視覚化されたＪＳＰカスタム・タグに対する編集機能の拡張

堀内　芳雄　　武田　成史
日本アイ・ビー・エム株式会社　ソフトウェア開発研究所

1. はじめに

JSP カスタム・タグ[1]（以下，単にカスタム・タグと呼ぶ）は動的な Web ページの作成に欠かせないものであり，編集時におけるその取り扱いは，Web ページ・オーサリングツール（以下，単にオーサリングツールと呼ぶ）の使いやすさを向上する上で，重要な課題のひとつである．

オーサリングツールの一例である WebSphere® Studio の Page Designer は，VCT(Visual Custom Tag)[2]と呼ばれるカスタム・タグを HTML タグの組み合わせに置き換えて視覚化する枠組みを備えており，充実した表示環境を提供する．VCT の初期におけるデザインはカスタム・タグの視覚化を目指したもので，描画された領域は読み取り専用であったが，現在は次の段階として，視覚化されたタグの直接的な編集の実現を目指している．

今回，VCT の一部の編集機能，描画サイズの変更など，を拡張することが求められた．これに対し，筆者らはそれぞれの編集機能を，機能単位で置き換えることを可能とするフレームワークを定義することで対応した．本論文では，編集機能を拡張するフレームワークの実装とその有効性について述べる．

2. VCT とは

2.1 VCT によるカスタム・タグの視覚化

従来のオーサリングツールでは，カスタム・タグは単にアイコンで表示する，あるいはアプリケーション・サーバーを実行してその出力結果を編集時にも表示する，などの方法をとる．一方，VCT は，実行時と同様の表示はもちろん，オーサリングに特化した情報を付加して表示することも可能であり，個々のケースに応じて柔軟な表現ができる点で有用性が高い．（図１）

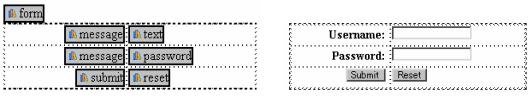

図１，カスタム・タグの表示例（左：アイコンによる表示，右：VCT による視覚化）

VCT は Visualizer と呼ばれる，タグ・ハンドラーを模した Java コードが関連付けられたカスタム・タグであり，処理手順は図２に示される． まず JSP が読み込まれ DOM(Document Object Model)[3]ツリーが作られる（①）．次にカスタム・タグに関連付けられた Visualizer が呼び出され（②），その出力で DOM ツリーを書き換えて（③）視覚化される．

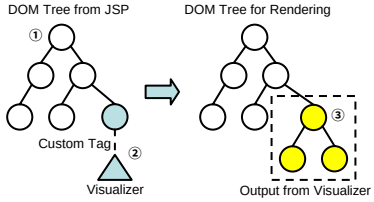

図２，視覚化の手順

2.2 直接編集の要求とその問題点

VCT はオーサリングツール上で，通常の HTML タグの組み合わせとして描画されるため，直接マウスでその大きさを変更したり，内部の配置を変えたりすることは，極めて自然な要求である．

しかし，VCT により視覚化されたタグの直接編集は困難であった．例えば描画領域のサイズを変更する場合，カスタム・タグの描画サイズを決定している属性がどれであるか既知でなければならない．オーサリングツール自体にカスタム・タグの編集機能を持たせるには，個々のカスタム・タグの詳細な定義をもとに機能を作りこむ必要があるが，追加，拡張が容易なカスタム・タグすべてに対応しようとすることは現実的ではない．

3. 編集機能拡張のフレームワーク

3.1 編集機能拡張の基本方針

VCT の設計者は Visualizer を実装できる程度にはカスタム・タグの定義に精通している．つまり，VCT の設計者が編集機能を提供することは可能である．ところで本論文で述べるオーサリングツールは，Eclipse[4]を利用するもので，編集機能の多くは Eclipse の枠組みである GEF(Graphical Editing Framework)[5]のコマンドによって実現され，機能とコマンドは比較的わかりやすく対応付けられている．コマンドの実行の流れは図３に示される．ユーザが画面を操作すると（①），対応するコマンドが生成され（②）DOM が編集される（③）．

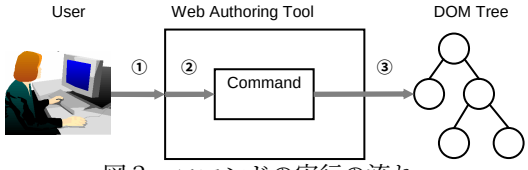

図３，コマンドの実行の流れ

これらの点を考慮し，筆者らは，外部から編集用のコマンドを Visualizer と同様の方法で置き換えてオーサリングツールに実行させることができるように拡張することにした．

An Editing Extension Framework for Visualized JSP Custom Tags
Yoshio HORIUCHI, Shigefumi TAKEDA,
Software Development Laboratory – Yamato (YSL)
IBM Japan, Ltd.

3.2 エクステンション・ポイント

すべての VCT が独自の編集機能を持つ必要はない．そこで Visualizer とは別に Eclipse のエクステンション・ポイント”designActionProvider”を定義し，既存の Visualizer に変更を加えずに済むようにした．エクステンション・ポイントを拡張する際には後述の CommandProvider オブジェクトを指定する．

3.3 CommandProvider インターフェース

CommandProvider インターフェースを以下のように定義する．

```
public interface CommandProvider {
  Command getCommand(String action, Request request);
  EditHint getEditHint(String type, Node realNode, Node node);
}
```

ここで getCommand()はコマンドを，getEditHint()は編集時の補足情報を返すメソッドである．

getCommand()の引数 action は置き換える対象となる機能を識別するための文字列であり，有効な値は機能拡張にあたって個別に定義される．たとえば描画領域のサイズ変更コマンドは”resize”であらわされるが，新たな値を定義することで，他の機能の拡張に対応できる．引数 request には機能別に定義された，コマンドを実行するために必要な情報が渡される．

オーサリングツールは起動時にエクステンション・ポイントを拡張するプラグインを調べて command タグを列挙し，必要に応じて CommandProvider を実装するクラスを初期化する．

3.4 描画サイズを変更するコマンドの拡張例

表示されるオブジェクトの単位は GEF で定義される EditPart であり，EditPart を選択するとその周囲にはトラッカー（サイズ変更のための操作ポイント）が配置される．視覚化されたカスタム・タグの描画サイズ変更を可能にするために，まず，適切なトラッカー情報をオーサリングツールに与えなければならない．CommandProvider は以下に定義される TrackerHint インターフェースを実装するオブジェクトを，引数 type が”tracker”で指定される getEditHint()メソッド経由で返せばよい．

```
public interface TrackerHint extends EditHint {
  int getHandleHint(); // サイズ変更方向の情報
  ...
}
```

getCommand()の引数 action には”resize”が，また，引数 request には以下のオブジェクトが渡される．

```
public interface ResizeRequest extends Request {
  Rectangle getBounds(); // サイズ変更後の大きさ
  Rectangle getOriginalBounds(); // 元の大きさ
  Element getElement(); // 表示上のエレメント
  Element getRealElement(); // DOM 上のエレメント
  ...
}
```

CommandProvider は action と request を吟味して，コマンドを用意できる場合には getCommand()メソッド経由で返し，そうでない場合は null を返せばよい．処理手順は図 4 に示される．ユーザの操作（①）に対し，CommandProvider からコマンド取得を試み（②），有効なコマンドが取得された場合には（③）それを使って DOM を編集する（④）．

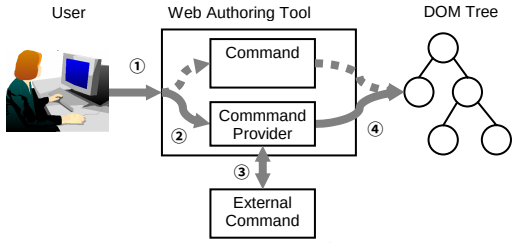

図 4，拡張コマンドの実行の流れ

3.5 フレームワークの効果と注意点

オーサリングツールを修正することなく，外部プラグインからカスタム・タグの編集機能を追加することができるようになった．機能の追加や置き換えは，描画サイズの変更などの個々の編集機能単位で可能であり，必要な機能のみを置き換え，その他の機能はオーサリングツールの基本編集機能に任せることができるため，機能を提供する側の負担を減らすことができる．

注意点として，編集機能の追加，置き換えの実装においては，パフォーマンスを考慮する必要があることがあげられる．CommandProvider のメソッドは編集中頻繁にアクセスされるため，優先度の高い，振る舞いの悪い CommandProvider が一つ存在するだけで全体のパフォーマンスが低下してしまう．現状ではフレームワーク側でこの問題を解決することは難しいため，拡張コマンドのコーディングガイドを出すなど，適切な情報を開示し，注意の呼びかけをすることが必要である．

4. まとめ

VCT の直接編集機能のサポート要求に対し，汎用な機能拡張のフレームワークを定義することによって対応した．要求を満たすとともに，今後の拡張を容易にする，有効な手法である．現時点では描画サイズの変更などのサポートのみであるが，他の機能の置き換えにも対応する予定である．

コマンドの中には非常に多くの作業を行っているものもあり，単純な置き換えでは，機能を拡張しようとする側に多大な労力を強いる場合がある．また，既存のコマンドに作業を追加するといった使い方も予想される．今後は，これらの点に留意し，コマンドより細かい単位での機能拡張について考慮したい．

参考文献

[1] Java Server Pages Technology (http://java.sun.com/products/jsp/index.jsp)

[2] Kaoru Hsokawa, Programming WebSphere Studio Page Designer Visual Custom Tags IBM developerWorks, (http://www-136.ibm.com/developerworks/)

[3] W3C Document Object Model (http://www.w3.org/DOM/)

[4] eclipse.org (http://www.eclipse.org)

[5] Graphical Editing Framework (http://www.eclipse.org/gef/)