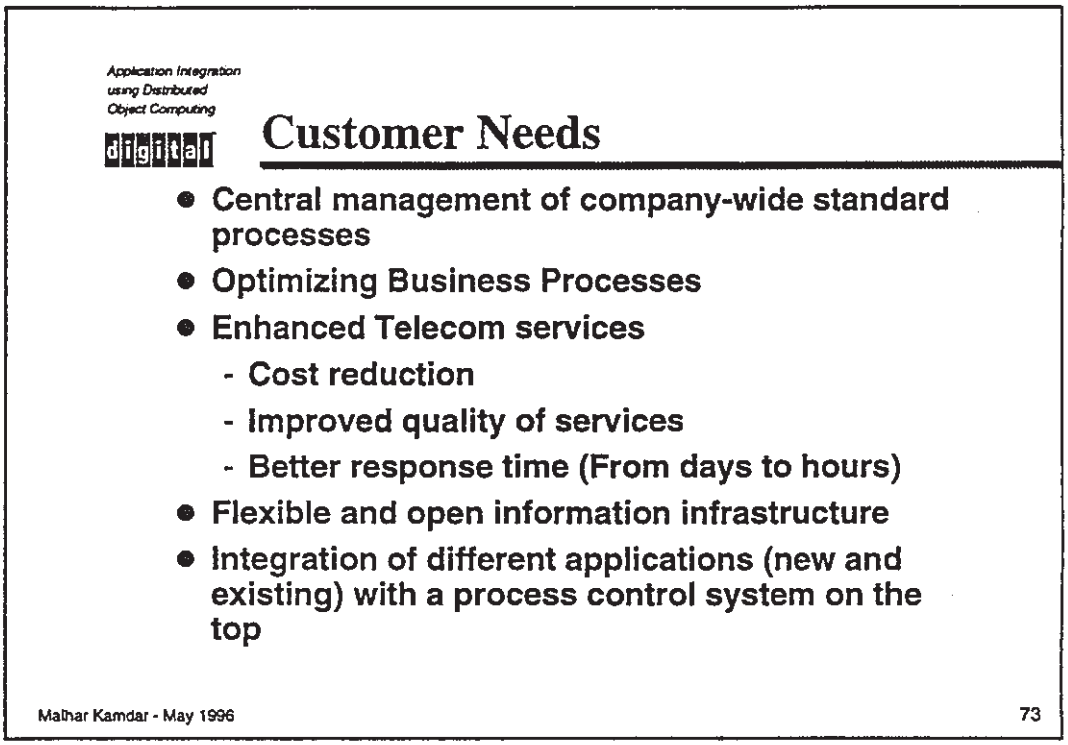
- ビジネス・カスタマのインストール・ベースを維持する
- 市場の自由化に備える
- 携帯電話の販売促進
- コストの引き下げ
- 新しくモダンなイメージを築く
- 国際化に備える
- 新しい国外の市場を開拓
- 製品とサービスの拡大と適応



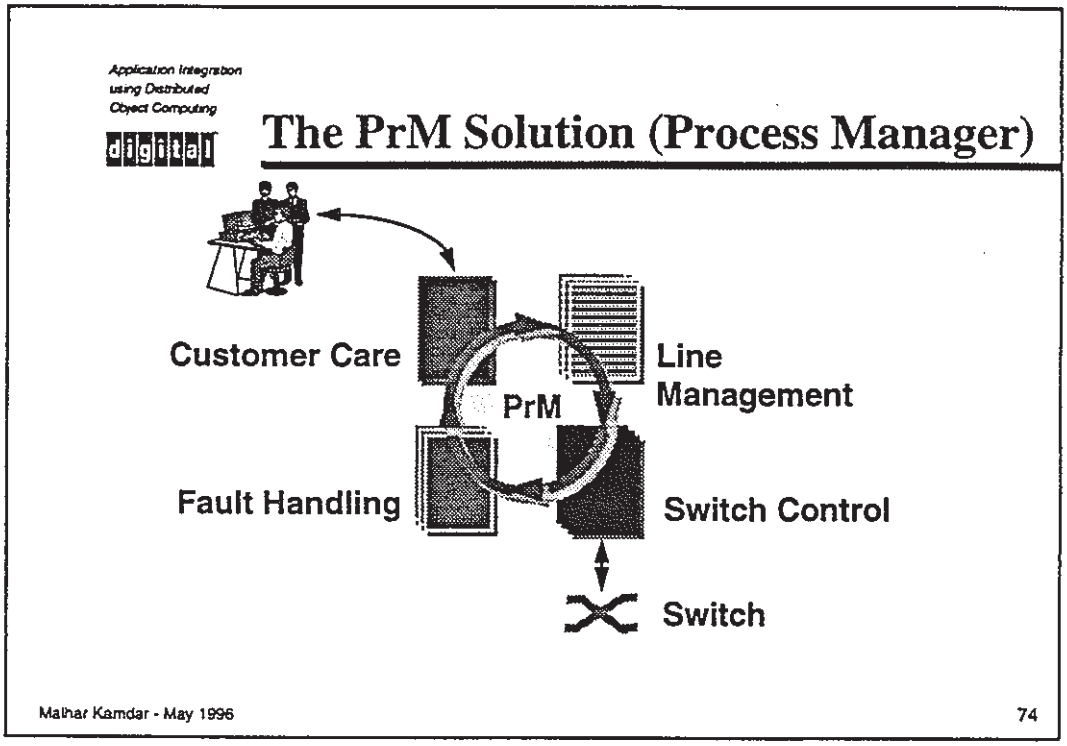
現在の状況
- 統合されていないアプリケーション
- アプリケーション間は紙ベースで依頼
- オーダー操作が冗長
- 異なるアプリケーションでのデータの一貫性が問題
- オーダー操作が完了するまで数日かかる
- 年間３００万件のオーダー



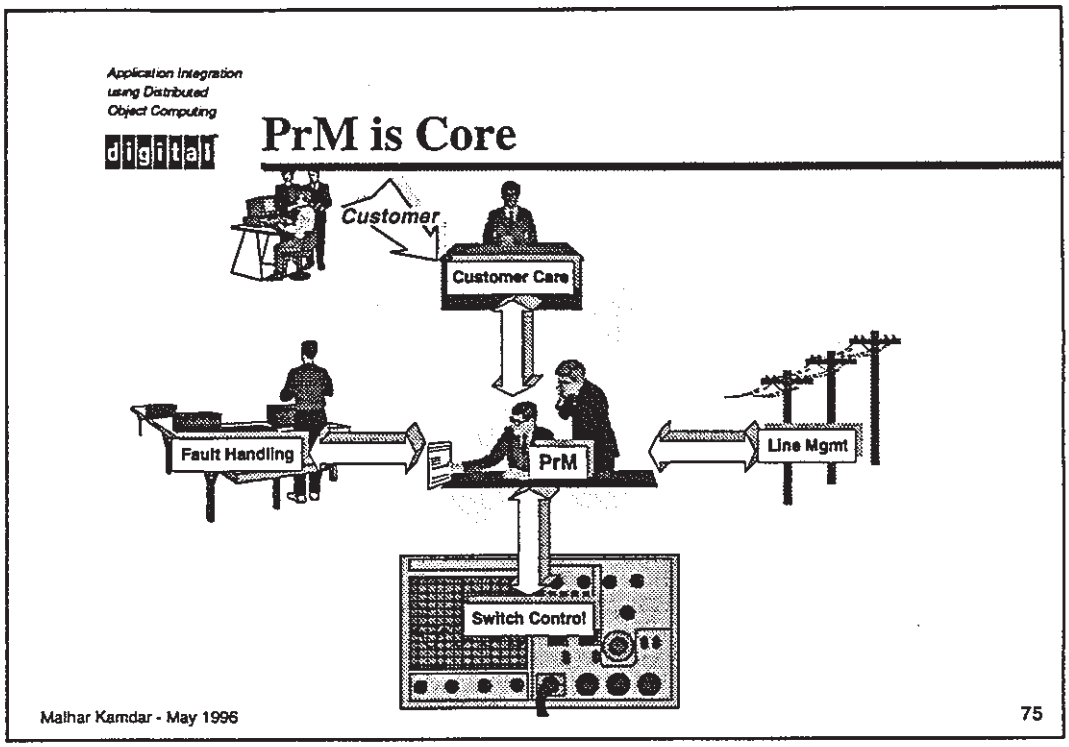
現在ははどうなっているか
- 年間２０００万枚以上の紙が交換される
- ということは１日で６万枚



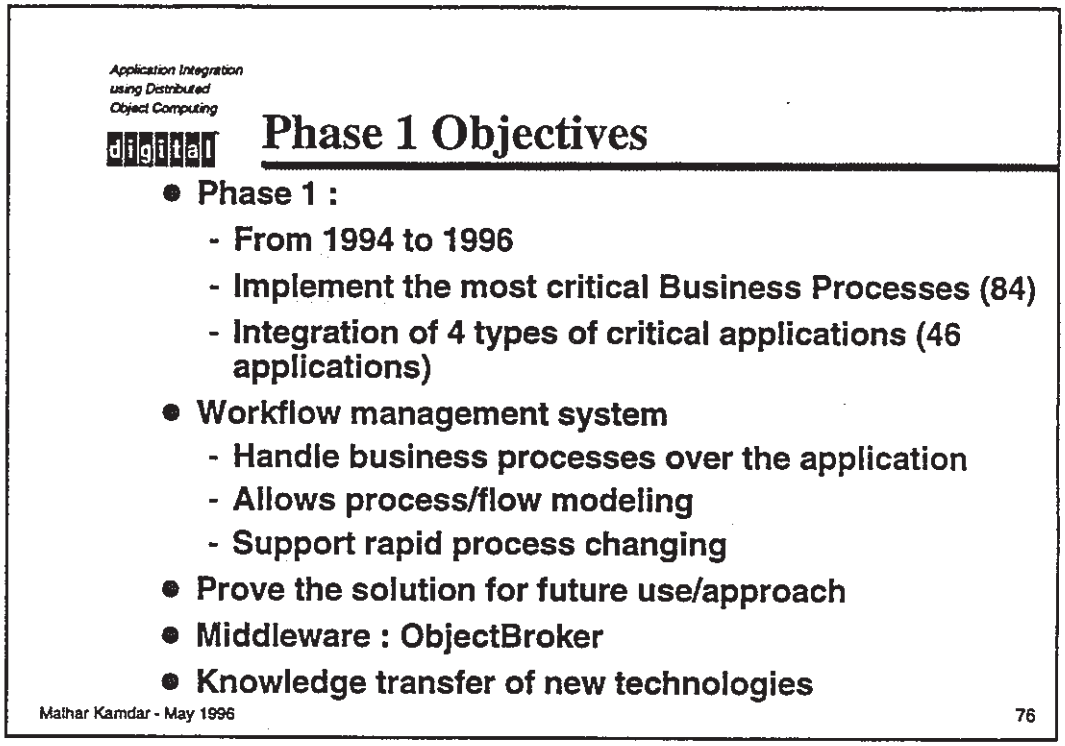
カスタマのニーズ
全社の標準的作業プロセスを集中管理
ビジネス・プロセスの最適化
テレコム・サービスの拡充
- コストの引き下げ
- サービスの品質向上
- 応答時間の改善
柔軟でオープンな情報基盤
アプリケーション統合



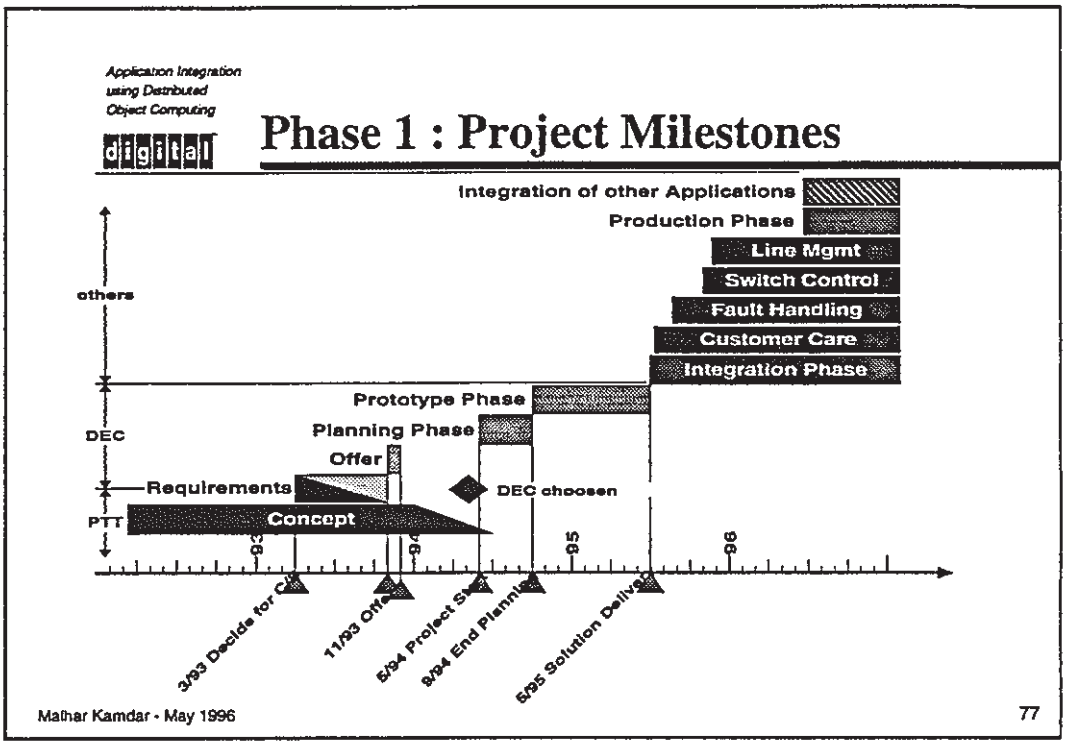
ＰｒＭ（プロセス・マネージャ）ソルーション


Application Integration
using Distributed
Object Computing
digital
PrM is Core
Customer
Customer Care
Fault Handling
PrM
Line Mgmt
Switch Control
Malhar Kamdar - May 1996
75

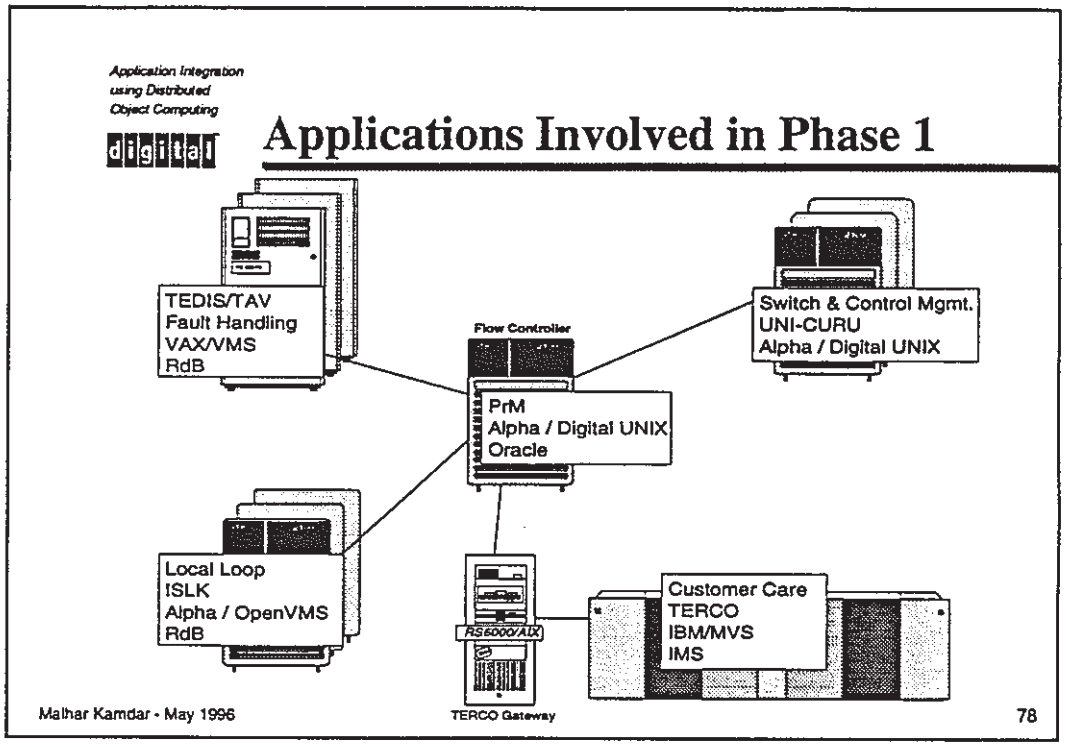
ＰｒＭが中心



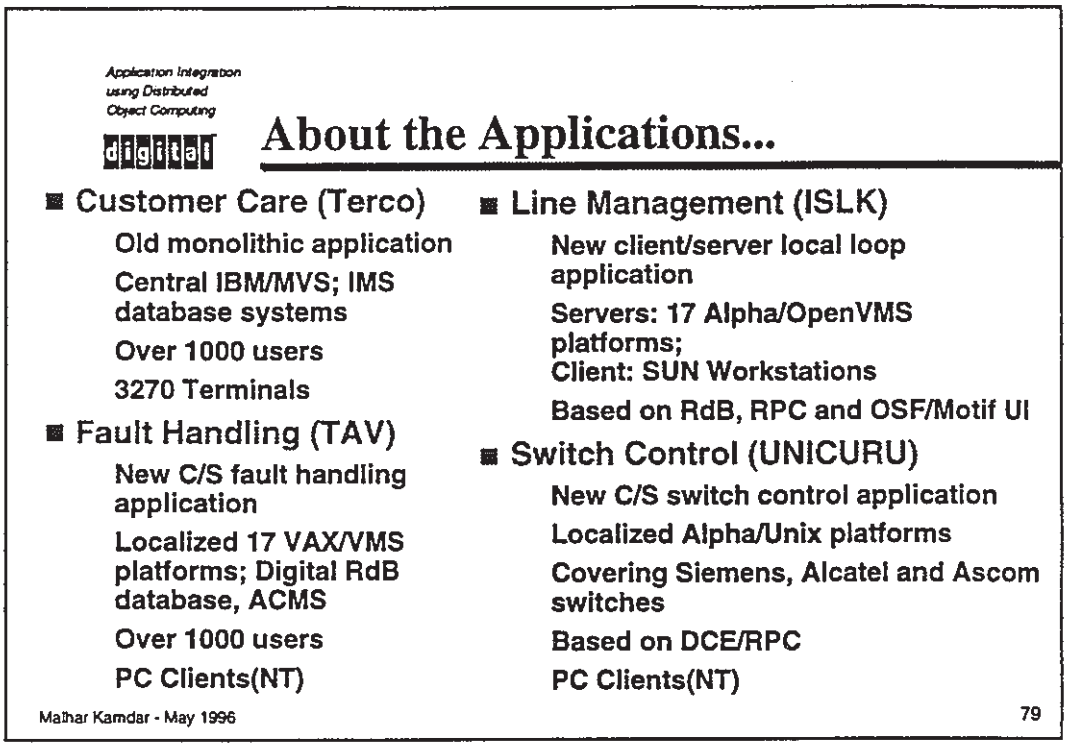
フェーズ１の目的
フェーズ１：
－ １９９４年から１９９６年
－ 最も重要なビジネス・プロセスの実装
－ 重要なアプリケーションの４つのタイプの統合
ワークフロー・マネージメント・システム
－ アプリケーションをまたぐビジネス・プロセスを扱う
－ プロセス／フローのモデリングを可能にする
－ 迅速なプロセスの変更を可能にする
将来の利用／アプローチのためのソルーションを示す
ミドルウェア：ObjectBroker
新しいテクノロジに関する知識の移管



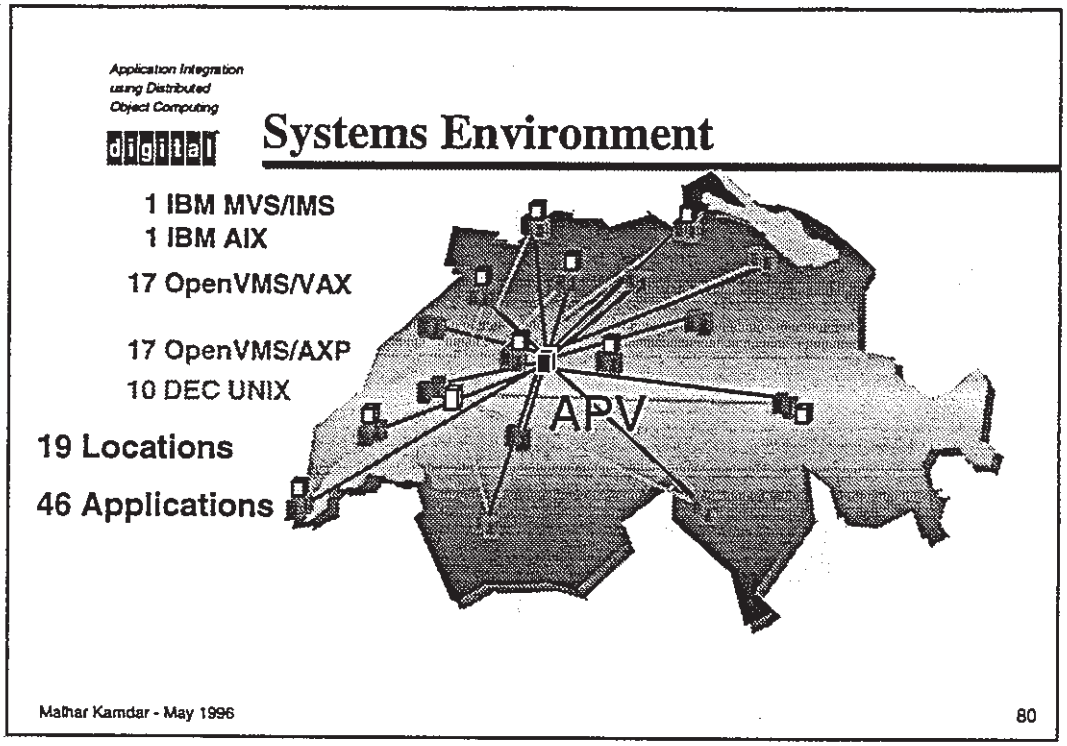
フェーズ１：プロジェクトの計画



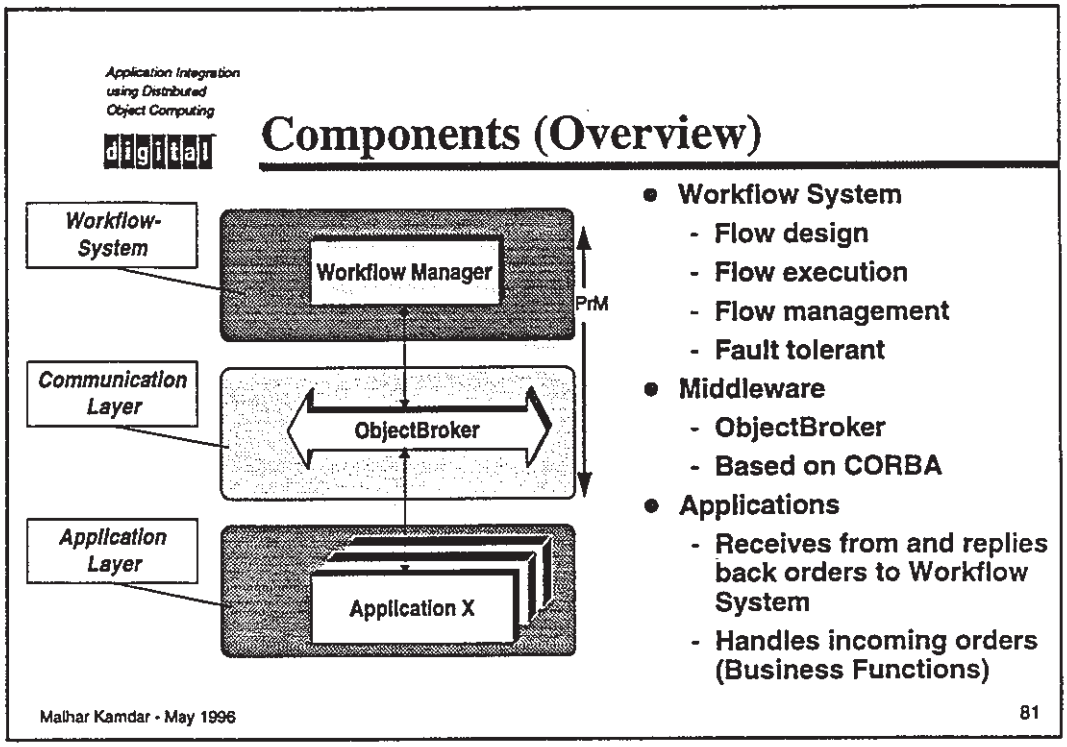
フェーズ1に関係するアプリケーション



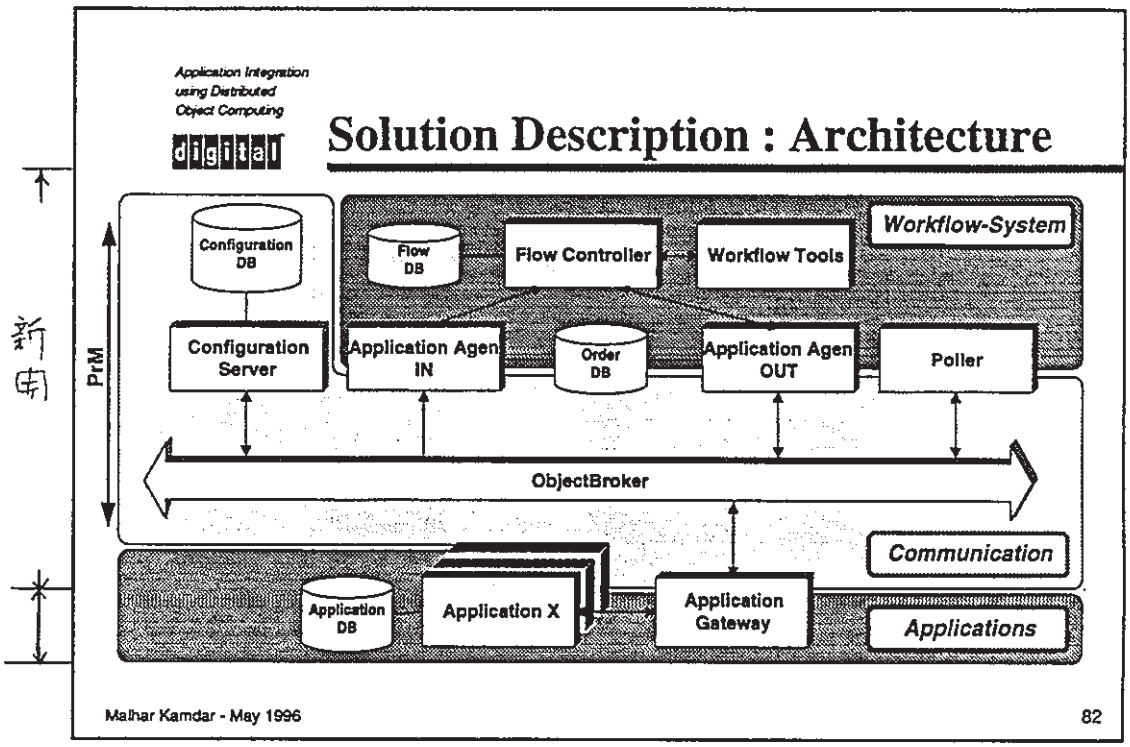
アプリケーションに関して


Application Integration
using Distributed
Object Computing
digital
Systems Environment
1 IBM MVS/IMS
1 IBM AIX
17 OpenVMS/VAX
17 OpenVMS/AXP
10 DEC UNIX
19 Locations
46 Applications
APV
Mathar Kamdar - May 1996
80

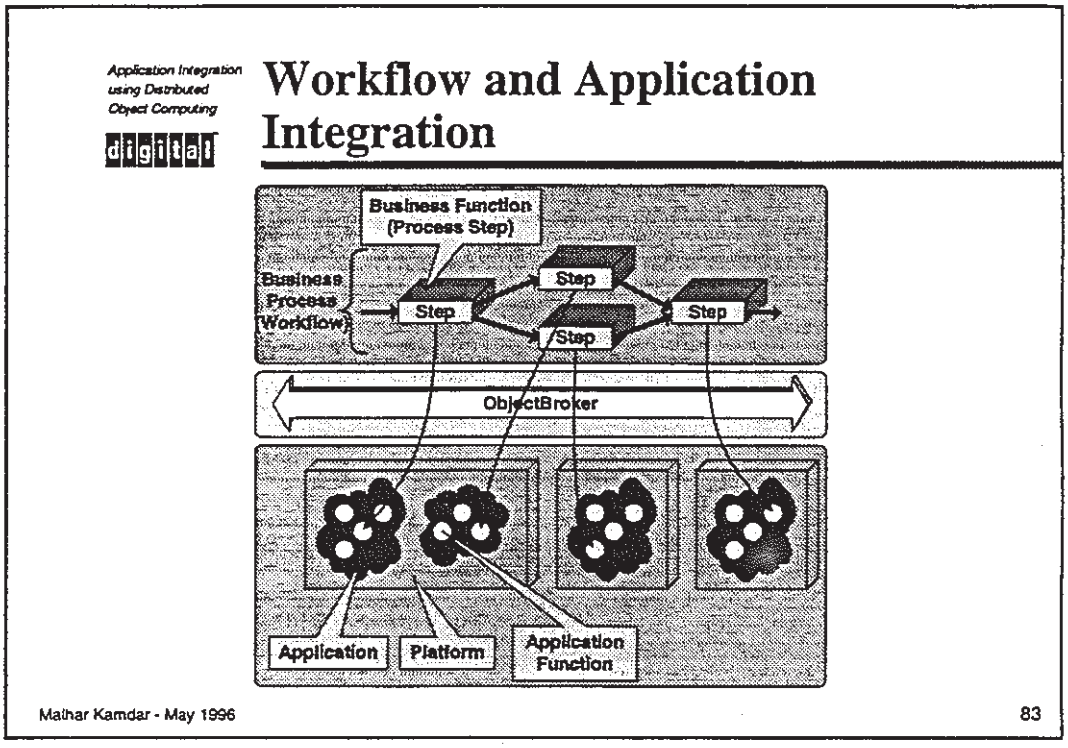
システムの環境


Application Integration using Distributed Object Computing
digital
Components (Overview)
Workflow-System
Workflow Manager
PrM
Communication Layer
ObjectBroker
Application Layer
Application X
• Workflow System
- Flow design
- Flow execution
- Flow management
- Fault tolerant
• Middleware
- ObjectBroker
- Based on CORBA
• Applications
- Receives from and replies back orders to Workflow System
- Handles incoming orders (Business Functions)
Malhar Kamdar - May 1996
81

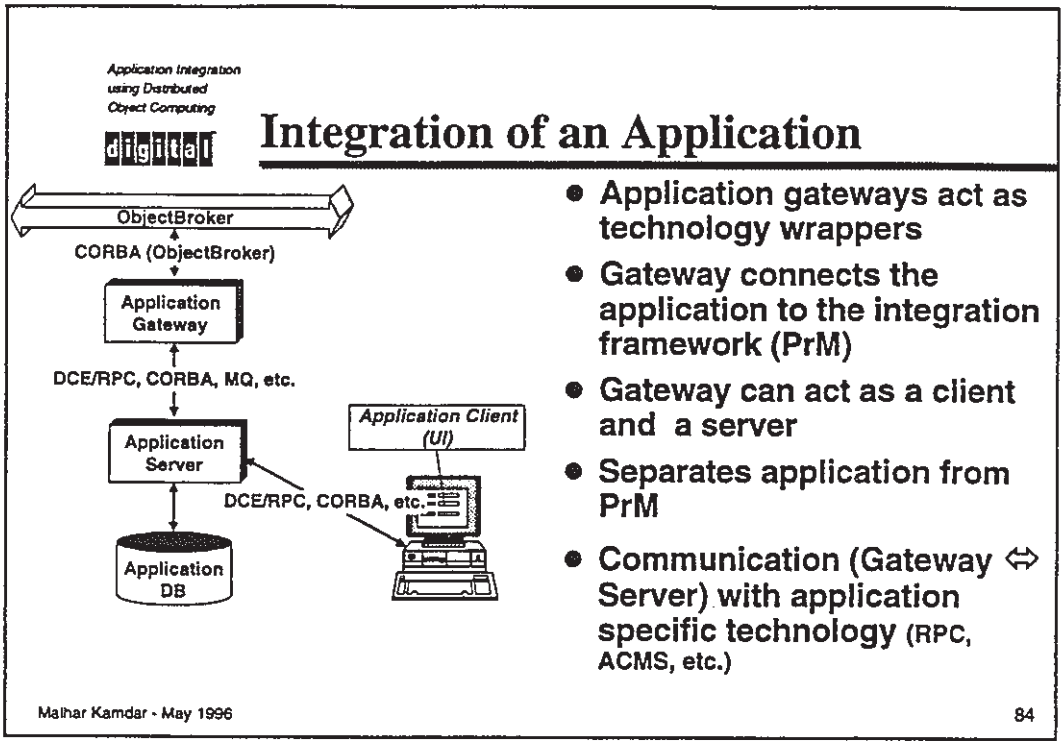
コンポーネントの概要



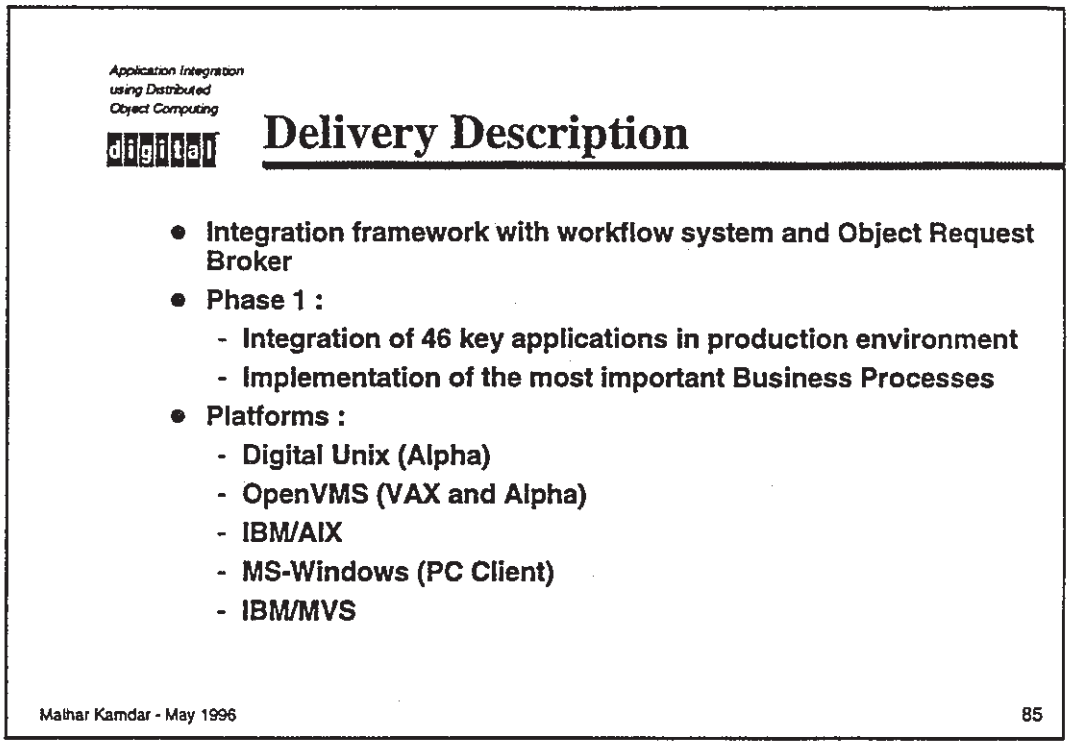
ソルーションの説明：アーキテクチャ



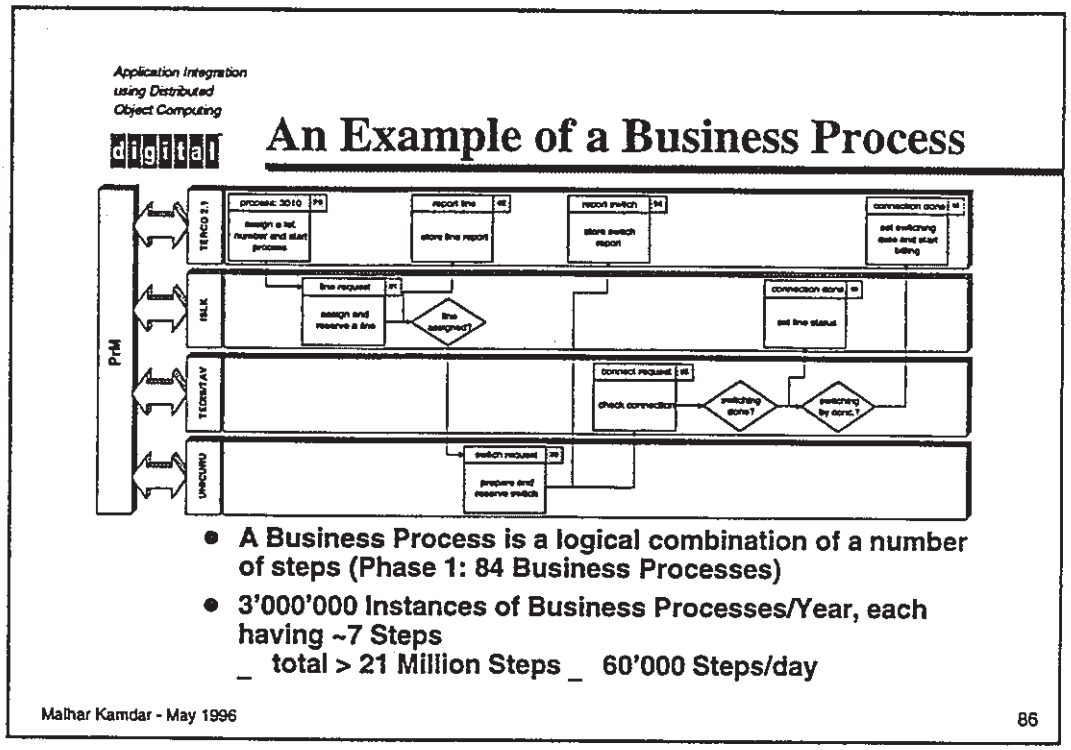
ワークフローとアプリケーションの統合



アプリケーションの統合
アプリケーション・ゲートウェイはテクノロジ・ラッパーとして振る舞う
ゲートウェイはアプリケーションを統合フレームワークに接続する
ゲートウェイはクライアントとサーバになれる
ＰｒＭからアプリケーションを分離する
ゲートウェイとサーバの通信はアプリケーションに特化する



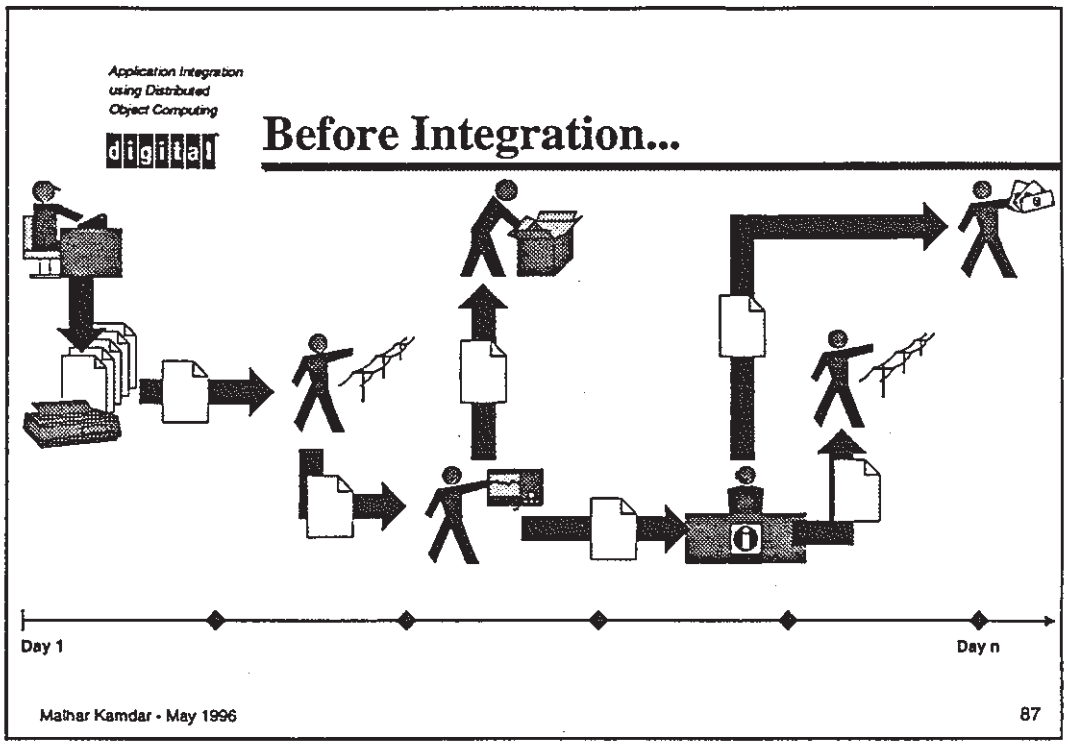
納入仕様の説明
統合フレームワークにワークフロー・システムとＯＲＢ
フェーズ１
－ 製造環境の４６の重要なアプリケーションの統合
－ 最も重要なビジネス・プロセスの実装
プラットフォーム



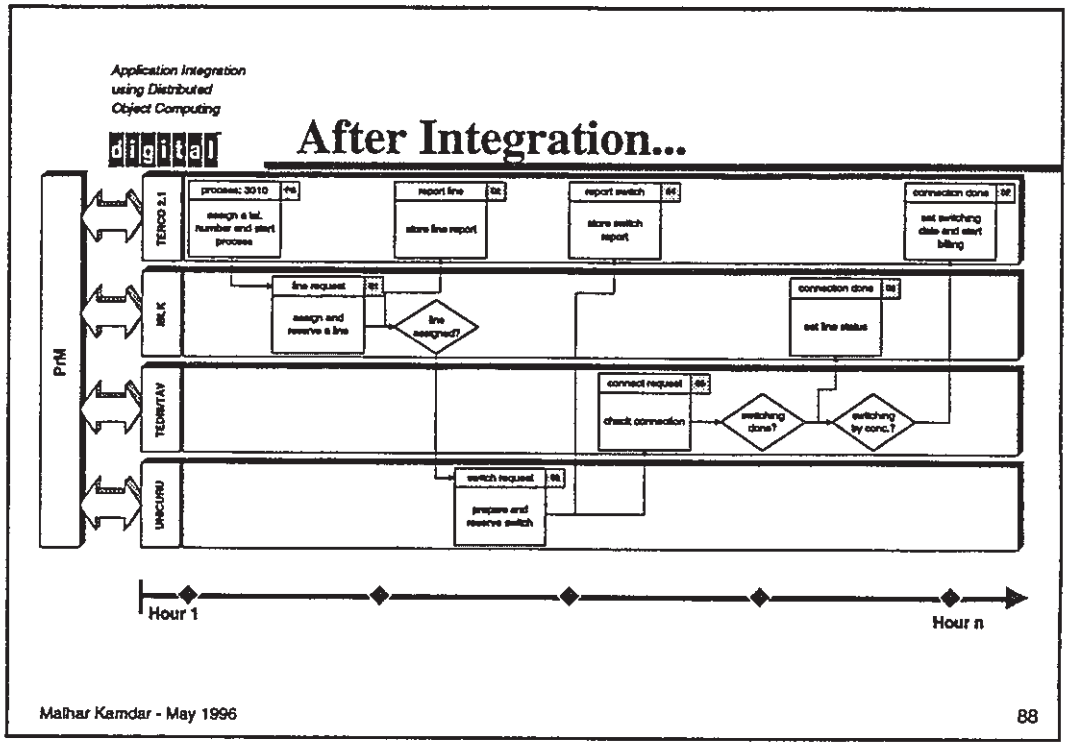
ビジネス・プロセスの例

ビジネス・プロセスは多くのステップの論理的な結合（フェーズ１では８４ビジネス・プロセス）

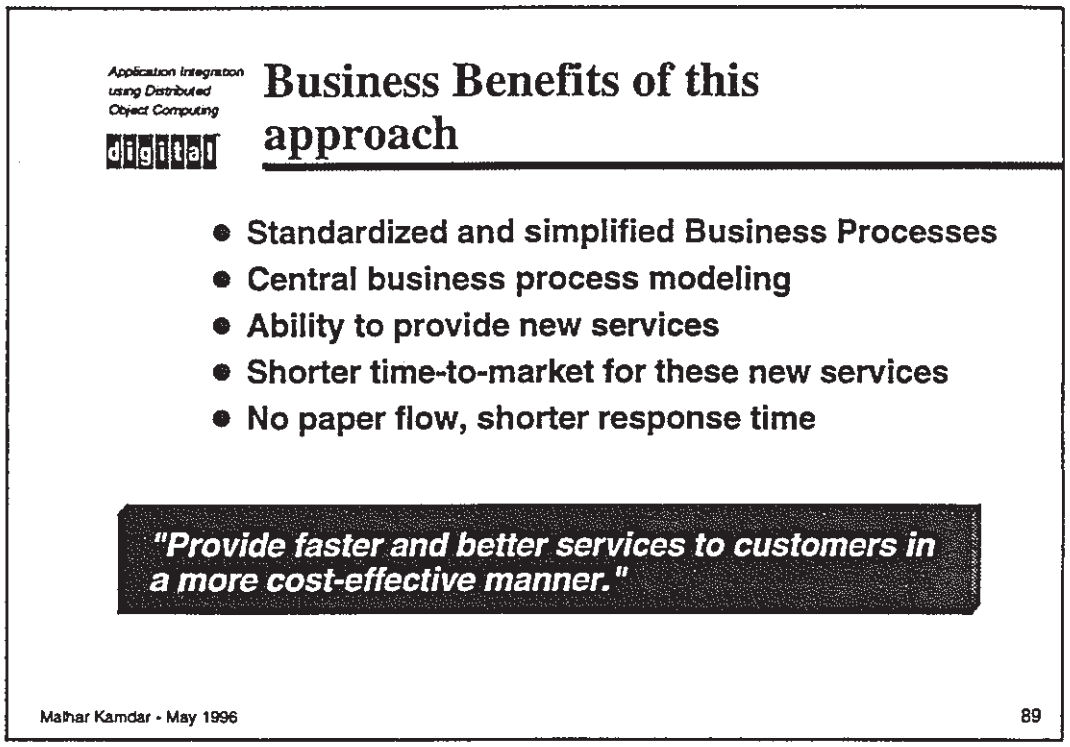
年間３００万のビジネス・プロセスのインスタンス、インスタンスそれぞれが７ステップ程、トータルで年間２１００万以上のステップ、一日で６万ステップ


Application Integration
using Distributed
Object Computing
digital
Before Integration...
Day 1
Day n
Mathar Kamdar - May 1996
87

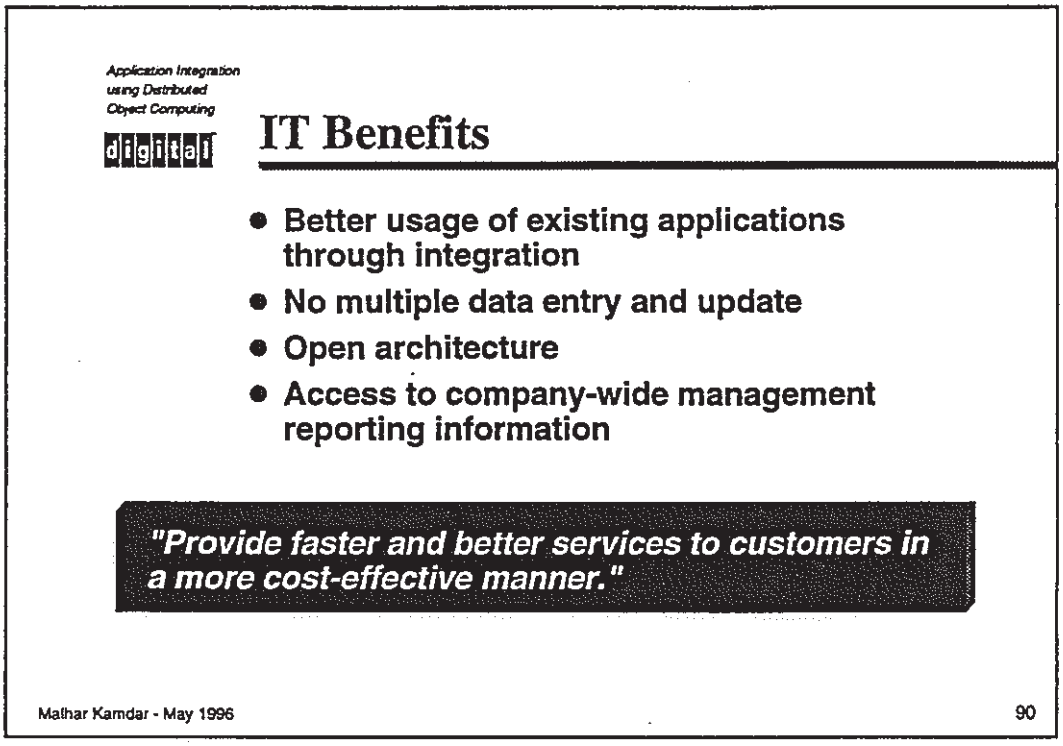
統合前の状態


Application Integration
using Distributed
Object Computing
digital
After Integration...
PrM
TERO 2.1
IBLK
TEDIRITAV
UNICURU
process: 3010
assign a tel. number and start process
report line
store line report
report switch
store switch report
connection done
set switching data and start billing
line request
assign and reserve a line
line assigned?
connection done
set line status
connect request
check connection
switching done?
switching by conc.?
switch request
prepare and reserve switch
Hour 1
Hour n
Malhar Kamdar - May 1996
88

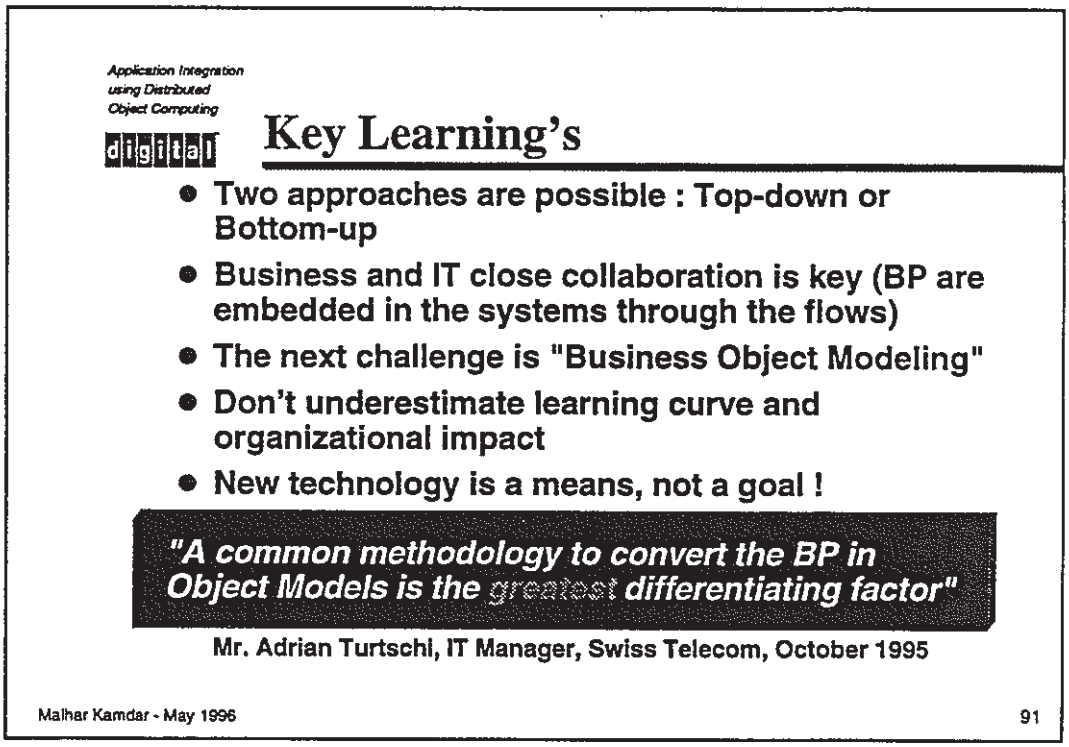
統合後の状態



このアプローチがビジネスに与えるメリット
ビジネス・プロセスの標準化と単純化
中心的ビジネス・プロセスのモデリング
新しいサービスを提供する機能
新しいサービスの迅速な市場への投入
紙ベースの作業依頼の廃止、応答時間の短縮
より早く、よりよいサービスをより効果的にカスタマに提供



ＩＴに与えるメリット
統合することで既存のアプリケーションの活用
データ登録・更新の重複の防止
オープンなアーキテクチャ
全社的管理情報へのアクセス
より早くよりよいサービスをより効果的にカスタマに提供



主要な学ぶべき点

二つのアプローチが可能：トップダウンかボトムアップ

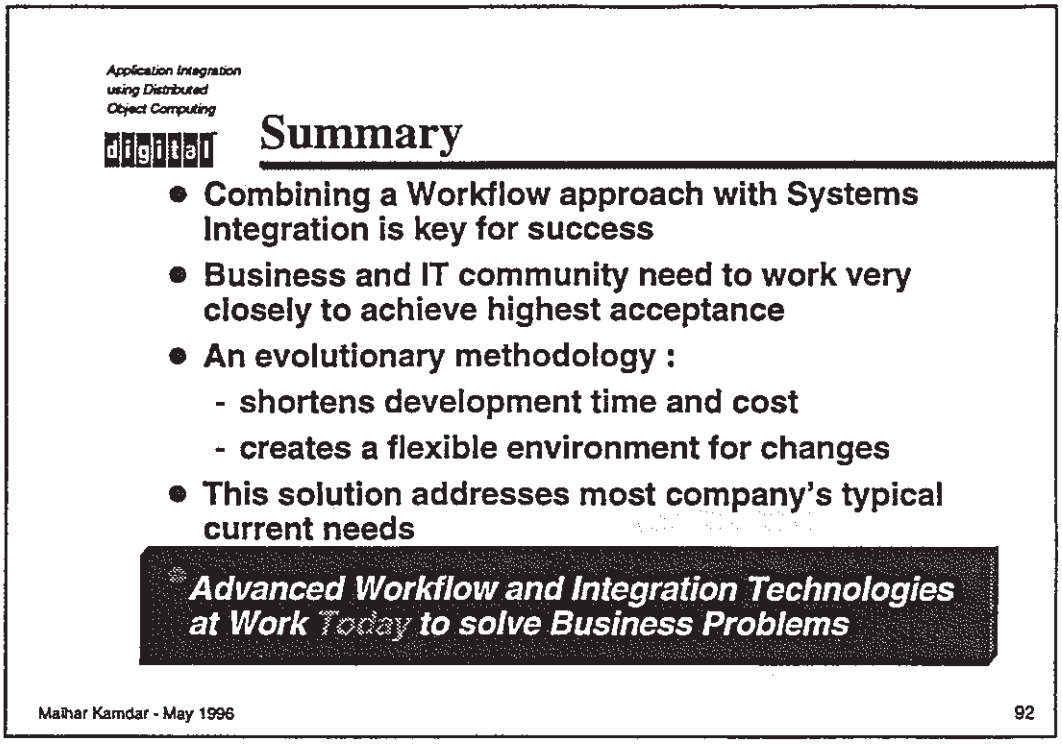
ビジネスとＩＴの密接な協調が鍵（ビジネス・プロセスはフローを通じてシステムに埋めこまれる）

次の挑戦は”ビジネス・オブジェクト・モデリング”

学習曲線と組織に与えるインパクトを過少評価しない

新しいテクノロジは手段であってゴールではない

ビジネス・プロセスをオブジェクト・モデルに変換する方法論は最大の差別化の要因



まとめ

ワークフロー・アプローチとシステム統合の結合が成功への鍵

最高度の可用性を実現するために、ビジネスとＩＴは密接して働く必要がある

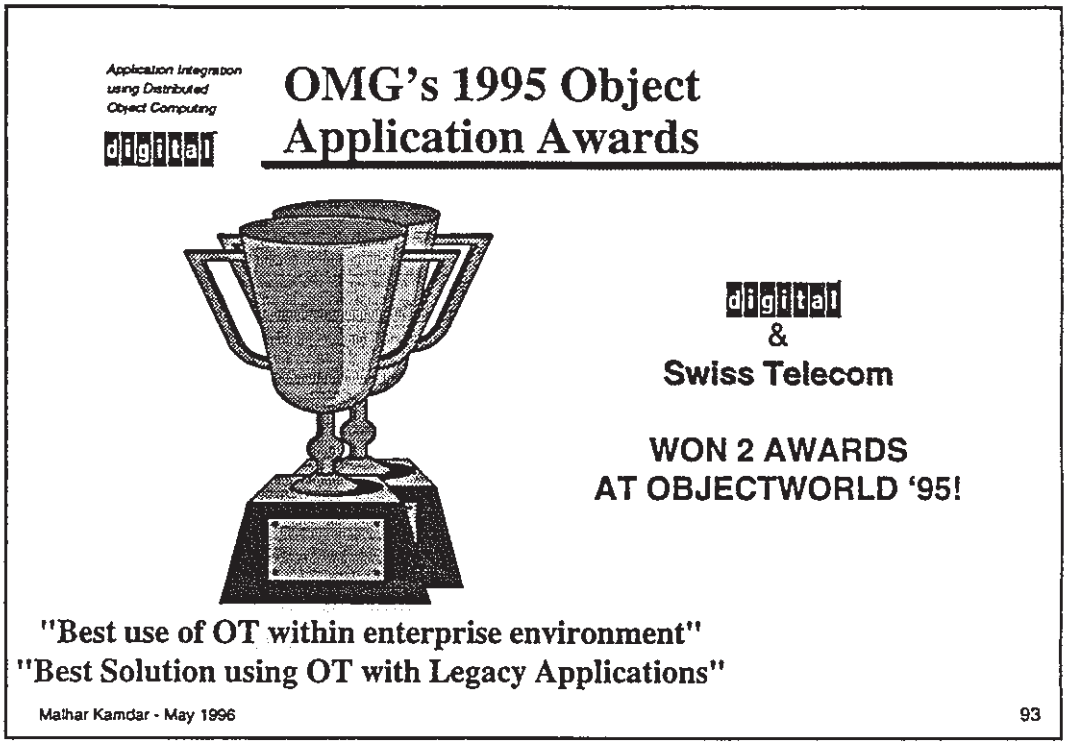
漸進的アプローチ：

－ 開発時間の短縮とコストの削減

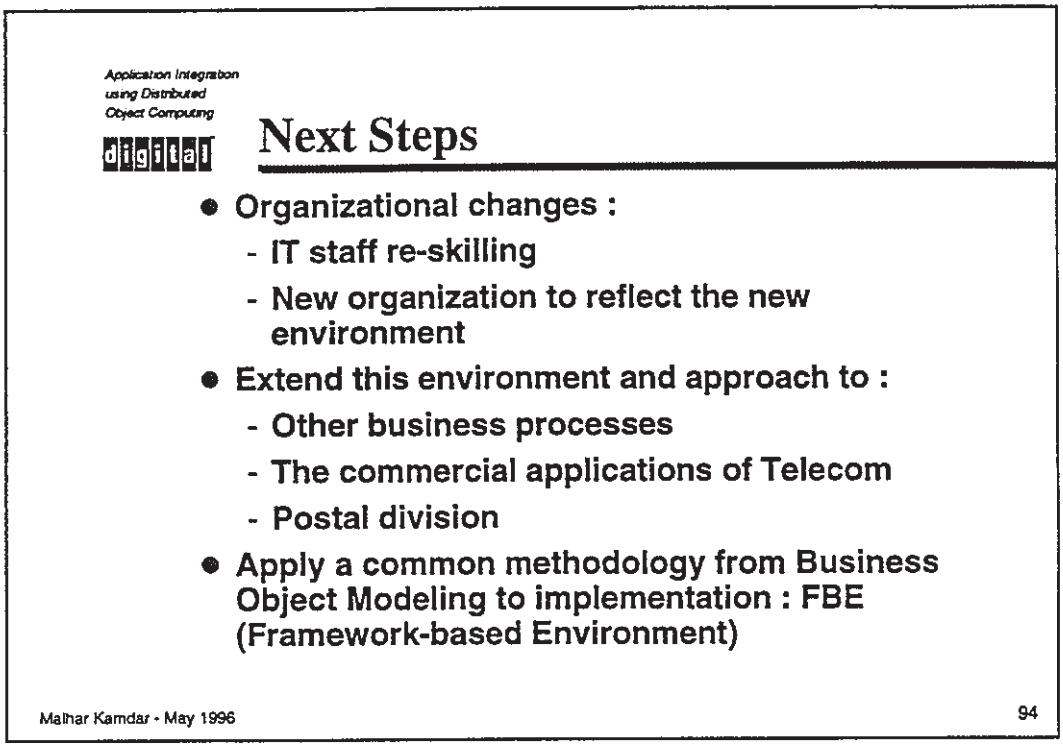
－ 変更に備えて柔軟な環境を構築

本ソルーションは今日のほとんどの会社の典型的なニーズに適応する

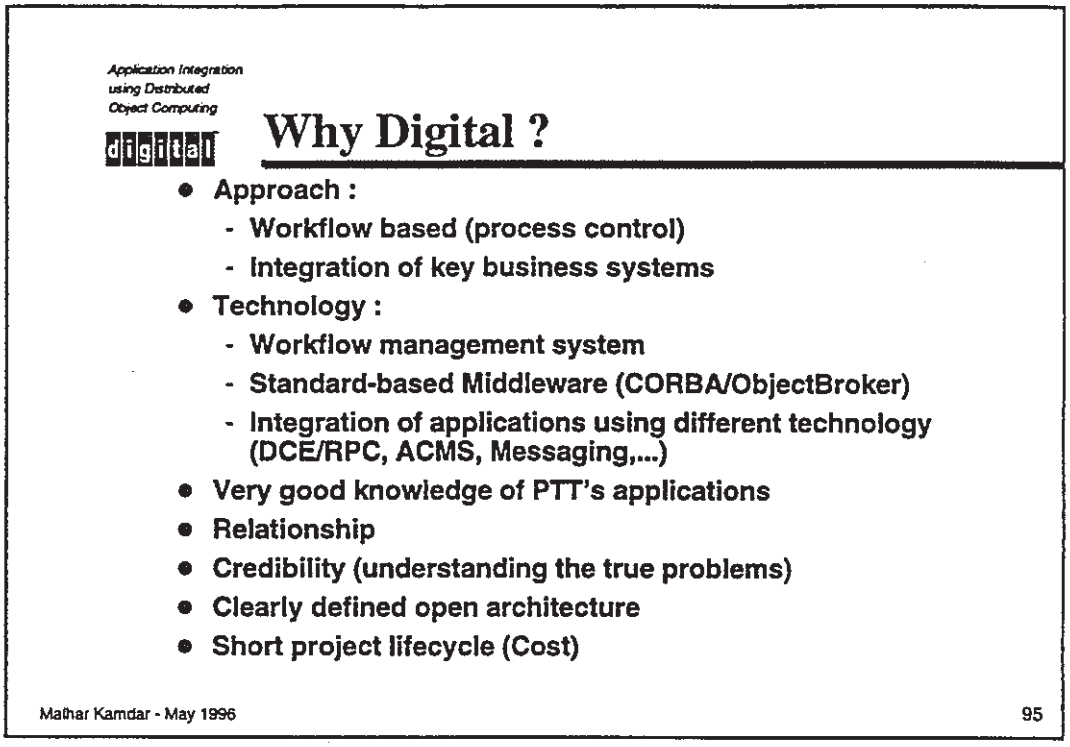
先進的なワークフローと統合のテクノロジは、ビジネス上の諸問題を解決する



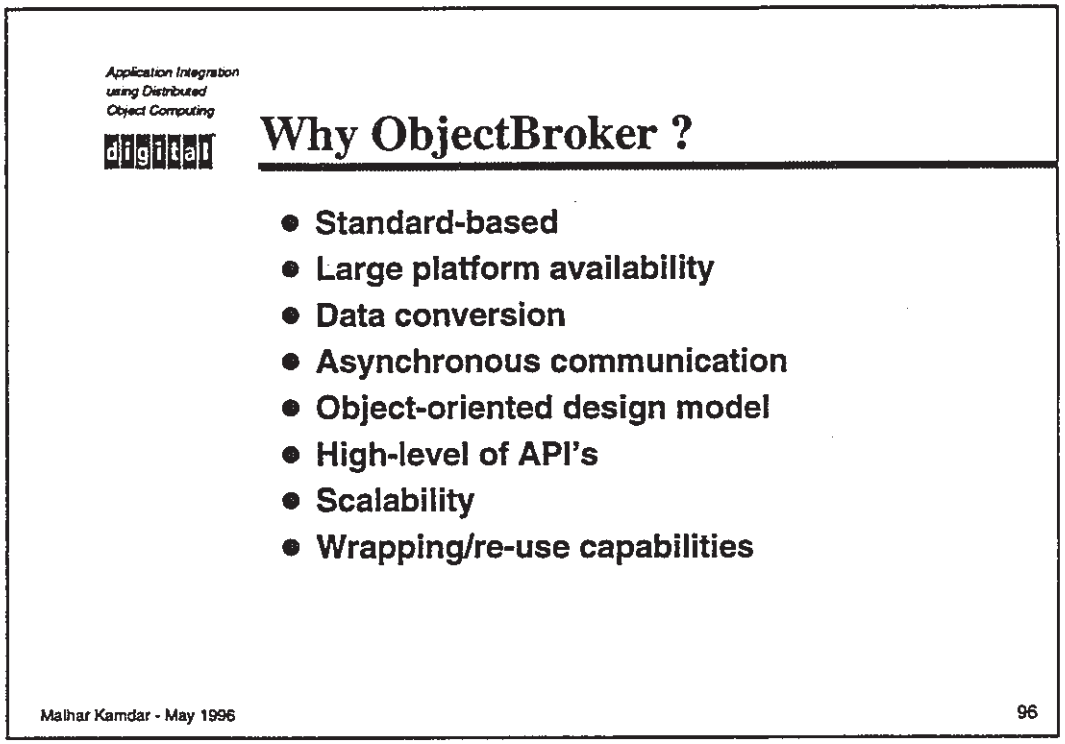
ＯＭＧの１９９５年オブジェクト・アプリケーション・アワード授賞



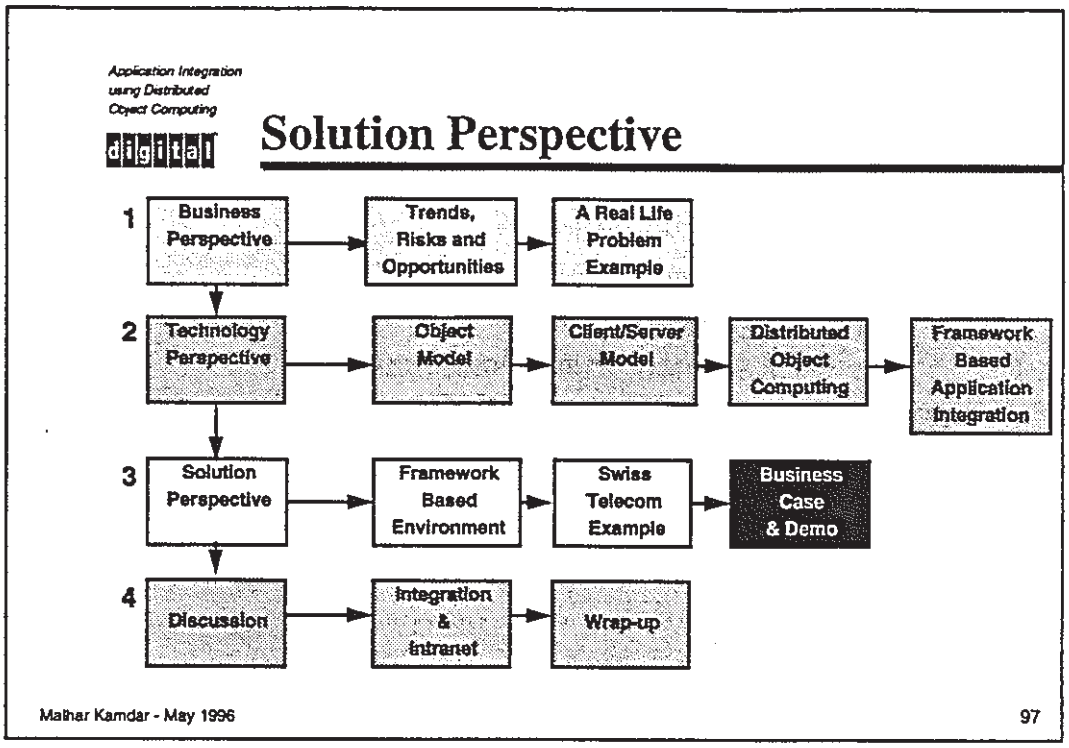
次のステップ
組織的な変革 _
－ ＩＴスタッフの再教育
－ 新しい環境を反映する新しい組織
環境とアプローチの拡大
－ その他のビジネス・プロセス
－ テレコムの商用アプリケーション
－ 郵便部門
ビジネス・オブジェクト・モデリングから実装へ共通の方法論の適応：ＦＢＥ



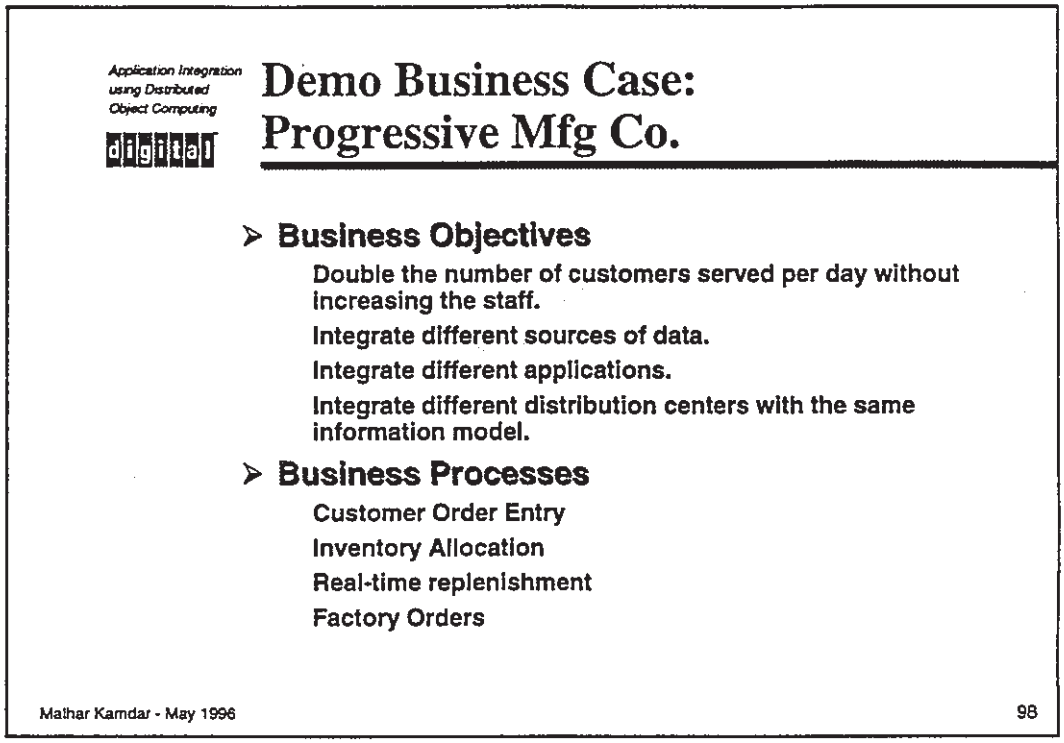
なぜＤＥＣか？
アプローチ：
－ ワークフロー・ベース
－ 主要なビジネス・システムの統合
テクノロジ
－ ワークフロー・マネージメント・システム
－ 標準準拠のミドルウェア(CORBA/ObjectBroker)
－ 異なるテクノロジを使用したアプリケーション統合(DCE/RPC,ACMS...)
ＰＴＴアプリケーションに関する造詣
関係
信頼性
明確に定義されたオープン・アーキテクチャ
短いプロジェクトのライフサイクル



なぜＯｂｊｅｃｔＢｒｏｋｅｒか？
標準準拠
多彩なマルチ・プラットフォーム
データ変換
非同期通信
オブジェクト指向のデザイン・モデル
高レベルのＡＰＩ
拡張性
ラッピング／再利用の機能



ソルーションの展望 － ビジネス事例とデモ



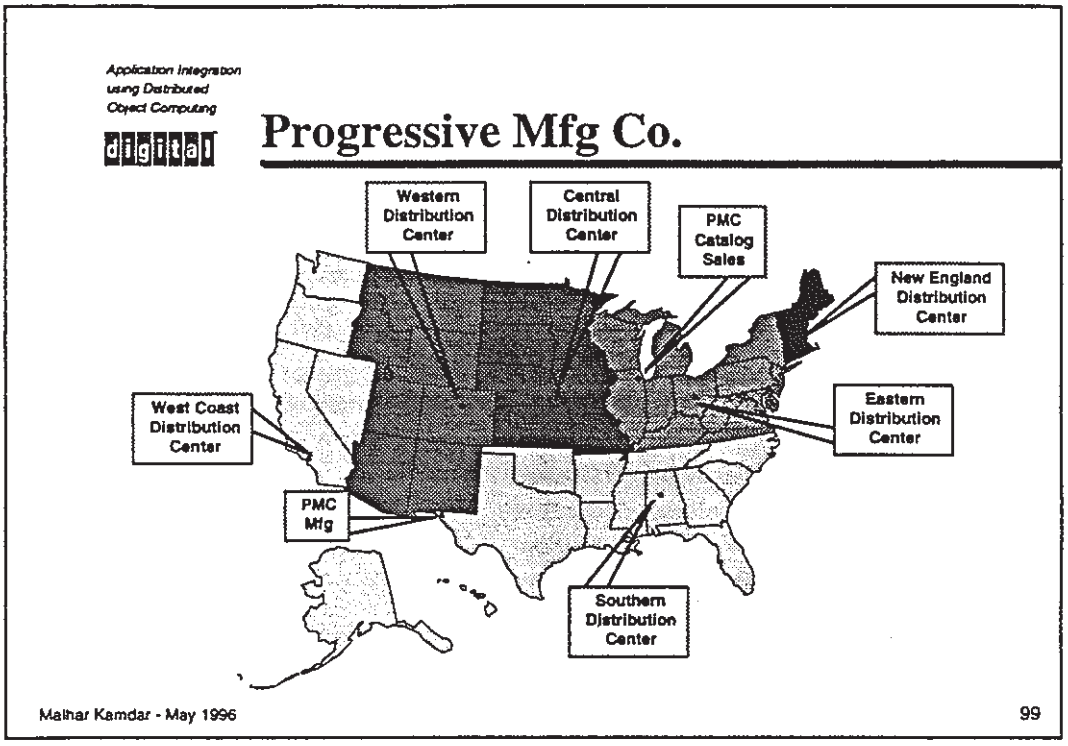
デモ事例：プログレッシヴ製造会社

ビジネスの目的

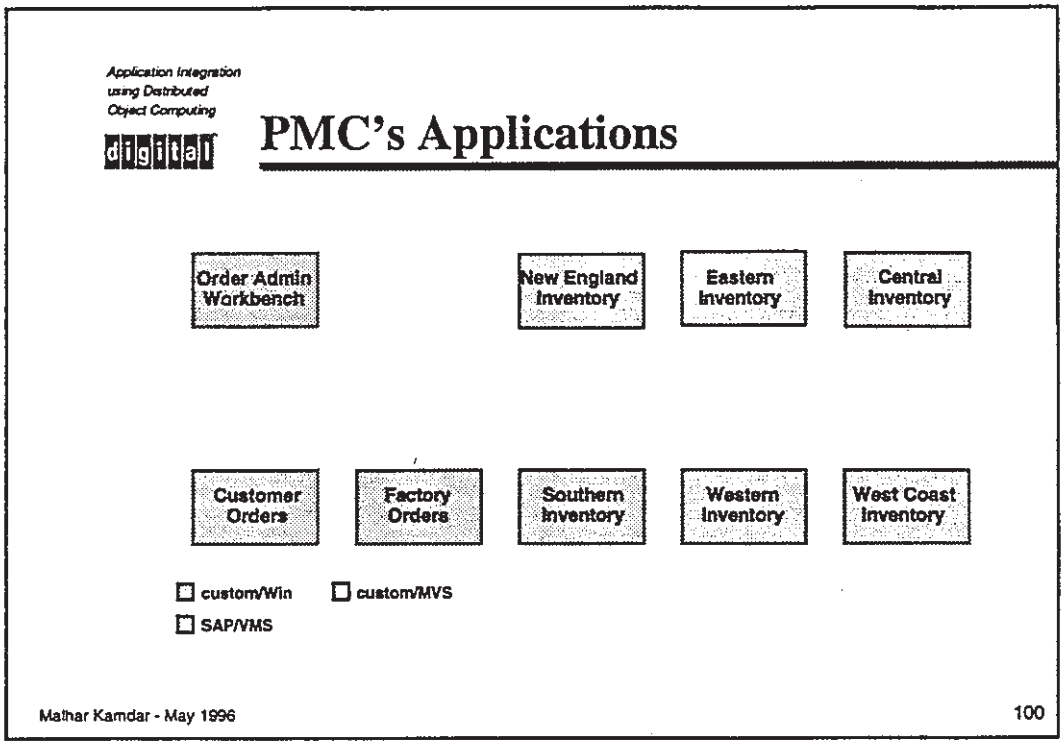
- スタッフの人数を増やすことなく一日のカスタマの対応を倍増させる
- 異なるソースのデータを統合する
- 異なるアプリケーションを統合する
- 同一のインフォメーション・モデルで異なる配送センターを統合する

ビジネス・プロセス

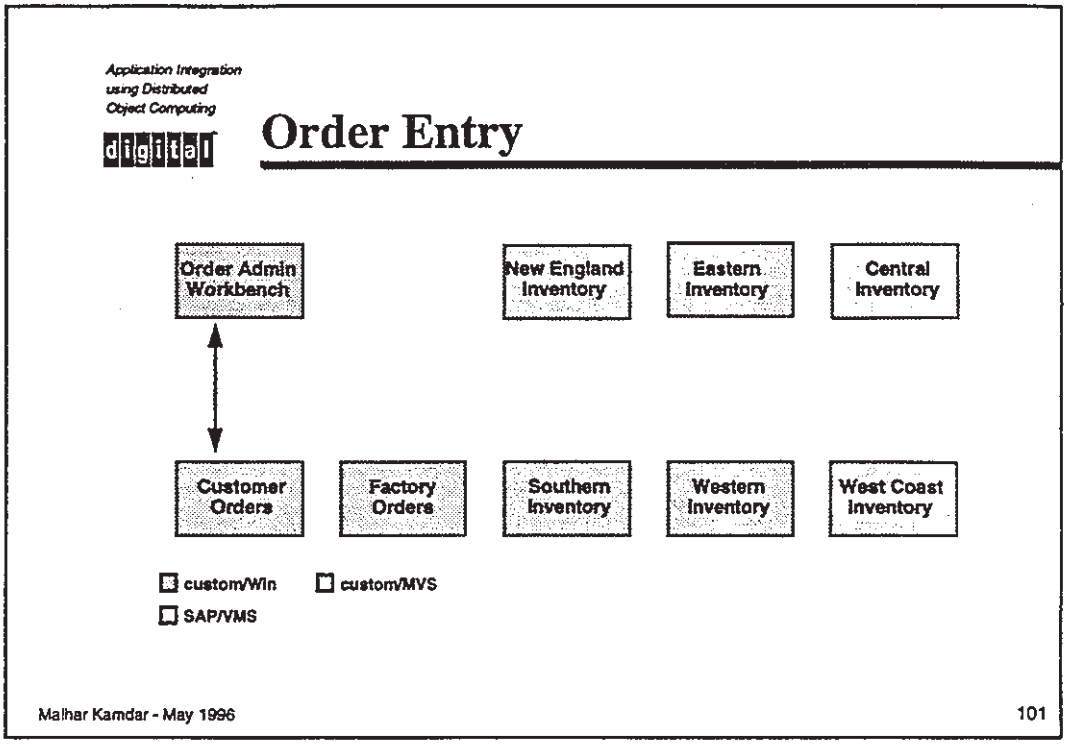
- カスタマのオーダー・エントリ
- 在庫の確保
- リアルタイムの補充
- 発注



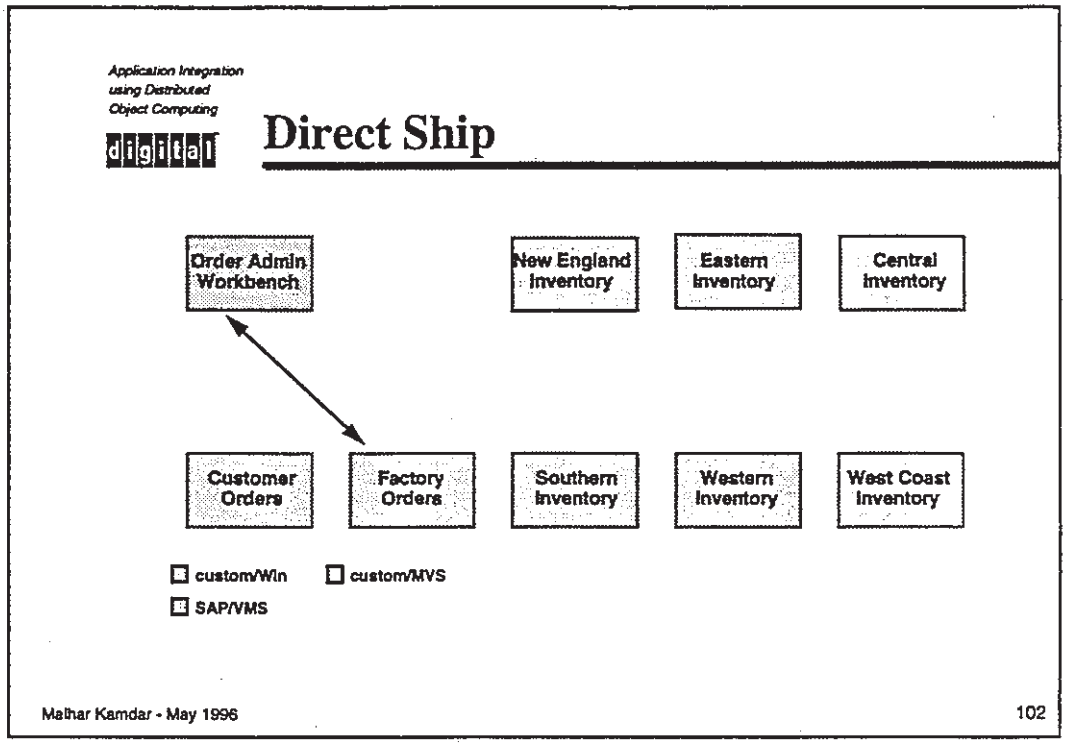
プログレッシヴ製造会社の展開



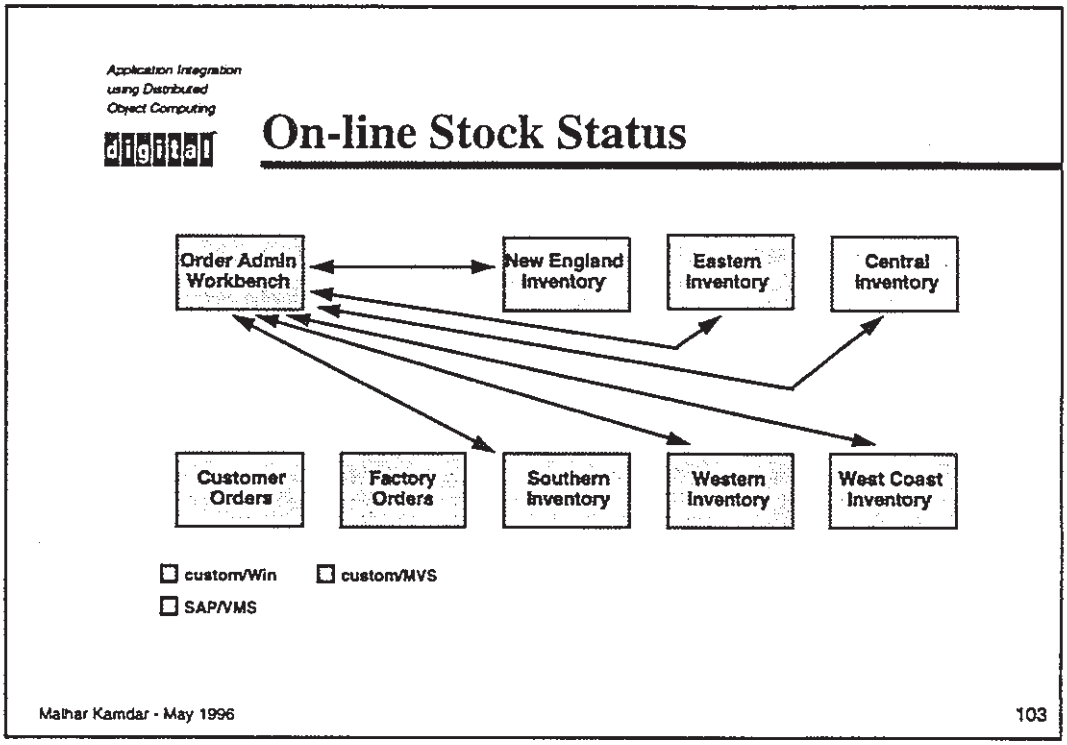
ＰＭＣのアプリケーション


Application Integration
using Distributed
Object Computing
digital
Order Entry
Order Admin Workbench
New England Inventory
Eastern Inventory
Central Inventory
Customer Orders
Factory Orders
Southern Inventory
Western Inventory
West Coast Inventory
custom/Win
custom/MVS
SAP/VMS
Malhar Kamdar - May 1996
101

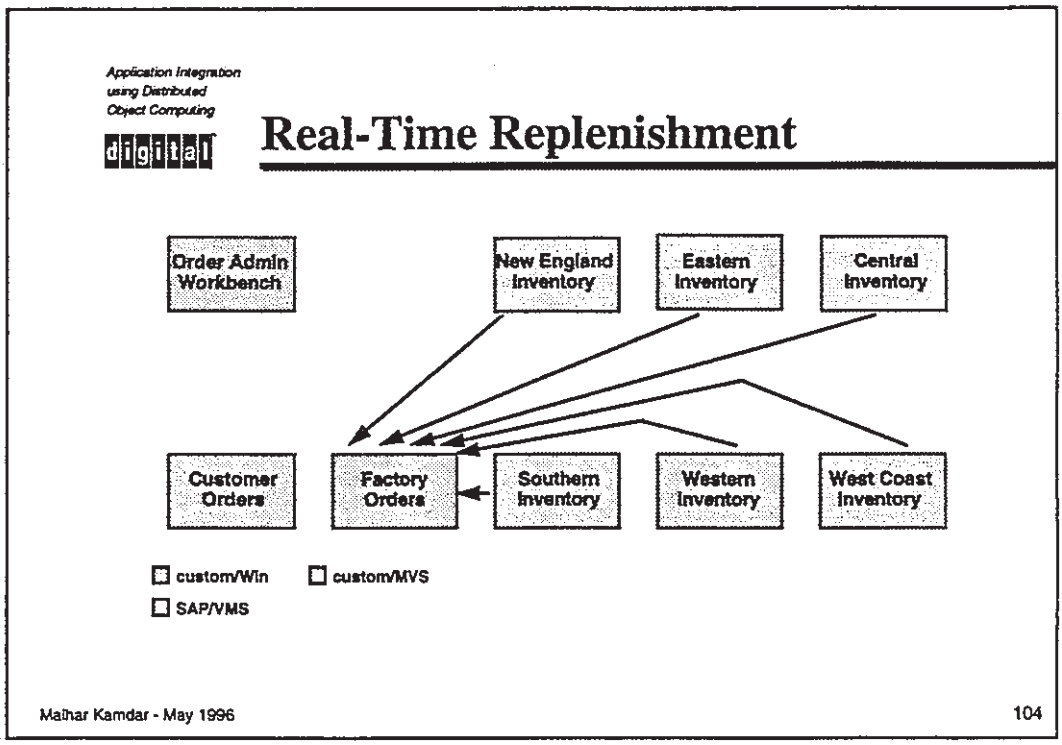
受注


Application Integration using Distributed Object Computing
digital
Direct Ship
Order Admin Workbench
New England Inventory
Eastern Inventory
Central Inventory
Customer Orders
Factory Orders
Southern Inventory
Western Inventory
West Coast Inventory
custom/Win
custom/MVS
SAP/VMS
Malhar Kamdar - May 1996
102

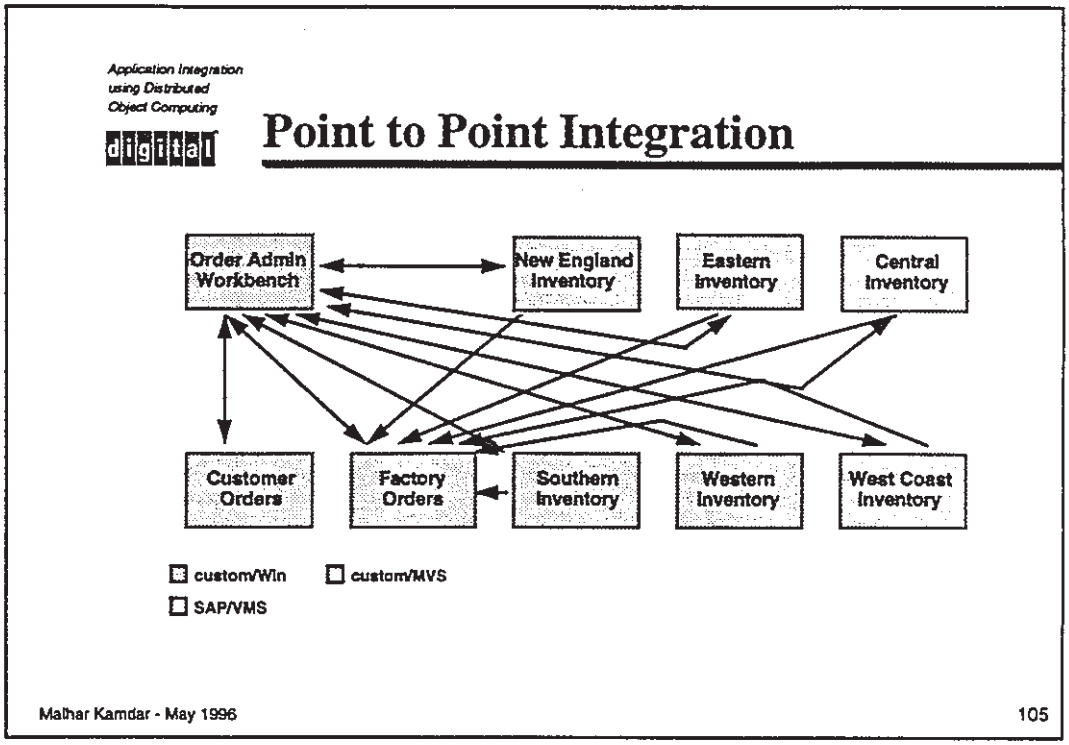
直接輸送


Application Integration
using Distributed
Object Computing
digital
On-line Stock Status
Order Admin Workbench
New England Inventory
Eastern Inventory
Central Inventory
Customer Orders
Factory Orders
Southern Inventory
Western Inventory
West Coast Inventory
custom/Win
custom/MVS
SAP/VMS
Malhar Kamdar - May 1996
103

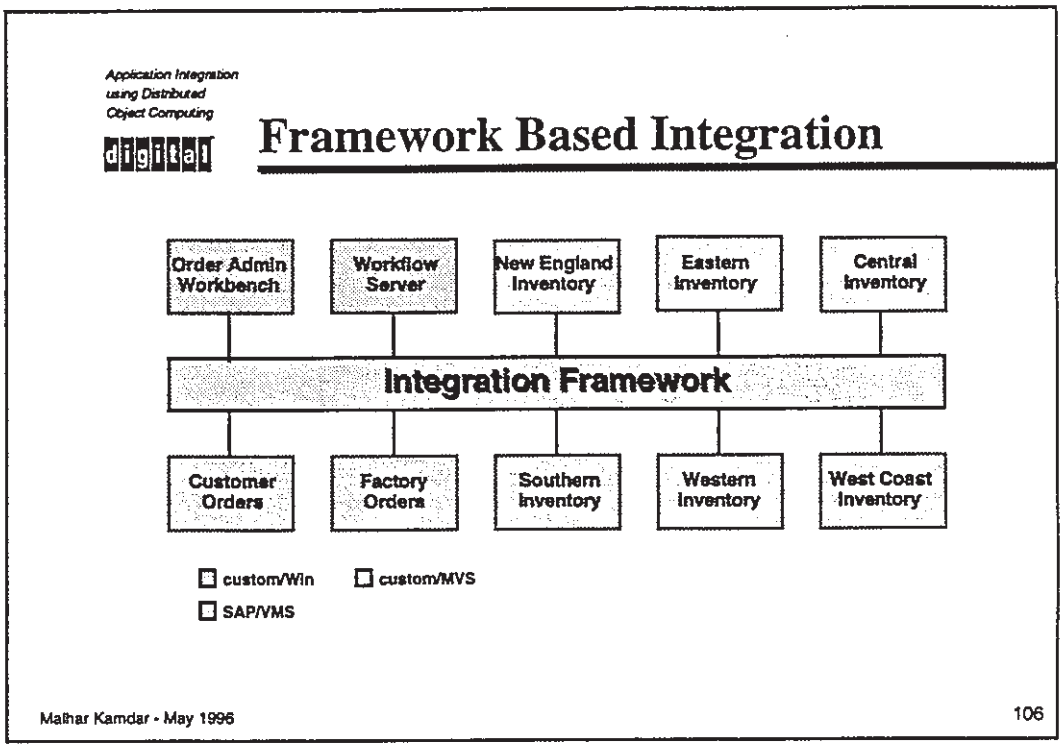
オンライン在庫状況


Application Integration using Distributed Object Computing
digital
Real-Time Replenishment
Order Admin Workbench
New England Inventory
Eastern Inventory
Central Inventory
Customer Orders
Factory Orders
Southern Inventory
Western Inventory
West Coast Inventory
custom/Win
custom/MVS
SAP/VMS
Malhar Kamdar - May 1996
104

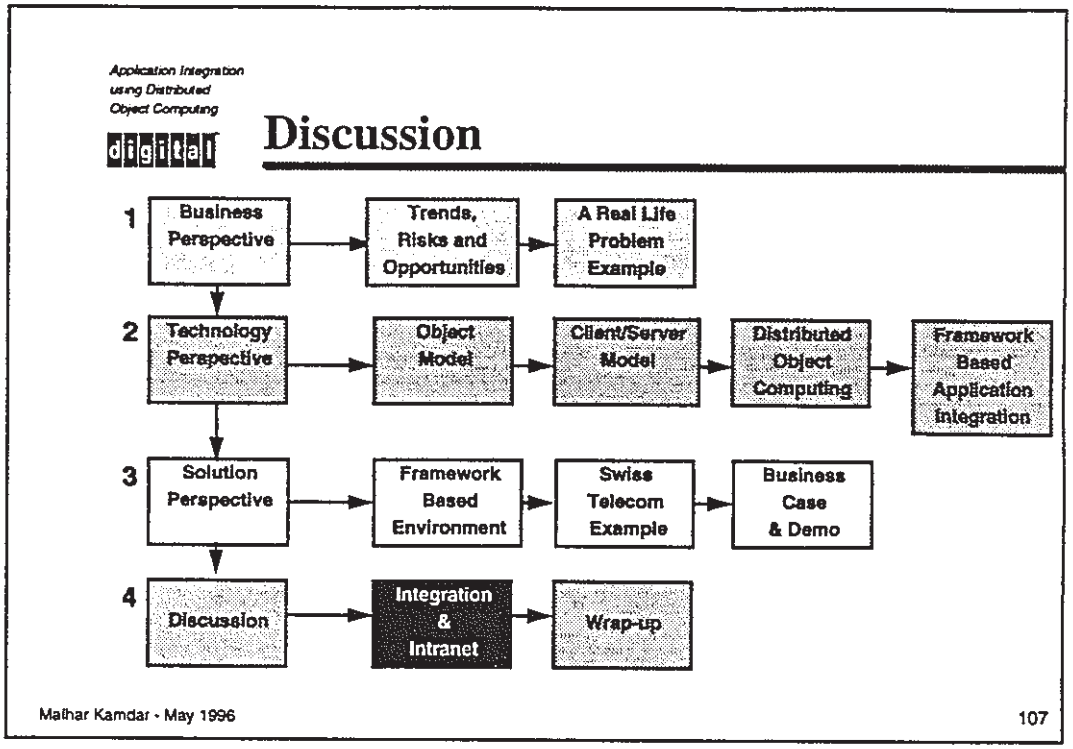
リアルタイム補充



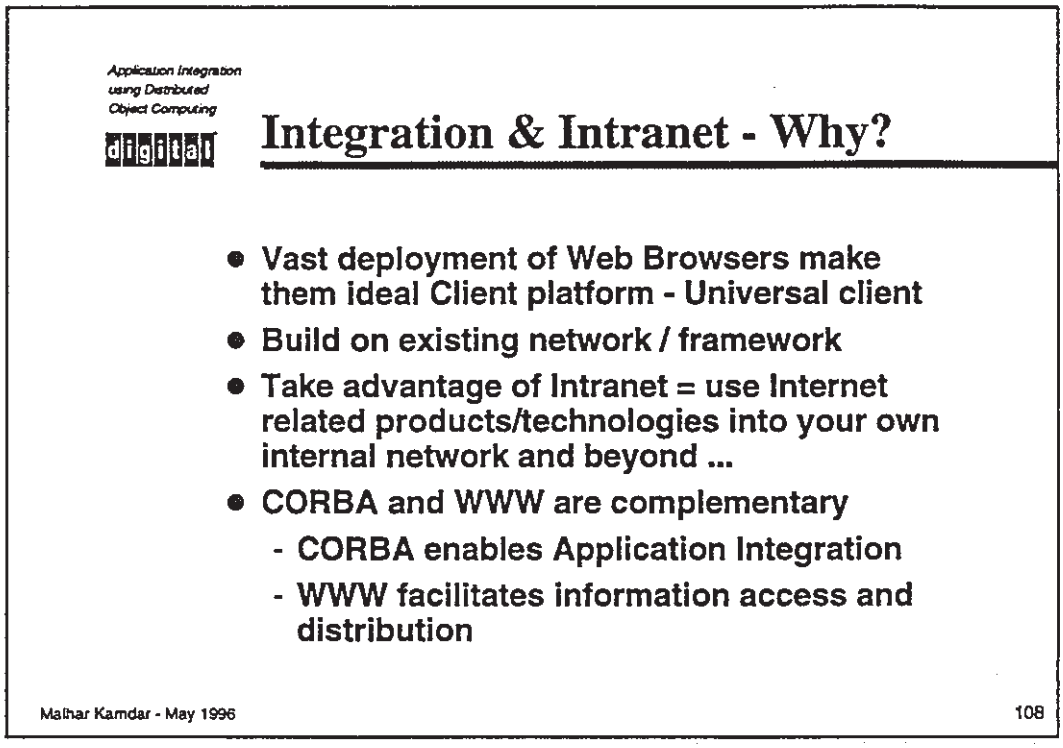
ポイント・ツーポイント統合



フレームワーク・ベース統合



ディスカッション：統合とイントラネット



統合とイントラネット

Ｗｅｂブラウザの広がりは理想的なクライアント・プラットフォームをもたらす

－ 汎用クライアント

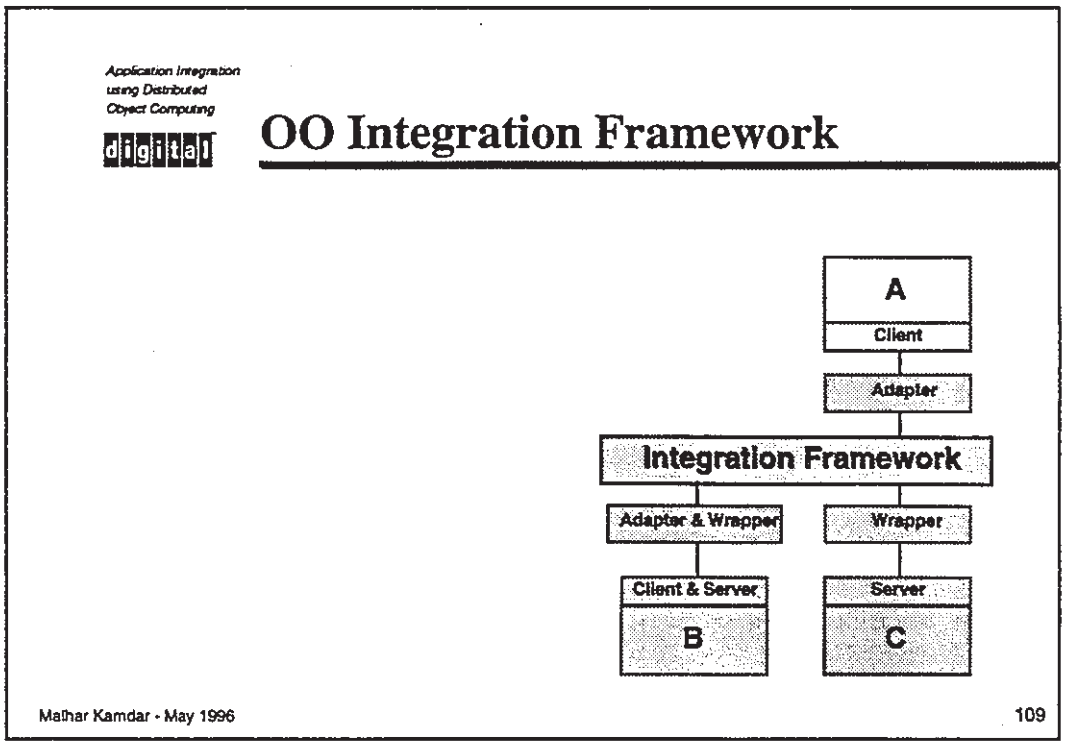
既存のネットワーク／フレームワーク上に構築する

イントラネットを利用する＝インターネットに関連するプロダクト／テクノロジをインターナルなネットワークに利用する

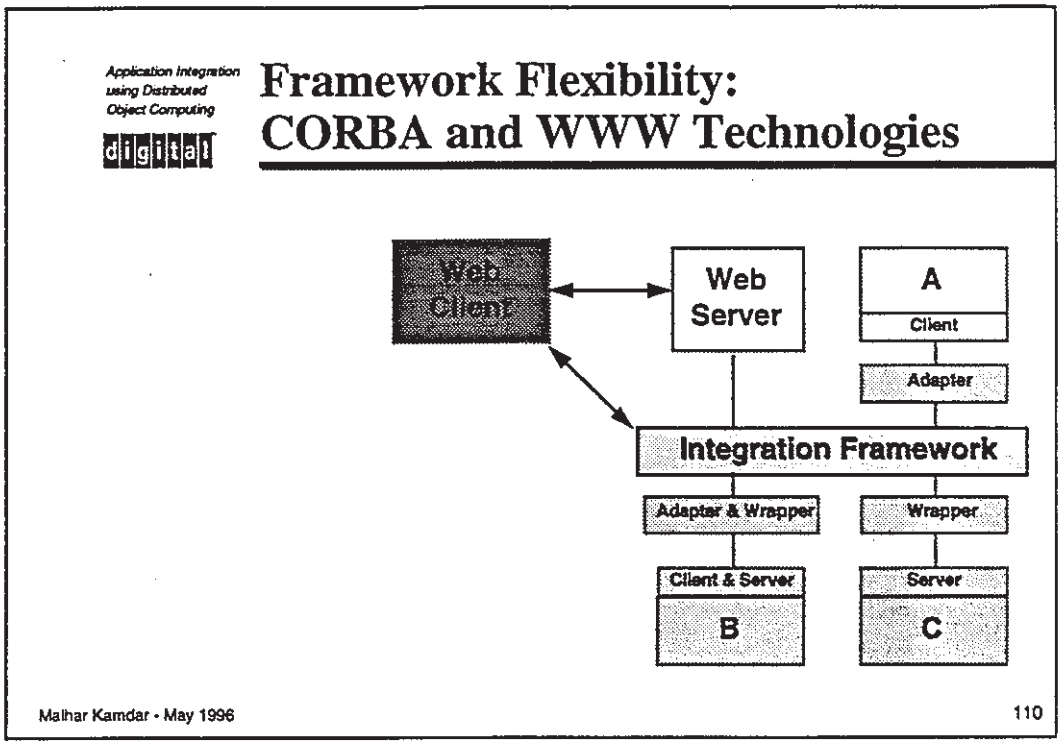
ＣＯＲＢＡとWWWは補完的な関係にある

－ ＣＯＲＢＡはアプリケーション統合を可能にする

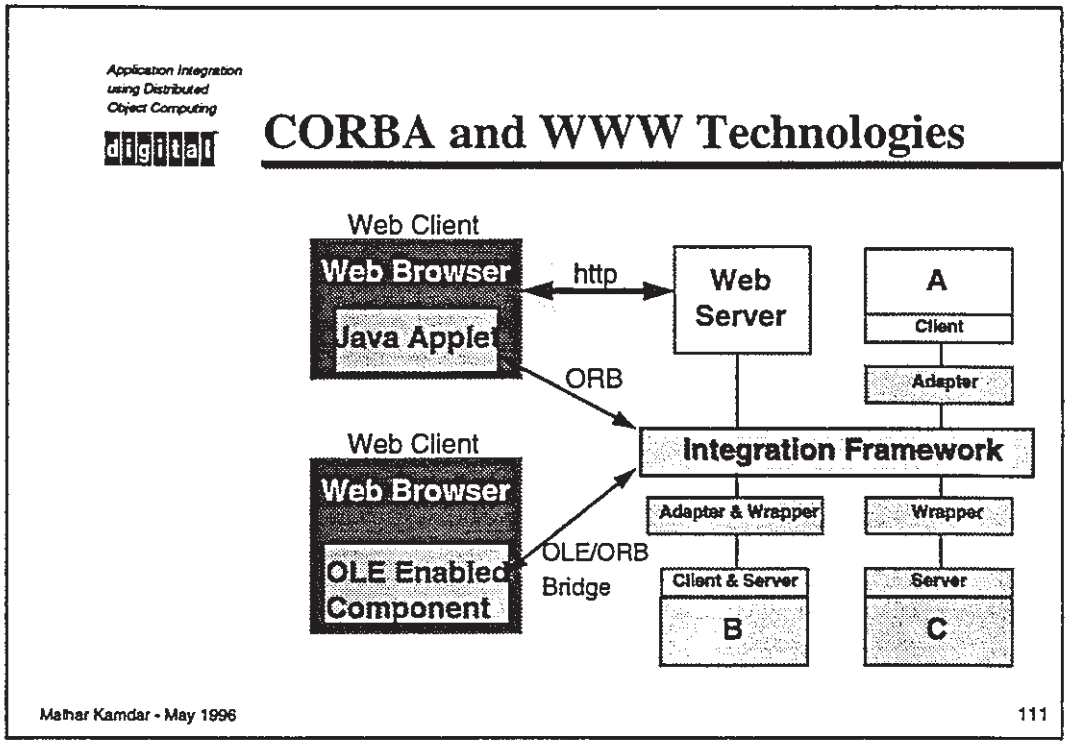
－ WWWはインフォメーション・アクセスと配布を容易にする



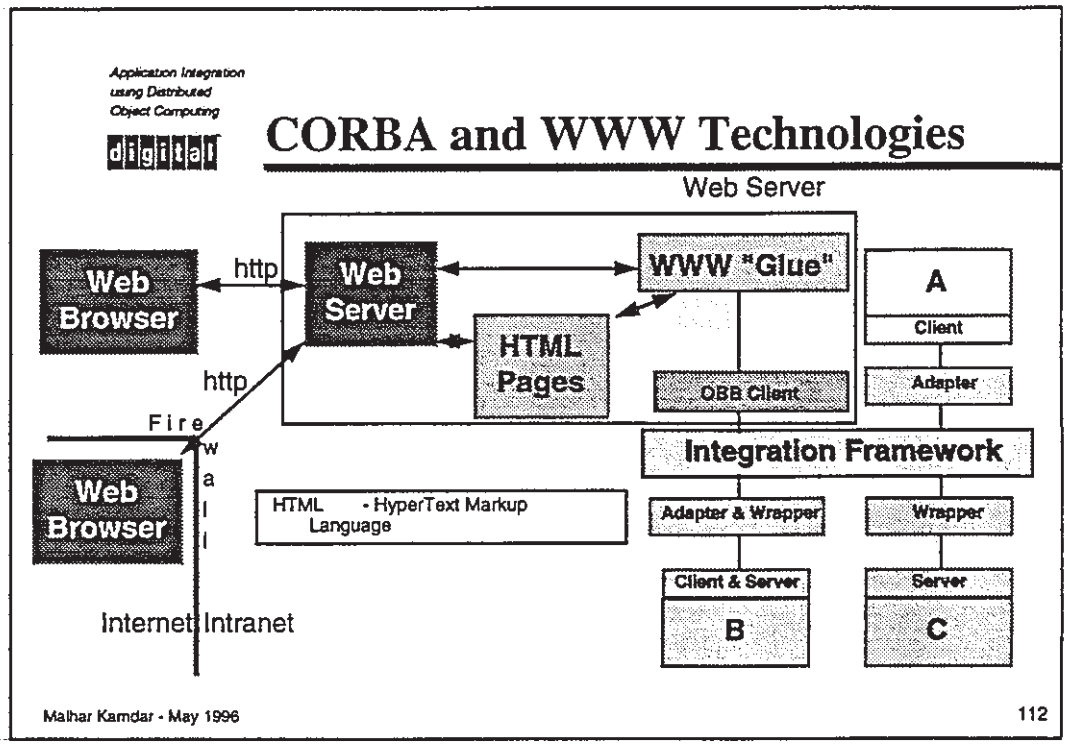
オブジェクト指向の統合フレームワーク



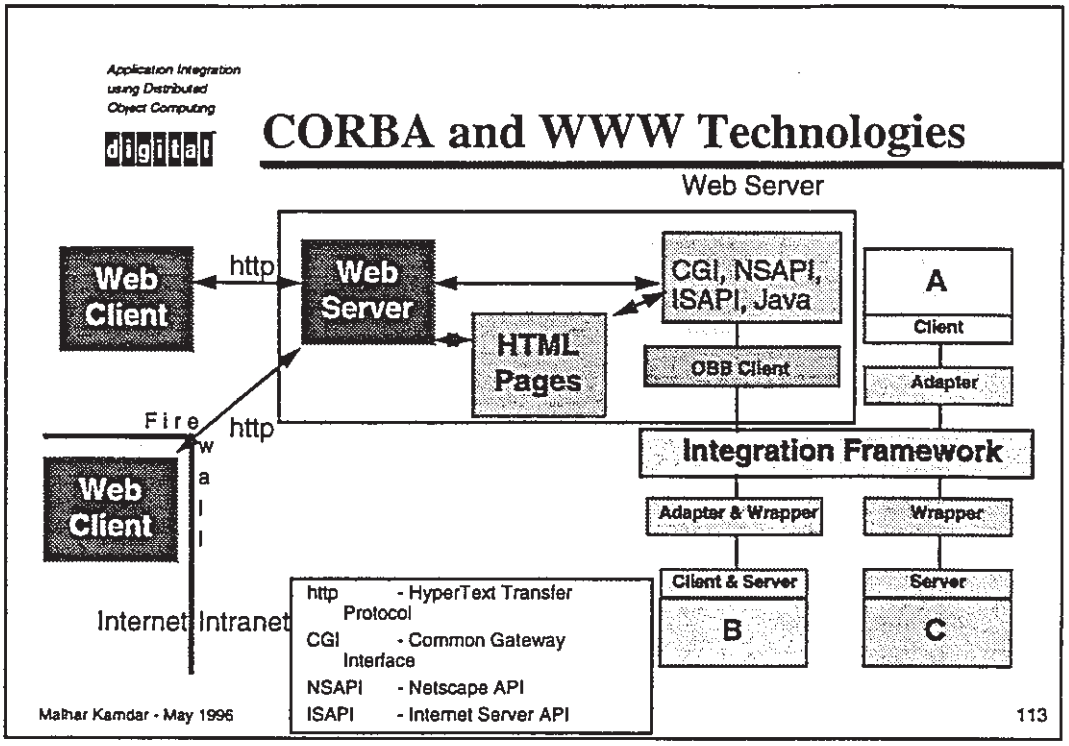
フレームワークの柔軟性：ＣＯＲＢＡとWWWテクノロジ



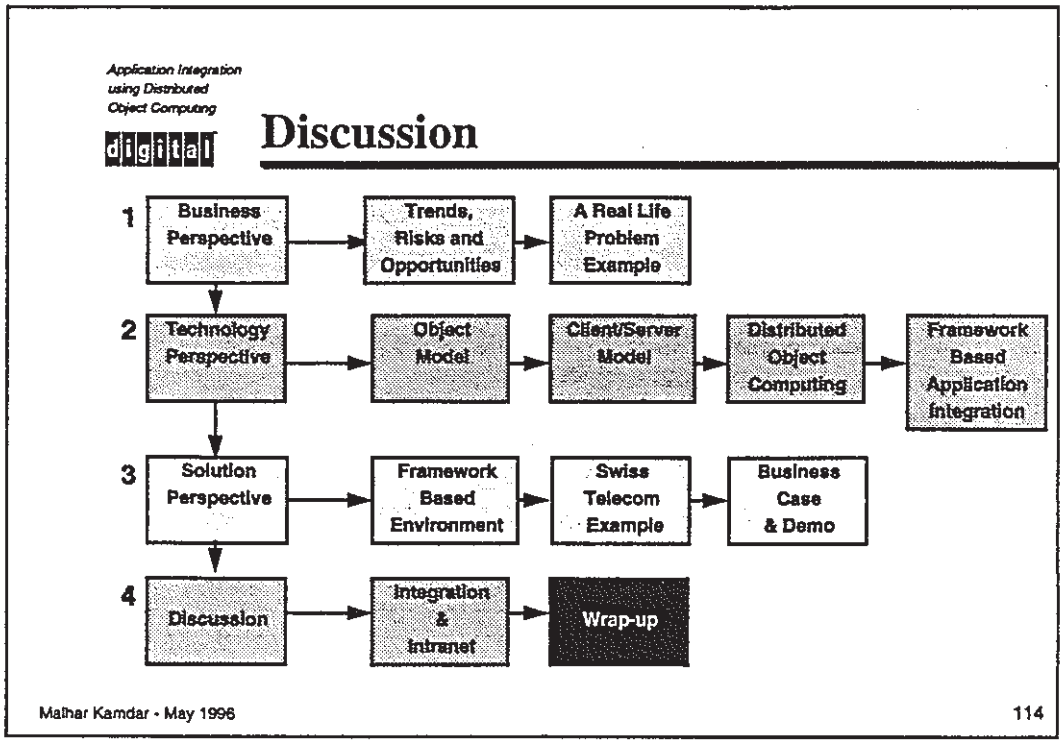
ＣＯＲＢＡとWWWテクノロジ



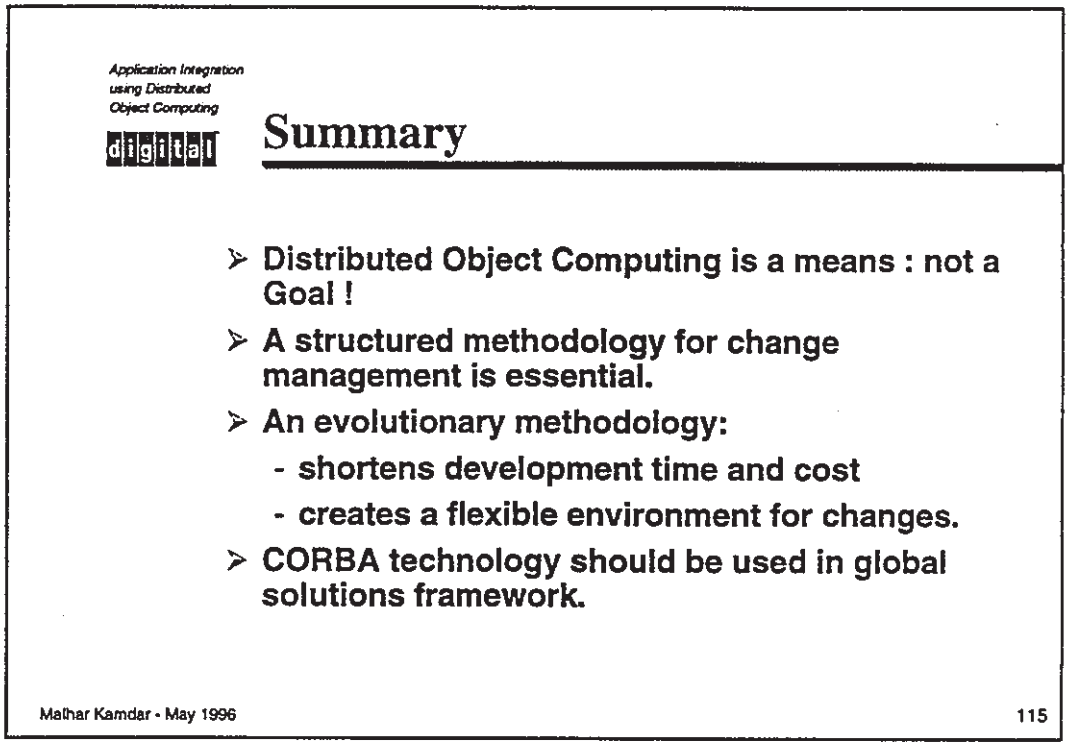
ＣＯＲＢＡとWWWテクノロジ



ＣＯＲＢＡとWWWテクノロジ

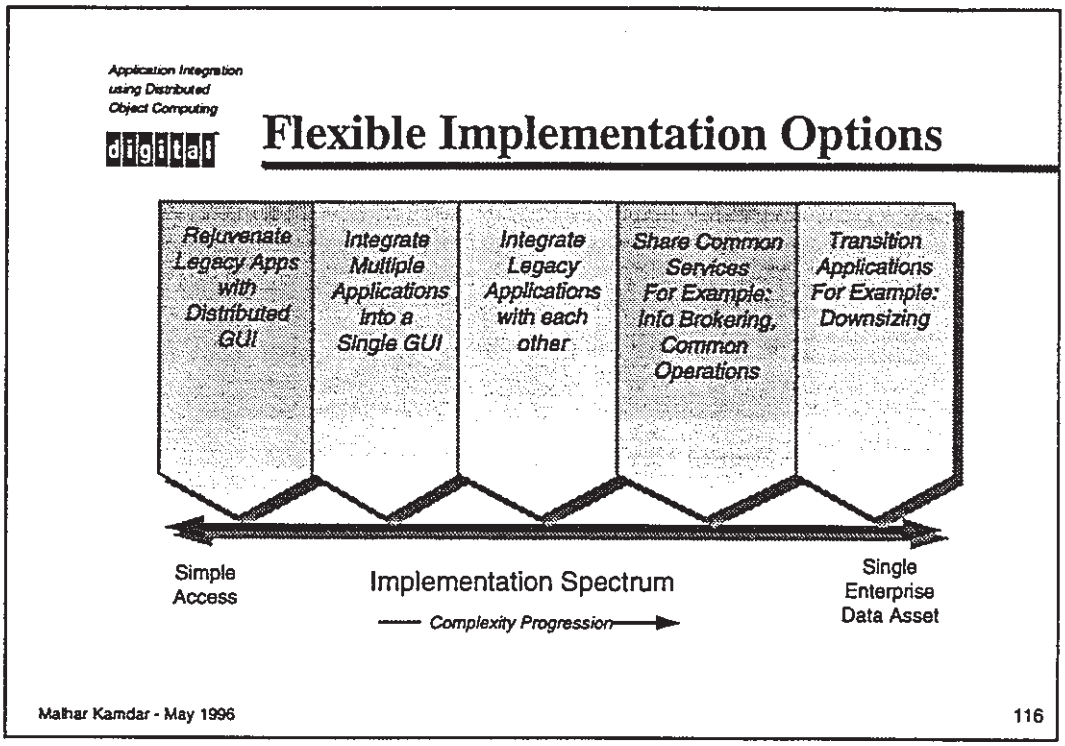


ディスカッション：まとめ



まとめ
分散オブジェクト・コンピューティングは手段であってゴールではない
変更管理のための構造的手法は重要
漸進的手法
－ 開発時間とコストの削減
－ 変更に対する柔軟な環境を構築
ＣＯＲＢＡを全体のソルーションのフレームワークとして使用

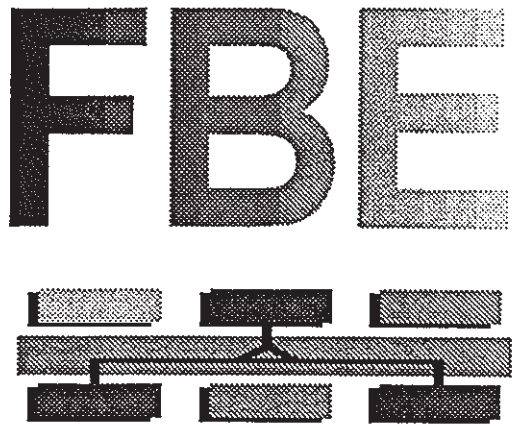

digital™
Digital Equipment Corporation c 1996
WHATEVER IT TAKES



柔軟な実装時のオプション

付録Ｅ　　フレームワーク環境　ＦＢＥ

digital フレームワーク環境



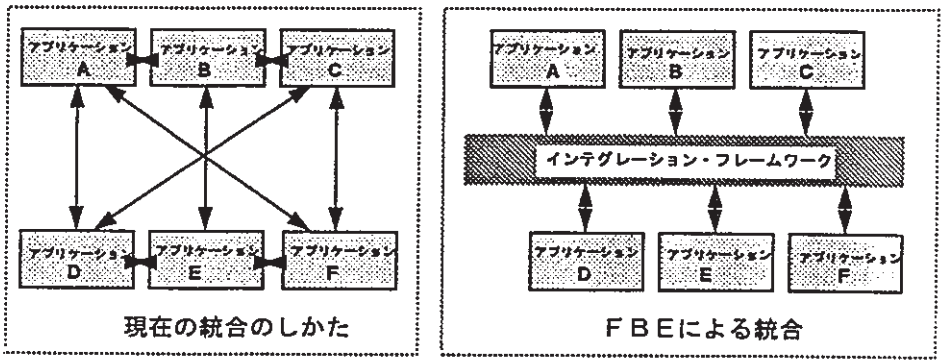
クライアント/サーバ技術センター
統合システム本部
日本ディジタルイクイップメント株式会社

digital 項目

- FBE ~~分析手法~~ MethodF の紹介
  FBE技術
- MethodF 分析サポートツール ObjectPlus の紹介
- FBE で提供される
  基本クラス、産業別クラス

クライアント/サーバ技術センター
統合システム本部
日本ディジタルイクイップメント株式会社

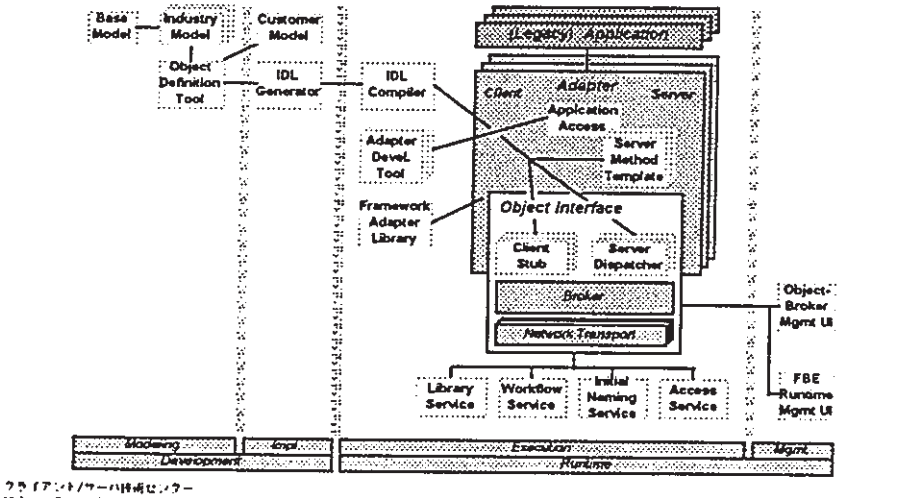
digital FBE フレームワーク環境



ＦＢＥは、現存のＩＴ環境に依存することのない、容易なアプリケーション 統合を可能にする

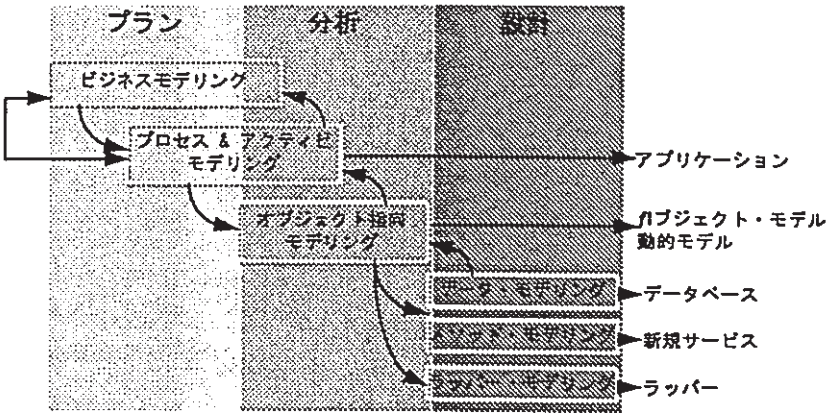
クライアント/サーバ技術センター
統合システム本部
日本ディジタルイクイップメント株式会社

digital FBE アーキテクチャー



クライアント/サーバ技術センター
統合システム本部
日本ディジタルイクイップメント株式会社

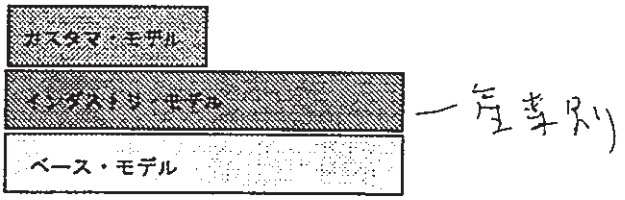
digital プラン、分析、設計



クライアント/サーバ技術センター
統合システム本部
日本ディジタルイクイップメント株式会社

digital ビジネス・オブジェクトのモデリング

- ビジネス・オブジェクトのモデリングが FBE の基盤
- オブジェクトとはプロセスとデータを一つに結び付けたもの
- ビジネス・プロセスとデータをビジネス・オブジェクトとして定義する
- カスタマ・モデルの構築には、ベース・モデルとインダストリ・モデルを基盤として用いる



一定基準

クライアント/サーバ技術センター
統合システム本部
日本ディジタルイクイップメント株式会社

digital FBE ツールキット

- Object Plus
  ProtoSoft 社 Paradigm Plus スペシャル版
- FBE モデル開発キット V1.1
  - IDL ジェネレータ
  - Use Case 拡張
- FBE モデル・キット V1.1
  - ベース・ビジネス・モデル
  - 産業別モデル
- FBE アダプター開発キット V1.0
- FBE ランタイム・サービス・キット V1.0

クライアント/サーバ技術センター
統合システム事部
日本ディジタルイクイップメント株式会社

digital FBE モデル開発キット V1.1

- Use Case モデル
- Use Case、 RAMS テンプレート
- CORBA IDL ジェネレータ
- 基本オブジェクト Type
  (リファレンス・モデル)
- レポート・ツール
- オンライン・ドキュメント

クライアント/サーバ技術センター
統合システム事部
日本ディジタルイクイップメント株式会社

digital FBE モデル・キット V1.1

- FBE ベース・ビジネス・モデル

システム・オブジェクトと基本モデル

- FBE 製造業モデル

製造業での一般的なモデル

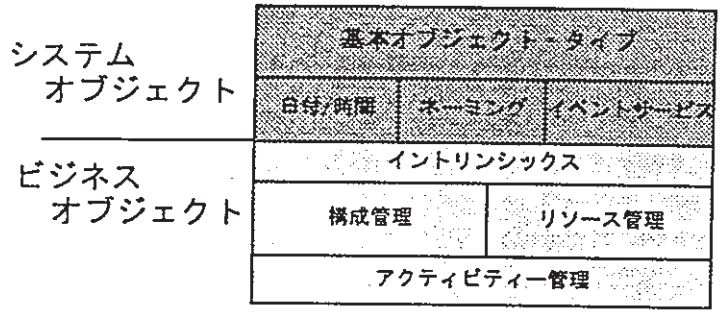
- FBE 産業別拡張モデル

各産業別の製造分野拡張モデル

クライアント/サーバ技術センター
統合システム本部
日本ディジタルイクイップメント株式会社

digital FBE ベース・モデル

- 階層的スコープ
- 境界は概念的なもの



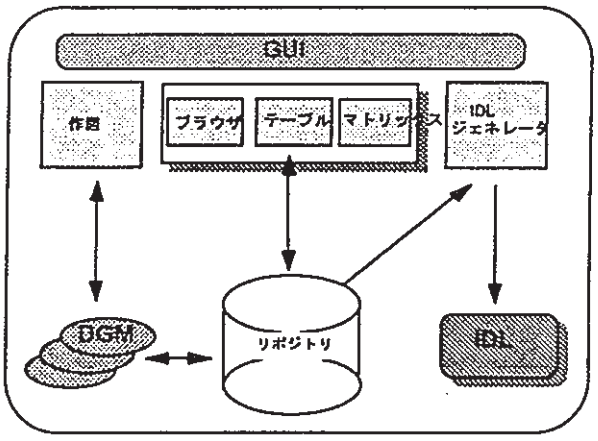
クライアント/サーバ技術センター
統合システム本部
日本ディジタルイクイップメント株式会社

digital FBE 製造業モデル

- Order Management
  - ・製品やサービスの配達リクエスト
  - ・ANSI X.12 EDI に基づく
- Product Definition, Structure, Configuration
  - ・何をどのように生産するか
  - ・ISO STEP に基づく
- Production Management
  - ・オペレーションの制御
- Scheduling

クライアント/サーバ技術センター
統合システム本部
日本ディジタルイクイップメント株式会社

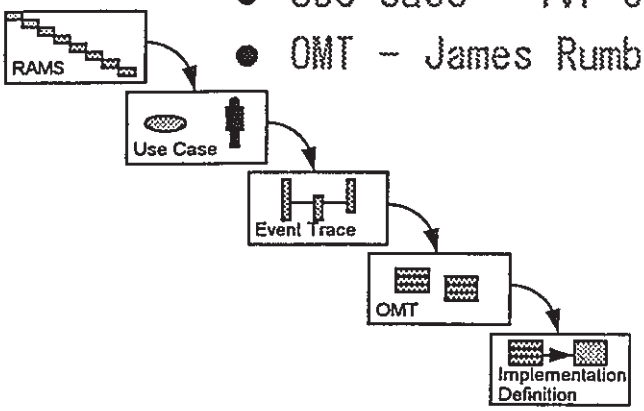
digital Object Plus の構成



クライアント/サーバ技術センター
統合システム本部
日本ディジタルイクイップメント株式会社

digital MethodFで用いる分析技法

- Use Case - Ivr Jacobson
- OMT - James Rumbaugh



クライアント/サーバ技術センター
統合システム本部
日本ディジタルイクイップメント株式会社

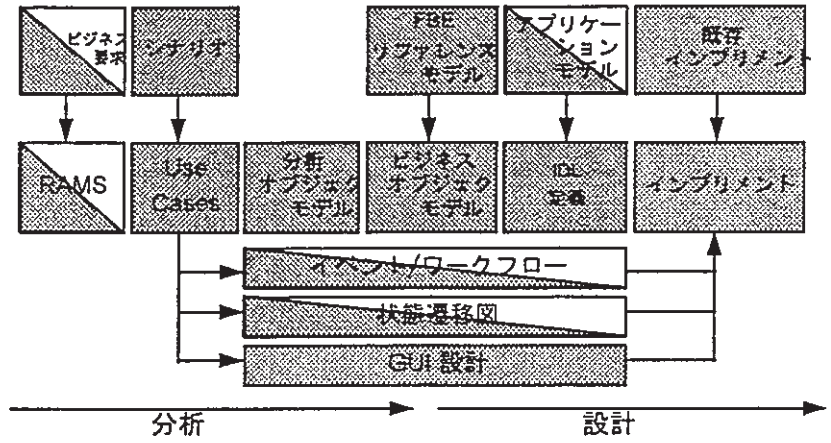
FBE Ver.3
Ver.4を開発中

digital MethodF の手順

- 現状の記述
- キーコンセプト用語集作成
- Actor とその役割を分析
- システムの改善点、目的を定義
- シナリオを書く
- シナリオを分類し Use Case 記述
- Use Case 記述を一般化する
- オブジェクトの選定を行う
- Analysis Model を作成
- Analysis Types を System Types へマップ

クライアント/サーバ技術センター
統合システム本部
日本ディジタルイクイップメント株式会社

digital MethodF



クライアント/サーバ技術センター
統合システム本部
日本ディジタルイクイップメント株式会社

digital RAMS マトリックス

WinD. エトセトラ カタログなど

Actor

カタログ検索	オーダーした製品			
無効オーダ状態	オーダー入力	顧客オーダ		
	製造状態	オーダー処理		製造オーダ
カタログ		請求	製造管理	
	出荷日			スケジューリング

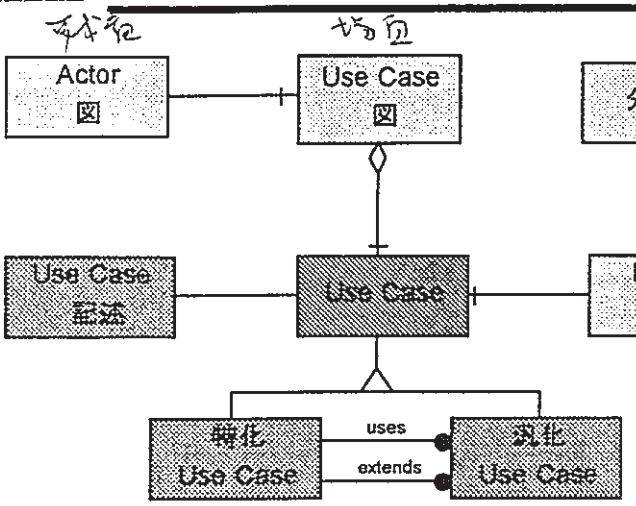
統合システム本部
日本ディジタルイクイップメント株式会社

上下に入力
左右に出力

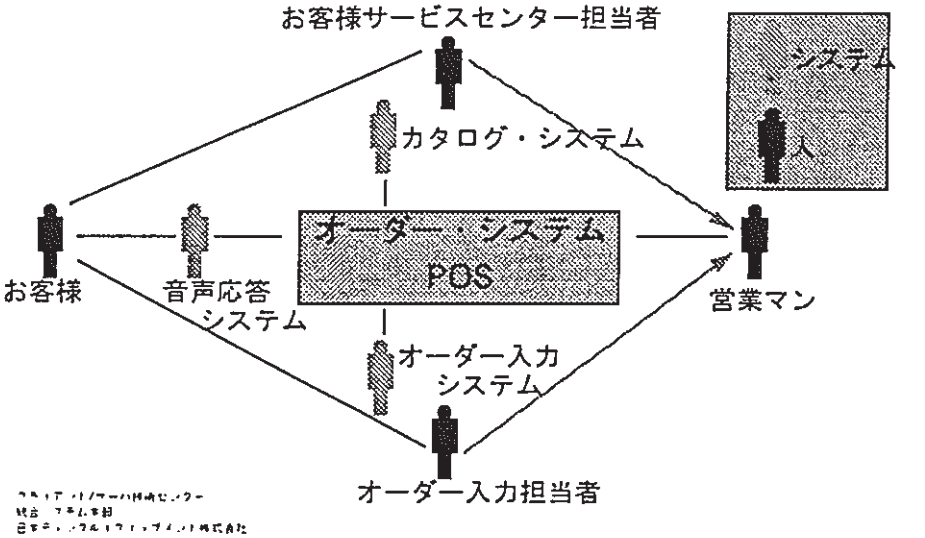
Page 8

全体業務の流れをモデル化する?
Integration Flow

digital Use Case 分析全体構成



digital Actor 図

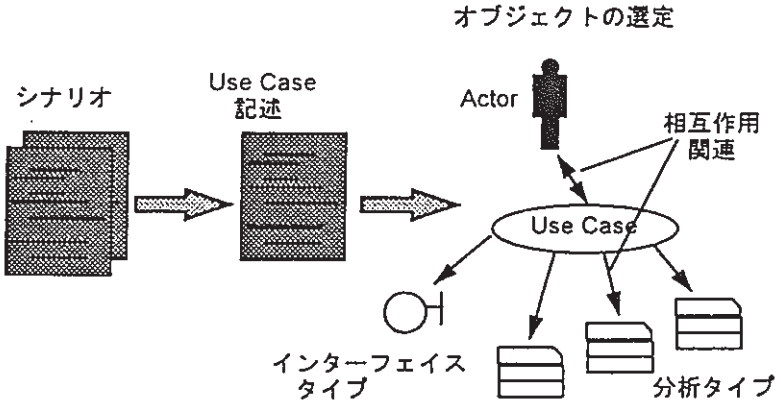


digital シナリオ、Use Case 記述

- 営業マンはカタログ検索ツールを使って全分野のカタログを問い合せる
- 営業マンは１つの製品分野を選択する
- カタログ検索ツールは選択された製品分野で機能一覧と製品一覧を問い合せる
- 営業マンは製品を選択する
- カタログ検索ツールは選択された製品の価格在庫数を返す
- Actor　　営業マン
- インターフェース　　カタログ検索ツール

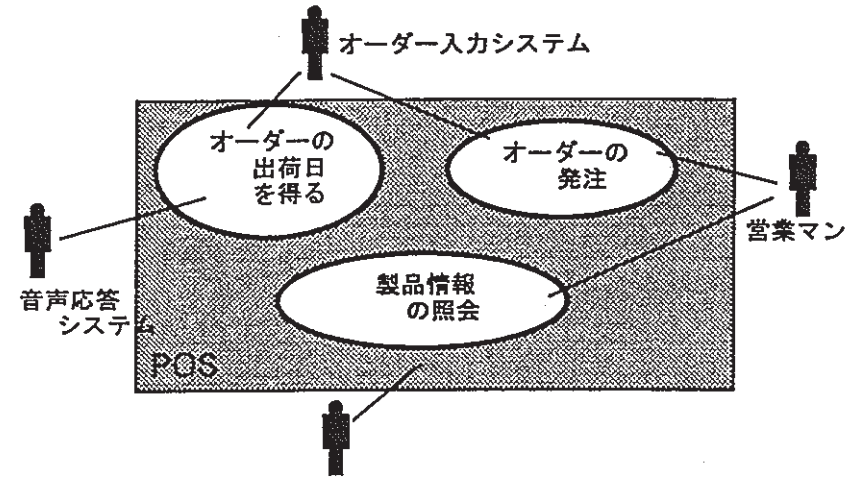
クライアント/サーバ技術センター
統合システム本部
日本ディジタルイクイップメント株式会社

digital Use Case 分析　クラス



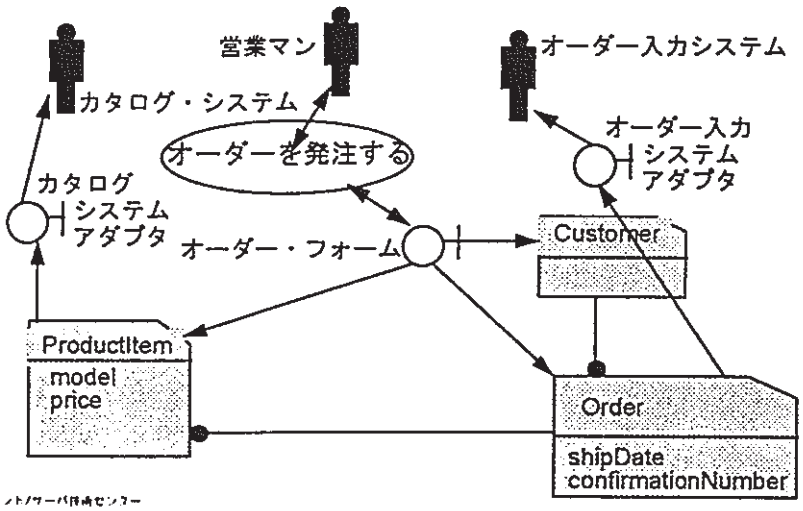
クライアント/サーバ技術センター
統合システム本部
日本ディジタルイクイップメント株式会社

digital Use Case 図



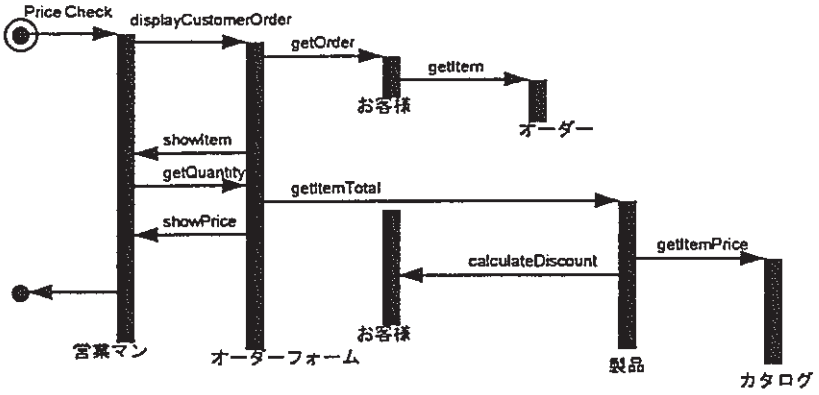
クライアント/サーバ技術センター
統合システム本部
日本ディジタルイクイップメント株式会社

digital オブジェクトの選定、ビュー図



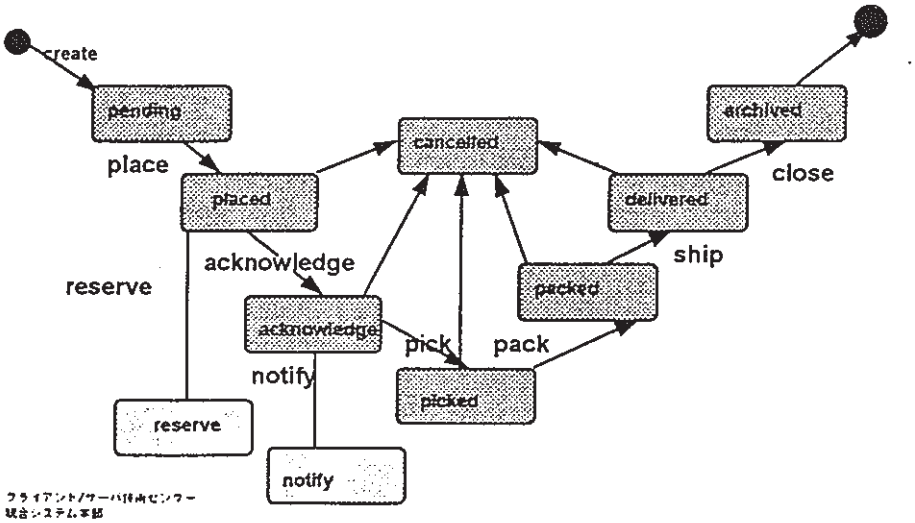
クライアント/サーバ技術センター
統合システム本部
日本ディジタルイクイップメント株式会社

digital イベント・トレース図



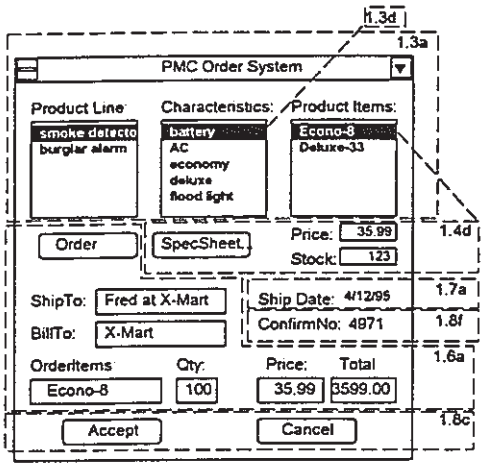
クライアント/サーバ技術センター
統合システム本部
日本ディジタルイクイップメント株式会社

digital 状態遷移図



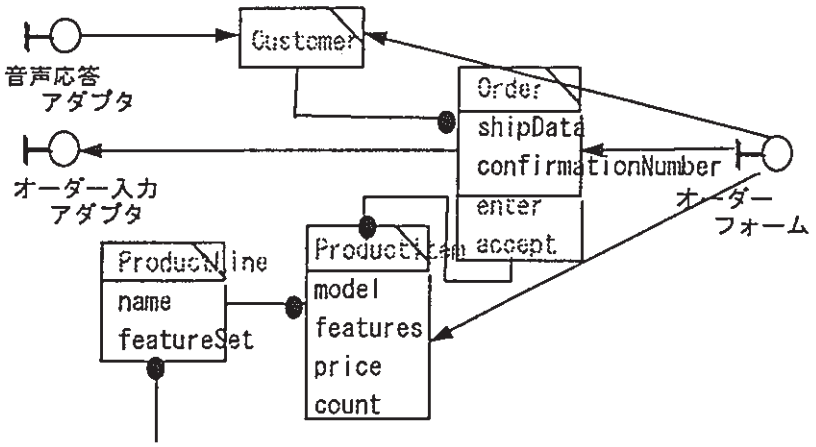
クライアント/サーバ技術センター
統合システム本部
日本ディジタルイクイップメント株式会社

ユーザーインターフェイス スケッチ



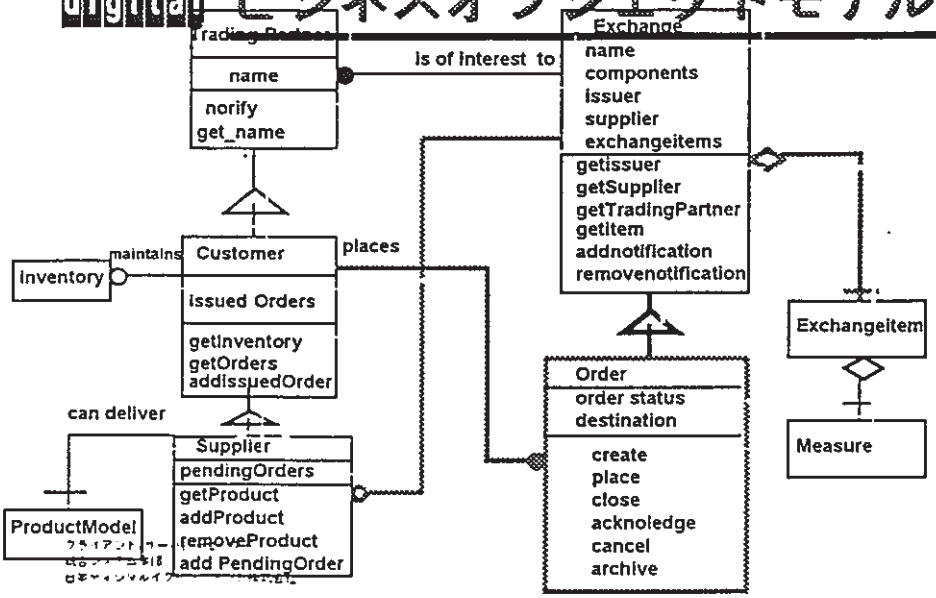
クライアント/サーバ技術センター
統合システム本部
日本ディジタルイクイップメント株式会社

digital 分析モデル Analysis Model



クライアント/サーバ技術センター
統合システム本部
日本ディジタルイクイップメント株式会社

digital ビジネスオブジェクトモデル

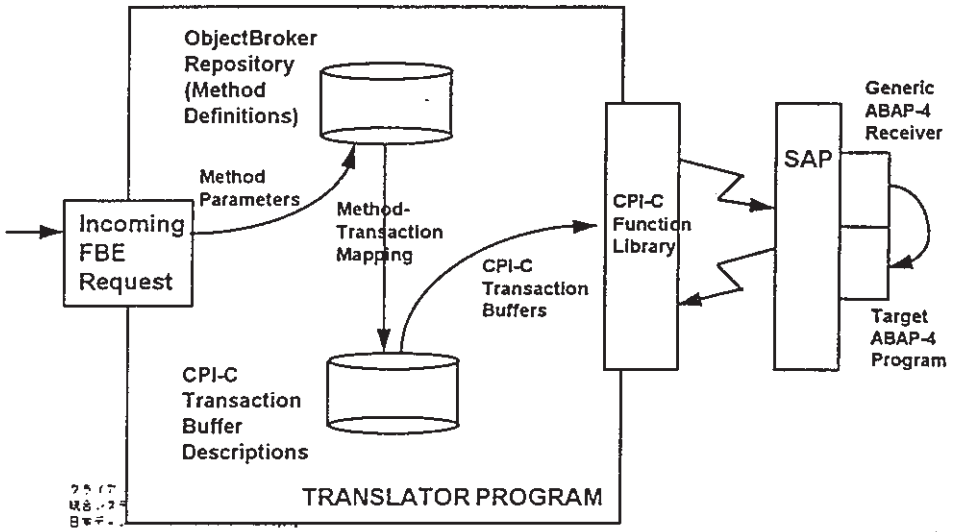


CMT お採用
CMTでお勧め

digital CORBA IDL 生成 -> ObjectBroker

```
module Business {
interface Company;
interface Employee;
// ------------------------------------------------------------------------
// Company
// ------------------------------------------------------------------------

interface Company : FBE::BusinessObject
{
};

typedef sequence < Employee > sequence_Employee;
typedef sequence < Company > sequence_Company;
// ------------------------------------------------------------------------
// Employee
// ------------------------------------------------------------------------

interface Employee : FBE::BusinessObject
{
};

};
```

digital ラッパー開発 (例 SAP R/3)



digital™

Digital Equipment Corporation

付録F　　オブジェクトブローカ入門

オブジェクトブローカ入門

日本ディジタルイクイップメント株式会社

目次

図

表

1. 概要

1.1. オブジェクトブローカ

1.1.1. 製品の特徴

- アプリケーション統合を実現する、オブジェクト指向に基づいた汎用ミドルウェア
- 分散オブジェクト環境を提供
- ＯＭＧ／ＣＯＲＢＡ Ｖ１．２に準拠
- １１のプラットフォームに広く対応
- トランスポートはＴＣＰ／ＩＰ、ＤＥＣｎｅｔに対応
  (Microsoft Windows 環境では WinSock V1.1 インタフェース上で動作)
- Ｍｉｃｒｏｓｏｆｔ Ｗｉｎｄｏｗｓ上での開発環境としてＶｉｓｕａｌ Ｂａｓｉｃ のＡＰＩを提供
- Ｍｉｃｒｏｓｏｆｔ ＷｉｎｄｏｗｓのＯＬＥ，ＤＤＥとの連係機能
- 各種ＧＵＩユーティリティ
- セキュリティ機能（プロキシ設定）
- 開発用キット（Development）とランタイムキット（Runtime）で提供
- 提供メディアは３．５インチＦＤ（Microsoft Windows, Macintosh）とＣＤ－ＲＯＭ
- コモン・オブジェクト・モデル（ＣＯＭ）によるＯＬＥ２．０との相互連係（開発中）

1.1.2 動作環境

表 1 オブジェクトブローカが動作する環境 （1994/11 現在）

ハードウェア	オペレーティングシステム	ネットワーク・ソフトウェア
VAX	OpenVMS VAX V5.2 以降	TCP/IP Services V3.0, DECnet V5.4-5.5
Alpha AXP	OpenVMS AXP V1.5 以降	TCP/IP Services V3.0
Alpha AXP	OSF/1 V2.0	TCP/IP (OS にバンドルされるもの)
Alpha AXP	Microsoft WindowsNT V3.1	TCP/IP (OS にバンドルされるもの), DECnet
ULTRIX/RISC	ULTRIX V4.3 以降	TCP/IP (OS にバンドルされるもの), DECnet V4.2
HP 9000-7xx, -8xx	HP-UX Version 8.07	TCP/IP (OS にバンドルされるもの)
IBM AIX	AIX V3.2	TCP/IP (OS にバンドルされるもの)
Sun Sparc	SunOS V4.1.1	TCP/IP (OS にバンドルされるもの), DECnet
IBM-PC 互換機	Microsoft WindowsNT V3.1	TCP/IP (OS にバンドルされるもの), DECnet
IBM-PC 互換機	Microsoft Windows V3.1	TCP/IP (WinSock V1.1 対応ソフトウェア), DECnet
Macintosh	System 7.0, 7.1	MacTCP V1.1.1-V2.04, TSS-Net V2.4.1, DECnet など

1.2 従来のアプリケーション

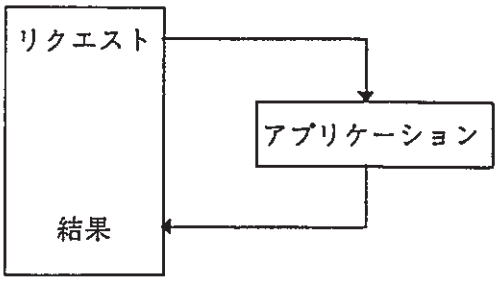


図 1 従来のアプリケーション

従来のアプリケーションでは、それぞれのシステムがユーザからの入力をもとにして処理を実行します。その処理結果はユーザに返すことになります。

この構造では、複数のアプリケーションを組み合わせる場合に、それぞれのアプリケーションへのデータ入力および出力の取り込をユーザがおこなうことになります。

1.3 クライアント／サーバシステムとは

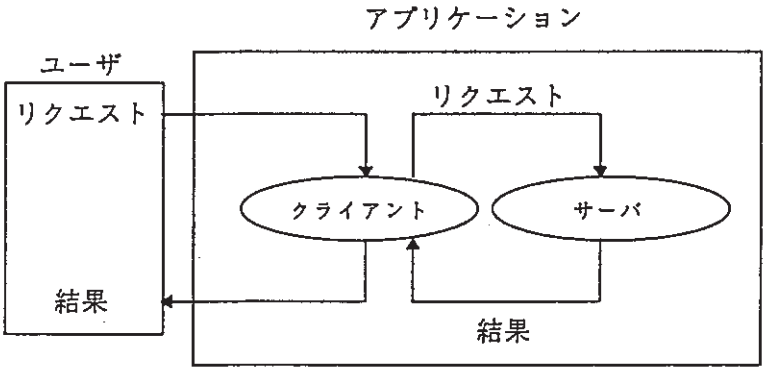


図 2 クライアント／サーバシステム

クライアント／サーバシステムとはソフトウェアシステムの一形態で、処理を依頼するアプリケーションと依頼された処理を実行するアプリケーションに分割されているシステムです。処理を依頼するアプリケーションをクライアントアプリケーション、依頼された処理を実行するアプリケーションをサーバアプリケーションといいます。

1.3.1. クライアント／サーバのシステムの利点

クライアント／サーバシステムではクライアントアプリケーションとサーバアプリケーションが分離しているため、次のような利点があります。

- 分散環境に適しています
  通常、クライアントアプリケーションとサーバアプリケーションが別のプロ

セスで実装されるため、それぞれを異なるノードで実行することも可能となります。

- ＰＣやＷＳを利用したＧＵＩを利用できます
  ユーザとのインターフェースは、ＰＣやＷＳでクライアントアプリケーションが処理をして、データベースのアクセスや計算能力を必要とする数値計算などをサーバアプリケーションで実行することが可能となります。
- コスト的に有利です
  従来の集中型のシステムでは、計算機が本来の計算処理だけではなく、ユーザインターフェースも処理していました。しかしクライアント／サーバシステムでは、ユーザインターフェースはＰＣやＷＳなどが担当するため、以前よりも小規模な計算機で十分な計算能力が得られることになります。

1.3.2 クライアント／サーバシステムの問題点

クライアント／サーバシステムでは、クライアントアプリケーションとサーバアプリケーションの関係の管理が問題となります。

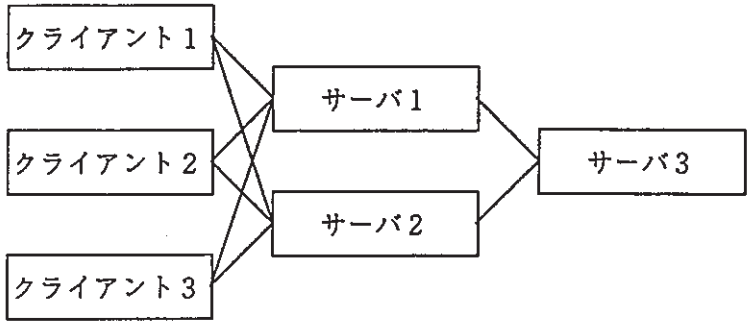


図 3 クライアント／サーバの組み合わせ

単純なクライアント／サーバシステムでは、基本的に関係が一対一に対応しており、その個々において整合性を取る必要があります。

このとき、特にクライアントアプリケーションとサーバアプリケーションの数が増えた場合にはこの管理が困難になります。例えば、クライアントアプリケーションとサーバアプリケーションを別種類の計算機で実現した場合、それぞれの計算機で利用している数値表現の変換をおこなう必要があります。

1.4 フレームワーク方式による３層式クライアント／サーバシステム

オブジェクトブローカを使用した分散オブジェクト環境を利用することで、「フレームワーク方式による３層式クライアント／サーバシステム」を構築すること

ができます。

1.4.1. フレームワーク方式

フレームワーク方式とは、クライアントアプリケーションとサーバアプリケーションが1対1に接続するのではなく、汎用的なプロトコルで接続する形態です。

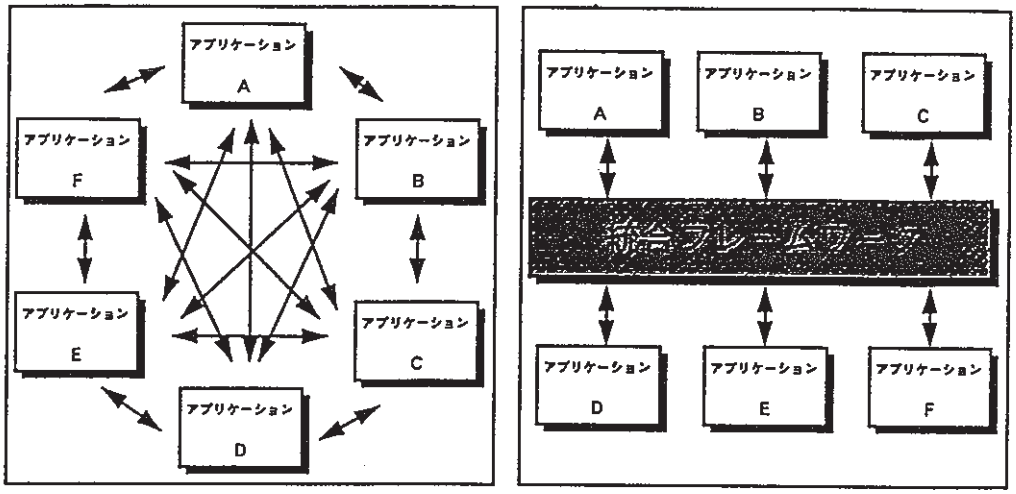


図 4 フレームワーク方式のクライアント・サーバシステム

これをプログラミング言語の形態と比較すると、（表 2）のようになります。

表 2 プログラミング言語とネットワークプロトコルの比較

プログラミング言語	ネットワークプロトコル	問題点
アセンブラ	ソケット通信	パラメータの渡しかた
サブルーチンコール	ＲＰＣ	1対1の対応
オブジェクト指向言語	フレームワーク方式	

1.4.2 3層式クライアント／サーバシステム

一般的なシステムは、次の三つの論理的な層に分離できます。

- プレゼンテーション層
  プレゼンテーション層はユーザにもっとも近い層で、ユーザとシステムとのインターフェースを処理します。基本的にこの層ではデータ処理はおこないません。
- コントロール層
  コントロール層はシステムが扱うデータを処理する層です。ユーザとの入出力やデータの保管はおこないません。
- データ層

データ層はデータの保管をおこなう層です。この層ではユーザとの入出力やデータ処理はおこないません。

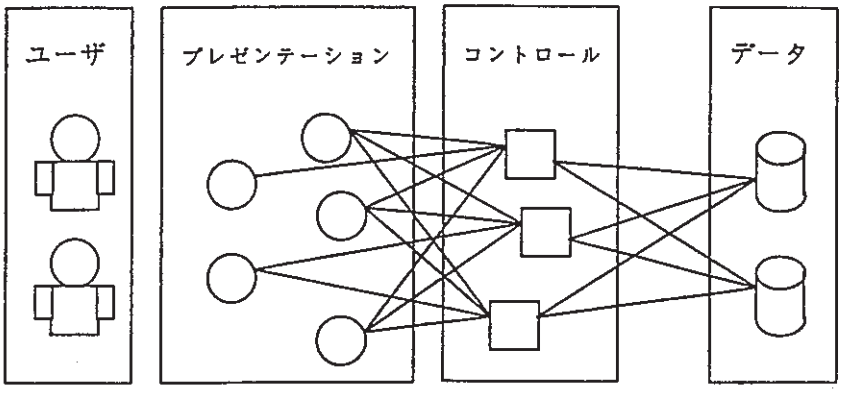


図 5 3層クライアント／サーバ構造

一般に「システムのロジック」と呼ばれる部分はコントロール層が担当します。システムのロジックはビジネス上の要求に依存するため、その変更にともないコントロール層がその影響を受けることになります。

3層型ではそれぞれの層を独立したアプリケーションで構成します。その他のクライアント／サーバシステム、例えばＯＤＢＣを利用したシステムではクライアントアプリケーションがプレゼンテーション層とコントロール層を担当する2層型で構成されています。

2層型の構成の場合には、ビジネス上の要求が変更された場合に、プレゼンテーション層を含んだクライアントアプリケーションを変更することになります。3層型ではコントロール層のアプリケーションのみの変更であり、プレゼンテーション層のアプリケーションは変更する必要がありません。

一般にプレゼンテーション層のアプリケーションは、コントロール層に属する複数のアプリケーションの機能を利用していると考えられます。逆にいうと、一つのコントロール層の機能が複数のプレゼンテーション層で利用されていることになります。このためコントロール層の変更にともなうアプリケーション層の修正はなるべく避けるべきです。3層型ではコントロール層の影響を最小限に押さえることが可能です。

1.5 分散オブジェクト環境

オブジェクトブローカでは、従来のクライアント／サーバシステムにオブジェクト指向を取り入れることで上記の問題を解決します。オブジェクトブローカが実

現するクライアント／サーバ環境を「分散オブジェクト環境」と呼びます。

分散オブジェクト環境については、オブジェクト・マネージメント・グループ（OMG）という標準化団体が規格を定めています。OMGはオブジェクト指向に関するコンピュータベンダおよびユーザの非営利団体で、およそ４００社が参加しています。（1994/11 現在）

OMGでは「オブジェクト・マネージメント・アーキテクチャ（OMA）ガイド」を発表し、その中で「リファレンスモデル」として分散オブジェクト技術の枠組みを定めています。（OMAリファレンスモデルの概要については 1.6 節で紹介します）

OMAリファレンスモデルで定められる分散オブジェクト環境では、サーバアプリケーションをオブジェクトとしてみなします。それぞれのサーバアプリケーションを種類ごとにクラスとして定義することになります。

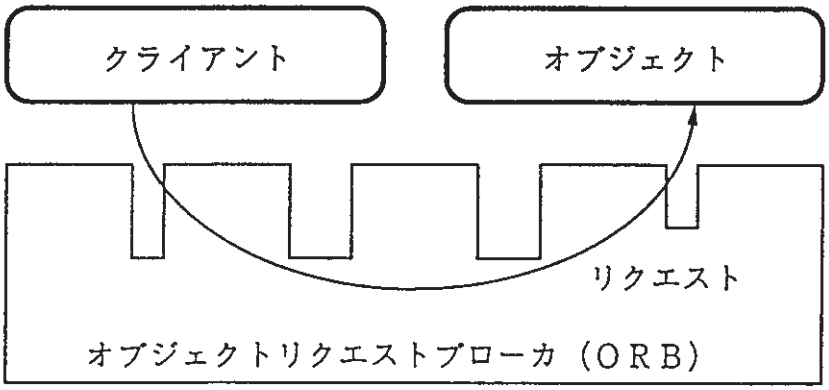


図 6 ＯＲＢを通じて送られるリクエスト

クライアントアプリケーションから処理を依頼するときには、オブジェクトリクエストブローカに、目的のオブジェクトを定義したクラスとそのメッセージを送ります。すると、オブジェクトリクエストブローカは指定されたクラスとメッセージから導かれるサーバアプリケーションにリクエストを伝えます。このようにしてクライアントからオブジェクトにアクセスできるようになります。

クライアントアプリケーションとサーバアプリケーションは、分散オブジェクト環境に対応することで、お互いの組み合わせを考慮する必要がなくなります。

「共通オブジェクトリクエストブローカアーキテクチャ（ＣＯＲＢＡ）」は、このオブジェクトリクエストブローカの規格を定めたものです。オブジェクトブローカは、このＣＯＲＢＡに準拠したＤＥＣの製品です。

1.6. OMAリファレンスモデル

OMAリファレンスモデルは次の要素から構成されています。

- オブジェクトリクエストブローカ（ＯＲＢ）
- オブジェクトサービス
- 共通ファシリティ
- アプリケーションオブジェクト

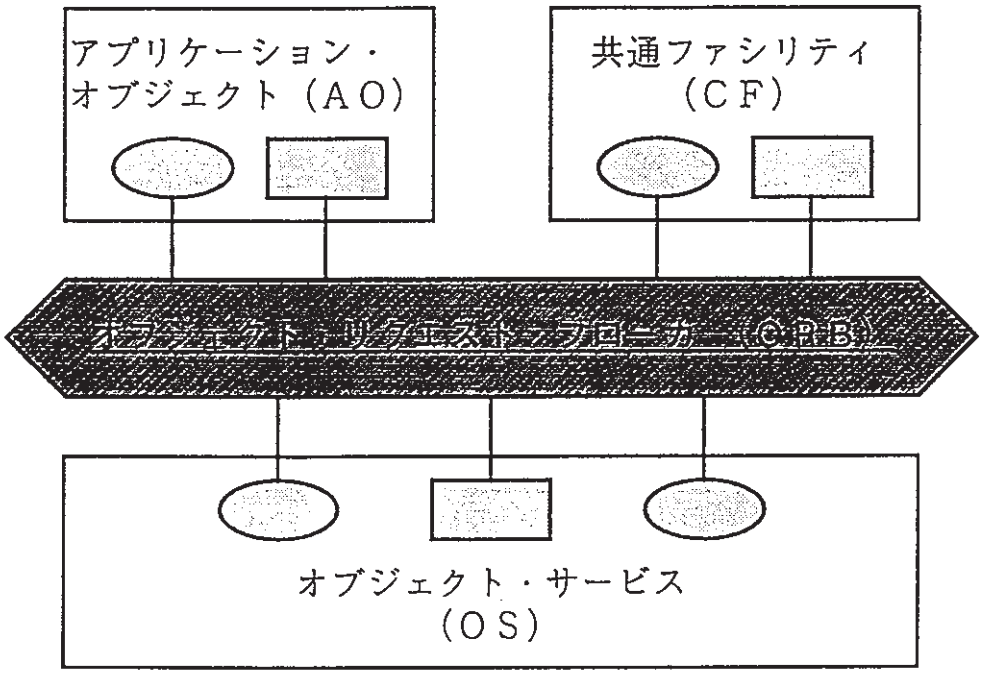


図 7 OMAリファレンスモデル

1.6.1. オブジェクトリクエストブローカ（ＯＲＢ）

オブジェクトリクエストブローカは、オブジェクトがリクエストを生成したり、レスポンスを受け取る環境を提供します。この環境によって分散した異なるマシン上のアプリケーションを結び付けることができるようになります。オブジェクトリクエストブローカに関してはＣＯＲＢＡ１．１として正式な仕様がまとめられています。

1.6.2 オブジェクトサービス

オブジェクトサービスでは、すべてのクラスで実装あるいは継承する可能性があるような基本的なオペレーションを定義します。オブジェクトサービスによって提供されるオペレーションはオブジェクトリクエストブローカを通じて利用されることになります。

多数のオブジェクトによって実装される汎用的かつ抽象的な機能は共通ファシリティで提供されることが望ましいですが、共通ファシリティ自体はオブジェクトサービスによって実装されることになると思われます。

オブジェクトサービスとしては、次のような機能が考えられています。

- オブジェクト指向データベース
- トランザクション管理
- ディレクトリサービス
- ファイルサービス
- イベントサービス
- セキュリティ

1.6.3 共通ファシリティ

共通ファシリティでは多くのアプリケーションで利用することになるような機能を提供します。共通ファシリティの候補は次のリストです。

- クラスとオブジェクトとの分類機能やブラウザ機能
- リンク管理
- 再利用可能なユーザインターフェース（例:テキストエディタ）
- 印刷とスプール管理
- エラー報告
- ヘルプ・ファシリティ
- 電子メール・ファシリティ

共通ファシリティは次の項目を満足する必要があります。

- オブジェクトリクエストブローカを通じた通信
- OMGが採用を決定したファシリティを実装
- OMA準拠のオブジェクトインターフェースに対応

1.6.4 アプリケーションオブジェクト

アプリケーションオブジェクトとは従来のアプリケーションの概念に対応します。アプリケーションをOMAに準拠させることの価値は、複数のクラスをまとめたアプリケーションを構築できるという点です。

アプリケーションオブジェクトはOMAの機能的なレベルでは共通ファシリティ

と同じです。違いは共通ファシリティがOMGに標準として採用されている一般的な機能に対して、アプリケーションオブジェクトは個別のアプリケーションに特化していてOMGによって標準化されていないことです。

2. オブジェクトリクエストブローカの仕組み

2.1. 分散オブジェクト環境での処理形態

オブジェクトリクエストブローカを介してクライアントアプリケーションから処理を依頼するときのおおよその動作をみていきます。

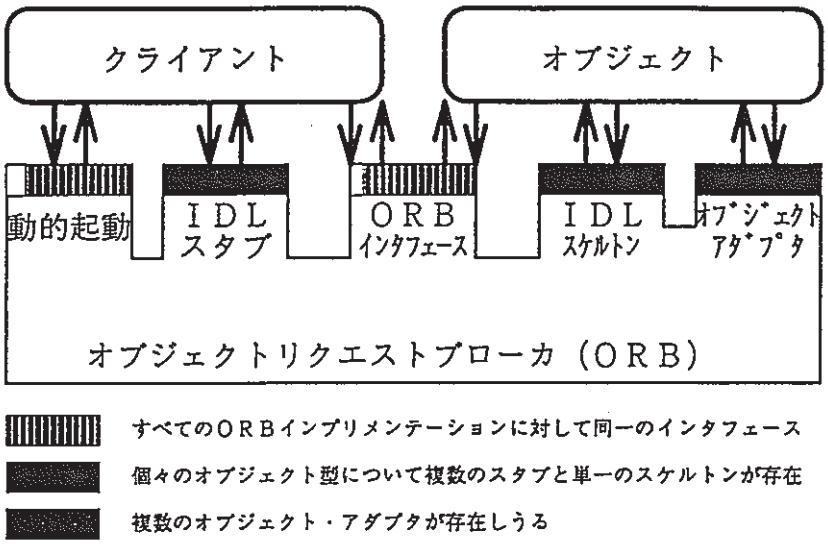


図 8 ORBインタフェースの構成

1. まず、クライアントが使用したいオブジェクトの「オブジェクト・リファレンス（オブジェクトの識別子）」と「インタフェース定義（オブジェクトに作用できる機能）」を知る
2. クライアントは、このオブジェクト・リファレンスとインタフェースのオペレーション（オブジェクトが受け付ける処理）をリクエストとしてオブジェクトリクエストブローカに渡す
3. オブジェクトリクエストブローカはこの識別子を使って最適なオブジェクトを選択し、リクエストを引き渡す
4. オブジェクトが作用し、その結果をオブジェクトリクエストブローカに戻す
5. オブジェクトリクエストブローカは、その結果をクライアントに返す

上記の流れをもとにして、さらに詳しくみていきます。

2.1.1. オブジェクト

オブジェクトとは、ユーザが使用している計算機およびネットワークで接続された遠隔の計算機上にあるさまざまな資源をさします。ＣＯＲＢＡの狙いは、このオブジェクトをクライアントから一様に取り扱える仕組みを提供することです。

このオブジェクトを抽象的に表現したものをインタフェースといいます。開発者は、属性（オブジェクトが保持するデータ）とオペレーション（オブジェクトが受け付ける処理）からインタフェースを定義します。

- 属性
  - 名前・タイプ・値
- オペレーション
  - 名前
  - パラメータ

クライアントからオブジェクトにアクセスするときは、このインタフェースのオペレーション（とその引数）を指定します。インタフェースの定義はオブジェクトの内部と外部の定義ですから、その実装とは無関係でなくてはなりません。

オペレーションを実装したものをメソッドといいます。オブジェクトブローカでは一つのオペレーションに複数のメソッドを対応させることができます。実行時にオブジェクトブローカが最適なメソッドを選択します。

2.1.2 クラス

クラスとはアプリケーションやデータをモデル化する場合に利用するオブジェクトです。オブジェクトは階層構造、継承関係、一般化/特殊化をサポートします。

2.1.3 オブジェクト・リファレンス

オブジェクトへの識別子のことをオブジェクト・リファレンスといいます。

しかし、クライアントからのリクエストを発行した時点ではオブジェクト・リファレンスがオブジェクトと関連付けられているとは限りません。その場合には、オブジェクトリクエストブローカが実行時にインタフェース・リポジトリとよばれるデータベースにアクセスして、呼び出すオブジェクトを決めることになります。

オブジェクト・リファレンスは、クライアント側で作成することができません。したがって、クライアントは以下のような方法によってオブジェクト・リファレ

ンスを入手します。

- ネーム・サービスからの入手
- ファイル渡し
- プログラム中へのハード・コーディング

クライアントからこのオブジェクト・リファレンスを使用してリクエストを発行するわけですが、これは定義されているクラスにメッセージを送ることになります。オブジェクトリクエストブローカは指定されたクラスとメッセージから、最適なサーバアプリケーションを選択します。選択されたサーバアプリケーションが、クライアントから依頼された処理を実行することになります。

2.1.4. 静的起動と動的起動

クライアントからリクエストを発行する方法には静的起動と動的起動の２種類があります。どちらの方法でオブジェクトリクエストブローカにリクエストを渡してもオブジェクトへは等価に伝えられます。

- 静的起動
  プログラムから直接リクエストを発行する方法です。Ｃ言語でいうと関数の呼び出しを行うことになります。この関数はインタフェース定義から作成され、これをスタブといいます。この方法ではコンパイル時にリクエスト発行メカニズムが作成されるために実行効率が良いといった特徴があります。
- 動的起動
  プログラム内でインタフェース定義にしたがってリクエストのデータ列の構造体を作成し、動的にリクエストを発行する方法です。このインタフェース情報は実行時にインタフェース・リポジトリから取り出すこともできます。この方法は、柔軟性がある反面、実行効率が落ちるといった特徴があります。

2.1.5. インプリメンテーション

インプリメンテーションはクライアントが要求したリクエストを実行します。

インプリメンテーション定義では、オブジェクトを生成し、そのオブジェクトが対応したサービスを提供するために必要な情報を定義します。

インプリメンテーションは、オペレーションを実行するための関数を提供することによってインタフェースをサポートします。ここで、実際に実行されるコード自体はメソッドになります。すなわち、メソッドはインプリメンテーションの一部であるといえます。

2.1.5.1. 活性化の方針

リクエストされたサービスを実行するためには、メソッドや状態を実行コンテキ

ストにコピーする必要があります。このプロセスを活性化（activation）といいます。

活性化の方針には以下の４種類があります。

表 3 活性化の方針

共用	サーバはインプリメンテーションに対して複数のオブジェクトをサポート
非共用	サーバはインプリメンテーションに対して一度に１つのオブジェクトだけをサポート
メソッドごとのサーバ	メソッドを起動するたびに新しいサーバが使用される
永続的	サーバは自動的には起動されない。起動後は共用方針と同じ

2.1.5.2 インプリメンテーションの型

インプリメンテーションの形態には以下のものがあります。

表 4 インプリメンテーションの型

プログラム	実行可能プログラム
静的ロード	メソッドはクライアントと同じプロセスにリンクされる
動的ロード	実行時に活性化される動的ライブラリの実行可能コード
スクリプト	ＯＳのスクリプトやコマンドプロシージャとして実現される
スタートアップのみ	インプリメンテーションが起動されるのみで、後の相互作用はなし

2.1.6 マーシャリング

分散オブジェクト環境では、各マシンがサポートするデータ型やエンディアンが異なる場合があります。この問題を解決するために、クライアント／サーバ間でマシンの差異を意識することなくデータのやりとりを行う機能が必要になります。このデータ変換の機能をマーシャリング機能といいます。

2.1.7 セキュリティ

クライアントマシンからサーバマシンのオブジェクトを操作するときには、セキュリティが大きな問題となります。このため、オブジェクトブローカではプロキシ（Ｐｒｏｘｙ）機能を提供しています。

このプロキシ機能により、ノードＡのユーザＡＬＰＨＡはノードＢのユーザＢＥＴＡの特権の範囲内でアクセスできる、といった使用法が可能になります。

2.2 オブジェクトブローカの構造

オブジェクトブローカは図９のような構造でシステムを構成します。

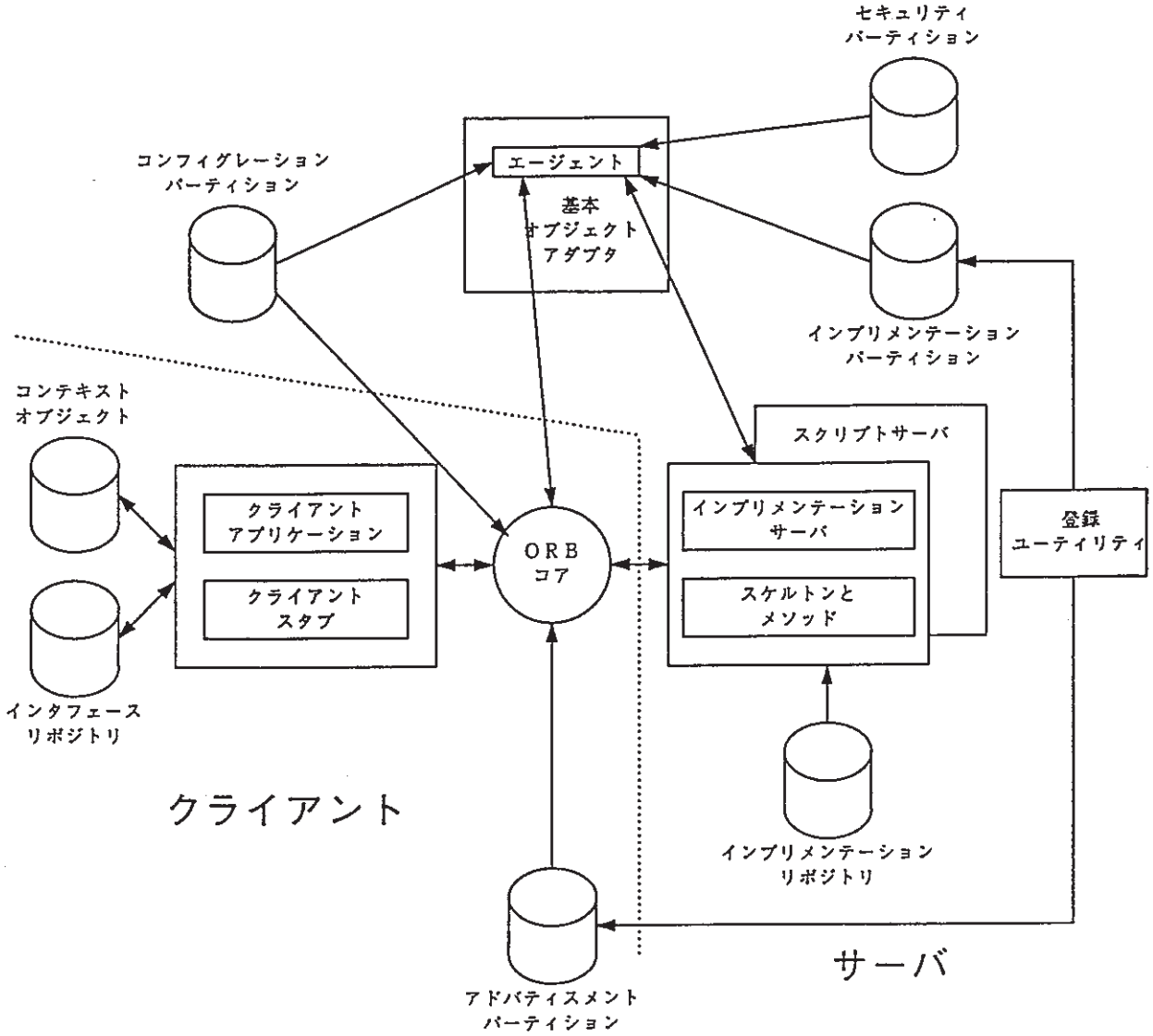


図 9 オブジェクトブローカの構造

2.3 サーバへのアクセスの仕組み

図 10ではクライアントアプリケーションがinterface_A と通して、サーバアプリケーションであるserver_P,server_Q,server_R の機能を利用する仕組みを紹介しています。

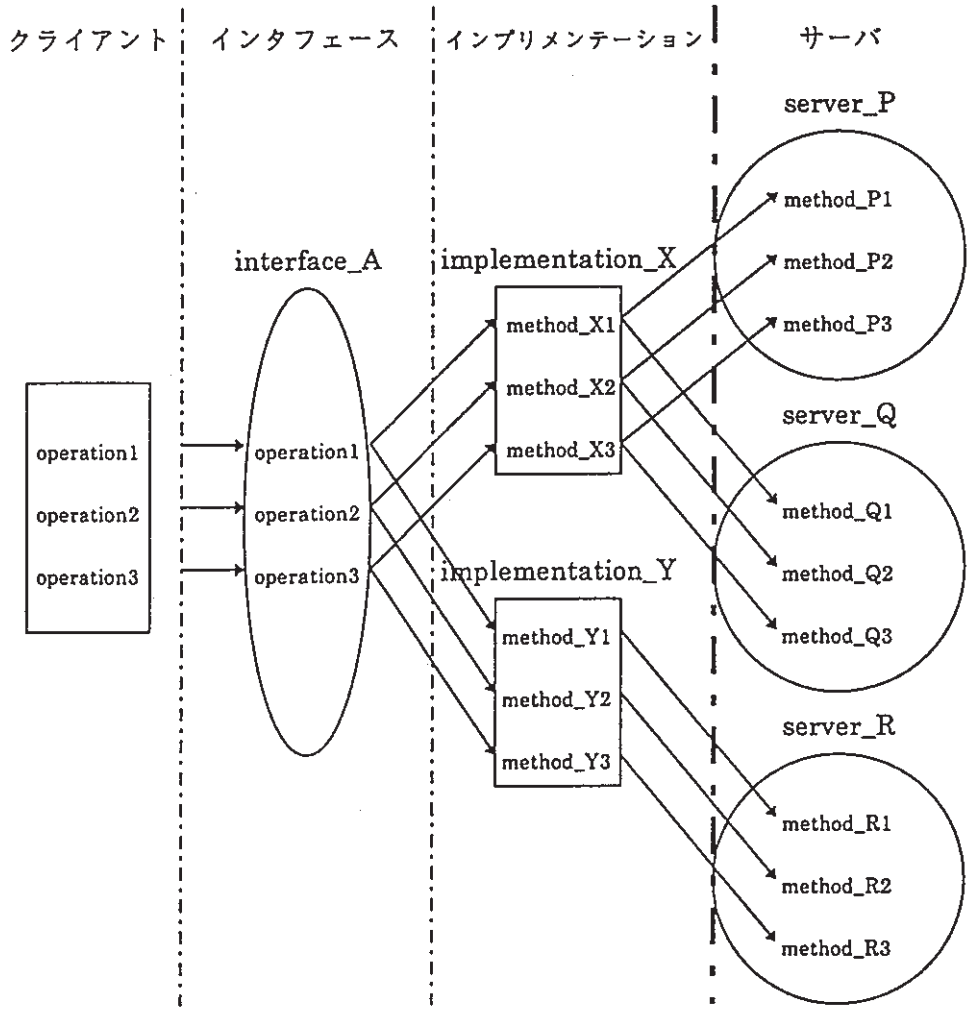


図 10 インタフェースとインプリメンテーションの関係

3. 開発手順

オブジェクトブローカでクライアント／サーバシステムを構築する場合、次の手順で開発することになります。

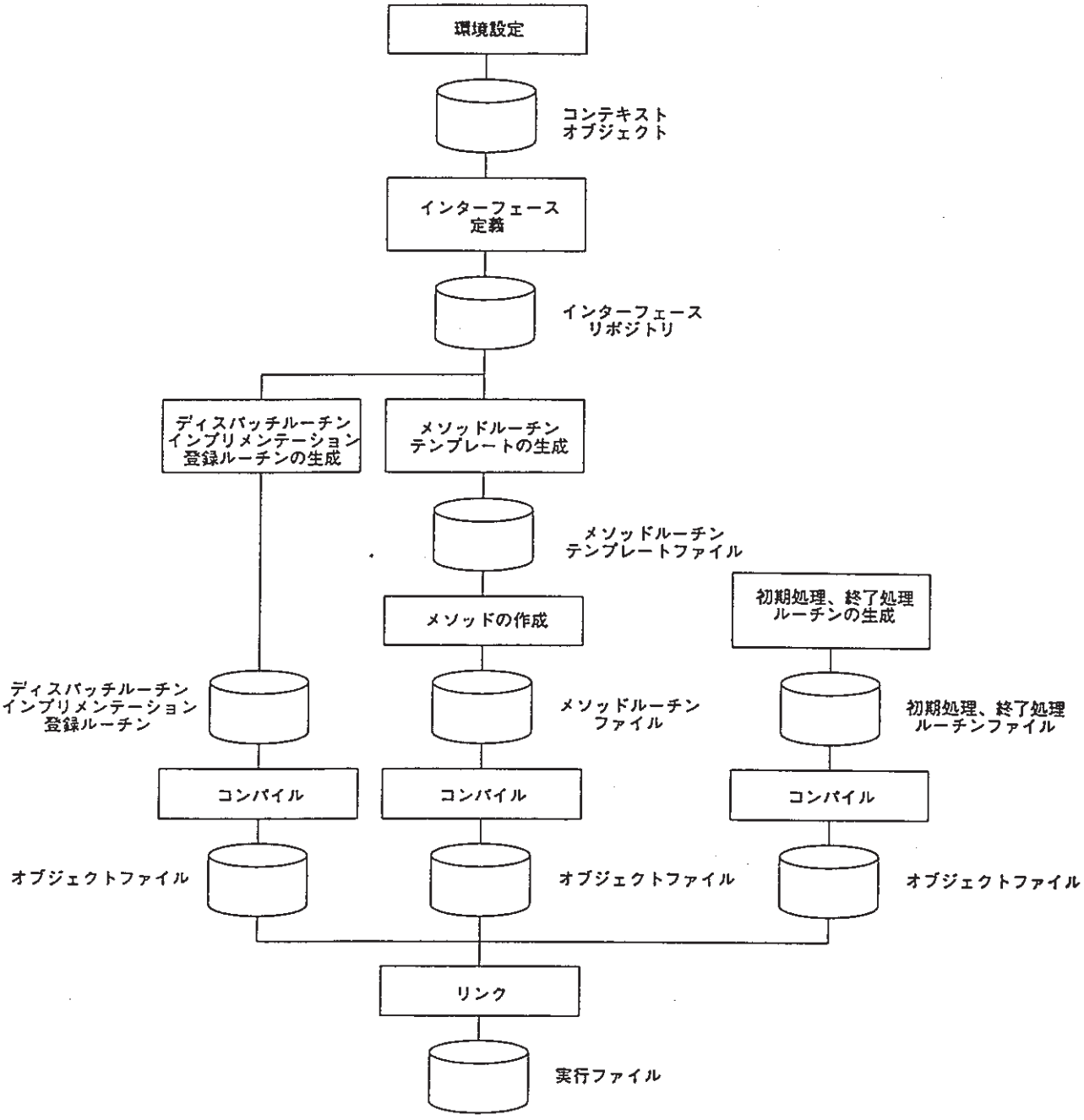


図 11 開発手順

1. 環境を設定します。この作業は主にシステム管理者が行ないます
   - ノードの登録
   - ユーザの登録

アプリケーションを定義するインターフェースリポジトリを作成します。インターフェースリポジトリでは、サーバアプリケーションのインターフェースとインプリメントを定義します。オブジェクトブローカを利用する場合には、クライアントアプリケーションはサーバアプリケーションを直接アクセスするのではなく、インターフェースリポジトリを参照してアクセスすることになります。この方法を利用することでクライアントアプリケーションとサーバアプリケーションが互いに依存しないことになります

2. クライアントを作成します。
3. サーバアプリケーションを作成します。作成の方法はサーバアプリケーションの種類によって異なります
   - スクリプトメソッドの場合
     スクリプトメソッドとはメソッドをスクリプトで実装する形式です。既存のアプリケーションを統合する場合などで利用します。メソッドに対応するスクリプトを作成してください
   - メソッドサーバの場合
     - インターフェースリポジトリからディスパッチルーチンとインプリメンテーション登録ルーチンを生成します。

     B. ディスパッチルーチンはクライアントアプリケーションから送られてきたオペレーションを、それぞれのオペレーションを処理するメソッドルーチンに振り分けます。インプリメンテーション登録ルーチンはインプリメンテーションをエージェントに登録します

     C. インターフェースリポジトリからメソッドルーチンのテンプレートを生成します。

     D. 生成した部分以外のサーバアプリケーションを作成します。サーバアプリケーションは次の処理を行なう必要があります。
        - インプリメンテーション登録ルーチンの呼び出し
        - メインループの呼び出し
        - 終了処理

3.1. システム管理

オブジェクトブローカを利用するためには、動作環境を設定する必要があります。

- トランスポート
- インプリメンテーション
- エージェント
- 登録簿
- セキュリティ
- 他のノードへのサービスの提供

3.1.1. コンテキストオブジェクト

オブジェクトブローカでは、一般的な環境設定にはコンテキストオブジェクトを利用します。コンテキストオブジェクトを利用するためには、コンテキストオブジェクト言語(Context Object Language(COL))を利用します。

3.1.1.1. コンテキストオブジェクトの構造

コンテキストオブジェクトは複数のテーブルから構成されます。それぞれのテーブルで、任意の数のプロパティを定義することができます。テーブルは、参照するアプリケーションや設定するプロパティのカテゴリなどで分割することになります。プロパティには文字列のみが設定できます。任意の数の値が設定できます。

```
CONTEXTOBJECT
        TABLE table_A
                attribute_A1 = value_A1X,value_A1Y;
                attribute_A2 = value_A2;
        end table

        table table_B
                attribute_B1 = value_B1;
                attribute_B2 = value_B2:
        end table
end contextobject
```

デフォルトで参照されるテーブルとしてOBB_DEFAULT_TABLEが用意されています。このテーブルに設定することになるプロパティとして、OBB_REPOSITORY と OBB_DEFAULT_NODES があります。

OBB_REPOSITORY にはインターフェースリポジトリ、インプリメンテーションリポジトリのファイル名を指定します。OBB_DEFAULT_NODES にはサーバのデフォルトノードを指定します。

プロパティ値としてOBB_ENVIRONMENT_VARIABLE を設定すると、プロパティ名と同じ名前の環境変数(UNIX の場合)あるいは論理名(VMS の場合)を参照

することになります。

3.1.1.2 コンテキストオブジェクトのレベル

コンテキストオブジェクトには、ユーザ・グループ・システムの3レベルがあります。それぞれのレベルのコンテキストオブジェクトには、ファイルのディレクトリおよびファイル名で参照します。

- VMS の場合
  - ユーザレベル　カレントディレクトリのOBB_USER_CONTEXT.CO
  - グループレベル　カレントディレクトリのOBB_GROUP_CONTEXT.CO
  - システムレベル　OBB$LIBRARY:OBB_SYSTEM_CONTEXT.CO
- UNIX の場合
  - ユーザレベル　カレントディレクトリのOBB_USER_CONTEXT
  - グループレベル　カレントディレクトリのOBB_GROUP_CONTEXT
  - システムレベル　/var/adm/ObjectBroker/OBB_SYSTEM_CONTEXT
- Windows の場合
  - ユーザレベル　カレントディレクトリのOBBUSRC.CO
  - グループレベル　カレントディレクトリのOBBGRPC.CO
  - システムレベル　C:¥OBROKER¥REPOS¥OBBSYSC.CO
- WindowsNT の場合
  - ユーザレベル　カレントディレクトリのOBBUSRC.CO
  - グループレベル　カレントディレクトリのOBBGRPC.CO
  - システムレベル　C:¥WIN32APPL¥OBROKER¥REPOS¥OBBSYSC.CO

ユーザコンテキストオブジェクトとグループコンテキストオブジェクトを、環境変数あるいは論理名でデフォルトから変更することができます。

表 5 デフォルトコンテキストオブジェクト

レベル	UNIX およびWindowsNT の環境変数	VMS での論理名
ユーザ	OBB_USER_CONTEXT	OBB_USER_CONTEXT
グループ	OBB_GROUP_CONTEXT	OBB_GROUP_CONTEXT

コンテキストオブジェクトを参照する場合、フェイルオーバー機能を利用することができます。フェイルオーバー機能とは、ユーザ・グループ・システムレベルの順に該当する属性が存在するレベルまで、検索する機能です。

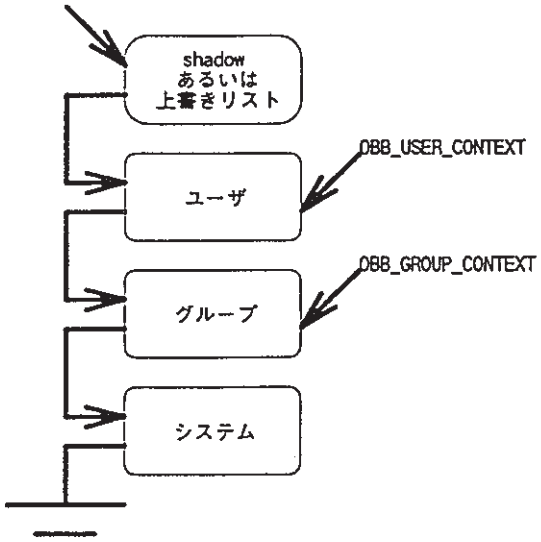


図 12 コンテキストオブジェクトのフェイルオーバー機能

3.1.1.3 コンテキストオブジェクトのロード

コンテキストオブジェクトを利用するには、定義ファイルを次のコマンドで変換する必要があります。

OpenVMS:
LOAD CONTEXT_OBJECT コンテキストオブジェクト定義ファイル
UNIXおよびWindowsNT:
obbldctx コンテキストオブジェクト定義ファイル

表 6 コンテキストオブジェクトのロードコマンドのオプション

	OpenVMS	UNIX およびWindowsNT	
コンテキストオブジェクト定義	/USER	-U	設定するコンテキストオブジェクトのレベルを指定します
	/GROUP	-G	
	SYSTEM	-S	
コンテキストオブジェクトファイル名	/CONTEXT_OBJECT	-C	変換するコンテキストオブジェクトファイルを指定します
新規作成	/CREATE	-c	コンテキストオブジェクトを新規に作成します

3.1.1.4 コンテキストオブジェクトの参照

コンテキストオブジェクトを次のコマンドで参照することができます。

OpenVMS:
SHOW CONTEXT_OBJECT
UNIXおよびWindowsNT:
obbshctx

表 7 コンテキストオブジェクトの表示コマンドのオプション

	OpenVMS	UNIX およびWindowsNT	
コンテキストオブジェクト定義	/USER /GROUP SYSTEM	-U -G -S	参照するコンテキストオブジェクトのレベルを指定します
コンテキストオブジェクトファイル名	/CONTEXT_OBJECT	-C	参照するコンテキストオブジェクトファイルを指定します
テーブル名	/TABLE	-T	参照するコンテキストオブジェクトのテーブル名を指定します
環境変数あるいは論理名の参照	/EXPAND_VARIABLES	-e	参照している環境変数あるいは論理名を表示します

3.1.2 環境設定

環境として、以下の項目をセットできます。

- ネットワークプロトコル
  - TCP/IP
  - DECnet
- セキュリティレベル
  - 有効
  - 無効
- ログ記録レベル
  - スタートアップイベント
  - オブジェクト活性化イベント
  - なし
- リポジトリオブジェクトのキャッシュサイズ
- 有効なサーバの最大数
- エージェントの起動時に、自動的に起動するインプリメンテーション

以上の項目をまとめた環境に名前をつけて、複数の環境を切り替えることができます。

3.1.2.1. 環境の設定

環境を作成するには次のコマンドを利用します。作成された環境はデフォルトの環境がコピーされています。

OpenVMS:
CREATE CONFIGURE 環境名
UNIXおよびWindowsNT:
obbcrcfg 環境名

次のコマンドを利用して環境をコピーすることができます。

OpenVMS:
COPY CONFIGURE 元の環境名 新しい環境名
UNIXおよびWindowsNT:
obbcpcfg 元の環境名 新しい環境名

環境を切り替えるには次のコマンドを利用します。

OpenVMS:
SET CONFIGURE 環境名
UNIXおよびWindowsNT:
obbstcfg 環境名

それぞれの環境を変更するには次のコマンドを利用します。

OpenVMS:
MODIFY CONFIGURE 環境名
UNIXおよびWindowsNT:
obbmdcfg 環境名

表 8 環境設定の変更コマンドのオプション

	OpenVMS	UNIX および WindowsNT	
認証パッケージ	/AUTHENTICATION_PROVIDER	-a	使用する認証パッケージ名を指定します
認証機能	/STATE	-Se -Sd	認証機能の有効あるいは無効を指定します
ネットワーク機能	/NETWORK_PROVIDER	-n	使用するネットワークパッケージを指定します
ログ記録レベル	/EVENT_LOGGING	-en -ep -eo -ea	ログを記録するレベルを指定します
リポジトリオブジェクトのキャッシュサイズ	/REPOSIOTRY=CHACH_SIZE	-c	キャッシングするリポジトリオブジェクトの数を指定します
有効なサーバの最大数	/MAX_ACTIVE_SERVERS	-m	自動的に起動する起動するサーバの最大数を指定します
自動起動	/AUTO_START	-s	エージェントの起動時に自動的に起動するサーバを指定します

環境を削除するには次のコマンドを利用します。

OpenVMS:
DELETE CONFIGURE 環境名
UNIXおよびWindowsNT:
obbdlcfg 環境名

3.1.2.2 環境の表示

環境を表示するには次のコマンドを利用します。

OpenVMS:
SHOW CONFIGURE 環境名
UNIXおよびWindowsNT:
obbshcfg 環境名

次のコマンドを利用すると登録されているすべての環境を表示することができます。

OpenVMS:
SHOW CONFIGURE/ALL
UNIXおよびWindowsNT:
obbshcf -a

3.1.3 セキュリティ

オブジェクトブローカでは2つのレベルでセキュリティを管理しています。ひとつめがノードレベルでのセキュリティの管理で、もう一つはインプリメントレベルでの管理です。

ノードレベルでのセキュリティの管理では、リモートノードでの特定のユーザを自ノードのどのユーザとして扱うかを指定します。指定したノードのユーザからリクエストが送られた結果としてサーバを起動する場合には、設定してあるユーザのプロセスとして起動します。

インプリメンテーションレベルでのセキュリティ管理では、各インプリメンテーションを実行可能であるユーザを定義します。

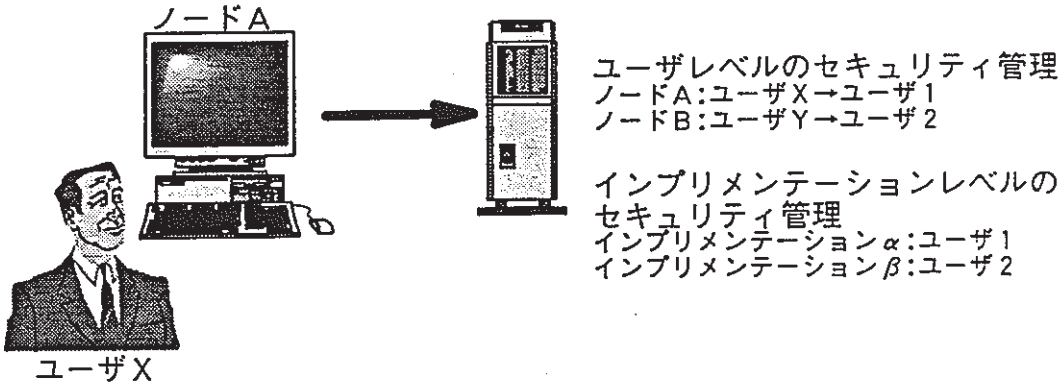


図 13 セキュリティの設定

3.1.3.1. セキュリティの設定

次のコマンドを利用して、リモートノードのユーザを登録することができます。

OpenVMS:
ADD PROXY/REMOTE_USER=(USER=ユーザ名,HOST=ノード名) ユーザ名
UNIXおよびWindowsNT:
obbadpxy -u ユーザ名 -h ノード名 ユーザ名

登録したユーザは次のコマンドで削除することができます。

OpenVMS:
REMOVE PROXY/REMOTE_USER=(USER=ユーザ名,HOST=ノード名) ユーザ名
UNIXおよびWindowsNT:
obbrmpxy -u ユーザ名 -h ノード名 ユーザ名

次のコマンドでユーザが利用可能なインプリメンテーションやメソッドを指定することができます。

OpenVMS:
GRANT AUTHORIZATION ユーザ名
UNIXおよびWindowsNT:
obbgrath ユーザ名

表 9 コンテキストオブジェクトの表示コマンドのオプション

	OpenVMS	UNIX および WindowsNT	
インプリメンテーション名	/IMPLEMENTATION	-i	ユーザがアクセスできるインプリメンテーションを指定します
メソッド名	/METHOD	-m	ユーザがアクセスできるメソッド名を指定します
アクセスレベル	/ADMIN_METHODS	-a	ユーザがアクセスできるレベルを指定します

次のコマンドでユーザが利用可能なインプリメンテーションやメソッドを削除することができます。

OpenVMS:
REVOKE AUTHORIZATION ユーザ名
UNIXおよびWindowsNT:
obbrvath ユーザ名

表 10 コンテキストオブジェクトの表示コマンドのオプション

	OpenVMS	UNIX および WindowsNT	
インプリメンテーション名	/IMPLEMENTATION	-i	アクセスを不可能にするインプリメンテーションを指定します
メソッド名	/METHOD	-m	アクセスを不可能にするメソッド名を指定します

3.1.3.2 セキュリティの参照

次のコマンドを利用して、登録したリモートノードのユーザを参照することができます。

OpenVMS:
SHOW PROXY/REMOTE_USER=(USER=ユーザ名,HOST=ノード名) ユーザ名
UNIXおよびWindowsNT:
obbshpxy -u ユーザ名 -h ノード名 ユーザ名

次のコマンドでユーザが利用可能なインプリメンテーションやメソッドを参照することができます。

OpenVMS:
SHOW AUTHORIZATION ユーザ名
UNIXおよびWindowsNT:
obbshath ユーザ名

3.2 オブジェクト定義

オブジェクトブローカでクライアント／サーバシステムを構築する場合、サーバ

アプリケーションの論理的な定義としてインターフェース、物理的な定義としてインプリメントを記述します。

3.3 インターフェース定義

インターフェースの定義にはIDL(Interface Definition Language:インターフェース定義言語)を利用します。IDLの仕様はOMG/CORBAの規格として定義されています。IDLでインターフェースを定義する場合には、次の手順で行ないます。

1. module文を利用して比較的関係のあるインターフェースを、モジュールとしてグループ化します。モジュール名はクライアントアプリケーションやサーバアプリケーションが、インターフェースを参照する名前の一部に組み込まれます。
2. interface文を利用してオブジェクトをinterfaceとして定義します。オブジェクト名もクライアントアプリケーションやサーバアプリケーションが、インターフェースを参照する名前の一部になります。
3. インターフェースにオペレーションを定義します。オペレーションには次の項目を定義します。
   - オペレーション名
   - オペレーションの型
   - パラメータ名
   - パラメータの方向
   - パラメータの型
4. インプリメンテーションが返すエラー情報として、ユーザ例外を定義します。ユーザ例外の定義にはexception文を利用します。
5. インターフェース属性を定義します。インターフェース属性はオブジェクトの状態を表わす変数などに利用します。

3.3.1. CORBA IDLの例

オブジェクトブローカのキットに含まれているサンプルのIDLです。

```
module BNK1
// We wanted to name this module BANK, which would give the stub calls nice
// names like BANK_Account_Deposit, BANK_Account_Withdraw, etc. However,
// we decided it was better to be consistent with the naming conventions we
// supplied in the writeup, and make the module name the same as the file name.
// Life is full of tough decisions.
{
 // Some constants used in the example (Note that putting them here
 // eliminates the need for a separate user-written header file to share
 // this information between client and server sources)
 const short      MAXLINELEN=   512;
 const short      BFALSE=       0;
 const short      BTRUE=        1;
 const string STR_OBJ_FILNAM = "BNK1IO.DAT";
 // This is an enumerated list of recognized account types.
 enum ACCT_TYPES   {Savings,Checking};
```

```
// An enumerated list of application-specific errors, accompanied by
// explanatory text strings.
enum ERR_TYPES    {
      ErrCustNonex,    // no such customer id
      ErrAcctNonex,    // customer does not have that acct type
      ErrAcctInsuf     // acct balance insufficient for request
};
// Define the complex data types common to the interfaces
// in this module.
typedef short CustomerId;   // We use this in a lot of places; defining
                            // it here just makes it easier to change from
                            // short to long in the future.
// this structure allows us to associate a customer with various accounts
// (checking, savings, etc)
struct AccountId {
      CustomerId CustomerNo;
      short      AccountType;
};
// everything we need to know about a customer...
struct CustomerData {
      CustomerId CustomerNo;
      string     Name;
      string     Surname;
};
// ...and everything we need to know about an account
struct AccountData {
      AccountId  AccountNo;
      float      Balance;
};
// We define one user (ie, application-specific) exception structure. All
// BNK1 user exceptions return an enumerated value that indicates the
// specific reason for the exception. IF the exception is "insufficient
// funds", the exception also returns the current account balance (we do
// that because when a method raises an exception, the "out" (returned)
// method arguments are undefined).
exception UserExcep {
      ERR_TYPES Reason;
      float Balance;
};
// The Account interface, with its three operations
interface Account {
      void Deposit(     in    AccountId  AccountNo,
                        in    float      Amount,
                        out   float      CurrentBalance)
            raises      (UserExcep);
      void Withdraw(    in    AccountId  AccountNo,
                        in    float      Amount,
                        out   float  CurrentBalance)
            raises      (UserExcep);
      void Balance(     in    AccountId  AccountNo,
                        out   float  CurrentBalance)
            raises      (UserExcep);
};    // end of interface Account
// The Customer interface, with one operation
interface Customer {
      void  Validate(   in    CustomerId CustomerNo,
                        out   stringName,
                        out   stringSurname)
            raises      (UserExcep);
};    //end of interface Customer
};    //end of module BNK1
```

3.3.2 モジュールの定義

module ステートメントでモジュールを定義します。複数のインターフェースをモ

ジュールとしてまとめることができます。異なるモジュールで定義されているインターフェースは、同じ名前で定義することもできます。

3.3.3 インターフェースの定義

interface ステートメントでインターフェースを定義します。interface 定義では、他のインターフェースの定義を継承することができます。

interface 新しいインターフェース名 : 継承されるインターフェース名

3.3.4. CORBAデータ型の指定

IDL で定義できるデータ型はOMG/CORBA で定義されています。IDL のデータ型は 2 種類あります。

- 単純データ型
- 複合データ型

IDL のデータ型はC 言語と Visual Basic でサポートされています。

3.3.4.1. 単純データ型

単純データ型は要素が一つのデータ型です。

表 11 単純データ型

単純データ型	IDLデータ型宣言	C言語へのマッピング	C言語での型定義
整数	long short unsigned long unsigned short	long あるいは int short unsigned long unsigned short	CORBA_long CORBA_short CORBA_unsigned_long CORBA_unsigned_short
文字	char	char	CORBA_char
論理値	Boolean	char	CORBA_Boolean
オブジェクトリファレンス	Object	void*	CORBA_Object
8 ビットデータ	octet	unsigned char	CORBA_octet
浮動小数点	float double	float double	CORBA_float CORBA_double

3.3.4.2 複合データ型

複合データ型は、要素の数が複数であるデータ型です。C 言語マッピングの例ではすべてTEST モジュール内で定義していることを仮定しています。

表 12 複合データ型

複合データ型	例	C言語へのマッピング
文字列	typedef string<8> Password;	typedef CORBA_string TEST_Password
データ構造体	typdef struct AccountDataStruct { Account AccountID; float Balance; } AccountData;	typedef struct TEST_AccountDataStruct { TEST_Account AccountID; CORBA_float Balance; } TEST_AccountData;
列挙体	typedef enum AcctTypeEnum{ Savings, Checking }AcctType;	#define TEST_Savings 1 #define TEST_Checkings 2
配列	typedef long Arraylong[10];	typedef CORBA_long TEST_ArrayLong[10];
サイズ指定のあるシーケンス	typdef sequence <Object,25> ObjectSeq;	typedef struct { unsigned long _maximum; unsigned long _length: CORBA_Object * _buffer; } TEST_ObjectSeq:
サイズ指定のないシーケンス	typedef sequence <Object> ObjectSeq;	typedef struct { unsigned long _maximum; unsigned long _length: CORBA_Object * _buffer; } TEST_ObjectSeq:
汎用体	typedef any myany;	typedef struct any { CORBA_TypeCode _type; void* _value; } TEST_myany;

共用体	typedef union myunion switch (long){
	case 1: long along;
	case 2: float afloat;
	} Aunion;
		typedef struct TEST_myunion {
		CORBA_long _d;
		union {
		CORBA_long along;
		CORBA_float afloat;
		} _u;
		} TEST_Aunion;

3.3.5 パラメータの方向

パラメータの方向とはパラメータの値が渡される方向を意味します。

表 13 パラメータの方向

in	クライアントからサーバにデータが渡されます
out	サーバからクライアントにデータが渡されます
Inout	クライアントからサーバとサーバからクライアントの両方向でデータが渡されます

3.3.6 例外

例外を利用してオペレーションが何らかの理由で正常終了しなかった場合に、原因をクライアントに返します。エラー情報を例外という形態にまとめることで、オペレーションの定義はより正確になります。例外の定義は次のようにおこないます。

exception　例外名　　メンバ宣言;

表 14　例外定義

例外の宣言	指定内容
exception	例外定義の開始
例外名	例外を表わす名前
メンバ宣言	例外情報を渡すデータ構造

3.3.7. オペレーション

オペレーションの定義は次のようにおこないます。

[oneway]{void|オペレーション型} オペレーション名([{in|out|inout} データ型 引数名][,{in|out|inout} データ型 引数名,…][raises (例外名[,例外名,…])][context("コンテキスト属性名"[,…])

表 15 オペレーション定義

IDL ステートメント	指定内容
oneway	出力引数と例外がなく、型がvoid であるオペレーションにのみ、この指定は有効です
void|オペレーションの型	オペレーションが返すデータの型を指定します。データを返さない場合には void を指定します
オペレーション名	オペレーションの名前を指定します
([in|out|inout データ型 引数名[,in|out|inout データ型 引数名…]])	オペレーションの引き数を指定します
raises(例外名[,例外名,…])	このオペレーションで返される例外を記述します。ユーザ例外のみを指定します。ここで指定しないとユーザ定義の例外を使用することができませんが、システム例外を使用することはできます
context("コンテキスト属性"[,…])	メソッドの実行で使用するコンテキストオブジェクトのプロパティを指定します。

3.3.8. インターフェース属性

インターフェース属性の定義は次のようにおこないます。

[readonly] attribute データ型 属性名

オブジェクトの状態を保持するデータを定義します。クライアントは属性の値を設定したり、設定した値を取り出すことができます。

属性を指定することは二つのオペレーションを宣言することと同じです。一つは属性の値を取り出すオペレーションで、もう一つは属性の値を設定するオペレーションです。

属性には次の特性があります。

- 任意のIDL でサポートするデータ型、あるいはユーザ定義のデータ型を指定することができる
- 修正可能あるいは読み出し専用に宣言することができます。読み出し専用と指定すると値の読み取りオペレーションのみが生成されます
- ユーザ例外を利用することはできません
- コンテキスト情報を利用することはできません

- パラメータの数が決められています。読み取りオペレーションはパラメータはありません。設定オペレーションは値を設定するパラメーターつです。

3.3.9. インターフェース定義にたいするUUIDの設定

インターフェース定義ファイルはインターフェースリポジトリにロードすることができます。インターフェースリポジトリを利用すると、複数のファイルに分割されたインターフェース定義を扱いやすくなります。インターフェースリポジトリファイルはコンテキストオブジェクトで指定すると便利です。

インターフェースリポジトリにロードする前に、インターフェース定義に記述されている各オブジェクトにUUID(Universal Unique Identifier)を割り当てる必要があります。UUID はインターフェースやインプリメント、オペレーション、メソッド、タイプコードなどのオブジェクトに割り当てます。次のコマンドを利用すると、IDL ファイルのインターフェース定義にUUID を割り当てることができます。

OpenVMS:
GENERATE UNIQUE_IDNETIFIER /FILE=インターフェース定義ファイル名 出力ファイル名
UNIXと WindowsNT:
obbgen -u -f インターフェース定義ファイル名 [出力ファイル名]

UUID を利用することで、複数のシステムで定義ファイルを分散しても、オブジェクトを一意に認識することができます。

UUID を定義ファイルで指定していないと、定義ファイルをリポジトリにロードするとき、あるいは定義ファイルからスタブファイルを生成するときに、オブジェクトブローカが自動的にオブジェクトにUUID を割り当てます。この場合、定義ファイルをロードしたりスタブファイルを生成するたびに、異なるUUID を割り当てることになります。したがって同じオブジェクトであってもシステムごとに異なるUUID を割り当てられることになります。オブジェクトブローカは書くオブジェクトをUUID で識別しているため、これらの異なるUUID が設定されているオブジェクトを同じオブジェクトとは認識しません。

3.3.10. インターフェース定義のロード

UUID を定義したら、次のコマンドを利用してインターフェース定義ファイルを、インターフェースリポジトリにロードすることができます。

OpenVMS:
LOAD REPOSITORY_OBJECT インターフェース定義ファイル
UNIXおよびWindowsNT:
obbldrep インターフェース定義ファイル

表 16 インターフェース定義ファイルのロードコマンドのオプション

	OpenVMS	UNIX およびWindowsNT	
リポジトリ名	/REPOSITORY	-r	ロードするリポジトリ名を指定します
新規作成	/CREATE	-c	リポジトリファイルを新規に作成します
マクロ定義	/DEFINE	-d	インターフェース定義ファイルを変換する場合に参照するマクロ定義を指定します
参照ディレクトリ	/INCLUDE_DIRECTORY	-I	#include 文で参照するファイルを検索するディレクトリを指定します

次のコマンドを利用して、インターフェースリポジトリからリポジトリオブジェクトを削除できます。

OpenVMS:
DELETE REPOSITORY_OBJECT オブジェクト名
UNIXおよびWindowsNT:
obbdel オブジェクト名

表 17 リポジトリオブジェクトの削除コマンドのオプション

	OpenVMS	UNIX およびWindowsNT	
リポジトリ名	/REPOSITORY	-r	ロードするリポジトリ名を指定します
識別子	/IDENTIFIER	-i	削除するオブジェクトの識別子を指定します

3.3.11. インターフェース定義の表示

次のコマンドを利用してインターフェースリポジトリの内容を表示することができます。

OpenVMS:
SHOW REPOSITORY_OBJECT オブジェクト名
UNIXおよびWindowsNT:
obbidrep オブジェクト名

表 18 インターフェースリポジトリの表示コマンドのオプション

	OpenVMS	UNIX およびWindowsNT	
リポジトリ名	/REPOSITORY	-r	ロードするリポジトリ名を指定します
サブクラスの表示	/INHERITANCE /NOINHERITANCE	-n	指定したオブジェクトのサブクラスを表示するかどうかを指定します
オブジェクトのタイプ	/TYPE	-t	表示するオブジェクトのタイプを指定します

3.4. インプリメント定義

インプリメンテーションとはオペレーションを実際に処理するための関数の集合

です。関数自体はメソッドと呼ばれます。メソッドは関数以外にもスクリプトで実装することもできます。

インプリメンテーションは、IML(Implementation Mapping Language:インプリメンテーション定義言語)で定義します。IMLファイルのテンプレートを、次のコマンドで生成することができます。

OpenVMS:

```
GENERATE INTERFACE/DEFAULT=IMPLEMENTAION=IMLファイル名 インターフェース名
COMPILE INTERFACE/DEFAULT=IMPLEMENTAION=IMLファイル名 -
インターフェース定義ファイル名
```

UNIXおよびWindowsNT:

```
obbgen -I IMLファイル名 インターフェース名
obbcomp -I IMLファイル名 インターフェース定義ファイル名
```

3.4.1. IMLの例

```
// ************************************************************************
// Created Tue Jun 21 08:42:27 1994 by OBB T2.5-05 (COMPILE/GENERATE)
// ************************************************************************
//
// OBB Default IML file
//
// Use this file as a base for your additional IML definitions.
//
implementation AccountImpl
{
        activation_type ( program );
        activation_string ( "<tba> - command to run implementation" );
        implementation_identifier ( "67d5a95cd9da.02.10.20.30.12.00.00.00" );
        registration_attribute string implementationName = "AccountImpl";
        method_dispatcher_routine AccountImpl__dispatch ();
        registration_routine AccountImpl__register ();
        Deposit ( )
                implements ( BNK1::Account::Deposit )
                invoke_builtin ( "AccountImpl_Deposit" )
                ;
        Withdraw ( )
                implements ( BNK1::Account::Withdraw )
                invoke_builtin ( "AccountImpl_Withdraw" )
                ;
        Balance ( )
                implements ( BNK1::Account::Balance )
                invoke_builtin ( "AccountImpl_Balance" )
                ;
};
implementation CustomerImpl
{
        activation_type ( program );
        activation_string ( "<tba> - command to run implementation" );
        implementation_identifier ( "67d5a95ce151.02.10.20.30.12.00.00.00" );
        registration_attribute string implementationName = "CustomerImpl";
        method_dispatcher_routine CustomerImpl__dispatch ();
        registration_routine CustomerImpl__register ();
        Validate ( )
                implements ( BNK1::Customer::Validate )
                invoke_builtin ( "CustomerImpl_Validate" )
                ;
};
```

3.4.2 インプリメンテーションの指定

IML ファイルではIML を利用してインプリメンテーションを定義します。ここではIML の概略を説明します。

3.4.2.1. implementationステートメント

implementation ステートメントではインプリメンテーションの名前を指定します。implementation ステートメントでは他のインプリメンテーションの定義を参照して継承することができます。

Implementation 新しいインプリメンテーション名 : 継承されるインプリメンテーション名

新しいインプリメンテーションはメソッドやその他の指定すべてを継承します。継承したメソッドの定義を変更することはできませんが、その他の指定を変更することは可能です。

3.4.2.2 activation_policyステートメント

activation_policy()ステートメントでは、オブジェクトを処理するインプリメンテーションがサポートする方針を指定します。activation_policy()ステートメントでは次の値を指定できます。

表 19 activation_policy()ステートメントの値

	説明	使用する状況
shared	インプリメンテーションは複数のオブジェクトをサポートします	複数のオブジェクトを一つのインプリメンテーションと関連付ける場合
unshared	インプリメンテーションとオブジェクトは一対一の関係になります	オブジェクトが他のオブジェクトから独立している場合
server_per_method	メソッドを起動するたびに新しいサーバを起動します	メソッド間でオブジェクトの状態を保存しない場合 この方針はスクリプトサーバのみが利用します
persistant	サーバを自動的には起動しません。起動されている場合には、'shared と同じです	サーバプロセスの数を限定したいなどの場合

3.4.2.3 activation_typeステートメント

activation_type()ステートメントでは選択されたインプリメンテーションを起動する方法を指定します。

activation_type()ステートメントでは次の値を指定します。

表 20 activation_type()ステートメントの値

	説明	使用する状況
program	サーバは一つ以上のインプリメンテーションをリンクした実行ファイルになります	インプリメンテーションがメソッドにアクセスする前に、活性化する必要がある場合
static_load	メソッドはクライアントと同じプロセスで処理されます	インプリメンテーションを分散させるより性能を優先する場合
dynamic_load	サーバは実行時に活性化する動的ライブラリになります	必要な場合のみにロードされるライブラリで実装した場合 dynamic_library()ステートメントでインプリメンテーションやメソッドのあるライブラリを指定します
script	サーバはコマンドレベルのスクリプトやコマンドプロシージャになります	既存のアプリケーションなどをカプセル化する場合
startup_only	インプリメンテーショを起動した後で、クライアントからの相互作用ができません	サーバとして、単純にアプリケーションを起動すればいい場合

3.4.2.4. activation_stringステートメント, dynamic_libraryステートメント

activation_string()ステートメントとdynamic_library()ステートメントでは、sactivation_type()に program,startup_only,dynamic_load を指定したインプリメンテーションを、活性化する方法を指定します。

activation_string()ステートメントやdynamic_library ステートメントでは次の値を指定します。

表 21 activation_type()ステートメントの値

activation_type()の値	説明	指定内容
program	プログラムインプリメンテーション含むサーバをオブジェクトブローカが活性化します	activation_string()ステートメントに、オペレーティングシステムでサーバを起動するコマンドを指定します
static	サーバとインプリメンテーションは事前に起動しているはずです	activation_string()と dynamic_library()を使用することはできません
dynamic_load	メソッドが含まれている動的ライブラリをオブジェクトブローカがロードします	ロードするライブラリの名前をdynamic_library()ステートメントで指定します 指定されたライブラリはOBB_Server_Initialize()を呼び出せるように作成する必要があります
script		activation_string()と dynamic_library()を使用することはできません
startup_only	オブジェクトブローカがサーバを活性化します	activation_string()ステートメントに、オペレーティングシステムでサーバを起動するコマンドを指定します

3.4.2.5. activation_contextステートメント

activation_context()ステートメントではインプリメンテーションが参照するコンテキストオブジェクトの属性を指定します。ここで指定した属性はサーバプロセ

スの起動時に設定されます。

activation_context()ステートメントは次の形式で設定します。

表 22 activation_context()ステートメントの値

指定形式	説明	例
コンテキスト属性名	コンテキストオブジェクトの属性が、インプリメンテーションのコンテキストオブジェクトにコピーされます	activation_context("DISPLAY");
環境変数名とコンテキストオブジェクト名	コンテキストオブジェクト属性が、指定された環境変数にコピーされます	activation_context("DISPLAY=DECW$DISPLAY");
コンテキストオブジェクトの一部にワイルドカードを利用する	ワイルドカードと一致するコンテキストオブジェクトが、インプリメンテーションのコンテキストオブジェクトにコピーされます	activation_context("DECWS*");
環境変数と、ワイルドカードを含むコンテキストオブジェクト名	ワイルドカードと一致するコンテキストオブジェクト属性が、同じ名前の環境変数にコピーされます	activation_context("=DECWS*");
ワイルドカードのみ	すべてのコンテキストオブジェクトが、インプリメンテーションのコンテキストオブジェクトにコピーされます	activation_context("*");
環境変数と、ワイルドカード	すべてのコンテキストオブジェクトが、同じ名前の環境変数にコピーされます	activation_context("=*");

3.4.2.6. implementation_identifierステートメント

implementation_identifier()ステートメントにはインプリメンテーションの識別子を指定します。ここで指定する識別子はインプリメンテーションを登録する場合に利用されます。識別子を生成する場合には次のコマンドを利用します。

OpenVMS:
GENERATE UNIQUE_IDENTIFIER
UNIXおよびWindowsNT:
orbgen -u
WindowsおよびMacintosh:
システムアドミニストレータのコマンド領域でUNIXと同様にorbgenコマンドを実行します

3.4.2.7. registeration_attributeステートメント

registration_attribute()ステートメントでは、インプリメンテーションを登録する場合に指定する属性を指定します。registration_attribute()ステートメントはオプションです。

registration_attribute()ステートメントには次の値を指定します。

表 23 activation_context()ステートメントの値

	データ型	説明	使用する状況
Hidden	boolean	インプリメンテーションはオブジェクトブローカからは選択されませんので、クライアントにインプリメンテーションをバインドしたオブジェクトを渡す必要があります	インプリメンテーションはバインドされたオブジェクトのみのリクエストを処理する場合
ServerInstanceId	string	インプリメンテーションを指定するサーバ識別子と関係付けます	特定のサーバインスタンスを利用する必要がある場合
ImplementationName	string	インプリメンテーションの名前を指定します	活性化されているインプリメンテーションをリスト表示する場合に利用されます

3.4.2.8. selection_attributeステートメント

selection_attribute()ステートメントには、サーバを選択する場合に参照する属性を定義します。MML(後述)の select_server ステートメントなどでここで指定された属性によるサーバ選択をおこないます。MML では参照されない属性を定義することもできます。

3.4.2.9. registration_routineステートメント

registration__routine()ステートメントには、インプリメンテーションを初期化するための登録ルーチンを指定します。ここで指定した名前の登録ルーチンをオブジェクトブローカが生成します。このステートメントはオプションで、指定しない場合には次のルーチン名で生成されます。

インプリメンテーション名__register

登録ルーチンはサーバスケルトンファイルに生成されます。生成された登録ルーチンを変更しないでください。

3.4.2.10. method_dispatcher_routineステートメント

method_dispatcher_routine()ステートメントには、インプリメンテーションのメ

ソッドディスパッチルーチンを指定します。ここで指定した名前のディスパッチルーチンをオブジェクトブローカが生成します。このステートメントはオプションで、指定しない場合には次のルーチン名で生成されます。

インプリメンテーション名__dispatch

ディスパッチルーチンはサーバスケルトンファイルに生成されます。生成されたディスパッチルーチンを変更しないでください。

3.4.2.11. イベント通知ルーチン定義ステートメント

オブジェクトやインプリメンテーションが活性化あるいは非活性化される場合に、オブジェクトブローカはイベントを生成します。イベント通知ルーチン定義ステートメントでは、サーバが受け取るイベントを指定します。イベント通知ルーチンを利用すると、オブジェクトが活性化された時点で効率的に初期化したり、非活性化された時点でオブジェクトが確保していたリソース開放する処理を行なうことができます。

次のイベントに対して通知ルーチンを定義することができます。

表 24 イベント通知ルーチン

イベント	呼び出される理由
event_notifier_routine	サーバのアクションを必要な要求が、オブジェクトブローカから出された場合 このルーチンはユーザ独自のイベントディスパッチャを作成する場合に指定します
activate_impl_routine	基本オブジェクトアダプタ(BOA)がインプリメンテーションを活性化した場合
activate_obj_routine	基本オブジェクトアダプタ(BOA)がオブジェクトを活性化した場合
deactivate_impl_routine	基本オブジェクトアダプタ(BOA)がインプリメンテーションを非活性化した場合
deactivate_obj_routine	基本オブジェクトアダプタ(BOA)がオブジェクトを非活性化した場合

3.4.3. メソッドの指定

インプリメンテーションのメソッドごとにメソッドの名前、メソッドがサポートするオペレーション、メソッドを実行する方法などを定義します。メソッドと、そのメソッドがサポートするオペレーションとは、シグネチャが一致している必要があります。

一般的にインプリメンテーションは、対応するインタフェースのすべてのオペレーションをサポートするメソッドが定義されることになりますが、分割して定義することも可能です。

3.4.3.1. methodステートメント

method ステートメントはインプリメンテーションにメソッドを定義するために使用します。

メソッドごとに次の項目を定義します。

表 25 method ステートメント

	指定内容	使用する状況
メソッド名	メソッドの名前	すべてのメソッドで指定します
implements	メソッドがサポートするオペレーション	すべてのメソッドで定義します
invoke_メソッドタイプ	メソッドを起動する方法を指定します 組込みメソッド: invoke_builtin()を使用してルーチン名を指定します 動的ロードメソッド: invoke_dynamic_load()を使用して動的ライブラリ名とルーチン名を指定します スクリプトメソッド: invoke_script()を使用してスクリプトファイルを指定します。	それぞれのメソッドタイプでのデフォルト 組込みメソッド: invoke_builtin("ルーチン名") 動的ロードメソッド: invoke_dynamic_load ("ルーチン名"["ライブラリ名"])
context	メソッドにコピーされるコンテキストオブジェクトの属性を指定します	オペレーションに指定されていても、ここで指定されていない属性はコピーされません

3.4.4. インプリメンテーション定義のロード、表示

インプリメンテーション定義のロードおよび表示は、インターフェース定義のロードおよび表示と同じコマンドを利用します。3.3.10および3.3.11を参照してください。

3.5 メソッドマップ定義

メソッドマップでは、リクエストされたオペレーションを処理するために実行することになるメソッドを指定します。

メソッドマップでは次の項目を定義します。

- オペレーションに対応するメソッド
- インプリメンテーションを選択する条件
- サーバインスタンスを選択する条件

メソッドマップは、MML(Method Mapping Language:メソッドマッピング定義言語)で定義します。MML ファイルのテンプレートを、次のコマンドで生成することができます。

OpenVMS:

```
GENERATE INTERFACE/DEFAULT=MAPPING =MMLファイル名 インターフェース名
COMPILE INTERFACE/DEFAULT=MAPPING =MMLファイル名 -
インターフェース定義ファイル名
```

UNIXおよびWindowsNT:

```
obbgen -m MMLファイル名 インターフェース名
obbcomp -m MMLファイル名 インターフェース定義ファイル名
```

3.5.1. MMLの例

```
// ****************************************************************
// Created Thu Jun 16 14:07:08 1994 by OBB T2.5-05 (COMPILE/GENERATE)
// ****************************************************************
//
// OBB Default MML file
// ---------------
//
// Use this file as a base for your additional MML definitions.
//
method_map AccountMap interface BNK1::Account
{
        Deposit :
        {
                default:
                        select_implementation
                                by_name ( AccountImpl )
                                by_identifier ( "67717481f40c.0c.2b.4d.00.00.00.00.00" )
                        ;
        };
        Withdraw :
        {
                default :
                        select_implementation
                                by_name ( AccountImpl )
                                by_identifier ( "67717481f40c.0c.2b.4d.00.00.00.00.00" )
                        ;
};
        Balance :
        {
                default :
                        select_implementation
                                by_name ( AccountImpl )
                                by_identifier ( "67717481f40c.0c.2b.4d.00.00.00.00.00" )
                        ;
        };
};
method_map CustomerMap interface BNK1::Customer
{
        Validate :
        {
                default :
                        select_implementation
                                by_name ( CustomerImpl )
                                by_identifier ( "67717481f410.0c.2b.4d.00.00.00.00.00" )
                        ;
        };
};
```

3.5.2 メソッドマップの指定

MML ファイルではMML を利用してメソッドマップを利用します。ここではMMLの概略を説明します。

3.5.2.1. method_mapステートメント

method_map ステートメントではメソッドマップの名前を指定します。

3.5.2.2 interfaceステートメント

interfaceステートメントではメソッドマップに関係付けられるインターフェースを指定します。インターフェース名はCORBA IDLの interface ステートメントで指定されていなくてはいけません。

3.5.2.3. search_scopeステートメント

search_scope ステートメントは、メソッド選択で参照するコンテキストオブジェクトのレベルを指定します。指定できる値はuser,group,system のいずれかです。

3.5.2.4. オペレーションステートメント

interfaceステートメントで指定したインターフェースの各オペレーションに対して、condition ステートメントあるいはdefault ステートメントを利用して、メソッド決定で利用する条件を定義します。

3.5.3. メソッド決定基準の指定

メソッド決定の指定には次の三種類のMML ステートメントを利用します。

- condition ステートメント
- select_implementation ステートメント
- select_server ステートメント

3.5.3.1. conditionステートメント

condition ステートメントでは実行時に比較することになる値を指定します。値が一致した場合に、メソッドが選択されたことになります。複数のcondition ステートメントで条件が真になる場合には、最初に定義されているcondition ステートメントが有効になります。

コンテキストオブジェクトの属性値を含む、任意の値を指定することができます。コンテキストオブジェクトの属性値を参照するにはcontext 関数を利用します。

3.5.3.2 select_implementationステートメント

select_implementation ステートメントでは、インプリメンテーションを指定します。オペレーション要求に対応するインプリメンテーションの名前あるいは識別子がわかっている場合に利用します。

表 26 select_implementation ステートメント

	使用方法	使用する状況
by_name	IML で定義したインプリメンテーション名を指定します。	特定のインプリメンテーションが必要な場合。
by_identifier	インプリメンテーションに設定した識別子を指定します。	特定のインプリメンテーションが必要な場合。識別子を利用する場合には登録簿は参照しません

3.5.3.3 select_serverステートメント

select_server ステートメントは、サーバ選択条件を指定します。一つのインプリメンテーションに対して、複数のサーバが対応している場合に、特定のサーバを指定することができます。

select_server ステートメントを指定しないと、次の順序でサーバを探索します。

1. アドバティスメント
2. クライアントと同じノード
3. デフォルトノードリスト

次の項目を指定できます。

表 27 select_server ステートメント

	使用方法	使用する状況
by_identifier	サーバの識別子を指定します	識別子が既知であるサーバを指定する場合
selection_policy	ノード探索順序を指定します	デフォルトのノード探索順序が有効ではない場合
startup_policy	サーバを起動するノードの探索順序を指定します	デフォルトのサーバ起動ノード探索順序が有効でない場合

4. クライアントプログラミング

4.1. 概要

オブジェクトブローカのクライアントのインプリメンテーションでは、次の処理を行なう必要があります。

- オブジェクトの管理
- メソッドの起動
- メソッドからの結果の処理

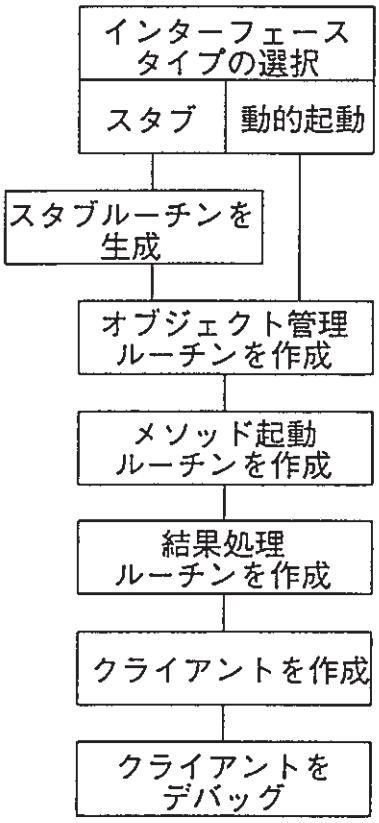


図 14 クライアントの開発手順

4.2 インターフェースタイプの選択

クライアントではインターフェースタイプとして、次のいずれかを選択します。

- スタブ—スタブインターフェースでは、クライアントが固定されたスタブル

ーチンを利用することになります。スタブルーチンでは、メソッドを起動するコードが組み込まれているため、標準的なサブルーチンコールと同じ形式で、サーバのサービスを利用することができます。

- 動的—動的インターフェースでは、クライアントが実行中にリクエストを作成します。動的インターフェースを利用するとクライアントが柔軟的になり、遅延同期オペレーションを利用することができます。

スタブインターフェースと動的インターフェースは次の手順で選択します。

1. メソッドの起動方法

クライアントの条件が以下の場合には、スタブインターフェースを利用します。

- 同期要求のみを利用する
- 固定したメソッドマップを利用する
- 開発後に定義するインターフェースを利用しない

クライアントが次の条件を要求する場合には、動的インターフェースを利用します。

- 遅延同期リクエストを利用する
- 開発後に定義するインターフェースにアクセスする
- VisualBasic からオブジェクトブローカを利用する

2. 最終製品が必要なクライアントのパッケージング形式

- リポジトリを利用しない場合には、スタブインターフェースを利用することになります。このためIDL ファイルとMML ファイルをアプリケーションとともにパッケージする必要がありません。
- 動的インターフェースでは実行時にリポジトリを参照します。このためIDL ファイルとMML ファイルをアプリケーションとともにパッケージして、アプリケーションをインストールするときに、それらをインターフェースリポジトリにロードする必要があります。

4.3 スタブインターフェースの生成

このステップではクライアント用のスタブファイルを生成します。

4.3.1. GENERATE INTERFACE コマンドあるいはCOMPILEコマンドによるスタブインターフェースの生成

スタブインターフェースはGENERATE INTERFACEコマンドあるいはCOMPILE コマンドを利用して生成します。GENERATE INTERFACEコマン

ドはインターフェースリポジトリの定義を利用してスタブインターフェースを生成するのにたいして、COMPILE コマンドはIDL,MML,IML ファイルを直接使用してスタブファイルを生成する点が異なります。

4.3.2 スタブインターフェースを生成するための準備:UUIDの生成

スタブインターフェースを生成する前に、インターフェース定義に記述されている各オブジェクトにUUID(Universal Unique Identifier)を割り当てる必要があります。UUID はインターフェースやインプリメント、オペレーション、メソッド、タイプコードなどのオブジェクトに割り当てます。次のコマンドを利用すると、IDL ファイルのインターフェース定義にUUID を割り当てることができます。

OpenVMS:
```
$APPL/BROKER GENERATE UNIQUE_IDNETIFIER /FILE=ファイル名
```
UNIXと WindowsNT:
```
obbgen -u -f ファイル名
```

4.3.2.1. UUIDを利用する理由

UUID を利用することで、複数のシステムで定義ファイルを分散しても、オブジェクトを一意に認識することができます。

UUID を定義ファイルで指定していないと、定義ファイルをリポジトリにロードするとき、あるいは定義ファイルからスタブファイルを生成するときに、オブジェクトブローカが自動的にオブジェクトにUUID を割り当てます。この場合、定義ファイルをロードしたりスタブファイルを生成するたびに、異なるUUID を割り当てることになります。したがって同じオブジェクトであってもシステムごとに異なるUUID を割り当てられることになります。オブジェクトブローカは書くオブジェクトをUUID で識別しているため、これらの異なるUUID が設定されているオブジェクトを同じオブジェクトとは認識しません。

4.3.3 手順

1. 開発環境でインターフェース定義やメソッドマップ(IDLファイルとMMLファイル)を格納するためにリポジトリファイルを使用するかどうかを決めます。
2. リポジトリファイルを使用しない場合には、IDL ファイルとMML ファイルから直接スタブインターフェースを生成することになります。デフォルトのメソッドマップを生成する場合には、/DEFAULT 修飾子あるいは-m オプションを指定します。

   OpenVMS:
   ```
   COMPILE/STUB=(CLIENT=スタブファイル名)/LANGUAGE=C -
   [/DEFAULT=(MAPPING=MML-ファイル名) IDL-ファイル名
   ```

UNIXおよびWindowsNT:
obbcomp [-m MML-ファイル名] -c スタブファイル名
IDL-ファイル名

3. リポジトリファイルを使用する場合には、まずIDL ファイルをリポジトリにロードします。
OpenVMS:
LOAD REPOSITORY_OBJECT [/CREATE][/REPOSITORY=リポジトリファイル名] -
IDL-ファイル名
UNIXおよびWindowsNT:
obbldrep[-c][-r リポジトリファイル名] IDL-ファイル名
4. リポジトリで定義されているインターフェース定義からスタブインターフェースを生成します。コンパイルされたメソッドマップを使用する場合には、この段階でスタブファイルにメソッドマップを埋め込みます。
OpenVMS:
GENERATE INTERFACE/STUB=(CLIENT=スタブファイル名) -
[/REPOSITORY_LIST=リポジトリファイル名]/LANGUAGE=C -
インターフェース名
UNIXおよびWindowsNT:
obbgen -c スタブファイル名[-r リポジトリファイル名]
インターフェース名

4.4. オブジェクトの管理

オブジェクトブローカでは、クライアントはサーバにアクセスする前に初期オブジェクトにアクセスする必要があります。クライアントではオブジェクトを作成できないため、クライアントはなんらかの方法で初期オブジェクトに対するオブジェクトリファレンスを入手する必要があります。

クライアントが初期オブジェクトへのオブジェクトリファレンスを入手する方法として次の三つの手段があります。

- オブジェクトリファレンスをクライアントのソースプログラムに埋め込む方法があります。この場合、クライアントアプリケーションはサーバアプリケーションと平行して開発するか、クライアントアプリケーションの開発時にオブジェクトリファレンスをなんらかの手段で入手する必要があります。
- クライアントはオブジェクトリファレンスを外部から入手する方法があります。例えばオブジェクトリファレンスをファイルに格納することができます。またオブジェクトリファレンスを文字列形式で格納するデータベースを用意することもできます。
- オブジェクトブローカでは初期オブジェクトのオブジェクトリファレンスを登録簿のアドバティスメントパーティションに格納しておくことができます。クライアントアプリケーションは共通オブジェクトサービスのネーミングサービスルーチンによって、オブジェクトリファレンスにアクセスできることになります。オブジェクトブローカではOMG で定義している共通オブジェクトサービスの一部である、ネーミングサービスのサブセットが実装されています。
サーバをインストールするときに、REGISTER OBJECTコマンドあるいはobbreg -Oコマンドでアドバティスメントパーティションに初期オブジェク

トのオブジェクトリファレンスを格納することになります。
クライアントアプリケーションは、オブジェクトを登録するときに指定された名前で、オブジェクトを検索することになります。ネーミングサービスには名前から該当するオブジェクトリファレンスを返すオペレーションが定義されています。
ネーミングサービスには登録されているオブジェクトリファレンスの一覧を参照するオペレーションも用意されています。

4.4.1. ネームサービスのIDL定義

```
module Naming {
        typedef string lstring;
        struct NameComponent {
                lstring id:
                lstring kind;
        };
        typedef sequence<NameComponent> Name;

        Interface NamingContext {
                enum NotFoundReason {
                        missing_node,not_context,not_object
                };
                exception NotFound {
                        NotFoundReason why;
                        Name rest_of_name;
                };
                exception CannotProceed {
                        NamingContext ctx;
                        Name rest_of_name;
                };
                exception InvalidName{};
                Object resolve (in Name n)
                        raises(NotFound, CannotProceed, InvalidNam);
        };
};
```

4.4.2 ネームサービスのC言語へのマッピング

```
typedef CORBA_Object CosNaming_NamingContext;
typedef CORBA_string CosNaming_lstring;
typedef struct CosNaming_NameComponent {
        CosNaming_lstring id;
        CosNaming_lstring kind;
} CosNaming_NameComponent;
typedef struct {
        CORBA_unsigned_long _maximum;
        CORBA_unsigned_long _length;
        CosNaming_NameComponent* _buffer;
}       CORBA_sequence_CosNaming_NameComponent;
typedef CORBA_unsigned_long CosNaming_NamingContext_NotFoundReason;
#define CosNaming_NamingContext_missing_node 1
#define CosNaming_NamingContext_not_context 2
#define CosNaming_NamingContext_not_object 3
typedef struct CosNaming_NamingContext_NotFound {
        CosNaming_NamingContext_NotFoundReason why;
        CosNaming_Name rest_of_name;
} CosNaming_NamingContext_NotFound;
#define ex_CosNaming_NamingContext_NotFound "674a1868c2ef.0c.03.4d.00.00.00.00.00"
typedef struct CosNaming_NamingContext_CannotProceed {
        CosNaming_NamingContext cxt;
        CosNaming_Name rest_of_name;
} CosNaming_NamingContext_CannotProceed;
```

```
#define ex_CosNaming_NamingContext_CannotProceed "674a1868c2f0.0c.03.4d.00.00.00.00.00"
#define ex_CosNaming_NamingContext_InvalidName "674a1868c2f1.0c.03.4d.00.00.00.00.00"
CORBA_Object OBB_EXPORT CosNaming_NamingContext_resolve (
        CosNaming_NamingContext object,
        CORBA_Environment * ev,
        CosNaming_Name * n);
```

4.4.3. ネームサービスを利用するプログラム例

次の例は"test1"というオブジェクトのオブジェクトリファレンスを参照する例です。

```
CosNaming_NameComponent name_comp;
CosNaming_Name          name;
CORBA_Object            obj;
CORBA_Environment       ev;

name_comp.id = "test1";
name_comp.kind = NULL;
name._length = 1;
name._maximum = 1;
name._buffer = &name_comp;
obj = CosNaming_NamingContext_resolve
 (COSNAMING_OBJECT_NAMECONTEXT, &ev, &name);
if ((ev._major != CORBA_NO_EXCEPTION) || (obj == NULL)) {
        printf("no object found n");
}
```

4.5. メソッドの起動

クライアントはオブジェクトリファレンスを入手すると、そのオブジェクトに対してリクエストを送ることができます。

4.5.1. メソッドとのデータの入出力

それぞれのデータ型に関する、入力パラメータ、出力パラメータ、戻り値としての取り扱いについて説明します。

4.5.1.1. 単純データ型 (short, long, unsigned short, unsigned long, float, double, boolean, char, octet, enum)

入力パラメータの場合には、値をそのまま指定します。

出力パラメータの場合には、値を設定する領域を指定します。メソッドが指定された領域に値を設定します。

入出力パラメータの場合には、値を設定した領域を指定します。メソッドが指定された領域の値を変更します。

戻り値としては値がそのまま返されます。

4.5.1.2 オブジェクトリファレンス型(Object)

入力パラメータの場合には、値をそのまま指定します。

出力パラメータの場合には、オブジェクトリファレンスを設定する領域を指定します。メソッドが指定された領域にオブジェクトリファレンスを設定します。設定されたオブジェクトリファレンスが不要になった場合には、CORBA_Object_release()ルーチンを利用して、オブジェクトリファレンスの領域を開放します。

入出力パラメータの場合には、オブジェクトリファレンスを設定した領域を指定します。メソッドは指定された領域のオブジェクトリファレンスを修正することはできません。オブジェクトリファレンスが不要になった場合には、CORBA_Object_release()ルーチンを利用して、オブジェクトリファレンスの領域を開放します。

戻り値の場合には、そのままオブジェクトリファレンスが返されます。オブジェクトリファレンスが不要になった場合には、CORBA_Object_release()ルーチンを利用して、オブジェクトリファレンスの領域を開放します。

4.5.1.3 最大長を指定した配列型(bound sequence)

入力パラメータの場合には以下のように指定します。

1. IDL ファイルで指定したサイズを_maximum フィールドに設定します。
2. _length フィールドには、項目数を設定します。
3. _buffer フィールドには、データ配列の先頭アドレスを指定します。
4. メソッドにはポインタを指定します。

出力パラメータの場合には以下のように指定します。

1. メソッドにはポインタを指定します。
2. _length フィールドを参照して、項目数を読み取ります。
3. _buffer フィールドを参照して、データ配列の先頭アドレスを読み取ります。
4. データが不要になった場合には、_buffer フィールドに設定されている領域をCORBA_free()ルーチンで開放します。

入出力パラメータの場合には以下のように指定します。

1. IDL ファイルで指定したサイズを_maximum フィールドに設定します。
2. _length フィールドには、項目数を設定します。
3. _buffer フィールドには、指定した大きさのデータ配列の先頭アドレスを指定します。
4. メソッドにはポインタを指定します。

5. メソッドで項目数を変更している場合があるので、_length フィールドを参照して項目数を読み取ります。
6. _buffer フィールドを参照して、データ配列の先頭アドレスを読み取ります。
7. データが不要になった場合には、_buffer フィールドに設定されている領域をCORBA_free()ルーチンで開放します。

戻り値の場合には以下のように参照します。

1. _length フィールドを参照して、項目数を読み取ります。
2. _buffer フィールドを参照して、データ配列の先頭アドレスを読み取ります。
3. データが不要になった場合には、_buffer フィールドに設定されている領域をCORBA_free()ルーチンで開放します。

4.5.1.4. 最大長を指定しない配列型(unbound sequence)

入力パラメータの場合には以下のように指定します。

1. 確保した領域のサイズを_maximum フィールドに設定します。
2. _length フィールドには、項目数を設定します。
3. _buffer フィールドには、データ配列の先頭アドレスを指定します。
4. メソッドにはポインタを指定します。

出力パラメータの場合には以下のように指定します。

1. メソッドにはポインタを指定します。
2. _length フィールドを参照して、項目数を読み取ります。
3. _buffer フィールドを参照して、データ配列の先頭アドレスを読み取ります。
4. データが不要になった場合には、_buffer フィールドに設定されている領域をCORBA_free()ルーチンで開放します。

入出力パラメータの場合には以下のように指定します。

1. 確保した領域のサイズを_maximum フィールドに設定します。
2. _length フィールドには、項目数を設定します。
3. _buffer フィールドには、指定した大きさのデータ配列の先頭アドレスを指定します。
4. メソッドにはポインタを指定します。
5. メソッドで項目数を変更している場合があるので、_length フィールドを参照して項目数を読み取ります。
6. _buffer フィールドを参照して、データ配列の先頭アドレスを読み取ります。
7. データが不要になった場合には、_buffer フィールドに設定されている領域をCORBA_free()ルーチンで開放します。

戻り値の場合には以下のように参照します。

1. _length フィールドを参照して、項目数を読み取ります。
2. _buffer フィールドを参照して、データ配列の先頭アドレスを読み取ります。
3. データが不要になった場合には、_buffer フィールドに設定されている領域をCORBA_free()ルーチンで開放します。

4.5.1.5. 共用体型(union)

入力パラメータの場合には以下のように指定します。

1. _d フィールドに型指定の値を設定します。
2. 設定した型に対応するフィールドに値を設定します。
3. メソッドにはポインタを渡します。

出力パラメータの場合には以下のように指定します。

1. メソッドにはポインタを渡します。
2. _d フィールドの値を参照して、設定されているデータ型を調べます。
3. 設定されている型に対応するフィールドを参照して、データを読み取ります。
4. データが不要になった場合には、_d フィールドに設定されているデータ型にしたがって領域を開放します。

入出力パラメータの場合には以下のように指定します。

1. _d フィールドに型指定の値を設定します。
2. 設定した型に対応するフィールドに値を設定します。
3. メソッドにはポインタを渡します。
4. メソッドから返された_d フィールドの値を参照して、設定されているデータ型を調べます。
5. 設定されている型に対応するフィールドを参照して、データを読み取ります。
6. データが不要になった場合には、_d フィールドに設定されているデータ型にしたがって領域を開放します。

戻り値の場合には以下のように参照します。

1. _d フィールドの値を参照して、設定されているデータ型を調べます。
2. 設定されている型に対応するフィールドを参照して、データを読み取ります。
3. データが不要になった場合には、_d フィールドに設定されているデータ型にしたがって領域を開放します。

4.5.1.6. 汎用型(any)

入力パラメータの場合には以下のように指定します。

1. _type フィールドに設定するデータのタイプコードを指定します。
2. _value フィールドに指定したタイプコードに対応するデータ領域を確保します。
3. _value フィールドに指定した領域に値を設定します。
4. メソッドにはポインタを渡します。

出力パラメータの場合には以下のように指定します。

1. メソッドにはポインタを渡します。
2. _type フィールドを参照して、設定されているデータ型を調べます。データ型を調べるにはCORBA_TypeCode_Kind(),CORBA_TypeCode_equal()ルーチンを利用します。
3. _value フィールドから設定されている値を読み取ります。
4. データが不要になった場合には、_value フィールドに設定されている領域を開放します。領域を開放する場合、オブジェクトリファレンスの場合には

CORBA_Object_release()ルーチン、それ以外の場合にはCORBA_free()ルーチンを利用します。また_type フィールドはCORBA_Object_release()ルーチンを利用して、領域を開放します。

入出力パラメータの場合には以下のように指定します。

1. _type フィールドに設定するデータのタイプコードを指定します。
2. _value フィールドに指定したタイプコードに対応するデータ領域を確保します。
3. _value フィールドに指定した領域に値を設定します。
4. メソッドにはポインタを渡します。
5. _type フィールドを参照して、設定されているデータ型を調べます。データ型を調べるにはCORBA_TypeCode_Kind(),CORBA_TypeCode_equal()ルーチンを利用します。
6. _value フィールドから設定されている値を読み取ります。
7. データが不要になった場合には、_value フィールドに設定されている領域を開放します。領域を開放する場合、オブジェクトリファレンスの場合にはCORBA_Object_release()ルーチン、それ以外の場合にはCORBA_free()ルーチンを利用します。また_type フィールドはCORBA_Object_release()ルーチンを利用して、領域を開放します。

戻り値の場合には以下のように参照します。

1. _type フィールドを参照して、設定されているデータ型を調べます。データ型を調べるにはCORBA_TypeCode_Kind(),CORBA_TypeCode_equal()ルーチンを利用します。
2. _value フィールドから設定されている値を読み取ります。
3. データが不要になった場合には、_value フィールドに設定されている領域を開放します。領域を開放する場合、オブジェクトリファレンスの場合にはCORBA_Object_release()ルーチン、それ以外の場合にはCORBA_free()ルーチンを利用します。また_type フィールドはCORBA_Object_release()ルーチンを利用して、領域を開放します。

4.5.1.7. 文字列型(string)

入力パラメータの場合は、メソッドにはデータ領域へのポインタを指定します。

出力パラメータの場合は、メソッドにはデータ領域へのポインタを指定します。メソッドが指定された領域にデータを設定します。設定されたデータが不要になった場合、CORBA_free()ルーチンを利用して領域を開放します。

入出力パラメータの場合は、メソッドにはデータ領域へのポインタを指定します。メソッドが指定された領域のデータを変更します。

戻り値の場合に、データ領域へのポインタが返されます。返されたデータが不要になった場合、CORBA_free()ルーチンを利用して領域を開放します。

4.5.1.8 配列型(array)

入力パラメータの場合には、配列の各要素を初期化してから、メソッドに配列の先頭要素へのポインタを指定します。

出力パラメータの場合には、メソッドに配列へのポインタを指定します。メソッドが指定された配列にデータを設定します。配列の型によっては不要になりしだい、領域を開放します。

入出力パラメータの場合には、配列の各要素を初期化してから、メソッドに配列の先頭要素へのポインタを指定します。メソッドが指定された配列のデータを変更します。

戻り値の場合には、配列へのポインタが返されます。配列の型によっては不要になりしだい、領域を開放します。

4.5.1.9 構造体型(struct)

入力パラメータの場合には、構造体の各要素を初期化してから、メソッドに構造体へのポインタを指定します。

出力パラメータの場合には、メソッドに構造体へのポインタを指定します。メソッドが指定された構造体にデータを設定します。構造体の要素の型によっては不要になりしだい、領域を開放します。

入出力パラメータの場合には、構造体の各要素を初期化してから、メソッドに構造体へのポインタを指定します。メソッドが指定された構造体のデータを変更します。

戻り値の場合には、構造体へのポインタが返されます。構造体の要素の型によっては不要になりしだい、領域を開放します。

4.5.2 リクエストオブジェクトの作成

動的インターフェースの場合には、リクエストオブジェクトを利用して要求するオペレーションを指定します。リクエストオブジェクトは擬似オブジェクトのため、ユーザがIDLファイルで定義する必要はありません。リクエストオブジェクトを作成するには、三種類の方法があります。ここでは一般的な作成方法に関して説明します。

なお一度利用したリクエストオブジェクトを、別のメソッド起動で利用する場合には、CORBA_NVList_free_memory()ルーチンで、前のリクエスト要求で確保

された領域を解放する必要があります。

1. CORBA_Object_create_request()ルーチンでリクエストオブジェクトを作成します。
2. CORBA_Request_add_arg()ルーチンを利用して、リクエストオブジェクトに引き数を設定します。すべての引き数に対して、領域へのポインタとタイプコードをリクエストオブジェクトに設定します。

次のようなIDLで定義されているインターフェースへのリクエストオブジェクトの作成例です。

```
module MODULE1 {
        interface Interface1 {
                long message1(double arg1);
        };
};

CORBA_NamedValue            result;
CORBA_long                  longresult;
CORBA_Status                status;
CORBA_Environment           ev;
CORBA_Context               ctx;
CORBA_Object                object;
CORBA_NVList                arg_list;
CORBA_Request               request;
CORBA_double                value1;
result.len                = sizeof(longresult);
result.name               = NULL;
result.arg_modes          = 0;
result.ValueFlags         = 0;
result.argument._type     = TC_long;
result.argument._value    = (CORBA_char*)&longresult;
status = CORBA_Object_create_request(
        object,
        &ev,
        ctx,
        operation_name,
        NULL,
        &result,
        &request,
        request_flags);
status = CORBA_Request_add_arg(
        request,
        &ev,
        "arg1",
        TC_double,
        (CORBA_void*)&value1,
        sizeof(CORBA_double),
        CORBA_ARG_IN);
```

4.5.3 メソッドの呼び出し

オブジェクトリファレンスを利用してメソッドを起動します。

スタブインターフェースの場合には、オブジェクトリファレンスと指定されたオペレーションに対するパラメータで、スタブルーチンを呼び出します。

動的インターフェースの場合には、オブジェクトリファレンスとリクエストオブジェクトで、CORBA_Request_invoke(), CORBA_Request_send(),

CORBA_send_multiple_requests()ルーチンのいずれかを呼び出します。

CORBA_Request_invoke()ルーチンでは同期呼出し、CORBA_Request_send()ルーチンでは遅延同期呼出し、CORBA_send_multiple_requests()ルーチンでは複数のリクエストを遅延同期で呼び出します。

```
status = CORBA_Request_Invoke(request, &ev, 0);
status = CORBA_Request_send(request, &ev, invoke_flags);
```

表 28 CORBA_Request_send の invoke_flags

フラグ	
CORBA_INV_NO_RESPONSE	起動側は応答や出力引き数を期待しない

4.5.4. リクエストオブジェクトの解放

動的インターフェースの場合には、リクエストオブジェクトが不要になった時点で、CORBA_Request_delete()ルーチンを利用して、リクエストオブジェクトの領域を解放します。

リクエストオブジェクトに設定したパラメータもCORBA_free()ルーチンあるいはCORBA_Object_release()ルーチンを利用して領域を解放します。

4.5.5. クライアントの終了処理

クライアントを終了する前に、OBB_ORB_rundown()ルーチンを呼び出して、オブジェクトブローカに関する内部データをすべて解放します。

4.6. メソッドの結果に対する処理

メソッドでの処理が終了すると、クライアントはORB から受け取る情報を処理します。ORB からは次の情報が返されます。

- 正常終了　　メソッドは要求されたオペレーションを正常に処理
- エラー　　メソッドの選択あるいは実行に問題
  メソッドの実行中に問題

4.6.1. メソッドの終了状態の確認

各 ORB ルーチンが必要とするCORBA_Environement データに返された値を調べます。この構造体の_major フィールドに次のデータのいずれかが設定されています。

1. CORBA_NO_EXCEPTION(正常終了)
   オペレーションに対応するメソッドは正常に実行されています。
2. CORBA_USER_EXCEPTION(ユーザ例外)
   オペレーションに対するメソッドが、メソッドの実行中に問題が発生したために正常に処理されませんでした。パラメータに問題があるなどの可能性があります。
3. CORBA_SYSTEM_EXCEPTION(システム例外)
   オペレーションに対するメソッドが、システムエラーのために正常に処理されませんでした。トランスポートやホスト、サーバ、インプリメンテーション、メソッドなどに問題がある可能性があります。

CORBA_USER_EXCEPTION あるいはCORBA_SYSTEM_EXCEPTION が設定されている場合には、出力パラメータや戻り値の値は保証されません。値を読み取ったり領域を解放しないでください。

4.6.2 エラー情報の取り出し

メソッドが正常に処理されていない場合には、次のルーチンを利用して詳細なエラー情報を取り出すことができます。

- CORBA_exception_id()
  エラーを記述している文字列を返します。この文字列はユーザ例外の場合には、IDL ファイルでオペレーションごとに定義している例外名となります。システム例外の場合には、エラーメッセージが設定されています。
- CORBA_exception_value()
  例外ごとに異なる構造体へのポインタを返します。この構造体はIDL ファイルの例外定義で記述しておきます。

4.6.3 エラー情報領域の解放

メソッドから返されたエラー情報が不要になった時点で、CORBA_exception_free()ルーチンを利用してエラー情報領域を解放します。

4.7. クライアントの構築

4.7.1. タイプコードファイルの生成

動的インターフェースの場合には、外部タイプコードファイルを生成します。

OpenVMS:
GENERATE INTERFACE/TYPECODE=ファイル名 インターフェース名
UNIXおよびWindowsNT:
obbgen -T ファイル名 -X インターフェース名

4.7.2 ソースプログラムのコンパイル

次のソースファイルをコンパイルします。

- クライアントスタブファイル(スタブインターフェースを利用する場合)
- クライアントアプリケーションプログラム
- 外部タイプコードファイル(動的インターフェースを利用する場合)

ソースファイルをコンパイルする場合にorb.h をインクルードする必要があります。

表 29 オブジェクトブローカヘッダファイルの位置

プラットフォーム/言語	ヘッダファイルの位置
Windows	オブジェクトブローカルートディレクトリの下のINCLUDE サブディレクトリ
WindowsNT	¥win32app¥obroker¥include
OSF/1,ULTRIX,SunOS,HP-UX,AIX	/usr/include
Macintosh	ObjectBroker:Interfaces:folder
OpenVMS	SYS$LIBRARY
VisualBasic	オブジェクトブローカルートディレクトリのINCLUDE サブディレクトリにある次のファイルをプロジェクトに追加します ORB.BAS OBBAPI.BAS OBBMSG.BAS

orb.h は表 29のディレクトリにあります。

4.7.3 クライアントのリンク

コンパイルしたファイルをリンクして、実行ファイルを作成します。表 30はリンクに必要なライブラリファイルです。

表 30 オブジェクトブローカライブラリファイル

プラットフォーム/言語	ライブラリファイルの位置
Windows	¥WINDOWS¥OBB.LIB ¥WINDOWS¥OBBCOS.LIB
WindowsNT	¥win32app¥obroker¥lib¥obb.lib ¥win32app¥obroker¥lib¥obbcos.lib
OSF/1	/usr/lib/libobb.so /usr/lib/libobbcos.so
ULTRIX	/usr/lib/libobb.a /usr/lib/libobbtrn.a /usr/lib/liblmf.a /usr/lib/libobbcos.a /usr/lib/dnetstub.o(TCP/IP の場合)
Sun	/usr/lib/libobb.so.2.5 /usr/lib/libobbcos.so.2.5
HP-UX	/usr/lib/libobb.sl /usr/lib/libobbcos.sl
AIX	/usr/lib/libobb.a /usr/lib/libobbcos.a
Macintosh	ObjectBroker:Interfaces:folder;obb.o ファイル ObjectBroker:Interfaces:folder;obbcos.o ファイル
OpenVMS	SYS$LIBRARY:OBB$SHARE.EXE SYS$LIBRARY:OBB$COSSHR.EXE

4.8 クライアントのデバッグ

クライアントをデバッグするために、トレース情報を参照することができます。

4.8.1. トレース情報の設定

OBB_TRACE_FLAGS環境変数あるいは論理名にトレースフラグを設定します。

表 31 トレースフラグ

トレースフラグ	説明
A	オブジェクトブローカのすべての処理をトレースします
C	コマンドラインでの処理をトレースします
E	トレース情報を標準エラー出力に出力します
F	トレース情報をファイルに出力します
I	メソッドの起動をトレースします
L	リポジトリへのアクセスをトレースします
O	トレース情報を標準出力に出力します
P	ディスパッチャに関する処理をトレースします
R	メソッド決定をトレースします
S	スクリプトサーバのみに関する処理をトレースします
T	パラメータに関する処理をトレースします
U	オブジェクトブローカのユーティリティに関する処理をトレースします
X	コンテストオブジェクトに関する処理をトレースします

Macintoch では環境変数や論理名の代わりに、コンテキストオブジェクトの OBB_DEFAULT_TABLE の OBB_TRACE_FLAGS にトレースフラグを指定し

ます。

デフォルトではトレース情報は標準出力に出力されます。トレース情報をファイルにリダイレクトする場合には、OBB_TRACE 環境変数あるいは論理名にファイル名を設定します。設定しない場合のデフォルトのファイル名は下の表のようになります。

表 32 デフォルトのトレースファイル名

プラットフォーム	デフォルトのトレースファイル名
Windows および WindowsNT	OBBTRACE.LOG
OSF/1,ULTRIX,SunOS,HP-UX,AIX	obb_trace.log
VMS	OBB_TRACE.LOG
Macintosh	OBB_TRACE.LOG

Macintosh では環境変数や論理名の代わりに、コンテキストオブジェクトの OBB_DEFAULT_TABLE の OBB_TRACE にトレースファイル名を指定します。

5. サーバプログラミング

5.1. 概要

オブジェクトブローカのサーバは以下の手順で開発することになります。

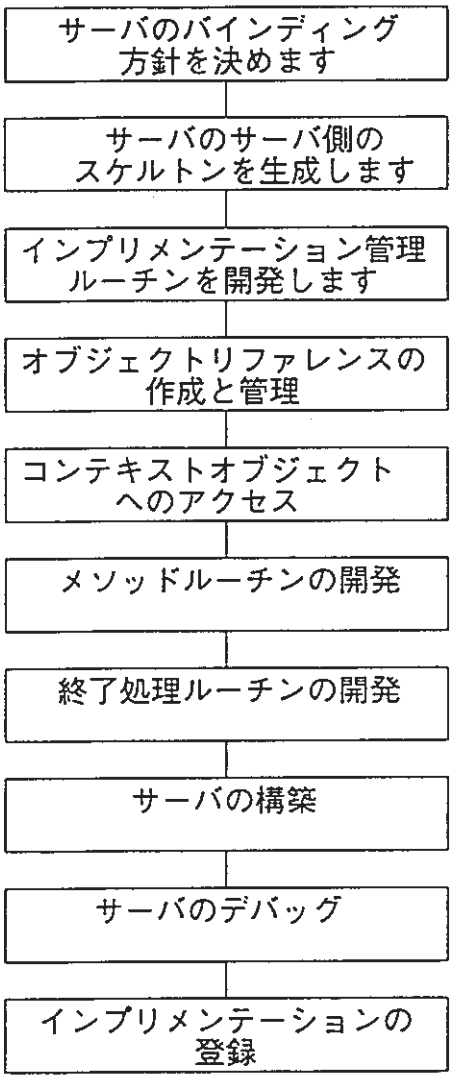


図 15 サーバの開発手順

5.2 バインディング方針の選択

バインディング方針とはインプリメンテーションとサーバの関係を意味します。

バインディング方針には次の3種類があります。

- 静的
  ある特定のサーバが、インプリメンテーションに対するすべてのメソッドを実行します。かならず固有のサーバを利用する場合などに利用します。
- 自動
  最初に選択する場合には、そのインプリメンテーションを処理できるサーバから選択されますが、二度目以降は同じサーバで処理されます。オブジェクトへの複数のリクエストで情報を共有する必要がある場合などに利用します。
- 動的
  各選択時ごとに異なるサーバを選択することができます。各リクエストごとに異なるサーバを選択する場合などに利用します。

サーバの活性化方針とバインディング方針の関係について説明します。

- 共用
  - 静的
    バインディングされているサーバが選択されます。選択可能なサーバがない場合には、新たにサーバを起動します。2度目以降の選択では最初と同じサーバが選択されます。
  - 自動
    選択可能な状態にあるサーバが選択されます。選択可能なサーバがない場合には、新たにサーバを起動します。2度目以降の選択では最初と同じサーバが選択されます。
  - 動的
    選択可能な状態にあるサーバが選択されます。選択可能なサーバがない場合には、新たにサーバを起動します。2度目以降の選択でも最初と同様の選択を繰り返します。
- 永続的
  - 静的
    バインディングされているサーバが選択されます。バインディングされているサーバが起動されていない場合には、エラーになります。2度目以降の選択では最初と同じサーバが選択されます。
  - 自動
    選択可能な状態にあるサーバが選択されます。選択可能な状態にあるサーバがない場合には、エラーになります。2度目以降の選択では最初と同じサーバが選択されます。
  - 動的
    選択可能な状態にあるサーバが選択されます。選択可能な状態にあるサーバがない場合には、エラーになります。2度目以降の選択でも最初と同様の選択を繰り返します。
- 非共用
  - 静的

他のオブジェクトと関連付けられていない、バインディングされているサーバが選択されます。バインディングされているサーバが起動されていない場合には、新たにサーバを起動します。2度目以降の選択では最初と同じサーバが選択されます。

- 自動
  他のオブジェクトと関連付けられていない、選択可能な状態にあるサーバが選択されます。選択可能な状態にあるサーバがない場合には新たにサーバを起動します。2度目以降の選択では最初と同じサーバが選択されます。
- 動的
  利用しているオブジェクトに関連付けられている、選択可能な状態にあるサーバを選択します。選択可能な状態にあるサーバがない場合には、新たにサーバを起動します。2度目以降の選択でも最初と同様の選択を繰り返します。

- メソッド対応のサーバ
  - 静的
    この組み合わせは利用できません。
  - 自動
    この組み合わせは利用できません。
  - 動的
    常にサーバを起動します。

5.3 サーバスケルトンの生成

サーバスケルトンファイルには、サーバを基本オブジェクトアダプタ(BOA)と統合するために必要なソースコードが含まれています。オブジェクトブローカはIMLファイルの定義内容からサーバスケルトンファイルを生成します。サーバスケルトンファイルには次の2つのファイルがあります。

- ディスパッチャルーチンファイル
  - サーバを起動した直後に呼び出すインプリメンテーション登録ルーチン
  - 基本オブジェクトアダプタがメインループの内部で呼び出すメソッドディスパッチャルーチン
  - IMLファイルで指定したオブジェクトインプリメンテーション通知ルーチン
    - オブジェクトの活性化、非活性化通知ルーチン
    - インプリメンテーションの活性化、非活性化通知ルーチン
- メソッドルーチンファイル
  - メソッドルーチンのテンプレート

サーバスケルトンファイルは次の手順で生成します。

1. インプリメンテーションリポジトリを利用するかどうかを決めます。
2. インプリメンテーションリポジトリを利用しない場合には、IDLファイルと

IMLファイルからサーバスケルトンファイルを生成します。

OpenVMS:

```
COMPILE/DISPATCHER=ディスパッチャルーチンファイル名 -
/TEMPLATES=メソッドテンプレートファイル名 IDL-ファイル名,IML-ファイル名
```

UNIXおよびWindowsNT:

```
obbcomp -d ディスパッチャルーチンファイル名
-t メソッドテンプレートファイル名 IDL-ファイル名 IML-ファイル名
```

3. インプリメンテーションリポジトリを利用する場合には、最初にIMLファイルをリポジトリにロードします。

OpenVMS:

```
LOAD REPOSITORY_OBJECT[/CREATE] -
[/REPOSITORY=リポジトリファイル名] IDL-ファイル名,IML-ファイル名
```

UNIXおよびWindowsNT:

```
obbldrep[ -c][ -r リポジトリファイル名] IDL-ファイル名 IML-ファイル名
```

4. インプリメンテーションリポジトリを利用して、サーバスケルトンファイルを生成します。

OpenVMS:

```
GENERATE IMPLEMENTATION [/REPOSITORY=リポジトリファイル名]-
/DISPATCHER=ディスパッチャルーチンファイル名 -
/TEMPLATES=メソッドテンプレートファイル名 インプリメンテーション名
```

UNIXおよびWindowsNT:

```
obbgen[ -r リポジトリファイル名]
-d ディスパッチャルーチンファイル名
-t メソッドテンプレートファイル名 インプリメンテーション名
```

5.4. インプリメンテーション管理ルーチンの開発

各サーバに対して次のような初期処理と終了処理が必要です。

- 初期処理
  - インプリメンテーションの登録します。
  - 次のいずれかの方法でインプリメンテーションを活性化します。
    - インプリメンテーションをすぐに活性化して、クライアントからの要求を処理できるように準備します。
    - クライアントからの要求を受け取った後で、インプリメンテーションを活性化します。
  - クライアントからの要求を受け取るために、メインループに入ります。
- 終了処理
  - インプリメンテーションを登録を解除します。
  - リソースを解放します。

5.4.1. インプリメンテーションを直ちに活性化する共用サーバ

1. サーバを初期化します。OBB_BOA_Initialize()ルーチンの引き数は下の表のようになります。

表 33 インプリメンテーションを直ちに活性化する共用サーバの場合に設定するOBB_BOA_Initialize()ルーチンのパラメータ

	指定内容
boa	CORBA_DEC_BOA_OBJECT
ev	エラー情報データへのポインタ
impldef	インプリメンテーション定義データへのポインタ
unsharedobj	0
flags	0

2. 各インプリメンテーションに対して、サーバスケルトンファイルで生成された登録ルーチンを呼び出して、サーバがサポートするすべてのインプリメンテーションを登録します。登録ルーチンはオブジェクトの生成で利用することになるインプリメンテーション定義を返します。
3. 各インプリメンテーションはCORBA_BOA_impl_is_ready()ルーチンを利用して、クライアントからのリクエストを処理する準備ができたことをエージェントに知らせます。
4. クライアントからのリクエストを処理するために、メインループに入ります。
5. サーバがクライアントからのリクエストを受け付けなくなる時点で、CORBA_BOA_deactive_obj ルーチンを使用してオブジェクトに関連するリソースを解放します。
6. インプリメンテーションがすべての作業を終了した時点でCORBA_BOA_deactive_impl()ルーチンを使用して、インプリメンテーションを非活性化します。
7. 適当なルーチンを利用して、クライアントからのリクエストを処理するためのメインループからぬけます。たとえばOBB_BOA_main_loop()を呼び出していた場合には、OBB_BOA_exit_main_loop()ルーチンを利用します。
8. OBB_BOA_impl_unregister()ルーチンを利用して、エージェントのアクティブなインプリメンテーションのリストから、該当するインプリメンテーションを削除します。
9. CORBA_BOA_dispose()ルーチンを利用して、特定のオブジェクトと関連するリソースをすべて解放します。
10. OBB_ORB_rundown()ルーチンを利用して、オブジェクトブローカが利用しているリソースをすべて解放します。

5.4.2 クライアント要求を受け取った後でインプリメンテーションを活性化する共用サーバ

1. サーバを初期化します。OBB_BOA_Initialize()ルーチンの引き数は下の表のようになります。

表 34 クライアント要求を受け取った後でインプリメンテーションを活性化する共用サーバの場合に設定するOBB_BOA_Initialize()ルーチンのパラメータ

	指定内容
boa	CORBA_DEC_BOA_OBJECT
ev	エラー情報データへのポインタ
impldef	インプリメンテーション定義データへのポインタ
unsharedobj	0
flags	0

2. 各インプリメンテーションに対して、サーバスケルトンファイルで生成された登録ルーチンを呼び出して、サーバがサポートするすべてのインプリメンテーションを登録します。登録ルーチンはオブジェクトの生成で利用することになるインプリメンテーション定義を返します。
3. クライアントからのリクエストを処理するために、メインループに入ります。
4. クライアントからリクエストが送られた時点でインプリメンテーションを活性化します。このインプリメンテーションに対する最初の要求が送られた時に通知されるように、インプリメンテーションの活性化通知ルーチンを設定しておきます。通知を受けたらCORBA_BOA_impl_is_ready ルーチンを利用して、インプリメンテーションがクライアントのリクエストを処理する準備ができたことをエージェントに知らせます。
5. サーバがクライアントからのリクエストを受け付けなくなる時点で、CORBA_BOA_deactive_obj ルーチンを使用してオブジェクトに関連するリソースを解放します。
6. インプリメンテーションがすべての作業を終了した時点でCORBA_BOA_deactive_impl()ルーチンを使用して、インプリメンテーションを非活性化します。
7. 適当なルーチンを利用して、クライアントからのリクエストを処理するためのメインループからぬけます。たとえばOBB_BOA_main_loop()を呼び出していた場合には、OBB_BOA_exit_main_loop()ルーチンを利用します。
8. OBB_BOA_impl_unregister()ルーチンを利用して、エージェントのアクティブなインプリメンテーションのリストから、該当するインプリメンテーションを削除します。
9. CORBA_BOA_dispose()ルーチンを利用して、特定のオブジェクトと関連するリソースをすべて解放します。
10. OBB_ORB_rundown()ルーチンを利用して、オブジェクトブローカが利用しているリソースをすべて解放します。

5.4.3 インプリメンテーションを直ちに活性化する非共用サーバ

1. サーバを初期化します。OBB_BOA_Initialize()ルーチンの引き数は下の表のようになります。

表 35 インプリメンテーションを直ちに活性化する非共用サーバの場合に設定するOBB_BOA_Initialize()ルーチンのパラメータ

	指定内容
boa	CORBA_DEC_BOA_OBJECT
ev	エラー情報データへのポインタ
impldef	インプリメンテーション定義データへのポインタ
unsharedobj	クライアントが使用するオブジェクトリファレンスへのポインタ
flags	0

2. 各インプリメンテーションに対して、サーバスケルトンファイルで生成された登録ルーチンを呼び出して、サーバがサポートするすべてのインプリメンテーションを登録します。登録ルーチンはオブジェクトの生成で利用することになるインプリメンテーション定義を返します。
3. あるオブジェクトに対応するインプリメンテーションはCORBA_BOA_obj_is_ready()ルーチンを利用して、クライアントからのリクエストを処理する準備ができたことをエージェントに知らせます。
4. クライアントからのリクエストを処理するために、メインループに入ります。
5. サーバがクライアントからのリクエストを受け付けなくなる時点で、CORBA_BOA_deactive_obj ルーチンを使用してオブジェクトに関連するリソースを解放します。
6. 適当なルーチンを利用して、クライアントからのリクエストを処理するためのメインループからぬけます。たとえばOBB_BOA_main_loop()を呼び出していた場合には、OBB_BOA_exit_main_loop()ルーチンを利用します。
7. OBB_BOA_impl_unregister()ルーチンを利用して、エージェントのアクティブなインプリメンテーションのリストから、該当するインプリメンテーションを削除します。
8. CORBA_BOA_dispose()ルーチンを利用して、特定のオブジェクトと関連するリソースをすべて解放します。
9. OBB_ORB_rundown()ルーチンを利用して、オブジェクトブローカが利用しているリソースをすべて解放します。

5.4.4. クライアント要求を受け取った後でインプリメンテーションを活性化する非共用サーバ

1. サーバを初期化します。OBB_BOA_Initialize()ルーチンの引き数は下の表のようになります。

表 36 クライアント要求を受け取った後でインプリメンテーションを活性化する非共用サーバの場合に設定するOBB_BOA_Initialize()ルーチンのパラメータ

	指定内容
boa	CORBA_DEC_BOA_OBJECT
ev	エラー情報データへのポインタ
impldef	インプリメンテーション定義データへのポインタ
unsharedobj	クライアントが使用するオブジェクトリファレンスへのポインタ
flags	0

2. 各インプリメンテーションに対して、サーバスケルトンファイルで生成された登録ルーチンを呼び出して、サーバがサポートするすべてのインプリメン

テーションを登録します。登録ルーチンはオブジェクトの生成で利用することになるインプリメンテーション定義を返します。

3. クライアントからのリクエストを処理するために、メインループに入ります。
4. クライアントからリクエストが送られた時点でオブジェクトを活性化します。このオブジェクトに対する最初の要求が送られた時に通知されるように、オブジェクトの活性化通知ルーチンを設定しておきます。通知を受けたらCORBA_BOA_obj_is_ready ルーチンを利用して、オブジェクトがクライアントのリクエストを処理する準備ができたことをエージェントに知らせます。
5. サーバがクライアントからのリクエストを受け付けなくなる時点で、CORBA_BOA_deactive_obj ルーチンを使用してオブジェクトに関連するリソースを解放します。
6. インプリメンテーションがすべての作業を終了した時点でCORBA_BOA_deactive_impl()ルーチンを使用して、インプリメンテーションを非活性化します。
7. 適当なルーチンを利用して、クライアントからのリクエストを処理するためのメインループからぬけます。たとえばOBB_BOA_main_loop()を呼び出していた場合には、OBB_BOA_exit_main_loop()ルーチンを利用します。
8. OBB_BOA_impl_unregister()ルーチンを利用して、エージェントのアクティブなインプリメンテーションのリストから、該当するインプリメンテーションを削除します。
9. CORBA_BOA_dispose()ルーチンを利用して、特定のオブジェクトと関連するリソースをすべて解放します。
10. OBB_ORB_rundown()ルーチンを利用して、オブジェクトブローカが利用しているリソースをすべて解放します。

5.4.5 永続サーバ

1. サーバを初期化します。OBB_BOA_Initialize()ルーチンの引き数は下の表のようになります。

表 37 永続サーバの場合に設定するOBB_BOA_Initialize()ルーチンのパラメータ

	指定内容
boa	CORBA_DEC_BOA_OBJECT
ev	エラー情報データへのポインタ
impldef	0
unsharedobj	0
flags	0

2. 永続サーバは手動で起動する必要があります。
3. 各インプリメンテーションに対して、サーバスケルトンファイルで生成された登録ルーチンを呼び出して、サーバがサポートするすべてのインプリメンテーションを登録します。登録ルーチンはオブジェクトの生成で利用することになるインプリメンテーション定義を返します。
4. 各インプリメンテーションはCORBA_BOA_impl_is_ready()ルーチンを利用して、クライアントからのリクエストを処理する準備ができたことをエージェントに知らせます。
5. クライアントからのリクエストを処理するために、メインループに入ります。
6. サーバがクライアントからのリクエストを受け付けなくなる時点で、CORBA_BOA_deactive_obj ルーチンを使用してオブジェクトに関連するリ

ソースを解放します。

7. インプリメンテーションがすべての作業を終了した時点でCORBA_BOA_deactive_impl()ルーチンを使用して、インプリメンテーションを非活性化します。
8. 適当なルーチンを利用して、クライアントからのリクエストを処理するためのメインループからぬけます。たとえばOBB_BOA_main_loop()を呼び出していた場合には、OBB_BOA_exit_main_loop()ルーチンを利用します。
9. OBB_BOA_impl_unregister()ルーチンを利用して、エージェントのアクティブなインプリメンテーションのリストから、該当するインプリメンテーションを削除します。
10. CORBA_BOA_dispose()ルーチンを利用して、特定のオブジェクトと関連するリソースをすべて解放します。
11. OBB_ORB_rundown()ルーチンを利用して、オブジェクトブローカが利用しているリソースをすべて解放します。

5.5. オブジェクトリファレンスの作成と管理

サーバがオブジェクトを作成して、クライアントはオブジェクトへのポインタであるオブジェクトリファレンスを利用して、オブジェクトにアクセスします。サーバはオブジェクトおよびオブジェクトリファレンスを管理することになります。各サーバごとにオブジェクトリファレンスを作成する機能を組み込みます。

5.5.1. オブジェクトリファレンスの作成

1. オブジェクトに対応するリファレンスデータを指定します。インプリメンテーションはリファレンスデータを参照して、オブジェクトリファレンスが対応するインスタンスを判別します。
   有効なリファレンスデータとなるようなデータがない場合には、OBB_generate_unique_id()ルーチンを利用して、識別子を生成できます。オブジェクトに割り当てる一意にユニークな識別子がある場合には、新たに識別子を割り当てる必要はありません。
2. インプリメンテーションに対するデフォルトのバインディング方針は自動バインディングです。自動バインディング以外のバインディング方針が必要な場合には、OBB_BOA_set_impl_binding()ルーチンを利用して、静的バインディングあるいは動的バインディングに変更します。
   OBB_BOA_set_impl_binding()ルーチンでは次のインプリメンテーションの設定を変更できます。

表 38 OBB_BOA_set_impl_binding()のパラメータ

	値
バインディング方針	OBB_BINDPOLICY_STATIC OBB_BINDPOLICY_AUTOMATIC OBB_BINDPOLICY_DYNAMIC
ロケーション	トランスポートファミリ:ノード名
既知の識別子	既知のサーバインスタンス識別子

バインディング方針を静的バインディングに変更する場合には、IMLあるいはインプリメンテーションの登録属性で、インプリメンテーションに対する既知の識別子を指定する必要があります。

3. CORBA_BOA_create()ルーチンを使用して、オブジェクトリファレンスを作成します。
boa —ORB_DEC_BOA_OBJECT
ev —エラー情報データへのポインタ
id —オブジェクトにユニークな識別子
intf —このインプリメンテーションオブジェクトをサポートするインターフェース定義。次の二種類の方法で取得できます。
   - サーバスケルトンファイルから取得します。次の形式で生成される定数です。
     IDLモジュール名_IDLインターフェース名_OBJ
   - OBB_Repository_lookup_fullname()ルーチンを利用して、インターフェースリポジトリからIDL定義を取得します。

   impl—インプリメンテーション定義。次の3種類の方法で取得できます。

   - インプリメンテーションの登録ルーチンからの戻り値がインプリメンテーション定義です。
   - サーバスケルトンファイルから取得します。次の形式で生成される定数です。
     IML-インプリメンテーション名_OBJ
   - OBB_Repository_lookup_fullname()ルーチンを利用して、インプリメンテーションリポジトリからIML ファイルで定義されたインプリメンテーション定義を取得します。

5.5.2 オブジェクトリファレンスの解放

1. CORBA_Object_release()ルーチンを利用して、オブジェクトリファレンスに関連つけられているリソースを解放すると、オブジェクトリファレンス自体はそれまで通り利用できます。
2. CORBA_Object_dispose()ルーチンを利用して、オブジェクトリファレンスを削除すると、削除されたオブジェクトリファレンス自体が利用できなくなります。オブジェクトを削除できるのはサーバだけです。

5.6 コンテキスト情報へのアクセス

サーバは2種類のコンテキスト情報を受け取ります。

- クライアントがメソッドの起動時に指定するコンテキスト情報
- IML定義ファイルでインプリメンテーション定義で指定されたコンテキスト情報

5.6.1. クライアントが起動時に指定したコンテキスト情報

クライアント環境のコンテキストオブジェクトに設定されているプロパティの中で、IDL ファイルのオペレーションステートメントで指定されているプロパティがサーバに渡されます。

5.6.2 IML定義ファイルでインプリメンテーション定義で指定されたコンテキスト情報

IML 定義ファイルのACTIVATION_CONTEXT ステートメントで指定されたプロパティとして、クライアント環境のコンテキストオブジェクトに設定されているプロパティが、サーバに渡されます。

サーバに渡されたコンテキストオブジェクトのプロパティには、次の手順でアクセスします。

1. CORBA_ORB_get_default_context()ルーチンを利用して、コンテキストオブジェクトハンドルを入手します。
2. コンテキストオブジェクトハンドルを、CORBA_Context_get_values()ルーチンで利用してコンテキストオブジェクトのプロパティにアクセスします。

コンテキストオブジェクトの値を変更する方法も 2 種類あります。

- プロセスごとのコピーされたコンテキストオブジェクトを変更
- ファイルに登録されたコンテキストオブジェクトを変更

5.6.3. プロセスごとにコピーされたコンテキストオブジェクトを変更

1. CORBA_Context_create_child()ルーチンを利用して、プロセスごとにコピーされたコンテキストオブジェクトハンドルを入手します。
2. プロパティを一つだけ設定する場合には、CORBA_Context_set_one_value()ルーチンを使用します。
3. 複数のプロパティを同時に設定する場合には、CORBA_ORB_create_list()ルーチンとCORBA_NVList_add_item()ルーチンを利用して、設定すべきプロパティの新しい値を指定したリストを作成します。作成したリストを使用して、CORBA_Context_set_values()ルーチンでコンテキストオブジェクトにプロパティを設定します。
4. コンテキストオブジェクトが不要になれば、CORBA_Context_delete()ルーチンでコンテキストオブジェクトを削除します。

5.6.4. ファイルに登録されたコンテキストオブジェクトの変更

1. OBB_ORB_open_context()ルーチンで、指定したコンテキストオブジェクトを参照するコンテキストオブジェクトハンドルを取得します。
2. プロパティを一つだけ設定する場合には、CORBA_Context_set_one_value()ルーチンを使用します。

3. 複数のプロパティを同時に設定する場合には、CORBA_ORB_create_list()ルーチンとCORBA_NVList_add_item()ルーチンを利用して、設定すべきプロパティの新しい値を指定したリストを作成します。作成したリストを使用して、CORBA_Context_set_values()ルーチンでコンテキストオブジェクトにプロパティを設定します。
4. コンテキストオブジェクトの変更が終了したら、OBB_Context_close()ルーチンでコンテキストオブジェクトをクローズします。

5.7. メソッドルーチンの開発

メソッドルーチンの開発がサーバの開発作業の大部分を占めます。ここではメソッドルーチンで必要なオブジェクトブローカに関する処理をついて説明します。

5.7.1. クライアントとのデータの入出力

5.7.1.1. 単純データ型 (short, long, unsigned short, unsigned long, float, double, boolean, char, octet, enum)

入力パラメータの場合には、値をそのまま参照します。

出力パラメータの場合には、サーバにはポインタが渡されます。渡されたポインタが指定している領域にデータを設定します。

入出力パラメータの場合にも、サーバにはポインタが渡されます。渡されたポインタが指定している領域をアクセスして、値を参照したり、データを設定します。

戻り値としては値をそのまま返します。

5.7.1.2 オブジェクトリファレンス型(Object)

入力パラメータの場合には、値をそのまま参照します。

出力パラメータの場合には、サーバにはポインタが渡されます。渡されたポインタはオブジェクトリファレンスを設定する領域を指定しています。メソッドルーチンは指定された領域にオブジェクトリファレンスを設定します。OBBarg_set_objref()ルーチンを使用して、メソッドルーチンが終了したときに設定したオブジェクトリファレンスを解放するように、基本オブジェクトアダプタに指定する必要があります。

入出力パラメータの場合にも、サーバにはポインタが渡されます。渡されたポインタを参照してオブジェクトリファレンスにアクセスします。メソッドルーチン

は指定された領域のオブジェクトリファレンスを修正することはできません。

戻り値の場合には、そのままオブジェクトリファレンスを返します。OBBarg_set_objref()ルーチンを使用して、メソッドルーチンが終了したときに設定したオブジェクトリファレンスを解放するように、基本オブジェクトアダプタに指定する必要があります。

5.7.1.3 最大長を指定した配列型(bound sequence)

入力パラメータの場合には以下のように参照します。

1. サーバにはポインタが渡されます。
2. _length フィールドには、項目数が設定されています。
3. _buffer フィールドには、データ配列の先頭アドレスが設定されています。

出力パラメータの場合には以下のように設定します。

1. サーバにはポインタが渡されます。
2. _maximum フィールドにはIDL 定義ファイルで指定した最大長を設定します。
3. _length フィールドにはメソッドルーチンで設定する項目数を指定します。
4. OBBarg_malloc()ルーチンを利用して、指定した大きさのデータ配列の領域を確保します。確保された領域はメソッドルーチンが終了すると解放されます。
5. 確保した領域にデータをセットします。
6. _buffer フィールドには、確保した領域へのアドレスを指定します。

入出力パラメータの場合には以下のように指定します。

1. サーバにはポインタが渡されます。
2. _length フィールドには、項目数が設定されています。
3. _buffer フィールドに設定されているデータ配列の先頭アドレスを参照して、データにアクセスします。
4. 項目数を変更する場合には、_length フィールドには、新しい項目数を設定します。
5. OBBarg_malloc()ルーチンを利用して、指定した大きさのデータ配列の領域を確保します。確保された領域はメソッドルーチンが終了すると解放されます。
6. 確保した領域にデータをセットします。
7. _buffer フィールドには、確保した領域へのアドレスを指定します。

戻り値の場合には以下のように参照します。

1. _maximum フィールドにはIDL 定義ファイルで指定した最大長を設定します。
2. _length フィールドにはメソッドルーチンで設定する項目数を指定します。
3. OBBarg_malloc()ルーチンを利用して、指定した大きさのデータ配列の領域を確保します。確保された領域はメソッドルーチンが終了すると解放されます。
4. 確保した領域にデータをセットします。

5. _buffer フィールドには、確保した領域へのアドレスを指定します。

5.7.1.4. 最大長を指定しない配列型(unbound sequence)

入力パラメータの場合には以下のように参照します。

1. サーバにはポインタが渡されます。
2. _length フィールドには、項目数が設定されています。
3. _buffer フィールドには、データ配列の先頭アドレスが設定されています。

出力パラメータの場合には以下のように設定します。

1. サーバにはポインタが渡されます。
2. _maximum フィールドには最大長を設定します。
3. _length フィールドにはメソッドルーチンで設定する項目数を指定します。
4. OBBarg_malloc()ルーチンを利用して、最大長として指定した大きさのデータ配列の領域を確保します。確保された領域はメソッドルーチンが終了すると解放されます。
5. 確保した領域にデータをセットします。
6. _buffer フィールドには、確保した領域へのアドレスを指定します。

入出力パラメータの場合には以下のように指定します。

1. サーバにはポインタが渡されます。
2. _length フィールドには、項目数が設定されています。
3. _buffer フィールドに設定されているデータ配列の先頭アドレスを参照して、データにアクセスします。
4. 最大長を変更する場合には、_maximum フィールドには新しい最大長を設定します。
5. 項目数を変更する場合には、_length フィールドには、新しい項目数を設定します。
6. OBBarg_malloc()ルーチンを利用して、最大長として指定した大きさのデータ配列の領域を確保します。確保された領域はメソッドルーチンが終了すると解放されます。
7. 確保した領域にデータをセットします。
8. _buffer フィールドには、確保した領域へのアドレスを指定します。

戻り値の場合には以下のように参照します。

1. _maximum フィールドには最大長を設定します。
2. _length フィールドにはメソッドルーチンで設定する項目数を指定します。
3. OBBarg_malloc()ルーチンを利用して、最大長として指定した大きさのデータ配列の領域を確保します。確保された領域はメソッドルーチンが終了すると解放されます。
4. 確保した領域にデータをセットします。
5. _buffer フィールドには、確保した領域へのアドレスを指定します。

5.7.1.5. 共用体型(union)

入力パラメータの場合には以下のように指定します。

1. サーバにはポインタが渡されます。

2. _d フィールドを参照して、指定されている型を読み取ります。
3. 設定されている型に対応するフィールドの値を参照します。

出力パラメータの場合には以下のように指定します。

1. サーバにはポインタが渡されます。
2. _d フィールドに型指定の値を設定します。
3. 設定した型に対応するフィールドに値を設定します。データ型がオブジェクトリファレンスの場合には、CORBA_BOA_create()ルーチンでオブジェクトリファレンスを作成します。作成したオブジェクトリファレンスは、メソッドルーチンが終了したときに解放されるようにOBBarg_set_objref()ルーチンで基本オブジェクトアダプタに知らせます。データ型がsequence あるいは string の場合には、OBBarg_malloc()ルーチンでメモリを確保します。

入出力パラメータの場合には以下のように指定します。

1. サーバにはポインタが渡されます。
2. _d フィールドを参照して、指定されている型を読み取ります。
3. 設定されている型に対応するフィールドの値を参照します。
4. 設定している型を変更する場合には、_d フィールドに新しい型指定の値を設定します。動的な型から非動的な型への変更、あるいは非動的な型から動的な型への変更はできません。
5. 設定した型に対応するフィールドに値を設定します。

戻り値の場合には以下のように参照します。

1. _d フィールドに型指定の値を設定します。
2. 設定した型に対応するフィールドに値を設定します。データ型がオブジェクトリファレンスの場合には、CORBA_BOA_create()ルーチンでオブジェクトリファレンスを作成します。作成したオブジェクトリファレンスは、メソッドルーチンが終了したときに解放されるようにOBBarg_set_objref()ルーチンで基本オブジェクトアダプタに知らせます。データ型がsequence あるいは string の場合には、OBBarg_malloc()ルーチンでメモリを確保します。

5.7.1.6 汎用型(any)

入力パラメータの場合には以下のように指定します。

1. サーバにはポインタが渡されます。
2. _type フィールドに設定されているデータのタイプコードを参照します。
3. CORBA_TypeCode_kind()ルーチンと CORBA_TypeCode_equal()ルーチンを利用して、データ型を判別します。
4. _value フィールドに設定されているデータにアクセスします。

出力パラメータの場合には以下のように指定します。

1. サーバにはポインタが渡されます。
2. _type フィールドにデータ型を設定します。
3. 設定するデータ型に対応する領域を確保します。データ型がオブジェクトリファレンスの場合には、CORBA_BOA_create()ルーチンでオブジェクトリファレンスを作成します。作成したオブジェクトリファレンスは、メソッドルーチンが終了したときに解放されるようにOBBarg_set_objref()ルーチンで基本オブジェクトアダプタに知らせます。データ型がsequence あるいは

string の場合には、OBBarg_malloc()ルーチンでメモリを確保します。
4. 確保した領域にデータを設定します。
5. _value フィールドにデータを設定します。

入出力パラメータの場合には以下のように指定します。

1. サーバにはポインタが渡されます。
2. _type フィールドに設定されているデータのタイプコードを参照します。
3. CORBA_TypeCode_kind()ルーチンとCORBA_TypeCode_equal()ルーチンを利用して、データ型を判別します。
4. _value フィールドに設定されているデータにアクセスします。
5. データ型を変更する場合には、_type フィールドに新しいデータ型を設定します。
6. 設定するデータ型に対応する領域を確保します。データ型がオブジェクトリファレンスの場合には、CORBA_BOA_create()ルーチンでオブジェクトリファレンスを作成します。作成したオブジェクトリファレンスは、メソッドルーチンが終了したときに解放されるようにOBBarg_set_objref()ルーチンで基本オブジェクトアダプタに知らせます。データ型がsequence あるいは string の場合には、OBBarg_malloc()ルーチンでメモリを確保します。
7. 確保した領域にデータを設定します。
8. _value フィールドにデータを設定します。

戻り値の場合には以下のように参照します。

1. _type フィールドにデータ型を設定します。
2. 設定するデータ型に対応する領域を確保します。データ型がオブジェクトリファレンスの場合には、CORBA_BOA_create()ルーチンでオブジェクトリファレンスを作成します。作成したオブジェクトリファレンスは、メソッドルーチンが終了したときに解放されるようにOBBarg_set_objref()ルーチンで基本オブジェクトアダプタに知らせます。データ型がsequence あるいは string の場合には、OBBarg_malloc()ルーチンでメモリを確保します。
3. 確保した領域にデータを設定します。
4. _value フィールドにデータを設定します。

5.7.1.7. 文字列型(string)

入力パラメータの場合は、サーバに渡されたポインタを利用して文字列にアクセスします。

出力パラメータの場合は、サーバに渡されたポインタにOBB_arg_malloc()ルーチンで確保した領域を設定します。確保した領域に文字列を設定します。

入出力パラメータの場合は、サーバに渡されたポインタを利用して文字列にアクセスします。文字列を変更する場合には、サーバに渡されたポインタにOBB_arg_malloc()ルーチンで確保した領域を設定します。確保した領域に文字列を設定します。

戻り値の場合に、OBB_arg_malloc()ルーチンで確保した領域に文字列を設定します。

5.7.1.8 配列型(array)

入力パラメータの場合には、サーバに渡されたポインタを利用して配列にアクセスします。

出力パラメータの場合は、サーバに渡されたポインタにOBB_arg_malloc()ルーチンで確保した領域を設定します。確保した領域に配列を設定します。

入出力パラメータの場合は、サーバに渡されたポインタを利用して配列にアクセスします。配列を変更する場合には、サーバに渡されたポインタにOBB_arg_malloc()ルーチンで確保した領域を設定します。確保した領域に配列を設定します。

戻り値の場合に、OBB_arg_malloc()ルーチンで確保した領域に配列を設定します。

5.7.1.9 構造体型(struct)

入力パラメータの場合には、サーバに渡されたポインタを利用して構造体にアクセスします。

出力パラメータの場合は、サーバに渡されたポインタを利用して構造体にデータをセットします。

入出力パラメータの場合には、サーバに渡されたポインタを利用して構造体にアクセスして、データを参照あるいは変更します。

戻り値の場合に、データを設定したポインタを返します。

5.8 終了処理ルーチンの開発

サーバプログラムでは、終了処理としてインプリメンテーションあるいはオブジェクトの非活性化をする必要があります。インプリメンテーションを非活性化する場合にはCORBA_BOA_deactivate_impl(),オブジェクトを非活性化する場合には、CORBA_BOA_deactivate_obj()を利用します。

また登録したインプリメンテーションを削除する必要もあります。インプリメンテーションの登録を削除するには、OBB_BOA_impl_unregister()を利用します。

5.9 サーバの構築

サーバを構築するには次の手順に従ってください。

5.9.1. サーバスケルトンファイルの生成

以下のファイルを生成します。

- ディスパッチャルーチンファイル
- インプリメンテーション登録ルーチンファイル
- メソッドテンプレートファイル

5.9.2 メソッドルーチンを作成

生成したメソッドルーチンをベースにして、メソッドルーチンを完成させます。

5.9.3 ソースファイルのコンパイル

以下のソースファイルをコンパイルします。

- メソッドルーチンファイル
- ディスパッチャルーチンファイル
- インプリメンテーション登録ルーチンファイル
- タイプコードファイル（外部形式の場合のみ）

ソースファイルをコンパイルする場合に必要なヘッダファイルは次のディレクトリにあります。

表 39 オブジェクトブローカヘッダファイルの位置

プラットフォーム/言語	ヘッダファイルの位置
Windows	オブジェクトブローカルートディレクトリのINCLUDE サブディレクトリ
WindowsNT	¥win32app¥obroker¥include
OSF/1,ULTRIX,SunOS,HP-UX,AIX	/usr/include
OpenVMS	SYSSLIBRARY

5.9.4 サーバのリンク

コンパイルしたファイルをリンクして、実行ファイルを作成します。

表 40 オブジェクトブローカライブラリファイル

プラットフォーム/言語	ライブラリファイルの位置
Windows	¥WINDOWS¥OBB.LIB
WindowsNT	¥win32app¥obroker¥lib¥obb.lib
OSF/1	/usr/lib/libobb.so
ULTRIX	/usr/lib/libobb.a /usr/lib/libobbtrn.a /usr/lib/liblmf.a /usr/lib/dnetstub.o(TCP/IP の場合)
Sun	/usr/lib/libobb.so.2.5
HP-UX	/usr/lib/libobb.sl
AIX	/usr/lib/libobb.a
OpenVMS	SYS$LIBRARY:OBB$SHARE.EXE

5.10. サーバのデバッグ

サーバは通常はエージェントから起動されるため、サーバからは何も出力されないことになってしまします。そのためサーバのデバッグを行なうには、次のコマンドでエージェントをデバッグモードに設定して、サーバからの出力をファイルに保存することになります。

OpenVMS:
APPL/BROKER SET AGENT/STARTUP_DEBUG
UNIXおよびWindowsNT:
obbmset -c -d1
出力されるファイルはホームディレクトリのOBBxxxxxxxx.logファイルになります。ここでxxxxxxxxはランダムに生成される１６進の文字列になります。

5.11. インプリメンテーションの登録

クライアントからリクエストがあった時点でサーバを起動する場合には、インプリメンテーションの情報をインプリメンテーションパーティションに登録する必要があります。インプリメンテーションの情報を登録するには、最初にインプリメンテーション定義ファイルを次のコマンドで生成します。

OpenVMS:
GENERATE IMPLEMENTATION/REGISTATION_FILE=ファイル名 インプリメンテーション名
UNIXおよびWindowsNT:
obbgen -a ファイル名 インプリメンテーション名
生成されたインプリメンテーション定義ファイルを、次のコマンドでインプリメンテーションパーティションに登録します。

OpenVMS:
REGISTER IMPLEMENTATION ファイル名
UNIXおよびWindowsNT:

obbreg -I ファイル名

次のコマンドで登録されているインプリメンテーションを表示することができます。

OpenVMS:
SHOW ADVERTISMENT [/IMPLEMENTATION インプリメンテーション名]
UNIXおよびWindowsNT:
obbshadv [-I インプリメンテーション名]

6. スクリプトメソッド

スクリプトサーバとは、UNIX システムのB-shell や OpenVMS システムのDCL などのスクリプト言語を利用して開発したスクリプトを、メソッドとして実行する仕組みです。

スクリプトメソッドを開発する手順です。

1. スクリプトを使用することが妥当であるかどうかを判断します。
2. インプリメンテーション定義を変更します。
3. スクリプトを作成します。

6.1. スクリプト使用の妥当性を確認

スクリプトサーバでは、通常のサーバプログラムに対して以下のような制約があります。

- 利用できるデータ型が以下のデータ型に制限されています。
  - short
  - unsigned short
  - long
  - unsigned long
  - Boolean
  - char
  - octet
  - float
  - double
  - string

  オブジェクトリファレンスや配列、構造体、列挙体、any などはサポートされていません。
- スクリプトサーバでは、クライアントがリクエストを送るたびに、新しいサーバプロセスが作成されます。サーバプロセスが、スクリプトのプロセスを作成します。このためプロセスの作成によるオーバーヘッドが必ず発生しますので、すばやいレスポンスが要求されるシステムを、スクリプトサーバで開発することは困難な場合があります。
- スクリプトサーバでは、クライアントアプリケーションがリクエストを送るために必要なオブジェクトリファレンスを作成することができません。

6.2 インプリメンテーション定義の修正

インプリメンテーション定義をスクリプトメソッドに対応するように、変更する必要があります。

1. activation_type で script を指定します。
2. activation_string を削除します。
3. activation_policy で server_per_method を指定します。
4. 定義している各メソッドごとに、invoke_builtin ステートメントを削除します。
5. 定義している各メソッドごとに、invoke_script で実行すべきスクリプトコマンドを指定します。スクリプトの内部で、オブジェクトブローカのコンビニエンスコマンドを利用する必要がある場合には、-c オプションを指定します。

6.3 コンテキストオブジェクトへのアクセス

オブジェクトブローカのコンビニエンスコマンドを利用して、コンテキストオブジェクト属性にアクセスできます。コンテキストオブジェクトにアクセスするには、次のコマンドを利用します。

OpenVMS:

GET PROPERTY プロパティ名

UNIX:

obbgetp プロパティ名

このコマンドには次のオプションを指定することができます。

表 41 コンテキストオブジェクトにアクセスするコマンドのオプション

	OpenVMS	UNIX	
シェル型	N/A	-c	C shellに対応したコマンドを出力します
コンテキストオブジェクトレベル	ユーザ /USER グループ /GROUP システム /SYSTEM	ユーザ -U グループ -G システム -S	各プロパティのアクセスごとにこの指定が必要です
コンテキストオブジェクトファイル名	/CONTEXT_OBJECT=ファイル名	-C ファイル名	ユーザ、グループ、システム以外のコンテキストオブジェクトにアクセスする場合に利用します
環境変数あるいはシンボル名	/VALUE=シンボル名	-V 環境変数名	デフォルトではプロパティと同じ名前の環境変数あるいはシンボルを設定します。異なる名前の環境変数あるいはシンボルに設定する場合に利用します
プロパティデータ型	/DATA_TYPE=シンボル名	-d 環境変数名	プロパティのデータ型を参照する場合に利用します
プロパティの値の数	/COUNT=シンボル名	-N 環境変数名	プロパティに設定されている値の数を参照する場合に利用します
参照するプロパティのインデックス	/INDEX=インデックス	-i インデックス	複数の値が設定されているプロパティを参照する場合に、最初以外に設定されている値を参照する場合に利用します
フェイルオーバー機能	/FAILOVER	-n	指定したプロパティが、指定したコンテキストオブジェクトで見つからない場合に利用します
状態コードの保存	/STATUS=シンボル名	-s 環境変数名	デフォルトのステータスコードを変更したくない場合に利用します

UNIX ではコマンドの出力を、スクリプトファイルに書き込みます。このスクリプトファイルを評価して環境変数にコンテキストオブジェクトのプロパティの値を設定します。

OpenVMS のコマンドプロシージャでは、プロパティの値が直接シンボルに設定されます。

6.4. 入出力の制御

スクリプトメソッドは、ユーザと直接の入出力をおこなうことができません。スクリプトからの出力は、ファイルにリダイレクトされない場合、消えてしまうことになります。スクリプトメソッドの出力が必要な場合には、ファイルに書き込む必要があります。

UNIX のシェルスクリプトでは、>あるいは>>で出力をファイルに変更することができます。invoke_script ステートメントあるいはスクリプトファイル内部でファイルを出力先に指定します。

OpenVMS のコマンドプロシージャでは、/OUTPUT=ファイル名で出力をファイルに変更できます。invoke_script でコマンドプロシージャではなく、コマンドをしている場合には、出力をファイルに変更する方法はコマンドごとに異なります。

6.5 スクリプトでのパラメータアクセス

クライアントからのパラメータに、スクリプトメソッドがアクセスするには、次の 2 種類の方法があります。

- 置換指示文を利用
- コンビニエンスコマンドを利用

置換指示文を利用する方法では、クライアントから渡されたパラメータを参照できるだけです。コンビニエンスコマンドを利用する方法では、クライアントから渡されたパラメータを参照するほかに、クライアントにパラメータを返すこともできます。

6.5.1. 置換指示文を利用

置換指示文を起動コマンドの一部に置き換えておくことで、実行時に入力パラメータあるいは組み込み関数値が起動文字列に組み込まれます。置換指示文には次の 2 種類があります。

- 引数置換(%'引き数名'あるいは%'引き数番号')
  置換文字と、その後の単一引用符あるいは二重引用符で囲んだ引き数名あるいは引き数番号を指定すると、実行時に引き数の値に置き換えることがなります。
- 関数置換(%関数名)
  置換文字と、その後の関数名を指定すると、オブジェクトブローカの組込み関数を参照することになります。現在、提供されている関数です。

表 42 置換指示文の関数

関数名	処理内容
RefData	オブジェクトリファレンスのリファレンスデータ。スクリプトメソッドでは文字列として扱います
ImplementationId	インプリメンテーションのUUID
MethodId	メソッドのUUID

置換指示文に関する規則

- 置換文字のデフォルトは%です。IML 定義で置換文字を変更することができます。invoke_script に設定する文字列で置換文字列自身が必要な場合似には、置換文字列を重ねます。
- 引き数名あるいは引き数番号は引用符で囲む必要があります。引き数番号は1から始まります。
- スクリプトの内部で、パラメータは一度も参照されなくても、また何回も参照されてもかまいません。
- 無効なパラメータ指定の場合には、空文字列が代入されます。

6.5.2 コンビニエンスコマンドを利用

6.5.2.1. パラメータを参照

次のコマンドでスクリプトメソッドはクライアントからのパラメータを参照することができます。

OpenVMS:
GET ARGUMENT 引き数名
UNIX:
obbgeta-引き数名
このコマンドには次のオプションを指定することができます。

表 43 パラメータを参照するコマンドのオプション

	OpenVMS	UNIX	
シェル型	N/A	-c	C shellに対応したコマンドを出力します
環境変数あるいはシンボル名	/VALUE=シンボル名	-V 環境変数名	デフォルトではパラメータと同じ名前の環境変数あるいはシンボルを設定します。異なる名前の環境変数あるいはシンボルに設定する場合に利用します
パラメータデータ型	/DATA_TYPE=シンボル名	-d 環境変数名	パラメータのデータ型を参照する場合に利用します
引き数フラグ	/FLAGS=シンボル名	-f 環境変数名	引き数フラグを参照する場合に利用します
状態コードの保存	/STATUS=シンボル名	-s 環境変数名	デフォルトのステータスコードを変更したくない場合に利用します

UNIX ではコマンドの出力を、スクリプトファイルに書き込みます。このスクリプトファイルを評価して環境変数にパラメータの値を設定します。

OpenVMS のコマンドプロシージャでは、パラメータの値が直接シンボルに設定されます。

6.5.2.2 パラメータを設定

次のコマンドでスクリプトメソッドは、クライアントにパラメータを返すことができます。

OpenVMS:
SET ARGUMENT 引き数名
UNIX:
obbseta 引き数名
このコマンドには次のオプションを指定することができます。

表 44 パラメータを設定するコマンドのオプション

	OpenVMS	UNIX	
シェル型	N/A	-c	C shellに対応したコマンドを出力します
環境変数あるいはシンボル名	/VALUE=シンボル名	-V 環境変数名	デフォルトではパラメータと同じ名前の環境変数あるいはシンボルから設定します。異なる名前の環境変数あるいはシンボルから設定する場合に利用します
パラメータデータ型	/DATA_TYPE=シンボル名	-d 環境変数名	パラメータのデータ型を指定する場合に利用します
状態コードの保存	/STATUS=シンボル名	-s 環境変数名	デフォルトのステータスコードを変更したくない場合に利用します

6.6 戻り値の設定

次のコマンドでスクリプトメソッドはクライアントへの戻り値を設定することができます。

OpenVMS:
SET STATUS 戻り値
UNIX:
obbsets 戻り値
このコマンドには次のオプションを指定することができます。

表 45 戻り値を設定するコマンドのオプション

	OpenVMS	UNIX	
シェル型	N/A	-c	C shell に対応したコマンドを出力します
状態コードの保存	/STATUS=シンボル名	-s 環境変数名	デフォルトのステータスコードを変更したくない場合に利用します

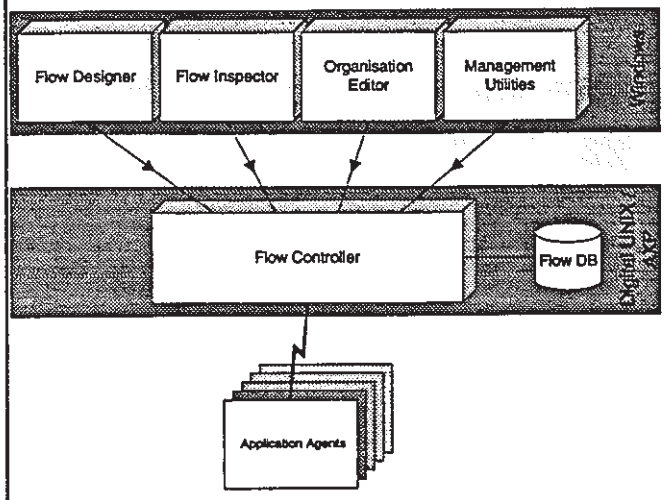
付録G　　ワークフロー・システム

EXTRA

Application Integration using Distributed Object Computing

digital

Workflow System

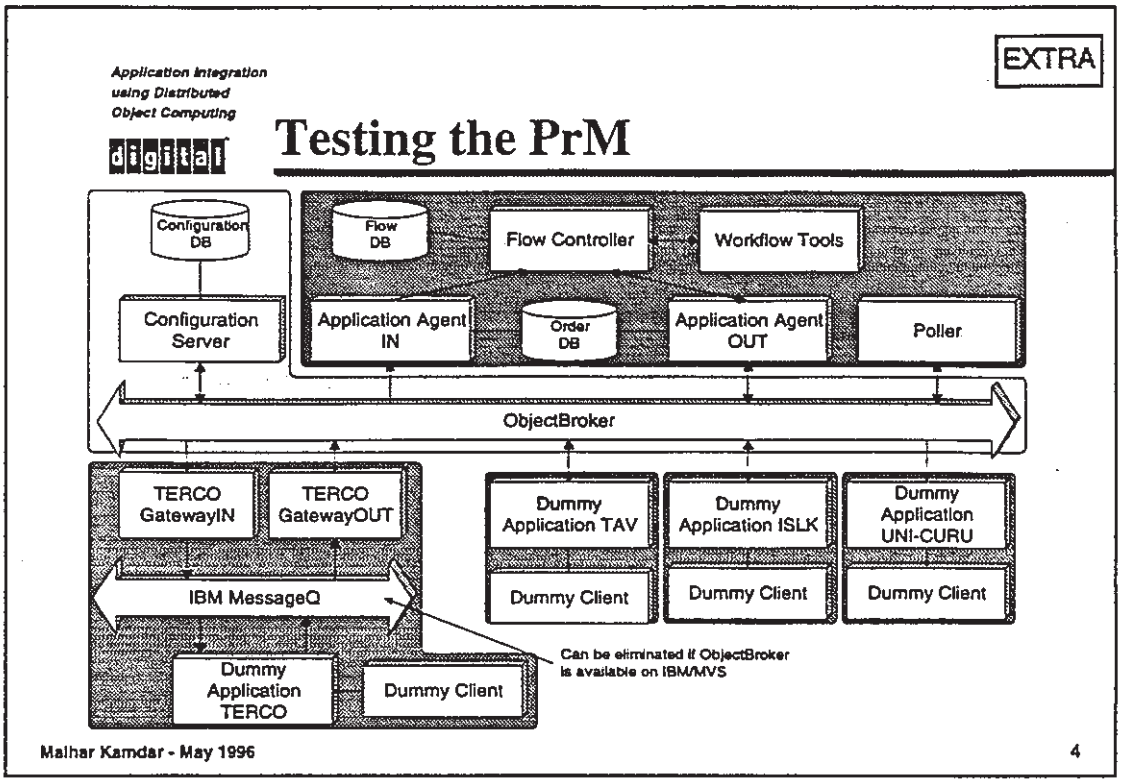


- Designer
  - Design tool for workflows
- Manager
  - Analyzing flows, tracking flows, etc.
- Organization editor
  - Define and modify organizational data used by workflow system
- Management utilities
  - Installation of Flows
- Flow Controller
  - Workflow engine
- Application agents
  - Acts as Client to the Workflow-System

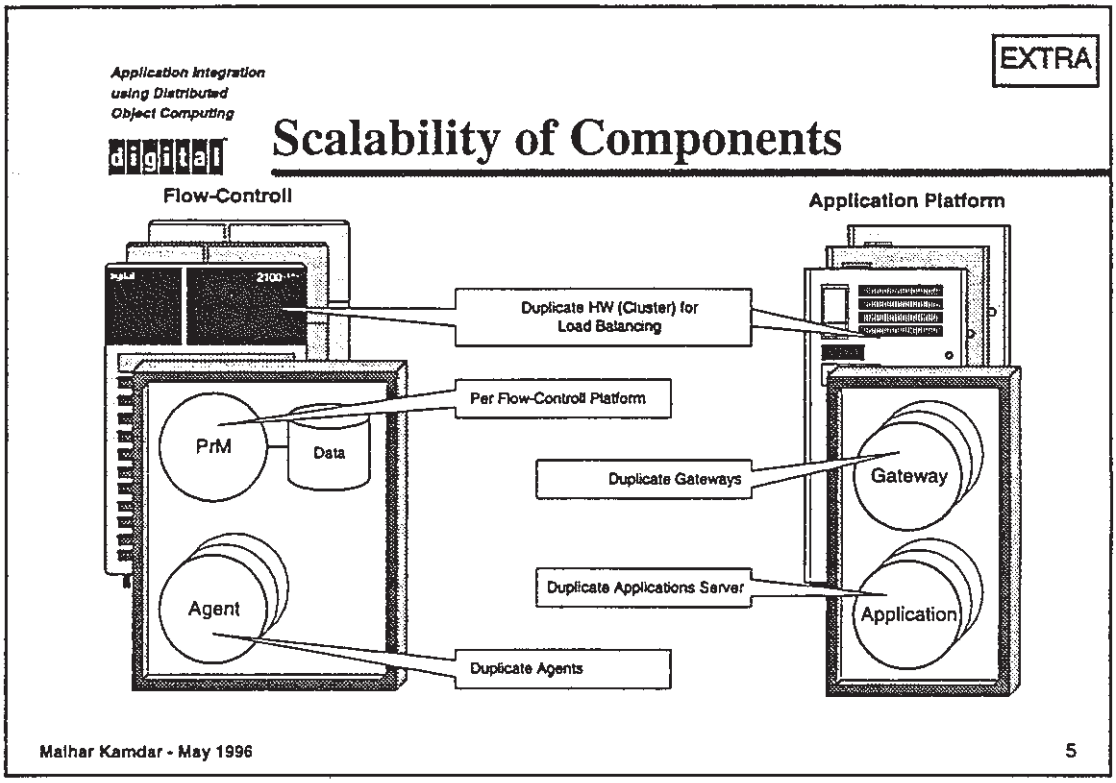
Malhar Kamdar - May 1996

3

ワークフロー


Application Integration using Distributed Object Computing
digital
EXTRA
Testing the PrM
Configuration DB
Configuration Server
Flow DB
Flow Controller
Workflow Tools
Application Agent IN
Order DB
Application Agent OUT
Poller
ObjectBroker
TERCO GatewayIN
TERCO GatewayOUT
IBM MessageQ
Dummy Application TERCO
Dummy Client
Dummy Application TAV
Dummy Client
Dummy Application ISLK
Dummy Client
Dummy Application UNI-CURU
Dummy Client
Can be eliminated if ObjectBroker is available on IBM/MVS
Malhar Kamdar - May 1996
4

ＰｒＭのテスト



コンポーネントの拡張性

Application Integration
using Distributed
Object Computing

digital

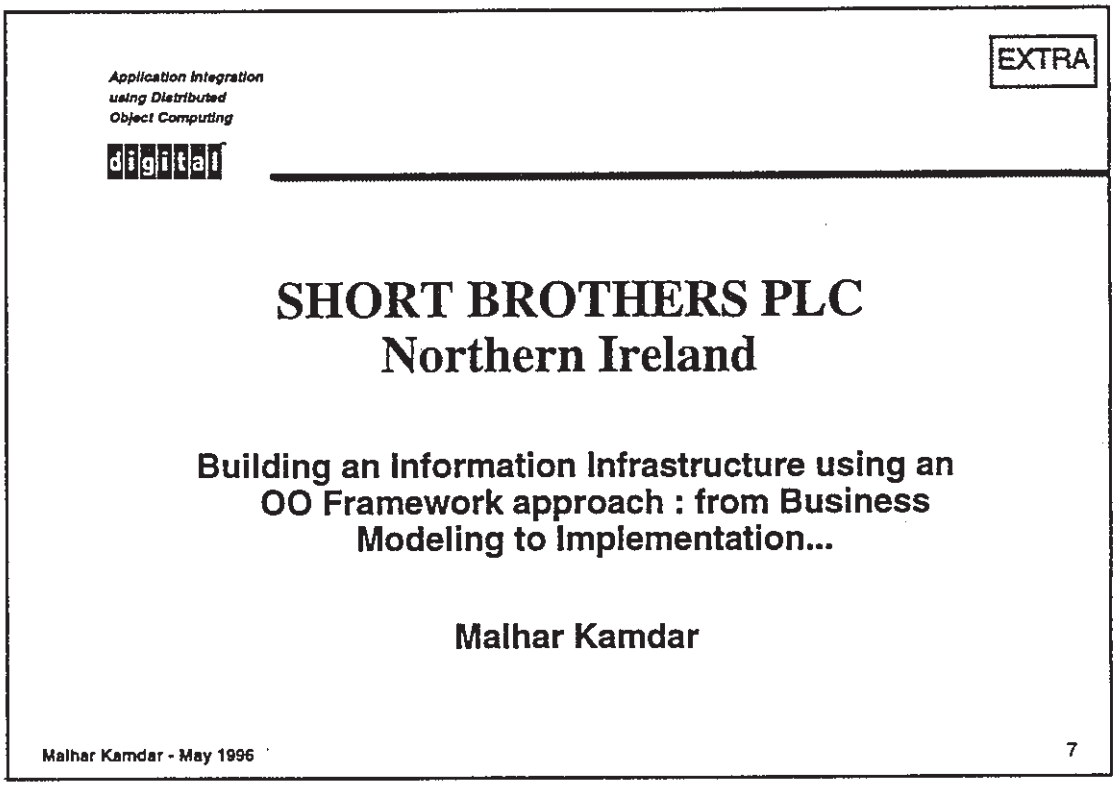
EXTRA

Next Steps

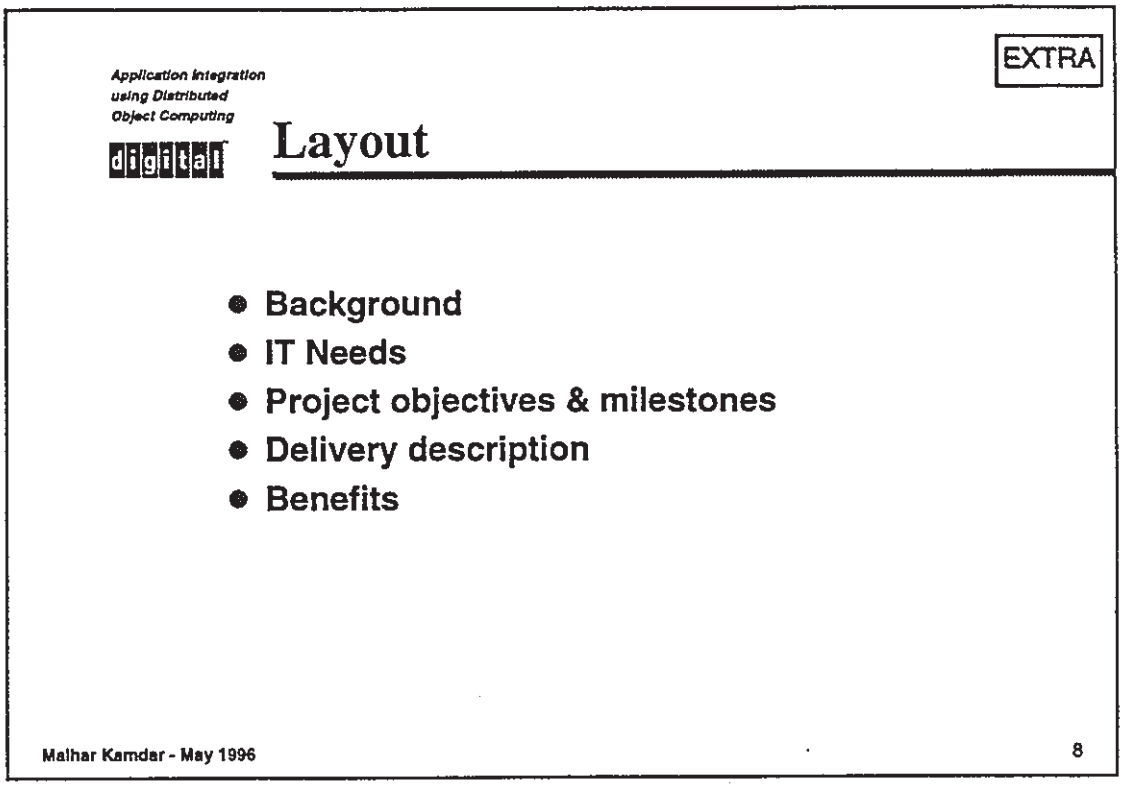
- Organizational changes :
  - IT staff re-skilling
  - New organization to reflect the new environment
- Extend this environment and approach to :
  - Other business processes
  - The commercial applications of Telecom
  - Postal division
- Apply a common methodology from Business Object Modeling to implementation : FBE (Framework-based Environment)

Malhar Kamdar - May 1996 6

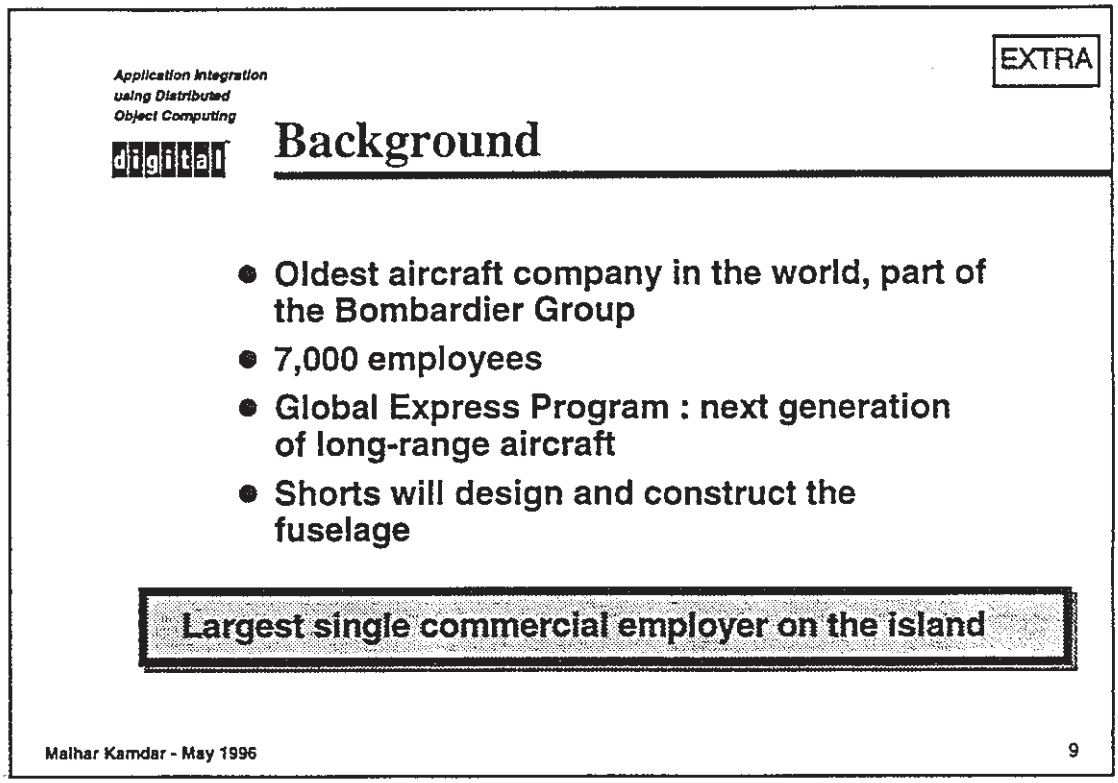
次のステップ
組織の変革
－ ＩＴスタッフの再教育
－ 新しい環境を反映した新しい組織
この環境とアプローチの展開
－ 他のビジネス・プロセス
－ テレコムの商用アプリケーション
－ 郵便部門
ビジネス・オブジェクトのモデリングから実装まで共通の方法論を適応する：ＦＢＥ



北アイルランドのSHORT BROTHERS PLCの事例
オブジェクト指向のフレームワーク・アプローチを使用した情報インフラの構築
－ ビジネス・モデリングから実装まで



レイアウト
背景
ＩＴのニーズ
プロジェクトの目的とマイルストーン
成果物の説明
利点



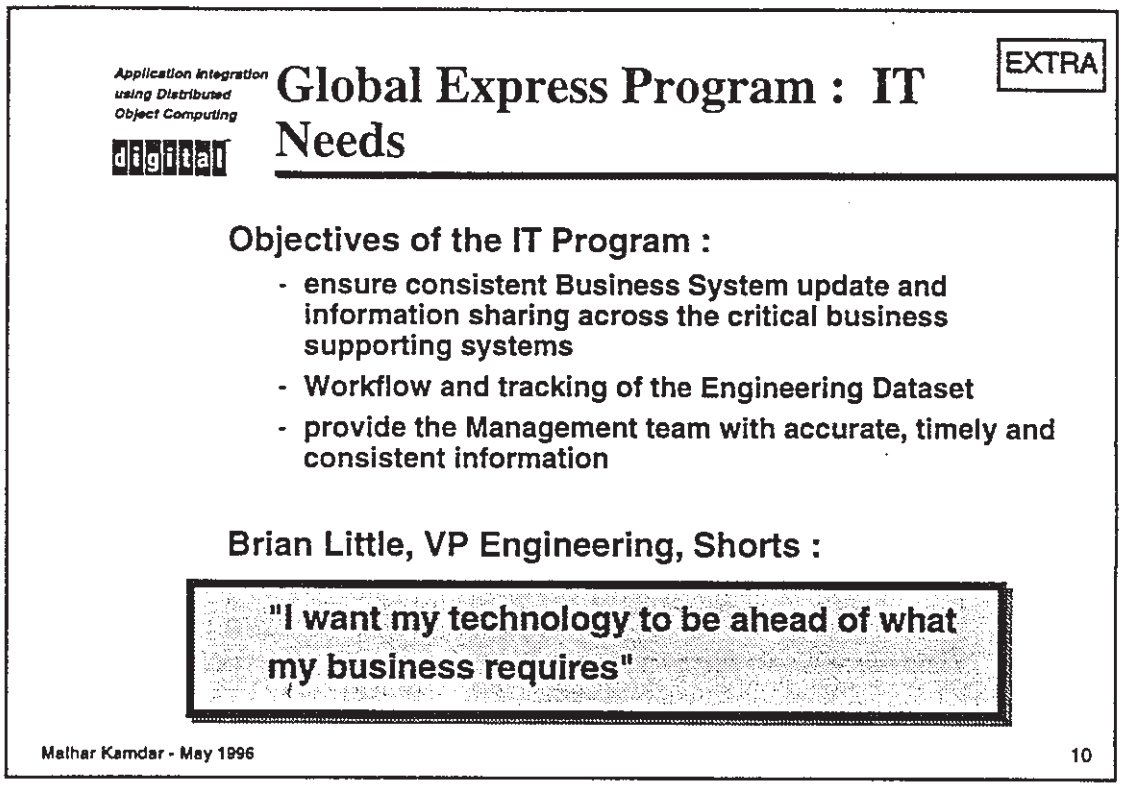
背景

世界で最古の飛行機会社、ボンバディエ・グループの一部である

７０００人の従業員

グローバル・エクスプレス計画：次世代の長距離飛行機

ショーツ社は機体の設計と組み立てを行う



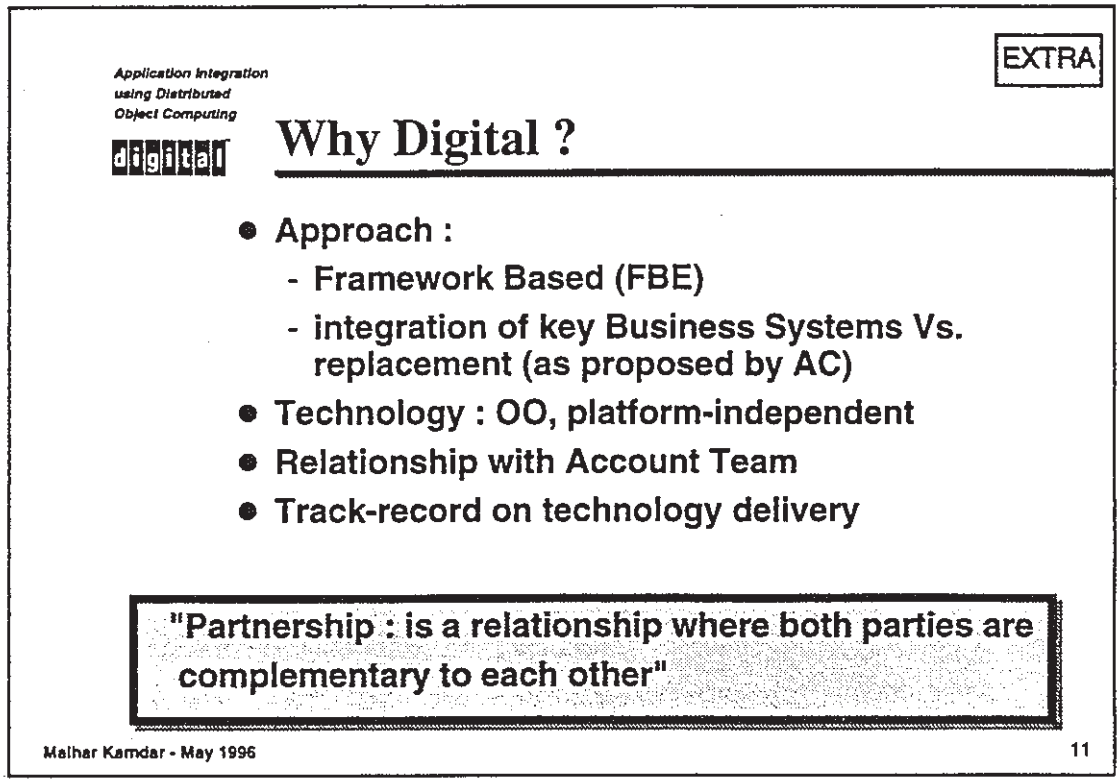
グローバル・エクスプレス計画：ＩＴのニーズ

ＩＴプログラムの目的

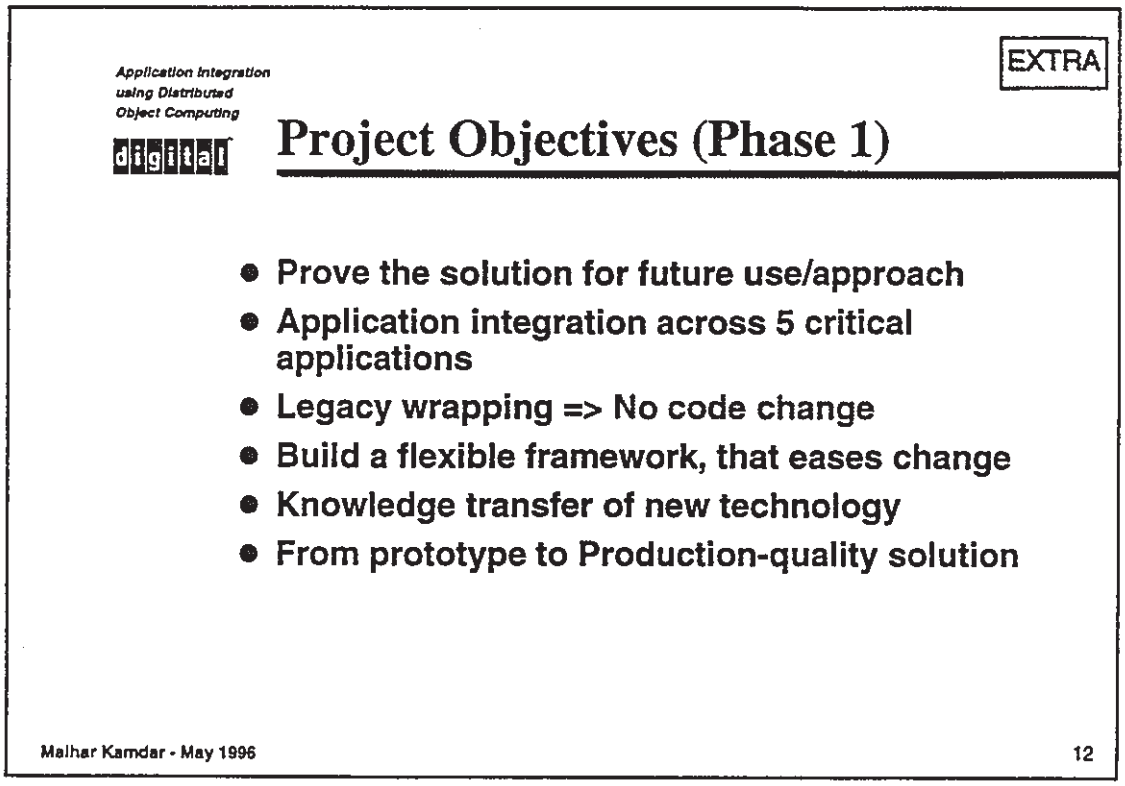
－ ビジネス・システムのアップデイトの整合性と重要なビジネス・サポーティング・システム間の情報の共有を保証する

－ エンジニアリング・データセットの追跡とワークフロー

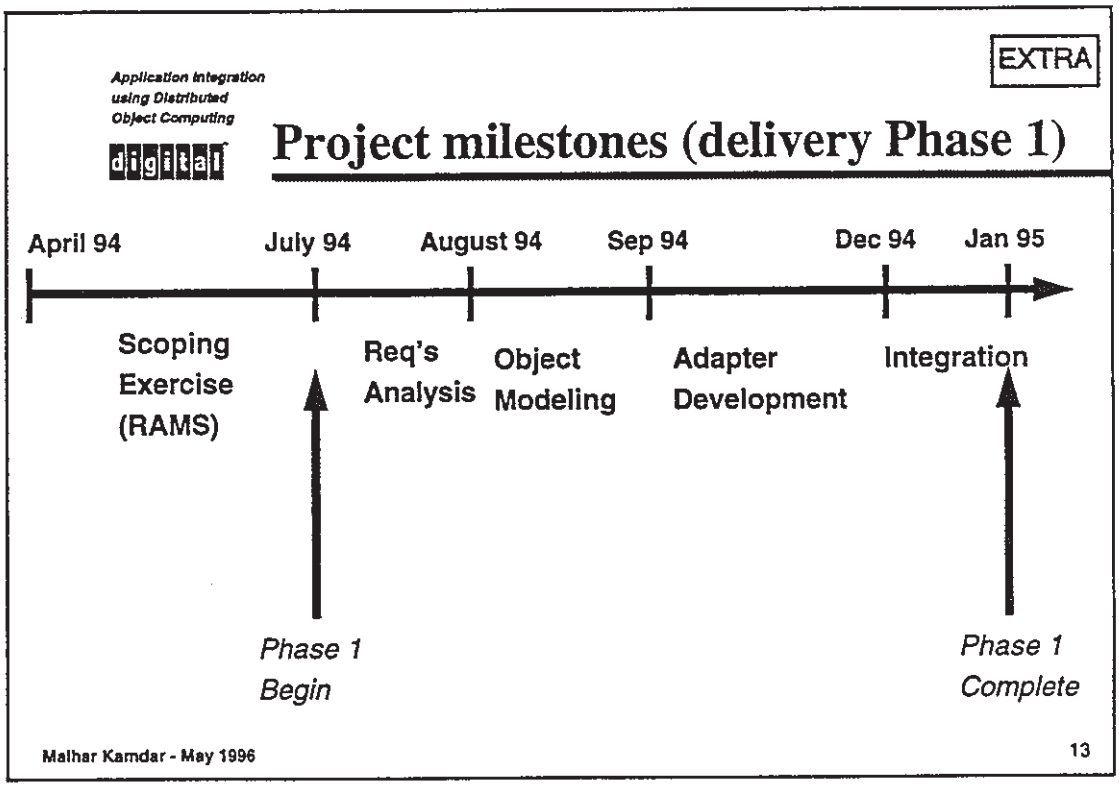
－ マネージメント・チームに対して正確で、タイムリーで、矛盾のない情報を供給する



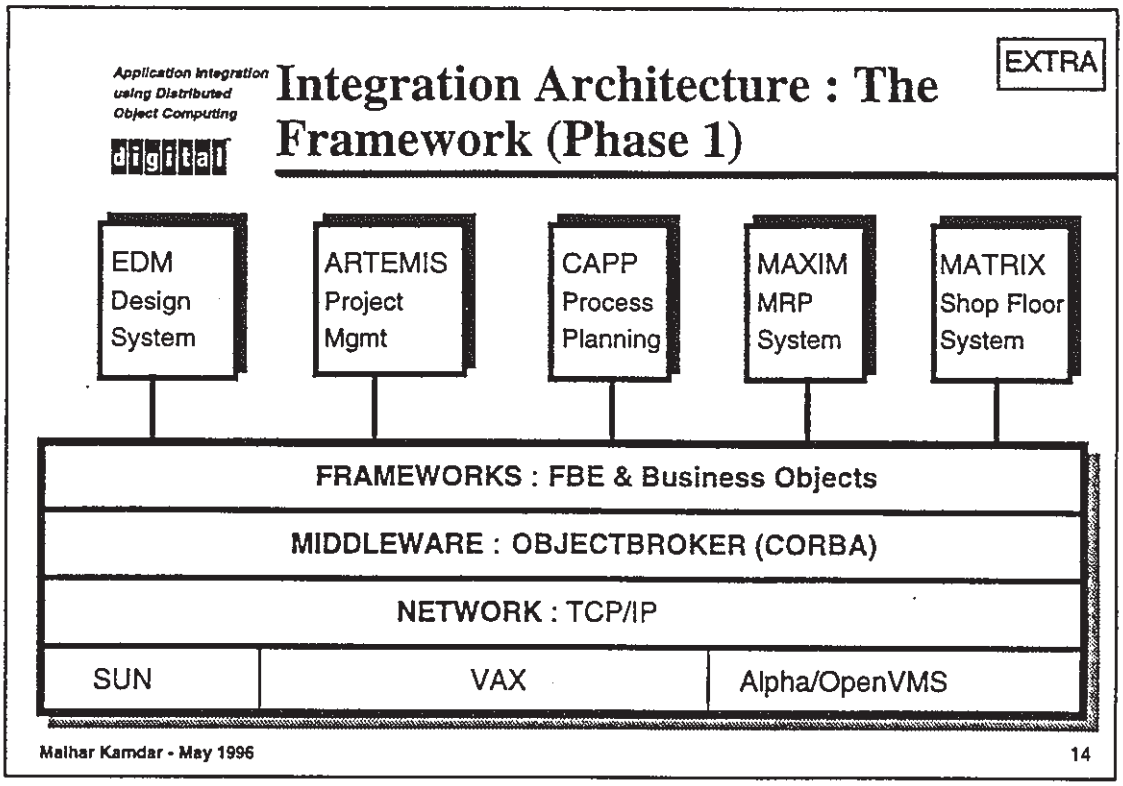
なぜＤＥＣか？
アプローチ：
－ フレームワーク・ベース（ＦＢＥ）
－ 重要なビジネス・システムの統合 ＶＳ リプレース（ＡＣ社により提案された）
テクノロジ： オブジェクト指向、プラットフォーム独立
アカウント・チームとの関係
テクノロジ伝達の記録



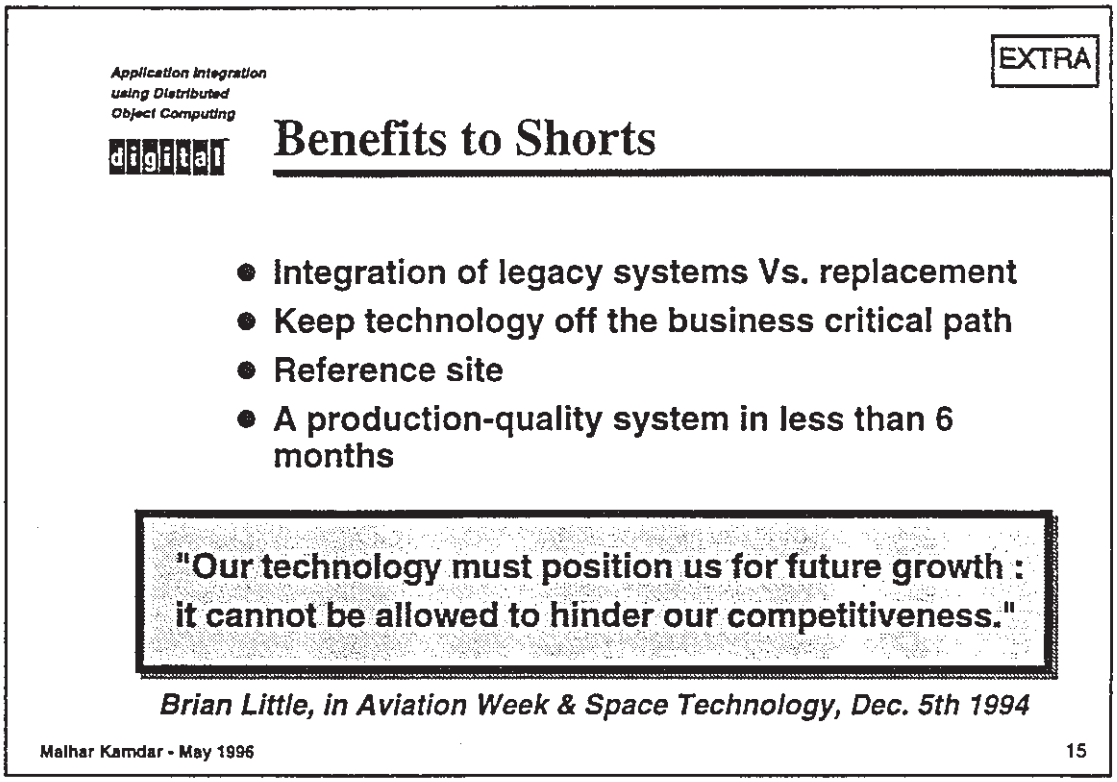
プロジェクトの目的（フェーズ１）
ソルーションの将来の可用性と／アプローチを証明する
五つの重要なアプリケーションの統合
レガシー・アプリケーションのラッピング → コードは変更しない
変更が容易な、柔軟なフレームワークを構築する
新しいテクノロジの知識を移管する
プロトタイプから製品レベルの品質のソルーションまで



プロジェクトの線表（フェーズ1）



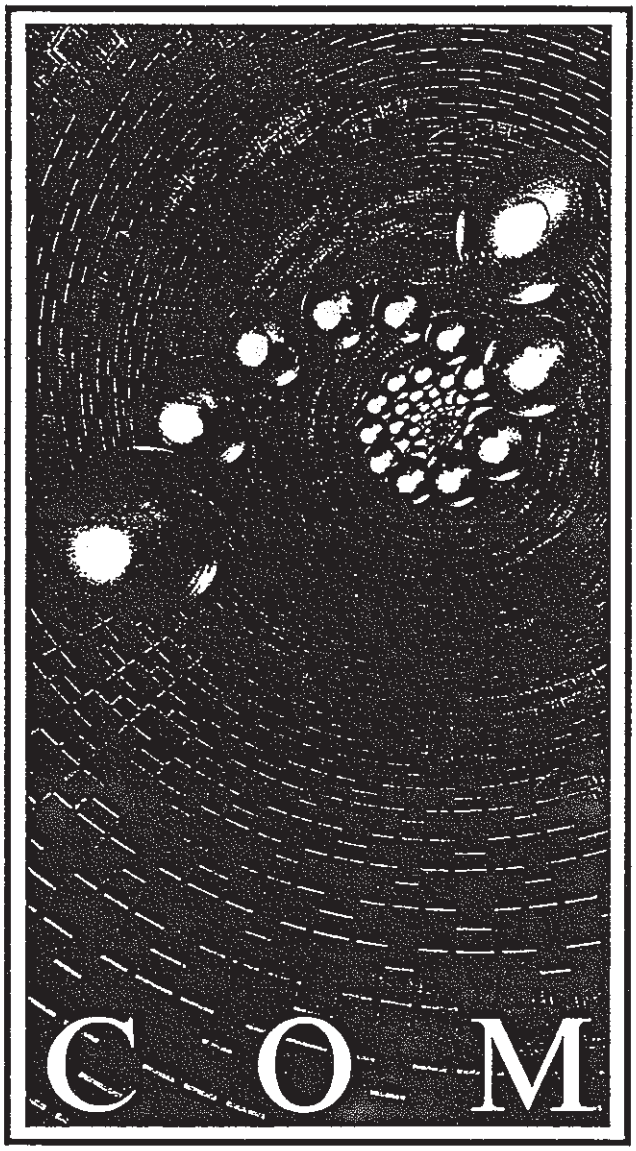
統合アーキテクチャ：フレームワーク（フェーズ1）



ショーツ社にとっての利点
レガシー・システムの統合 ＶＳ リプレース
ビジネスの重要な部分にテクノロジを近づけないでおく
リファレンス・サイト
６ヶ月以内で製品レベルの品質のシステム

付録H　　CommonObjectModel
分散オブジェクト環境の統合

digital



Common Object Model

分散オブジェクト環境の統合

CONTENTS

1 現代のソフトウェア開発と分散オブジェクト

今日のビジネス活動はあらゆる場面でコンピュータ・システムの支援を必要としています。現代社会のあらゆる側面にコンピュータ・システムが浸透した背景には、ハードウェアの低価格化やクライアント/サーバ・コンピューティングなどに代表される急速な技術革新が継続して行われてきたことがあげられます。

それではコンピュータ・システムの将来に問題は何もないのでしょうか。現実のコンピュータ・システムを例にとって少し詳しく調べてみましょう。

■

1つはある企業が社内で開発した業務アプリケーション・システムです。このアプリケーションはメインフレーム用にCOBOLで開発されました。今となってみるとユーザ・インターフェイスも古めかしく、ビジネス環境の変化に対応できていません。しかしながらシステムの改良や再構築を行うだけのリソースがないため、結局ユーザに負担をかけながらも古いシステムを使いつづけています。

もう1つは、あるソフトハウスが開発したビジネス向けパッケージ・ソフトです。ユーザからの要求にあわせて様々なプラットフォームへのポーティングと機能改良を続けてきましたが、現在ではそれが非常な負担となってバージョン・アップもままなりません。その上、このパッケージを他の業種へ横展開しようと考えていますが、既に作成済みのモジュールも手直しが必要なため結局簡単に再利用することはできそうにありません。

■

このようにハードウェアによるコンピュータ・システムの技術革新が急速な成長を続けている半面で、ソフトウェア開発の生産性はあまり向上していないようです。なぜこのようなことが起こるのでしょうか。

例えば、コンピュータ・ハードウェアの開発と比較してみましょう。コンピュータ・ハードウェアのような複雑なシステムを開発する場合にはシステムを機能分割して、それぞれの構成要素を部品化します。このように部品化を行うことによって複雑なシステム開発が容易に分業化できるとともに、一度開発した部品の再利用が可能になり、全体として大幅な生産性の向上が図れるわけです。

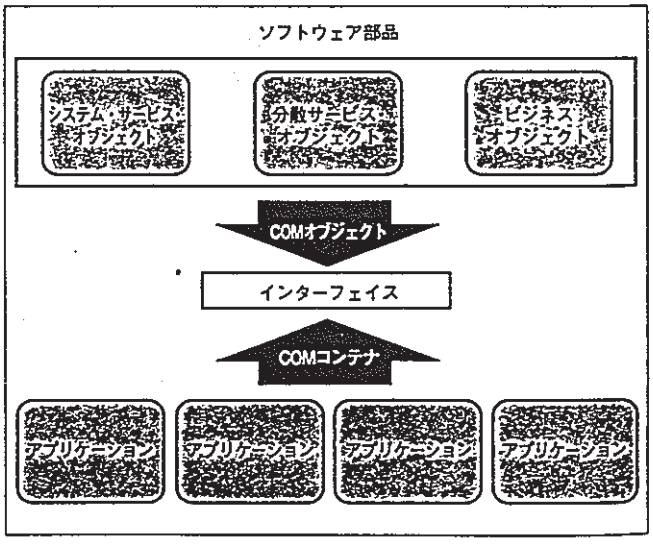
複雑なシステム開発にはこのように部品化という手法が非常に有効なのですが、ソフトウェア開発ではこの部品化が徹底して行われていないところに問題がありそうです。これはソフトウェア開発者が部品化の重要性を認識していなかったわけではなく、ソフトウェア・システムにおいて部品化を実現する基盤が存在しなかったからなのです。

部品化を可能にするためには、部品を提供する側と利用する側の間に部品の使い方に関する合意が必要になります。コンピュータ・ハードウェアのように物理的なシステムの場合、このような合意を取り決めることは比較的容易ですが、ソフトウェアのような抽象的な論理体系の場合、開発にあたっての自由度が高すぎるため合意をすることが困難になっているのです。

■

このことを現実の状況でみてみましょう。例えば、企業の中の開発部隊の場合はどうでしょうか。ソフトウェア開発にあたっては構造化手法やモジュール化を標準的に実施しており、モジュールの再利用も意識されてはいます。しかしながら実際の開発にあたって1つのプロジェクトや小さなグループの中ではモジュールの再利用は行えても、全社的に部品化や再利用を実現することは非常に困難です。同様にソフトハウスでも社内的なモジュールの再利用は少しずつ行っていますが、ソフトウェア部品自体を販売・流通させたり、他社の部品を利用したりすることはとてもできません。

なぜこのようになってしまうのか、その理由を考えてみますと、部品を供給する側と利用する側が日常的に情報交換を行えるような小さなグループでは現状でもある程度部品化は可能ですが、企業全体やソフト産業全体を考える場合、従来のソフトウェア開発手法では実務レベルでの部品化や再利用は困難なことがわかります。つまり企業全体やソフト産業全体で認知された標準的な部品化の規格が必要なわけです。

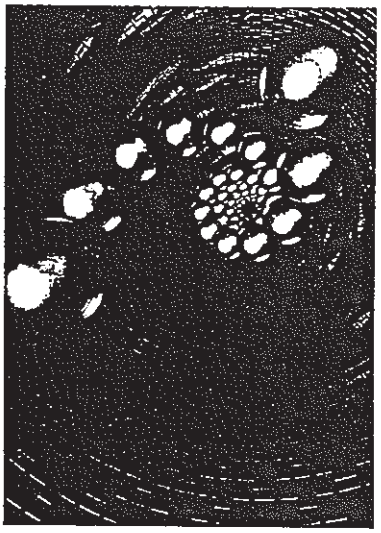


このような状況に対応するために、Microsoft社はWindows環境で部品化を実現するための標準技術OLE(Object Linking and Embedding)を提唱しています。
OLEでは部品を提供する側をオブジェクト、部品を利用する側をコンテナと呼び、オブジェクトとコンテナの間の合意にあたるインターフェイスを標準的に規定しています。標準的なインターフェイスを規定することによって、OLEに準拠して開発されたオブジェクトとコンテナであれば、問題なく部品化が実現され、ソフトウェア・システム開発の生産性を向上することが可能になるわけです。

■

それではこのOLEの採用によってソフトウェア開発に関するすべての問題は解決するのでしょうか。もう一度現実の環境に戻って確かめてみましょう。
ある企業の業務アプリケーションの場合はどうでしょうか。確かにOLEを採用することによりWindowsで業務アプリケーションを簡単に開発することはできそうです。しかしながらメインフレームで開発した既存のアプリケーションをすべてWindowsで再構築するしかないとしたら、問題の解決にはなりません。またOLEがWindows NTやMacintoshには対応できるとして、UNIXやオフコンは対応されないとしたら、全社レベルのシステムの基盤にはできません。
次に、あるソフトハウスのパッケージ・ソフトの場合を考えてみると、OLEの採用によって開発済みのソフトウェアの部品化が体系的に実現でき、再利用性も高まって、ソフトウェア開発の生産性向上は実現できそうです。しかしながら現在のOLEでは分散環境への対応ができないため、パッケージ・ソフトのクライアント/サーバ化は実現できないことになります。

このようにOLEによってソフトウェアの部品化が実現され、ソフトウェア開発の生産性向上は見込めます。しかしながら現在のコンピュータ・システムを特徴づける2つのトレンド、分散化とマルチプラットフォーム環境への対応が充分ではないため、大規模で本格的なシステム構築に全面的に採用することはできません。
逆にいえば、現代のコンピュータ・システムの開発を支援するためには、OLEの提供する技術を分散化したマルチプラットフォーム環境へ拡張する分散オブジェクト技術の確立が急がれているわけなのです。



2 COMとは何か

2.1 目的

DECとMicrosoft社は1993年にOLEを拡張する分散オブジェクト技術を共同で開発することに合意し、その技術をCOM(Common Object Model)と名付けました。

COMがなぜ開発されるのか、その必然性については前章で説明しましたが、ここではその目的をまとめておきましょう。

■

(1)分散化への対応

ここ数年クライアント/サーバ・コンピューティングが1種のトレンドとなっています。最近のコンピュータ・システムは従来のように1台の大型コンピュータを大勢で利用する集中形態から、一人または小人数で利用するコンピュータをネットワークで接続する分散形態へ変化してきております。このような分散形態のコンピュータ・システムに対応するためにソフトウェア・システムも複数のコンピュータ上のアプリケーションが相互に通信しあいながら協調して業務を遂行する形態、つまりクライアント/サーバ・コンピューティングを実現することが求められているのです。

■

OLEとその基盤となっているオブジェクト技術は本来的に分散化に対応できる能力をもっています。OLEに準拠したインターフェイスはオブジェクトとコンテナという別々のプロセスを接続できるように規定されています。インターフェイスがこのように規定されていれば、それをネットワーク越しに接続できるようにするのはそれ程困難なことではありません。問題はできるだけオープンで標準的な実現方式を決定することです。

COMではオープンで標準的な通信の基盤としてOSF/DCEのRPCを採用しました。オープン・システムの標準技術を開発するために主要なシステム・ベンダーが設立したOSF(Open System Foundation)がオープンな分散環境を実現するための様々な基盤技術を1つにパッケージ化したものがDCE(Distributed Computing Environment)です。その中でもRPC(Remote Procedure Call)は分散環境でアプリケーション同士の相互通信を実現する最も基礎的な技術です。

COMの基盤としてRPCを採用したのは、オープンで標準的な通信基盤を提供できることとともに、RPCが既に様々なプラットフォームで安定して提供されていること、更にRPCによってプラットフォームやネットワーク・プロトコルの違いを意識する必要がなくなることが評価されたからです。

(2)マルチプラットフォーム環境のサポート

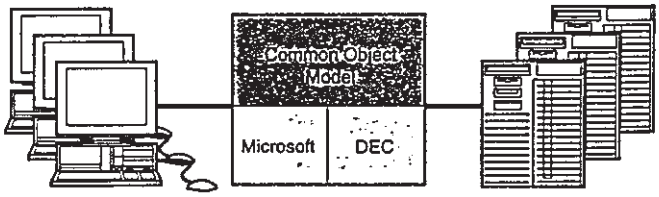
現在のクライアント/サーバ・コンピューティングを特徴づけるもう一つの側面がマルチプラットフォームです。一昔前ならば一種類かせいぜい二種類のプラットフォームを知っていればそれで充分でした。しかしながら最近では普通の職場でも様々な種類のコンピュータが使われています。現実に様々な種類のプラットフォーム上のアプリケーション同士が接続されることが必要になってきています。

■

OLEは元々Windows環境のために開発されてきた技術で、現在ではMacintoshやWindows NTをサポートしていますが、大規模なビジネス・システムを実現するためにはもっと多くのプラットフォームをサポートする必要があります。特にその中でもUNIXプラットフォームが重要です。UNIXプラットフォームはWindows NTとならんでOLEのオブジェクト側の機能を提供するプラットフォームとして位置付けられます。特に最近ではクライアント/サーバ・システムにおいてデータベース・システムのような高い性能と高い信頼性を要求されるサーバ・システムとしてますます重要性が高まっています。

COMの技術開発にあたってはOLEオブジェクトを実行するUNIXサーバ機能を提供することを目的として、OLEの環境をUNIX上でも提供します。つまりWindowsのGUIに依存しないOLEの機能をUNIX上でも提供することによって、OLEオブジェクトになるソフトウェアをUNIX上でWindows環境と同じ様に開発することが可能になるわけです。

さらにCOM自体はプラットフォームに依存しないように設計されていますので、これまで述べた以外のプラットフォームをサポートすることも可能です。



ここまで説明してきた2つの点、分散化への対応とマルチプラットフォームのサポートが前章でも述べたように分散オブジェクト技術の最も重要な課題であり、COMはその課題を解決することを目的としていることがおわかりいただけたかと思います。

しかしながらCOMにおいてはこの2つの課題以外にいくつかの重要な課題を解決することを目指しています。その中で重要なものを次に説明しておきましょう。

(3)様々なプログラム言語のサポート

プラットフォームと同様に様々なプログラム言語をサポートすることも現実のソフトウェア開発環境を考えた場合、重要です。例えばオブジェクト指向言語だけ取り上げてもC++とSmalltalkが考えられますし、一般的に利用されている言語を考えるとC, COBOL, FORTRAN, PL/Iなど非常に多くの言語が対象になってきます。

言語のサポートを考える場合重要なことは、OLEのオブジェクト側とコンテナ側で異なったプログラム言語を使っても問題なくインターフェイスが取れることです。COMではオブジェクトとコンテナの間のインターフェイスをバイナリ・レベルで規定していますので、個別の言語毎にインターフェイスを変える必要はありませんし、言語の仕様に影響を与えることなく言語サポートを行うことができます。

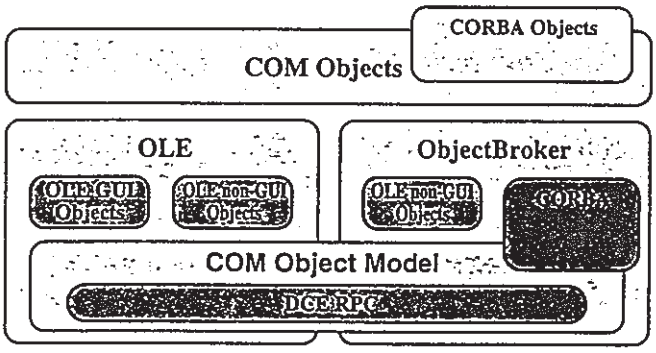
(4)他の分散オブジェクト環境との接続

COM/OLEの技術はプラットフォームやプログラム言語に依存しないものですからこの2つをすべての環境で提供すれば、コンピュータ・システムを構成するすべての環境がCOM/OLEによって実現できることになります。しかしながら現実的にはその様な状況が近い将来に実現するとは考えられません。なぜならCOM/OLEと同じく分散オブジェクト環境を実現できる他の技術標準が存在するからです。

その中で最も有力なものがCORBAに基づいた技術です。CORBAとは分散オブジェクト技術の標準化を推進することを目的にベンダー側とユーザ側の企業によって設立されたOMG(Object Management Group)が策定した分散オブジェクトを実現する基盤の規格であるCommon Object Request Broker Architectureのことです。

最近ではUNIXシステムを提供しているシステム・ベンダーを中心にしてCORBAに基づいた分散オブジェクト環境が実現されはじめています。現実の大規模なコンピュータ・システムを考える場合にはCORBAによる分散オブジェクト環境を考慮に入れる必要があります。

COMの技術開発においてはCOMとCORBAの間で双方向の通信が行えることが課題の1つになっており、このことによってCOMに対応したソフトウェアはCORBAのオブジェクトを利用することができますし、CORBAのアプリケーションからの要求を受け付けることも可能になるわけです。



■

2.2 役割

ここまでの説明でCOMがなぜ開発されなければならないか、その目的がおわかりいただけたと思います。次にCOMがコンピュータ・システムにかかわる人にとってどのように役立つか、その役割を説明していきます。

■

(1)アプリケーション開発者

アプリケーション開発をサポートすることがCOMの最大の目的なのですから、アプリケーション開発者にとってのCOMの役割も明確です。もう一度繰り返して述べるならば、分散化されたマルチプラットフォームなコンピュータ環境においてソフトウェアの部品化を実現することにより、クライアント/サーバ・システムでのソフトウェア開発の生産性の向上を実現する技術がCOMなのです。

ここでのキーポイントは生産性の向上にあります。ハードウェアの生産性と比較して非常に小さな向上しか実現できていないソフトウェアの生産性を大幅に上昇させることの可能な技術がCOMなのです。

■

この生産性の向上ということについて開発者の立場から注意しなければならない点があります。それはCOMのような分散オブジェクト技術を採用したからといって直ちにソフトウェア開発の生産性が向上するというものではないことです。ソフトウェアの部品化と再利用の拡大を通じて行われる生産性の向上は非常に長期間にわたって継続して実施する必要のあるプロセスなのです。

このような生産性の向上をできるだけ早くアプリケーション開発者が達成できるようにするために、システム・ベンダーやソフトウェア・ベンダーはすでに汎用的な分散オブジェクトの提供や分散オブジェクトを利用したシステム開発を支援するツールの提供を開始しています。COMの技術開発においてもこのような汎用的な分散オブジェクトや開発ツールの提供がプランに入っています。

もう一つ重要なことは、COMは様々なプラットフォームに対して統一的なソフトウェア開発環境を提供できることです。このことを開発者の側からみると、自分の技術力やシステムの要求仕様にあわせて自由にプラットフォームを選択できることを意味します。また1つのプラットフォームで開発したアプリケーションは他のプラットフォームから自由にその機能を利用できることになり、アプリケーションの利用範囲を大幅に拡大することが可能になるのです。

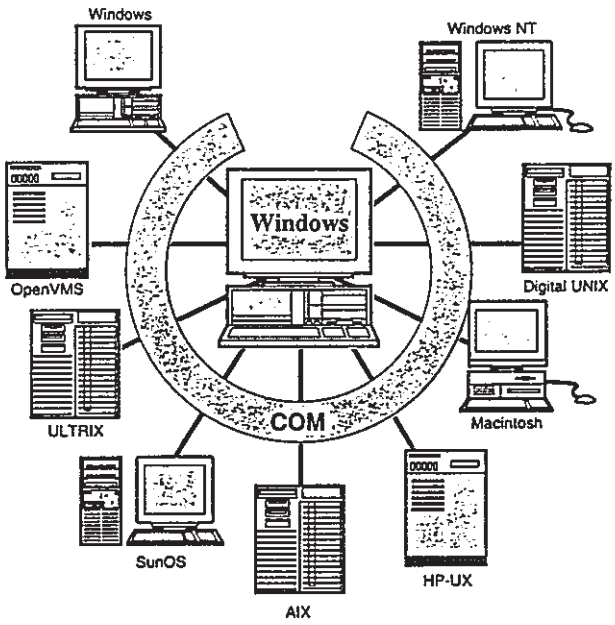
(2)エンド・ユーザ

これまで説明してきたことからすると、エンド・ユーザにとってCOMはソフトウェア開発にのみ関連した無関係なものなのでしょうか。そうではありません。確かにCOMの最大の目的はソフトウェア開発者を支援することですが、その機能はエンド・ユーザにとっても役立つものなのです。

最近では非常に容易なプログラミング環境(例えばVisual Basicなど)やプログラミングを必要としないアプリケーション開発環境(例えばスクリプト言語など)が提供されはじめています。COMのような分散オブジェクト技術は高度な機能を非常に容易なインターフェイスで提供することによってエンド・ユーザがアプリケーション開発を行う可能性を更に高めることができます。

特にOLEのコンパウンド・ドキュメントの機能によって、それと気付かないうちに分散オブジェクトの機能を利用したアプリケーションを日常業務の中で作成し、利用することが可能になるのです。

またCOMを利用したソフトウェアが市場に広まることによって、エンド・ユーザがソフトウェアを選択する幅が広くなります。例えばこれまでは1つの業務を支援するアプリケーションはあるソフトハウスが開発したパッケージ・ソフトで全部まかなう必要がありましたが、COMによってある機能はあるソフトハウスの部品を他の機能は別のソフトハウスの部品というように選択して組み合わせることが可能になるのです。このような部品化が進めばより安い価格でより高性能な業務アプリケーションを組み立てられる可能性が生まれてきます。



(3)経営者

企業の経営者からみたとき、COMはどれほどの意味をもつのでしょうか。現代のようにコンピュータ・システムがビジネスにとって必須のツールとなった場合、コンピュータ・システムへの投資は当然増加する傾向にあります。ダウンサイジングはこのようなコンピュータ投資が本当に投資対効果に見合ったものであったのかという疑問を提起し、見直しを図る動きであったといえます。

ハードウェアの分野ではこれに対応して、小型化、低価格化が進行しています。その半面ソフトウェアの全コストに占める比率が相対的に拡大する傾向にあるわけです。COMによるソフトウェア開発の生産性向上はソフトウェアにかかるコストの削減を可能にします。これは自社内の開発コストの削減とパッケージ・ソフトウェアの低価格化による購入コストの削減という2つの側面から可能になるわけです。

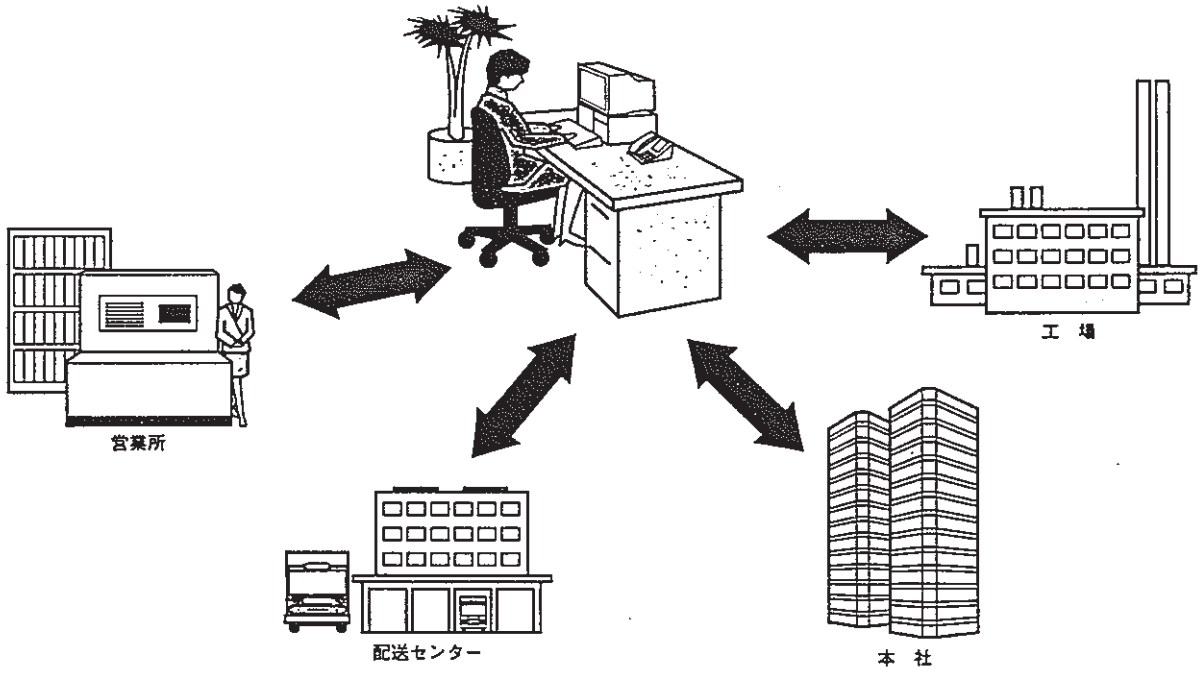
更にこのようにコンピュータ・システムへの一般的な投資の抑制が可能になったことにより、自由になったリソースをより戦略的なシステムの構築へ振り向けることが可能になります。現代のようにコンピュータ・システムとビジネス活動が一体となって新しいビジネス・プロセスを構築することが求められるようになってくると、戦略的なシステム構築に投資できないことは企業にとって致命的な問題になりかねません。逆にコンピュータ投資の削減によって戦略的なシステム構築へリソースが配分でき、そのシステム構築のコスト自体も削減できるわけですから、COMの役割は企業経営者にとっても重要であるといえるわけです。

(4)ソフトウェア・ベンダー

ソフトウェア・ベンダーにとってのCOMの役割はアプリケーション開発者のところで述べたことで充分なように思われるかもしれません。確かにソフトウェアの開発生産性が向上すれば、ソフトウェア生産原価の最大の項目である開発コストを削減することが可能になります。また様々なプラットフォームで統一的な開発環境が提供されれば、ソフトウェアのポーティングが容易になりますし、ソフトウェアの部品化の推進によって新しい市場への展開も可能になってきます。

しかしこのようなこのような効果は1社が独占的に享受できるものではありません。すべてのソフトウェア・ベンダーがCOMによって実現される新たな可能性を追求しはじめれば今まで以上の競争が行われることになります。

またCOMによるソフトウェア開発環境はこれまでのソフトウェア産業の構造を変革する潜在的な可能性を秘めています。少なくとも分散オブジェクトによる部品を提供する企業とその部品を組み合わせてビジネス・ソルーションを構築する企業の分化が起こるでしょう。またユーザによるアプリケーション構築が容易になるわけですから、従来のSIベンダーやソフトハウスの機能では不十分になり、より高度な専門的な能力を要求されることになるでしょう。



3 COMの機能と構造

ここまで述べてきたようにCOMはこれからのソフトウェア開発にとって非常に重要な役割を持つ技術です。それではCOMの技術そのものはどのようにしてその役割を実現するのでしょうか。COMの内部構造をここで簡単に説明していきます。

■

3.1 COMインターフェイス・モデル

COMとはソフトウェア開発の生産性を向上するために、ソフトウェアを分散オブジェクトという部品によって構成する技術です。部品化を実現するためには部品に関する厳密な規格を決めておく必要があります。例えばボルトやナットを部品化する場合で例えてみると、形状や寸法などをあらかじめ定めておく必要があります。このような標準規格の部品によって部品を作る側と利用する側がいちいち取り決めをしなくても安心して部品を流通させることが可能になるわけです。

■

分散オブジェクトのようにソフトウェアを部品化する場合において、ボルトやナットの形状や寸法にあたる部分は分散オブジェクトのインターフェイスになります。

個別のサービスを提供する分散オブジェクトのインターフェイスは当然それぞれ異なってきますが、共通部品として必要な最小限の部分の標準規格をインターフェイス・モデルといいます。

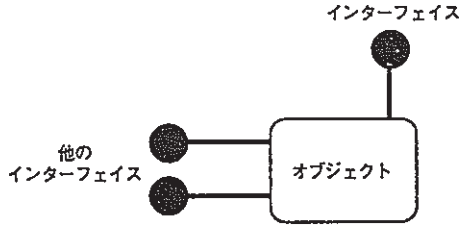
COMインターフェイス・モデルではコンテナからオブジェクトを利用するときのインターフェイスを規定します。分散オブジェクトがこの規定に従ったインターフェイスを提供することによって、どのようなコンテナからでもその分散オブジェクトを利用することが可能になるのです。つまりソフトウェアの部品化が実現されるわけです。

■

COMにおけるオブジェクトとコンテナのインターフェイスはバイナリ・レベルで規定されています。これはソースコード・レベルのインターフェイスでは分散オブジェクトの普及が従来のようにせいぜい1つのグループ内に留まってしまうからです。バイナリ・レベルのインターフェイスによって分散オブジェクトを商品として流通させることが可能になるのです。

このインターフェイスを定義し、インターフェイスに対応したソースコードを生成するツールとしてIDL(Interface Definition Language)とIDLコンパイラが提供されます。ただしIDLはCOMにおいては便利な道具であるにすぎません。COMに従ったソフトウェア開発はIDLを利用しなくとも可能であるからです。このことはCOMのインターフェイスがバイナリ・レベルで規定されているから可能になっているのです。

インターフェイス・モデルで次に重要なことはどのようにしてオブジェクトのインターフェイスを知るかということです。オブジェクトを使用するためにはオブジェクトのインターフェイスだけ知っていればいいのですが、非常に多くのオブジェクトのインターフェイスをすべて知っていることは困難ですし、ましてオブジェクトが商品として流通するようになった場合、事前にそのインターフェイスに対応しておくことは不可能です。このためCOMではすべてのオブジェクトが備えていなければならない共通のインターフェイスを規定し、このインターフェイスからそのオブジェクトに実際のサービスを利用するためのインターフェイスを検索できるようになっています。またこの共通インターフェイスではオブジェクトがいつまで有効でなければならないかを管理するための機能も提供しています。



3.2 分散サブ・システム

COMに準拠したオブジェクトの実現方式としてはイン・プロセス、ローカル・プロセス、リモート・プロセスの3つに分類されます。イン・プロセス方式は動的に起動されるライブラリと同じと考えられ、最も高いパフォーマンスを実現できる方式です。ローカル・プロセス方式は同じマシン上の別のアプリケーションがサービスを提供する方式で、例えばWORDとEXCELの間におけるようにWindowsアプリケーション間でサービスを提供しあうことが可能になります。リモート・プロセス方式はローカル・プロセス方式を分散マルチプラットフォーム環境へ拡張したものです。

■

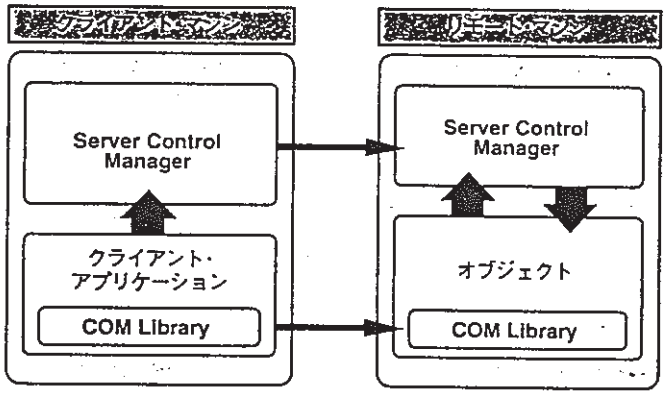
OLEを拡張するというCOMの目的からはこの内のリモート・プロセスによるオブジェクトの実現をサポートする分散サブ・システムの機能が重要です。もちろんオブジェクトを利用する側からはこの3つの実現方式の違いはわからないようになっている必要があります。

クライアントであるコンテナ・アプリケーションからオブジェクトへ要求が送られるとき、そのインターフェイスはCOMインターフェイス・モデルで規定されたものです。この要求はまずCOMのライブラリへ渡されてそこで実現方式にしたがってそれぞれ必要な処理が行われた上で実際のサービスの提供が行われることになります。

それではどのような処理が行われるのか、もう少し詳細にみていきましょう。
まずCOMライブラリの中ではSCM(Service Control Manager)に問い合わせることによって、要求を実行するオブジェクトがどこにいるかを決定し、そのオブジェクトと通信できるように準備を行います。つまりSCMは分散オブジェクトの電話番号案内と通話の接続を行う交換機の機能を兼ね備えたものと考えることができます。
SCMは相手のオブジェクトの実現方式によって異なった情報を提供しますので、それにしたがって、COMライブラリが処理を行います。イン・プロセスの場合はそのままオブジェクトを実現しているライブラリが処理を続行します。イン・プロセスでない場合、COMライブラリがクライアント側のプロセスでオブジェクトの代理となるProxyを生成し、このProxyを経由して処理が実現されるわけです。

■

ローカル・プロセスの場合は、1つのSCMですべての処理が実現できますが、リモート・プロセスの場合、相手先のSCMと協力して準備を行う必要があります。いずれにしてもSCMによる準備ができた後は相手のオブジェクト・プロセスと直接通信して、処理を行うことになります。
このプロセス間の通信を実現するために、COMではDCEのRPCを使用します。RPCを利用する理由は前にも述べた通り、プラットフォームやネットワーク・プロトコルに依存せずにプロセス間通信を実現できることです。



3.3 コンパウンド・ファイル

COMにおいて分散オブジェクトは、サービスを提供する役割とともにデータを伝達する役割を果してもいます。現在の情報処理においては、OLEのコンパウンド・ドキュメントに代表されるように、1つのドキュメントが画像、スプレット・シート、図表のような様々な形式のデータから構成され、それを1つのオブジェクトとして処理することが要求されます。

■

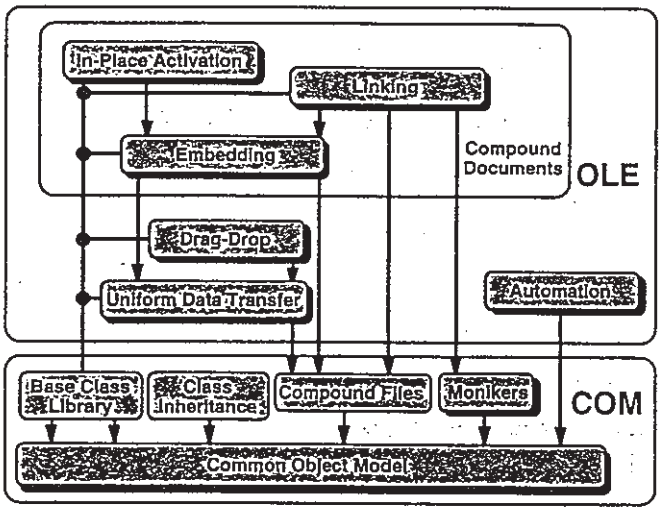
COMでは、このような複雑な構造の情報を取り扱うために、コンパウンド・ファイルを提供しています。
コンパウンド・ファイルでは1つのファイルの中に1つのファイル・システムを包含することが可能ですので、コンパウンド・ドキュメントを自然な形で1つのファイルとして扱うことができます。

■

3.4 モニカ

モニカはオブジェクトに名前をつけるためのCOMの標準的な方式です。モニカによって、COMの分散オブジェクトを名前で指定することが可能になるとともに、指定されたオブジェクトの中味や、利用方法についての情報を名前から入手することができます。モニカはこのように高機能な名前を提供する一種のオブジェクトなのです。

このほかにDECが独自に追加する機能には次のようなものがあります。



3.5 クラス・モデル

COMではインターフェイス・モデルによって分散オブジェクトのインターフェイス部分の再利用は容易に実現できます。しかしながらオブジェクトの実行コードを再利用するための手段はありません。一般的にオブジェクト・システムでは実行コードの再利用はクラス継承で実現されています。

このようにCOMにはクラス間の継承を実現する機能はありません。しかしながらオブジェクトの再利用性を更に高めるためにはクラス継承を実現することが望ましいと考えられていますのでDECはCOMの中でこのようなクラス継承を実現するクラス・モデルをオプションとして提供します。

分散オブジェクトにおけるクラス継承は継承されるべきベースとなるオブジェクトが分散しているため、これまでのオブジェクト・システムで実現されている方式では実現するのは困難です。

このためCOMのクラス継承では子供のクラスのオブジェクトを実現するときに実際は必要に応じて親になるクラスのオブジェクトが処理を行うことによってクラス継承を実現しています。オブジェクトへのインターフェイスはバイナリ・レベルで規定されているため、実際にどのオブジェクトが処理を行っているかはクライアント側からは知ることはできません。逆に親と子のオブジェクトが異なったプラットホームで異ったプログラム言語で実現されていてもクラス継承が可能なのです。

■

3.6 ベース・クラス・ライブラリ

COMによるソフトウェア開発を容易にするためには前に述べたように汎用的なオブジェクトはあらかじめライブラリとして用意しておくことが必要です。特にDECではCOMに対応した汎用的なオブジェクトをベース・クラス・ライブラリとしてサポートするプラットフォーム上に用意します。現在のところ主な対象はUNIXプラットフォームです。

■

3.7 CORBAとの結合

COMとCORBAという異なった分散オブジェクト環境が存在する以上、それらの間で相互に通信しあう手段を提供する必要があります。特にDECはCOMの提唱者であるとともにCORBAに対応した製品ObjectBrokerの開発元でもあります。

COMとCORBAを接続するためには2つの課題を解決することが必要です。1つはお互いのオブジェクトを見つけだす方式を定めることです。ObjectBrokerではCOMのSCMに対応した機能をAgent Processによって提供しています。したがってCOMのSCMとObjectBrokerのAgentの間でインターフェイスが確立されればこの課題は解決されます。

もう1つはクライアントとオブジェクトの間で直接通信を行うことです。COMとCORBAのインターフェイスは異なっていますので、インターフェイスの違いをどこかで吸収する必要があります。これはCOMライブラリとCORBAライブラリで実現される必要があります。

4 COMによるシステム開発

4.1 分散オブジェクト・モデルとクライアント/サーバ・アプリケーション

COMにおける分散オブジェクト・モデルは本質的にクライアント/サーバ形態のアプリケーションに適合しています。COMにおいて分散オブジェクトを提供する側をサーバ、オブジェクトを利用する側をクライアントと考えればこれはそのまま、クライアント/サーバ・アプリケーションが実現されるわけです。

ここではどんな手順でCOMに準拠したクライアント/サーバ・アプリケーションが開発されるのかを簡単にみていきます。

■

(1)クラス識別子

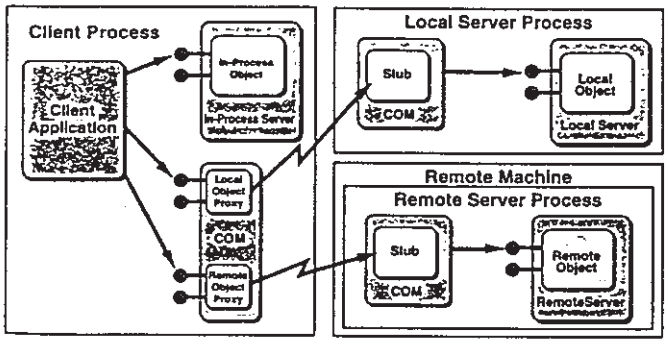
実際の開発過程をみてみる前に1つ注意しておくべきことがあります。

オブジェクトを利用するためにはそのオブジェクトのインターフェイスだけを知っていればいいはずですが、実際には同じインターフェイスをもった様々なオブジェクトが存在することができます。例えばバージョンの違うライブラリであるとか、同じ機能を提供する違うソフト・ベンダーのアプリケーションなど一般的にインターフェイスだけでオブジェクトを識別することはできません。

COMではオブジェクトを識別するためにCLSIDという識別子を利用します。この識別子は128ビットの番号で、絶対に同じ値が割り振られることはありません。従って別々のソフト・ベンダーが独立に開発したオブジェクトであっても必ず異なった番号になりますので識別が可能になるわけです。

COMではそのマシンに存在するすべてのオブジェクトのCLSIDを登録データベースへ登録しておきます。このデータベースを検索することによってCLSIDに対応するオブジェクトがどのように実現されているかがわかるわけです。

あるオブジェクトを利用したいクライアントはこのCLSIDを知っていればSCMを通してどのマシンにどのような方式でオブジェクトが実現されているかを探し出して実際のサービスを提供してもらうことが可能になるのです。



(2)クライアント

それではまずCOMのクライアント側がどのような手順で処理を行うのかみてみましょう。ここでクライアントとはオブジェクトを利用する側のソフトウェアのことを指しています。

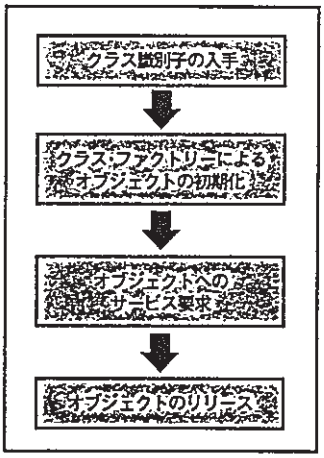
最初に必要なものはCLSIDです。利用するオブジェクトの属するクラス識別子をなんらかの方法で入手する必要があります。CLSIDを入手する標準的な方法はありません。そのオブジェクトにあった入手方式が提供されることになります。

次にCLSIDに対応したクラスのクラス・ファクトリーに要求してオブジェクトの初期化を行います。クラス・ファクトリーとはそのクラスに属するオブジェクトを生成して初期化する一種のオブジェクトです。クライアントはクラス・ファクトリーへのインターフェイスを入手して直接要求することもできますし、CLSIDによってオブジェクトの初期化まで実行するインターフェイスも提供されています。どちらにしてもオブジェクトが初期化されると、そのオブジェクトへのインターフェイスが提供されます。

この後は提供されたインターフェイスを使って、オブジェクトへサービスを要求していきます。

すべての業務が完了した後、オブジェクトをリリースします。これは不要なオブジェクトがシステム資源を占有しないようにオブジェクトの管理を行うためです。

クライアント・プログラム手順



■

(3)サーバ

次にサーバの処理手順をみていきます。サーバとはオブジェクトを実現するソフトウェアのことです。サーバの実現方式には大きく分けて3種類あります。これは先程説明したようにイン・プロセス方式、ローカル・プロセス方式、リモート・プロセス方式です。

イン・プロセス方式ではサーバはDLL(Dynamic Loading Library)で実現され、ローカル・プロセス方式とリモート・プロセス方式では独立したアプリケーションとして実行形式で実現されます。このように実際の構成は実現方式で異なりますが、サーバとして行う必要のある項目をまとめておきます。

■

まず、実現するオブジェクトの属するクラスのクラス識別子を割り当てる必要があります。先程説明したようにオブジェクトの利用にはクラス識別子CLSIDが必要ですので、この割り当てられたCLSIDはサーバの存在するマシンの登録データベースへ登録しなければなりません。

次にオブジェクトを生成し、初期化するクラス・ファクトリーを実現する必要があります。それぞれのオブジェクトに対応したクラス・ファクトリーが必要ですから、クラス・ファクトリーを実現するのもサーバの役割になります。

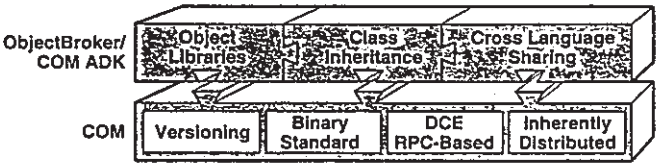
クラス・ファクトリーを実現したならば、それを利用できるようにCOMのシステム内に伝えておくことが必要です。

これらの作業が完了すれば、このサーバはオブジェクトとしてのサービスを開始することが可能になったわけです。

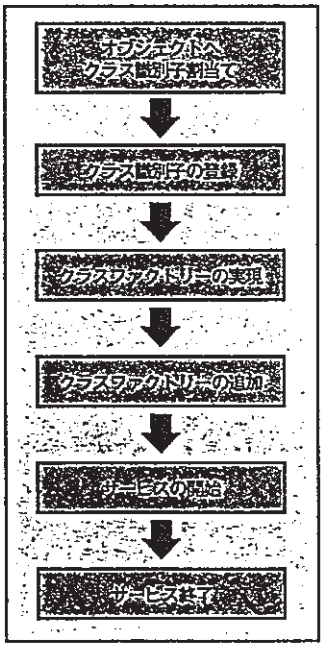
最後にサーバが不要になったときにサーバを停止する機能が要求されます。

■

以上でCOMに準拠したクライアント/サーバ・アプリケーションのおおまかな実現手順がおわかりいただけたと思います。



サーバ・プログラム手順



5 分散オブジェクトの動向とCOMのロードマップ

5.1 分散オブジェクト技術の動向

分散オブジェクト技術は今後のソフトウェア開発において必要不可欠の技術となるでしょう。そのためソフトウェア産業に属するすべての企業が分散オブジェクト技術の動向に注目しています。

現在既に複数の分散オブジェクト技術が提唱されており、またアプリケーションを開発するソフト・ベンダーも分散オブジェクト技術をどのように採用していくか現実の手順を検討しはじめているところです。この流れに取り残されたなら未来のソフトウェア産業では生き残ることが非常に難しくなるでしょう。従ってどの分散オブジェクト技術を選択するかは重要な戦略的決断になるわけです。

そこで簡単に分散オブジェクト技術を比較してみましょう。

分散オブジェクト技術として提唱されているものはたくさんありますが、COM/OLEに対抗できる技術は現在のところSOM/DSOM/OpenDocの技術です。これはIBMとAppleを中心に開発されている技術です。SOM/DSOMはCOMに、OpenDocはOLEにほぼ相当すると考えることができます。

SOM/DSOMをCOMと比較したとき、そのオブジェクト・モデルにおいてCOMはインターフェイスと実現方式の規定であるクラスを区別している点に特徴があります。このためCOMの方が複数言語サポートや分散オブジェクトへの拡張が容易にまたスムースに実現できています。

逆にSOM/DSOMではクラス概念の実現はより徹底して行われており、メタ・クラスやクラス継承などのより本格的な機能が提供されています。もちろんそのためにSOM/DSOMは膨大で非常に重い実現方式になっています。

またOpenDocとOLEの機能はほぼ同じと考えられます。最大の違いはOLEはすでに提供されている技術であり、OLEの機能を利用した多くのアプリケーションを持っているということです。

これらの点から考えればCOM/OLEが最も優れた分散オブジェクト技術であることがおわかりいただけることと思います。

項目	COM/OLE	SOM/DSOM/OpenDoc
実装方式	軽い	重い
クラス継承	オプション	本格的に実現
拡張性	大	小
コンパウンド・ドキュメント	同等	同等
実績	大	小

5.2 DECのCOMロードマップ

最後にDECのCOM技術の開発ロードマップをまとめておきます。

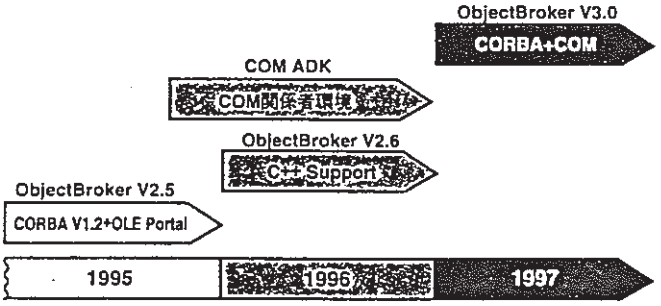
DECとMicrosoft社は1993年にCOMの共同開発を発表して以来、まずCOMの仕様開発を行い、1994年にCOMの仕様を主要なベンダーに公開して、評価を受けました。現在はその評価を元に修正された仕様に基づいてCOMの開発を行っているところです。

DECではCORBA V1.2に準拠したObjectBroker V2.5を1994年9月からリリースしています。このObjectBroker V2.5ではOLEとのインターフェイスを提供しておりOLEのクライアントからCORBAオブジェクトを呼び出すことが可能です。

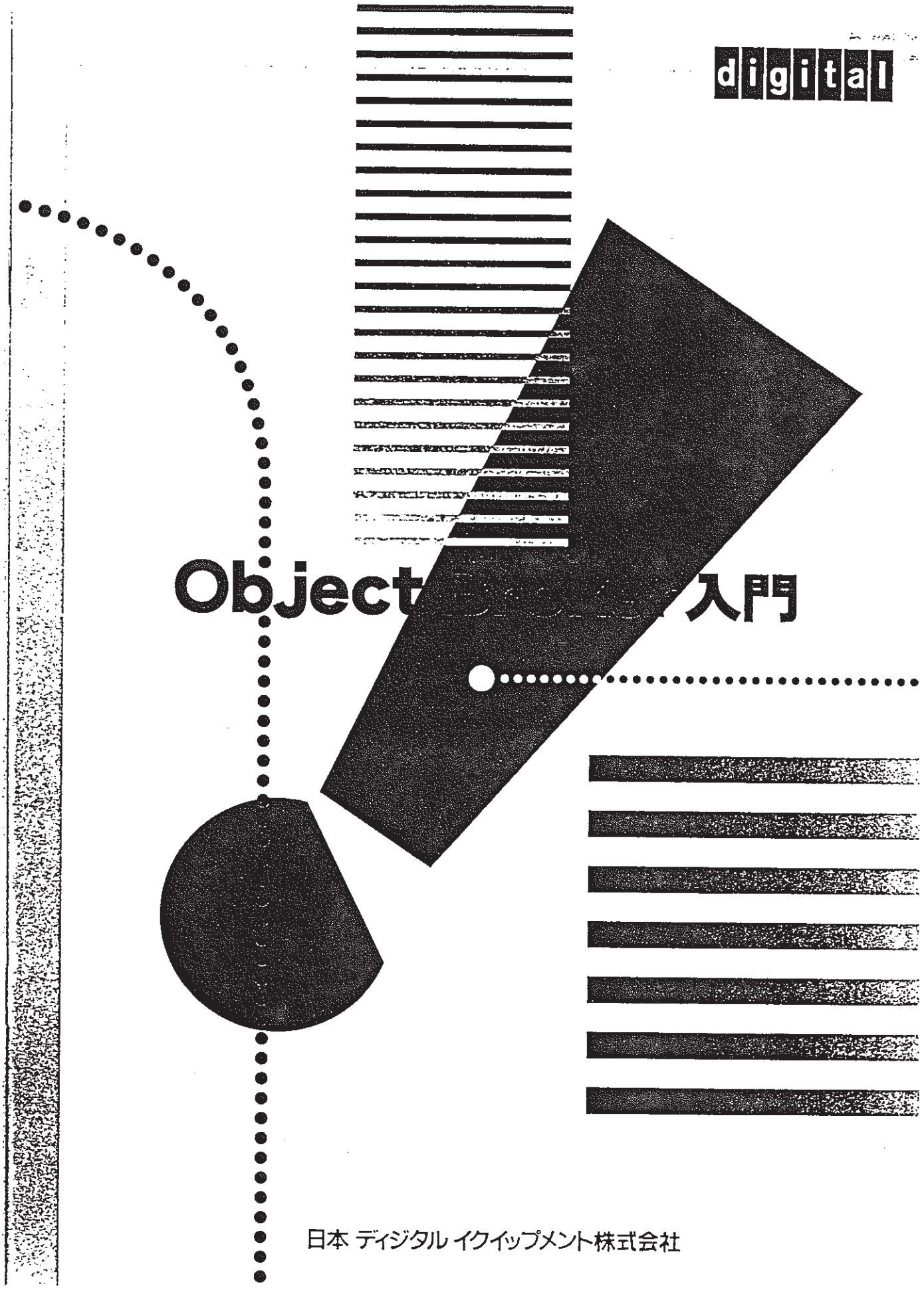
DECでは今後ともCORBAへの対応を継続する予定で、1996年始めにリリース予定のObjectBroker V2.6ではC++インターフェイス標準をサポートすることになっています。

これと平行してCOMによるソフトウェア開発を可能にするCOM ADKを1996年の前半までには提供する予定です。このことによってCOMによるソフトウェア開発に早期に着手する必要のあるソフトウェア・ベンダーをサポートします。

そして1997年上旬までにはCOMのすべての機能を提供するObjectBroker V3.0がリリースされる予定です。ObjectBroker V3.0はCORBAの機能も同時に提供するハイブリッドな分散オブジェクト環境になるわけです。



付録Ⅰ　　ＯｂｊｅｃｔＢｒｏｋｅｒ入門


digital
Object 入門
日本 ディジタル イクイップメント株式会社

1. 分散マルチベンダ環境

【1】2つのキーワード

現代のコンピューティング環境を読み解くためには、2つのキーワードを念頭において考察してみる必要があります。1つのキーワードは「集中から分散へ」であり、もう1つのキーワードは「単一から多様へ」です。これらのキーワードが示しているこれからのコンピューティング環境のあるべき姿は「分散マルチベンダ環境」という言葉で表現することができます。この「分散マルチベンダ環境」は80年代に急速に進められた分散化と多様化という2つの技術革新を取り込んで実現された現代のコンピューティング環境の発展の1つの帰着点ということができます。「分散マルチベンダ環境」の実現によってエンドユーザのみならず、アプリケーション開発者、システム管理者も様々なメリットを享受することができます。しかしながら、「分散マルチベンダ環境」を実現するためには全く新しいソフトウェア技術を必要としています。

【2】「集中から分散へ」

最初のキーワードについてもう少し検討してみましょう。ハードウェアがもっとも高価なコンピュータ資源であった時代には、ハードウェアをいかに効率よく利用できるかが最大の課題であり、そのためにはハードウェアを集中化して利用することが要求されていました。しかしながら、ハードウェアの急速な低価格化やネットワーク技術の発展によって、従来のような集中化されたコンピューティング環境を支えてきた技術的な制約が失われました。それに伴って、最も高価なコンピュータ資源がユーザとしての人間に置き代わり、集中化されたシステムに代わる人間の生産性を最も高めるシステム形態、分散化、が要求されるようになりました。

分散化への経済的、または技術的な前提が提供される一方で、分散化を要求する新しい要因が生じてきました。1つにはシステム化の対象となる業務がコンピュータ利用の拡大やビジネス活動自体の国際化によって、必然的に分散化を要求されてきたことです。更に競争の激化によるビジネス環境の流動化は、組織や業務内容の見直しを必要とし、ひいてはコンピュータ・システムの頻繁な変更を要求します。集中化されたシステムでは、このようなシステム変更に柔軟に対応することができません。システム構成やマシンの増減に容易に対応できる分散環境が要求されます。

このように分散化は現代のコンピューティング環境に要求される様々な課題に対応することを可能にします。しかしながら同時に分散化はコンピューティング環境に新しい課題を持ち込みます。それは分散化に伴う複雑性の増大をいかにコントロールするかという課題です。分散化によってシステムを構成するハードウェアやソフトウェアは地理的に分散します。エンドユーザ自身が自分のマシンを管理し、更には必要な資源を独自に導入することが日常的に起こるようになります。これはエンドユーザの側からは便利なことですが、システムを管理・運用する観点からはどのマシンがどこにあるか、そこにどのような資源が接続されているか、またソフトウェアのバージョンはどうなっているか、などの管理上の複雑性を非常に増大させます。この分散化に伴う複雑性をいかにコントロールするかということが「分散マルチベンダ環境」を実現するために必要な技術的課題の1つなのです。

【3】「単一から多様へ」

続いて第2のキーワードを検討してみましょう。初期のコンピューティング環境では1つのアーキテクチャ、1つのオペレーティング・システムで全体システムが構成されていました。集中化したシステム構成では他に選択の余地がありませんでしたし、最も容易にシステム構築ができる方法でした。一方でシステム規模や用途の違いに応じて様々なアーキテクチャやオペレーティング・システムが開発され提供されてきました。ネットワーク技術の発展により孤立していたシステム同士が接続されるようになると、それらの異なったアーキテクチャやオペレーティング・システムを結びつけた大規模なコンピュータ・システムを実現することが可能になりました。

このような様々なアーキテクチャやオペレーティング・システムの選択を可能にする多様化の進展によって、例えばオフィスや工場、流通、販売店などの多様な環境に対して、それぞれに最適なアーキテクチャとオペレーティング・システムを選び、相互に接続してシステムを構築できるようになったわけです。

しかしながら従来とは異なったアーキテクチャやオペレーティング・システム間で通信が行われるようになると、アーキテクチャやオペレーティング・システム間の互換性が問題になってきました。複数のアーキテクチャやオペレーティング・システムを採用したシステムにおいて、それらの間での通信を実現する環境を整備したり、アプリケーションを開発することは非常に困難です。このような多様化に伴う複雑性のコントロールが「分散マルチベンダ環境」を実現するために必要なもう1つの技術的課題となっているのです。

【4】「分散マルチベンダ環境」

これまで考察された2つのキーワードから指し示される現代のコンピューティング環境は「分散マルチベンダ環境」という名前で表現することができます。この名前が表現しようとする環境は、ネットワークで結合された様々なアーキテクチャやオペレーティング・システムのマシンから構成される非常に大規模なコンピュータ・システムです。分散マルチベンダ環境では、分散化と多様化というキーワードに集約された現代のコンピューティング技術が提供できる様々なメリットを実現しています。
分散化と多様化という2つの発展の方向性の帰着点としての分散マルチベンダ環境が提供するメリットは次のようにまとめることができます。

- 分散している資源に自由に容易にアクセスできること。
- 利用環境に応じた最適なアーキテクチャとオペレーティング・システムを選択して大規模なシステム構築を可能にすること。
- 環境の変化に応じて、柔軟にしかも短期間でシステムの再構築ができること。

このようなメリットを提供する分散マルチベンダ環境は、現代のコンピューティング環境を構築する基盤となります。何よりも重要なことは自由でオープンなコンピュータ・システムを提供することによって、ユーザ層を拡大し、更にエンドユーザがシステムの発展に参加できる環境を提供することです。コンピュータ・システムを開発する側と利用する側の壁を出来るだけ低くすることによって、エンドユーザ・コンピューティングを可能にし、全般的な生産性の向上を実現することができ

ます。分散マルチベンダ環境はエンドユーザ・コンピューティングを可能にするばかりでなく、それと結びついてより高度で自律的なコンピューティング環境を実現する第1歩なのです。

【5】複雑性の問題と新しい技術の必要性

しかしながら分散マルチベンダ環境はそのような要求を実現する前提ではありますが、そのままで要求を実現できるわけではありません。分散化と多様化を推進してきた様々な技術的な成果によって上記のメリットを提供する基盤は与えられましたが、そのメリットを実現しすべてのユーザに享受させるためには、従来の技術の延長線にはない新しい技術を必要としているのです。

分散マルチベンダ環境で直面する新しい問題は複雑性をいかにコントロールするかという問題に帰着します。もちろん複雑性のコントロールという問題は分散マルチベンダ環境以前から存在していましたが、それは基本的に規模の複雑性のコントロールという線型な問題を解決することに帰着します。分散マルチベンダ環境において問題は単に規模の複雑性のみから、分散化による複雑性、多様化による複雑性を含んだ多次元的な問題に拡張され、その解決は飛躍的に困難になりました。従来のソフトウェア技術が対象としてきたのは主に規模の複雑性をコントロールすることでした。今新しい技術に求められるのは、多面的な複雑性を同時にコントロールできる基盤技術なのです。

この新しい技術が解決しなければならない課題は、分散化および多様化にともなって現われてきた問題を解決することと分散マルチベンダ環境におけるメリットをユーザに提供すること、つまり以下の項目にまとめられます。

- 分散環境において様々なコンピュータ資源がどこにどのような状態で接続されているかというような物理的な情報なしでアクセスを行うこと。
- マルチベンダ環境における多様なプラットフォームの上で、容易に移植性の高いアプリケーションを開発できること。
- 様々なサービスを提供するソフトウェアをシステムのビルディング・ブロックとして独立して開発し、共通で利用できる枠組みの提供すること。

2. 分散オブジェクト技術

【1】分散オブジェクト技術の目指すもの

分散マルチベンダ環境を実現するための新しい技術的基盤を提供するものが分散オブジェクト技術です。分散オブジェクト技術の解決しなければならない課題は、まず何よりも分散マルチベンダ環境において増大した複雑性のコントロールです。このために分散オブジェクト技術は80年代に急速に進歩したオブジェクト指向技術を利用し分散環境に拡張することによって、分散マルチベンダ環境によって導入された分散化と多様化による複雑性の問題のみならず、従来から存在する規模の複雑性の問題にも同時に対応することを課題としています。

更に分散オブジェクト技術は現代のコンピューティング環境の基盤として、分散マルチベンダ環境において様々なサービスを体系的に提供する枠組みを提供しなければなりません。この機能は

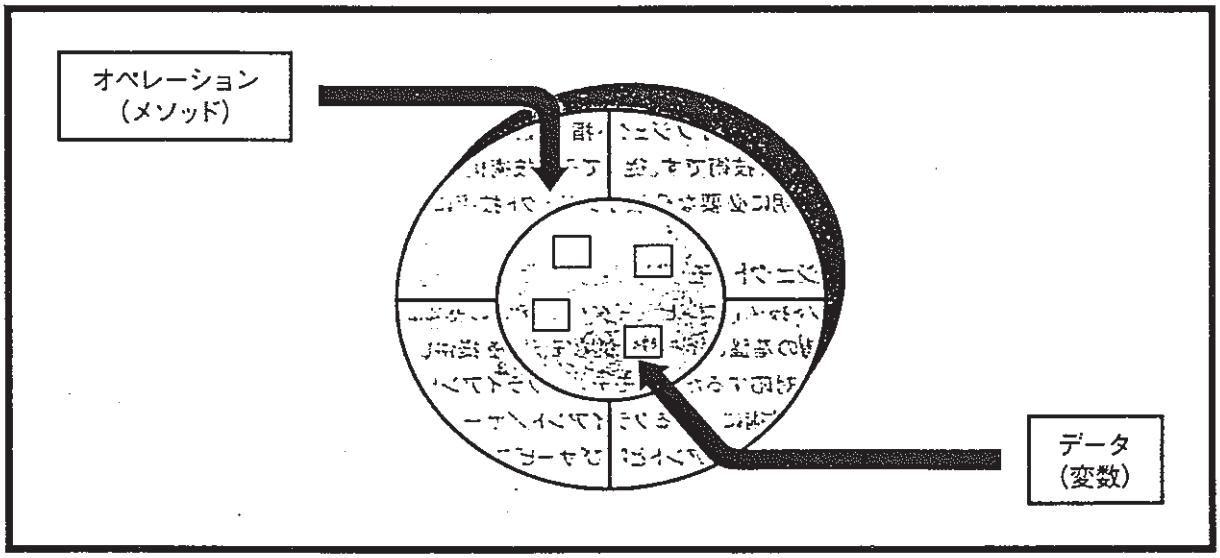
従来はオペレーティング・システムとシステム・ライブラリ群によって提供されていたものですが、分散マルチベンダ環境では分散オブジェクト技術が代わって提供することになります。しかもそのときにエンドユーザやアプリケーション開発者が容易に分散マルチベンダ環境に対応してゆけるような枠組みを提供することが求められます。このことによって分散オブジェクト技術はエンドユーザ・コンピューティングの基盤技術ともなるのです。

【2】オブジェクト指向技術と分散オブジェクト技術

オブジェクト指向技術は80年代に飛躍的な進歩を遂げた比較的新しい技術であり、その目的は当時のソフトウェア工学の最大の課題であった規模の複雑性をいかにコントロールするかという問題を解決することでした。オブジェクト指向技術による解決はソフトウェア開発の基盤をそれまでの処理を単位とした設計から「もの」を単位とした設計へ変換したことです。より変更の起こりやすい処理から、より永続性のある「もの」へ設計の単位を変換したことにより、安定した開発の基盤が提供可能となったのです。また「もの」を単位とした設計は現実の環境を忠実に反映することが可能であり、環境の変化にも容易に対応して設計変更を行うことができます。

オブジェクト指向技術による、処理から「もの」への設計単位の変換は単純なアイディアですが現実に処理を単位として設計されている従来のコンピュータ・システムに適応していくためには様々な技術革新が必要でした。その中で一番重要なものがカプセル化の技術です。「もの」をソフトウェアの設計単位にしたといっても現実に存在する物をそのままソフトウェアの世界に取り込めるわけではありません。「もの」の性質を表現するデータと「もの」への処理を実現するオペレーションによって現実の存在をシミュレーションすることになります。このようにデータとオペレーションによってシミュレーションされたソフトウェア上の「もの」の表現がオブジェクトです。カプセル化はオブジェクトに対する外部からの操作を厳密に定義されたインターフェイスに限定することによって、オブジェクトの内部的な処理を実現するインプリメンテーションに対する誤った、または不正な干渉を防ぎます。インターフェイスとインプリメンテーションの分離を実現するカプセル化によっ

■ オブジェクトの構造



て、システム環境の変化に対応するインプリメンテーションの変更が直接対象となるオブジェクトに局所化され、関連するオブジェクトやアプリケーションには影響をあたえません。このようにカプセル化は規模による複雑性をオブジェクトの内部に局所化することによりコントロール可能にし、オブジェクト相互の関連による大局的なシステム設計を実現しました。

またもう1つ重要なものは、再利用の技術です。カプセル化による複雑性のコントロールもシステム全体がオブジェクトによって構築されなければ大きな効果をあげることは困難です。オブジェクトを基盤としたシステム環境の全般的な再構築に大きな威力を発揮するのが、一度開発したオブジェクトをできるかぎり有効に再活用していく再利用技術です。オブジェクトはインターフェイスとインプリメンテーションから構成されますので、それぞれの再利用技術が必要です。オブジェクト指向技術ではこれは継承機能によって提供されてきました。継承機能ではベースとなるオブジェクトのインターフェイスやインプリメンテーションを下位のオブジェクトが受け継いでいくことによって再利用を実現します。更にインターフェイスとインプリメンテーションの関係を自由に変更していける機能も、特にインターフェイスの再利用のために必要です。これはポリモーフィズム機能によって実現されます。ポリモーフィズム機能は1つのインターフェイスに対して複数のインプリメンテーションを対応づけられる能力であり、特に対応づけを実行時まで遅延させられる機能によって、動的なシステム変更を実現できます。

分散オブジェクト技術は以上に述べてきたカプセル化や再利用技術といったオブジェクト指向技術の成果を基盤としています。しかし従来のオブジェクト指向技術は、分散化や多様化以前のコンピューティング環境を前提に開発されてきました。そのままでは分散マルチベンダ環境に対応することはできません。そのために分散オブジェクト技術は必要な拡張をオブジェクト指向技術に対して行っています。特に重要なことはカプセル化によるインターフェイスとインプリメンテーションの分離をより厳密にしたことで、これによって分散化や多様化による複雑性をオブジェクトの内部に局所化することが可能になりました。また再利用技術においても分散化と多様化に対応するためには既存の継承機能やポリモーフィズム機能そのまま利用することはできません。逆に分散化や多様化に対応するために従来のオブジェクト指向技術の見直しが行われ、より厳密で有効な技術の再定義が行われてきました。その結果として分散オブジェクト技術が誕生したのです。

【3】分散オブジェクト技術の構成要素

分散オブジェクト技術はオブジェクト指向技術を中心にして、分散化と多様化に対応する技術成果を取り入れた複合技術です。従ってその技術構造も複数の構成要素から成り立っています。ここでは以降の説明に必要な分散オブジェクト技術に共通する構成要素を紹介します。

●分散オブジェクト・モデル

分散オブジェクト技術はコンピュータ・システムの基本設計にかかわる技術であり、そのためにまずシステム構造の基盤となる概念的なモデルを提供します。分散オブジェクト技術では分散マルチベンダ環境に対応するためのモデルをクライアント／サーバ型のシステム形態に求めました。分散オブジェクト技術におけるクライアント／サーバ・モデルでは、サービスを要求するユーザやアプリケーションをクライアントと呼びサービスを提供するオブジェクトをサーバと呼びます。サーバとなるオブジェクトはインターフェイスとインプリメンテーションから構成されます。クライアントからオブジェクトへの要求はすべてインターフェイスに基づいたリクエスト転送によって行われます。

●オブジェクト・インターフェイス

分散オブジェクト管理においてはクライアントがオブジェクトにアクセスできる方法はオブジェクト・インターフェイスとして定義されたオペレーションに対するリクエスト転送しかありません。逆に言えばインターフェイス定義だけで外部から見えるオブジェクトのすべてを表現することができます。このように分散オブジェクト技術にとってオブジェクト・インターフェイスは非常に重要なものですので、専用のインターフェイス記述言語(IDL)が提供されています。IDLで記述されるオブジェクト・インターフェイスは一連の関連するオペレーション群で構成されます。1つのインターフェイスをクラスと呼び、同じインターフェイスを提供するオブジェクトは同じクラスに属することになります。

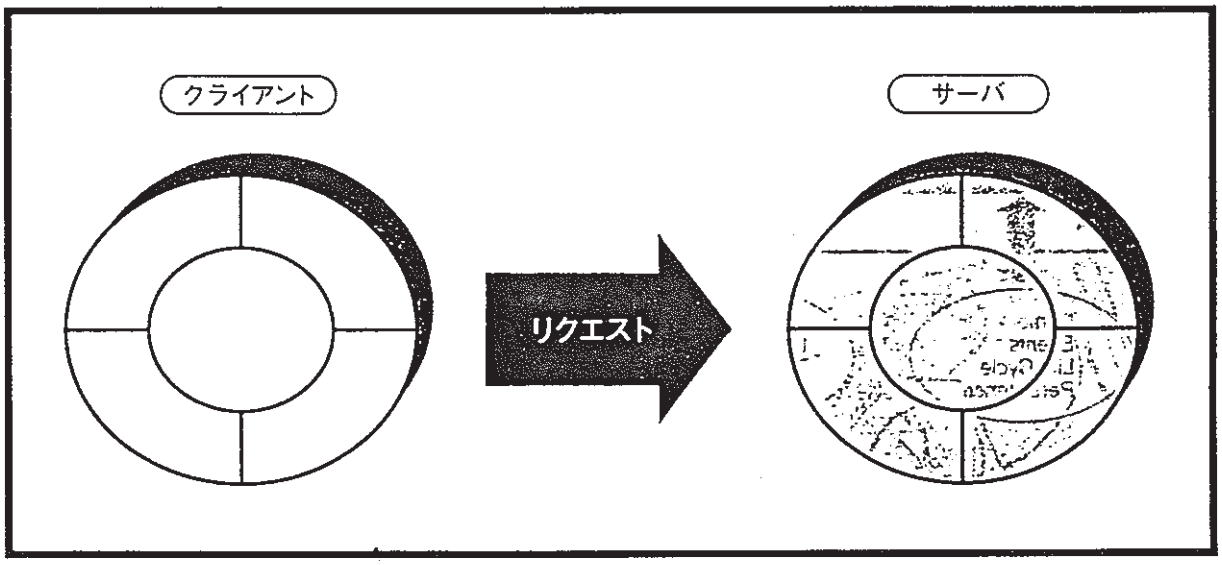
●オブジェクト・インプリメンテーション

オブジェクトがサービスを提供するための内部的な処理の実現はオブジェクト・インプリメンテーションによって行われます。オブジェクト・インターフェイスの個々のオペレーションに対応するインプリメンテーションをメソッドと呼びます。分散オブジェクト管理におけるオブジェクト・インプリメンテーションではメソッドの他に、オブジェクトの起動やメソッドの選択を行うためのサーバ機能を提供する必要もあります。

●リクエスト転送

クライアントからオブジェクトへのサービスの要求はリクエスト転送によって行われます。分散マルチベンダ環境におけるリクエスト転送は、ネットワークやオペレーティング・システムの違いを吸収する必要があります。特にプラットフォームの違いによるリクエスト・データの非互換性を吸収する機能をマーシャリングと呼びます。またクライアントとオブジェクトの関係を動的に切り替えてゆく機能もリクエスト転送によって提供されます。

■ 分散オブジェクト・モデル



3. OMGとCORBA

【1】OMGの活動

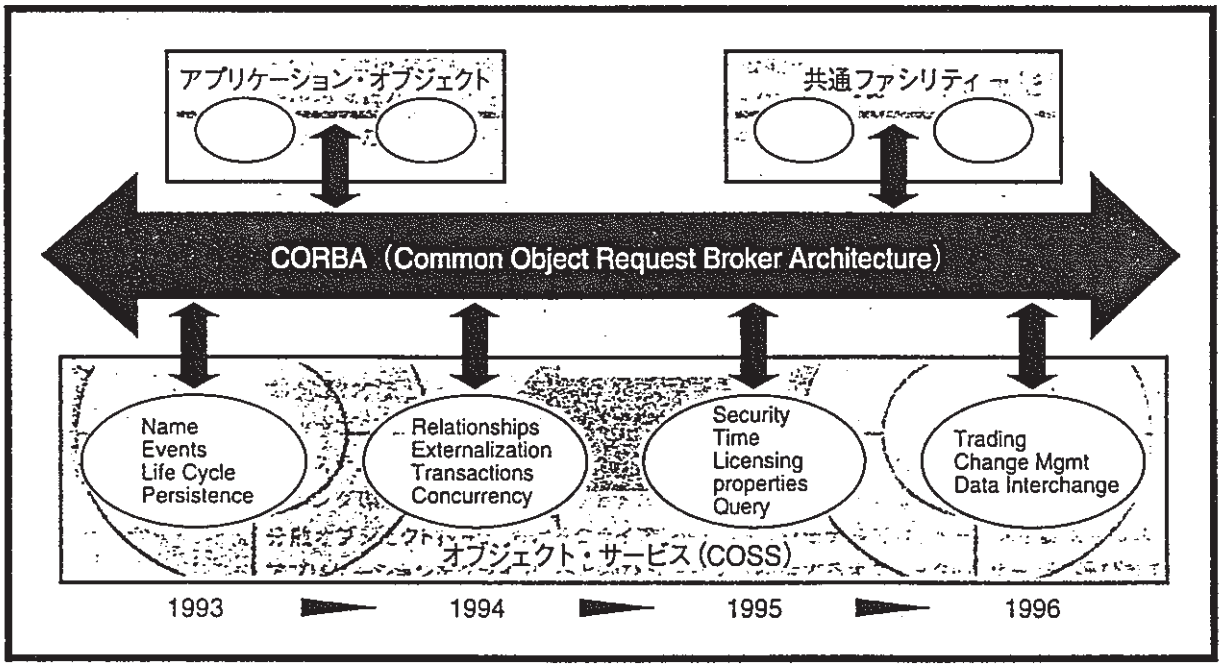
OMG(Object Management Group)は分散オブジェクト技術の標準化と普及のために1989年に設立された非営利団体です。OMGはDECをはじめ、HP、IBM、SunSoftなどのベンダを中心に現在では475社が参加し、分散オブジェクト技術に関する国際的な標準化をX／Openなどの他の標準化団体と連携して行っています。

OMGにおける標準化はRFPという標準技術の要求仕様を公開して参加企業に標準技術案の提示を求め、応募された標準案を技術委員会で検討した後に、全会員による投票で標準化を決定するというオープンな標準化手順を採っています。また採用された標準化技術をドキュメント化し出版するというような普及活動も併せて行っています。

【2】OMGによる分散オブジェクト技術

OMGによる分散オブジェクト技術はまずOMA(Object ManagementArchitecture)という全体的なモデルの規定からはじめられました。OMAは1990年に標準化され、OMAガイドとして出版されました。このOMAで提示した分散オブジェクト技術の全体像は下図に示すように4つの部分から構成されています。

■ OMGによる分散オブジェクト体系



その中でも中心に位置するCORBA(Common Object RequestBroker Architecture)は分散オブジェクト・モデルを規定し、分散オブジェクトの実現と管理を行うという最も重要な役割を果たしています。このために、標準化作業が最初に着手されました。

オブジェクト・サービスは分散オブジェクト管理上の基本的な機能、オブジェクト・ライフサイクルやセキュリティなどを提供するための技術を標準化するものであり、現在CORBAに続いて標準化作業が行われています。既にその1部がCOSS1として出版されています。

共通ファシリティは、様々なアプリケーションで共通に必要な機能を標準化して提供するものです。この分野については現在作業中で公表されたものはありません。

アプリケーション・オブジェクトは分散オブジェクト技術を利用したアプリケーションです。これはソフトウェア・ベンダやユーザによって開発されるソフトウェアでOMGからは提供されません。

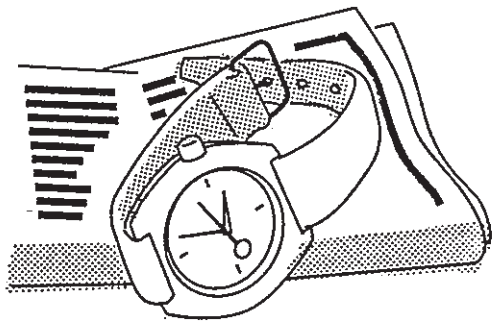
【3】CORBAの開発経過と将来

CORBAはOMGの分散オブジェクト技術の中で中心的な役割を演じる技術です。CORBAの具体的な機能と構造についてはObjectBrokerの解説とともに行うこととして、ここではCORBAの標準化動向だけを説明します。

CORBAの最初の標準化は1991年に行われ、V1.1として出版されました。続いて1993年にV1.1の修正版としてV1.2が標準化されました。これはAPIのネーミングを変更するなどの記述面の変更が行われた程度で本質的な変更は行われていません。従って現在のCORBAの標準はV1.2ということになります。

現在はCORBA V2.0の標準化が進行中です。この標準化では異なったベンダのORB製品同士のインターオペラビリティを提供することとオブジェクトの初期化の手順を規格化することを目標としており、既に各社から提案がされている状況です。(1994年12月に決定されました。)それと同時にC++言語の実装方式の標準化も進行しています。

今後の標準化の対象としては、サポートする言語を拡張していくこと(Lisp、COBOLなど)と異なった分散オブジェクト技術とのゲートウェイ機能の提供が予想されます。またオブジェクト・サービスの強化もCORBAをサポートするために必須となる点です。



4. ObjectBroker

【1】ObjectBroker概要

ObjectBrokerはDECが開発したCORBA準拠のミドルウェア製品です。現在販売中のV2.5はC-ORBA V1.2に100%準拠しています。ここではObjectBroker V2.5を前提にCORBAが提供する分散オブジェクト技術の機能と構造を説明してゆくことにしますが、その前に少しだけ製品としてのObjectBrokerの紹介をさせていただきます。

●CORBA V1.2に100%準拠した製品です。

ObjectBrokerは既に3年以上の実績をもった製品であり、多くのシステムで稼動中です。昨年公開されたばかりのCORBA V1.2を最初にサポートした製品の1つでもあります。

●最も広いプラットフォームをサポートします。

現在のサポート範囲は下表に示す12プラットフォームです。これはCORBA準拠製品中のみならず、ミドルウェア製品としても最も広いプラットフォームをサポートしていることになります。今後もPC98など新しいプラットフォームにサポートを拡大するため開発を続けています。

●アプリケーションを開発・運用していくために必要なツールを提供し、完結した分散オブジェクト環境を提供します。

リポジトリのブラウザーやコード・ジェネレータ、環境のセットアップ・ツールなどアプリケーション

■ ObjectBroker のサポート・プラットフォーム

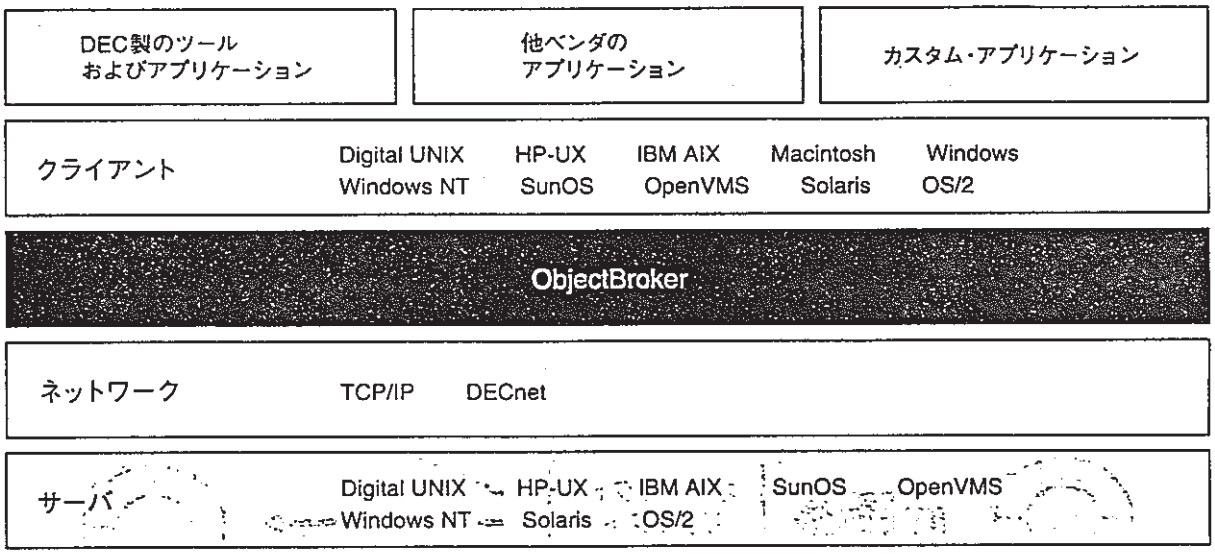
システム	クライアント	サーバ
Apple Macintosh System 7	Yes	No
Digital OpenVMS Alpha	Yes	Yes
Digital OpenVMS VAX	Yes	Yes
Digital UNIX	Yes	Yes
Hewlett-Packard HP-UX	Yes	Yes
IBM AIX	Yes	Yes
IBM OS/2	Yes	Yes
Microsoft Windows	Yes	No
Microsoft Windows NT AXP	Yes	Yes
Microsoft Windows NT Intel	Yes	Yes
Sun Microsystems SunOS	Yes	Yes
Sun Solaris	Yes	Yes

の開発・運用に必要なユーティリティ類を提供しており、分散オブジェクト技術を利用するための完結した環境が提供されています。

●Windows環境をサポートするための機能が提供されています。

OLE／DDEとのインターフェイスやVisual Basic用のAPIを提供してWindows上のアプリケーション開発が容易になるような機能強化を行っています。

■ ObjectBrokerの実行環境



【2】分散オブジェクト・モデル

分散オブジェクト・モデルという概念的なレベルでは現在実用化されている分散オブジェクト技術の間で際立った違いはないと思われます。いずれの分散オブジェクト・モデルでもクライアント／サーバ型の分散モデルを前提のシステム形態にして、サーバとしてのオブジェクトがインターフェイスとインプリメンテーションから構成されるという基本概念になっており、ObjectBrokerもこの点まではまったく同じです。

基本レベルで違いが出てくるのは、クライアント・アプリケーションからの言語レベルのインターフェイスをどのように提供するかという問題と、オブジェクトへのアクセスをどのように開始するかという問題からです。以下でこれらの点について説明します。

●言語バインディング

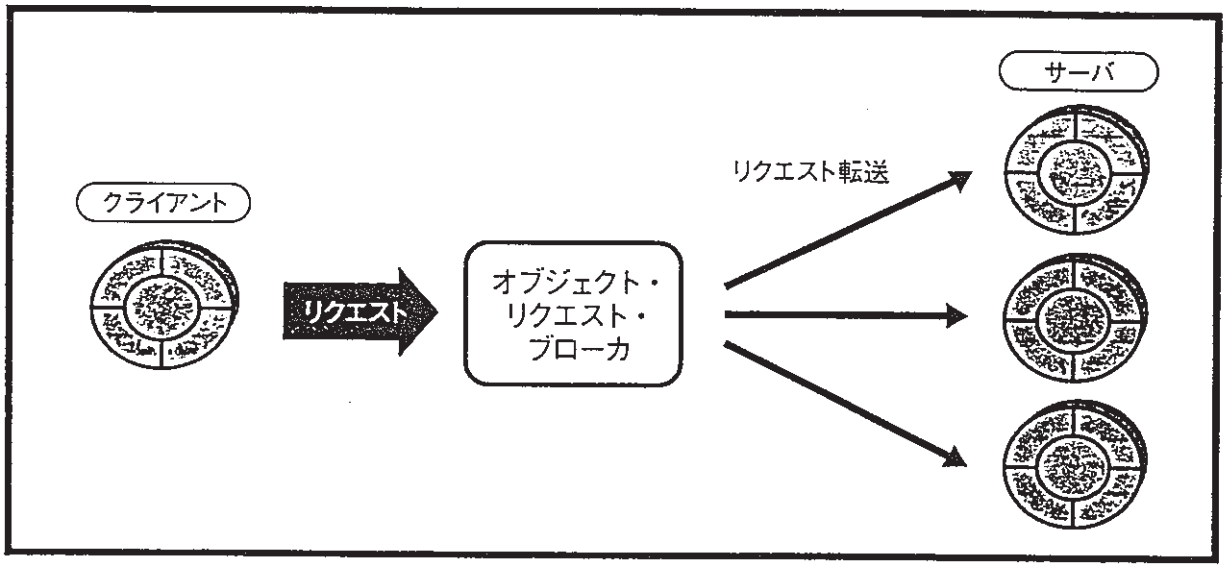
CORBAでサポートすることが求められる環境上では様々なプログラミング言語が使用されています。従ってCORBAの環境を利用して分散オブジェクトへリクエストを送るアプリケーションも様々な言語で記述できる必要があります。しかしながら現実に存在するプログラミング言語では例えばC++、Lisp、Smalltalkなどを比較すれば分かるとおり、それぞれ非常に異なった言語仕様に基づいており、同じ機能を提供するためにも異なった実装が要求されます。

CORBAでは言語ごとにそれぞれのAPIを標準化することでこの問題に対応しています。現在はC言語に対する標準化だけが完了しており、引き続きC++言語の標準化作業が行われています。ObjectBrokerではすべてのプラットフォームにC言語のインターフェイスを提供しており、Windows環境については独自の機能強化としてVisual Basicのインターフェイスを提供しています。

●オブジェクト・リファレンス

CORBAの規格ではクライアントがオブジェクトにリクエスト転送を行うときに相手のオブジェクトを特定する必要があります。このためにCORBAではオブジェクトを識別するための情報としてオブジェクト・リファレンスを提供することが求められています。ObjectBrokerではオブジェクトを一意に識別できるデータを利用して、オブジェクト・リファレンスを作成するルーチンを提供しています。

■ ObjectBrokerの分散オブジェクト・モデル



クライアント・アプリケーションで最も重要な問題はこのオブジェクト・リファレンスをリクエスト転送の前にいかにして入手するかということです。CORBAではこの点に関して明確に規定されていません。ObjectBrokerはこの問題に対して3つの方法を提供しています。1つはクライアント・アプリケーションに埋めこんでしまう方法であり、もう1つは外部のデータベースに格納する方法ですが、最も標準的な方法はObjectBrokerが提供するレジストリに登録して、ネーミング・サービスを利用して取り出す方法です。

【3】オブジェクト・インターフェイス

CORBA V1.2までの標準化における中心の課題はクライアントから見たオブジェクトのインターフェイスを厳密に規定することでした。このためにCORBA IDLというインターフェイス言語を提供しています。CORBA IDLにおいてインターフェイスの再利用は継承機能によって実現されています。継承機能の特長について少し詳しく説明します。またCORBAではクライアント・アプリケー

ションが実際にオブジェクトへリクエスト転送を行うインターフェイスに静的なタイプと動的なタイプを提供しています。2つのタイプのインターフェイスの違いについても紹介します。

●CORBA IDL

CORBAで規定するインターフェイス定義言語CORBA IDLは、C++言語のシンタックスに準拠したオブジェクト・インターフェイスを記述的に定義する言語です。記述的というのはインターフェイスの定義を行うだけで、インプリメンテーションからは完全に分離された言語であることを示しています。

IDLの実際の内容についてはのちほど紹介するプログラム例を参照していただくこととして、簡単にその構造を説明すると、1つのIDL仕様は複数のインターフェイスから構成されます。この個々のインターフェイスは1つのオブジェクトのクラスに相当し、そのクラスのオブジェクトがサポートしなければならないオペレーションを定義しています。1つのインターフェイスに対応するオブジェクトは定義されたすべてのオペレーションのインプリメンテーションを提供する必要があります。オペレーション定義はオペレーションの名称とパラメータおよびリターン値のデータ型の定義で構成されます。IDLではこのほかに属性(Attribute)、コンテクスト、例外を定義することができます。

IDLはあくまで記述的な言語であるので、これを実際のプログラミング言語にマッピングしていく必要があります。CORBAではリクエスト転送方式の違いに対応して2種類のマッピング方式を提供しています。その詳細はのちほど説明しますが、いずれにしてもIDLによるインターフェイス定義によってマッピングに必要なすべての情報が提供されます。

●多重継承

CORBAにおけるインターフェイス定義の再利用は継承機能として提供されます。CORBAの提供する継承機能はオブジェクト指向技術における継承機能と同様にベースとなるインターフェイス定義を別のインターフェイス定義で受け継いで再利用します。具体的にはベース・インターフェイスで定義されたすべてのオペレーション、属性、コンテクスト、例外を引き継ぐことになります。

IDL上で継承機能を表現するシンタックスはC++言語のシンタックスに類似しています。更にCORBAの継承機能はC++言語と同じく多重継承機能を提供しています。この多重継承機能は複数のベース・インターフェイスからの継承を許す機能で複数のインターフェイスで定義されたものをすべて引き継ぐことになります。多重継承は非常に有効な場合があるかわりに、問題を引き起こすこともあります。例えば、複数のベース・インターフェイスで同じ名前のオペレーションが定義されている場合どのように継承すべきかという問題が考えられます。いずれにしてもあまり複雑な多重継承はシステムの見通しを悪くし、様々な障害の原因となるのでさけるべきです。

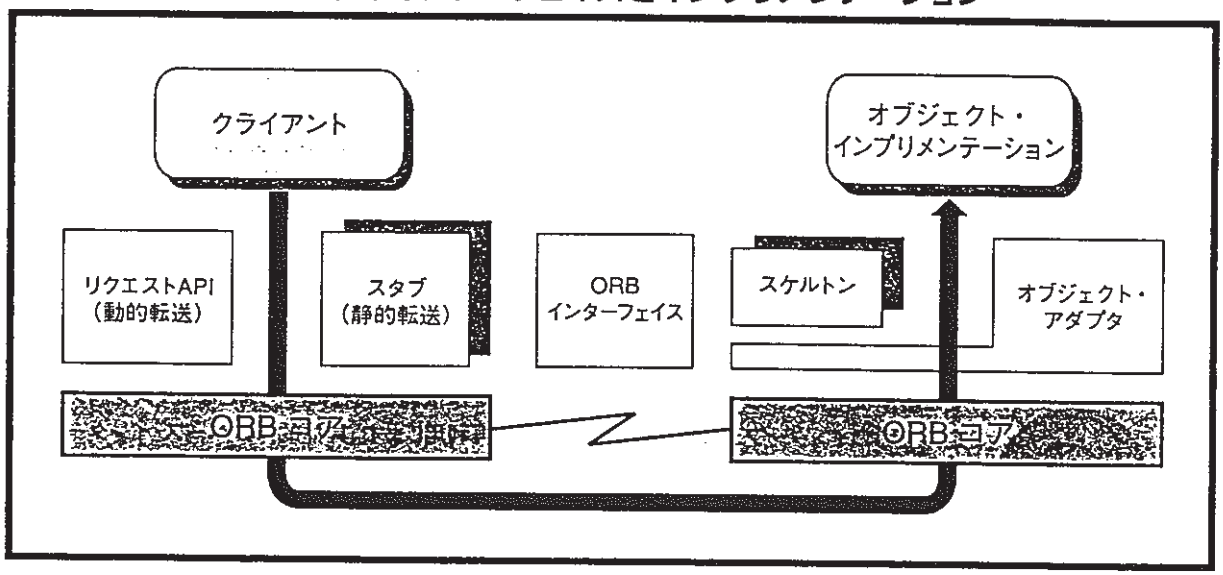
●リクエスト転送インターフェイス

前に説明したように、CORBAではクライアントからリクエストを転送する方式に対応して2種類のプログラム・マッピング方式が提供されます。1つは静的な転送方式に対応したスタブによるマッピングであり、もう1つは動的な転送方式に対応した専用のAPIによるマッピングです。
静的な転送方式とはインターフェイス定義を動的に変更する必要性のない場合に利用される転送方式で、RPC(Remote Procedure Call)に類似したインターフェイスを提供します。静的な転

送方式に対応するプログラム・マッピングはIDLから生成されるスタブを利用して行われます。スタブを利用するクライアント・アプリケーションは通常のプロシジャ・コールと同じ様にオペレーション名に対応したプロシジャを呼びます。実際のリクエスト転送処理はリンクされたスタブの内部で行われるため、クライアントからは見えない形になります。

一方、動的な転送方式ではインターフェイス定義の動的な変更に対応することが可能です。この方式ではクライアント・アプリケーションが転送されるリクエスト・データを作成し、専用のAPIで転送処理を行います。従ってインターフェイス定義を参照してリクエスト・データを作成するアルゴリズムを作りこめば、インターフェイス定義の動的な変更に対応することができますが、クライアントからリクエスト転送の処理が見える形になります。また転送するリクエスト・データのチェックやマーシャリングに必要な情報を動的に取得するために、IDLからインターフェイス定義を格納するインターフェイス・リポジトリを生成して利用します。

■ CORBAオブジェクトのインターフェイスとインプリメンテーション



【4】オブジェクト・インプリメンテーション

CORBAではインターフェイス定義中のそれぞれのオペレーションにそれが提供するサービスを実現するためのソフトウェアをメソッドと呼びます。オブジェクトのインプリメンテーションはこのメソッドを提供することに尽きるのですが、分散オブジェクト環境では、メソッドを実行可能な状態にし、個々のリクエストを対応するメソッドに割り当てるなどのサーバ機能が必要になります。このメソッドを束ねるサーバ・ソフトウェアをどのように構成し、どのように配置するかということが問題になります。ObjectBrokerではサーバの構成を記述するためにIMLを、サーバの配置を決定するためにMMLという記述言語を提供しています。またオブジェクト・インプリメンテーションを実現するために共通に必要な機能を基本オブジェクト・アダプタとして提供しています。なお、CORBAではインプリメンテーションの実現の詳細は規定していません。従ってIML、MMLなどはObject-Broker独自の機能です。

●IMLとサーバ・インプリメンテーション

IML(Implementation Mapping Language)では1つのサーバ・ソフトウェア中でどのメソッドが提供され、それぞれのメソッドがどのインターフェイスのどのオペレーションに対応するかを記述します。最も簡単でおそらく最も一般的なのは、1つのインターフェイス定義中のすべてのオペレーションを1つのサーバ・ソフトウェアで提供する場合で、このときはインターフェイス定義を記述したIDLからIMLを自動的に生成できます。しかしながらCORBAの規定では1つのインターフェイスを複数のインプリメンテーションで実現することが可能ですし、1つのインプリメンテーションが複数のインターフェイスをサポートすることも可能です。このようにCORBAではインターフェイスとインプリメンテーションの多対多の関係づけが可能ですが、IMLではこの内あくまでインプリメンテーション側から見た関係づけだけを記述して、全体の関係づけを表現するためには別の記述言語MMLを必要とします。これはIMLではサーバ・ソフトウェアを実現するのに必要な情報だけを提供し、実行時に必要なマッピング情報は別途定義することによってマッピングの動的な性質を持たせることを意図しています。

IMLではこのほかにインプリメンテーションを開発するための様々な情報を記述します。1つにはメソッドの実現方式があり、これはメソッドをサーバ・ソフトウェアへの組み込みルーチンで実現するか、ダイナミック・ライブラリで実現するかまたはスクリプトで実現するかという選択ができます。またサーバ・ソフトウェアの登録や起動方式などが記述されます。

サーバ・ソフトウェアを実現するためにはメソッドの開発以外に、サーバの起動や登録、リクエストの振り分けなどの機能を提供する必要があります。これらの機能を提供するコードの大半はIDLとIMLからObjectBrokerのユーティリティによって自動的に生成できます。またメソッド・ルーチンのテンプレートになるコードも同時に生成されます。

●MMLとメソッド・マッピング

オブジェクトのそれぞれのオペレーションに対応するメソッドを動的に決定していく機能をメソッド・マッピングと言います。例えばインターフェイス定義の継承機能に対応して継承されたオペレーションに対応するメソッドにベース・インターフェイスに対応するメソッドを再利用することが可能です。この場合は1つのメソッドが複数のオペレーションに対応します。逆に1つのオペレーションに対してバージョンが異なったり、ロケーションが異なったりするメソッドを振り分けて利用することも可能です。この場合は1つのオペレーションに複数のメソッドが対応します。

分散オブジェクト環境で実行時にメソッド・マッピングを行うためにはIDLとIMLの情報だけでは不足です。メソッド・マッピングを実現するために必要な情報を記述するためにObjectBrokerはMML(Method Mapping Language)を提供しています。MMLではそれぞれのインターフェイス定義のオペレーションに対して対応するメソッドを指定していきます。もちろん特定のサーバの特定のメソッドを直接指示することも可能ですが、一般的にはメソッドを選択する条件を指定して動的に条件判断しながらメソッドを決定することになります。条件としてはどのロケーションにあるサーバを選択するか、どのインプリメンテーション定義に対応するサーバを選択するかという判断が可能です。

●基本オブジェクト・アダプタ(BOA)

IMLでも定義できるようにCORBAでは様々なメソッドの実現方式が提供できます。例えばCORBAで採り上げられている例では別プロセスのサーバ・アプリケーションとして構成する例、同じプロセス内に動的にローディングするライブラリとして構成する例、およびオブジェクト指向データベースに格納したオブジェクトとして構成する例が採り上げられています。この他にObjectBrokerではスクリプト言語で構成することが可能なように様々な実現方式が提供される可能性があります。

このような様々な実現形態のオブジェクトをすべてサポートするサービス機能を規定することは困難です。CORBAではそれぞれのオブジェクトの実現形態に適合したサービス機能を提供する一連のライブラリ・ルーチンをまとめてオブジェクト・アダプタと呼んでいます。従って先ほどのオブジェクト実現方式のそれぞれの事例に対応して、サーバ・アプリケーション向けのオブジェクト・アダプタ、動的ライブラリ向けのオブジェクト・アダプタなどが提供される必要があります。しかしながら、CORBAそのものでは個別のオブジェクト・アダプタの規定はありません。

その代わりにCORBAでは基本オブジェクト・アダプタとしてすべてのオブジェクト・アダプタで共通して必要になる機能に関するAPIを規定しています。このAPIは擬似的なオブジェクトであるBOAのオペレーションとして定義されます。BOAで提供される機能は、オブジェクト・リファレンスの生成、インプリメンテーションの活性化と非活性化、インプリメンテーションの登録、オブジェクトへのアクセス権の管理などです。

【5】リクエスト転送

クライアントがオブジェクトへリクエストを転送する仕組みは直接アプリケーションやユーザには見えない部分ですが、分散オブジェクト・システムの開発者や管理者とっては十分理解しておく必要があります。CORBAではリクエスト転送の機能はORBコアによって提供されます。CORBAによって規定されるメソッド・マッピングの動的な性格から、それをサポートするリクエスト転送機能はブローカリングと呼ばれています。ORBコアがブローカリングを実行するときに必要な情報は様々な記述言語によって提供されますが、それらは実行時に参照しやすいようにリポジトリに格納されます。またそれ以外の有効な情報をレジストリと呼ばれるファイルに格納します。またリクエスト転送において必要な機能としてプラットフォームごとのデータ形式の違いを意識せずにリクエスト・データを操作できるようにするマーシャリング機能が提供されます。

●ORBコアの機能とブローカリング

CORBAの枠組みの中で規定されているリクエスト転送の機能には、クライアントやサーバからの要求をネットワークを通じて相手側へ転送するという受動的な部分とリクエスト中で指定されたオペレーションに対応したメソッドを検索し必要に応じて起動処理まで行う能動的な部分があります。

受動的な役割を果たす部分をObjectBrokerではORBコアと呼びます。これはOS上の一種のレイヤとみなすことができます。ORBコアを利用するインターフェイスはこれまで述べてきたようにクライアントでは2種類のリクエスト転送インターフェイスであり、オブジェクト側ではオブジェクト・アダプタです。この他に両者で共通に利用される機能についてはORBインターフェイスが提供

されます。ORBコアという1つのレイヤによって転送機能を提供することによって、クライアントもオブジェクトも相手先のプラットフォームやその間のネットワークなどの物理的な環境を意識する必要がありません。

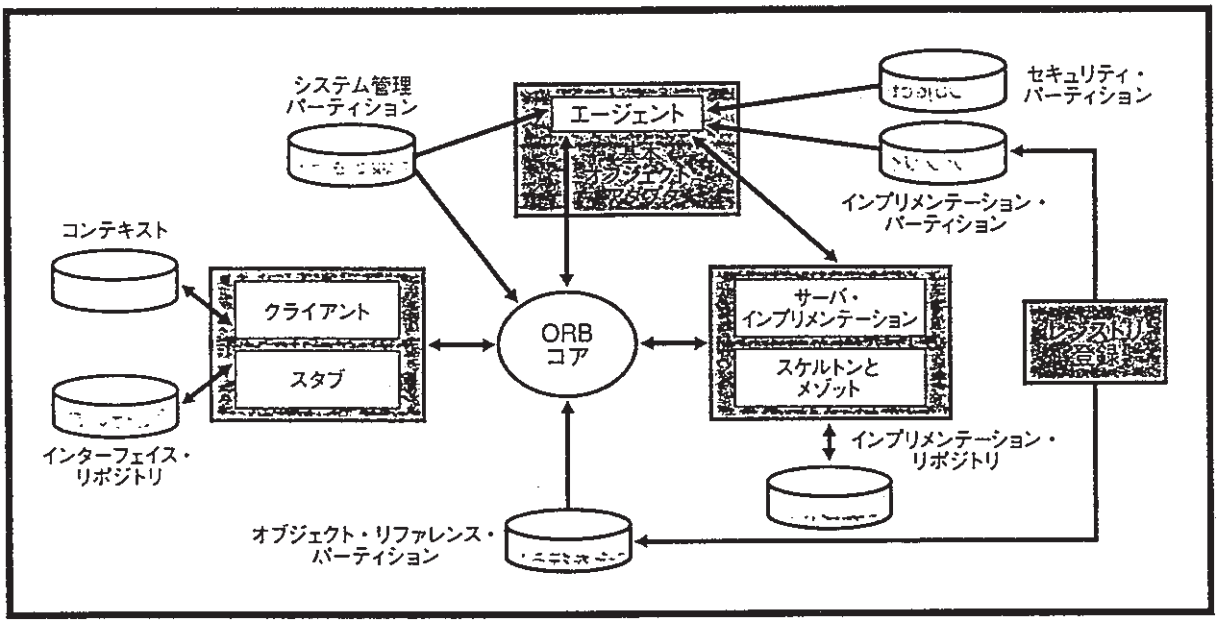
能動的な役割を果たす部分はObjectBrokerでエージェントと呼びます。これはそれぞれのクライアント・マシン上に存在するプロセスとして実現されます。クライアントがリクエスト転送を行うとエージェントはオブジェクト・リファレンスやオペレーション名、クライアントの環境上のコンテクストなどを基にリポジトリとレジストリの情報から対応するオブジェクトのインプリメンテーションとそのロケーションを決定します。

●リポジトリとレジストリ

CORBAではブローカリングを実行するときに必要な情報をリポジトリとレジストリというファイルに格納して提供しています。この内リポジトリはCORBAで規定しているIDL、IML、MMLという3種類の記述言語で定義された情報を格納しています。レジストリはそれ以外の管理情報を専用のユーティリティを利用して設定します。

CORBAでは2種類のリポジトリを規定しています。IDLで記述されたインターフェイス定義を格納するのがインターフェイス・リポジトリで、これはCORBAによって標準仕様が規定されています。一方、IML、MMLで記述されるインプリメンテーションとメソッド・マッピングの定義を格納するのはインプリメンテーション・リポジトリです。これはCORBAで十分に規定されていないためObject-Brokerは独自の仕様を規定しています。
レジストリは大きく4つのパーティションで構成されています。それぞれが提供する情報はインプリメンテーションの配置、初期オブジェクト・リファレンスの提供、分散オブジェクト・システムの管理

■ ObjectBroker のシステム構造



情報、セキュリティ情報です。

●マーシャリング

分散オブジェクト・システムは様々なプラットフォームから構成されることが一般的であり、そのため転送されるリクエスト・データのデータ形式がプラットフォームごとに異なった場合、問題が発生します。データ形式の違いをアプリケーションに意識させないために、CORBAではORBコアの機能として自動的なデータ形式の変換を行っており、これをマーシャリングと呼びます。

マーシャリングの対象となるのはオペレーション中のパラメータとリターン値です。このためにIDLのオペレーションの定義部分でパラメータとリターン値のデータ・タイプを抽象的なデータ型を指定し、マッピング時にクライアントとサーバそれぞれのプラットフォームに適合したデータ・タイプを決定して、データ形式の変換を行うことになります。

5. Object Brokerによるアプリケーション開発

【1】ObjectBrokerによる開発手順

ここまでの説明でObjectBrokerの機能と構造についてはご理解いただけたかと思います。しかしながらソフトウェアの機能を理解することとそれを利用してアプリケーション開発を行うことの間には大きな隔たりがあります。ここではその隔たりの一部分でも埋めるために簡単な例題を使ってObjectBrokerによるアプリケーション開発の実際を紹介したいと思います。もちろん非常に簡単な事例ですので詳細までは説明できませんが、その雰囲気の一端なりとも感じていただけたらと思います。なお、ここで紹介する事例の完全なソースコードは製品に含まれています。またそこにはもっと高度な機能を使った事例も含まれています。

ObjectBrokerによる開発が従来の分散アプリケーションと最も異なっている点は、クライアントが利用する様々なサービスを分散オブジェクトとしてあらかじめモデル化し、定義した上で実際のアプリケーション開発を行うことです。インターフェイス定義をきちんと行った分散オブジェクトを前提にしてアプリケーション開発を行うことによって、クライアントと分散オブジェクトの開発を分離して行うことが可能になり、システム開発の見通しがよくなります。また一度開発した分散オブジェクトは容易に再利用することができ、生産性の向上にもつながります。

このようなObjectBrokerの特性を引き出すためには、アプリケーション開発の手順が重要になります。図5.1に示すようにObjectBrokerによる開発手順はモデル化、定義、アプリケーション開発、配布という4つのステップで行われます。以下ではこのステップに沿って説明を行います。

その前に今回の説明で利用する事例の紹介をしておきます。この事例は銀行システムと呼ばれますが、顧客ごとの口座処理を行うごく簡単な事例です。この事例では顧客が預金口座に対して預金の預け入れ、引き出し、残高照会を行うというものです。それでは早速実際の開発手順を見ていきましょう。

【2】ステップ1:モデル化

ObjectBrokerに代表される分散オブジェクト環境では、システムは分散オブジェクトによって構成されます。従って、何を分散オブジェクトとするか、その分散オブジェクトはどのようなサービスを提供するか、ということが一番重要なシステム設計上の課題になります。このような最も基礎的な課題を解決するステップをモデル化と呼んでいます。

モデル化は非常に重要なステップですが、モデル化を行うための定型的な方式があるわけではありません。試行錯誤的に分散オブジェクトの対象とそのサービスを決めてシステムのプロトタイプを作成し、問題点を洗い出して再モデル化を行うという手順のくり返しが必要です。

ここで先ほどの銀行システムに関するモデル化を行ってみます。非常に簡単な事例ですので分散オブジェクトの対象としては直観的に顧客と口座という2つになります。もちろん本格的なシステムでは、分散オブジェクトの対象を選択する過程はもっと複雑になるでしょうが、モデル化された結果は直観的であることが重要です。

次にそれぞれの分散オブジェクトごとに提供するサービスを決定します。これもこの事例ではほぼ直観的に決定することができます。まず口座に関しては預け入れ、引き出し、残高照会の3つのサービスが必要です。顧客は口座へのサービス要求前に顧客の確認を行うサービスが必要です。(下表参照)どの分散オブジェクトにどのサービスを割り付けていくかという問題は分散オブジェクトを決定することよりも少しテクニカルな課題です。1つには分散オブジェクトを体系的に定義し、再利用性を高めるために継承機能の利用を検討する必要があります。また拡張性のためのポリモーフィズムの利用なども検討課題になります。

分散オブジェクト	サービス
口座	預け入れ 引き出し 残高照会
顧客	確認

モデル化をサポートしたり、プロトタイピングを行う方法論やツールが各ベンダから提供されています。現在のところ特にObjectBrokerやCORBAと相性のいい方法論があるわけではありませんし、ObjectBrokerを直接サポートするツールが提供されているわけでもありません。特にCORBAとオブジェクト指向設計手法との関係は中立的なものになっています。ツールについては将来的にはObjectBrokerを直接サポートするものが出てくると予想されます。

【3】ステップ2:定義

モデル化で決定された分散オブジェクトとその提供するサービスをObjectBrokerが処理できる形に定義していくステップが定義です。定義ではその名前の通り、ObjectBrokerが提供する3つの記述言語を利用してモデル化されたシステムを定義していきます。

まずIDLでのインターフェイス定義を行います。基本的には分散オブジェクトごとにインターフェイスを作成し、それぞれのサービスをオペレーションとして定義していけばいいのですが、オペレーションの名前、パラメータ、リターン値などをここできちんと定義しておく必要があります。ここでのIDL定義の一部分をリスト5.1に示しておきます。

次にIMLによるインプリメンテーション定義を行います。ここではインターフェイスに対応するインプリメンテーションをどのように構成するかを定義していきます。具体的にはどのオペレーションに対応したメソッドをまとめて1つのインプリメンテーションにするか、そしてそのインプリメンテーションはどのように起動するかといったことが定義されます。実は銀行システムでのIML定義IDLから自動的に生成することができます。単純なインプリメンテーションの場合はこのようにわざわざIMLを記述する必要はありません。

■ リスト5.1 ■

```
module BNK1
{
/*  中略      */

 // The Account interface, with its three operations

 interface Account {

   void Deposit  (in  AccountId  AccountNo,
                  in  float      Amount,
                  out float      CurrentBalance)
      raises  (UserExcep);

   void Withdraw (in  AccountId  AccountNo,
                  in  float      Amount,
                  out float      CurrentBalance)
      raises  (UserExcep);

   void Balance  (in  AccountId  AccountNo,
                  out float      CurrentBalance)
      raises  (UserExcep);

 }; // end of interface Account


 // The Customer interface, with one operation

 interface Customer {

   void Validate (in  CustomerId  CustomerNo,
                  out string      Name,
                  out string      Surname)
      raises  (UserExcep);

 }; //end of interface Customer

}; //end of module BNK1
```

定義ステップの最後はMMLによるマッピング定義です。動的にメソッド選択を行う場合には選択条件などの定義を行っておく必要があります。実は銀行システムではオペレーションとメソッドが1対1に対応しているのでマッピング定義の必要はありません。

これで定義ステップは終わりです。IDL、IML、MMLによる定義によって、開発を行うアプリケーションのインターフェイス定義やインプリメンテーションの枠組みが明確にされました。従ってこれ以降の実際のアプリケーション開発はクライアント側とサーバ側に完全に分離して行うことができます。

【4】ステップ3の1:クライアント開発

これから実際にアプリケーションを開発するステップに入っていきますが、まずは比較的簡単なクライアント開発から見ていきます。分散オブジェクト環境におけるクライアント・アプリケーションは様々な形態が考えられますが、基本は分散オブジェクトへリクエストを転送してその結果を受け取るという一連のプロセスにあります。このプロセスを実現するための手順を以下に述べていきます。

(a)リクエスト転送方式の選択

既に述べたようにCORBAでは2つのリクエスト転送方式を提供しているので、どちらの方式を使うかを決める必要があります。1つのアプリケーション中で2つの方式を使うことも可能です。銀行システムでは静的な転送方式を使用します。

(b)スタブまたはリポジトリの生成

静的な転送方式ではスタブの生成が、動的な転送方式ではリポジトリの生成が必須です。ただしObjectBrokerではリポジトリの生成は必ず行います。銀行システムは静的な転送方式を利用しますので、スタブの生成を行います。ObjectBrokerではスタブの生成はコマンドによって実行されます。

(c)オブジェクト管理ルーチンの開発

最初のオブジェクト・リファレンスを取得することが分散オブジェクトのクライアント・アプリケーションのキーポイントの1つです。これを実行するのがオブジェクト管理ルーチンですが、ObjectBrokerでは最初のオブジェクト・リファレンスを取得するために3つの方法を提供しています。クライアントに組み込む方法、外部ファイルに格納しておく方法、レジストリに登録しておく方法です。銀行システムではソースコードを簡単にするために外部ファイルへ格納しておく方法を利用します。外部ファイルから取り出したオブジェクト・リファレンスは格納のためのフォーマットになっていますので、ORBインターフェイスのライブラリを使って実行形式に変換します。(リスト5.2参照)

(d)リクエスト転送ルーチンの開発

実際にリクエスト転送を行うのがリクエスト転送ルーチンです。この部分は2つのリクエスト転送方式によって異なります。銀行システムでは静的な転送方式ですのでスタブをコールするだけで終わりです。(リスト5.3参照)

（ｅ）応答処理ルーチンの開発

リクエストに対する分散オブジェクトからの応答を処理するルーチンが必要です。特にエラーに対する回復処理をきちんと作っておくことはシステムの信頼性を高めるとともに、デバッグにも有効です。

（ｆ）クライアントの構築

以上のルーチンやスタブをリンクしてクライアント・アプリケーションを構築します。

【5】ステップ3の2：サーバ開発

続いて分散オブジェクトを実現するサーバの開発を見ていきます。分散オブジェクト環境におけるサーバは本質的にイベント駆動型です。つまりクライアントからのリクエストを受け入れ、そのタ

■ リスト5.2 ■

```
void InitObjects(
  CORBA_Environment *Ev,  /* in,  CORBA environment structure */
  CORBA_Object *AcctObj,  /* out, Account object */
  CORBA_Object *CustObj)  /* out, Account object */
{

/*          中略              */

  fgets(FileRec,BNK1_MAXLINELEN,FileHandle);  /* get account obj string */
  FileRec[strlen(FileRec)-1] = '\0'; /* Remove newline char from string */
  *AcctObj = CORBA_ORB_string_to_object( /* see ObjectBroker Reference Manual */
            CORBA_DEC_ORB_OBJECT,      /* ObjectBroker's ORB object */
            Ev,                        /* CORBA environment structure */
            FileRec);                  /* "stringified" object reference */

/*          中略              */

  return;
} /* end of routine InitObjects */
```

■ リスト5.3 ■

```
void Deposit(
  CORBA_Environment *Ev,     /* in,  CORBA environment structure */
  CORBA_Object *AcctObj,     /* in,  account object */
  BNK1_CustomerId *ValidId)  /* in,  valid cust ID, type spec'd by .IDL */
{

/*          中略              */

  /* Call the appropriate client stub routine; see BNK1.H for prototype */
  BNK1_Account_Deposit(
    *AcctObj,            /* account object */
    Ev,                  /* environment structure */
    &AccountNo,          /* account id structure (cust-id and acct-type ) */
    Amount,              /* amount to deposit */
    &CurrentBalance);    /* returned balance after deposit */

/*          中略              */

  return;
} /* end of routine Deposit */
```

イプをチェックし、対応するメソッドを実行するという構成になっています。

メソッドの形態としてはコマンド・スクリプトで実現する場合や動的ライブラリで実現する場合などがありますが、ここでは最も一般的なサーバ・プログラムを開発して実現する場合について以下に手順を述べていきます。

(a)スケルトンの生成

サーバの制御部分の大半はIDL、IMLから生成することができます。生成はコマンドによって実行されます。ここで生成されるスケルトンには、インプリメンテーションの登録ルーチン、メソッド・ディスパッチャ、メソッドのテンプレートなどがあります。

(b)サーバ・メインルーチンの開発

メインルーチンはサーバの初期化、インプリメンテーションの登録、活性化をまず行い、メインループに入ります。実際のメソッドのコールはORBがディスパッチャ・ルーチンから行います。最後にまたメインルーチンに戻って終了処理を行います。銀行システムのメインルーチンのリスト(リスト5.4参照)から組み立ての大枠を理解してください。

(c)オブジェクト・リファレンスの生成と管理

クライアントがリクエスト転送時に指定する必要があるオブジェクト・リファレンスをサーバは起動時に生成し、管理する必要があります。オブジェクト・リファレンスの管理のためには様々なルーチンが提供されていますのでそれらを利用して管理用のルーチンを開発します。銀行システムではオブジェクト・リファレンスの生成を行い、外部ファイルへ格納するルーチンを開発します。(リスト5.5参照)

(d)メソッドの開発

生成されたテンプレートに基づいてメソッドを開発します。メソッドの中身はそれぞれのサービスを実現するコードになります。注意する点としては分散環境におけるメモリ管理の手順です。特にクライアントから渡されたパラメータやリターン値の取り扱いが重要です。(リスト5.6参照)

(e)サーバの構築

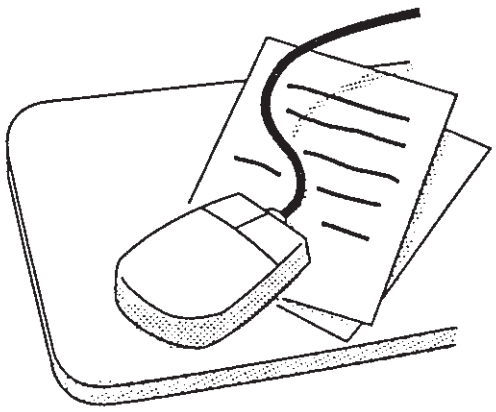
以上で開発してきたスケルトンやメインルーチン、メソッドなどをリンクしてサーバを構築します。

【6】ステップ4:配布

これまでのステップで開発されたObjectBrokerのアプリケーションを実行環境に配布する必要があります。ObjectBrokerの環境ではアプリケーションそのものだけでなく、リポジトリやレジストリなどの管理情報を設定することが必要になります。この部分はアプリケーションによって様々な形態を取りますし、一般的に議論することは困難ですが、特に大規模なシステムではシステム設計時から検討する必要のある重要な項目です。

■ リスト5.4 ■

```
main ()
{

/*          中略              */

 /* Register the implementations that this server provides */
 RegisterImpls(&Ev,
          &AccountImpl,
          &CustomerImpl);

 /* Provide initial object references to the client */
 CreateObjRefs(&Ev,
          &AccountImpl,
          &CustomerImpl);

 /* Inform ObjectBroker that account and customer implementations in this
    server are ready to receive requests. */
 CORBA_BOA_impl_is_ready(
  CORBA_DEC_BOA_OBJECT,
  &Ev,
  AccountImpl);
 CORBA_BOA_impl_is_ready(
  CORBA_DEC_BOA_OBJECT,
  &Ev,
  CustomerImpl);

 /* Enter the main loop for an ObjectBroker server program, We will wait
    in that loop until a client asks for a service we provide, at which
    point one of the routines in BNK1SM.C (the generated methods module) will
    be called. */
 OBB_BOA_main_loop(
  CORBA_DEC_ORB_OBJECT,
  &Ev,
  (CORBA_Flags)NULL);

/*          中略              */

 /* For the BNK1 example, we never expect to return here. We will discuss
    proper server rundown in a later example. */
 exit(BNK1_BFALSE);

} /* end of routine main */
```



■ リスト5.5 ■

```
void CreateObjRefs(
  CORBA_Environment *Ev,
  CORBA_ImplementationDef *AccountImpl,
  CORBA_ImplementationDef *CustomerImpl)
{

/*            中略                  */


  /* Create initial account and customer objects */
  AcctObjRef = CORBA_BOA_create(
               CORBA_DEC_BOA_OBJECT,
               Ev,
               &RefData,
               BNK1_Account__OBJ,
               *AccountImpl);

  CustObjRef = CORBA_BOA_create(
               CORBA_DEC_BOA_OBJECT,
               Ev,
               &RefData,
               BNK1_Customer__OBJ,
               *CustomerImpl);

  /* "Stringify" the object references */
  AcctObjString = CORBA_ORB_object_to_string (
                  CORBA_DEC_ORB_OBJECT,
                  Ev,
                  AcctObjRef);

  CustObjString = CORBA_ORB_object_to_string (
                  CORBA_DEC_ORB_OBJECT,
                  Ev,
                  CustObjRef);

  /* Write the stringified object references to the output file */
  fprintf(FileHandle,"%s\n",AcctObjString);
  fprintf(FileHandle,"%s\n",CustObjString);

/*            中略                  */


  return;
} /* end of routine CreateObjRefs */
```

■ リスト5.6 ■

```
void METHOD_AccountDeposit(
  CORBA_Object object,
  CORBA_Environment * Ev,
  BNK1_AccountId * AccountNo,
  CORBA_float Amount,
  CORBA_float * CurrentBalance)
{

/*          中略              */

  CurrentAccount = FindAccount(AccountNo);
  if (CurrentAccount == NULL)
  {

/*          中略              */

  }
  else
  {
    printf("\n  ...result: deposit successful\n\n");
    CurrentAccount->Info.Balance += Amount;
    *CurrentBalance = CurrentAccount->Info.Balance;
  }
  return;
} /* end of routine METHOD_AccountDeposit */
```

6. COM

【1】2つのCOM

これまでの説明では、分散オブジェクト技術の実例としてCORBAとその製品化であるObjectBrokerを紹介してきました。ところでMicrosoft社とDECは1993年に新しい分散オブジェクト技術COM(Common Object Model)の共同開発を発表しました。ここではCOMの提供する機能とその構造を説明するとともに、製品化のスケジュール、特にObjectBrokerとの関係を説明します。

その前に1つ注意する必要のある事柄があります。同じCOMという名前を持ったもう1つの技術が存在するということです。しかもこの技術ははっきりと区別する必要がありますが、これから説明するCOMと密接な関係を持った技術なのです。もう少し具体的に言うと後者のCOM(Component Object Model)はOLE2の基盤としてMicrosoft社が開発した技術です。このOLE2のCOMはOLE2が実現するコンパウンド・ドキュメントの環境の基盤となるもので、分散環境を前提としたものではありません。これに対して前者のCOMは分散オブジェクト環境を提供するための技術です。従って2つのCOMははっきりと区別する必要があります。

しかしながら同時に2つのCOMは密接な関係を持っています。OLE2によるアプリケーションの統合を推進するためには、分散環境への拡張を行う必要があります。そのためには従来のOLE2のCOMを分散オブジェクト環境へ拡張する技術開発が必要となります。これはCommon Object Modelが目的としていることそのものです。つまりCommon Object Modelは、OLE2のCOMの発

展という関係にあるわけです。それではCommon Object ModelのCOMについてより詳しく説明していきましょう。

【2】COM機能と構造

ここではCommon Object ModelのCOMについて説明をしていきますが、2つのCOMは密接な関係にあります。OLE2のCOMの発展した形態がCommon Object ModelのCOMという関係にあるので、その機能と構造は非常に類似しています。ここでの説明ではその類似した点も含めてCOMの全体像をCORBAと比較しながら紹介することにします。

●分散オブジェクト・モデル

COMにおいてもシステム構築の基本モデルをクライアント／サーバ型に置いている点はCORBAと同じです。

違いの1つは、言語バインディングをCORBAのようにプログラミング言語ごとの標準APIで実現する代わりに、バイナリ標準を定めてそれぞれのプログラミング言語に応じてそれを利用することです。つまりCOM側はプログラミング言語ごとの実装を準備する必要はなくなり、プログラミング言語への依存性はなくなりますが、コーディングの段階ではプログラミング言語ごとのバインディング・テクニックが必要になるわけです。

もう1つの違いはオブジェクトの参照方式です。これは次のオブジェクト・インターフェイスとも絡みますが、COMではオブジェクトはクライアント側からはオブジェクトへのインターフェイスのポインタで表現されます。更にこのオブジェクトへのインターフェイスは実際はそのインターフェイスが提供するメソッド・ルーチンへのポインタ・テーブルです。この形式が先ほどのバイナリ標準の実態です。つまりオブジェクト・リファレンスのような抽象的な識別子の代わりに実際のコール先のアドレスを提供しているわけです。もちろんこの場合のアドレスはリモートのオブジェクトではクライアント側の代理のモジュールのアドレスになります。

更に最初のオブジェクトの取得方法も異なります。COMではクライアントがオブジェクトを取得するためにクラス・ファクトリと呼ばれる一種のオブジェクトを提供します。このクラス・ファクトリはオブジェクトのクラスごとに存在しますがクライアントは必要なオブジェクトに対応したクラス・ファクトリへリクエストしてオブジェクトへのポインタを入手します。

●オブジェクト・インターフェイス

オブジェクトのインターフェイスについては既に述べた通り、メソッドへのポインタ・テーブルとして実現されています。ポインター・テーブルの定義などを生成するためのツールとしてCOM IDLが提供されていますが、CORBA IDLのようなシステムの主要な構成要素というよりも便利なツールといった性格のものでIDLを記述しなくてもシステム開発は可能です。

COMにおけるオブジェクト・インターフェイスで特徴的な機能としては、インターフェイス検索用のインターフェイス(IUnknown)が提供されていることです。IUnknownはCOMオブジェクトが必ず備えていなければならないインターフェイスで、オブジェクトが提供するインターフェイスを検索して、インターフェイス・ポインタを返します。COMオブジェクトは本質的に複数のインターフェイ

スをサポートしますので、1つのインターフェイス・ポインタを入手しても、そのオブジェクトのすべての機能を利用できるわけではありません。別のインターフェイスへのポインターを入手するためにはIUnknownを利用することになります。

もう1つの特徴はAggregationというオブジェクトの再利用のための機能です。これはCORBAにおける継承機能に対応するものですが、その実現方式は大きく異なっています。継承機能ではベース・インターフェイスの定義を受け継ぐことによってインターフェイスの再利用が実現されましたが、Aggregationでは、複数のインターフェイスをそのままままとめて新しいインターフェイスを定義していきます。COMオブジェクトが本質的に複数のインターフェイスをサポートするのはこの機能から来ています。Aggregationでは新しいオブジェクトを構築するためにまとめられた、インターフェイスを実現するオブジェクトをそのまま取り込んで実現します。このことによって継承機能では問題になったオペレーションに対応するメソッドの確定などがシンプルに処理できます。

●オブジェクト・インプリメンテーション

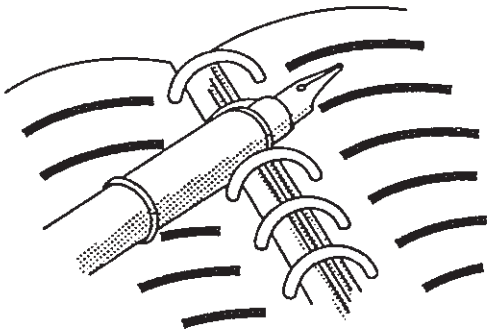
COMでは大きく分けて2種類のインプリメンテーションを提供します。1つはクライアントと同じプロセス上に存在する形態で、これは動的ライブラリによって実現されます。もう1つはクライアントとは別のプロセスとして実行する形態で、こちらは独立したサーバ・アプリケーションとして実現されます。どちらの形態でも分散環境に対応することができます。

オブジェクト・インプリメンテーションの最も重要な役割はメソッドを提供することですが、その前提としてクライアントがオブジェクトを取得するためのクラス・ファクトリを生成し、管理することが必要です。COM自身がクラス・ファクトリを管理するための様々なユーティリティを提供していますので、これらを利用してクラス・ファクトリを実現し、クライアントから利用できるように公開します。

Aggregationによってインプリメンテーションの再利用も容易に実現できます。Aggregationではとりこまれたオブジェクトのインプリメンテーションのメソッドが新しいオブジェクトのインターフェイスを実現するためにそのまま再利用できます。

●リクエスト転送

COMではリクエスト転送を行う基盤としてDCEのRPCを拡張したObject RPCを利用します。Object RPCの使用により、COMのリクエスト転送はネットワーク・プロトコルに依存することがなくなり、大規模なシステム構築にも対応できます。



オブジェクト・インプリメンテーションがサーバ・アプリケーションとして実現される場合には、クライアント側のCOM環境にプロクシーと呼ばれる分散オブジェクトの代理となるインターフェイスがクライアントのプロセス内に作られます。リクエスト・データはプロクシーに送られた後、Object RPCによってサーバ・アプリケーション側のCOM環境に転送されます。サーバ側のCOM環境にはオブジェクト・インターフェイスに対応したスタブが存在し、これが最終的にオブジェクトのメソッドを起動することになります。

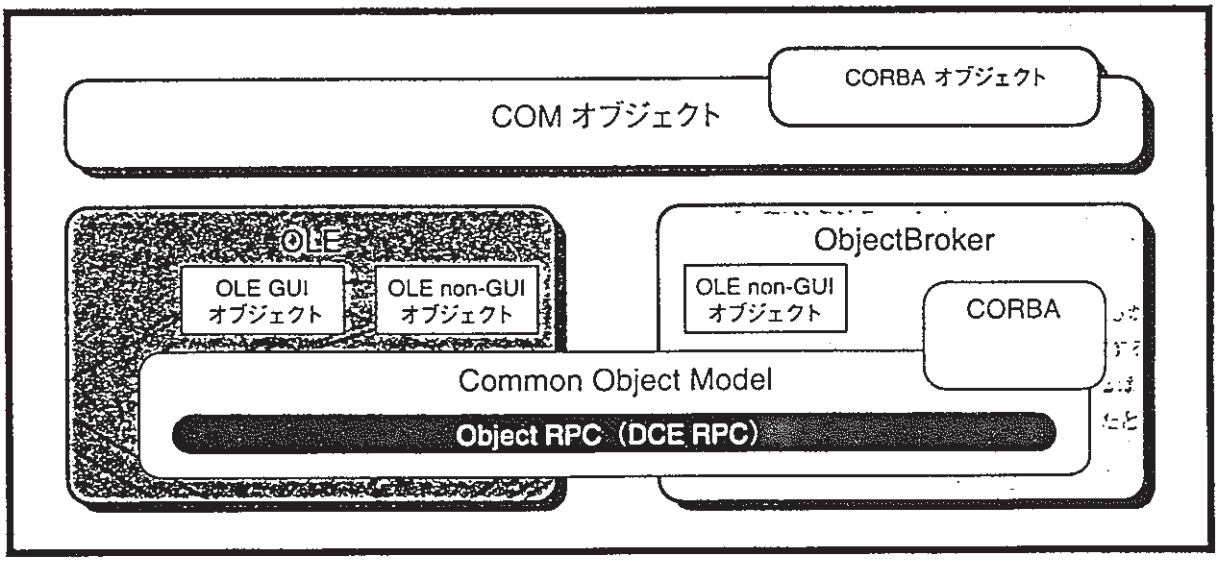
リクエスト・データのマーシャリングは、プロクシーとスタブの間でアプリケーションからは見えない形で行われます。COMは標準的なマーシャリング機能を提供していますが、カスタマイズされたマーシャリングを実現することも可能です。

【3】COMとObjectBroker

現在、Microsoft社とDECは共同でCOMの開発を行っています。この中でMicrosoft社はCOMの分散対応やインターフェイス・モデルの構築などを行っており、DECはCOMのクラス・モデルとクラスを構築するツールなどを開発しています。その中にCOMとCORBAを接続する機能の開発も含まれています。

DECではCOMで開発された技術をObjectBrokerの中に統合していくことを計画しています。1995年前半にCOMの開発環境をテスト的に提供し、1995年後半でObjectBrokerに統合した形でCOMを提供することになります。その時点でObjectBrokerはCOMとCORBAという2つの分散オブジェクト環境を統合するための基盤ソフトウェアとして提供されることになるわけです。

■ COMとObjectBroker



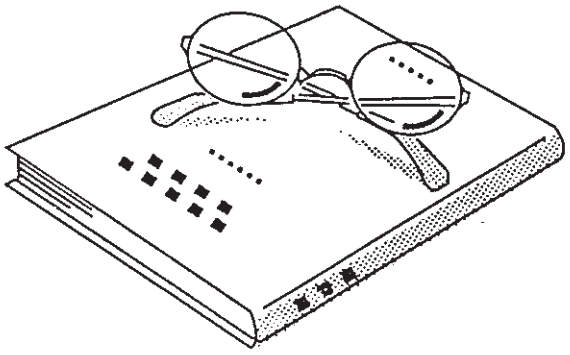
7. 開かれたシステムへ

【1】分散マルチベンダ環境の発展

これまで述べてきたように、分散マルチベンダ環境を実現するための技術基盤は確立されており、現実的な分散マルチベンダ環境によるシステム構築が可能になってきています。しかしながら、実システムとしての分散マルチベンダ環境を考えた場合、現状は分散化の面からも、多様化の面からもまだ端緒についたばかりというところです。これから始まる本格的なマルチベンダ環境の出現は、コンピュータ・システムに新しい展望を与えるとともに、新しい問題を提供することになると思われます。

まず展望の面から眺めてみると、現在存在している分散マルチベンダ環境と呼ばれるものも、実際にはせいぜい部門レベルの分散化と複数ベンダのUNIXサーバとPCを含む多様化を提供するにすぎません。これから登場する本格的な分散マルチベンダ環境は、企業レベル、社会レベル、さらには地球レベルの分散化を実現するものです。また、多様化の面からもあらゆるコンピュータ・システム、すべての電子機器、さらには日常の活動すべてに対するサービスを提供することになります。これだけの広さと深さを備えたとき、初めて分散マルチベンダ環境は完成された形態を持つことになりますし、従来のコンピュータ・システムが社会活動に与えてきた影響力を越えた「社会の全面的なコンピュータ化」を実現できることになります。もちろん、ほかの科学技術の場合と同様に、それが人類にとって幸福なことであるのか、不幸なことであるのかについては判断の分かれるところでしょう。

次に問題の面から考えてみると、分散オブジェクト技術によって分散マルチベンダ環境を構築するための基本的な枠組みは与えられました。しかしながら本格的な分散マルチベンダ環境を、現実に利用するシステムとして構築するためには、システムの信頼性、安全性を高めるための技術を提供する必要があります。例えば、フォールト・トレラントなシステムを構築するためにサーバを多重化し、その上で、実行される複数の分散オブジェクトを自由に切り替える機能や、緊急時に重要度の低い分散オブジェクトが実行を自主的に停止して緊急性の高い分散オブジェクトへリソースを明け渡す機能、さらには分散オブジェクトが複数のサーバ上へ移動しながら作業を遂行する機能などが考えられます。これらの機能を分散オブジェクト技術の上で実現するための枠組みはまだ提供されていません。しかしながら、システムの信頼性、安全性を高める技術を提供できないままで本格的な分散マルチベンダ環境を構築することは、困難であるだけでなく危険ですらあります。



【2】分散オブジェクト技術の進化

以上のような問題を解決するためには、分散オブジェクト技術そのものが進化していく必要があります。その進化の方向として、1つには現在の分散オブジェクト技術の構成技術をより洗練していくことがあります。CORBAとCOMという2つの事例を眺めただけでも、現在の分散オブジェクト技術が発展途上の技術であることは明白です。たとえば、オブジェクトの再利用技術にしても、決定的といえる方式はまだ存在しないと言うべきでしょう。この洗練のための技術開発は、分散オブジェクト技術が現実のシステム開発で利用されていく過程で、複数の技術の中から淘汰や統合を経て実現されることになるでしょう。

もう1つは、言うまでもなく先ほど述べた信頼性と安全性を提供する技術です。これは単に信頼性や安全性を提供する機能やツールを構築するだけではなく、分散オブジェクト技術の枠組みの拡張によってオブジェクトの本質的な性質として提供されるべきです。もっと具体的には、現在の分散オブジェクト技術が提供する本質的に受動的なオブジェクトから、自律的で能動的なオブジェクトが実現される必要があります。この自律的で能動的なオブジェクトは、単純に与えられたリクエストに対応して、サービスを提供することに留まらず、サービスを安定的に提供するため他のオブジェクトと協同しながら自律的に処理を実行し、必要なときには能動的に判断して実行や停止を行います。

このような自律的で能動的なオブジェクトを実現するためには、分散オブジェクト技術の進化が必要です。少なくとも現在の分散オブジェクト・モデルが依存しているクライアント/サーバ・モデルではこのような要求には対応できないでしょう。さらに、現在の分散オブジェクト技術が前提にしているオペレーティング・システムやネットワーク技術で、自律的で能動的なオペレーティングが実現できるのか否かも問題になるでしょう。

【3】開かれた社会と開かれたシステム

本格的な分散マルチベンダ環境を実現するためには、解決すべき課題がたくさんあります。しかし、確実に分散マルチベンダ環境は現実のものになりつつありますし、本格的な分散マルチベンダ環境がそう遠くない将来に実現されることになるでしょう。

それは先ほども述べた「社会の全面的なコンピュータ化」が現実のものとなるときでもあります。しかし、全面的にコンピュータ化された社会が "モダンタイムス" に表現されたような機械に支配された社会を意味しなければならない必然性があるわけではありませんし、また、そうなるべきではないはずです。

逆に分散マルチベンダ環境は本質的に開かれたシステムを指向する技術です。だれもが自由に参加し、あらゆる多様性を受け入れられる開かれた技術です。分散化と多様化を実現する開かれたシステムの構築は、自由で自主的な参加による開かれた社会の実現を可能にするはずです。そのような開かれた社会が実現されたとき、初めて分散マルチベンダ環境が完成したと言うことができるのです。

オブジェクトブローカ
スタートアップ
実習テキスト

日本ディジタルイクイップメント株式会社

1. 実習の準備

実習をおこなう前に必要な準備に関して説明します。

1.1. 開発環境

実習をおこなう計算機にはオブジェクトブローカをインストールしておく必要があります。計算機は最低でも 1 台あれば実習は行なえますが、1 台の場合にはリモート環境でのクライアント／サーバシステムの実習はできません。

オブジェクトブローカを利用してアプリケーションを開発するために必要な開発環境も計算機にインストールしておく必要があります。ここで注意が必要なプラットフォームとしてはSunOS と Windows があります。

オブジェクトブローカを利用してアプリケーションを開発する場合には、C コンパイラがANSI に準拠している必要があります。ところが SunOS の場合には、OS に付属の C コンパイラは ANSI に準拠していません。このため、SunOS でオブジェクトブローカのアプリケーションを開発する場合には、二つの方法が考えられます。

一つは ANSI に対応した C コンパイラを購入することです。詳しくは Sun の購入先におたずねください。もう一つは gnu の C コンパイラを利用する方法です。この場合、ANSI の標準ライブラリに含まれますが SunOS では提供されていない strerror と strtoul という二つの関数を用意する必要があります。

Windows 環境では Visual C++が必要です。Windows3.1 では 16bit 環境、WindowsNT では 32bit 環境をインストールしてください。Windows95 は現在は Windows3.1 用のキットを利用しますので、16bit 環境をインストールします。

1.2. ディレクトリ構成

実習用のファイルがその他のファイルと混乱しないように、自習用のディレクトリを作成してから実習を開始してください。ここではホームディレクトリは/home/user,実習用のディレクトリを/home/user/prac とします。

2. コンテキストオブジェクトの作成

2.1. コンテキストオブジェクト定義と変換

最初にコンテキストオブジェクトを作成します。コンテキストオブジェクトではインターフェースやインプリメントを定義するリポジトリファイルを指定します。

コンテキストオブジェクトを作成するには、コンテキストオブジェクト定義ファイルを作成します。コンテキストオブジェクト定義ファイルはコンテキストオブジェクト定義言語(COL)で記述します。

たとえばコンテキストオブジェクト定義ファイルを prac1.col とします。コンテキストオブジェクトで指定するリポジトリファイルを prac1.ir とすると、コンテキストオブジェクト定義ファイルの内容は次のようになります。

```
contextobject
      table obb_default_table
            obb_repository = "prac1.ir";
      end table
end contextobject
```

コンテキストオブジェクト定義ファイルが作成できましたら、定義ファイルをコンテキストオブジェクトに次のコマンドで変換します。

```
% obbldctx prac1.col
%
```

定義ファイルにエラーがなければなにもメッセージを表示しないで、変換コマンドは終了します。カレントディレクトリにある OBB_USER_CONTEXT(UNIX) あるいは OBBUSRC.CO(Windows)がコンテキストオブジェクトファイルになります。

コンテキストオブジェクトの内容を表示するには次のコマンドを実行します。

```
% obbshctx
```

2.2. コンテキストオブジェクトの指定

コンテキストオブジェクトを表示する場合には、カレントディレクトリにあるコンテキストオブジェクトがデフォルトになりますが、オプションや環境変数で参照するコンテキストオブジェクトファイルを変更することができます。

コンテキストオブジェクトを指定するオプションは-C です。たとえば実習用ディレクトリ以外でコンテキストオブジェクト表示するには次のコマンドを実行します。

```
% obbshctx -C /home/user/prac/OBB_USER_CONTEXT
```

OBB_USER_CONTEXT 環境変数にコンテキストオブジェクトファイルを指定することもできます。

```
% setenv OBB_USER_CONTEXT /home/user/prac/OBB_USER_CONTEXT
```

2.3.環境変数への参照

コンテキストオブジェクトの値として環境変数を参照することが可能です。プロパティの値にOBB_ENVIRONMENT_VARIABLEを設定しますと、プロパティと同じ名前の環境変数の値がプロパティの値になります。

コンテキストオブジェクト定義を次のように変更します。

```
contextobject
      table obb_default_table
            obb_repository = OBB_ENVIRONMENT_VARIABLE;
      end table
end contextobject
```

次にコンテキストオブジェクトをもう一度変換します。環境変数にリポジトリファイル名を設定します。

```
% setenv OBB_REPOSITORY /home/user/prac/pracl.ir
```

環境変数に設定されている値を参照するには次のコマンドを実行します。

```
% obbshctx -e
```

3. オブジェクトの定義

オブジェクトを定義します。オブジェクトの定義には、インターフェース・インプリメント・メソッドマップの３種類を定義します。

3.1. インターフェースの定義

この実習では計算機をオブジェクトとして定義します。二つの整数を加算および減算するオペレーションと、サーバを終了するオペレーションを定義します。prac1.idl に次の内容を定義します。

```
module PRACTICE {
      interface Calc {
            long plus(long arg1, long arg2);
            long minus(long arg1, long arg2);
            void end();
      };
};
```

インターフェースなどに UUID を定義します。UUID を定義するには次のコマンドを実行します。

```
% obbgen -u -f prac1.idl
```

このコマンドを実行した後で、prac1.idl に UUID が追加されていることを確認してください。

3.2. インプリメント、メソッドマップの生成

prac1.idl をもとにしてインプリメントとメソッドマップを生成します。次のコマンドでインプリメントファイル(prac1.iml)とメソッドマップファイル(prac1.mml)を生成します。

```
% obbcomp -i prac1.iml -m prac1.mml prac1.idl
```

ここでは生成したファイルは変更しません。

3.3. リポジトリの作成

作成したインターフェース、インプリメント、メソッドマップのそれぞれの定義ファイルを利用してリポジトリファイルを作成します。リポジトリファイルを作成する前に、コンテキストオブジェクトがリポジトリファイルを指定していることを確認してください。

次のコマンドでリポジトリを作成することができます。

```
% obbidrep prac1.idl prac1.iml prac1.mml
```

作成したリポジトリの内容を次のコマンドで表示することができます。

```
% obbshrep
```

4. サーバプログラムの作成

4.1. ディスパッチャファイルの生成

次のコマンドでディスパッチャファイルを生成します。

```
% obbcomp -d disp.c pracl.idl pracl.iml
```

生成したディスパッチャファイルは変更しないでコンパイルします。

インターフェースやインプリメントの定義を変更した場合にはディスパッチャファイルを生成し直す必要があります。

4.2. メソッドファイルの生成と修正

次のコマンドでメソッドファイルのテンプレートを生成します。

```
% obbcomp -t method.c pracl.idl pracl.iml
```

生成したメソッドファイルはメソッドルーチンのテンプレートを含んでいますので、メソッドの内容を追加して完成する必要があります。

```
CORBA_long CalcImpl_plus
(CORBA_Object           object,
 CORBA_Environment*     ev,
 CORBA_long             arg1,
 CORBA_long             arg2)
{
      return( arg1 + arg2 );
}
```

4.3. メインルーチンファイルの作成

メインルーチンを作成します。メインルーチンでは次のような処理を記述します。

- 初期処理
  - サーバの登録
  - オブジェクトリファレンスの作成、保存
- 終了処理
  - サーバの登録削除

メインルーチンファイルは、メソッドファイルやディスパッチャファイルを生成したときに同時に生成されるインプリメント・ヘッダファイル(ここではPRACTICE.imh)をインクルードする必要があります。

4.4. サーバの登録

サーバアプリケーションは起動した後で、オブジェクトブローカに登録します。サーバを登録するルーチンはディスパッチャファイルに入っています。登録ルーチン名はインプリ

メント定義で記述します。

```
implement CalcImpl {
      ...
      registration_routine = CalcImpl__register();
      ...
};
```

サーバアプリケーションは登録ルーチンを呼び出します。

```
CORBA_ImplementationDef ObjImpl;
CORBA_Environment       ev;
CORBA_NVList            RegServerAttrList;
CORBA_NVList            SelServerAttrList;
CORBA_Flags             flags;

ObjImpl = CalcImpl__register(CalcImpl__OBJ, &ev, NULL, NULL, 0);
```

CalcImpl__OBJ はディスパッチャファイルで定義され、インプリメント・ヘッダファイルで宣言されている外部変数で、インプリメントを登録するために必要なデータです。ここではサーバの登録属性や選択属性は指定しません。

4.5.オブジェクトリファレンスの作成と保存

クライアントがメッセージを送るために必要なオブジェクトリファレンスを作成します。オブジェクトリファレンスを生成するには CORBA_BOA_create を利用します。

```
CORBA_Object      ObjRef;
CORBA_Environment ev;
CORBA_ReferenceData     RefData;
CORBA_InterfaceDef      ObjInt;
CORBA_ImplementationDef ObjImpl;

RefData._length = 0;
RefData._maximum = 0;
RefData._buffer = NULL;

ObjRef = CORBA_BOA_create(CORBA_DEC_BOA_OBJECT, &ev, &RefData, PRACTICE_CALC__OBJ, ObjImpl);
```

CORBA_DEC_BOA_OBJECT はオブジェクトブローカが定義する定数です。RefData には作成するオブジェクトリファレンスのリファレンスデータを指定しますが、ここではなにも定義しないことにします。

作成したオブジェクトリファレンスは文字列に変換してファイルに書き出すことで、クライアントアプリケーションがオブジェクトリファレンスを利用できるようにします。

```
CORBA_string      objref_string;
CORBA_Object      objref;
CORBA_Environment ev;
FILE*             fp

objref_string = CORBA_ORB_object_to_string(CORBA_DEC_ORB_OBJECT, &ev, ObjRef);
fp = fopen("object.dat", "w");
fprintf(fp, "%s", objref_string);
fclose(fp);
CORBA_free(objref_string);
CORBA_Object_free(objref, &ev);
```

4.6.メインループ

サーバアプリケーションは初期処理を実行した後で次のようにメインループに入って、クライアントアプリケーションからのリクエストを待ちます。

```
CORBA_Status      status;
CORBA_Environmet* ev;

status = OBB_BOA_main_loop(CORBA_DEC_BOA_OBJECT, &ev, 0);
```

クライアントアプリケーションからリクエストが送られるとメインループの中で自動的にディスパッチルーチンが呼び出されて、メソッドルーチンへ分岐します。

4.7.サーバの登録削除

クライアントアプリケーションから end メッセージが送られると、サーバアプリケーションを終了することにします。サーバアプリケーションを終了するには、次のようにイベントループを終了する必要があります。

```
CORBA_void  CalcImpl_end
(CORBA_Object           object,
 CORBA_Environment*     ev)
{
      CORBA_Status            status;

      status = OBB_BOA_exit_main_loop(CORBA_DEC_BOA_OBJECT, &ev, 0);
}
```

サーバアプリケーションは終了する前に、次のように登録を削除する必要があります。

```
CORBA_Status            status;
CORBA_Environment       ev;
CORBA_ImplementationDef impl

status = OBB_BOA_impl_unregister(CORBA_DEC_BOA_OBJECT, &ev, impl, 0);
```

impl は登録ルーチンから戻り値です。

4.8.コンパイルとリンク

すべてのプログラムを作成したら、コンパイル・リンクします。コンパイルには、スタブルーチンを生成したときに自動的に生成されるヘッダファイルも必要です。リンクには-lobb オプションが必要です。SunOS で gcc を利用している場合には、strerror と strtoul をリンクすることも必要です。

5. クライアントプログラムの作成

ここではスタブ形式によるクライアントプログラムを作成します。

5.1. スタブルーチンの生成

次のコマンドでスタブルーチンを生成します。

```
% opctomp -c stub.c TEST.idl
```

生成したスタブルーチンは変更しないでコンパイルします。

5.2. オブジェクトリファレンスの変換

サーバアプリケーションが作成したオブジェクトリファレンスを変換する部分を作成します。

```
FILE*              fp;
char*              objref_string[1024];
CORBA_Object       objref;
CORBA_Environment  ev;

fp = fopen("object.dat", "r");
fscanf(fp, "%s", objref_string);
fclose(fp);
objref = CORBA_ORB_string_to_object(CORBA_DEC_ORB_OBJECT, &ev, objref_string);
```

5.3. スタブルーチンの起動

変換したオブジェクトリファレンスを引数としてスタブルーチンを呼び出すことで、メッセージを送ります。

```
CORBA_Object       objref;
CORBA_Environment  ev;
CORBA_long         arg1, arg2, result;

result = PRACTICE_Calc_plus(objref, &ev, arg1, arg2);
```

ev は例外情報でサーバアプリケーション側で発生したエラー情報が設定されています。サーバアプリケーションでの状況は、_major メンバーを参照します。

CORBA_NO_EXECPTION	正常終了
CORBA_SYSTEM_EXCEPTION	オブジェクトブローカでのエラー
CORBA_USER_EXCEPTION	サーバアプリケーションでのエラー

オブジェクトブローカでのエラーの場合には、status メンバーを参照することでエラーコードがわかります。また次のようにエラーメッセージを表示することもできます。

```
CORBA_Environment       ev;

fprintf(stderr, "%s\n", OBB_exception_errortext(&ev, 0));
```

サーバアプリケーションでのエラーの場合には、CORBA_exception_value でユーザ例外にアクセスすることができます。

5.4.オブジェクトリファレンスの解放

不要になったオブジェクトリファレンスは次のように解放します。

```
CORBA_Object      objref;
CORBA_Environment ev;

CORBA_Object_release(objref, &ev);
```

6. デバッグ

6.1. クライアントのトレース

クライアントアプリケーションを実行する場合に、環境変数 OBB_TRACE_FLAGS を設定することでトレース情報を表示することができます。

```
% setenv OBB_TRACE_FLAGS OR
```

6.2. サーバのデバッグ情報

次のコマンドを実行すると、サーバの実行状況をログファイルに記録します。ただしこのコマンドは root(UNIX)や administrator(WindowsNT)でのみ実行できます。

```
% obbmset -A -dl
```

ログファイルはサーバアプリケーションを起動するユーザのホームディレクトリに作成されます。

6.3. サーバの登録情報

サーバアプリケーションが登録した情報は次のコマンドで参照することができます。

```
% obbmsho
```

サーバアプリケーションが異常終了した場合には、サーバの登録を削除できないことになります。これは次のコマンドで削除できます。

```
% obbmstp -p
```

7. リモートアクセスの設定

7.1. プロキシの設定

クライアントアプリケーションとサーバアプリケーションを異なるノードで実行する場合には、プロキシを設定する必要があります。プロキシを設定するには、次のコマンドをサーバアプリケーションを起動するノードで実行します。

```
% obbadpxy -h client -u 1001 user1
```

この例はclientノードの1001というUIDをもったユーザがアクセスした場合には、user1としてコマンドを実行することになります。

7.2. クライアントの移動

クライアントアプリケーションを異なるノードに移動する場合には、実行イメージ以外にコンテキストオブジェクトやリポジトリファイルも移動する必要となることがあります。

8. サーバの自動起動

8.1. インプリメント情報の登録

サーバアプリケーションを自動起動するためには、インプリメントの情報を登録する必要があります。最初に次のコマンドで登録ファイルを生成します。

```
% obbcomp -a Calc.dat prac1.idl prac1.imi
```

次のコマンドで生成したファイルを登録します。

```
% obbreg -I -a Calc.dat
```