ObjectWorks+によるアプリケーション開発

■アプリケーション情報の定義シートを作成する(1)

ObjectWorks＋による開発の流れは、「定義シート作成」→「ソースコード自動生成」→「コーディング」→「単体テスト」と4つの作業に大別されます。まずは、アプリケーション開発支援ツール ObjectWorks＋/STUDIO(以下、STUDIO)での「定義シート作成」から、開発がスタートします。

この工程では、画面遷移設計やデータ項目の定義など、アプリケーションの基本設計の中でも、「内部設計」に相当する設計を行います。その前段には、前回解説したようにサイト構成図の「Conversation 分割」が完了し、STUDIO の定義シートの設計範囲が切り分けられていることが必要です。

STUDIO では、「画面遷移定義」「データ定義」「アクション定義」「サービス定義」と、Excel をベースとした 4 つの定義シートを準備します(図 1)。

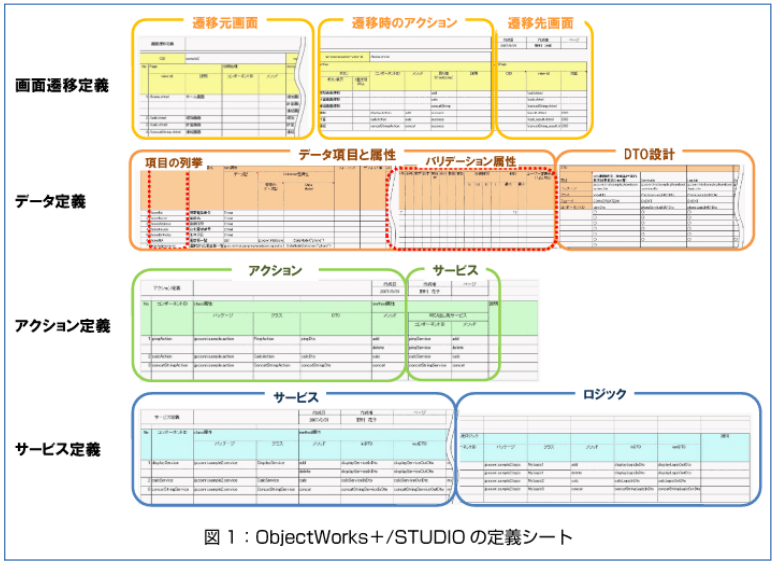

図 1：ObjectWorks＋/STUDIO の定義シート

★画面遷移定義

まず始めに、「画面遷移定義シート」を作成します。画面遷移定義シートは、大きく分けると「遷移元画面」「遷移時のアクション」「遷移先の画面」の 3 つの領域があります。アプリケーション内(Conversation 内)のすべての画面について、その中で発生するアクションを書き出し、その処理結果に応じて遷移先の画面を定義します。

★データ定義

次に、「データ定義シート」を作成します。データ定義シートは、「データ項目と属性の定義部」と「DTO(Data Transfer Object)設計部」の 2 つの領域に分かれます。まず、シートの左端に、そのアプリケーション内に登場するすべてのデータ項目を列挙します。

データ項目列挙後、それぞれについて属性情報を記入していきます。属性情報としては、データ型や日本語名称、デフォルト値などのほかに、バリデーション属性があります。ここで「必須」や「全角」といった欄にチェックを入れておくことで、後ほど自動生成される DTO にこれらのバリデーションを実装するアノテーションが付与されます。

続いて、シートの右端に、任意のデータ項目をフィールドとして持つ DTO を設計します。DTO をどの程度細かく分割して定義するかは、プロジェクトの標準化ルールによります。画面(XHTML)とアクション・クラスとの間でデータのやり取りに用いる「Web 層 DTO」を、サービス／ロジック・クラスの入出力にもそのまま用いるように単純化するケースもあれば、サービス／ロジックの種類に応じて最低限必要なデータ項目だけを持たせた入力 DTO、出力 DTO を用意する場合もあります。DTO は、後述する「アクション定義シート」、「サービス定義シート」とも同時並行的に設計していくことになります。

■アプリケーション情報の定義シートを作成する(2)

★アクション定義

次に、「アクション定義シート」を作成します。アクション定義シートは、「画面遷移定義シート」で記述した、遷移時に呼び出されるアクションを設計します。画面遷移定義シートで記述したアクションのコンポーネント ID(＝アクション ID)に対して、そのアクションが呼び出すサービスのコンポーネント ID(＝サービス ID)を記述していきます。

また、アクション・クラスと画面(XHTML)との間でデータをやり取りするための DTO を指定します。ここには、先ほどの「データ定義シート」で設計した DTO の名前を指定します。

★サービス定義

最後に、「サービス定義シート」を作成します。「アクション定義シート」で記述したサービス ID に対して、そのサービスが呼び出すロジック(ビジネス・ロジックの実装部の本体)を記述していきます。

また、サービス・クラスの入出力に用いる DTO およびロジック・クラスの入出力に用いる DTO も、ここに定義します。この入出力 DTO を、サービスやロジックごとに適切に設計することで、サービスやロジックといった BL(ビジネス・ロジック)層のクラスが PL(プレゼンテーション・ロジック)層の画面やアクションと切り離されて管理しやすくなります。

以上で、最初のステップとなる「定義シート作成」の工程は完了です。STUDIO の定義シートは、アプリケーションのクラス構造に、理解しやすい単純なアーキテクチャーを導入して標準化する役割を果たしていますが、役割はそれだけではありません。

アプリケーションの設計情報を、これらのスプレッド・シート上に一元化することで、アプリケーション全体の構造を俯瞰したり、サービスの入出力のようなインターフェースが、個々の開発者によって変更されないよう管理を集約するなどの役割も担っています。

■定義シートからソースコードを自動生成する

STUDIO の定義シートが完成したら、定義シートで設計したアプリケーション・ソースコードの自動生成を行います。自動生成は、STUDIO に付属する Ant スクリプトで、Eclipse などの統合開発環境やコマンド・プロンプトなどから実行できます。各定義シートから自動生成されるファイルの内訳や関係は、図 2 のようになります。

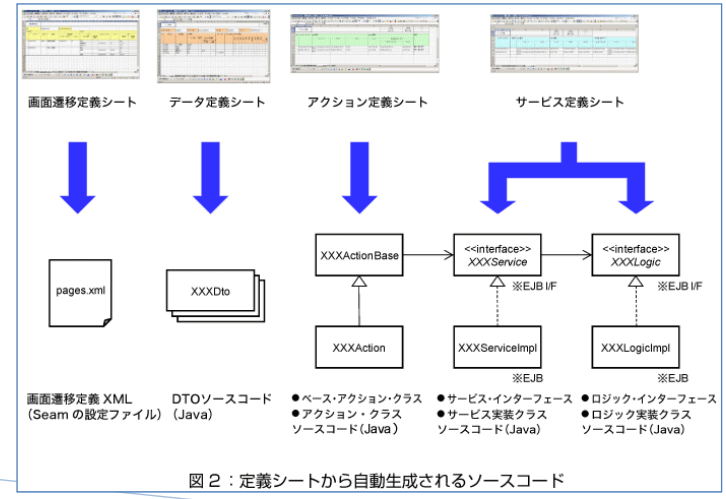

図 2：定義シートから自動生成されるソースコード

ここで、それぞれ自動生成されるファイルについて解説します。

★pages.xml

画面遷移定義シートの情報は、すべてこの XML ファイルに出力されます。これは、JBoss Seam(以下、Seam)が Java EE 5 標準の MVC フレームワーク「JSF」をラップして画面遷移を制御するためのファイルです。pages.xml は生成後は一切編集せず、アプリケーション内の所定の位置に配置することで動作します。

★DTO ソースコード

データ定義シートに記述したデータ項目をフィールドに持ち、setter/getter メソッドのみを持ったシンプルな Java ソースコードが生成されます。データのバリデーション属性定義部の記述に対応して、バリデーション用のアノテーションがフィールドに付与され、Java クラス自体がメタ情報としてバリデーション属性を持っています。

★アクション層ソースコード

アクション層では、ベース・アクション・クラスとアクション・クラスの 2 種類が生成されます。ベース・アクション・クラスには、アクション定義シートに定義したサービスの呼び出しが実装されています。このソースコードには、一切手を入れないことが標準化ルールとして推奨されています。

アクション・クラスは、ベース・アクション・クラスを継承し、何も記述されていない空クラスです。ビジネス・ロジックに依存しない何らかの共通処理を実装したい場合はこちらに任意のコーディングを施すことができます。実際に、画面(XHTML)からはこのアクション・クラスが呼び出されて、親クラスで定義されている通りにサービス呼び出しが実行されます。

★サービス層／ロジック層ソースコード

サービス層とロジック層では、「サービス定義シート」に記述したサービス／ロジックが、インターフェースと実装クラスの 2 つ 1 組のソースコードとして出力されます。

インターフェースのソースコードには、EJB のビジネス・インターフェースであることを示すアノテーション(@Local など)が付与されています。

また、実装クラスのソースコードには、EJB の実装クラスであることを示すアノテーション(@Stateless など)と、Seam の管理する Seam コンポーネントであることを示す@Name アノテーションが付与されています。@Name アノテーションで指定する Seam コンポーネント ID には、定義シートに記述したサービス／ロジック ID が利用されます。

このように、STUDIO の定義シートから生成されるビジネス・ロジックのクラスは、自動的に Seam（とその裏側にある EJB3）上で管理されるシンプルな Java クラスとなっています。

■自動生成したソースコードと定義シートの整合性を保つ

設計シートからアプリケーションのソースコードを自動生成する設計／開発方式では、設計シートとソースコードの 2 カ所にアプリケーションの定義情報が併存することになります。こうした場合、一般にはこれらの整合性をいかに維持するかが課題となります。

STUDIO では、アプリケーションの定義情報を常に定義シート側で一元管理することを標準化ルールとしています。これは、マルチベンダーで分担して複数のサブシステムを開発する場合や、定義シートによる設計から先のコーディングをオフショアで実施する場合などにも、アーキテクチャーの標準化などの統制を利かせ、開発をコントロールできるようにするためです。

STUDIO の定義シートで記述している内容に変更がある場合は、手元で Java ソースコードや XML ファイルを変更せず、必ず定義シートを修正してファイルを再度自動生成することを推奨しています。

画面遷移定義やデータ定義については、生成されたファイルに手を加える必要がないため、一元化された定義シート上で変更を管理することに違和感はないはずです。ただ、アクション定義やサービス定義から生成される Java クラスのソースコード類は、（ほとんどの場合ロジック・クラスのみですが）生成後にコーディングをすることで自動生成時の内容から変更が加わっています。この部分についてはどのように定義情報を管理しているのでしょうか。

実は、これまでの説明からも分かる通り、STUDIO の定義シートではビジネス・ロジックの本体については管理をしていません。Java クラスのインターフェース（コンポーネントのインターフェース名と、入出力に用いる DTO）や、クラス間の呼び出し関係だけを定義しています。

これは、インターフェースや呼び出し関係といった設計情報は、必ずスプレッド・シート形式の定義シートで管理をするようにすることで、複数人での開発時に場当たり的な変更による混乱を避け、アーキテクチャーの統制をとりやすくできるためです。一方、ビジネス・ロジック本体の詳細な仕様については、個々のアプリケーション実装クラスの中で管理したほうが、生産性も阻害されず適しているからです。

このように、STUDIO では Java クラスのインターフェース、呼び出し関係の仕様は、定義シート上で管理することでメンテナンス性を高く保つ標準化ルールを定めています。しかし、ソースコードを生成してその上にコーディングをする以上、個々のアプリケーション開発者によって標準化ルール外の実装が施される可能性は残されています。

そこで STUDIO では、定義シートで定める Java クラスのインターフェース呼び出し関係と、Java アプリケーションのソースコード（の現状）とを比較し、相違点の有無をチェックするツール（図 3）を用いることで、設計の整合性を保っています。

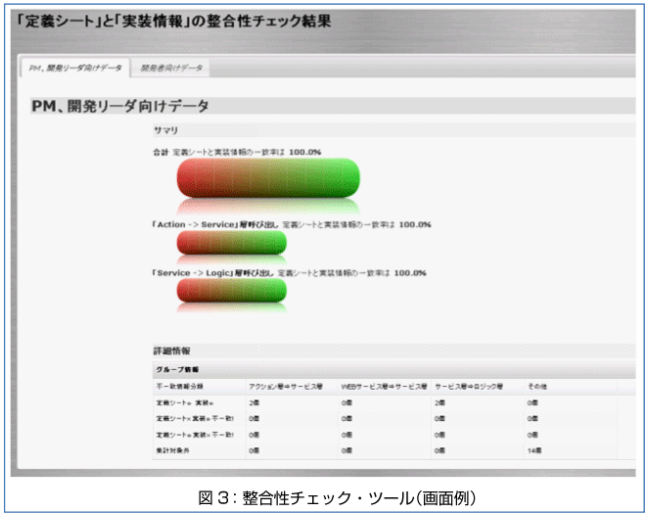

図 3：整合性チェック・ツール（画面例）

■アプリケーションのコーディング

定義シートからアプリケーションのソースコードを自動生成したら、業務要件に従ってコーディングを行います。開発者が直接コーディングを行う主な開発対象物としては、以下のものがあります。

・ロジック・クラス（Java）
・画面（XHTML、JSP）

ロジック・クラスは、前ページで解説したように EJB 兼 Seam コンポーネントとして自動生成され、業務処理本体を実装するメソッドの「ガワ」だけが記述されているスケルトン・ソースコードの状態となっています。あとは、そこにメソッドの引数で与えられた入力

DTO からデータを取得し、ビジネス・ロジックの処理を実施し、出力 DTO に詰めて return する処理を記述するのみです。

データ・アクセス処理は必要に応じて DAO（Data Access Object）クラスとして独立させ、Java EE 標準 O/R マッピング・ツールの JPA の実装（Hibernate など）を用いてコーディングします。この部分には、ObjectWorks＋特有の制約などはあまりなく、標準的なツールの仕様に従って開発ができます。

ここでは、画面開発に関して、ObjectWorks＋で特長的な点を解説します。

Seam では、JSF 上のテンプレート・エンジンである「Facelets」を内部で採用しており、動的な画面ファイルを JSP でなく XHTML で開発することができます（この Facelets も、Java EE 6 に含まれる JSF2.0 に取り込まれて標準規格となりました）。Facelets を使った画面開発には下記のような特長があります。

・理解しやすい XHTML で画面を開発可能
・テンプレート機能により画面の部品化／再利用が容易

シンプルな XHTML として画面開発ができるので、オープンソース（無償）で入手できる統合開発環境プラグイン（図 4）でも、プレビューを見ながら画面部品（タグ）のドラッグ・アンド・ドロップによる開発を行うことが可能です。

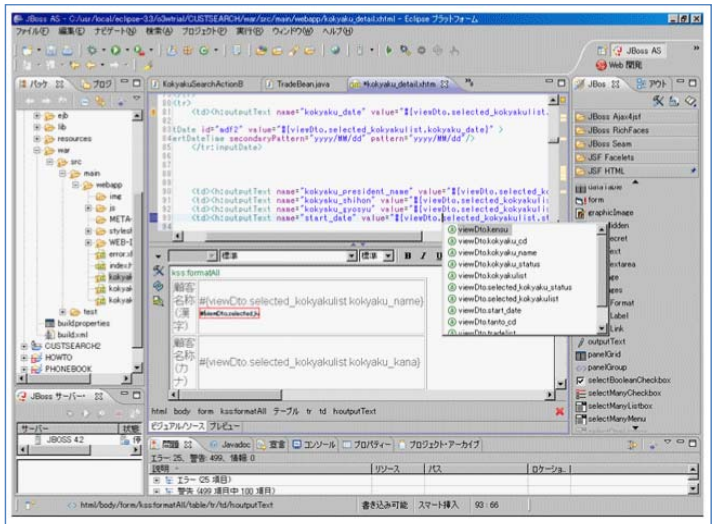

図4：Seam を使った開発に適した「JBoss Tools」（画面例）

ObjectWorks＋による画面開発の例としては、前回解説した ObjectWorks＋/CORE「アプリケーション共通機能」のうち、値入力時のバリデーション実装方法を解説します。

ObjectWorks＋では自動生成された DTO にバリデーション属性のアノテーションが付与され、データ項目ごとにバリデーション属性の情報が一元管理できるようになっています。

このバリデーションを画面で実行するには、XHTML ファイルのソースコード中で、バリデーションを実施したいデータ入力フィールドなどの HTML 部品タグの範囲を、ObjectWorks＋の「バリデーション起動用タグ」で囲むだけです。

```
------------------------------------------------------------------------
…
<kss:validateAll onblur="true"> # ← ObjectWorks＋のバリデータを起動するカスタム・タグ
  <input type="text" jsfc="h:inputText" name="date" value="#{xxxDto.date}" />
</kss:validateAll>
…
※xxxDto の持つ"date"というデータ項目に対して、アノテーションで設定されているバリデーションが実行される。
------------------------------------------------------------------------
```

これで、カスタム・タグに囲まれた範囲では、DTO に付与されたアノテーションに従ってバリデーションが実施されるようになります。また、このカスタム・タグはクライアント・サイドの JavaScript によるバリデーション部品とも連携しており、タグの属性値を変更するだけでクライアント・サイド／サーバー・サイドのバリデーションの有無を制御することができます。

■支援ツールを用いた単体テスト

ObjectWorks＋では、開発した画面（XHTML）、サービス／ロジック／DAO（Java クラス）を単体テストする際に利用できる、「単体テスト支援ツール」を提供しています。これは、「BL（ビジネス・ロジック）単体テスト支援機能」と「PL（プレゼンテーション・ロジック）単体テスト支援機能」の、大きく 2 つに分かれます。

★BL 単体テスト支援機能

サービスとロジックなど、複数のクラスを結合した状態で単体テスト観点のテストを行うためには、オブジェクトを開発者（テスト開発者）が組み立てる必要があります。また、

データベース・アクセスを行うプログラムのテストの場合、データベースの初期化などの手間がかかります。

BL 単体テスト支援機能では、上記を解決するため、Excel から読み込んだテスト・データからの DTO などの作成や、モックの差し込みの制御、Excel から読み込んだデータベースのテーブル情報による初期化などの機能を提供しています(図5)。

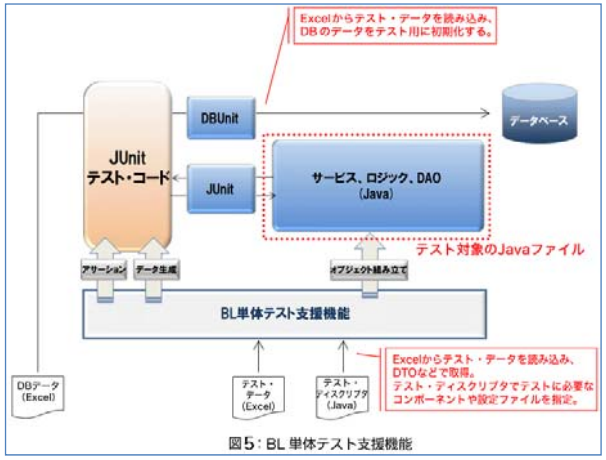

図5：BL 単体テスト支援機能

★PL 単体テスト支援機能

画面ファイル(XHTML)ごとに、周辺の DTO などと組み合わせて単体テスト観点のテストを行う際に利用します。まだ BL 層のクラス(サービス・クラスなど)が作成されていない段階でも、BL 層の呼び出し処理を横取りして疑似的な結果を返すモック・インターセプタによって、画面ファイル単体の動作確認を可能にします。

モック・インターセプタは、あらかじめ用意しておいた Excel シートから、シナリオに応じた返却データを DTO にして画面へ返す機能を提供しています(図6)。

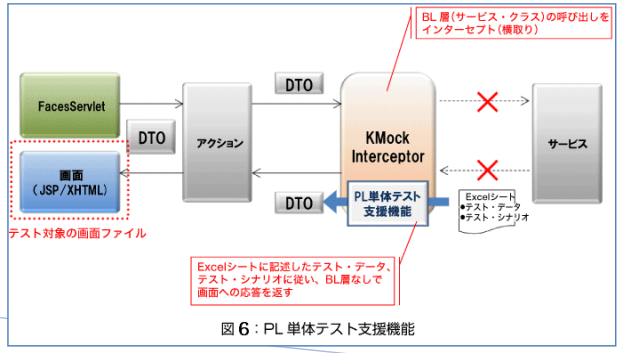

図6：PL 単体テスト支援機能

単体テストの実施範囲や粒度は、プロジェクトによってまちまちです。始めから PL や BL のファイルを結合したテストを行うケースもあれば、まずは BL を単体テストして、順次結合しながらテストを繰り返していくケースなどもあります。いずれにせよ、上記のようなツール類によってテスト・コードの作成やデータ投入といった作業を簡単化し、テストの生産性を高めることができます。

■これからの SI フレームワークに求められること

Java の世界では技術の選択肢が多様であり、自社の汎用的な開発標準、というものを定めるのが困難でしたが、近年の Java EE 拡張の動きによって「使える標準規格」が整ってきています。

これからの SI フレームワークでは、こうしたオープンで標準化された安定的な技術をベースとしているかどうか、ということもポイントの 1 つとなってきます。ベースに標準的な技術基盤を利用しつつ、拡張機能やアーキテクチャー標準化などの仕組みをいかに標準仕様にのっとった形で提供されているかが、その SI フレームワークで開発したアプリケーションの安定性や保守性に大きく影響してきます。

今後、フレームワークの進化を追っていく際には、こうした標準化の観点からも技術動向を見極めていきましょう。

【執筆者】 深津 康行
株式会社野村総合研究所(NRI) 基盤ソリューション事業本部 副主任システムコンサルタント。入社以来、Java/Web 系フレームワークの開発・導入支援や、開発標準化コンサルティングなどに従事。現在は NRI の SI フレームワーク「ObjectWorks＋」を中心に、開発基盤ソリューションの企画と顧客への提案を行っている。

※本文章は、2010 年 1 月[Think IT]に掲載されたものを再編集しています。