2009－SE－163 (6)
2009／3／18

# ソースコード生成を利用したオブジェクト指向開発

森望美† 塚本享治†
東京工科大学メディア学部†

エンタープライズ・アプリケーションは情報技術の発達によって大規模化・複雑化しており、開発管理が難しく欠陥を生んでしまうことも多い。このような状況に対応するために J2EE、JavaEE などの開発環境が発展し、それに伴いシステムの開発容易性は向上した。しかし部品間の相互関係を記述するのはいまだに煩雑である。そこで ER 図や HTML などを元に JavaEE 環境に則した Web アプリケーションの基盤を生成するツールを作成した。さらにオブジェクト指向開発方法に沿って分析・設計を行い、その後このツールを用いてコード生成する開発実験を行った。その結果ビジネスロジック層が適切に部品化でき、本ツールの生成物と組み合わせて実際に動作するアプリケーションが開発できた。

# Experiments on Object Oriented Development using a Code Generator

Nozomi Mori† Michiharu Tsukamoto†
Tokyo University of Technology†

The enterprise applications have includes their scales and complexities. J2EE and its successor JavaEE architectures were proposed to make their development easy. However it is as complex as ever to describe the relations and to adjust the parameters among source programs and configuration files. To overcome the problems, we have developed a code generator that supports the development of web applications. We carried out experiments of code generation according to object oriented development process. This paper describes the result that the business logic layer was able to be made components appropriately.

## 1 はじめに

エンタープライズ・アプリケーションは、ネットワークを介した連携、データベースなどリソースの分散などによって、ますます大規模化・複雑化している。そのため全体の管理が難しく、欠陥を生んでしまうことも多い。このような状況に対応するために J2EE、JavaEE などの開発環境が発展してきた。提供される機能や開発生産性が強化されてきているが、それでもシステム全体の設定部分を正しく記述するのは複雑で手間がかかる。

そこで[3]ではモデル図や HTML などからソースコードを生成しさらにアプリケーションの大部分を生成するツールを作成した(以下本ツール)。このツールを用いた場合、システム固有の処理であるビジネスロジック層は開発者が開

発することになる。そこで本稿では実際に用いられている一般的なソフトウェア開発方法において本ツールを適用し、ソフトウェア開発工程において本ツールがどのように利用できるか検討し、それに従って開発実験を行った。

## 2 ソースコード生成ツール[3]

### 2.1 アプローチ

JavaEE 環境ではデータベース側は EJB3.0 エンティティ Bean、画面側は JSF、ビジネスロジックの部分には EJB3.0 セッション Bean を用いるのが一般的である。これらを一つのモデルから一度に生成することはできないので、各部分ごとに変換し Web アプリケーションに必要なソースコードを生成した。

図の描画には XMI にも対応しているシステム設計支援ツール JUDE professional[2]を使用した。

### 2.2 ER 図からのエンティティ Bean 生成

データベース側のエンティティ Bean は以前から広くデータベース設計に用いられている ER 図から生成した。

エンティティ Bean はデータベースのテーブルに対応する Java オブジェクトで、属性のセッター・ゲッターが記述される。EJB3.0 で導入された Java Persistence API(JPA)が提供するアノテーションを付加することによって様々な設定を施すことができる。ER 図はデータベース設計に用いる図で、エンティティ Bean とは含む要素がほぼ同じである。

図 2-1 にエンティティ Bean の生成過程を示す。JUDE で ER 図を描画し、XMI 形式にして出力し、この XMI に XSLT 変換を行ってエンティティ Bean を生成した。

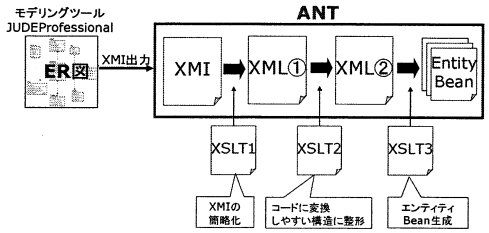


図 2-1：エンティティ Bean 生成過程

生成可能なアノテーションは表 2-1 の通りである。ER 図から情報が得られないため、カスケード、継承のタイプ指定などは現状では実装していない。

表 2-1：生成可能なアノテーション

| | 種類 | 属性 |
|---|---|---|
| プライマリーキー | Id | |
| | GeneratedValue | *strategy=AUTO |
| | IdClass | value=主キークラス名.class |
| Temporal | Temporal | value=DATE、CALENDER |
| リレーションシップ | ManyToOne<br>OneToMany<br>OneToOne<br>ManyToMany | targetEntity=クラス名.class<br>mappedBy=変数名<br>**cascade=タイプ名<br>fetch=EAGER、LAZY |
| ジョインテーブル | JoinTable | joinColumns<br>inverseJoinColumns |
| ジョインカラム | JoinColumn | name<br>referencedColumnName |
| | **PrimaryKeyJoinColumn | |
| 継承 | Inheritance | *strategy=SINGLE_TABLE、JOINED |

同時に DAO(Data Access Object)も生成した。これは本稿においてはエンティティマネージャーがエンティティの永続化などの操作をするクラスを指すものである。アプリケーションにおいて、プレゼンテーション層からエンティティにアクセスしたい場合、エンティティがエンティティマネージャーの管理下になければアクセスできない。このため必要な時にエンティティにアクセスできないという問題が生じることがある。これを解決するために、本ツールではエンティティマネージャーの生存期間を拡張した状態で生成した。具体的にはエンティティマネージャーの変数に対し拡張のためのアノテーションを付加している[5]。

### 2.3 HTML からの JSF ページ生成

プレゼンテーション層に位置する JSF ページは HTML ページから生成した。また、最終的にできあがる Web アプリケーションを動かす上で必要なファイルも生成した。

HTML ページから JSF ページと必要なファイルを生成する過程を図 2-2 に示す。

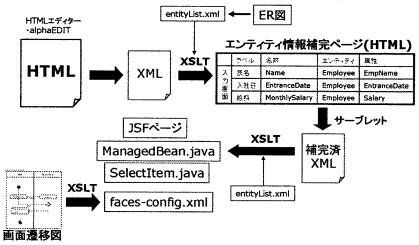


図 2-2：Web アプリケーションの生成過程

アプリケーションに対する入出力情報は何らかのエンティティの属性であることが多いので、エンティティ Bean と連携する JSF ページの生成を行なうことにした。連携させるには、JSF ページにおいてフォーム部品の value 属性の値を “#{エンティティ名．属性名} “という記述にすればよい。この記述を持った JSF ページを生成することが目的である。

生成する上でモデル図から得られない情報は

図 2-3 のような config.xml として用意する。

```
<war-config>
  <earname dir="C:/jboss/createJsf-war.war">www</earname>
  <entitydir>D:/ERCreate/23-Manager</entitydir>
  <selectItems>
    <selectItem>
      <entity>Department</entity>
      <field>departmentName</field>
    </selectItem>
    <selectItem>
      <entity>Job</entity>
      <field>JobName</field>
    </selectItem>
  </selectItems>
</war-config>
```

図 2-3 : config.xml

以下、生成方法を説明する。

(1) HTML ページを用意する。

(2) マッシュアップツールで HTML ページを XML に変換し、XSLT でエンティティ情報を付加するための対応表に変換する。

(3) エンティティ情報とボタンのアクション先を付加する。

(4) 整形された XML に対して XSLT 変換を行い JSF ページを生成する。JSF ページ生成と同時に、マネージド Bean、faces-config.xml、application.xml、build.properties などの Web アプリケーションを動かすために必要なファイルも生成する。faces-config.xml には画面遷移情報を記述する必要があるが、本ツールでは UML のアクティビティ図で描画した画面遷移図を入力とし画面遷移情報を生成する。

## 3 本ツールを用いたオブジェクト指向開発

2 章で述べたようなツールを[3]で作成したが、このツールを実際の開発方法の中で利用できることが必要である。そこで、一般的なオブジェクト指向開発法に沿って本ツールを用いた開発方法を明確にし、本ツールの位置づけを明らかにすることにした。

### 3.1 ビジネスロジック層の部品化

本ツールでは全てのアクションメソッドを持つマネージド Bean が 1 つだけ生成されるようになっている。ほぼ全てのロジックをこのマネージド Bean に書いてもアプリケーション全体の動作にあまり影響はない。しかしシステムが大規模で複雑になるにつれ、複数人での開発やプログラムの再利用、保守性、品質などの観点から部品化は必要不可欠となる。そこで、実際に使われている開発方法の中で本ツールを用いることで、部品化を考慮したシステムが開発できるかどうかを検討する。

### 3.2 ソフトウェア開発における本ツールの位置づけ

実際のソフトウェア開発では分析・設計・実装という流れがある。一般的な開発方法では、分析の段階でユースケース図、ユースケース記述、分析クラス図などが導出され、システムに必要なオブジェクトは画面、コントローラ、エンティティに分類される(図 3-1)。

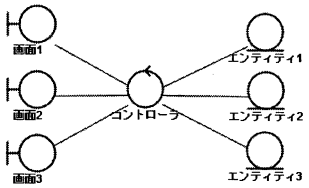


図 3-1 : 分析によって抽出されるシステムの構造(ロバストネス図)

設計段階を終えた時点でエンティティの関係図、画面遷移図、画面構成などができあがる。これらを実装段階で本ツールへ入力し、設計通りに画面やエンティティを生成する。

残ったビジネスロジック層は分析・設計の流れで通常通り開発することになる。開発の仕方は開発者に依るところであり、処理をどの部品に持たせるか、部品をどれくらいに分割するかなどは自由である。

ソフトウェア開発において本ツールは、設計どおりに実装することを可能にするとともに、定型的な部品の開発をロジック部分から分離させることによって、開発者をビジネスロジック開発に専念させることが可能となる。

### 3.3 継承を用いた生成コードの拡張とオーバーライド

実際に動くアプリケーションにするためには不足している処理を書き足す必要がある。しかし生成物に直接手を加えると、変えてはいけない部分まで変えてしまう恐れがある。そこで生成物を継承したクラスを作り、そこに固有の処理を書くことにした。自分の作成したものと本ツールが生成したものの区別が明確になり、生成したクラスをテストし直す必要がなくなる。継承の対象は以下の 2 つである。

(1)マネージド Bean

継承元のマネージド Bean はエンティティ Bean のセッター・ゲッターと空のアクションメソッドが書かれているものである。これを継承してアクションメソッドの内容を追加する。

(2)DAO

DAO は EJB3.0 セッション Bean として実装する。セッション Bean はインタフェースをインプリメントするルールになっているので、図 3-2 のようにインタフェースも実装側も継承する。

セッション Bean では、インタフェースにはアノテーション@Remote かアノテーション@Local、実装にはアノテーション@Stateless かアノテーション@Stateful がそれぞれ必要であるが、継承を用いる場合には親か子のどちらかに付加されていれば動作する。

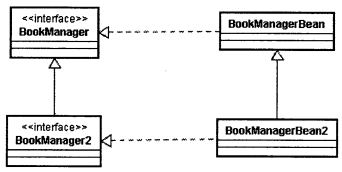


図 3-2:DAO の継承関係

### 3.4 ビジネスロジック層の部品の利用方法

開発工程を経て導出されたビジネスロジック層の部品は、画面上のボタンが押された時に動作するべき処理を持っている。しかし本ツールで生成される画面は同時に生成されるマネージド Bean と整合が取られているため、ボタンのアクション先をビジネスロジック部品に応じて変えることは困難である。そこで、画面側のアクション先は変更せず、生成されたマネージド Bean にビジネスロジック部品の呼び出し処理を記述することで使用することにした。図 3-3 はビジネスロジック部分を部品化したシステム構成のイメージ図である。

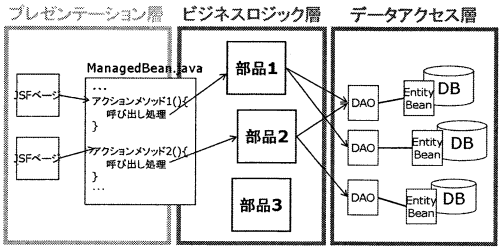


図 3-3：部品化したシステムのイメージ図

## 4 本ツールを用いたシステム開発実験

これまでに述べた生成方法に従って開発実験を行った。まず、本ツールで生成したマネージド Bean に全ての処理を記述する方法の実験について述べ、次に 3 章で述べたオブジェクト指向開発に則った方法の実験について述べる。その後それらの方法を比較する。

### 4.1 開発実験対象「オンラインブックストア」

題材に用いたのはマスタリング JavaEE5[1]の第 33 章で取り上げられているオンラインブックストア(以下、例題)である。掲載されていたソースコードを基に図 4-1 の ER 図と図 4-2 の画面遷移図を作成し本ツールへの入力とした。

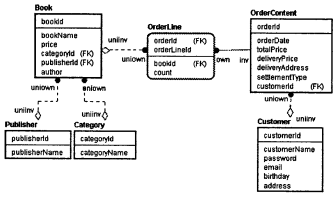


図 4-1：生成実験用 ER 図

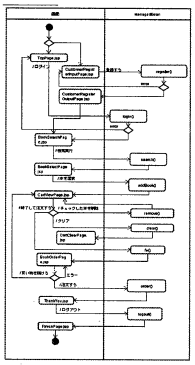

図 4-2：生成実験用画面遷移図

### 4.2 実験 1:マネージド Bean に全ての処理を記述する方法

まず本ツールで生成されるマネージド Bean に全処理を記述する方法で開発実験を行った。

#### 4.2.1 実験手順

次の順序で開発実験を行った。

(1)本ツールを用いて ER 図からエンティティ Bean、アクティビティ図から画面遷移情報を生成する。

(2)config.xml、HTML を作成し、本ツールを用いて JSF ページと伴にアプリケーションの基盤を生成する。

(3)生成されたマネージド Bean や DAO にアプリケーション固有の処理を追加する。

#### 4.2.2 実験結果

実験 1 で開発したシステムの全体図を図 4-3 に示す。開発者が補ったのは点線で囲んだ部分である。

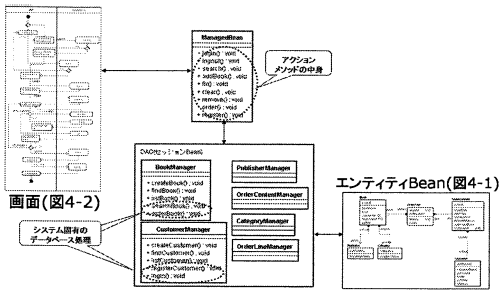


図 4-3：生成ツールを利用し開発したオンラインブックストア全体図

表 4-1 は生成されたファイルとその行数である(空行含む)。

表 4-1：生成されたファイルと行数

| ファイル | ファイル数 | 総行数 |
|---|---|---|
| エンティティBean | 6+主キーファイル | 506行 |
| DAO | 12 | 468行 |
| Jsfページ | 10 | 331行 |
| ManagedBean | 1 | 95行 |
| SelectItem.java | 1 | 40行 |
| faces-config.xml | 1 | 143行 |
| build.properties | 1 | 4行 |
| Application.xml | 1 | 12行 |
| | 合計 | 1623行 |

表 4-2 は書き足した行数である(空行含む)。

表 4-2：書き足したファイルと行数

| ファイル | 行数 |
|---|---|
| ManagedBean.java(補) | 150 |
| CartItemBean.java(新) | 80 |
| BookManagerBean.java(補) | 65 |
| CustomerManagerBean(補) | 31 |
| 計 | 326 |

新:新規にファイルを作成
補:生成されたファイルへの書き足し

ManagedBean.java には各処理をすべて記述したため書き足した行数が多い。CartItemBean.java はカート処理の際に選択した商品の情報を格納するクラスで、新規に作成したものである。BookManagerBean.java、CustomerManagerBean.java はエンティティマネージャーが管理するデータベース操作を記述するクラスで、主キーによる検索、全データの取得、新規オブジェクトの作成の処理が生成時に用意されている。BookManagerBean.java には検索処理と注文における挿入処理を追加し、CustomerManagerBean.java にはログインにおける顧客の検索処理と顧客登録における挿入処理を追加した。

### 4.3 実験 2:オブジェクト指向開発方法

3 章で述べた、オブジェクト指向開発に則った方法で実験を行った。

#### 4.3.1 実験手順

次の順序で開発実験を行った。

(1)分析・設計する。

(2)本ツールを用いて ER 図からエンティティ Bean、アクティビティ図から画面遷移情報を生成する。

(3)config.xml、HTML を作成し、本ツールを用いて JSF ページとともにアプリケーションの基盤を生成する。

(4)ビジネスロジック層を実装し、部品を呼び出す処理をマネージド Bean に記述する。

#### 4.3.2 オンラインブックストアの分析

3 章で述べたような開発を行うためにオンラインブックストアの分析を行った。ICONIX プロセス[4]に則り、ユースケース記述、シーケンス図、ロバストネス図を洗練していく方法をとった。第 1 段階では大まかな分析を行う。ユースケース記述を作成しエンティティを抽出する。第 2 段階では詳細な分析としてユースケース記述とシーケンス図を詳細化する。第 3 段階では設計を行う。本ツールを使うことを前提として必要な部品を追加していく。以下に詳細を示す。

(1)第 1 段階：分析

①ユースケース図

図 4-4 に示すように 4 つのユースケースを抽出した。

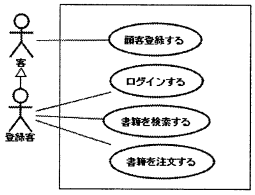


図 4-4：オンラインブックストアのユースケース図

②ユースケース記述

ユースケースごとに簡単なユースケース記述を作成する。例として「書籍を注文する」のユースケース記述を示す。

| ユースケース | 本を注文する |
|---|---|
| 事前条件 | ログインしている |
| 事後条件 | 本が注文されている |
| 基本フロー | 1.ユーザは本を選択し注文を依頼する。<br>2.システムは注文処理を行い、注文完了を通知する。 |

③ロバストネス図

ユースケース記述から抽出したエンティティオブジェクトに必要と思われるインタフェース、コントローラを付加しロバストネス図を作成する。図 4-5 に示す。

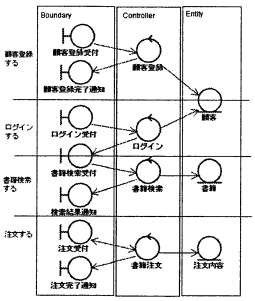


**図 4-5：分析で導出したオンラインブックストアのロバストネス図**

④シーケンス図

ユースケース記述からシーケンス図を作成する。例として「書籍を注文する」のシーケンス図を図 4-6 に示す。

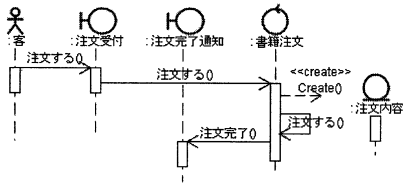


**図 4-6：第 1 段階分析で導出したシーケンス図(書籍を注文する)**

(2)第 2 段階：詳細分析

第 1 段階のユースケース記述を詳細化し、不足しているクラスなどを洗い出す。

①ユースケース記述

例として「書籍を注文する」のユースケース記述を示す。第 1 段階の分析では書籍を一冊だけ選びそのまま注文するという処理になっていたが、現実的には複数選んで注文するという処理が必要である。そこでカートの概念を導入しユースケース記述を詳細化した。

| ユースケース | 本を注文する |
|---|---|
| 事前条件 | ログインしている |
| 事後条件 | 本が注文されている |
| 基本フロー | 1.ユーザは検索結果から本を選択し、カートへの追加を依頼する。<br>2.システムは選択された本をカートに追加し、カート内容を通知する。<br>3.必要な回数 1、2 を繰り返す。<br>4.ユーザはシステムに注文を依頼する。<br>5.システムは注文内容を通知する。<br>6.ユーザはシステムに注文を依頼する。<br>7.システムは注文処理をし、注文完了を通知し、本ユースケースが終了する。 |
| 代替フロー | 3-a.選択した本を削除したい場合、(1)本を選択しシステムに削除を依頼する。 (2)システムは更新されたカートの内容を通知し、基本フロー2 に戻る。<br>3-b.カートを空にしたい場合、(1)システムにカートのクリアを依頼する。(2)システムはクリア完了を通知する。<br>7-a.買い物を続けたい場合は基本フロー1 に戻る。 |

②シーケンス図

詳細化したユースケース記述からシーケンス図を作成する。引き続き「書籍を注文する」の場合を示す。カート処理が必要であるため、コントローラである「追加」、一時的に追加する書籍の内容を保持するエンティティである「追加書籍」、カート内容を表示する画面である「カート内容表示画面」、カートクリア完了を通知する「カートクリア完了画面」を新たに加えた。図 4-7 に示す。

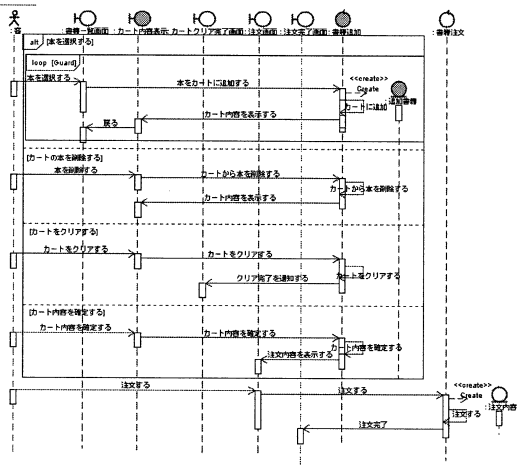


**図 4-7：第 2 段階分析で導出したシーケンス図(書籍を注文する)**

③修正したロバストネス図

詳細分析で新たに加わったカート処理に必要な要素を反映すると図 4-8 のようになる。

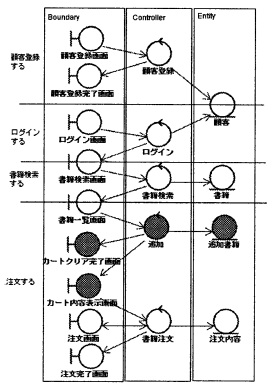


**図 4-8：洗練後カート処理が追加されたオンラインブックストアのロバストネス図**

(3)第 3 段階：設計

分析で得られた要素を基に設計していく。本ツールを使うことを前提とした実装設計が必要

である。本ツールではJSFを用いるため、画面からのアクションを受けるマネージドBeanが必要となる。また、データベースの操作はDAOに持たせる。そのクラス図は図4-9のようになった。

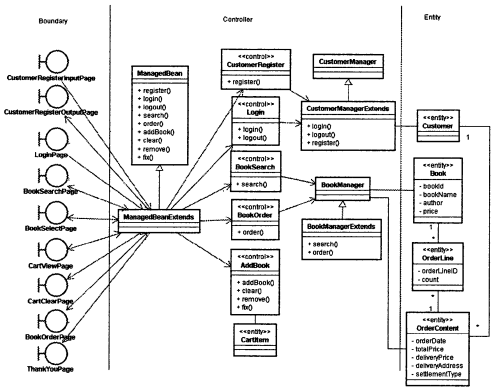


図 4-9：本ツールを利用することを考慮したオンラインブックストアの設計クラス図

最終的に抽出したビジネスロジック層の部品とそのメソッドは以下のとおりである。

(1)CustomerRegister：顧客登録
(2)Login：ログイン、ログアウト
(3)BookSearch：検索
(4)BookOrder：注文
(5)AddBookBean：カートへの追加、カートのクリア、カート内要素の削除、カート内容の確定

### 4.3.3 実験結果

図4-10に実験2で開発したシステムの全体図を示す。開発者が補ったのは点線で囲んだ部分である。

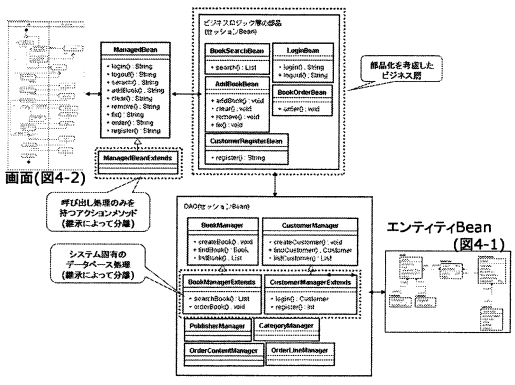


図 4-10：オブジェクト指向開発法に沿って開発したオンラインブックストア全体図

生成されたファイルと行数は表4-1と同じである。表4-3は分析、設計を経て作成したファイルとその行数である。

表 4-3：実験2で作成したファイルと行数

| 種類 | ファイル | 行数 |
|---|---|---|
| JSFページ | CartViewPage.jsp | 26 |
| マネージドBean | ManagedBean.java | 70 |
| ビジネスロジック層 | LoginBean.java | 60 |
| | BookSearchBean.java | 57 |
| | AddBookBean.java | 134 |
| | BookOrderBean.java | 73 |
| | CustomerRegisterBean.java | 42 |
| | CartItemBean.java | 80 |
| DAO | BookManagerExtends.java | 11 |
| | BookManagerBeanExtends.java | 88 |
| | CustomerManagerExtends.java | 10 |
| | CustomerManagerBeanExtends.java | 34 |
| | 合計 | 685 |

実験2では生成されたファイルに処理を書き足すということはせず、新規作成するか継承によって処理を補完する方法をとった。実験1でManagedBean.javaに書いていた全ての処理は、5つの部品とカート処理に必要なCartItemBean.javaに部品化された。ManagedBean.javaは継承し、ビジネスロジック層の部品を呼び出す処理を記述した。DAOについては、実験1では生成されたファイルに直接処理を追加していたが、実験2では3.3で述べたように継承し拡張している。インタフェースBookManager.javaを継承したBookManagerExtends.javaには検索と注文に必要なデータベース操作をそれぞれ追加した。インタフェースCustomerManager.javaを継承したCustomerManagerExtends.javaにはログインと顧客登録に必要なデータベース処理を追加した。

実験1で書き足した行数(表4-2)より350行あまり増加しているが、これは部品化したことによるインポート文や呼び出し処理の増加分である。ManagedBeanExtends.javaは継承する際メソッドの記述が一部重複するため、その分も増加している。

## 5 評価と考察

(1)エンティティBean

生成されたエンティティBeanは手直しを加えることなく使用することができた。DAOをステートフルセッションBeanとして実装[5]したことによって、プレゼンテーション層からエンティティBeanに直接アクセスできるようになり、対象エンティティに関連するエンティティを取り出すなど、柔軟なエンティティBeanの利用が可能になった。

しかし、指定できないアノテーションもあるため、それらの情報をどのように取り入れるかを考える必要がある。

(2)JSF ページ

画面においては入出力フォームのエンティティへの対応付けやボタンのアクション先については正常に変換され動作した。しかし、対応表から入力された情報を XML にする際には出現順に処理しているわけではないため、細かなレイアウトなどは反映できない。また見出しやフォーム部品以外のテキストは処理の対象にしていないため反映しない。

JSF ページに必要な情報を得た後はサーブレットで処理し XML にしているが、出現順に処理することができないため、このサーブレット部分に DOM や SAX などを用いることも考えられる。

(3)ビジネスロジック層の開発

プログラミングの観点で見ると、実験 1 のようにマネージド Bean に直接全てのビジネスロジックを書き加えるのは現実的ではない。むやみに書き足すことによって自動生成で保障されていた部分までが不明確になっていく。一方実験 2 のように継承を用いて拡張する方法をとると独自に開発した部分が明確になり安全である。

ビジネスロジック層の部品化の効果としては、アクションを受け取る部分であるマネージド Bean のアクションメソッドが呼び出し処理に特化するため小規模化し、呼び出し処理が一か所にまとまる。またユースケースごとにビジネスロジックを持つ部品が存在することになり内部がシンプルで明確となる。

さらに、実際の現場で利用することを想定し、オブジェクト指向開発法に則った開発方法を検討した結果、本ツールを使用するための開発方法を新たに考案するのではなく、通常の開発方法の中で部分的に本ツールを利用できることが分かった。定型的な部品の開発を自動化し、開発負担を減らすことで、ビジネスロジック開発に専念できるようになる。また負担軽減だけでなく、設計を実装にそのまま反映できる点でも本ツールの使用は有効である。

(4)コードの自動生成による効果

エンティティ Bean や JSF ページの非常に複雑な関係の記述が、本ツールによる自動生成によって容易に実現できるようになった。

さらに、エンティティ Bean と JSF ページを使用したアプリケーションの設定作業の負担も大幅に軽減された。本ツールを使用することで、JSF ページとそれに使用するエンティティ Bean との関係、画面遷移情報とマネージド Bean の登録を記述する faces-config.xml の整合性などが正確に記述できる。またアプリケーションをデプロイするために必要なプロパティファイル、アーカイブの詳細を記述するファイルなど、アプリケーションの実行に必要な部品までが生成できるようになった。一部の変更が自動的に反映されデバッグの労力が軽減できた。

## 6 おわりに

ER 図からエンティティ Bean、HTML ページから JSF ページを生成するツールを開発し、実験を行ってそのツールがオブジェクト指向開発法の中でどのように利用できるかを明らかにした。エンティティ Bean と JSF ページは自動生成することができ、ビジネスロジック層を開発者が加えることで動作するアプリケーションを開発することができた。

システム全体を対象にするのではなく、通常のオブジェクト指向開発の中に本ツールを位置づけることで、比較的取入れ易いものになった。

各変換において改善点は多くあるが、動作するアプリケーションを実験で開発することができた。

## 参考文献

[1]斉藤賢哉,"マスタリング JavaEE5",翔泳社, 2007

[2](株)チェンジビジョン,"JUDE professional 5.2.1",http://jude.change-vision.com/

[3]森望美, 塚本享治 : HTML と ER 図からの Web アプリケーションの生成, 情報処理学会研究報告, 2008-SE-162, 2008

[4]ダグ・ローゼンバーグ, マット・ステファン, "ユースケース駆動開発実践ガイド", 翔泳社, 2007

[5]松信嘉範, "Java データアクセス実践講座", chapter11, 翔泳社, 2008