BEA Tuxedo

BEA Tuxedo CORBA アプリケーション入門

BEA Tuxedo リリース 8.0
8.0 版
2001 年 10 月 31 日

Copyright

Copyright © 2001, BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement.It is against the law to copy the software except as specifically allowed in the agreement.This document may not, in whole or in part, be copied photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems.THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Trademarks or Service Marks

BEA, WebLogic, Tuxedo, and Jolt are registered trademarks of BEA Systems, Inc. How Business Becomes E-Business, BEA WebLogic E-Business Platform, BEA Builder, BEA Manager, BEA eLink, BEA WebLogic Commerce Server, BEA WebLogic Personalization Server, BEA WebLogic Process Integrator, BEA WebLogic Collaborate, BEA WebLogic Enterprise, and BEA WebLogic Server are trademarks of BEA Systems, Inc.

All other product names may be trademarks of the respective companies with which they are associated.

BEA Tuxedo CORBA アプリケーション入門

Document Edition	Date	Software Version
8.0	2001 年 10 月 31 日	BEA Tuxedo 8.0

目次

3. BEA Tuxedo CORBA アプリケーションの開発

4. セキュリティの使い方

5. トランザクションの使い方

索引

このマニュアルについて

このマニュアルでは、BEA Tuxedo CORBA の概要、および BEA Tuxedo ソフトウェアを使用した分散 CORBA アプリケーションの開発プロセスについて説明します。

このマニュアルでは、BEA Tuxedo CORBA の各機能については説明しませんが、BEA Tuxedo CORBA プログラミング環境を使用して典型的なアプリケーションをビルドする方法について概説します。BEA Tuxedo CORBA 機能の詳細については、BEA Tuxedo オンライン・マニュアルのホーム・ページを参照してください。

このマニュアルでは、以下の内容について説明します。

- 「第 1 章 BEA Tuxedo CORBA 環境の概要」では、BEA Tuxedo 製品の CORBA 機能について概説します。
- 「第 2 章 BEA Tuxedo CORBA プログラミング環境」では、BEA Tuxedo 製品で使用できる CORBA プログラミング環境、およびこの環境を構成するコンポーネントについて説明します。
- 「第 3 章 BEA Tuxedo CORBA アプリケーションの開発」では、典型的な BEA Tuxedo CORBA アプリケーションのビルド方法について、Simpapp サンプル・アプリケーションを例に使用して説明します。
- 「第 4 章 セキュリティの使い方」では、BEA Tuxedo CORBA アプリケーションに組み込まれるセキュリティのしくみについて説明します。Security サンプル・アプリケーションを例として使用します。
- 「第 5 章 トランザクションの使い方」では、BEA Tuxedo CORBA アプリケーションに組み込まれるトランザクションのしくみについて説明します。Transactions サンプル・アプリケーションを例として使用します。

対象読者

このマニュアルは、BEA Tuxedo CORBA プログラミング環境について理解し、BEA Tuxedo 製品を使用して分散 CORBA アプリケーションを作成する必要のあるプログラマを対象としています。

e-docs Web サイト

BEA Tuxedo 製品のマニュアルは BEA 社の Web サイトで参照することができます。BEA ホーム・ページの [製品のドキュメント] をクリックするか、または http://edocs.beasys.co.jp/e-docs/index.html に直接アクセスしてください。

マニュアルの印刷方法

このマニュアルは、ご使用の Web ブラウザで一度に 1 ファイルずつ印刷できます。Web ブラウザの [ファイル] メニューにある [印刷] オプションを使用してください。

このマニュアルの PDF 版は、Web サイト上にあります。また、マニュアルの CD-ROM にも収められています。BEA Tuxedo この PDF を Adobe Acrobat Reader で開くと、マニュアル全体または一部をブック形式で印刷できます。PDF 形式を利用するには、BEA Tuxedo マニュアルのホーム・ページにある [PDF 版] ボタンをクリックして、印刷するマニュアルを選択します。

Adobe Acrobat Reader をお持ちでない場合は、Adobe 社の Web サイト (http://www.adobe.co.jp/) から無償でダウンロードできます。

関連情報

CORBA、BEA Tuxedo、分散オブジェクト・コンピューティング、トランザクション処理、C++ プログラミング、および Java プログラミングの詳細については、BEA Tuxedo オンライン・マニュアルの「Bibliography」を参照してください。

サポート情報

皆様の BEA Tuxedo マニュアルに対するフィードバックをお待ちしています。ご意見やご質問がありましたら、電子メールで **docsupport-jp@bea.com** までお送りください。お寄せいただきましたご意見は、BEA Tuxedo マニュアルの作成および改訂を担当する BEA 社のスタッフが直接検討いたします。

電子メール メッセージには、BEA Tuxedo 8.0 リリースのマニュアルを使用していることを明記してください。

BEA Tuxedo に関するご質問、または BEA Tuxedo のインストールや使用に際して問題が発生した場合は、www.bea.com の BEA WebSUPPORT を通して BEA カスタマ・サポートにお問い合わせください。カスタマ・サポートへの問い合わせ方法は、製品パッケージに同梱されているカスタマ・サポート・カードにも記載されています。

カスタマ・サポートへお問い合わせの際には、以下の情報をご用意ください。

- お客様のお名前、電子メール・アドレス、電話番号、Fax 番号
- お客様の会社名と会社の住所
- ご使用のマシンの機種と認証コード
- ご使用の製品名とバージョン
- 問題の説明と関連するエラー・メッセージの内容

表記上の規則

このマニュアルでは、以下の表記規則が使用されています。

規則	項目
太字	用語集に定義されている用語を示します。
Ctrl + Tab	2 つ以上のキーを同時に押す操作を示します。
イタリック体	強調またはマニュアルのタイトルを示します。

等幅テキスト	コード・サンプル、コマンドとオプション、データ構造とメンバ、データ型、ディレクトリ、およびファイル名と拡張子を示します。また、キーボードから入力するテキストも等幅テキストで表示します。
	例 :
	`#include <iostream.h> void main ( ) the pointer psz`
	`chmod u+w *`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

太字の等幅テキスト	コード内の重要な語を示します。 例 : `void` **`commit`** `( )`
斜体の等幅テキスト	コード内の変数を示します。 例 : `String` *`expr`*

規則	項目
大文字のテキスト	デバイス名、環境変数、および論理演算子を示します。 例 : LPT1 SIGNON OR
{ }	構文の行で、選択肢の組み合わせを示します。かっこは入力しません。
[ ]	構文の行で、オプション項目を示します。かっこは入力しません。 例 : `buildobjclient [-v] [-o name ] [-f file-list]...[-l file-list]...`
|	構文の行で、相互に排他的な選択肢の区切りとして使います。記号は入力しません。
. . .	コマンド・ラインで、以下のいずれかの場合を示します。 ■ コマンド・ラインで、同じ引数を繰り返し使用できることを示します。 ■ 文で、追加のオプション引数が省略されていることを示します。 ■ 追加のパラメータ、値、またはその他の情報を入力できることを示します。 記号は入力しません。 例 : `buildobjclient [-v] [-o name ] [-f file-list]...[-l file-list]...`
. . .	コード例または構文の行で、項目が省略されていることを示します。記号は入力しません。

1 BEA Tuxedo CORBA 環境の概要

ここでは、次の内容について説明します。

- BEA Tuxedo CORBA 環境の紹介
- BEA Tuxedo CORBA 環境の特徴

BEA Tuxedo CORBA 環境の紹介

BEA Tuxedo 製品の CORBA 環境は、性能、スケーラビリティ、および信頼性に優れたエンタープライズ・アプリケーションを開発するプログラミング・モデルとしての CORBA 標準に基づいています。BEA Tuxedo CORBA は、オンライン・トランザクション処理 (OLTP) 機能でオブジェクト・リクエスト・ブローカ (ORB) モデルを拡張します。BEA Tuxedo CORBA のデプロイメント・インフラストラクチャは、管理された環境にトランザクションに関与する安全な分散アプリケーションを配置します。

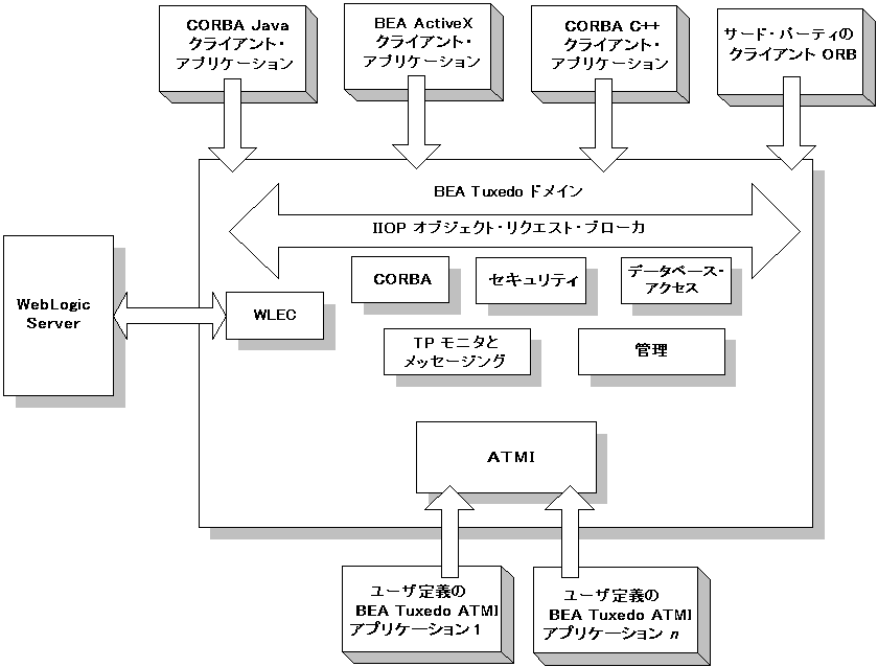
BEA Tuxedo 製品でビルドされた CORBA オブジェクトは、CORBA Object Management Group (OMG) のインターネット ORB 間プロトコル (IIOP) で通信する Web ベースのアプリケーションからアクセスできます。IIOP は、インターネットまたは企業内のイントラネットで行われる通信の標準プロトコルです。

BEA Tuxedo CORBA には、IIOP のネイティブ・インプリメンテーションがあります。このインプリメンテーションによって、インターネット、イントラネット、およびエンタープライズ・コンピューティング環境で利用できる高性能、相互運用可能な分散オブジェクト・アプリケーションが実現されます。複数のプログラミング・モデルを使用して、統合されたエンタープライズ・アプリケーションをビルドできます。CORBA アプリケーションおよびアプリケーション・トランザクション・モニタ・インターフェイス (ATMI) アプリケーションは、完全に統合されたトランザクション管理、セキュリティ、管理、および信頼性の機能を利用して開発できます。

BEA Tuxedo CORBA に組み込まれた相互運用技術は、CORBA 環境と WebLogic Server 環境のスケーラブルな接続を可能にします。相互運用性については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo の相互運用性』を参照してください。

図 1-1 は、BEA Tuxedo CORBA 環境を示しています。

図 1-1　BEA Tuxedo CORBA



次の節では、CORBA 環境の特徴をまとめています。

BEA Tuxedo CORBA 環境の特徴

BEA Tuxedo 製品の CORBA 環境には以下の特徴があります。

- C++ サーバ側 ORB
- 次のようなクライアント・アプリケーションのオプション
  - CORBA Java クライアント
  - ActiveX クライアント
  - CORBA C++ クライアント
  - サード・パーティ・クライアント ORB
- 電子商取引トランザクション・アプリケーションのホストとして実績のある実行時インフラストラクチャ。クライアント接続のコンセントレータ、高性能のメッセージ・ルーティングとロード・バランシング、および高可用性といった機能が提供されます。
- トランザクション処理 (TP) フレームワーク。CORBA アプリケーションでオブジェクトの状態とトランザクションを管理します。
- 管理情報ベース (MIB)。CORBA アプリケーションの主要な管理属性を定義します。また、MIB にアクセスするためのプログラミング・インターフェイスとスクリプト機能もあります。
- Administration Console のグラフィカル・ユーザ・インターフェイス (GUI)。CORBA アプリケーションの管理に使用します。
- CORBA トランザクション・サービス (OTS)。トランザクションが複数のプログラミング・モデル、データベース、およびアプリケーションにまたがる場合でもデータベースの整合性を確保します。
- セキュリティ・サービス。CORBA 環境の CORBA オブジェクトのリソースにアクセスする必要のあるプリンシパルの認証を処理します。
- セキュア・ソケット・レイヤ (SSL) プロトコル。クライアントとサーバのネットワーク経由の通信を暗号化します。SSL のサポートには、IIOP 接続プールも含まれています。
- CORBA のセキュリティ・サービス・プラグイン・インターフェイス (SPI)。サード・パーティのセキュリティ・プラグインを統合できるようにします。

- ノーティフィケーション・サービス。イベント・ポスト・メッセージを受信してフィルタ処理し、それらのメッセージをサブスクライバに配布します。ノーティフィケーション・サービスでは、CORBA ベースのインターフェイスと BEA 固有の単純化されたインターフェイスが提供されます。
- CosLifeCycle サービスのインプリメンテーション
- CosNaming のインプリメンテーション。BEA Tuxedo CORBA サーバ・アプリケーションで、論理名を使用してオブジェクト・リファレンスを宣言できるようにします。
- インターフェイス・リポジトリ。BEA Tuxedo CORBA オブジェクトのメタ情報を格納します。メタ情報とは、モジュール、インターフェイス、オペレーション、属性、および例外についての情報です。
- 動的起動インターフェイス (DII) のサポート。BEA Tuxedo CORBA クライアント・アプリケーションで、コンパイル時に定義されなかったオブジェクトの要求を動的に作成できます。

以降の章では、BEA Tuxedo CORBA のプログラミング環境と CORBA アプリケーションの開発プロセスについて説明します。

2 BEA Tuxedo CORBA プログラミング環境

ここでは、次の内容について説明します。

- BEA Tuxedo CORBA のプログラミング機能の概要
- BEA Tuxedo CORBA オブジェクト・サービス
- BEA Tuxedo CORBA の構成要素
- BEA Tuxedo CORBA のクライアント・アプリケーションとサーバ・アプリケーションのやり取り

BEA Tuxedo CORBA のプログラミング機能の概要

BEA Tuxedo では、分散オブジェクトの開発と管理を容易にする堅牢な CORBA プログラミング環境が提供されます。以下の節では、プログラミング環境の機能について説明します。

- IDL コンパイラ
- 開発コマンド
- 管理ツール
- ActiveX Application Builder

IDL コンパイラ

BEA Tuxedo CORBA プログラミング環境では、CORBA オブジェクトの開発を容易にする次のようなインターフェイス定義言語 (IDL) コンパイラが提供されます。

- `idl` － OMG IDL ファイルをコンパイルし、C++ でインプリメントされるインターフェイス定義で必要なクライアント・スタブ・ファイルとサーバ・スケルトン・ファイルを生成します。
- `idltojava` － Java クライアントをビルドするためのコンパイラです。OMG で定義されている IDL と Java のマッピングに基づいて IDL ファイルを Java ソース・コードにコンパイルします。この `idltojava` コンパイラは、Sun Microsystems 提供のオリジナルのコンパイラをいくつかの点で拡張したものです。BEA Tuxedo 独自の修正点は次のとおりです。
  - フラグの振る舞いとデフォルトが Sun Microsystems のマニュアルの記述とは異なります。
  - 新しい #pragma タグ `#pragma ID name Repository_id` が追加されています。
  - 新しい #pragma タグ `#pragma version name m.n` が追加されています。
  - `#pragma` 接頭辞が拡張されて、内部スコープを扱えるようになりました。空白の接頭辞が戻されます。
  - 論理区別子を利用した共用体が認められています。
  - 複合型の内部で宣言を入れ子にできます。

注記　`m3idltojava` コンパイラは、このリリースでは使用しないでください。BEA 社では、`idltojava` コンパイラを使用して CORBA Java クライアントおよび CORBA Java 共同クライアント / サーバのクライアント・スタブを生成することをお勧めします。

IDL コンパイラの使い方については、「第 3 章 BEA Tuxedo CORBA アプリケーションの開発」を参照してください。`idl` の説明については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo コマンド・リファレンス』を参照してください。

idltojava コマンドの使い方については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo コマンド・リファレンス』および『BEA Tuxedo CORBA idltojava コンパイラ』を参照してください。

開発コマンド

表 2-1 は、BEA Tuxedo CORBA プログラミング環境で CORBA アプリケーションの開発とインターフェイス・リポジトリの管理のために用意されているコマンドのリストです。

表 2-1 BEA Tuxedo CORBA の開発コマンド

開発コマンド	説明
buildobjclient	C++ クライアント・アプリケーションを作成します。
buildobjserver	C++ サーバ・アプリケーションを作成します。
genicf	インプリメンテーション・コンフィギュレーション・ファイル (ICF) を生成します。ICF ファイルでは、C++ サーバ・アプリケーションの活性化とトランザクションの方針を定義します。
id12ir	インターフェイス・リポジトリを作成して、インターフェイス定義をロードします。
ir2idl	インターフェイス・リポジトリの内容を表示します。
irdel	インターフェイス・リポジトリから指定されたオブジェクトを削除します。

開発コマンドを使用してクライアント・アプリケーションおよびサーバ・アプリケーションを開発する方法については、「第 3 章 BEA Tuxedo CORBA アプリケーションの開発」を参照してください。

開発コマンドの説明については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo コマンド・リファレンス』を参照してください。

管理ツール

BEA Tuxedo CORBA のプログラミング環境では、CORBA アプリケーションを管理するためのツール一式が提供されます。BEA Tuxedo CORBA アプリケーションは、コマンドまたはグラフィカル・ユーザ・インターフェイスを使用するか、スクリプトで管理ユーティリティを使用することで管理できます。

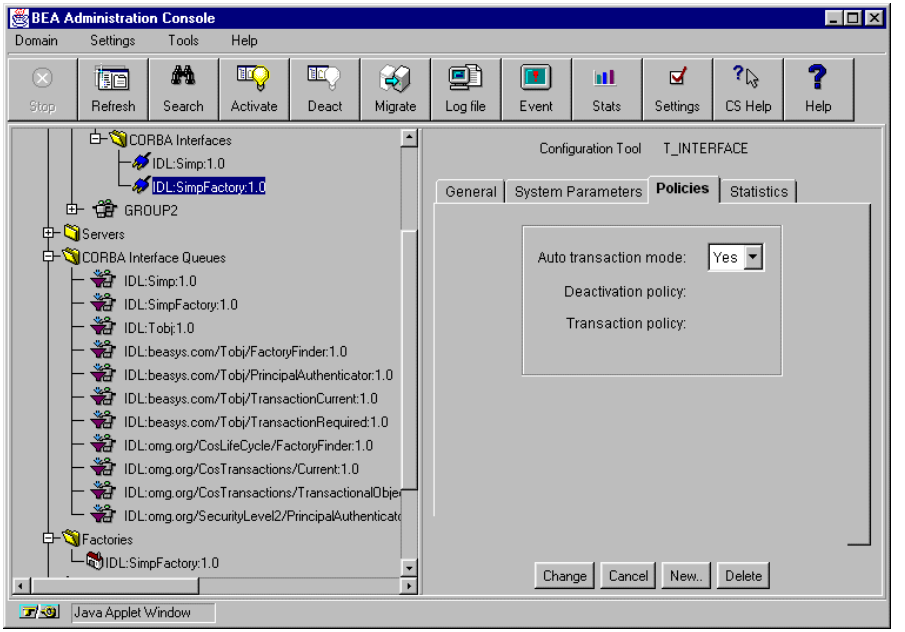
表 2-2 のリストにあるコマンドを使用すると、CORBA アプリケーションの管理タスクを実行できます。

表 2-2 管理コマンド

管理コマンド	説明
tmadmin	現在のコンフィギュレーション・パラメータの情報を表示します。
tmboot	指定されたコンフィギュレーション・ファイルで参照される BEA Tuxedo CORBA アプリケーションをアクティブにします。使用するオプションによって、アプリケーション全体またはその一部が起動します。
tmconfig	BEA Tuxedo CORBA アプリケーションのコンフィギュレーション情報を動的に更新および取得します。
tmloadcf	コンフィギュレーション・ファイルを解析し、バイナリ形式のコンフィギュレーション・ファイルをロードします。
tmshutdown	指定されたサーバ・アプリケーションをシャットダウンするか、コンフィギュレーション・ファイルからインターフェイスを削除します。
tmunloadcf	コンフィギュレーション・ファイルをアンロードします。

Administration Console は、インターネット・ブラウザにダウンロードでき、BEA Tuxedo CORBA アプリケーションのリモート管理に使用する Java ベースのアプレットです。Administration Console では、システム・イベントの監視、システム・リソースの管理、管理オブジェクトの作成とコンフィギュレーション、システム統計の表示といった管理タスクを実行できます。図 2-1 は、Administration Console のメイン・ウィンドウです。

図 2-1　Administration Console のメイン・ウィンドウ



また、AdminAPI というユーティリティ・セットも利用できます。AdminAPI を使用すると、BEA Tuxedo 製品の管理情報ベース (MIB) のシステム設定を直接アクセスおよび操作できます。AdminAPI の利点は、ログ・ファイルの監視やアプリケーションの動的な再コンフィギュレーションなどの管理タスクを自動化し、人が行う作業を減らせることです。

管理コマンドについては、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo のファイル形式とデータ記述方法』を参照してください。

Administration Console およびそのしくみの説明については、Administration Console のグラフィカル・ユーザ・インターフェイス (GUI) に統合されているオンライン・ヘルプを参照してください。

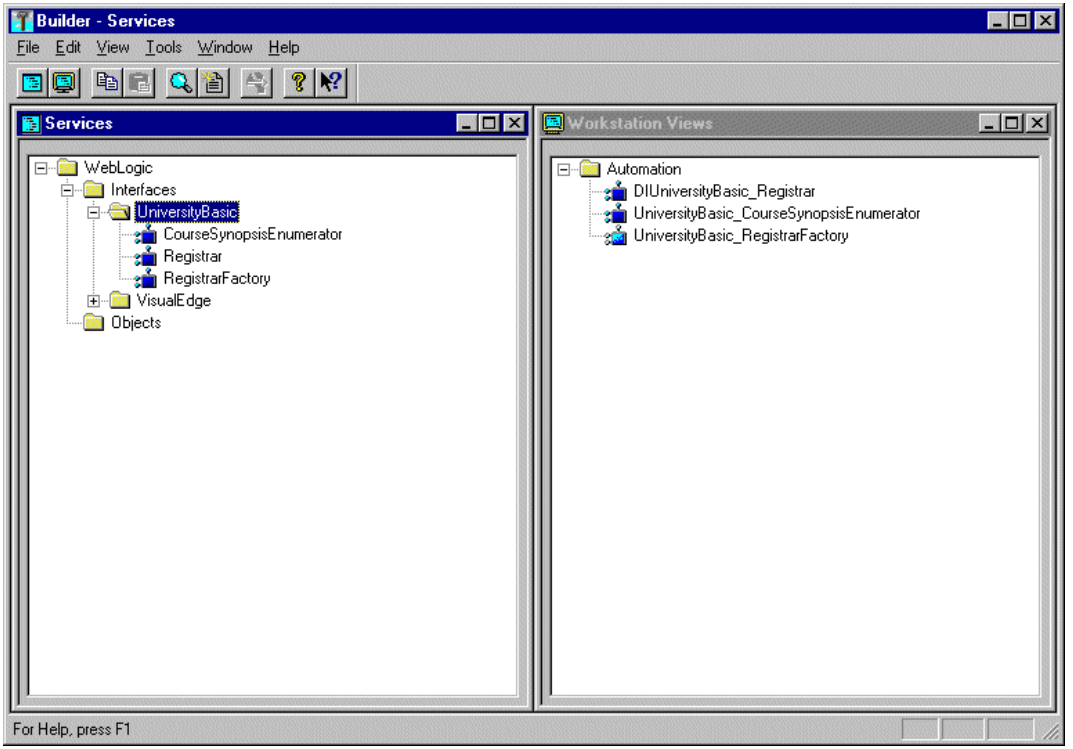
AdminAPI については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo アプリケーションの設定』を参照してください。

ActiveX Application Builder

ActiveX Application Builder は、クライアント開発ツール (Visual Basic など) と共に使用する開発ツールであり、ActiveX クライアント・アプリケーションがやり取りする BEA Tuxedo ドメイン内の CORBA インターフェイスを選択するために使用します。また、ActiveX Application Builder では、CORBA インターフェイスのオートメーション・バインディングを作成したり、CORBA オブジェクトの ActiveX ビューをクライアント・マシンにデプロイするパッケージを作成したりすることもできます。

図 2-2 は、ActiveX Application Builder のメイン・ウィンドウです。

図 2-2　ActiveX Application Builder のメイン・ウィンドウ



ActiveX Application Builder およびそのしくみの説明については、ActiveX Application Builder のグラフィカル・ユーザ・インターフェイス (GUI) に統合されているオンライン・ヘルプを参照してください。ActiveX クライアント・アプリケーションの作成については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo CORBA ActiveX』を参照してください。

BEA Tuxedo CORBA オブジェクト・サービス

BEA Tuxedo 製品には、BEA Tuxedo ドメインの CORBA クライアント・アプリケーションにオブジェクト・サービスを提供する環境オブジェクトのセットがあります。環境オブジェクトには、特定の BEA Tuxedo ドメインのサービスにアクセスするブートストラップ・プロセスを通じてアクセスします。

BEA Tuxedo CORBA では、次のサービスが提供されます。

- オブジェクト・ライフ・サイクル・サービス

  オブジェクト・ライフ・サイクル・サービスは、FactoryFinder 環境オブジェクトを通じて提供されます。FactoryFinder オブジェクトは、ファクトリを見つけるために使用できる CORBA オブジェクトです。ファクトリでは、CORBA オブジェクトのオブジェクト・リファレンスを作成できます。ファクトリと FactoryFinder オブジェクトは、CORBA サービスのライフ・サイクル・サービスのインプリメンテーションです。BEA Tuxedo CORBA アプリケーションでは、オブジェクト・ライフ・サイクル・サービスを使用してオブジェクト・リファレンスを検索します。

  オブジェクト・ライフ・サイクル・サービスの使い方については、第 2 章の 17 ページ「BEA Tuxedo CORBA のクライアント・アプリケーションとサーバ・アプリケーションのやり取り」を参照してください。

- セキュリティ・サービス

  セキュリティ・サービスには、SecurityCurrent 環境オブジェクトまたは PrincipalAuthenticator オブジェクトを通じてアクセスします。SecurityCurrent オブジェクトと PrincipalAuthenticator オブジェクトは、適切なセキュリティを備える BEA Tuxedo ドメインにアクセスするクライアント・アプリケーションを認証するために使用します。BEA Tuxedo ソフト

ウェアでは、CORBA サービスのセキュリティ・サービスのインプリメンテーションが提供されます。

セキュリティの使い方については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo CORBA アプリケーションのセキュリティ機能』を参照してください。

- トランザクション・サービス

  トランザクション・サービスには、TransactionCurrent 環境オブジェクトまたは TransactionFactory オブジェクトを通じてアクセスします。TransactionCurrent オブジェクトおよび TransactionFactory オブジェクトを使用すると、クライアント・アプリケーションがトランザクションに参加できます。BEA Tuxedo ソフトウェアでは、CORBA サービスのオブジェクト・トランザクション・サービス (OTS) のインプリメンテーションが提供されます。

  トランザクションの使い方については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo CORBA トランザクション』を参照してください。

- インターフェイス・リポジトリ・サービス

  インターフェイス・リポジトリ・サービスには、InterfaceRepository オブジェクトを通じてアクセスします。InterfaceRepository オブジェクトは、利用可能なすべての CORBA インターフェイスのインターフェイス定義と CORBA インターフェイスのオブジェクト・リファレンスを作成するファクトリが格納される CORBA オブジェクトです。InterfaceRepository オブジェクトは、DII を使用するクライアント・アプリケーションで使用します。

  DII の使い方については、『BEA Tuxedo CORBA クライアント・アプリケーションの開発方法』を参照してください。

BEA Tuxedo CORBA では、次のプログラミング環境の環境オブジェクトが提供されます。

- C++
- Java
- オートメーション (ActiveX クライアント・アプリケーションで使用)

BEA Tuxedo CORBA では、初期オブジェクト・リファレンスを取得するための、サード・パーティ・クライアントによる OMG CORBA インターオペラブル・ネーミング・サービス (INS) の使用もサポートされています。

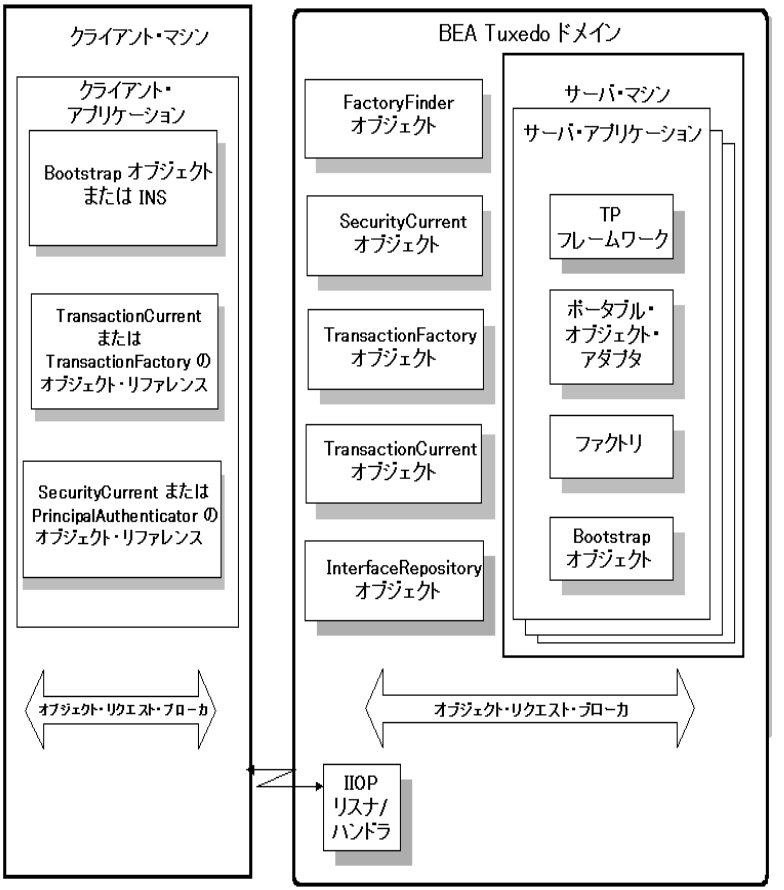
BEA Tuxedo CORBA の構成要素

この節では、BEA Tuxedo CORBA プログラミング環境の次の構成要素について説明します。

- BEA Tuxedo ドメインのブートストラップ処理
- IIOP リスナ / ハンドラ
- ORB
- TP フレームワーク

図 2-3 は、BEA Tuxedo CORBA アプリケーションの構成要素を示しています。

図 2-3　BEA Tuxedo CORBA アプリケーションの構成要素



BEA Tuxedo ドメインのブートストラップ処理

ドメインとは、オブジェクトとサービスを管理エンティティとして 1 つのグループにまとめる手段のことです。BEA Tuxedo ドメインは、少なくとも 1 つの IIOP リスナ / ハンドラを持ち、名前で識別されます。1 つのクライアント・アプリケーションから、複数の Bootstrap オブジェクトを使用して複数の BEA Tuxedo ドメインに接続できます。

クライアント・アプリケーションと BEA Tuxedo ドメインの通信は、そのドメインをブートストラップ処理することで確立されます。ブートストラップ処理のメカニズムには、BEA 社のメカニズムと OMG が規定した CORBA インターオペラブル・ネーミング・サービス (INS) のメカニズムがあります。BEA CORBA クライアント・ソフトウェアを使用する場合は、BEA 社のメカニズムを使用します。別のベンダのクライアント ORB を使用する場合は、CORBA INS メカニズムを使用します。BEA Tuxedo ドメインのブートストラップ処理の詳細については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo CORBA プログラミング・リファレンス』を参照してください。

起動後のクライアント・アプリケーションで最初に行われることの 1 つは、Bootstrap オブジェクトの作成です。その際には、次のいずれかの URL アドレス形式で IIOP リスナ / ハンドラのホストとポートを指定します。

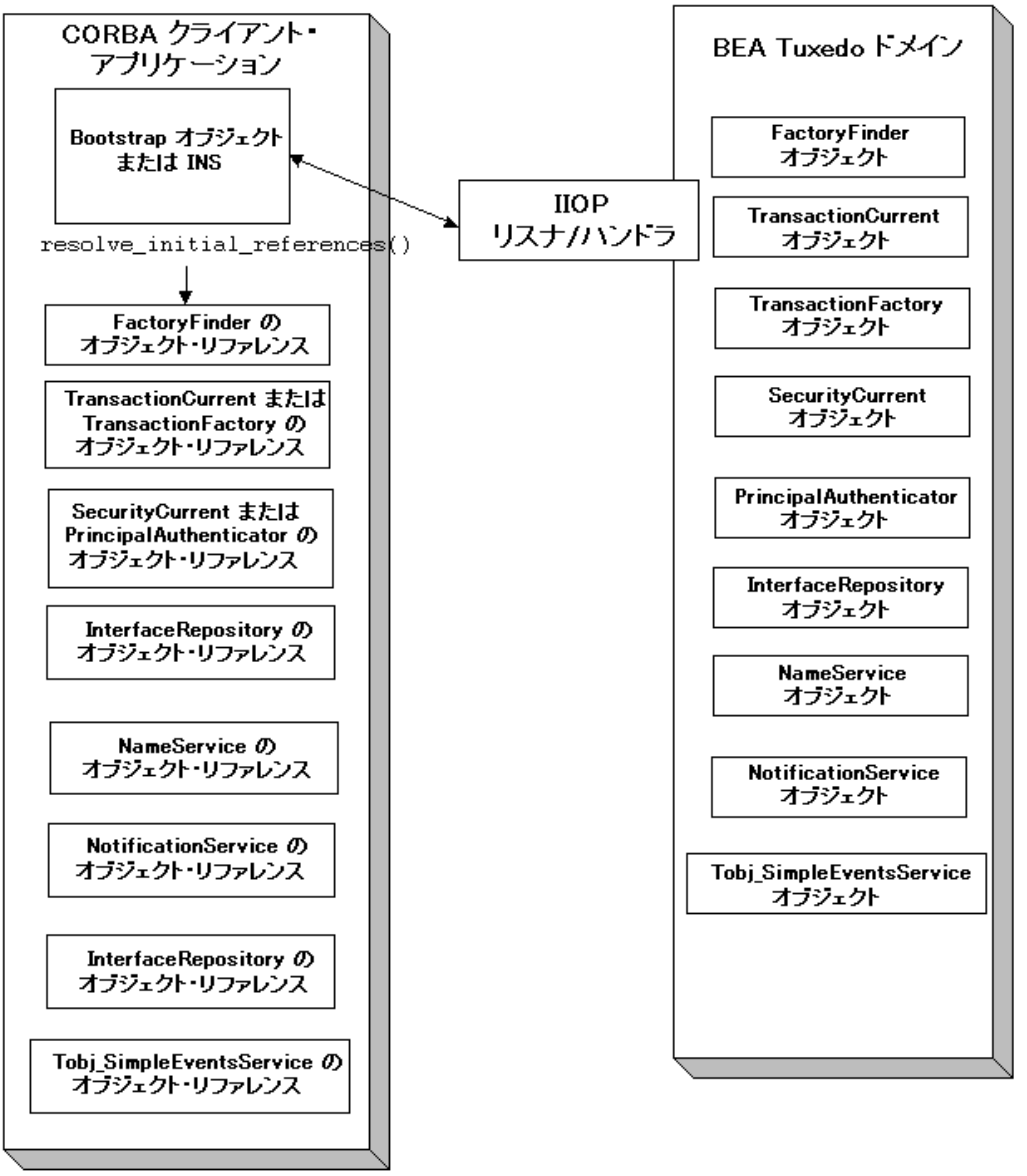
- `//host:port`
- `corbaloc://host:port`
- `corbalocs://host:port`

Bootstrap URL アドレス形式の詳細については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo CORBA アプリケーションのセキュリティ機能』を参照してください。

次に、クライアント・アプリケーションでは Bootstrap オブジェクトまたは INS ブートストラップ処理メカニズムを使用して BEA Tuxedo ドメインにあるオブジェクトのリファレンスを取得します。Bootstrap オブジェクトがインスタンス化されると、CORBA サービスを提供する BEA Tuxedo ドメイン内のオブジェクトのリファレンスを取得するために、`resolve_initial_references()` メソッドがクライアント・アプリケーションによって呼び出されます (引数として `string id` が渡される)。

図 2-4 は、Bootstrap オブジェクトまたは INS メカニズムが BEA Tuxedo ドメインでどのように機能するのかを示しています。

図 2-4　Bootstrap オブジェクトまたは INS のしくみ



IIOP リスナ / ハンドラ

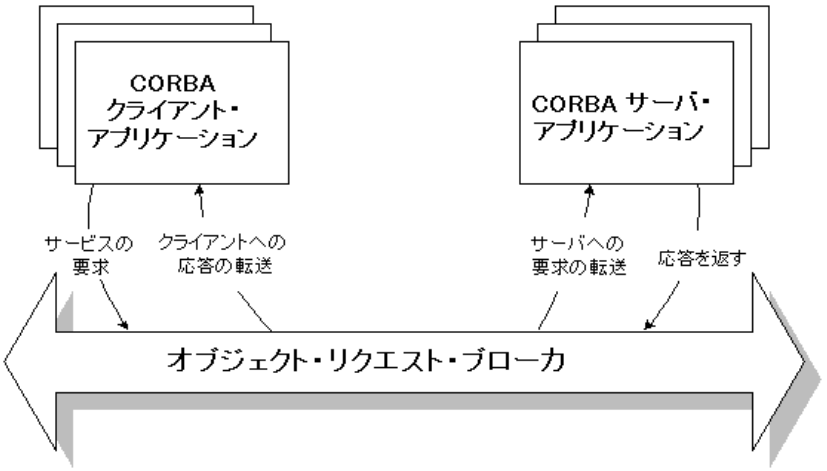
IIOP リスナ / ハンドラは、IIOP を使用して送信された CORBA クライアントの要求を受信し、その要求を適切な CORBA サーバ・アプリケーションに転送するプロセスです。IIOP リスナ / ハンドラは、通信のコンセントレータとして機能し、必要不可欠なスケーラビリティ機能を実現します。IIOP リスナ / ハンドラを使用すると、CORBA サーバ・アプリケーションでクライアント接続を管理する必要がなくなります。IIOP リスナ / ハンドラのコンフィギュレーションについては、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo アプリケーションの設定』および『BEA Tuxedo コマンド・リファレンス』の ISL コマンドの説明を参照してください。

ORB

ORB は、CORBA クライアント・アプリケーションから CORBA サーバ・アプリケーションに送信される要求の媒介として機能します。ORB が介在することで、それらのアプリケーションではお互いについての情報が必要なくなります。ORB は、要求を満たすことができるインプリメンテーションを見つけたり、要求を受信するオブジェクトのインプリメンテーションを準備したり、要求を構成するデータを伝達したりするために必要なすべてのメカニズムについて責任を持ちます。BEA Tuxedo CORBA 製品では、C++ のクライアント / サーバ ORB と BEA バージョンの Java IDL クライアント ORB (Sun Microsystems の JDK 付属) が提供されます。

図 2-5 は、ORB、CORBA クライアント・アプリケーション、および CORBA サーバ・アプリケーションの間の関係を示しています。

図 2-5 CORBA クライアント / サーバ環境の ORB



クライアント・アプリケーションから IIOP を使用して BEA Tuxedo ドメインに要求が送信されると、ORB では次の機能が実行されます。

- 各要求とその引数を調べて、必要なすべての引数が指定されていることを確認します。
- CORBA クライアント・アプリケーションからの要求を満たすことができる CORBA オブジェクトを見つけるためのメカニズムを管理します。そのために、ORB ではポータブル・オブジェクト・アダプタ (POA) とやり取りします。POA では、要求を受信するオブジェクトのインプリメンテーションが準備され、要求のデータが伝達されます。
- データをマーシャリングします。クライアント・マシンの ORB は、要求と関連付けられているデータを標準形式で書き込みます。ORB は、このデータを受信し、サーバ・アプリケーションが動作しているマシンに適切な形式に変換します。サーバ・アプリケーションからクライアント・アプリケーションにデータが送り返されるときに、ORB はデータをマーシャリングして標準形式に戻し、そのデータをクライアント・マシンの ORB に送ります。

TP フレームワーク

TP フレームワークでは、高水準の性能を実現しながら、CORBA インターフェイスの複雑さを隠蔽するプログラミング・モデルが提供されます。TP フレームワークでは CORBA アプリケーションの迅速な構築がサポートされており、デザイン・パターンに従って適切な TP アプリケーションをビルドすることが容易になっています。

TP フレームワークはポータブル・オブジェクト・アダプタ (POA) および CORBA アプリケーションとやり取りするので、アプリケーションでは POA を直接呼び出す必要がありません。また、TP フレームワークではトランザクションと状態の管理が BEA Tuxedo CORBA アプリケーションに統合されます。

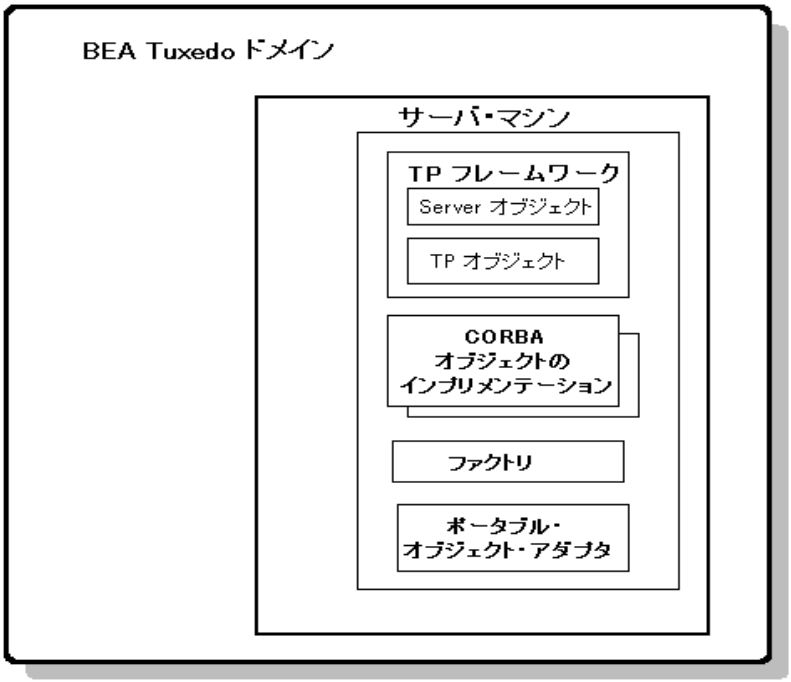
アプリケーション・プログラマは、標準の CORBA アプリケーションで必要な機能の多くを自動化するアプリケーション・プログラミング・インターフェイス (API) を使用します。アプリケーション・プログラマのすることは、CORBA アプリケーションのビジネス・ロジックを記述し、TP フレームワークのデフォルトのアクションを上書きすることだけです。

TP フレームワークの API では、CORBA アプリケーションで必要な次の機能を実行するルーチンが提供されます。

- CORBA サーバ・アプリケーションを初期化し、起動とシャットダウンのルーチンを実行する。
- オブジェクト・リファレンスを作成する。
- オブジェクト・ファクトリの登録と登録削除を行う。
- オブジェクトとオブジェクトの状態を管理する。
- CORBA サーバ・アプリケーションを BEA Tuxedo CORBA システム・リソースに関連付ける。
- ORB を取得および初期化する。
- オブジェクトのハウスキーピングを実行する。

TP フレームワークを利用すると、クライアントの要求が調和のとれた予測可能な方法で実行されます。TP フレームワークでは、BEA Tuxedo アプリケーションで利用可能なオブジェクトとサービスが適切なときに的確な順序で呼び出されます。また、TP フレームワークではオブジェクトによるシステム・リソースの再利用が最大限に行われます。図 2-6 は、TP フレームワークを示しています。

図 2-6 TP フレームワーク



TP フレームワークは単一のオブジェクトではなく、CORBA アプリケーションのデータとビジネス・ロジックを格納およびインプリメントする CORBA オブジェクトを管理するためにまとまって機能するオブジェクトの集合です。

TP フレームワークのオブジェクトの 1 つに Server オブジェクトがあります。Server オブジェクトは、サーバ・アプリケーションの初期化や解放といったタスクを実行するオペレーションをインプリメントするユーザの記述によるプログラミング・エンティティです。サーバ・アプリケーションの場合、TP フレームワークではクライアントの要求を満たすために必要な CORBA オブジェクトがインスタンス化されます。

サーバ・アプリケーションで現時点でアクティブではなく、メモリに存在しないオブジェクトを必要とするクライアントの要求が届くと、TP フレームワークではそのオブジェクトをインスタンス化するために必要なすべてのオペレーション

が調整されます。この調整には、ORB や、クライアントの要求を適切なオブジェクト・インプリメンテーション・コードに渡すための POA との調整も含まれます。

BEA Tuxedo CORBA のクライアント・アプリケーションとサーバ・アプリケーションのやり取り

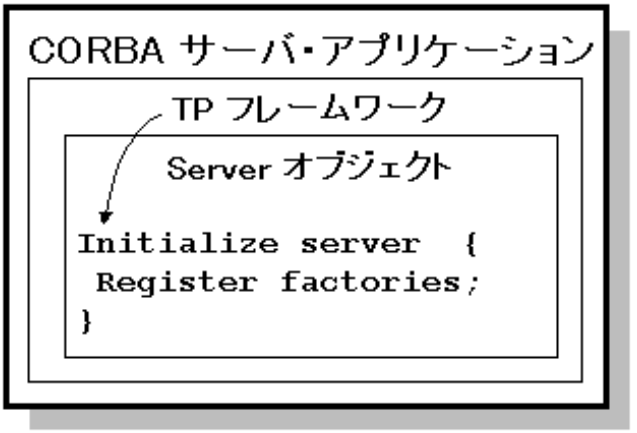
BEA Tuxedo CORBA のクライアント・アプリケーションとサーバ・アプリケーションのやり取りは次のように行われます。

1. CORBA サーバ・アプリケーションが初期化されます。
2. CORBA クライアント・アプリケーションが初期化されます。
3. BEA Tuxedo ドメインで CORBA クライアント・アプリケーションが認証されます。
4. ビジネス・ロジックを実行するために必要な CORBA オブジェクトのリファレンスが CORBA クライアント・アプリケーションで取得されます。
5. CORBA オブジェクトのオペレーションが CORBA クライアント・アプリケーションによって呼び出されます。

以降の節では、各ステップを詳しく説明します。

ステップ 1: CORBA サーバ・アプリケーションの初期化

システム管理者が、BEA Tuxedo ドメイン内のマシンで `tmboot` コマンドを入力して BEA Tuxedo CORBA サーバ・アプリケーションを起動します。TP フレームワークでは、`Server` オブジェクトの `initialize()` オペレーションを呼び出してサーバ・アプリケーションが初期化されます。

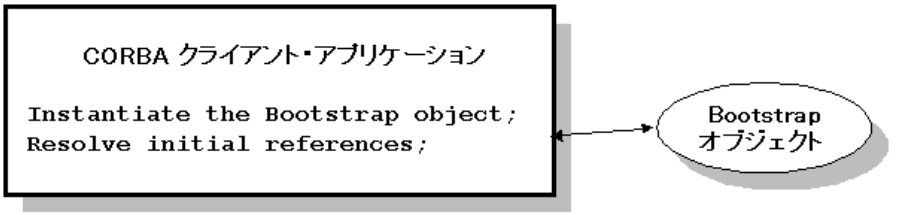


初期化プロセスで、Server オブジェクトでは次のことが行われます。

1. Bootstrap オブジェクトまたは INS を使用して FactoryFinder オブジェクトのリファレンスを取得します。
2. 通常は、FactoryFinder オブジェクトにファクトリを登録します。
3. 必要に応じて ORB のオブジェクト・リファレンスを取得します。
4. プロセス全体の初期化を実行します。

ステップ 2: CORBA クライアント・アプリケーションの初期化

初期化時に、CORBA クライアント・アプリケーションでは BEA Tuxedo ドメインにあるオブジェクトの初期リファレンスを取得します。



Bootstrap オブジェクトからは、BEA Tuxedo ドメインの FactoryFinder、SecurityCurrent、TransactionCurrent、NameService、および InterfaceRepository の各オブジェクトのリファレンスが返されます。

ステップ 3: BEA Tuxedo ドメインでの CORBA クライアント・アプリケーションの認証

BEA Tuxedo ドメインでセキュリティ・モデルが設定されている場合、BEA Tuxedo ドメインで認証を受けないと CORBA クライアント・アプリケーションでは CORBA サーバ・アプリケーションのオペレーションを呼び出すことができません。BEA Tuxedo ドメインでの認証のために、CORBA クライアント・アプリケーションでは次のことが行われます。

1. Bootstrap オブジェクトを使用して FactoryFinder オブジェクトのリファレンスを取得します。
2. PrincipalAuthenticator オブジェクトの `logon()` オペレーションを呼び出します。PrincipalAuthenticator オブジェクトは、SecurityCurrent オブジェクトから取り出します。

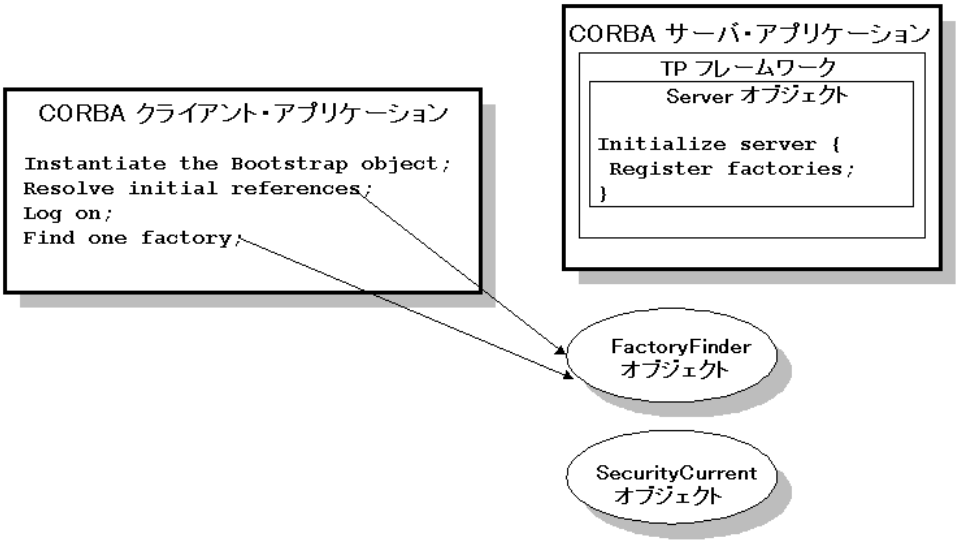
注記　証明書に基づく認証の方法については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo CORBA アプリケーションのセキュリティ機能』を参照してください。

ステップ 4: CORBA クライアント・アプリケーションによる、ビジネス・ロジックの実行に必要な CORBA オブジェクトのリファレンスの取得

CORBA クライアント・アプリケーションでは、以下のことを実行する必要があります。

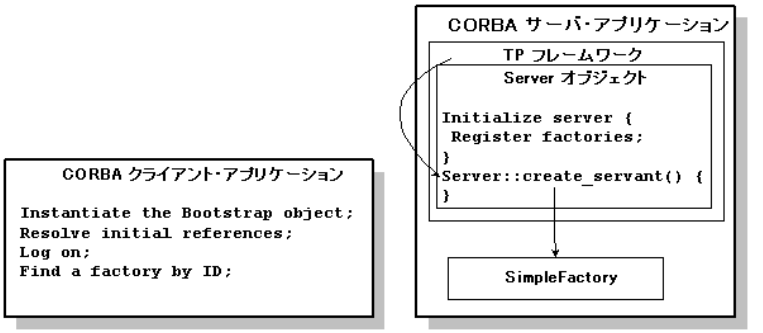
1. 必要なオブジェクトのファクトリのリファレンスを取得します。

   たとえば、クライアント・アプリケーションでは SimpleFactory オブジェクトのリファレンスが必要です。次の図で示されているように、このファクトリのリファレンスは FactoryFinder オブジェクトから取得します。

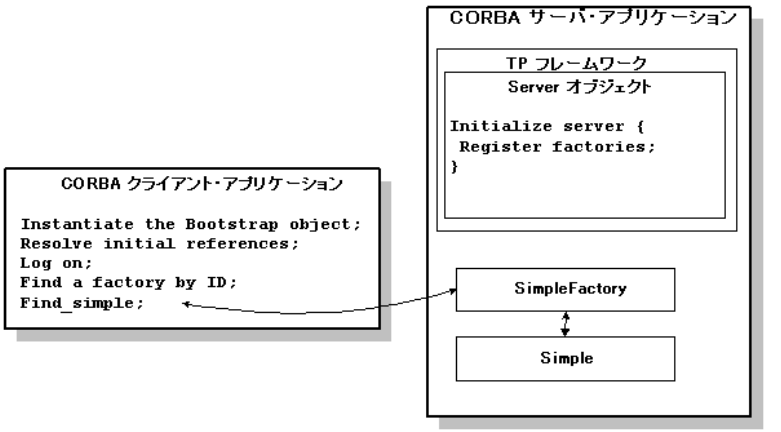


2. SimpleFactory オブジェクトを呼び出して、Simple オブジェクトのリファレンスを取得します。

   SimpleFactory オブジェクトがアクティブでない場合は、次の図で示されているように、TP フレームワークが Server オブジェクトの Server::create_servant メソッドを呼び出して SimpleFactory オブジェクトをインスタンス化します。



3. TP フレームワークでは、次の図で示されているように、SimpleFactory オブジェクトの activate_object() オペレーションと find_simple() オペレーションを呼び出して Simple オブジェクトのリファレンスを取得します。

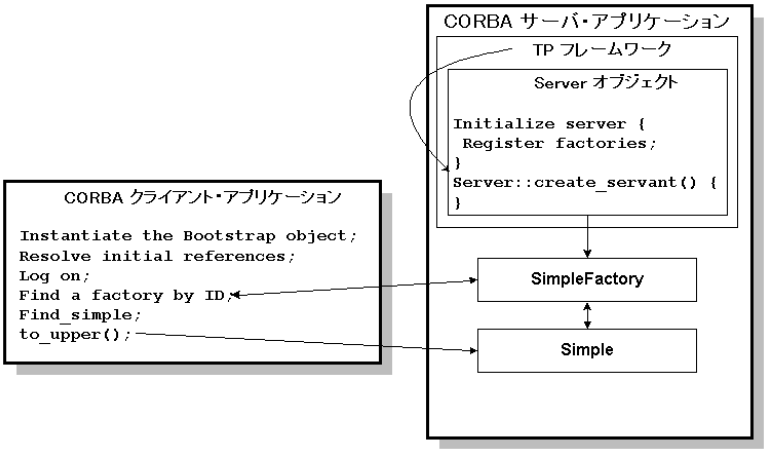


SimpleFactory オブジェクトからクライアント・アプリケーションに Simple オブジェクトのオブジェクト・リファレンスが返されます。

注記 TP フレームワークはデフォルトでオブジェクトをアクティブにするので、Simpapp サンプル・アプリケーションでは SimpleFactory オブジェクトで `activate_object()` オペレーションを明示的には使用しません。

ステップ 5: CORBA クライアント・アプリケーションによる、CORBA オブジェクトのオペレーションの呼び出し

ファクトリからクライアント・アプリケーションに返された CORBA オブジェクトのリファレンスを使用して、クライアント・アプリケーションからオブジェクトのオペレーションを呼び出します。たとえば、現時点で Simple オブジェクトのオブジェクト・リファレンスがあるので、クライアント・アプリケーションでは Simple オブジェクトの `to_upper()` オペレーションを呼び出すことができます。クライアントの要求で必要な Simple オブジェクトのインスタンスは、次の図のように作成されます。



3 BEA Tuxedo CORBA アプリケーションの開発

ここでは、次の内容について説明します。

- BEA Tuxedo CORBA アプリケーションの開発プロセスの概要
- Simpapp サンプル・アプリケーション
- ステップ 1: OMG IDL コードの記述
- ステップ 2: CORBA クライアント・スタブおよびスケルトンの生成
- ステップ 3: CORBA サーバ・アプリケーションの記述
- ステップ 4: CORBA クライアント・アプリケーションの記述
- ステップ 5: XA リソース・マネージャの作成
- ステップ 6: コンフィギュレーション・ファイルの作成
- ステップ 7: TUXCONFIG ファイルの作成
- ステップ 8: CORBA サーバ・アプリケーションのコンパイル
- ステップ 9: CORBA クライアント・アプリケーションのコンパイル
- ステップ 10: BEA Tuxedo CORBA アプリケーションの起動
- その他の BEA Tuxedo CORBA サンプル・アプリケーション

BEA Tuxedo CORBA のクライアント・アプリケーションとサーバ・アプリケーションの作成については、次の BEA Tuxedo オンライン・マニュアルを参照してください。

- 『BEA Tuxedo CORBA クライアント・アプリケーションの開発方法』
- 『BEA Tuxedo CORBA サーバ・アプリケーションの開発方法』

BEA Tuxedo CORBA アプリケーションの開発プロセスの概要

表 3-1 は、BEA Tuxedo CORBA アプリケーションの開発プロセスの概略です。

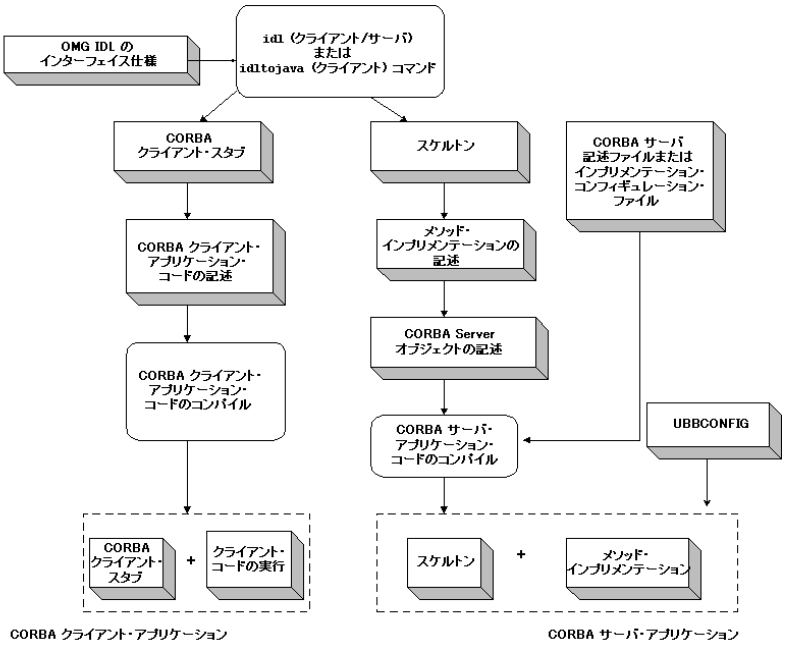
表 3-1 BEA Tuxedo CORBA アプリケーションの開発プロセス

手順	説明
1	BEA Tuxedo アプリケーションで使用する各 CORBA インターフェイスについて、Object Management Group (OMG) インターフェイス定義言語 (IDL) コードを記述します。
2	CORBA クライアント・スタブおよびスケルトンを生成します。
3	CORBA サーバ・アプリケーションを記述します。
4	CORBA クライアント・アプリケーションを記述します。
5	XA リソース・マネージャを作成します。
6	コンフィギュレーション・ファイルを作成します。
7	`TUXCONFIG` ファイルを作成します。
8	CORBA サーバ・アプリケーションをコンパイルします。
9	CORBA クライアント・アプリケーションをコンパイルします。
10	BEA Tuxedo CORBA アプリケーションを起動します。

開発プロセスの各手順は、以降の節で詳しく説明します。

図 3-1 は、BEA Tuxedo CORBA アプリケーションを開発するプロセスを示しています。

図 3-1 BEA Tuxedo CORBA アプリケーションの開発プロセス



Simpapp サンプル・アプリケーション

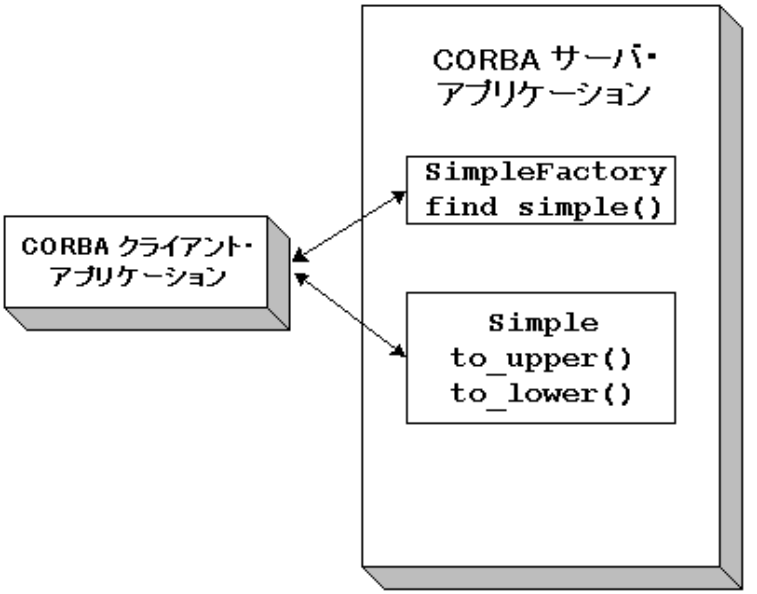
このトピックでは、Simpapp サンプル・アプリケーションを使用して開発プロセスを説明していきます。

Simpapp サンプル・アプリケーションの CORBA サーバ・アプリケーションでは、次の 2 つのメソッドを備えた CORBA オブジェクトのインプリメンテーションが提供されます。

- `upper` メソッド。CORBA クライアント・アプリケーションから受け取った文字列を大文字に変換します。
- `lower` メソッド。CORBA クライアント・アプリケーションから受け取った文字列を小文字に変換します。

図 3-2 は、Simpapp サンプル・アプリケーションのしくみを示しています。

図 3-2　Simpapp サンプル・アプリケーション



Simpapp サンプル・アプリケーションのソース・ファイルは、BEA Tuxedo ソフトウェアの `$TUXDIR\samples\corba\simpapp` ディレクトリにあります。Simpapp サンプル・アプリケーションをビルドおよび実行する手順は、同じディレクトリの `Readme.txt` ファイルに記載されています。

注記　Simpapp サンプル・アプリケーションは、CORBA C++ のクライアント・アプリケーションとサーバ・アプリケーションおよび CORBA Java のクライアント・アプリケーションのビルドを例示します。単純な ActiveX クライアント・アプリケーションのビルドについては、BEA Tuxedo オンライン・マニュアルの「Basic サンプル・アプリケーション」を参照してください。

BEA Tuxedo では、BEA Tuxedo CORBA アプリケーションの開発を例示および支援するサンプル・アプリケーションがいくつか用意されています。サンプル・アプリケーションの概要については、BEA Tuxedo オンライン・マニュアルのサンプルを参照してください。

ステップ 1: OMG IDL コードの記述

BEA Tuxedo CORBA アプリケーションの記述では、まず最初に、Object Management Group (OMG) インターフェイス定義言語 (IDL) を使用してすべての CORBA インターフェイスとそれらのメソッドを指定します。OMG IDL で記述されたインターフェイス定義では、CORBA インターフェイスが完全に定義され、各オペレーションの引数が完全に指定されます。OMG IDL は、純粋な宣言型言語です。このため、OMG IDL には実装の詳細は定義されません。OMG IDL で指定されるオペレーションは、CORBA バインディングを提供する任意の言語で記述し、呼び出すことができます。

Simpapp サンプル・アプリケーションは、表 3-2 のリストにある CORBA インターフェイスをインプリメントします。

表 3-2 Simpapp サンプル・アプリケーションの CORBA インターフェイス

インターフェイス	説明	オペレーション
SimpleFactory	Simple オブジェクトのオブジェクト・リファレンスを作成します。	find_simple()
Simple	文字列の大文字と小文字を変換します。	to_upper() to_lower()

リスト 3-1 は、Simpapp サンプル・アプリケーションの CORBA インターフェイスを定義する simple.idl ファイルを示しています。

リスト 3-1 Simpapp サンプル・アプリケーションの OMG IDL コード

```
#pragma prefix "beasys.com"

interface Simple
{
      // 文字列を小文字に変換 ( 新しい文字列を返す )
      string to_lower(in string val);

      // 文字列を大文字に変換 ( 置換 )
      void to_upper(inout string val);
};
```

```
interface SimpleFactory
{
      Simple find_simple();
};
```

ステップ 2: CORBA クライアント・スタブおよびスケルトンの生成

OMG IDL で定義されたインターフェイス仕様は、CORBA クライアント・アプリケーションの CORBA クライアント・スタブおよび CORBA サーバ・アプリケーションのスケルトンを生成するために IDL コンパイラで使用されます。CORBA クライアント・スタブは、CORBA クライアント・アプリケーションによってすべてのオペレーション呼び出しで使用されます。記述したコードと一緒にスケルトンを使用すると、CORBA オブジェクトをインプリメントする CORBA サーバ・アプリケーションを作成できます。

開発の過程では、次のコマンドのいずれかを使用して、OMG IDL ファイルをコンパイルし、BEA Tuxedo CORBA クライアント・アプリケーションおよびサーバ・アプリケーションの CORBA クライアント・スタブおよびスケルトンを生成します。

- CORBA C++ のクライアント・アプリケーションとサーバ・アプリケーションを作成する場合は、`idl` コマンドを使用します。`idl` コマンドの説明については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo コマンド・リファレンス』を参照してください。
- CORBA Java クライアントを作成する場合は、`idltojava` コマンドを使用します。`idltojava` コマンドの説明については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo コマンド・リファレンス』および『BEA Tuxedo CORBA idltojava コンパイラ』を参照してください。

表 3-3 は、`idl` コマンドで作成されるファイルのリストです。

表 3-3 idl コマンドで作成されるファイル

ファイル	デフォルト名	説明
CORBA クライアント・スタブ・ファイル	*application*_c.cpp	要求を送信するために生成されたコードを格納します。
CORBA クライアント・スタブ・ヘッダ・ファイル	*application*_c.h	OMG IDL ファイルで指定された各インターフェイスと型のクラス定義を格納します。
スケルトン・ファイル	*application*_s.cpp	OMG IDL ファイルで指定された各インターフェイスのスケルトンを格納します。実行時に、スケルトンでは CORBA クライアントの要求を CORBA サーバ・アプリケーションの適切なオペレーションにマッピングします。
スケルトン・ヘッダ・ファイル	*application*_s.h	スケルトンのクラス定義を格納します。
インプリメンテーション・ファイル	*application*_i.cpp	OMG IDL ファイルで指定されたインターフェイスのオペレーションをインプリメントするメソッドのシグニチャを格納します。
インプリメンテーション・ヘッダ・ファイル	*application*_i.h	OMG IDL ファイルで指定された各インターフェイスの初期クラス定義を格納します。

表 3-4 は、idltojava コマンドで作成されるファイルのリストです。

表 3-4 idltojava コマンドで作成されるファイル

ファイル	デフォルト名	説明
基本インターフェイス・クラス・ファイル	*interface*.java	Java で記述されたインターフェイスのインプリメンテーションを格納します。 このファイルをコピーして新しいファイルを作成し、ビジネス・ロジックをその新しいファイルに追加します。サンプルおよびこの文書の中では、このファイルの名前は規約に従って *interface*Impl.java とします。ファイル名は、インターフェイスの実際の名前に置き換えてください。この新しいファイルは、オブジェクト・インプリメンテーション・ファイルと言います。
CORBA クライアント・スタブ・ファイル	_*interface*Stub.java	要求を送信するために生成されたコードを格納します。
Holder クラス・ファイル	*interface*Holder.java	Holder クラスのインプリメンテーションを格納します。Holder クラスでは、CORBA にあるが、Java に正確にはマッピングされない、out 引数と inout 引数のオペレーションが提供されます。
Helper クラス・ファイル	*interface*Helper.java	Helper クラスのインプリメンテーションを格納します。Helper クラスでは、補助機能 (特に narrow メソッド) が提供されます。

ステップ 3: CORBA サーバ・アプリケーションの記述

BEA Tuxedo ソフトウェアでは、CORBA C++ サーバ・アプリケーションがサポートされています。CORBA サーバ・アプリケーションを作成する手順は次のとおりです。

1. 各インターフェイスのオペレーションをインプリメントするメソッドを記述します。
2. CORBA Server オブジェクトを作成します。
3. オブジェクトの活性化方針を定義します。
4. ファクトリを作成して登録します。
5. CORBA サーバ・アプリケーションを解放します。

各インターフェイスのオペレーションをインプリメントするメソッドの記述

OMG IDL ファイルをコンパイルした後は、ファイルの各インターフェイスのオペレーションをインプリメントするメソッドを記述する必要があります。インプリメンテーション・ファイルの内容は以下のとおりです。

- OMG IDL ファイルで指定された各オペレーションのメソッド宣言
- アプリケーションのビジネス・ロジック
- 各インターフェイスのインプリメンテーションのコンストラクタ (インプリメントは任意)
- `activate_object()` メソッドと `deactivate_object()` メソッド (オプション)

  `activate_object()` メソッドと `deactivate_object()` メソッドの中では、オブジェクトの活性化と非活性化に関連する特定のステップを実行するコードを記述します。詳細については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo CORBA サーバ・アプリケーションの開発方法』を参照してください。

インプリメンテーション・ファイルは手作業で記述できます。`idl` コマンドには、インプリメンテーション・ファイルのテンプレートを生成するオプションがあります。

リスト 3-2 は、Simpapp サンプル・アプリケーションの `Simple` インターフェイスと `SimpleFactory` インターフェイスの C++ インプリメンテーションを示しています。

リスト 3-2 Simple インターフェイスと SimpleFactory インターフェイスの C++ インプリメンテーション

```
// 文字列を小文字に変換する Simple_i::to_lower メソッド
// のインプリメンテーション

char* Simple_i::to_lower(const char* value)
{
     CORBA::String_var var_lower = CORBA::string_dup(value);
     for (char* ptr = var_lower; ptr && *ptr; ptr++) {
          *ptr = tolower(*ptr);
     }
     return var_lower._retn();
}

// 文字列を大文字に変換する Simple_i::to_upper メソッド
// のインプリメンテーション

void Simple_i::to_upper(char*& valuel)
{
     CORBA::String_var var_upper = value1;
     var_upper = CORBA::string_dup(var_upper.in());
     for (char* ptr = var_upper; ptr && *ptr; ptr++) {
          *ptr = toupper(*ptr);
     }
     value = var_upper._retn();
}
// Simple オブジェクトのオブジェクト・リファレンスを作成する
// SimpleFactory_i::find_simple メソッドのインプリメンテーション

Simple_ptr SimpleFactory_i::find_simple()
{
     CORBA::Object_var var_simple_oref =
          TP::create_object_reference(
               _tc_Simple->id(),
               "simple",
               CORBA::NVList::_nil()
          );
     }
```

CORBA Server オブジェクトの作成

Server オブジェクトでは、次のタスクが実行されます。

- CORBA サーバ・アプリケーションを初期化します。初期化の過程では、ファクトリの登録、CORBA サーバ・アプリケーションで必要とされるリソースの割り当て、および XA リソース・マネージャの起動 (必要な場合) が行われます。
- CORBA サーバ・アプリケーションのシャットダウンとクリーンアップを実行します。
- CORBA クライアントの要求を満たすために必要な CORBA オブジェクトをインスタンス化します。

CORBA サーバ・アプリケーションでは、Server オブジェクトは既にインスタンス化されており、Server オブジェクトのヘッダ・ファイルを利用することができます。サーバ・アプリケーションを初期化および解放するメソッド、およびサーバント・オブジェクトを作成するメソッド (必要な場合) は、独自にインプリメントしてください。

リスト 3-3 は、Simpapp サンプル・アプリケーションの Server オブジェクトの C++ コードを示しています。

リスト 3-3　CORBA C++ Server オブジェクト

```
static CORBA::Object_var static_var_factory_reference;

// サーバを起動するメソッド

CORBA::Boolean Server::initialize(int argc, char* argv[])
{
      // ファクトリのオブジェクト・リファレンスを作成

      static_var_factory_reference =
          TP::create_object_reference(
               _tc_SimpleFactory->id(),
               "simple_factory",
               CORBA::NVList::_nil()
          );
      // ファクトリ・リファレンスを FactoryFinder に登録

          TP::register_factory(
          static_var_factory_reference.in(),
```

```
            _tc_SimpleFactory->id()
      );
      return CORBA_TRUE;
}
// サーバをシャットダウンするメソッド

void Server::release()
{
// ファクトリの登録を削除

      try {
            TP::unregister_factory(
                  static_var_factory_reference.in(),
                  _tc_SimpleFactory->id()
            );
      }
      catch (...) {
            TP::userlog("Couldn't unregister the SimpleFactory");
      }
}
// サーバントを作成するメソッド

Tobj_Servant Server::create_servant(const char*
 interface_repository_id)
{
      if (!strcmp(interface_repository_id,
      _tc_SimpleFactory->id())) {
            return new SimpleFactory_i();
      }
      if (!strcmp(interface_repository_id,
      _tc_Simple->id())) {
            return new Simple_i();
      }
      return 0;
}
```

オブジェクトの活性化方針の定義

CORBA サーバを開発する過程では、オブジェクトの活性化方針を設定して、どのイベントでオブジェクトが活性化および非活性化されるのかを指定します。

CORBA サーバ・アプリケーションでは、インプリメンテーション・コンフィギュレーション・ファイル (ICF) でオブジェクトの活性化方針を指定します。`genicf` コマンドを使用すると、テンプレートの ICF ファイルが作成されます。

注記　トランザクション方針も ICF ファイルで定義します。BEA Tuxedo CORBA アプリケーションでのトランザクションの使用については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo CORBA トランザクション』を参照してください。

BEA Tuxedo ソフトウェアでは、表 3-5 のリストにある活性化方針がサポートされています。

表 3-5 活性化方針

活性化方針	説明
`method`	オブジェクトのオペレーションの呼び出しの間だけオブジェクトを活性化します。これはデフォルトの活性化方針です。
`transaction`	オペレーションが呼び出されたときにオブジェクトを活性化します。オブジェクトがトランザクションのスコープ内で活性化された場合、トランザクションがコミットされるか、ロールバックされるまでそのオブジェクトは活性化されたままです。
`process`	オペレーションが呼び出されたときにオブジェクトを活性化し、次のいずれかが起こったときのみオブジェクトを非活性化します。 ■ サーバ・アプリケーションの存在するプロセスがシャットダウンされた。 ■ `TP::deactivateEnable()` メソッド (C++) または `com.beasys.Tobj.TP.deactivateEnable()` メソッド (Java) がオブジェクトで呼び出された。

Simpapp サンプル・アプリケーションの `Simple` インターフェイスには、メソッドのデフォルトの活性化方針が割り当てられます。オブジェクトの状態の管理および活性化方針の定義の詳細については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo CORBA サーバ・アプリケーションの開発方法』を参照してください。

ファクトリの作成と登録

CORBA クライアント・アプリケーションから容易に見つかるように、CORBA サーバ・アプリケーションでファクトリを管理するには、そのファクトリを FactoryFinder オブジェクトに登録するコードを記述する必要があります。

CORBA サーバ・アプリケーションで管理されるファクトリを登録するコードを記述するには、次の手順に従います。

1. ファクトリのオブジェクト・リファレンスを作成します。

   `create_object_reference()` メソッドの呼び出しをコードに含めます。その際には、ファクトリの OMG IDL インターフェイスのインターフェイス・リポジトリ ID またはオブジェクト ID (OID) を文字列形式で指定します。ルーティング基準を指定することも可能です。

2. ファクトリを BEA Tuxedo ドメインに登録します。

   `register_factory()` メソッドを使用すると、BEA Tuxedo ドメインの FactoryFinder オブジェクトにファクトリを登録できます。`register_factory()` メソッドでは、ファクトリのオブジェクト・リファレンスと文字列識別子を指定する必要があります。

リスト 3-4 は、ファクトリを作成および登録する、Simpapp サンプル・アプリケーションのコードを示しています。

リスト 3-4 ファクトリの作成と登録の例

```
...
CORBA::Object_var v_reg_oref =
     TP:create_object_reference(
          _tc.SimpleFactory->id(),    // ファクトリのインターフェイス ID
          "simplefactory",            // オブジェクト ID
          CORBA::NVList::_nil()       // ルーティング基準
     );

     TP::register_factory(
          CORBA::Object_var v_reg_oref.in(),
          _tc_SimpleFactory->id(),
     );
...
```

リスト 3-4 については、次の点に注意してください。

- `tc.SimpleFactory->id()` は、SimpleFactory オブジェクトのインターフェイス・リポジトリ ID をタイプ・コードから抽出して指定します。
- `CORBA::NVList::_nil()` は、ルーティング基準が使用されないことを指定します。その結果として、Simple オブジェクトで作成されたオブジェクト・リファレンスは、そのオブジェクト・リファレンスを作成した SimpleFactory オブジェクトと同じグループにルーティングされます。

CORBA サーバ・アプリケーションの解放

CORBA サーバ・アプリケーションには、CORBA サーバ・アプリケーションを適切にシャットダウンするコードが必要です。`release()` メソッドは、その目的のために用意されています。`release()` メソッドの中では、次のような CORBA サーバ・アプリケーションに固有のクリーンアップ・タスクを実行できます。

- CORBA サーバ・アプリケーションで管理されるオブジェクト・ファクトリの登録削除
- リソースの割り当て解除
- データベースのクローズ
- XA リソース・マネージャのクローズ

シャットダウンの要求を受信した CORBA サーバ・アプリケーションは、ほかのリモート・オブジェクトからの要求を受信することができなくなります。このことは、管理タスクである CORBA サーバ・アプリケーションのシャットダウンの順序に影響します。たとえば、あるサーバ・プロセスがあるとして、2 番目のサーバ・プロセスの `release()` メソッドにその最初のサーバ・プロセスの呼び出しが含まれている場合はそのサーバ・プロセスをシャットダウンしないでください。

サーバをシャットダウンするときには、サーバ・アプリケーションの各ファクトリの登録を削除する必要があります。`unregister_factory()` メソッドの呼び出しは、`release()` インプリメンテーションの最初のアクションとして行わなければなりません。`unregister_factory()` メソッドでは、サーバ・アプリケーションのファクトリの登録が削除されます。このオペレーションでは、次の入力引数が必要です。

- ファクトリのオブジェクト・リファレンス
- ファクトリ・オブジェクトのインターフェイス・タイプ・コードに基づく文字列識別子 (オブジェクトの OMG IDL インターフェイスのインターフェイス・リポジトリ ID を識別するために使用する)

リスト 3-5 は、サーバ・アプリケーションを解放し、CORBA サーバ・アプリケーションのファクトリを登録解除する C++ コードを示しています。

リスト 3-5 BEA Tuxedo CORBA サーバ・アプリケーションの解放の例

```
...
public void release()
{
     TP::unregister_factory(
          factory_reference.in(),
          SimpleFactoryHelper->id
          );
}

...
```

ステップ 4: CORBA クライアント・アプリケーションの記述

BEA Tuxedo ソフトウェアでは、次のタイプの CORBA クライアント・アプリケーションがサポートされています。

- CORBA C++
- CORBA Java
- CORBA Java アプレット
- ActiveX

CORBA クライアント・アプリケーションを作成する手順は次のとおりです。

1. ORB を初期化します。
2. Bootstrap オブジェクトまたは CORBA INS ブートストラップ処理メカニズムを使用して、BEA Tuxedo ドメインとの通信を確立します。

3. FactoryFinder 環境オブジェクトの初期リファレンスを解決します。
4. ファクトリを使用して目的の CORBA オブジェクトのオブジェクト・リファレンスを取得します。
5. CORBA オブジェクトのメソッドを呼び出します。

注記 ActiveX クライアント・アプリケーションの作成については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo CORBA ActiveX』を参照してください。

CORBA クライアントの開発手順は、リスト 3-6 と リスト 3-7 で説明されています。それらのリストの内容は、Simpapp サンプル・アプリケーションのコードです。Simpapp サンプル・アプリケーションの CORBA クライアント・アプリケーションは、ファクトリを使用して Simple オブジェクトのオブジェクト・リファレンスを取得し、Simple オブジェクトの `to_upper()` メソッドと `to_lower()` メソッドを呼び出します。

リスト 3-6 Simpapp サンプル・アプリケーションの CORBA クライアント・アプリケーション

```
int main(int argc, char* argv[])
{
    try {
      //ORB を初期化
      CORBA::ORB_var var_orb = CORBA::ORB_init(argc, argv, "");

      // Bootstrap オブジェクトを作成
      Tobj_Bootstrap bootstrap(var_orb.in(), "");

      // Bootstrap オブジェクトを使用して FactoryFinder を検索
      CORBA::Object_var var_factory_finder_oref =
        bootstrap.resolve_initial_references("FactoryFinder");

      // FactoryFinder をナロー変換
      Tobj::FactoryFinder_var var_factory_finder_reference =
       Tobj::FactoryFinder::_narrow
       (var_factory_finder_oref.in());

      // ファクトリ・ファインダを使用して Simple ファクトリを検索
      CORBA::Object_var var_simple_factory_oref =
      var_factory_finder_reference->find_one_factory_by_id(
      _tc_SimpleFactory->id()
      );

      // Simple ファクトリをナロー変換
      SimpleFactory_var var_simple_factory_reference =
```

```
        SimpleFactory::_narrow(
                                var_simple_factory_oref.in());

      // Simple オブジェクトを検索
      Simple_var var_simple =
        var_simple_factory_reference->find_simple();

      // ユーザから文字列を取得
      cout << "String?";
      char mixed[256];
      cin >> mixed;

      // 文字列を大文字に変換
      CORBA::String_var var_upper = CORBA::string_dup(mixed);
      var_simple->to_upper(var_upper.inout());
      cout << var_upper.in() << endl;

      // 文字列を小文字に変換
      CORBA::String_var var_lower = var_simple->to_lower(mixed);
      cout << var_lower.in() << endl;

      return 0;
      }
}
```

リスト 3-7 Simpapp サンプル・アプリケーションの Java クライアント・アプリケーション

```
public class SimpleClient
{
      public static void main(String args[])

      //ORB を初期化
      ORB orb = ORB.init(args, null);

      // Bootstrap オブジェクトを作成
      Tobj_Bootstrap bootstrap = new Tobj_Bootstrap(orb, "");

      // Bootstrap オブジェクトを使用して FactoryFinder を検索
      org.omg.CORBA.Object factory_finder_oref =
      bootstrap.resolve_initial_references("FactoryFinder");

      // FactoryFinder をナロー変換
      FactoryFinder factory_finder_reference =
      FactoryFinderHelper.narrow(factory_finder_oref);

      // FactoryFinder を使用して Simple ファクトリを検索
      org.omg.CORBA.Object simple_factory_oref =
      factory_finder_reference.find_one_factory_by_id
      (SimpleFactoryHelper.id());
```

```
        // Simple ファクトリをナロー変換
        SimpleFactory simple_factory_reference =
        SimpleFactoryHelper.narrow(simple_factory_oref);

        // Simple オブジェクトを検索
        Simple simple = simple_factory_reference.find_simple();

        // ユーザから文字列を取得
        System.out.println("String?");
        String mixed = in.readLine();

        // 文字列を大文字に変換
        org.omg.CORBA.StringHolder buf = new
        org.omg.CORBA.StringHolder(mixed);
             simple.to_upper(buf);
        System.out.println(buf.value);

        // 文字列を小文字に変換
        String lower = simple.to_lower(mixed);
        System.out.println(lower);
          }
}
```

ステップ 5: XA リソース・マネージャの作成

BEA Tuxedo CORBA アプリケーションでトランザクションを使用する場合は、BEA Tuxedo CORBA アプリケーションの代わりにデータベースとやり取りするリソース・マネージャの CORBA サーバ・プロセスを作成する必要があります。使用するリソース・マネージャは、X/OPEN XA 仕様に準拠していなければなりません。リソース・マネージャについての次の情報が必要です。

- XA リソース・マネージャの名前を格納する xa_switch_t 型の構造体の名前
- XA リソース・マネージャの機能を示すフラグと実際の XA 関数の関数ポインタ
- XA インターフェイスのサービスを提供するオブジェクト・ファイルの名前
- XA リソース・マネージャの開閉に必要なコマンド。この情報は、UBBCONFIG コンフィギュレーション・ファイルの OPENINFO パラメータと CLOSEINFO パラメータで指定されます。

新しい XA リソース・マネージャを BEA Tuxedo システムに統合するときには、XA リソース・マネージャについての情報が格納されるようにファイル $TUXDIR\udataobj\RM を更新する必要があります。その情報は、XA リソース・マネージャの適切なライブラリを使用するため、そしてトランザクション・マネージャと XA リソース・マネージャのインターフェイスを自動的かつ適切に設定するために使用します。このファイルの形式は次のとおりです。

rm_name:rm_structure_name:library_names

rm_name は XA リソース・マネージャの名前、*rm_structure_name* は XA リソース・マネージャの名前を定義する xa_switch_t 構造体の名前、*library_names* は XA リソース・マネージャのオブジェクト・ファイルのリストです。空白類 (タブまたはスペース) は各値の前後で使用でき、*library_names* の中に挿入することもできます。コロン (:) は、どの値にも挿入できません。「#」マークで始まる行は、コメントと判断されて無視されます。

buildtms コマンドを使用すると、XA リソース・マネージャのサーバ・プロセスをビルドできます。buildtms コマンドの結果として生じるファイルは、$TUXDIR\bin ディレクトリにインストールする必要があります。

buildtms コマンドの詳細については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo コマンド・リファレンス』を参照してください。

ステップ 6: コンフィギュレーション・ファイルの作成

BEA Tuxedo ソフトウェアは優れた柔軟性と多くのオプションをアプリケーション設計者やプログラマに提供するので、同じ CORBA アプリケーションが 2 つできることはありません。たとえば、アプリケーションは小さく、単純であることもあれば (マシン上で動作する単一のクライアントやサーバ)、数千単位のクライアント・アプリケーションとサーバ・アプリケーションをまたがるトランザクションを処理できるぐらいに複雑になる場合もあります。このため、管理の対象となるすべての BEA Tuxedo CORBA アプリケーションについて、システム管理者はそのアプリケーションの構成要素 (ドメイン、サーバ・アプリケーション、クライアント・アプリケーション、インターフェイスなど) を定義および管理するコンフィギュレーション・ファイルを用意する必要があります。

コンフィギュレーション・ファイルを作成するときには、アプリケーションの実行可能バージョンを作成するために BEA Tuxedo ソフトウェアによって解釈されるパラメータのセットを使用して BEA Tuxedo CORBA アプリケーションを記述することになります。管理の設定段階において、システム管理者のする仕事はコンフィギュレーション・ファイルを作成することです。コンフィギュレーション・ファイルには、表 3-6 のリストにあるセクションが格納されます。

表 3-6 BEA Tuxedo CORBA アプリケーションのコンフィギュレーション・ファイルのセクション

コンフィギュレーション・ファイルのセクション	説明
RESOURCES	BEA Tuxedo CORBA アプリケーションのデフォルト (ユーザ・アクセスやメインの管理マシンなど) を定義します。
MACHINES	BEA Tuxedo CORBA アプリケーションで動作する各マシンのハードウェア固有の情報を定義します。
GROUPS	サーバ・アプリケーションまたは CORBA インターフェイスの論理的なグループを定義します。
SERVERS	BEA Tuxedo CORBA アプリケーションで使用されるサーバ・アプリケーション・プロセス (トランザクション・マネージャなど) を定義します。
SERVICES	BEA Tuxedo アプリケーションで提供されるサービスのパラメータを定義します。
INTERFACES	BEA Tuxedo CORBA アプリケーションの CORBA インターフェイスについての情報を定義します。
ROUTING	BEA Tuxedo CORBA アプリケーションのルーティング基準を定義します。

リスト 3-8 は、Simpapp サンプル・アプリケーションのコンフィギュレーション・ファイルを示しています。

リスト 3-8 Simpapp サンプル・アプリケーションのコンフィギュレーション・ファイル

```
*RESOURCES
      IPCKEY     55432
      DOMAINID   simpapp
      MASTER     SITE1
      MODEL      SHM
      LDBAL      N

*MACHINES
      "PCWIZ"
      LMID          = SITE1
      APPDIR        = "C:\TUXDIR\MY_SIM~1"
      TUXCONFIG     = "C:\TUXDIR\MY_SIM~1\results\tuxconfig"
      TUXDIR        = "C:\TUXDIR"
      MAXWSCLIENTS = 10

*GROUPS
      SYS_GRP
      LMID    = SITE1
      GRPNO   = 1
      APP_GRP
      LMID    = SITE1
      GRPNO   = 2

*SERVERS
      DEFAULT:
          RESTART = Y
          MAXGEN  = 5
      TMSYSEVT
          SRVGRP  = SYS_GRP
          SRVID   = 1
      TMFFNAME
          SRVGRP  = SYS_GRP
          SRVID   = 2
          CLOPT   = "-A -- -N -M"
      TMFFNAME
          SRVGRP  = SYS_GRP
          SRVID   = 3
          CLOPT   = "-A -- -N"
      TMFFNAME
          SRVGRP  = SYS_GRP
          SRVID   = 4
          CLOPT   = "-A -- -F"
      simple_server
          SRVGRP  = APP_GRP
          SRVID   = 1
```

```
        RESTART = N
    ISL
        SRVGRP  = SYS_GRP
        SRVID   = 5
        CLOPT   = "-A -- -n //PCWIZ:2468"

*SERVICES
```

ステップ 7: TUXCONFIG ファイルの作成

コンフィギュレーション・ファイルには、次の 2 つの形式があります。

- ASCII 形式のコンフィギュレーション・ファイル。任意のエディタで作成および編集します。BEA Tuxedo のマニュアル全体を通して、ASCII 形式のコンフィギュレーション・ファイルは UBBCONFIG ファイルと呼びます。コンフィギュレーション・ファイルの名前は、任意の名前を使用できます。
- TUXCONFIG ファイル。tmloadcf コマンドで作成されるバイナリ形式の UBBCONFIG ファイルです。tmloadcf コマンドを実行するときには、環境変数 TUXCONFIG を TUXCONFIG ファイルの名前とディレクトリに設定しなければなりません。tmloadcf コマンドでは、コンフィギュレーション・ファイルがバイナリ形式に変換されて、コマンドで指定された位置に書き込まれます。

tmloadcf コマンドの詳細については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo コマンド・リファレンス』を参照してください。

ステップ 8: CORBA サーバ・アプリケーションのコンパイル

C++ サーバ・アプリケーションのコンパイルとリンクには、buildobjserver コマンドを使用します。buildobjserver コマンドの形式は次のとおりです。

```
buildobjserver [-o servername] [options]
```

buildobjserver コマンドの構文要素を次に説明します。

- -o servername は、このコマンドで生成されるサーバ・アプリケーションの名前を表します。
- *options* は、buildobjserver コマンドのコマンド行オプションを表します。

マルチスレッドをサポートするサーバ・アプリケーションを作成する場合は、アプリケーションをビルドするときに buildobjserver コマンドで -t オプションを指定する必要があります。マルチスレッドをサポートするサーバ・アプリケーションの作成については、『BEA Tuxedo CORBA サーバ・アプリケーションの開発方法』を参照してください。

ステップ 9: CORBA クライアント・アプリケーションのコンパイル

CORBA クライアント・アプリケーションの開発の最終ステップは、実行可能なクライアント・アプリケーションを生成することです。そのためには、コードをコンパイルしてクライアント・スタブに対してリンクする必要があります。

CORBA C++ クライアント・アプリケーションを作成する場合、buildobjclient コマンドを使用して BEA Tuxedo CORBA クライアント・アプリケーションの実行可能ファイルを生成します。このコマンドでは、静的起動を使用するインターフェイスの CORBA クライアント・スタブ、関連付けられるヘッダ・ファイル、および標準の BEA Tuxedo ライブラリを組み合わせて CORBA クライアントの実行可能ファイルが生成されます。buildobjclient コマンドの構文については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo コマンド・リファレンス』を参照してください。

CORBA Java クライアント・アプリケーションを作成する場合、javac コマンドを使用して CORBA クライアント・アプリケーションの実行可能プログラムを生成します。Java クライアント・アプリケーションをコンパイルするときには、CLASSPATH で tuxdir\udataobj\java\jdk\m3envobj.jar ファイルを設定する必要があります。m3envobj.jar ファイルには、BEA Tuxedo 環境オブジェクトの Java クラスが格納されます。

ステップ 10: BEA Tuxedo CORBA アプリケーションの起動

tmboot コマンドを使用すると、BEA Tuxedo CORBA アプリケーションでサーバ・プロセスを起動できます。通常、CORBA アプリケーションは、UBBCONFIG ファイルの RESOURCES セクションで MASTER として指定されたマシンから起動します。

tmboot コマンドが実行可能ファイルを見つけられるようにするために、BEA Tuxedo システムのプロセスは $TUXDIR\bin ディレクトリに配置する必要があります。サーバ・アプリケーションは、コンフィギュレーション・ファイルで指定されているとおりに APPDIR に配置されていなければなりません。

サーバ・アプリケーションの起動時には、tmboot コマンドはコンフィギュレーション・ファイルから CLOPT、SEQUENCE、SRVGRP、SRVID、および MIN パラメータを使用します。サーバ・アプリケーションは、コンフィギュレーション・ファイルでの順序で起動されます。

tmboot コマンドの使い方については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo のファイル形式とデータ記述方法』を参照してください。

その他の BEA Tuxedo CORBA サンプル・アプリケーション

サンプル・アプリケーションは、BEA Tuxedo CORBA アプリケーションの開発に伴うタスクを例示し、BEA Tuxedo CORBA アプリケーションのビルドでプログラマが利用できるサンプル・コードを提供します。サンプル・アプリケーションのコードは、BEA Tuxedo 製品のすべての情報トピックで開発や管理の手順を説明するために使用します。

表 3-7 は、その他の BEA Tuxedo CORBA サンプル・アプリケーションを説明しています。

表 3-7 BEA Tuxedo CORBA サンプル・アプリケーション

BEA Tuxedo CORBA サンプル・アプリケーション	説明
Simpapp	CORBA C++ クライアント・アプリケーション、CORBA Java クライアント・アプリケーション、および C++ サーバ・アプリケーションを提供します。C++ サーバ・アプリケーションには、C++ クライアント・アプリケーションから受信した文字列を処理する 2 つのオペレーションがあります。
Basic	BEA Tuxedo CORBA のクライアント・アプリケーションとサーバ・アプリケーションを開発し、BEA Tuxedo アプリケーションをコンフィギュレーションする方法を説明します。C++ サーバ・アプリケーションと CORBA C++、CORBA Java、および ActiveX のクライアント・アプリケーションのビルドを例示します。
Security	BEA Tuxedo CORBA アプリケーションへの BEA Tuxedo 認証の追加を例示します。Security サンプル・アプリケーションのビルドと実行については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo CORBA アプリケーションのセキュリティ機能』を参照してください。

表 3-7 BEA Tuxedo CORBA サンプル・アプリケーション (続き)

BEA Tuxedo CORBA サンプル・アプリケーション	説明
Transactions	Basic サンプル・アプリケーションの CORBA C++ サーバ・アプリケーションと CORBA クライアント・アプリケーションにトランザクションに関与するオブジェクトを追加します。Transactions サンプル・アプリケーションは、インプリメンテーション・コンフィギュレーション・ファイル (ICF) を使用して CORBA オブジェクトのトランザクション方針を定義する方法を例示します。Transactions サンプル・アプリケーションのビルドと実行については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo CORBA トランザクション』を参照してください。
Wrapper	既存の BEA Tuxedo ATMI アプリケーションを CORBA オブジェクトとしてラッピングする方法を例示します。
Production	サーバ・アプリケーションの複製、状態を持たないオブジェクトの作成、およびファクトリ・ベースのルーティングのインプリメントを例示します。
Secure Simpapp	証明書認証をサポートするための Simpapp サンプル・アプリケーションに対する必要な開発および管理の変更をインプリメントします。Secure Simpapp サンプル・アプリケーションのビルドと実行については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo CORBA アプリケーションのセキュリティ機能』を参照してください。

表 3-7 BEA Tuxedo CORBA サンプル・アプリケーション (続き)

BEA Tuxedo CORBA サンプル・アプリケーション	説明
Introductory Events	CORBA 共同クライアント / サーバ・アプリケーションとコールバック・オブジェクトを使用して BEA Tuxedo CORBA アプリケーションでイベントをインプリメントする方法を例示します。C++ バージョンでは BEA Simple Events API を使用し、Java バージョンでは CosNotification API を使用します。Introductory Events サンプル・アプリケーションのビルドと実行については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo CORBA ノーティフィケーション・サービス』を参照してください。
Advanced Events	一時的なサブスクリプションと永続的なサブスクリプションおよびデータのフィルタ処理を利用したイベントのより複雑なインプリメンテーションを BEA Tuxedo CORBA アプリケーションで提供します。C++ バージョンでは Advanced CosNotification API を使用し、Java バージョンでは Advanced Simple Java API を使用します。Advanced Events サンプル・アプリケーションのビルドと実行については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo CORBA ノーティフィケーション・サービス』を参照してください。

4　セキュリティの使い方

ここでは、次の内容について説明します。

- セキュリティ・サービスの概要
- セキュリティのしくみ
- Security サンプル・アプリケーション
- 開発手順

注記　この章では、認証の使い方を説明します。CORBA セキュリティ環境で利用可能なセキュリティ機能の説明、およびそれらの機能をインプリメントする方法については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo CORBA アプリケーションのセキュリティ機能』を参照してください。

セキュリティ・サービスの概要

BEA Tuxedo 製品の CORBA 環境では、CORBA サービスのセキュリティ・サービスに基づくセキュリティ・モデルが提供されます。BEA Tuxedo CORBA のセキュリティ・モデルでは、CORBA サービスのセキュリティ・サービスの認証部分がインプリメントされます。

CORBA 環境では、ドメイン単位でセキュリティ情報を定義します。ドメインのセキュリティ・レベルは、コンフィギュレーション・ファイルで定義します。クライアント・アプリケーションでは、SecurityCurrent オブジェクトを使用して BEA Tuxedo ドメインにログオンするための認証情報を提示します。

利用できる認証レベルは以下のとおりです。

- TOBJ_NOAUTH

  認証はまったく必要ありません。ただし、クライアント・アプリケーションは認証を受け、ユーザ名とクライアント・アプリケーション名を指定することができます。パスワードは不要です。

- TOBJ_SYSAUTH

  クライアント・アプリケーションは BEA Tuxedo ドメインの認証を受け、ユーザ名、クライアント・アプリケーション名、およびアプリケーション・パスワードを指定する必要があります。

- TOBJ_APPAUTH

  TOBJ_SYSAUTH 情報のほかに、クライアント・アプリケーションはアプリケーション固有の情報を指定する必要があります。デフォルトの BEA Tuxedo CORBA 認証サービスがアプリケーション・コンフィギュレーションで使用される場合、クライアント・アプリケーションはユーザ・パスワードを指定する必要があります。それ以外の場合、クライアント・アプリケーションはそのアプリケーションのカスタム認証サービスによって解釈される認証データを指定します。

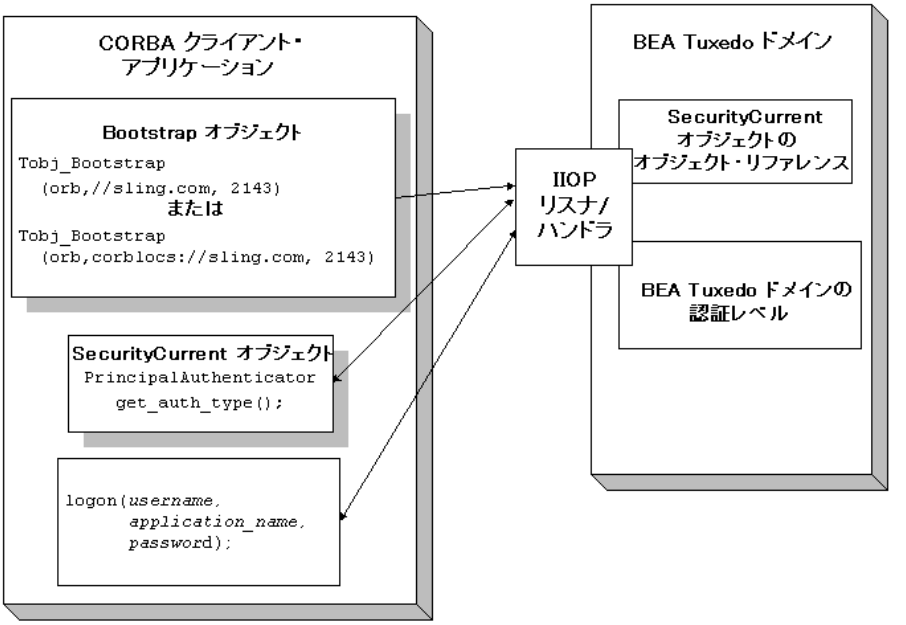
注記　クライアント・アプリケーションが認証を受けず、セキュリティ・レベルが TOBJ_NOAUTH の場合、BEA Tuxedo ドメインの IIOP リスナ / ハンドラはその IIOP リスナ / ハンドラに送信されるユーザ名とクライアント・アプリケーション名で CORBA クライアント・アプリケーションを登録します。

BEA Tuxedo CORBA セキュリティ環境では、SecurityCurrent オブジェクトのプロパティとして PrincipalAuthenticator と Credentials だけがサポートされます。SecurityLevel1::Current インターフェイスと SecurityLevel2::Current インターフェイスの詳細については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo CORBA プログラミング・リファレンス』を参照してください。

セキュリティのしくみ

図 4-1 は、BEA Tuxedo ドメインの CORBA セキュリティのしくみを示しています。

図 4-1 BEA Tuxedo ドメインの CORBA セキュリティのしくみ



CORBA セキュリティは、次のステップで実現されます。

1. クライアント・アプリケーションが、Bootstrap オブジェクトを使用して BEA Tuxedo ドメインの SecurityCurrent オブジェクトのオブジェクト・リファレンスを取得します。
2. クライアント・アプリケーションが、PrincipalAuthenticator を取得します。
3. クライアント・アプリケーションが、`Tobj::PrincipalAuthenticator::get_auth_type()` メソッドを使用して BEA Tuxedo ドメインの認証レベルを取得します。
4. 適切な認証レベルがクライアント・アプリケーションに返されます。

5. クライアント・アプリケーションが `Tobj::PrincipalAuthenticator::logon()` メソッドを使用し、適切な認証情報で BEA Tuxedo ドメインにログオンします。

注記　BEA Tuxedo CORBA では、CORBA インターオペラブル・ネーミング・サービス (INS) を利用してもセキュリティ・サービスの初期オブジェクト・リファレンスを取得できます。INS ブートストラップ処理メカニズムについては、『BEA Tuxedo CORBA プログラミング・リファレンス』を参照してください。

Security サンプル・アプリケーション

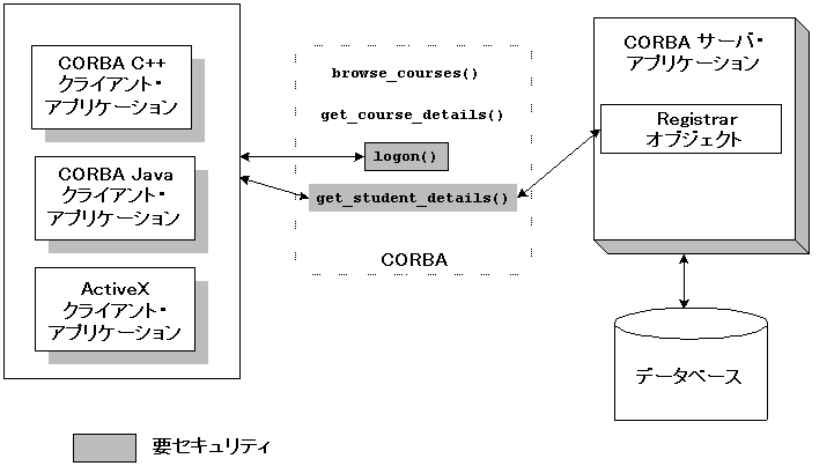
Security サンプル・アプリケーションは、パスワード認証を使用する方法を例示します。Security サンプル・アプリケーションでは、そのアプリケーションを使用する学生ごとに ID とパスワードが必要です。Security サンプル・アプリケーションは次のように機能します。

- クライアント・アプリケーションには `logon()` オペレーションがあります。このオペレーションでは、ドメインにログオンする過程で取得される PrincipalAuthenticator オブジェクトのオペレーションが呼び出されます。
- サーバ・アプリケーションでは、学生についての情報を返す `Registrar` オブジェクトの `get_student_details()` オペレーションがインプリメントされます。ユーザが認証された後、ログオンが完了し、`get_student_details()` オペレーションがクライアントのログオン処理で必要な学生情報を取得するためにデータベースの学生情報にアクセスします。
- Security サンプル・アプリケーションのデータベースには、コースと学生の情報が格納されます。

注記　証明書の認証は、Secure Simpapp サンプル・アプリケーションで説明されます。

図 4-2 は、Security サンプル・アプリケーションを示しています。

図 4-2 Security サンプル・アプリケーション



Security サンプル・アプリケーションのソース・ファイルは、BEA Tuxedo ソフトウェアの `\samples\corba\university` ディレクトリにあります。Security サンプル・アプリケーションのビルドと実行については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo CORBA アプリケーションのセキュリティ機能』を参照してください。

開発手順

表 4-1 は、認証セキュリティを採用する BEA Tuxedo CORBA アプリケーションの開発手順を示しています。

表 4-1 セキュリティを備える BEA Tuxedo CORBA アプリケーションの開発手順

手順	説明
1	コンフィギュレーション・ファイルでセキュリティ・レベルを定義します。
2	CORBA クライアント・アプリケーションを記述します。

ステップ 1: コンフィギュレーション・ファイルでのセキュリティ・レベルの定義

BEA Tuxedo ドメインのセキュリティ・レベルは、コンフィギュレーション・ファイルの RESOURCES セクションの SECURITY パラメータを目的のセキュリティ・レベルに設定することで定義します。表 4-2 は、SECURITY パラメータのオプションを示しています。

表 4-2 SECURITY パラメータのオプション

オプション	定義
NONE	ドメインでセキュリティはインプリメントされません。このオプションはデフォルトです。このオプションは、TOBJ_NOAUTH レベルの認証に対応します。
APP_PW	クライアント・アプリケーションでは、初期化時にアプリケーション・パスワードを提示することが必要です。tmloadcf コマンドがアプリケーション・パスワードを要求します。このオプションは、TOBJ_SYSAUTH レベルの認証に対応します。

表 4-2 SECURITY パラメータのオプション

オプション	定義
USER_AUTH	アプリケーション・パスワードが必要です。クライアント・アプリケーションの初期化時にユーザ単位の認証が実行されます。このオプションは、TOBJ_APPAUTH レベルの認証に対応します。

Security サンプル・アプリケーションでは、SECURITY パラメータが APP_PW (アプリケーション・レベルのセキュリティ) に設定されます。BEA Tuxedo CORBA アプリケーションにセキュリティを追加する方法については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo CORBA アプリケーションのセキュリティ機能』を参照してください。

ステップ 2: CORBA クライアント・アプリケーションの記述

次の処理を実行するクライアント・アプリケーション・コードを記述します。

1. Bootstrap オブジェクトを使用して、特定の BEA Tuxedo ドメインの SecurityCurrent オブジェクトのリファレンスを取得します。
2. SecurityCurrent オブジェクトから PrincipalAuthenticator オブジェクトを取得します。
3. PrincipalAuthenticator オブジェクトの get_auth_type() オペレーションを使用して、BEA Tuxedo ドメインが想定するタイプの認証を返します。

リスト 4-1 と リスト 4-2 は Security サンプル・アプリケーションの CORBA C++ クライアント・アプリケーションと CORBA Java クライアント・アプリケーションの一部分であり、セキュリティの開発手順を示しています。

リスト 4-1　CORBA C++ クライアント・アプリケーションのセキュリティの例

```
CORBA::Object_var var_security_current_oref =
       bootstrap.resolve_initial_references("SecurityCurrent");
SecurityLevel2::Current_var var_security_current_ref =
       SecurityLevel2::Current::_narrow(var_security_current_oref.in());


//PrincipalAuthenticator を取得
SecurityLevel2::PrincipalAuthenticator_var var_principal_authenticator_oref =
       var_security_current_ref->principal_authenticator();
//PrincipalAuthenticator をナロー変換
Tobj::PrincipalAuthenticator_var var_bea_principal_authenticator =
       Tobj::PrincipalAuthenticator::_narrow (
                                     var_principal_authenticator_oref.in());

// セキュリティ・レベルを指定
Tobj::AuthType auth_type = var_bea_principal_authenticator->get_auth_type();
Security::AuthenticationStatus status = var_bea_principalauthenticator->logon(
                                              user_name,
                                              client_name,
                                              system_password,
                                              user_password,
                                              0);
```

リスト 4-2　CORBA Java クライアント・アプリケーションのセキュリティの例

```
org.omg.CORBA.Object SecurityCurrentObj =
       gBootstrapObjRef.resolve_initial_references("SecurityCurrent");
org.omg.SecurityLevel2.Current secCur =
       org.omg.SecurityLevel2.CurrentHelper.narrow(SecurityCurrentObj);

//PrincipalAuthenticator を取得
org.omg.SecurityLevel2.PrincipalAuthenticator authlevel2 =
       secCur.principal_authenticator();
//PrincipalAuthenticator をナロー変換
com.beasys.Tobj.PrincipalAuthenticatorObjRef gPrinAuthObjRef =
       (com.beasys.Tobj.PrincipalAuthenticator)
       org.omg.SecurityLevel2.PrincipalAuthenticatorHelper.narrow(authlevel2);

// セキュリティ・レベルを指定
com.beasys.Tobj.Authtype authType = gPrinAuthObjRef.get_auth_type();
```

```
org.omg.Security.AuthenticationStatus status = gPrinAuthObjRef.logon
       (gUserName, ClientName, gSystemPassword, gUserPassword,0);
```

5　トランザクションの使い方

ここでは、次の内容について説明します。

- トランザクション・サービスの概要
- トランザクションのしくみ
- Transactions サンプル・アプリケーション
- 開発手順

注記　ここでは、CORBA サービスのオブジェクト・トランザクション・サービスに対する C++ インターフェイスの使い方について説明します。BEA Tuxedo 製品の CORBA 環境で利用可能なトランザクション機能の説明、およびそれらのトランザクション機能をインプリメントする手順については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo CORBA トランザクション』を参照してください。

トランザクション・サービスの概要

BEA Tuxedo 製品の最も基本的な機能の 1 つにトランザクション管理があります。トランザクションは、データベース・トランザクションが誤りなく完了し、高性能トランザクションのすべての ACID 特性 (原子性、一貫性、独立性、および持続性) が確保されるようにする手段です。BEA Tuxedo システムでは、データベースの更新が (さまざまなリソース・マネージャにまたがる場合でも) 誤りなく実行されるようにする完全なインフラストラクチャによってトランザクションの整合性が保護されます。

BEA Tuxedo システムでは、次の機能が利用されます。

- CORBA サービスのオブジェクト・トランザクション・サービス (OTS)

BEA Tuxedo 製品の CORBA 環境では、オブジェクト・トランザクション・サービスに対する C++ インターフェイスが提供されます。OTS には、TransactionCurrent 環境オブジェクトを通じてアクセスします。TransactionCurrent 環境オブジェクトの使い方については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo CORBA クライアント・アプリケーションの開発方法』を参照してください。

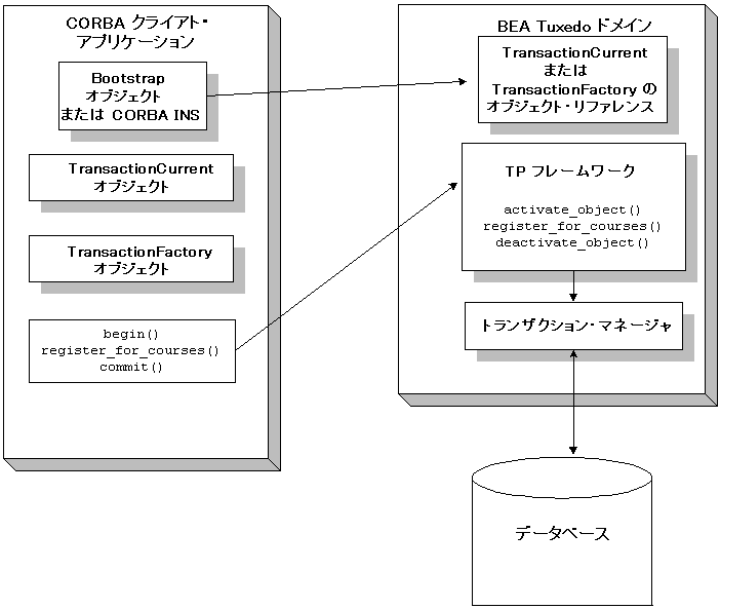
OTS では、ビジネス・トランザクションの次のサポートが提供されます。

- クライアント・アプリケーションによってトランザクションが開始されたときにグローバル・トランザクション識別子が作成されます。
- トランザクションに関与し、トランザクションがコミットされるときにコーディネートする必要のあるオブジェクトが TP フレームワークと共同でトラッキングされます。
- リソース・マネージャ (データベースが多い) がトランザクションの代わりにアクセスされたときにそのことがリソース・マネージャに通知されます。リソース・マネージャはトランザクションが終了するまでアクセスされたレコードをロックします。
- トランザクションが完了したときに 2 フェーズ・コミットが調整されます。その調整によって、トランザクションのすべてのパーティシパントで更新が同時にコミットされます。Open Group XA プロトコルを使用して、更新されるすべてのデータベースでコミットがコーディネートされます。この規格は、ほぼすべてのリレーショナル・データベースでサポートされています。
- トランザクションが停止されるときにロールバック手続きが実行されます。
- 障害が発生したときに回復手続きが実行されます。クラッシュ発生時にマシンでどのトランザクションがアクティブだったのかが判別され、続いて、トランザクションをロールバックすべきなのか、またはコミットすべきなのかが判断されます。

トランザクションのしくみ

図 5-1 は、BEA Tuxedo CORBA アプリケーションでトランザクションがどのように機能するのかを示しています。

図 5-1　BEA Tuxedo CORBA アプリケーションでのトランザクションのしくみ



基本的なトランザクションは次のように行われます。

1. クライアント・アプリケーションが、Bootstrap オブジェクトを使用して BEA Tuxedo ドメインの TransactionCurrent オブジェクトのオブジェクト・リファレンスを取得します。
2. クライアント・アプリケーションが、`Tobj::TransactionCurrent::begin()` メソッドを使用してトランザクションを開始し、TP フレームワークを通じて CORBA インターフェイスに要求を発行します。CORBA インターフェイスのすべてのオペレーションは、トランザクションのスコープ内で実行されます。

- それらのいずれかのオペレーションの呼び出しで (明示的にまたは通信障害の結果として) 例外が生成された場合は、その例外をキャッシュし、トランザクションをロールバックできます。
- 例外が発生しない場合、クライアント・アプリケーションでは `Tobj::TransactionCurrent::commit()` メソッドを使用して現在のトランザクションをコミットします。このメソッドでは、トランザクションが終了し、オペレーションの処理が開始されます。トランザクションは、すべてのパーティシパントがコミットに合意する場合だけコミットされます。

3. `Tobj::TransactionCurrent::commit()` メソッドにより、トランザクションを完了するために TP フレームワークからトランザクション・マネージャが呼び出されます。
4. トランザクション・マネージャがデータベースを更新します。

注記　BEA Tuxedo CORBA では、CORBA インターオペラブル・ネーミング・サービス (INS) を利用してもセキュリティ・サービスの初期オブジェクト・リファレンスを取得できます。INS ブートストラップ処理メカニズムについては、『BEA Tuxedo CORBA プログラミング・リファレンス』を参照してください。

Transactions サンプル・アプリケーション

Transactions サンプル・アプリケーションでは、コースの登録のオペレーションがトランザクションのスコープ内で実行されます。Transactions サンプル・アプリケーションで使用されるトランザクション・モデルは、会話型モデルと、単一クライアントの呼び出しでデータベースの複数の個別オペレーションが呼び出されるモデルを組み合わせたものです。

Transactions サンプル・アプリケーションは次のように機能します。

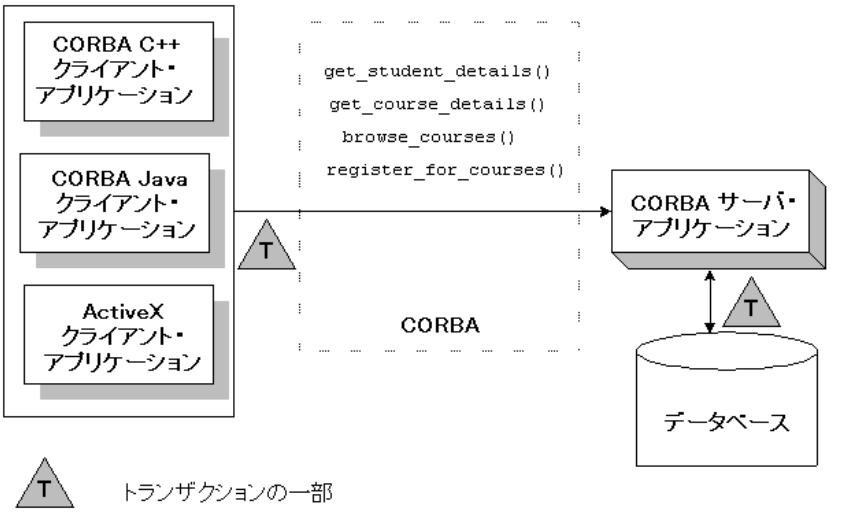
1. 学生が、登録したいコースのリストを提示します。
2. リストの各コースについて、CORBA サーバ・アプリケーションが次のことをチェックします。
   - コースがデータベースにあるかどうか。
   - 学生が既に登録されているかどうか。

- 学生が履修できる単位の最大数を超えているかどうか。

3. 次のいずれかが行われます。
   - コースがすべての基準を満たす場合は、CORBA サーバ・アプリケーションによってそのコースで学生が登録されます。
   - コースがデータベースにない場合、またはそのコースで学生が既に登録されている場合は、CORBA サーバ・アプリケーションによって、学生を登録できないコースのリストにそのコースが追加されます。すべての登録要求が処理された後、登録に失敗したコースのリストが CORBA サーバ・アプリケーションから返されます。CORBA クライアント・アプリケーションでは、トランザクションをコミットするか (登録要求が成功したコースで学生が登録される)、またはトランザクションをロールバックするか (どのコースにも学生は登録されない) を選択できます。
   - 学生が履修できる単位の最大数を超えている場合は、`TooManyCredits` ユーザ例外が CORBA サーバ・アプリケーションから CORBA クライアント・アプリケーションに返されます。CORBA クライアント・アプリケーションでは、要求が拒否されたことを説明する短いメッセージが表示されます。その後、CORBA クライアント・アプリケーションによってトランザクションがロールバックされます。

図 5-2 は、Transactions サンプル・アプリケーションのしくみを示しています。

図 5-2 Transactions サンプル・アプリケーション



Transactions サンプル・アプリケーションは、トランザクションをロールバックする次の 2 とおりの方法を示します。

- 致命的ではない場合。コースがデータベースに存在しないか、学生が既に登録されているためにコースの登録が失敗する場合は、それらのコースの数が CORBA サーバ・アプリケーションから CORBA クライアント・アプリケーションに返されます。トランザクションをロールバックするかどうかの決断は、CORBA クライアント・アプリケーションのユーザが行います。
- 致命的な場合。登録できる単位の最大数を超えているためにコースの登録が失敗する場合は、CORBA サーバ・アプリケーションによって CORBA 例外が生成され、その例外が CORBA クライアント・アプリケーションに返されます。この場合も、トランザクションをロールバックするかどうかは、CORBA クライアント・アプリケーションに依存します。

開発手順

ここでは、トランザクションが含まれる BEA Tuxedo CORBA アプリケーションの開発手順を説明します。表 5-1 は、開発手順を示しています。

表 5-1 トランザクションの含まれる BEA Tuxedo CORBA アプリケーションの開発手順

手順	説明
1	トランザクションに関与する CORBA インターフェイスの OMG IDL コードを記述します。
2	インプリメンテーション・コンフィギュレーション・ファイル (ICF) で CORBA インターフェイスのトランザクション方針を定義します。
3	CORBA クライアント・アプリケーションを記述します。
4	CORBA サーバ・アプリケーションを記述します。
5	コンフィギュレーション・ファイルを作成します。

Transactions サンプル・アプリケーションは、上記の開発手順を例示します。Transactions サンプル・アプリケーションのソース・ファイルは、BEA Tuxedo ソフトウェアの \samples\corba\university ディレクトリにあります。Transactions サンプル・アプリケーションのビルドと実行については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo CORBA University サンプル・アプリケーション』を参照してください。

ステップ 1: OMG IDL コードの記述

ほかの CORBA インターフェイスの場合と同じように、トランザクションに関与するインターフェイスを Object Management Group (OMG) インターフェイス定義言語 (IDL) で指定する必要があります。また、そのインターフェイスを使用して発生する可能性のあるすべてのユーザ例外を指定することも必要です。

Transactions サンプル・アプリケーションでは、Registrar インターフェイスと register_for_courses() オペレーションを OMG IDL で定義します。register_for_courses() オペレーションには、NotRegisteredList というパラメータがあります。このパラメータは、登録の失敗したコースのリストを CORBA クライアント・アプリケーションに返します。NotRegisteredList の値が空の場合、CORBA クライアント・アプリケーションはトランザクションをコミットします。TooManyCredits ユーザ例外を定義することも必要です。

リスト 5-1 は、Transactions サンプル・アプリケーションの OMG IDL コードを示しています。

リスト 5-1 Transactions サンプル・アプリケーションの OMG IDL コード

```
#pragma prefix "beasys.com"
module UniversityT

{
        typedef unsigned long CourseNumber;
        typedef sequence<CourseNumber> CourseNumberList;

        struct CourseSynopsis
        {
                CourseNumber  course_number;
                string        title;
        };
        typedef sequence<CourseSynopsis> CourseSynopsisList;
```

```
        interface CourseSynopsisEnumerator
        {
        // エントリがもうない場合は長さ 0 のリストを返す
        CourseSynopsisList get_next_n(
                in  unsigned long number_to_get, // 0 の場合はすべてが
返される
                out unsigned long number_remaining
        );

        void destroy();
        };
        typedef unsigned short Days;
        const Days MONDAY    =  1;
        const Days TUESDAY   =  2;
        const Days WEDNESDAY =  4;
        const Days THURSDAY  =  8;
        const Days FRIDAY    = 16;

// スケジューリングされたすべての曜日の、正時に始まる
// 同じ時間帯に制限されたクラス

struct ClassSchedule
{
        Days            class_days; // 曜日のビットマスク
        unsigned short start_hour; // 軍用時間による開始時刻
        unsigned short duration;   // 分数
};

struct CourseDetails
{
        CourseNumber   course_number;
        double         cost;
        unsigned short number_of_credits;
        ClassSchedule  class_schedule;
        unsigned short number_of_seats;
        string         title;
        string         professor;
        string         description;
};
        typedef sequence<CourseDetails> CourseDetailsList;
        typedef unsigned long StudentId;

struct StudentDetails
{
        StudentId         student_id;
        string            name;
        CourseDetailsList registered_courses;
};

enum NotRegisteredReason
{
        AlreadyRegistered,
        NoSuchCourse
};

struct NotRegistered
{
```

```
        CourseNumber        course_number;
        NotRegisteredReason not_registered_reason;
};
        typedef sequence<NotRegistered> NotRegisteredList;

exception TooManyCredits
{
        unsigned short maximum_credits;
};

//Registrar インターフェイスは、学生がデータベース
// にアクセスするためのメインのインターフェイス
interface Registrar
{
        CourseSynopsisList
        get_courses_synopsis(
                in string                     search_criteria,
                in unsigned long              number_to_get,
                out unsigned long             number_remaining,
                out CourseSynopsisEnumerator rest);

        CourseDetailsList get_courses_details(in CourseNumberList
         courses);
        StudentDetails get_student_details(in StudentId student);
        NotRegisteredList register_for_courses(
                in StudentId        student,
                in CourseNumberList courses
        ) raises (
                TooManyCredits
        );

};

// RegistrarFactory インターフェイスが Registrar インターフェイスを検索

interface RegistrarFactory
{
        Registrar find_registrar(
        );
};
```

ステップ 2: インターフェイスのトランザクション方針の定義

トランザクション方針は、インターフェイス単位で使用されます。設計時に、BEA Tuxedo CORBA アプリケーション内のどのインターフェイスでトランザクションを処理するかを決定します。次の表は、トランザクション方針のリストです。

トランザクション方針	説明
always	インターフェイスは常にトランザクションの一部である必要があります。インターフェイスがトランザクションの一部ではない場合、トランザクションは TP フレームワークによって自動的に開始されます。
ignore	インターフェイスはトランザクションに関与しません。ただし、トランザクションのスコープ内でこのインターフェイスに要求を行うことはできます。このインターフェイスの UBBCONFIG ファイルで指定される AUTOTRAN パラメータは無視されます。
never	インターフェイスはトランザクションに関与しません。このインターフェイス用に作成されるオブジェクトは、トランザクションに関与できません。BEA Tuxedo システムは、この方針が設定されているインターフェイスがトランザクションに関与した場合に INVALID_TRANSACTION 例外を生成します。
optional	インターフェイスはトランザクションに関与できます。要求がトランザクションに関係する場合、オブジェクトはトランザクションに関与できます。このトランザクション方針はデフォルトです。 **注記** 要求のトランザクション・プロパティは、autotran パラメータを使用しても定義できます。

開発段階では、トランザクション方針を割り当てることで、どのインターフェイスがトランザクションで実行されるのかを定義します。

CORBA サーバ・アプリケーションでは、インプリメンテーション・コンフィギュレーション・ファイル (ICF) でトランザクション方針を指定します。genicf コマンドを実行すると、テンプレートの ICF ファイルが作成されます。

Transactions サンプル・アプリケーションでは、Registrar インターフェイスのトランザクション方針が always に設定されます。

ステップ 3: CORBA クライアント・アプリケーションの記述

CORBA クライアント・アプリケーションでは、次のタスクを実行するコードが必要です。

1. Bootstrap オブジェクトから TransactionCurrent オブジェクトまたは TransactionFactory オブジェクトのリファレンスを取得します。
2. TransactionCurrent オブジェクトの `Tobj::TransactionCurrent::begin()` オペレーションを呼び出してトランザクションを開始します。
3. オブジェクトのオペレーションを呼び出します。Transactions サンプル・アプリケーションでは、CORBA クライアント・アプリケーションが (コースのリストを渡して) `Registrar` オブジェクトの `register_for_courses()` オペレーションを呼び出します。

リスト 5-2 は、トランザクションの開発手順を説明する Transactions サンプル・アプリケーションの CORBA C++ クライアント・アプリケーションの一部分です。

リスト 5-2 CORBA C++ クライアント・アプリケーションのトランザクション・コード

```
CORBA::Object_var var_transaction_current_oref =
     Bootstrap.resolve_initial_references("TransactionCurrent");
CosTransactions::Current_var var_transaction_current_ref=
     CosTransactions::Current::_narrow(var_transaction_current_oref.in());
// トランザクションを開始
var_transaction_current_ref->begin();
try   {
     // トランザクション内でオペレーションを実行
     pointer_Registar_ref->register_for_courses(student_id,
course_number_list);
     // ...
     // オペレーションの実行でエラーがない場合は、トランザクションをコミット
     CORBA::Boolean report_heuristics = CORBA_TRUE;
     var_transaction_current_ref->commit(report_heuristics);
}
catch (...) {
     // オペレーションの実行に問題がある場合は、トランザ
```

```
      // クションをロールバックして、オリジナルの例外を再び
      // スロー。ロールバックが失敗した場合は、例外を無視
      // して、オリジナルの例外を再びスロー
      try {
            var_transaction_current_ref->rollback();
      }
      catch (...) {
            TP::userlog("rollback failed");
      }
      throw;
}
```

ステップ 4: CORBA サーバ・アプリケーションの記述

CORBA サーバ・アプリケーションでトランザクションを使用する場合は、インターフェイスのオペレーションをインプリメントするメソッドを記述する必要があります。Transactions サンプル・アプリケーションでは、`register_for_courses()` オペレーションのメソッド・インプリメンテーションを記述します。

BEA Tuxedo CORBA アプリケーションでデータベースを使用する場合は、XA リソース・マネージャを開閉するコードが CORBA サーバ・アプリケーションで必要となります。それらのオペレーションは、Server オブジェクトの `Server::initialize()` オペレーションと `Server::release()` オペレーションに含められます。

リスト 5-3 は、XA リソース・マネージャを開閉する Transactions サンプル・アプリケーションの Server オブジェクトのコードの一部分です。

注記　トランザクションをインプリメントする C++ サーバ・アプリケーションの完全な例については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo CORBA トランザクション』の Transactions サンプル・アプリケーションを参照してください。

リスト 5-3 Transactions サンプル・アプリケーションの C++ Server オブジェクト

```
CORBA::Boolean Server::initialize(int argc, char* argv[])
{
       TRACE_METHOD("Server::initialize");
       try {
              open_database();
              begin_transactional();
              register_fact();
              return CORBA_TRUE;
       }
       catch (CORBA::Exception& e) {
              LOG("CORBA exception : " <<e);
       }
       catch (SamplesDBException& e) {
              LOG("Can't connect to database");
       }
       catch (...) {
              LOG("Unexpected exception");
       }
       cleanup();
       return CORBA_FALSE;
}

void Server::release()
{
       TRACE_METHOD("Server::release");
       cleanup();
}

static void cleanup()
{
       unregister_factory();
       end_transactional();
       close_database();
}

// トランザクション・リソース・マネージャを管理するユーティリティ
CORBA::Boolean s_became_transactional = CORBA_FALSE;
static void begin_transactional()
{
       TP::open_xa_rm();
       s_became_transactional = CORBA_TRUE;
}
static void end_transactional()
{
       if(!s_became_transactional){
              // クリーンアップは不要
              return;
       }
       try {
              TP::close_xa_rm ();
       }
```

```
        catch (CORBA::Exception& e) {
               LOG("CORBA Exception : " << e);
        }
        catch (...) {
               LOG("unexpected exception");
        }
        s_became_transactional = CORBA_FALSE;
}
```

ステップ 5: コンフィギュレーション・ファイルの作成

トランザクション対応 BEA Tuxedo CORBA アプリケーションのコンフィギュレーション・ファイルには、次の情報を追加する必要があります。

- SERVERS セクションで、CORBA サーバ・アプリケーションおよびデータベースを管理するアプリケーションのトランザクション・グループを指定します。
- GROUPS セクションで、サーバ・グループを定義します。GROUPS セクションの OPENINFO パラメータと CLOSEINFO パラメータで、データベースの XA リソース・マネージャを開閉するための情報を設定します。この情報は、データベースの製品マニュアルから取得します。デフォルトの com.beasys.Tobj.Server.initialize() オペレーションは、リソース・マネージャを自動的に開きます。
- TLOGDEVICE パラメータで、トランザクション・ログ (TLOG) のパス名を設定します。トランザクション・ログの詳細については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo アプリケーション実行時の管理』を参照してください。

リスト 5-4 は、Transactions サンプル・アプリケーションのコンフィギュレーション・ファイルでこの情報が定義されている部分です。

リスト 5-4　Transactions サンプル・アプリケーションのコンフィギュレーション・ファイル

```
*RESOURCES
        IPCKEY      55432
        DOMAINID    university
```

```
        MASTER    SITE1
        MODEL     SHM
        LDBAL     N
        SECURITY  APP_PW

*MACHINES
        BLOTTO
        LMID = SITE1
        APPDIR = C:\TRANSACTION_SAMPLE
        TUXCONFIG=C:\TRANSACTION_SAMPLE\tuxconfig
        TLOGDEVICE=C:\APP_DIR\TLOG
        TLOGNAME=TLOG
        TUXDIR="C:\tuxdir"
        MAXWSCLIENTS=10

*GROUPS
        SYS_GRP
          LMID      = SITE1
          GRPNO     = 1
        ORA_GRP
          LMID      = SITE1
          GRPNO     = 2

        OPENINFO  = "ORACLE_XA:Oracle_XA+SqlNet=ORCL+Acc=P
        /scott/tiger+SesTm=100+LogDir=.+MaxCur=5"
        OPENINFO  = "ORACLE_XA:Oracle_XA+Acc=P/scott/tiger
        +SesTm=100+LogDir=.+MaxCur=5"
        CLOSEINFO = ""
        TMSNAME   = "TMS_ORA"

*SERVERS
        DEFAULT:
        RESTART = Y
        MAXGEN  = 5

        TMSYSEVT
          SRVGRP  = SYS_GRP
          SRVID   = 1

        TMFFNAME
          SRVGRP  = SYS_GRP
          SRVID   = 2
          CLOPT   = "-A -- -N -M"

        TMFFNAME
          SRVGRP  = SYS_GRP
          SRVID   = 3
          CLOPT   = "-A -- -N"

        TMFFNAME
          SRVGRP  = SYS_GRP
          SRVID   = 4
          CLOPT   = "-A -- -F"

        TMIFRSVR
          SRVGRP  = SYS_GRP
          SRVID   = 5
```

```
        UNIVT_SERVER
          SRVGRP  = ORA_GRP
          SRVID   = 1
          RESTART = N

        ISL
          SRVGRP  = SYS_GRP
          SRVID   = 6
          CLOPT   = -A -- -n //MACHINENAME:2500

*SERVICES
```

トランザクション・ログ、およびコンフィギュレーション・ファイルのパラメータの定義については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo アプリケーションの設定』を参照してください。

索引

T

U

い

お

か

く

こ

さ

す

せ

と

に

ひ

ふ

ほ

ま

め

ゆ