EJB3.0 を対象としたビジネスロジックジェネレータの試作

吉田 優樹†　　塚本 享治‡
東京工科大学　メディア学部　メディア学科

1. はじめに

情報システムの開発期間は短期化している。そのため、システムの開発効率を向上させるためにあらゆるツールや方法論が提案されてきた。

Java などのオブジェクト指向言語では特に「部品の再利用化」が推進化されている。そこで本稿では、MDA 技術[1]の一部として EJB3.0 技術を用いたエンタープライズアプリケーション開発を対象とし、ロジックのフローをモデリングするだけで、自動的にクラスの呼び出し手続きなどのロジックが生成できるコードジェネレータを試作したので報告する。

2. コードジェネレータの位置付け

何らかの機能をもった部品を、そのまま再利用し組み合わせることでシステム開発は効率的になる。しかし、実際はその部品を呼び出すための手続きなどの記述が必要となる。

そこで、UML モデルによるシステム設計の延長線上として、「部品」の呼び出し手続きなどを含めたロジックのフローを図で視覚化し、そこから描かれたモデル図に対応するビジネスロジックのソースコードを生成できるようなコードジェネレータを試作しようと考えた。

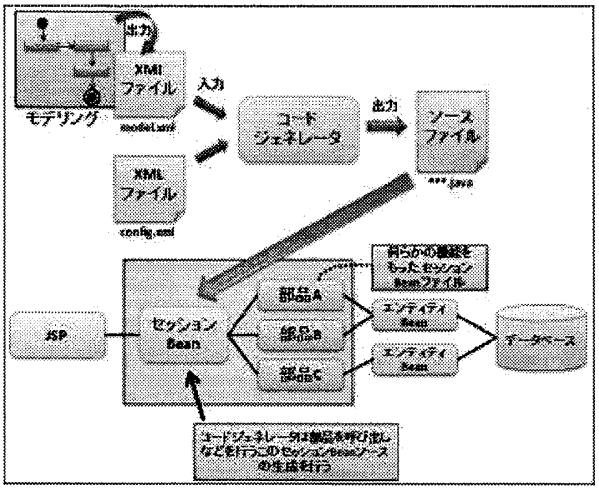

図2-1　コードジェネレータの基本構想と生成するコードの位置付け

EJB3.0 を利用したシステム開発は、セッション Bean に振舞いが記述されている。それら一つ一つのセッション Bean を部品として捉え、再利用すると考えたとき、部品を呼び出す記述が必要となる。その部分をコードジェネレータで生成した。コードジェネレータの基本構想と生成するコードの位置を図 2-1 に示す。

3. ビジネスロジックのコードジェネレータ開発

3.1. ロジックのフローのモデリング

ロジックの流れが理解しやすいだけでなく、生成されるコードがイメージしやすいことが、モデリングには求められる。

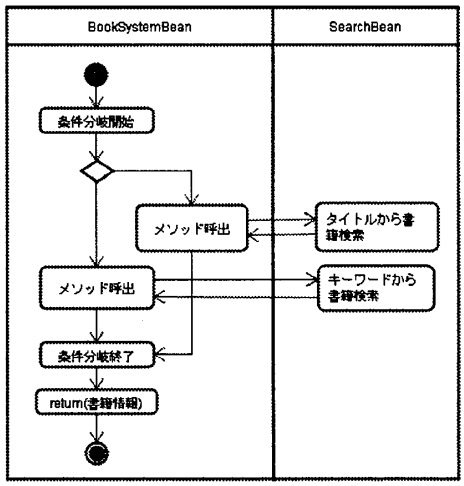

図 3-1　ロジックのフローのモデリング例

そこで、図 3-1 のような UML のアクティビティ図を基本としたモデリング方法がよいのではないかと考えた。図 3-1 で示したのは、コードジェネレータの開発にあたり題材として取りあげた研究室書籍管理システムの一部分である「書籍検索」の機能をフローに起こしたものである。

3.2. コードジェネレータの仕組み

図 2-1 でも示したように、コードジェネレータは二つのファイルを入力する。一つは model.xml というモデル情報が XML 形式で記述されている

Making of business logic generator for trial purposes intended for "EJB3.0"
†Yuki Yoshida, ‡Michiharu Tsukamoto
School of Media Science, Tokyo University of Technology

XMI ファイルであり、もう一つは Config.xml という XML ファイルであり、生成するソースファイルの import 文などの基本的な情報が記述されている。これら二つのファイルをコードジェネレータが解析し、ソースコードを生成する。また、モデルデータを XMI 形式で出力するために、モデリングツール JUDE Professional[2]を用いた。

3.3. コードジェネレータに依存したモデリング

コードをモデル図から生成するといっても、作図された図の情報だけでは難しいため、フローの図に使われているオブジェクト一つ一つが何の意味を持っているのかということを明確にする必要があった。そこで、モデルに対して「タグ付き値」というものを設定できる JUDE の機能を利用し、記述ルールを定めることでコード生成を実現した。

3.4. コードジェネレータの開発

XML 文書を解析するために、解析技術として SAX を利用した。最初に config.xml から生成するソースのクラス名やインターフェース名、import 文などを得る。そして細かいロジックの部分を、XMI ファイルである model.xml から生成する。作成されたロジックのフローのモデル図はオブジェクトとオブジェクトが矢印で繋がれており、一つの流れになっている。そのためオブジェクトを一つ一つ辿っていき、そのオブジェクトが何の意味を持つものなのかを解析して、ソースを生成していくことにした。

4. コードジェネレータの検証と考察

4.1. システムの試用実験

開発したコードジェネレータの完成度や利便性、を検証するために、このコードジェネレータを使って研究室書籍管理システムを対象として実際にコード生成を行った。

```
public List<Book>  goSearch(String select,
                   String title, String keyword) {
    List <Book> book = null;
    if(select.equals("title")) {
        book = search.titlesearch(title);
    }
    else {
        book = search.keywordsearch(keyword);
    }
    return (book);
}
```

図 4-1　実際にジェネレータで生成されたコード

結果から考察を行い、今後の課題を洗い出した。図 4-1 が図 3-1 にも示したモデル図から実際に生成したコードである。生成されたコードから、研究室書籍管理システムを実装することが出来た。

4.2. 考察

実験結果から、コードジェネレータは有効であると判断した。設計を行う感覚でロジックのフローをモデリングでき、ロジックのフロー自体も視覚的に理解しやすい。また、プレゼンテーション層は部品を意識せずに済むので、もし再利用する部品の細かい変更が生じても、モデリング部分を変更するだけで済むということは大きな利点といえる。

表 4-1　図 3-1 のモデリングに費やした時間

モデル描画のみ	3 分
タグの付与	12 分

しかし、開発したコードジェネレータの問題点もみつかった。ロジックのフローのモデリングを行う際、表 4-1 で示したようにモデルの描画だけなら非常に素早く行えるが、タグ付き値を付与するまでは少し時間が掛かってしまう。今回はタグを付与するという形でモデルを識別したが、もう少し工夫する余地があると考えられる。また、モデリング者がタグの書き方を覚えて、正確にモデリングしなければならないのも一つの問題点といえる。

5. おわりに

開発したコードジェネレータによって、ビジネスロジックにおける部品の再利用のフローを GUI で描くだけで、部品の呼び出しなどを行うセッション Bean のソースが生成できた。

今後は、複雑なビジネスフローも想定し、コードジェネレータの完成度を高める必要がある。より簡単に、モデリングミスが起こらないようにすることが大事だといえる。

参考文献

[1]. MDA
http://itpro.nikkeibp.co.jp/article/Keyword/20070823/280163/

[2]. JUDE
http://jude.change-vision.com/jude-web/index.html