社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2008－SE－162 (10)
2008／11／19

HTMLとER図からのWebアプリケーション生成

森望美† 塚本享治†
東京工科大学メディア学部†

ソフトウェア開発においてモデル駆動(MDA、MDD)という概念が用いられるようになり、様々なMDAツールが開発されている。本研究では設定ファイルが複雑なJavaEE環境での開発に着目し、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。エンタープライズシステムのデータアクセス層であるエンティティBeanをER図から、プレゼンテーション層であるJSFページをHTMLから生成し、これらを連携させたアプリケーションを生成する。オンラインブックストアを例に生成実験を行い、基盤として利用できるものが生成できることが分かった。

Code Generation for Web Application from HTML Pages and Entity-relationship Diagrams

Nozomi Mori† Michiharu Tsukamoto†
Tokyo University of Technology†

Developing Java EE applications we have to code java programs and configuration files. Even though it became easy, it is necessary to adjust a lot of parameters between source codes and configuration files. To overcome the problems, we have developed a tool that supports the development of the Web application. This paper presents the tool and discusses trial experiments.
Using the tool, entity beans of data access tier are generated from ER diagrams, JSF pages of presentation tier are generated from HTML pages, and configuration files with parameters between source codes are also automatically generated. We use an online bookstore sample from the technical book for the trial experiment, and discuss the generated codes and the book's codes.

1 はじめに

近年UMLの普及とシステム開発の大規模化、複雑化に伴いソフトウェア開発にモデル駆動型開発(MDD)が採用されるようになってきた。

一方、大規模なエンタープライズシステムを開発するために整えられたJavaEE環境での開発は、実装の記述が簡易である代わりに複数の設定ファイルを用意しなければならず、それらに現れる数多くの設定パラメータの整合を取ることに苦労する。

そこで従来開発で使用されてきた図やHTMLを変換することによって相互に整合性の取れたアプリケーションの基盤を生成することを考えた。本研究ではER図とHTMLからJavaEEの技術を用いたWebアプリケーションを生成するツール(以下本ツール)を作成したので報告する。

2　アプローチ

システムの各部分はそれぞれ性質が異なり設計方法も違う。図2-1に示すようにJavaEE環境ではデータベース側はエンティティBean、画面側はJSF、ビジネスロジックの部分にはセッション Bean を用いるのが一般的である。

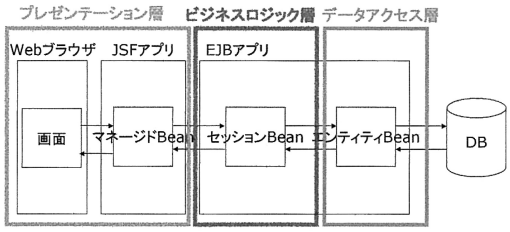

図 2-1：JavaEE 環境のシステム構造例

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、それらを統合することで Web アプリケーションの生成を実現しようと考えた。ただし、システムの核となるビジネスロジックは非常に複雑で機械的な生成に向いていないため、本ツールでは開発者が書くことを想定している。

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 図から生成する。同時にエンティティの基本的な操作をするセッション Bean も生成する。画面側の JSF ページは HTML ページから生成し、設定ファイルに記述する必要がある画面遷移情報の指定には UML のアクティビティ図を使用する。

ER 図とアクティビティ図の描画には XMI にも対応しているシステム設計支援ツール JUDE professional[2]を使用する。

3　ER 図からのエンティティ Bean 生成

データアクセス層に位置するエンティティ Bean は ER 図から生成する。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、カラムに対応する属性のセッター・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り変換が可能である。

3.1 生成手順

図 3-1 にエンティティ Bean の生成過程を示す。

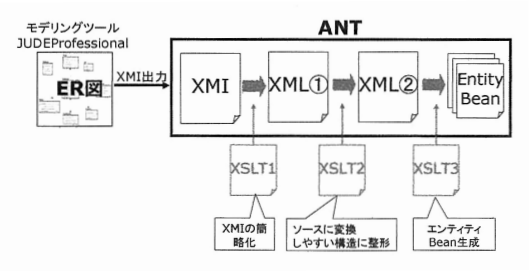

図 3-1：エンティティ Bean 生成過程

JUDE で ER 図を描画し、XMI 形式にして出力する。この XMI に XSLT 変換を行ってエンティティ Bean を生成した。

生成可能なアノテーションは表 3-1 の通りである。＊がついているカスケード、継承のタイプ指定は現状では実装していない。継承はデフォルトで SINGE_TABLE タイプに限定してあり、コメントで JOINED タイプも付けている。

表 3-1：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 OneToMany OneToOne ManyToMany	targetEntity=クラス名.class mappedBy=変数名 *cascade=タイプ名 fetch=EAGER、LAZY
ジョインテーブル	JoinTable	joinColumns inverseJoinColumns
ジョインカラム	JoinColumn	name referencedColumnName
	*PrimaryKeyJoinColumn	
継承	Inheritance	*strategy=SINGLE_TABLE、JOINED

XMI は XSLT を用いて 3 段階で変換する。変換手順を説明するための例として図 3-2 の ER 図を用いる。

図 3-2：従業員と部署の ER 図

3.1.1 第 1XSLT 変換

第 1 段階の変換では、XMI から必要な部分だけ抜き出して整形する。図 3-3 に示すようにリレーション情報には関連の各端から見た

情報が記述してある。

```
<JUDE:ERRelationship>
  <UML:Association.connection>
    <JUDE:ERRelationshipEnd>
      <UML:Feature.owner>
        <UML:Classifier xmi.idref="EmployeeへのID"/>
      </UML:Feature.owner>
      <UML:StructuralFeature.multiplicity>
        <UML:Multiplicity>
          <UML:Multiplicity.range>
            <UML:MultiplicityRange lower="0" upper="1"/>   相手の多重度
          </UML:Multiplicity.range>
        </UML:Multiplicity>
      </UML:StructuralFeature.multiplicity>
      <UML:AssociationEnd.participant>
        <UML:Classifier xmi.idref="DepartmentへのID"/>
      </UML:AssociationEnd.participant>
    </JUDE:ERRelationshipEnd>
    <JUDE:ERRelationshipEnd>
      <UML:Feature.owner>
        <UML:Classifier xmi.idref="DepartmentへのID"/>
      </UML:Feature.owner>
      <UML:StructuralFeature.multiplicity>
        <UML:Multiplicity>
          <UML:Multiplicity.range>
            <UML:MultiplicityRange lower="0" upper="-1"/>   相手の多重度
          </UML:Multiplicity.range>
        </UML:Multiplicity>
      </UML:StructuralFeature.multiplicity>
      <UML:AssociationEnd.participant>
        <UML:Classifier xmi.idref="EmployeeへのID"/>
      </UML:AssociationEnd.participant>
    </JUDE:ERRelationshipEnd>
  </UML:Association.connection>
</JUDE:ERRelationship>
```

片側から見たリレーション

片側から見たリレーション

図 3-3：XMI リレーション情報部分

エンティティ情報にはエンティティ名、属性(型、主キー、外部キー)を、リレーション情報には自分側エンティティと相手側エンティティ、相手側エンティティの多重度を図 3-4 のような構造の XML にまとめた。

```
<root>
 <Entity name="Employee">
  <Attribute name="EmployeeId" datatype="int" primarykey="true"
   foreignkey="" foreignOwner=""/>
  <Attribute name="EmployeeName" datatype="String" primarykey=""
   foreignkey="" foreignOwner=""/>
  <Attribute name="DepartmentId" datatype="String" primarykey=""
   foreignkey="true" foreignOwner="Department"/>
 </Entity>
 <Entity name="Department">
  <Attribute name="DepartmentId" datatype="String" primarykey="true"
   foreignkey="" foreignOwner=""/>
  <Attribute name="DepartmentName" datatype="String" primarykey=""
   foreignkey="" foreignOwner=""/>
 </Entity>
 <relationship>
  <relationshipEnd position="1" direction="">
   <association multi="One">Department</association>
   <owner>Employee</owner>
  </relationshipEnd>
  <relationshipEnd position="2" direction="">
   <association multi="Many">Employee</association>
   <owner>Department</owner>
  </relationshipEnd>
 </relationship>
</root>
```

図 3-4：第 1 変換で生成される XML

3.1.2 第 2XSLT 変換

エンティティ情報とリレーション情報をまとめてエンティティごとに整形する。もとの XMI ではエンティティ情報とリレーション情報は同じレベルにあるので、処理しているエンティティ名が Owner であるリレーション情報をたどってエンティティごとにひとまとめになるよう整形した。

また属性については、普通の変数かリレーションを保持する変数か、主キーはいくつ持っているか、外部キーか、どんなアノテーションがつくか、などの情報を持たせた。

リレーションを保持する属性には、どのクラスへのリレーションを保持しているのかという targetEntity、関連エンティティをどのタイミングで取得するかという Fetch 属性まで持たせ、双方向従側に必要な mappedBy 属性は第 3 変換で判断する。相手エンティティが One の場合は外部キーを持つ方が主側であるため、自分の属性の中に相手エンティティの属性があるかを調べ外部キーとして保持させる。図 3-5 は第 2 変換で生成される XML である。

```
<root>
 <Entity name="Employee" position="1" id_count="1">
  <Attribute name="EmployeeId" type="normal" datatype="int"
   foreignkey="" primarykey="true" count="1">
   <Annotation>@Id @GeneratedValue(strategy = GenerationType.AUTO)</Annotation>
  </Attribute>
  <Attribute name="EmployeeName" type="normal" datatype="String"
   foreignkey="" primarykey="" count="2"/>
  <Attribute name="DepartmentId" type="normal" datatype="String" foreignkey="true"
   primarykey="" count="3" foreignOwner="Department"/>
  <Attribute name="Department" type="relationField" datatype="Department">
   <Annotation direction="">@ManyToOne(targetEntity=Department.class,
   fetch=FetchType.EAGER</Annotation>
   <Foreignkey>DepartmentId</Foreignkey>
  </Attribute>
 </Entity>
 <Entity name="Department" position="2" id_count="1">
  <Attribute name="DepartmentId" type="normal" datatype="String" foreignkey=""
   primarykey="true" count="1">
   <Annotation>@Id @GeneratedValue(strategy = GenerationType.AUTO)</Annotation>
  </Attribute>
  <Attribute name="DepartmentName" type="normal" datatype="String" foreignkey=""
   primarykey="" count="2"/>
  <Attribute name="Employee" type="relationField" datatype="List&lt;Employee&gt;">
   <Annotation direction="">@OneToMany(targetEntity=Employee.class,
   fetch=FetchType.LAZY</Annotation>
  </Attribute>
 </Entity>
</root>
```

図 3-5：第 2 変換で生成される XML

3.1.3 第 3XSLT 変換

第 2 段階で生成された XML から Java ソースコードを生成する。以下、この XSLT 変換について述べる。

(1) エンティティごとにファイルに出力する。
(2) 基本的な import 文を出力する。
(3) 使用されるアノテーションやリストの import 文を出力する。
(4) 複合主キーを使っていれば@IdClass を出力するとともに主キークラスも生成する。
(5) 継承関係になっていれば親クラスに@inheritance を出力する。
(6) 引数なしのコンストラクタ、すべての属性を引数とするコンストラクタを出力する。
(7) 変数を出力する。
(8) セッター・ゲッターを出力する。属性の種類によって付加するアノテーションは以下のようになる。

- 普通の属性
  Date なら@Temporal をつける。
- OneToOne、ManyToOne の属性

外部キーを持っている方のエンティティが主側であり、リレーションアノテーションの他に@JoinClumn によって外部キー情報を付加する。従側にはリレーションアノテーションに mappedBy 属性をつけるが、単方向の場合は変数が不要なためその旨コメントとして記述する。

- ManyToMany の属性
  双方向の場合主従はどちらでもよいので、エンティティが出現する場所(position)が早い方を主側としたが、どちらが主側になっても良いようにコメントをつけている。リレーションアノテーションの他に@JoinTable も付ける。
- OneToMany の属性
  OneToOne、ManyToOne と同様に外部キーを持っている方のエンティティを主とする。ただし単方向の場合は外部キーを持たない One 側のみがリレーションを保持しているために@JoinColumn の変わりに@JoinTable が必要になるので、単方向の場合のためにコメントで JoinTable を用意している。

主従は区別がつくが双方向か単方向かは見分けがつかない。そこで、JUDE で描画する際リレーションの「動詞句」という項目に以下のいずれかの文字列を記述することで見分けることができるようにした。

- own：双方向主側
- inv：双方向従側
- uniown：単方向主側
- uniinv：単方向従側(リレーションを保持しないエンティティ)

この指定をしない場合でも先に説明したようなコメントがつくようになっている。

エンティティ Bean に加えて、エンティティの主キーによる検索、新規オブジェクトの作成、全データの取得といった基本的な操作を持つ Bean(マネージャーBean)も生成している。これは EJB3.0 のセッション Bean として用意するもので、インターフェースと実装クラスが必要である。

4 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 ページから生成する。また、最終的に出来上がる 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する。

4.1 エンティティ Bean と連携する JSF ページ

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く、JSF をそのエンティティと連携させるとデータの操作がしやすくなるため、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 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。今回はこの記述を持った JSF ページを生成するのが目的である。また、最終的に出来上がる 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する。そのため、JSF だけでなくエンティティ Bean や設定ファイルも含めたアプリケーションのための一式が生成される。

4.2 生成手順

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 4-1 に示す。

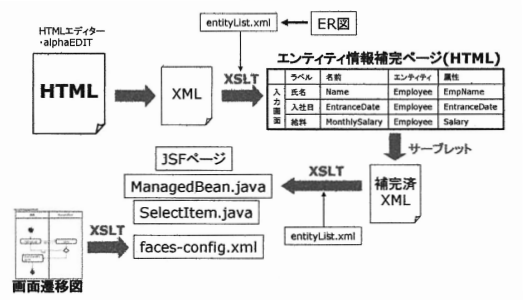

図 4-1： Web アプリケーションの生成過程

生成する上でモデル図から得られない情報は図 4-2 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 4-2：config.xml

- earname：生成するアプリケーションの ear ファイル名
- entitydir：アプリケーション生成に使用するエンティティ Bean 生成ディレクトリ

・selectItems：JSF ページに出現するセレクトボックスの情報。対応するエンティティと属性を記述する。

以下、生成方法を説明する。

(1) HTML ページを html.war に用意する。本ツールで用いる HTML ページには次のような制限を設けた。

- データ新規作成のようなタイプの画面ではラベルとフォーム部品を で囲む。
- 一覧表示タイプの画面では生成したいテーブルをヘッダ一行で表現する。

(2) マッシュアップツールで html.war の中にある 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。サーブレットで処理を行うために対応表を CreateJsf.war に配置しビルドする。これらの一連の処理は ant で行われる。

(3) 各ページにアクセスして各フィールドに対応するエンティティ情報とボタンに対応するアクション先をプルダウンから選び、ページ下にある「XML 化」ボタンを押すとサーブレットがエンティティ情報を含めた XML に整形する。この作業をページの数だけ繰り返す。

一覧タイプの画面の対応表ではその表の基本となるエンティティもプルダウンから選ぶ。この基本エンティティは関連する他のエンティティの属性を取得するために利用する。例えば基本エンティティを Employee とすると

”employee.department.departmentName"というようにピリオドで並べて記述することによって関連のある Department の DepartmentName を取得することができる。

(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以下に示す Web アプリケーションに必要なファイルも生成する。

- マネージド Bean：画面からのアクションを受け取って処理をする部品である。処理を担当するメソッドは 4.3 で説明する画面遷移図を利用して生成する。また、JSF ページでエンティティ Bean を利用するためにはマネージド Bean にエンティティのセッター・ゲッターを用意しなければならないので、エンティティ Bean を生成したときに出来上がるエンティティと属性の一覧ファイル entityList.xml を利用して生成した。

  一覧タイプの画面がある場合は、基本エンティティの全データを取得し List を返すゲッターを用意する。現段階では ManagedBean.java というマネージド Bean を 1 つだけ使用するという想定なので複数のマネージド Bean は生成できない。
- SelectItem.java：画面のプルダウンメニューのオプションにデータベースから取得した値を使用する場合に生成する。どのエンティティのどの属性を使用するかは config.xml に記述しておく。(画面情報から生成すると複数登場したときに不具合が生じるため)
- faces-config.xml：画面遷移情報とマネージド Bean の登録を記述する設定ファイルである。マネージド Bean の登録情報の生成には entityList.xml を利用し、ManagedBean.java に加えてエンティティ Bean 全てを登録するようにする。画面遷移情報は画面遷移図を描画しそれを元に生成する。生成方法は次節で述べる。
- application.xml：アーカイブファイルの情報を記述するファイルである。config.xml に指定した ear ファイル名を使用して生成する。
- build.properties：ビルドに使用するファイルである。config.xml に指定した ear ファイル名を使用して生成する。

最終的にビルドして ear ファイルにして使用するためにフォルダにひとまとめにする。ディレクトリや必要なファイルを整えたフォルダをあらかじめ用意しておき、生成したファイルをそのフォルダの然るべき場所にコピーすることでひとまとめにしたフォルダを生成している。

4.3 画面遷移情報の生成

前節の生成手順(4)で述べた必要なファイル faces-config.xml に記述する画面遷移情報を、ソフトウェア開発の過程で登場するような画面遷移図から生成することにした。画面遷移情報とは、どのページからどのページへ遷移するかを記述した XML 形式のもので、処理結果によって遷移先を分けるなどの指定ができる。

図としてフローを記述できる UML のアクティビティ図を使用し、描画は ER 図と同様に JUDE professional で行う。画面遷移図には画面だけでなく処理を担当するメソッドを描画し、マネージド Bean を生成する際に使用する。

画面遷移のパターンとして以下の 3 つが考えられる。

① 直接別ページに遷移するもの
② アクションメソッドの処理結果によって遷移先が分岐するもの
③ ページ側のリンクによってアクションメソッドが変わるもの

faces-config.xml の記述において、①の場合はページ間のリンクなので何も書く必要がない。②の場合は<from-outcome>タグで記述する返り値によって遷移先を分ける。③の場合はアクションによって処理メソッドが分かれるため、<from-action>タグでどのアクションメソッドから来たかによって遷移先を分ける。

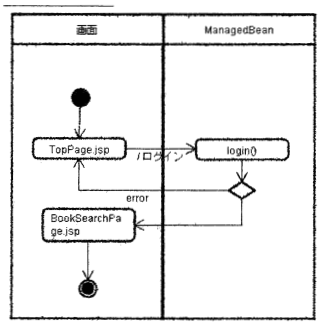

図 4-3：画面遷移例

図 4-3 は TopPage.jsp のボタンを押し login() で認証を行ってエラーなら TopPage.jsp に戻り、成功なら BookSearchPage.jsp に遷移する画面遷移図で、上記のパターン②にあたる。

要素が画面かメソッドか判断するために、画面を表す場合は".jsp"をつける。条件分岐ノードからのびる矢印に付けた文字列(図 4-3 では"error")をマネージド Bean からの返り値として使用する。図 4-4 は図 4-3 から生成された画面遷移情報で、実際には faces-config.xml に埋め込まれる。

```
<navigation-rule>
 <from-view-id>/TopPage.jsp</from-view-id>
 <navigation-case>
  <from-outcome>success</from-outcome>
  <to-view-id>/BookSearchPage.jsp</to-view-id>
 </navigation-case>
 <navigation-case>
  <from-outcome>error</from-outcome>
  <to-view-id>/TopPage.jsp</to-view-id>
 </navigation-case>
</navigation-rule>
```

図 4-4：生成した画面遷移情報

5 生成実験

5.1 題材「オンラインブックストア」

マスタリング JavaEE5[1]の第 33 章で用いられているオンラインブックストアの例を利用し生成実験を行った。このアプリケーションはログイン、顧客登録、書籍の検索、注文を行うものである。図 5-1 はその ER 図、図 5-2 は本研究用に手を加えた画面遷移図である。

5.2 実験結果

図 5-3 に本ツールで生成されたアプリケーションの構造を示す。

以下は生成結果である。

[生成されたファイル]
- エンティティ Bean：6＋主キークラス
- セッション Bean：12 ファイル
- マネージド Bean：1 ファイル
- JSF ページ：10 ファイル
- 設定ファイル：faces-config.xml、application.xml
- その他：SelectItem.java、build.properties

[書き足す部分]
- アクションメソッドの内容(図 5-3 の丸で囲んだ部分)

[動作確認]
- ログイン、顧客登録、検索

※カートの部分は多少複雑なためアクションメソッド以外も追加しなければならない。

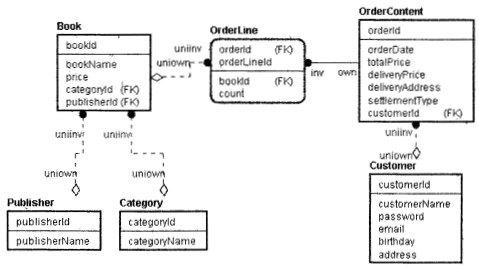

図 5-1：生成実験用 ER 図

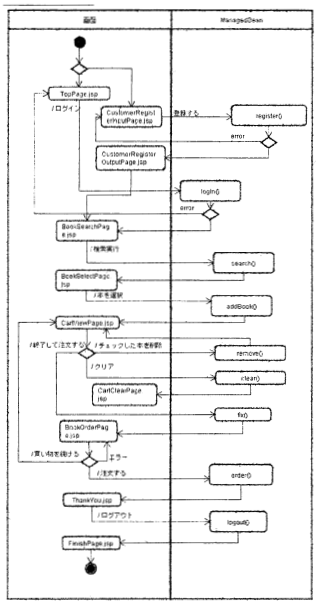

図 5-2：生成実験用画面遷移図

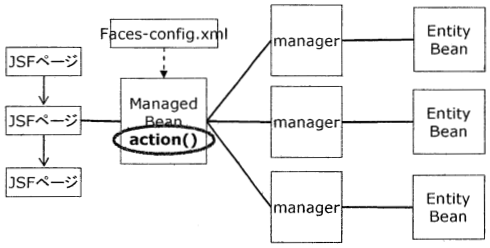

図 5-3：生成されたアプリケーションの構造

5.3 結果に対する考察

5.3.1 実際の例との違いについて

本ツールで生成したファイルは前節の通りであるが、実際に[1]で用いられている例では以下のような構造になっている。

・エンティティ Bean：6＋主キークラス

・セッション Bean：書籍に関する Bean、顧客に関する Bean、Exception、TO など 6 ファイル

・マネージド Bean：6 ファイル

・JSF ページ：10 ファイル

マネージド Bean は主にログイン、顧客情報登録、カート、書籍注文それぞれに関するものに分かれており、その他にアクションメソッドを持たない値を格納するだけの Bean も含め 6 ファイルある。

現時点で本ツールでは全てのアクションメソッドを含むマネージド Bean が 1 つだけ生成されるようになっている。部品化の点で問題があるが、同じような動作は可能である。将来はマネージド Bean やセッション Bean の構成を設計して組み込むことも考えられる。

また、例の JSF ページでは入力した値をアクション先のマネージド Bean で受け取るが、本ツールで生成する JSF ページではフォームの部品をエンティティ Bean に対応させるため、プレゼンテーション層からエンティティ Bean にアクセスすることになる。同じ理由から、エンティティに現れない値は JSF に変換できないため今後対応する必要がある。

5.3.2 部品化を考慮した開発への利用

本ツールではデータベース側と画面側を生成し、それらをつなぐ処理の部分はマネージド Bean に書き足すことを想定している。しかし、大規模で複雑なシステムの場合は処理を部品化するべきである。そこで本ツールを用いた開発でも部品化を考慮できないか考えた。

一つの方法として、処理を担当する部品(セッション Bean など)をあらかじめ用意しておき、マネージド Bean のメソッドを呼び出しのみに使うということが挙げられる。たくさん部品を用意しておき、必要なものだけ繋げて使うという方法である。この方法ならば自動生成部分や手を加える部分が明確に分かれ、複数人での開発にも対応できる。また、部品にフィールドを用意しておき JSF ページを作る対応表の選択肢に加えることで、エンティティには現れない、一時的に保持するだけの値にも対応できると考えられる。以上のこ

とを行うためには、本ツールの改良はわずかであると予想される。

6 既存ツールとの比較

本ツールに関連のある代表的な 3 つのツールの特徴を挙げ、本ツールと比較した。

6.1 jpa2web[3]

jpa2web は JPA を用いたエンティティ Bean のアノテーションを利用してデータベースのメンテナンスを行う Web ページを生成するツールで、エンティティの関連に対応したきれいな画面が生成される。

本ツールでは jpa2web が入力としているエンティティ Bean を図から生成するため、このツールと連携することもできると考えられる。また、メンテナンス用に画面を設計して似たような機能の Web ページを作ることもできる。

6.2 AndroMDA[4]

AndroMDA は Hibernate、Spring などを取り入れた MDA ツールで、XMI を入力として変換を行う。カートリッジと呼ばれるプラグイン機能があり、カスタマイズできるため、様々なプラットフォームに対応するプログラミング言語に変換することができる。この点では、本ツールでの生成対象は EJB3.0 のみである。また CRUD のマスターメンテナンスアプリケーションを自動生成する機能も備えている。本ツールでは画面設計し処理を書き足すことで生成できる。また、AndroMDA は OR マッピングにおいて Hibernate の設定ファイルを生成することができるが、本ツールは JPA を用いるエンティティ Bean を生成しているため OR マッピングにも対応している。

6.3 JBoss Seam[5]

JBoss Seam はアプリケーション開発のフレームワークで、EJB3.0、JSF などの技術をつなぎ合わせる。よってユーザーは 1 つのフレームワークを扱っているかのように開発を行うことができる。

JavaEE 環境での開発において、プレゼンテーション層からエンティティに直接アクセスができず、設計には存在しない DTO などのオブジェクトを使わなければならないという問題がある。しかし JBoss Seam では EJB を Seam コンポーネントとして扱うことでエンティティに直接アクセスできるようになっている。本ツールではロジック部分の設計を想定していないためこのようなコンテキストの問題には対応しておらず、エンティティ Bean において関連エンティティを一度に取得する EAGER タイプに設定することで対処している。今後コンテキストの問題にも取り組む必要がある。

7 おわりに

人が見て直感的に理解できる ER 図と HTML ページから Java ソースコード、JSF ファイル、設定ファイルを自動生成する変換ツールを開発し、書籍に記載されたブックストアの例と同じものを使った生成実験を行った。その結果、機械的な記述を大量に用意したり、各部分の整合を取ったりするのは大変な作業を行うことなく、一つのリソースから生成可能なことが確認できた。

しかし、本ツールではまだ実現できていないことも多い。エンティティ Bean の継承指定やカスケードのアノテーション、JSF ページの自由な画面設計、マネージド Bean の設計などに、改善の余地がある。また、生成実験には単純なものを扱ったが、将来これらの単純な処理を組み合わせて複雑なワークフローが扱えるようにしたいと考えている。

参考文献

[1]斉藤賢哉,"マスタリング JavaEE5",翔泳社
[2](株)チェンジビジョン,"JUDE professional 5.2.1",http://jude.change-vision.com/
[3] Maximo Gurmendez, "jpa2web", http://www.ibm.com/developerworks/jp/web/library/wa-aj-jpa2web/index.html
[4]"AndroMDA", http://www.andromda.org/
[5]"JBoss Seam" http://www.jboss.com/products/seam