Java/CORBA アプリケーション開発環境の実現

Java/CORBA Application Development Environment

溝 上 昌 宏

要 約 インターネットを基盤とする企業ビジネスが活発化する現在，Java や CORBA などの技術を取り入れた本格的なビジネスアプリケーションの開発に期待が集まっている．この分野では，市場の変化を的確にとらえたビジネスのサービスを実現するために，低コストで迅速な開発が求められる．

本稿では，CORBA を実行基盤とし，サーバ開発言語に Java を使用したビジネスアプリケーションの開発環境の実現について述べる．この開発環境では，特定のアプリケーションモデルに基づき，そのアプリケーション構造と制御ロジックをテンプレートに埋め込み，プログラムの自動生成を行っている．

Abstract Currently, the enterprise business over the Internet comes active and the development of the real business application based on technologies such as Java and CORBA has been a focus of expectation. In this field, the low-cost and swift systems development is needed to provide business services that reflect the change of the market exactly.

This paper describes the implementation of the development environment of the business application that adopts CORBA as the infrastructure of the runtime environment and Java as the development language for the server. This development environment is based on a specific application model and implements a program generation, embedding the application structure and the control logic in the templates.

1. はじめに

インターネットを基盤とする企業ビジネスが活発化する現在，Java や CORBA などの技術を取り入れた本格的なビジネスアプリケーションの開発に期待が集まっている．この分野では，市場の変化を的確にとらえたビジネスのサービスを実現するために，低コストで迅速な開発が求められる．

こうした要求から，CORBA を実行基盤とし，サーバ開発言語に Java を使用したビジネスアプリケーションの開発環境を整備してきた．この開発環境は特定のアプリケーションモデルを想定しており，そのモデル上に開発環境を実現する．特にモデルに含まれるアプリケーション構造と制御ロジックをテンプレート化し，プログラムの自動生成を行っている．

本稿では，まず，想定するインターネット型のアプリケーションモデルについて述べる．次に，アプリケーションの開発工程とツールの関わりについて述べる．最後に，テンプレートによるプログラム自動生成の実現について述べる．

2. アプリケーションモデル

インターネット型のビジネスアプリケーションは，インターネットを媒体として利用者に多様なビジネスのサービスを提供する．アプリケーションの実現形態は多様で

あるが，ここでは，CORBA と Java を基盤とした特定のアプリケーションモデルを想定する．

2.1 実 行 環 境

図 1 にアプリケーションモデルの実行環境を示す．網掛けになっているところは，アプリケーション開発者が開発する部分を示している．

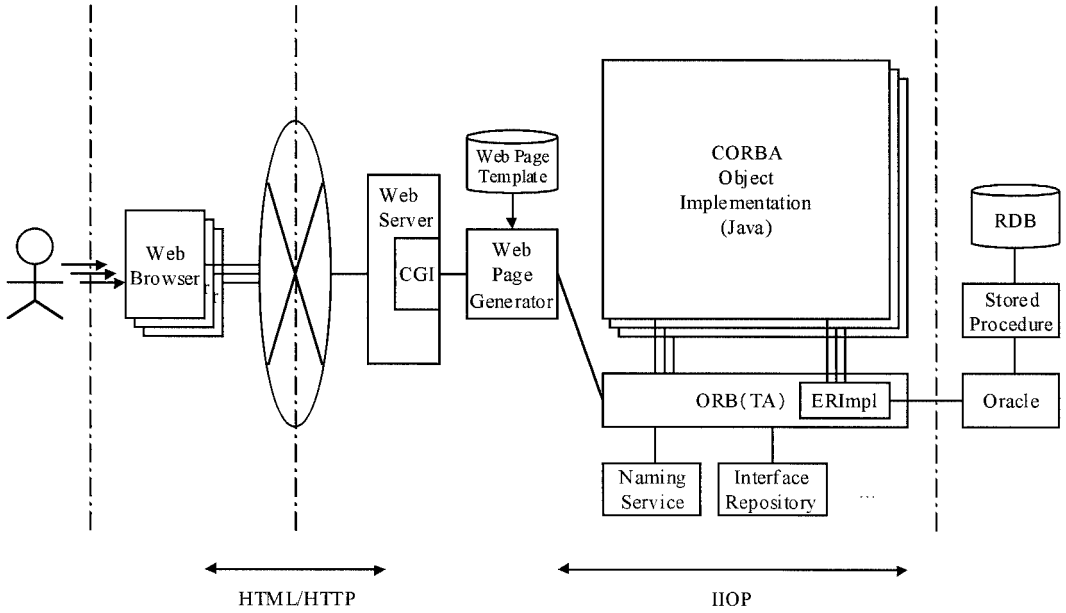

図 1 実行環境

利用者はサービスの要求を Web ブラウザ上のページから行い，結果も Web ブラウザ上に表示される．

Web Page Generator は Web ページテンプレートを入力として HTML を動的に生成する機構を持ち，Web サーバ上の CGI プログラムから起動される．Web ページテンプレートには CORBA オペレーションの呼出し命令を含むプレゼンテーションロジックを記述できる．HTTP 経由で伝送されてきた利用者からのサービス要求は，Web Page Generator による CORBA オブジェクト呼び出しとなる．

データストアは Oracle 上の RDB であり，ストアドプロシジャでアクセスする．CORBA オブジェクトからのデータアクセスは SYSTEM *v*[nju:] の TA に埋め込まれた ERImpl（Embedded Resource Implementation）経由で行われ，TA が制御するトランザクションの配下でストアドプロシジャを実行する．

オペレーションの処理結果は，Web Page Generator が Web ページテンプレートに基づいて Web ページを生成し，再び HTTP 経由で Web ブラウザ上に表示される．

2.2 アプリケーション構造

アプリケーションの構造に着目すると図 2 のようになる．網掛けが開発対象であり，それ以外の部分は実行環境が提供される．

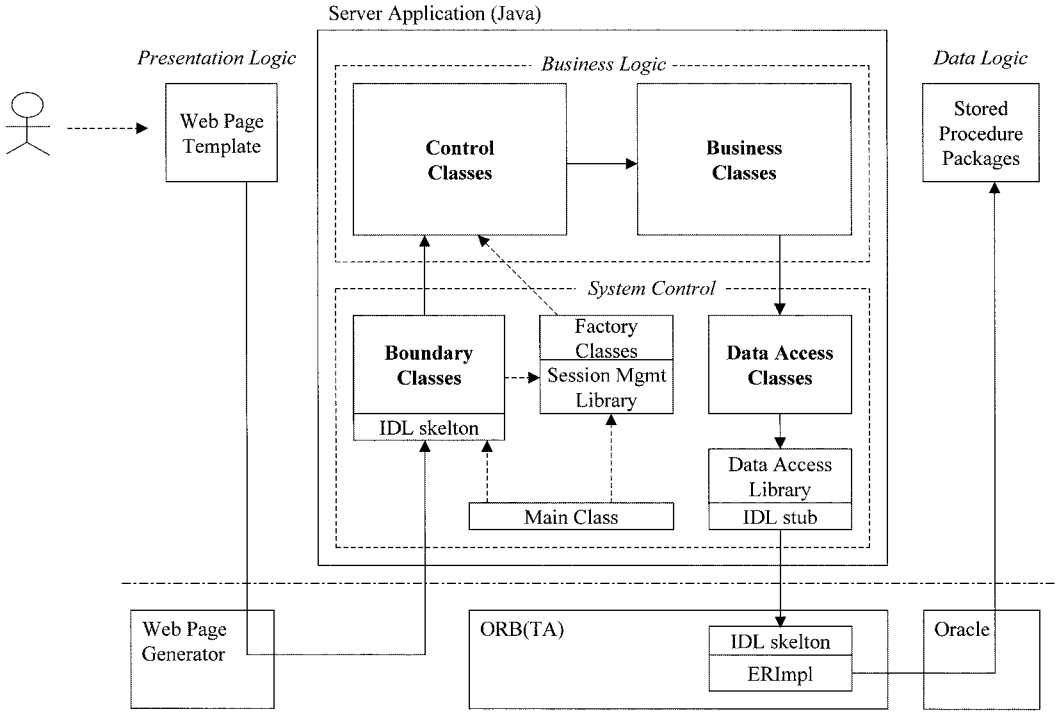

図 2　アプリケーション構造

サーバアプリケーションは Java で実装され，ORB 上のオブジェクトインプリメンテーションとなる．サーバアプリケーションを構成する Java のクラスは，次の四つのステレオタイプに分類される．

- Boundary
  IDL インタフェースを提供するオブジェクトのクラス．このオブジェクトはオペレーションを受け付けるとセッションに対応した Control Object に処理を委譲する．セッションと Control Object の管理にはセッション管理部品（Session Management Library）を使用している．
- Control
  オペレーションを実装するオブジェクトのクラス．このオブジェクトはセッション単位に生成され，セッションを生存期間とする．Business Object または他の Control Object を呼び出す．
- Business
  オペレーションに共通なサービスを提供するオブジェクトのクラス．このオブジェクトは Data Access Object または他の Business Object を呼び出す．
- Data Access
  データモデル(リレーショナルモデル)をオブジェクトとして隠蔽するクラス．このオブジェクトはデータアクセス部品（Data Access Library）を使用してストアドプロシジャを呼び出す．

これらのクラスのうち，アプリケーション固有のビジネスロジックを記述するのは

Control および Business クラスである．それ以外のクラスはアーキテクチャ固有のシステム制御を実現する．

プレゼンテーションロジックは Web Page Generator への入力となる Web ページテンプレートに実装する．データロジックはストアドプロシジャに実装する．

2.3 セッション管理のしくみ

Web アプリケーションでは，物理的なセッションは Web ブラウザ上の画面が切り替わるたびに切断される．このため，利用者が Web ブラウザからある業務サービスを要求してから終了するまでの間，情報を維持するために，論理的なセッションを管理するしくみが必要である．

セッション管理のしくみを図 3 に示す．

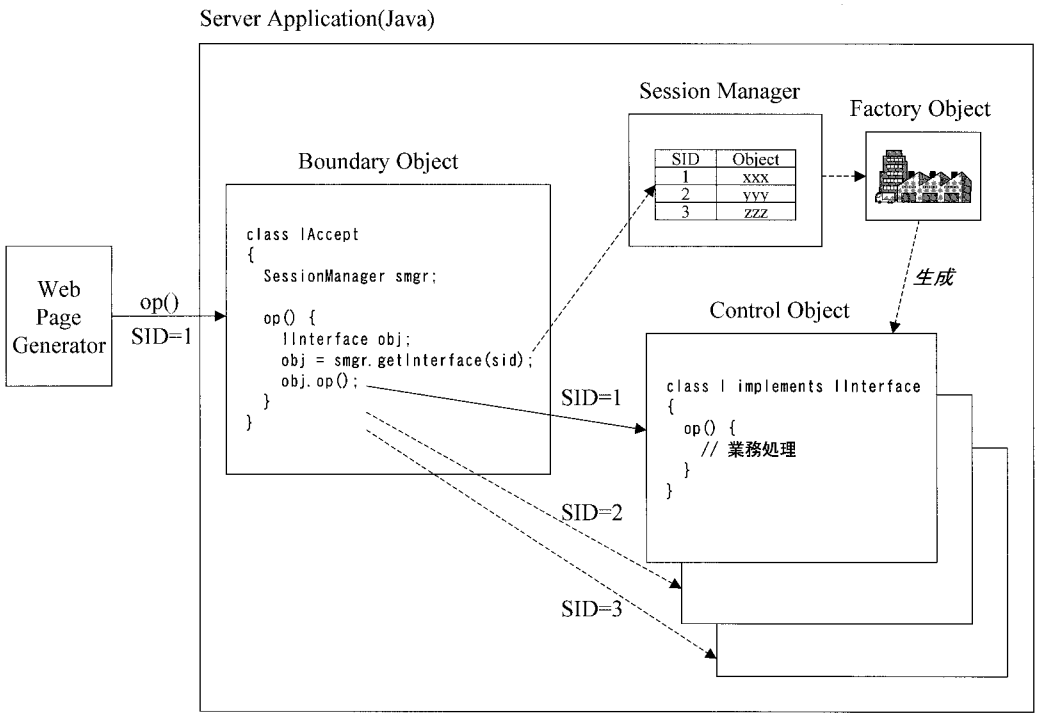

図 3 セッション管理のしくみ

Web ブラウザが Web サーバに接続すると，セッションにユニークなセッション ID が発行され，クライアントに Cookie として保持される．Web Page Generator からオペレーションを呼び出す時は，対応する URL とセッション ID をリクエストコンテキストに付加する．

セッション ID と Control Object の対応は Session Manager が保持している．オペレーション起動要求を受け付けた Boundary Object は，Session Manager を使用してセッション ID から Control Object を特定し，処理をこのオブジェクトに委譲する．これによって，同一セッション内では同一のオブジェクトが処理を行えるようになる．

Session Manager はセッションと Control Object の対応を次のように管理してい

る．

- そのセッションからの要求が初めての場合は，Factory　Object を使って新たな Control　Object を生成し，セッション ID とオブジェクトの対応をテーブルに登録する．そのセッションからの要求が二回目以降の場合は，セッション ID に対応するオブジェクトをテーブルから検索する．
- Web ブラウザの接続終了またはタイムアウトによりセッションは終了する．セッション終了時は，セッション ID とオブジェクトの対応を破棄し，そのオブジェクトを削除する．

3.　開発工程とツール

本章では，アプリケーション開発の流れと，各工程の成果物を明らかにし，最終的な成果物に対してツールがどのように開発を支援するかを述べる．

3.1　概　　観

図 4 に開発工程とツールの使われ方の概観を示す．開発工程は，最初に要求分析工程があり，その後に設計・実装工程が続く．

要求分析工程はオブジェクト指向（ユースケースモデリング）で行う．要求分析によって，システムのインタフェースが決定し，設計・実装工程への入力となる．

設計・実装工程は，インタフェースに対するインプリメンテーションを設計，実装

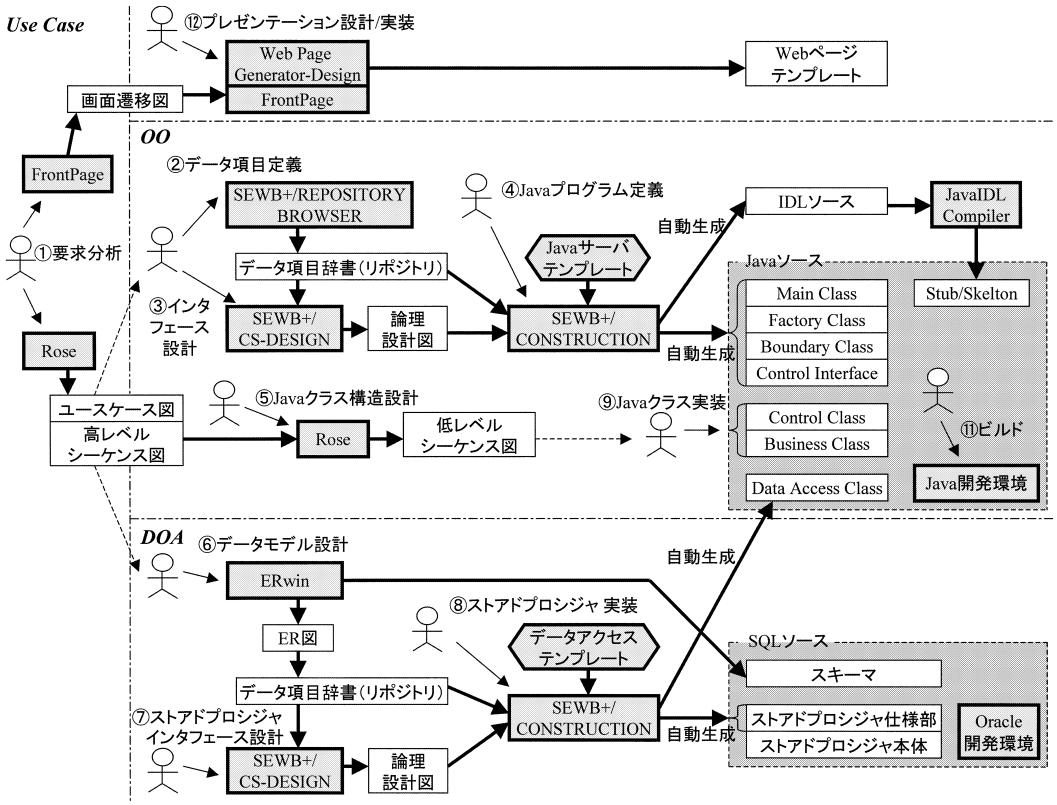

図 4　開発工程

する．プレゼンテーションロジック，ビジネスロジック，データロジックの開発は並行して行うことができる．ビジネスロジックはオブジェクト指向による開発を行う．データロジックはデータ中心アプローチ（DOA）および構造化手法による開発を行う．

網掛けされた太線の四角形はツールをあらわす．ツールを使って分析，設計，実装といった知的作業を行う工程には，それぞれ開発者をあらわすスティックマン・アイコンを置いている．単純にツールで自動生成するだけの工程にはスティックマンを置いていない．使用するツールには，Rational Rose，SEWB＋ファミリ，ERwin，FrontPage，Web Page Generator-Design がある．

白い四角形は各工程の成果物を示している．ツールと成果物はデータの流れをあらわす太い矢印で結ばれており，ツール間の連携を示している．破線の矢印はツール間の連携がないことを示している．

この開発工程の中で特筆すべきは SEWB＋/CONSTRUCTION を用いてプログラムの自動生成を行っている点である．六角形はプログラム生成の雛形であるテンプレートをあらわしている．テンプレートには次の 2 種類がある．

- Java サーバテンプレート
- データアクセステンプレート

テンプレートによるプログラム自動生成については，後の章で詳しく述べることにする．

3.2 要 求 分 析

要求分析工程では，ユースケースモデリングによってシステム化要求を分析し，システムの外部モデルを定める．システムの外側にあるアクタを識別し，アクタから見たシステムの機能をユースケースとして識別する（ユースケース図）．ユースケースはアクタとシステムのインタラクションの系列から構成される（シーケンス図）．また，アクタから見たインタラクションの視覚的表現として画面遷移図を作成する．

モデリングのツールには Rose を使用する．要求分析工程において Rose を使用することによる効果は次のような点である．

- モデリング表記法の標準である UML（Unified Modeling Language）に基づいたモデルが定義できるため，誰もが理解でき，エンドユーザとの要求仕様の確認に効果を発揮する．
- 複数のモデルに分割された情報の整合性を保つことができるので，モデルの変更や拡張が容易で，効率的なモデリングができる．

3.3 設 計 ・ 実 装

設計・実装工程では，要求分析によって抽出されたインタフェースのインプリメンテーションを実装する．

1） インタフェース設計（図 4②，③）

要求分析の結果をもとにインタフェースを設計する．個々のユースケースを一つのインタフェースに対応させ，インタラクションをオペレーションに対応させる．

インタフェース設計のツールには SEWB＋/CS-DESIGN を使用する．SEWB＋

/CS-DESIGN は CORBA インタフェースを視覚的に設計するツールであり，設計したインタフェース定義情報は論理設計図と呼ばれる SEWB＋独自形式のファイルに格納される．オペレーションのパラメタに使用する型情報は SEWB＋リポジトリに別途登録したデータ項目から参照する．

2) Java プログラム定義（Java ソース生成）（図 4④）

SEWB＋/CONSTRUCTION およびテンプレートを使用してプログラムの定義を行い，Java ソースを自動生成する．ここでは，サーバのメイン処理やセッション管理といったシステム制御を実現するクラスのほか，オペレーションを実装する Control クラスの Java インタフェースが生成される．

この工程で行うことは，インタフェース設計の成果物である論理設計図からインタフェースを選択し，テンプレートが要求するいくつかのパラメタを入力することだけであり，コーディングは不要である．

3) Java クラス構造設計（図 4⑤）

要求分析で作成したシーケンス図を詳細化し，システムの内部モデルを定義する．抽出されたクラスは Business クラスとなる．

4) Java クラス実装（図 4⑨）

ビジネスロジックを Control クラスおよび Business クラスとして任意の Java 開発環境を用いて実装する．

5) データモデル設計（図 4⑥）

ERwin/ERX を使用して ER 分析を行い，データモデルを設計する．ERwin/ERX は SEWB＋/REPOSITORY と連携しており，ERwin/ERX で設計したデータ項目を CSV 形式ファイル経由で SEWB＋リポジトリに登録することができる．

6) ストアドプロシジャインタフェース設計（図 4⑦）

データロジックを記述するストアドプロシジャのインタフェースを設計する．ストアドプロシジャインタフェースは，Data Access クラスのインタフェースおよびストアドプロシジャ仕様部の言語非依存表現である．

ツールはインタフェース設計と同様に SEWB＋/CS-DESIGN を使用し，設計情報は論理設計図に格納される．

7) ストアドプロシジャ実装（SQL/Java ソース生成）（図 4⑧）

SEWB＋/CONSTRUCTION およびテンプレートを使用してプログラムを定義し，SQL および Java ソースを自動生成する．ここでは，ストアドプロシジャのソースと Java の Data Access クラスが生成される．

この工程で行うことは Java プログラム定義とほぼ同じであるが，データロジック（ストアドプロシジャ本体）は SEWB＋/CONSTRUCTION の中から UOC（User Own Code）として実装する必要がある．

8) ビルド（図 4⑪）

生成および実装したソースを任意の開発環境でビルドする．

9) プレゼンテーション設計・実装（図 4⑫）

プレゼンテーションロジックを Web Page Generator-Design および

FrontPage を使用して Web ページテンプレートとして実装する．

4. テンプレートによるプログラム自動生成

開発工程の中でも最大の特徴は，SEWB＋/CONSTRUCTION を中心とする SEWB＋のツール群によって，Java のコード生成を行った点である．アプリケーション開発のノウハウをテンプレートに埋め込むことによって，開発者の記述量を大幅に減らすことができる．

4.1 SEWB＋/CONSTRUCTION によるプログラム生成のしくみ

SEWB＋/CONSTRUCTION によるプログラム生成の流れを図 5 に示す．

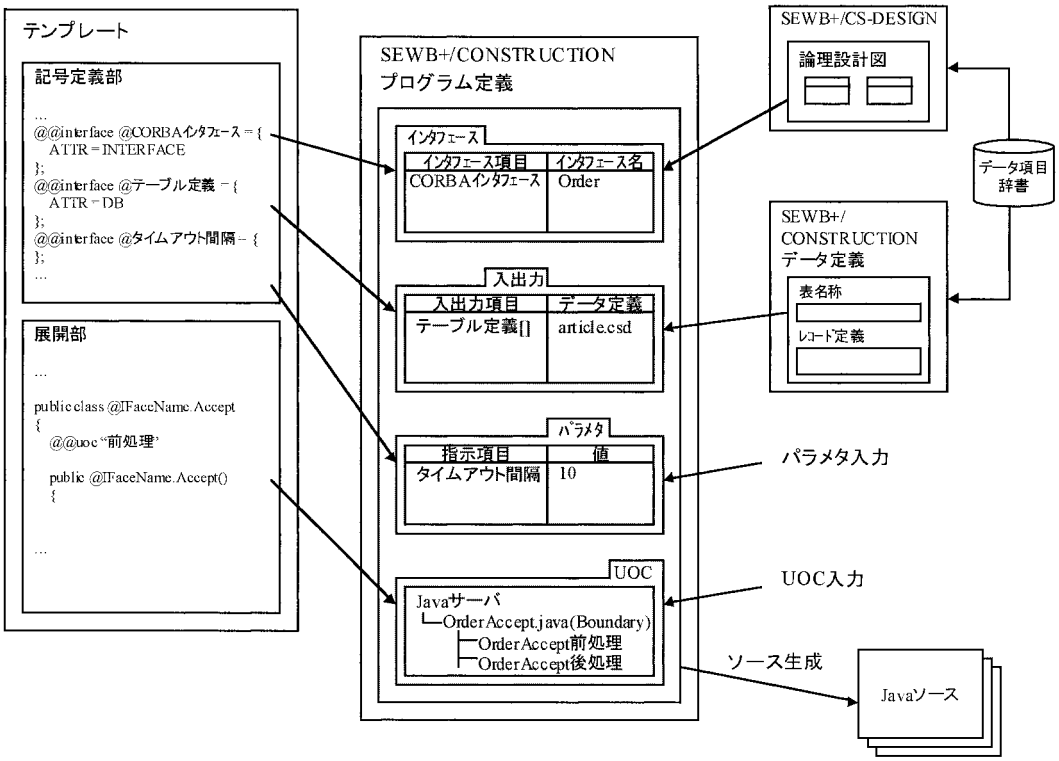

図 5　SEWB＋/CONSTRUCTION によるプログラム生成

テンプレートは記号定義部と展開部から構成される．記号定義部にはユーザに入力させるパラメタを定義できる．展開部にはプログラムの展開処理を記述したプログラム構造を定義する．プログラム展開処理にはプログラム生成時にそのまま出力されるコードのほかに，ユーザに処理を記述させる命令を含めることができる．

SEWB＋/CONSTURCTION にテンプレートを読み込むと，テンプレートに記述されたとおりにパラメタや UOC を入力できる．また，SEWB＋/CS-DESIGN で作成した論理設計図に含まれるインタフェース定義情報やデータ項目辞書の情報もプログラム定義の入力である．こうした一連の作業をプログラム定義と呼んでいる．プログラム定義が終われば，入力情報と展開部に記述されたプログラム展開処理にしたがって，

ソースプログラムを自動生成できる．

4.2 テンプレートの種類

テンプレートには CORBA 基本処理やセッション管理，データアクセスといったアプリケーションに共通な処理を埋め込む．テンプレートは開発者からシステム制御を隠蔽し，ビジネスロジックの開発に専念させる役割を持つ．

テンプレートはプログラム定義時の入力情報の違いから，次の 2 種類を用意している．

- Java サーバテンプレート
- データアクセステンプレート

図 6 は，アプリケーション構造の中でテンプレートが生成する部分を示している．

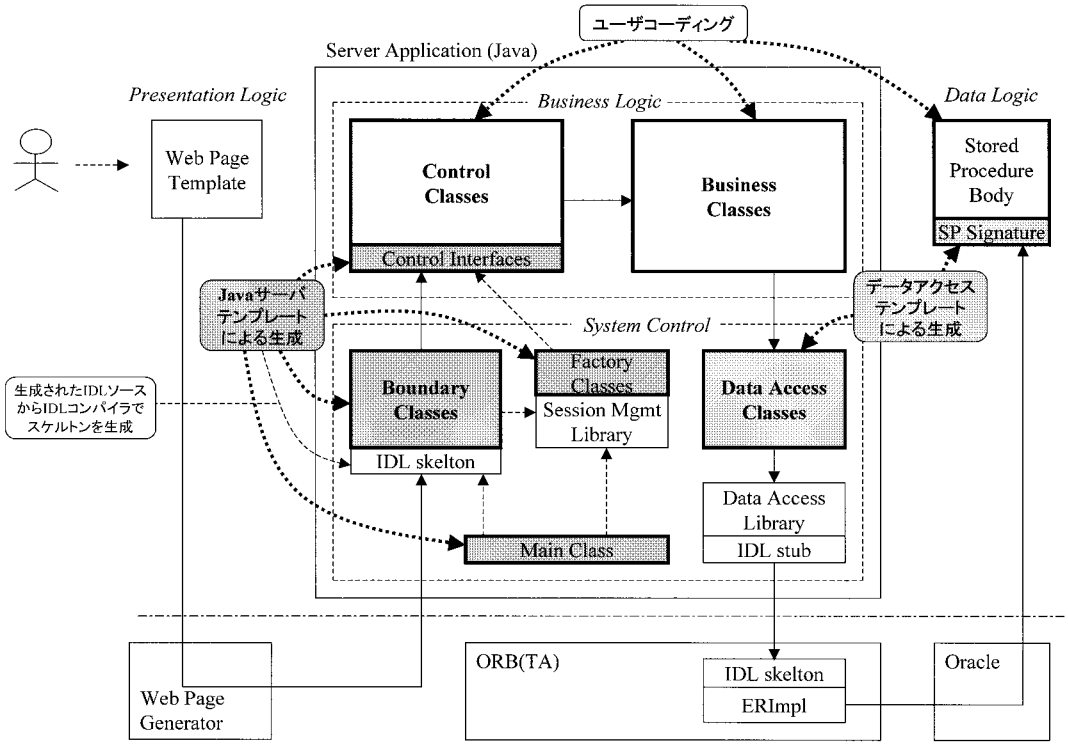

図 6　アプリケーション構造とテンプレート生成部

4.3 Java サーバテンプレート

Java サーバテンプレートは，インタフェース設計の成果物である論理設計図を入力として，次のソースを自動生成する．

- メインクラス（Java）
- Factory クラス（Java）
- Boundary クラス（Java）
- Control インタフェース（Java）
- IDL ソース（さらに，IDL コンパイラでスケルトンを生成）

メインクラスは，ORB と接続するシステム制御を実装する．Java の場合，基本的には次のようなコーディング・パターンが存在する．

1. ORB の初期化
2. 初期オブジェクトリファレンスの取得
3. オブジェクトの生成と ORB との通信の初期化
4. オブジェクトのネーミングサービスへの登録
5. クライアントからの処理要求待ち

また，メインクラスは，Session Manager の初期化などを行う必要があるため，セッション管理機能とも接続している．メインクラスを自動生成することによって，これらシステム制御部をアプリケーションから分離することができる．

Boundary クラス，Factory クラスもセッション管理機能と密接に接続したクラスであり，自動生成する．また，Boundary Object が処理を委譲する先の Control Object のインタフェースも自動生成する．次に，具体例で見てみよう．

プログラム定義の入力として次のようなインタフェースを与えたと仮定する．

```
// IDL
module OrderSystem {
    interface Order {
        makeOrder(in string orderinfo,
                  out string<6> orderid);
    };
};
```

モジュールが OrderSystem，インタフェースが Order，オペレーションが makeOrder である．

このとき，Boundary クラスは次のように生成される．

```
package OrderSystemObject;
public class OrderAccept extends _OrderImplBase {
    static SessionManager sgmr;
    …
    public void makeOrder( String orderinfo,
                           StringHolder orderid,
                           String[] context ) {
        try {
            OrderInterface obj;
            obj = sgmr.getInstance( context, OrderHelper.id() );
            obj.makeOrder();
        }
        …
```

```
    }
}
```

クラス OrderAccept はインタフェース Order を実装した CORBA オブジェクトのクラスになっている．メソッド makeOrder は，セッションに対応する Control Object を Session Manager から取得し，そのオブジェクトにインタフェース OrderInterface によって処理を委譲している．

インタフェース OrderInterface は次のように生成される．

```
package OrderSystemObject;
public interface OrderInterface {
    void makeOrder( String orderinfo,
                    StringHolder orderid,
                    String[] context );
}
```

Control クラスはこのインタフェースを implements して実装する．Control Object は Factory Object によって生成されるので，Factory クラスを自動生成するために，この実装クラス名が必要となる．Java サーバテンプレートではこのクラス名をパラメタとして与えることにしている．

```
public class OrderImpl implements OrderInterface {
    public void makeOrder() {
        // 処理
        // (Business Object または他の Control Object を呼び出す)
    }
}
```

4.4 データアクセステンプレート

データアクセステンプレートは，ストアドプロシジャインタフェース設計の成果物である論理設計図を入力として，次のプログラムを自動生成する．

- Data Access クラス（Java）
- ストアドプロシジャ仕様部（PL/SQL）

Data Access クラスは，Java アプリケーションからのストアドプロシジャ呼び出しを仮想化したクラスである．Data Access クラスを自動生成することによって，アプリケーションからデータアクセスに関するシステム制御を分離している．

ストアドプロシジャはアプリケーション開発者がデータロジックを記述する部分であるが，その仕様部については自動生成する．

ストアドプロシジャインタフェースは SEWB+/CS-DESIGN で設計され，言語非依存形式である Psuedo IDL で表現される．Psuedo IDL から Data Access クラスお

よびストアドプロシジャ仕様部を自動生成するために, IDL から Java および PL/SQL へのマッピングを定義している．このマッピングはテンプレートに記述されている．

表 1 に各言語要素のマッピングを示す．

表 1　データアクセステンプレートにおける言語要素のマッピング

ストアドプロシジャ インタフェース (IDL)	Data Access クラス (Java)	ストアドプロシジャ (PL/SQL)
モジュール	パッケージ	—
インタフェース	クラス	パッケージ
オペレーション	メソッド	プロシジャ

また，データ型についても，使用できる型とパラメタとして使用したときのマッピングをテンプレートで定義している．ここでも，具体例で見てみよう．

プログラム定義の入力として次のようなストアドプロシジャインタフェースを与えたと仮定する．

```
// IDL
module OrderSP {
    typedef string<6> article_tab__ar[1000];
    typedef unsigned long price_tab__ar[1000];
    typedef unsigned long amount_tab__ar[1000];

    interface Detail {
        select_detail_by_orderid(
            in string<6> orderid,
            out article_tab__ar article_tab,
            out price_tab__ar price_tab,
            out amount_tab__ar amount_tab);
    };
};
```

モジュールが OrderSP，インタフェースが Detail，オペレーションが select_detail_by_orderid である．

このとき，Data Access クラスは次のように生成される．

```
package OrderSP;
public class Detail {
    public static void select_detail_by_orderid(
        java.lang.String orderid,
        article_tab__arHolder article_tab,
        price_tab__arHolder price_tab,
        amount_tab__arHolder amount_tab,
```

```
            Nu.Control control) throws NuSPC.NuSPCLibException,
                                       java.lang.Exception
        {
            ...
        }
    }
```

メソッド select_detail_by_orderid の実装としては，データアクセス部品(Data Access Library)を使用して，入力引数をストアドプロシジャ呼び出しのための内部形式に変換し，プロシジャ select_detail_by_orderid を呼び出し，結果を取得するコードが自動生成されるが，ここでは省略している．

ストアドプロシジャ仕様部は次のように生成される．

```
-- PL/SQL
CREATE OR REPLACE PACKAGE Detail AS
    TYPE article_tab__ar IS TABLE OF CHAR(6)
        INDEX BY BINARY INTEGER;
    TYPE price_tab__ar IS TABLE OF NUMBER(9)
        INDEX BY BINARY INTEGER;
    TYPE amount_tab__ar IS TABLE OF NUMBER(9)
        INDEX BY BINARY INTEGER;

    PROCEDURE select_detail_by_orderid(
        orderid_ IN CHAR,
        article_tab_ OUT article_tab__ar,
        price_tab_ OUT price_tab__ar,
        amount_tab_ OUT amount_tab__ar);
END Detail;
```

5. おわりに

Java と CORBA によるアプリケーションモデルとその開発環境の実現について述べてきた．要求分析から設計，実装に至る各工程においてツールがどのような役割を果たすかを述べた．特に，テンプレートによるプログラム自動生成は，アプリケーションモデル固有の制御ロジックをアプリケーション開発者から隠蔽する役割を果たす．開発者はビジネスロジックの実装に専念することによって，開発の生産性を向上させることができる．

今後の課題としては，「ツール連携の強化」，「ビジネスロジックの生成」，「テスト支援の強化」などを挙げることができる．現状では，Rose から設計以降のツールへの連携はない．ビジネスロジックは依然として最初からコーディングする必要がある．実装したロジックのテストを支援するしくみも十分とはいえない．

インターネットビジネスが活発化している現在，Java，CORBA をキーワードにしたコンポーネントベースのアプリケーション開発は最も注目されている分野の一つであり，開発環境に対する期待も大きい．課題は少なくないが，今後更なる整備が求められる．

参考文献 ［1］ I. Jacobson, G. Booch, J. Rumbaugh,“ The Unified Software Development Process ”, Addison Wesley, 1999.
［2］ OMG,“ Unified Modeling Language Specification ”, 1997.
［3］ OMG, “ The Common Object Request Broker： Architecture and Specification ”, 1996.

執筆者紹介 溝　上　昌　宏（Masahiro Mizogami）
1965 年生．1989 年名古屋大学工学部情報工学科卒業．同年日本ユニシス(株)入社．KES II の受け入れ，GNOSIS II，TIPPLER を中心としたオブジェクト指向技術の適用支援，OSMOS の評価，SYSTEM ν［nju：］の開発等を経て，現在，ミドルウェアシステム部 CASE 技術開発室に所属し，UREP を中心としたリポジトリ技術の調査，適用支援に従事．