GSLetterNeo Vol.58
2013年5月

リバース・モデリングのススメ

オブジェクトモデリングスペシャリスト
土屋　正人
Masato Tsuchiya
m-tsuchi@sra.co.jp

UML のようなモデリング言語で描かれたモデルから実装言語を生成することをフォワードエンジニアリング、実装言語からモデルを生成することをリバースエンジニアリングと称します。多くの UML モデリングツールは、これらの機能を備えていますが、フォワードエンジニアリングで生成されるコードはクラスのスケルトン止まり、リバースエンジニアリングで生成されるモデルはクラス図のみ、というものが多く、利用率は高くないように思います。

◆ フォワードエンジニアリング

フォワードエンジニアリングの理想形のひとつとして、OMG（Object Management Group）が 2001 年に提唱した MDA（Model Driven Archtecture）があります。これは、プラットフォームから独立したモデル（PIM：Platform Independent Model）を作成、モデル変換ツールを通して実行可能なモデルやコードを生成、という手順を踏むことで機能設計とアーキテクチャ設計を切り離すもので、技術が変化しても PIM がそのまま利用できることを意図しています。このアプローチをとる場合、変更を行う対象はモデルに限定し、コードは変更したモデルから再度生成することになります。**開発者は「コーディング」ではなく「モデリング」を行う**ことになるわけです。

機械語やアセンブラから、FORTRAN,COBOL,C 等、抽象度の高い言語の登場によりプログラミング技術は進歩を遂げました。抽象度を上げることにより、思考をハードウエアから切り離して、ロジックに集中することが可能になりました。UML も言語です。モデルでシステム化対象領域を抽象化することで、思考を実装から切り離し、情報や振る舞いに集中することが可能になります。とはいえ、モデルを変換することで実行可能モジュールを作成するアプローチには様々な問題があり、まだまだ普及には至っていないのが現実でしょう。

◆ リバースエンジニアリング

リバースエンジニアリングは、前述のようにツールのサポートがクラス図のみの場合が多く、モジュール構造は把握できても連携を把握することは難しいといわざるを得ません。納品ドキュメント作成のために仕方なく行うというのが現実ではないでしょうか。

「UML のモデルは開発の役に立たない」ということを耳にすることがありますが、モデルの目的が理解されていないことが多いのではないかと思います。目的を見失うと、「ドキュメント用に仕方なく作る」ことになり、モデルを作っても実装には使わない、ということが起こります。

モデルからコードという一方通行ではなく、**コードからモデルを生成してみる**ことで、コードの品質検証の視点が得られます。具体的なものを抽象化――単純化してみることで、コードのレベルでは見えなかった問題が見つかるかもしれません。また、コードとモデルの対応関係もより深く理解されるでしょう。

◆ リバースモデリング

そのために、手動でリバースエンジニアリングを行って、コードからシーケンス図を作ってみることをお薦めします。ここではリバースモデリングと称します。すべてのクラスのコードをリバースするのではなく、ユースケースのシナリオをコントロールするクラスのコードをリバースしてシーケンス図を作ってみるのが良いと思います。

例えば、注文を扱うユースケースの基本シナリオとして注文登録を考えます。次のコードをリバースモデリン

グした結果を図 1 に示します。

```
public class OrderController
{
  ....
  public void accepted(Customer customer,
                       Product product)
  {
    Order order = new Order();
    order.setDate(Date.today());
    order.setCustomer(customer);
    order.setProduct(product);
    OrderDAO dao = OrderDAO.getInstance();
    dao.save(order);
  }
  ....
}
```

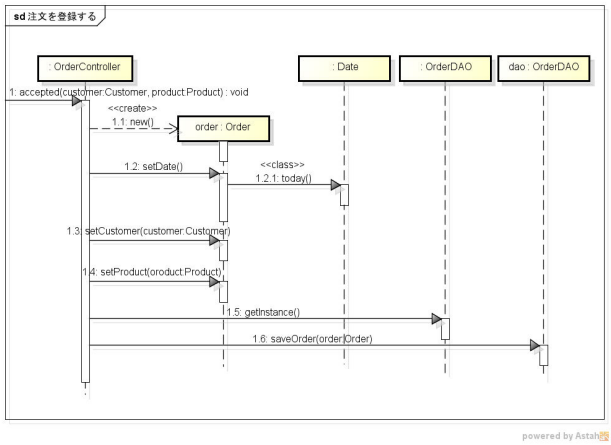

図 1　リバースしたシーケンス図

コードとモデルは 1 対 1 に対応していますが、OrderController が受ける"accepted"メッセージを処理するために、どのクラスやオブジェクトが必要か(用意する必要があるか)を視覚的に把握することが出来ます。

シーケンス図は横の流れを見るために使うことが多いかもしれませんが、ひとつのライフラインに着目すると、そのオブジェクトの仕事量を把握出来ます。**シーケンス図を縦方向に見る**わけです。OrderController に着目することで、"accepted"メッセージを受け取り、Order と OrderDAO へメッセージを送る様子が視覚化されます。OrderController と Order とのやり取りが煩雑なので、コンストラクタのパラメタとする選択を行うきっかけとなるかもしれません。そうすることで、知る必要のない詳細を隠蔽することが出来ます(図 2)。図 2 では、Order が"new"メッセージを受けた後のメッセージも図 1 との対比のために記述していますが、OrderController が知る必要のないメッセージであるため、不要になります。

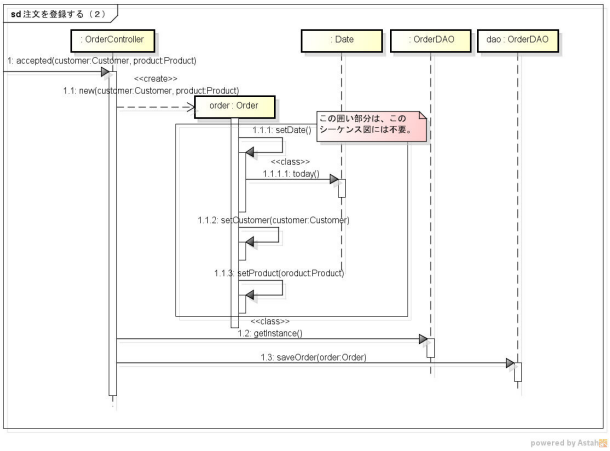

図 2　シーケンス図（改訂版）

シーケンス図は、横に長くなると可読性が著しく落ちます。横に長くなる原因のひとつとして、利用するコンポーネントやサブシステムの内部のメッセージも示そうとすることが挙げられます。コンポーネントやサブシステムをシーケンス図に登場させる場合は、それらのインタフェースをライフラインとして示し、シーケンス図にはインタフェースへのメッセージ送信までを明示します。そこから先のメッセージは、コンポーネントやサブシステムのシーケンス図を別に作ることで、図の肥大化を防ぐことができます。

極めて単純な例でしたが、作った**コードをシーケンス図で視覚化することで、コードの検証やコードとモデルの対応の理解を深め、モデルを活用する道が見えてくる**と思います。まずは自分のコードのリバースモデリングにトライしてみてはいかがでしょうか。

GSLetterNeo Vol.58
2013 年 5 月 20 日発行
発行者●株式会社 SRA 産業第 1 事業部
編集者●土屋正人、柳田雅子

バックナンバを公開しています●http://www.sra.co.jp/gsletter
ご感想・お問い合わせはこちらへお願いします●gsneo@sra.co.jp

株式会社SRA
〒171-8513 東京都豊島区南池袋２－３２－８