ラウンドトリップエンジニアリングを目指したWebアプリケーションのための意味モデル

松塚貴英[†] 阿草清滋[††] 山本晋一郎[†††]

今日，Webアプリケーションを実現する技術は急速に向上しているが，意味を記述する枠組みが不十分であるため，品質の確保が難しい．本論文では，Webアプリケーションの意味を記述するためのモデルであるWebWare意味モデルを提案する．このモデルの特徴は，特定の実装技術に依存しないことと，ラウンドトリップエンジニアリングを目指していることである．これにより，現在広く存在するWebアプリケーションの意味モデルを導出することができ，既存Webアプリケーションの継続する保守作業において品質の向上に貢献することが可能となる．その実効性を示すため，解析器を試作し，実際に既存のアプリケーションからWebWareモデルが導出できることを示す．さらに，導出により得られたWebWareモデルから再びWebアプリケーションが生成されラウンドトリップエンジニアリングが可能になることを示す．なお，産学共同研究プロジェクトの利点を生かし，実アプリケーションによる問題点のフィードバックを行ったり，早期のラウンドトリップエンジニアリングの実現を行ったりすることができた．

A Semantic Model for Round-trip Engineering of Web Applications

Takahide Matsutsuka,[†] Kiyoshi Agusa[††]
and Shinichiro Yamamoto[†††]

The technologies that construct Web applications are improving rapidly. In this situation, lack of frameworks that describe semantics of the Web applications makes difficult to improve the quality of the Web applications. The present paper introduces a semantic model, which describes semantics of the Web applications. The strengths of the model are two: 1) independent of specific implementation technologies, and 2) can be applied to round-trip engineering. These strengths improve the quality of the Web applications at a maintenance phase. To provide an example evidence for such improvement, an analyzer was developed, which then was applied to an existing Web application. Furthermore, we realized the round-trip engineering by re-generating the web application in close collaboration with Nagoya University and Fujitsu Laboratories.

1. はじめに

1.1 背景

今日，インターネット上ではさまざまなWeb技術が使われている．我々の研究は，Webアプリケーションなど，Web技術を含んだソフトウェアをWebWareと定義し，WebWareをサポートする開発環境WebWareワークベンチの構築を目的としている．

† 株式会社富士通研究所
Fujitsu Laboratories Limited
†† 名古屋大学大学院情報科学研究科
Department of Information Engineering, Graduate School of Information Science, Nagoya University
††† 愛知県立大学情報科学部情報システム学科
Department of Information Science and Technology, Aichi Prefectural University

現在，Webアプリケーションはインターネット上で実現される大規模でヘテロジニアスな分散ソフトウェアであり，Webアプリケーションに利用されるWeb技術は

(1) XMLやHTMLなどのコンテンツ記述
(2) レンダリングのためのCSSやXSLT
(3) JSPやServletなどのサーバ側プログラム
(4) JavaScriptに代表されるクライアント側プログラム

などのさまざまな技術から構成される．これらの個々の要素技術が持つ機能は急速に向上しているが，全体としての意味を記述する枠組みが欠けており，個々の要素がどのような制約を満たすべきかは表現されない．そのため，Webアプリケーションの信頼性の低下につながっている．

1.2 目　　的

Web アプリケーションの多様で複雑な意味を記述し，構成要素間の制約を管理するために，モデルをベースとして Web アプリケーションを開発することによる有効性が確認されつつある[1)]．

本論文では，WebWare 意味モデルを Web アプリケーションの意味を記述し，モデルベースの Web アプリケーション開発環境を提供するために必要なモデルと位置付け，この WebWare 意味モデルを定義する．このモデルは，特定の実装技術に依存しないことと，既存アプリケーションからの導出を考慮することにより，ラウンドトリップエンジニアリングを目指している．このことの妥当性を確認するために，解析器を試作し，ラウンドトリップエンジニアリングの検証を行う．

本論文の構成は次のとおりである．2 章では，本論文の目標であるラウンドトリップエンジニアリングについて議論する．3 章では，WebWare 意味モデルを検討する．4 章では，今回試作した解析器により動作を検証するほか，ラウンドトリップエンジニアリングの検証も行う．また，産学連携についての効果と関連研究について議論する．最後に 5 章ではまとめを行う．

2. ラウンドトリップエンジニアリング

2.1 要　　件

すでに多くの Web アプリケーションが稼動している現在，ラウンドトリップエンジニアリングを確立する手法が強く求められている．本論文で述べるラウンドトリップエンジニアリングは，既存の資産（ソースコードなど）からリバースエンジニアリングによりアプリケーションの意味モデルを抽出し，必要であればモデルを改変し，再びソースコードにすることである．そのために，次のような要件を設定する．

1)　画面遷移を表現する

WebWare モデルは，Web アプリケーションの画面間の遷移が表せるものである必要がある．これにより，アプリケーションの動作を直感的につかむことが可能となる．

そのため，ページとページ，ページとサーバプログラム間の関連を有向グラフとして明示する．

2)　特定の実装技術に依存しない

Webware 意味モデルは JSP や Struts といった特定の実装技術に非依存なものとする．そのためには，モデルはなるべく簡潔なものである必要がある．これにより，今後出現するものも含む各種の技術に適用可能なものとする．また，ラウンドトリップエンジニアリングでアプリケーションを異なるプラットフォームへ移行することが可能となる．

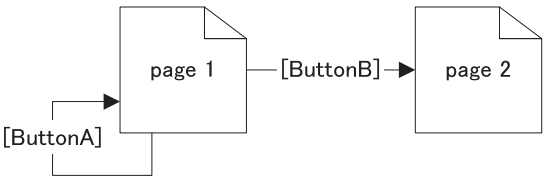

図 1　Model1 アプローチのモデル例
Fig. 1　A model example of Model-1 approach.

3)　リバースエンジニアリングによるモデル導出を可能とする

モデルは，既存のアプリケーションを解析することによっても得られるものである必要がある．前述のように Web アプリケーションは多様な技術を使っているため，その解析にも複数のプログラミング言語の解析や，それを組み合わせた整合性の検査などが必要となり，非常に複雑である．そのためには，なるべく意味モデルの抽象度を高くし，個別の技術に依存しない枠組みを作成する必要がある．

実際に作成した意味モデルのインスタンスが既存のアプリケーションから抽出可能であるかどうかは 4.1 節の試作により検証を行う．

2.2 Web アプリケーションの実装モデル

Web アプリケーションは，大別してページ中心型，MVC 型，そしてフレームワークに基づく 3 種類のアプローチで実装が行われる．本節では，Web アプリケーションの実装モデル別に，WebWare 意味モデルに対する要件を検討する．

1)　ページ中心型

ページ中心型の Web アプリケーションは Model1 アプローチとも呼ばれ，ページが直接クライアントからのリクエストを受け，ページに結び付けられたビジネスロジックを呼び出すことによって処理が行われる．

このとき，画面遷移は表示画面からブラウザ側で指示されたものになるのが普通である．つまり，ページで「次の」画面にリクエストを行う Form（または A タグ）が表示され，ユーザがリンクをクリックすることで次のページへとリクエストが送られる．そのため，ページ内に次のページへの遷移情報が直接記述される（**図 1**）．

2)　MVC 型

MVC 型の Web アプリケーションアーキテクチャは Model2 アプローチとも呼ばれ，ブラウザからのリクエストを単一のフロントコントローラが受け取り，実際のビジネスロジックはフロントコントローラから分

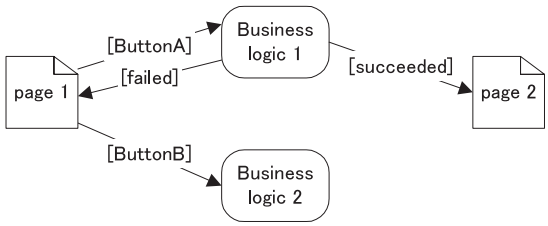

図 2 Model2 アプローチのモデル例
Fig. 2 A model example of Model-2 approach.

岐される．

図 2 は，page 1 より “ButtonA” を押して開始されたビジネスロジックの処理結果が “succeeded” か “failed” かに応じて，次に表示されるページが page 1 になるか page 2 になるか決まる例である．

このように，Model2 では，ビジネスロジックの結果に応じて動的に表示されるページが切り替わるため，ページのみではモデルを構築することができない．そのため，ビジネスロジックも含めたモデル化が必要である．

3) フレームワーク

Model2 アーキテクチャに基づく Web アプリケーションを構築する際に難しいのは，フロントコントローラの実装と，リクエストをどのようにビジネスロジックにディスパッチするか，処理結果をどのようにページに振り分けるかである．このことを効率的に構築するための仕組みは，フレームワークとしていくつか提案されている．

Web アプリケーションフレームワークでは，ビジネスロジックとページの対応付けを外部のマッピング情報で管理することにより，両者が切り離され，メンテナンス性の高いアプリケーションを構築することができる．たとえば Struts[2] では struts-config.xml というファイル，JSF[3] では faces-config.xml というファイル，UJI[4]（以降では Apcoordinator☆と表記）ではコマンドマップというファイルを使ってマッピング情報を管理している．

マッピング情報はページ間，またはページ，ビジネスロジック間の関連を表すための情報で，モデルを実装に変換したり，逆にリバースエンジニアリングの際に実装からモデルを得たりするときに必要になる．

2.3 効　　果

WebWare 意味モデルを導入しラウンドトリップエンジニアリングを実現することによる効果には以下がある．

☆ Apcoordinator は，富士通株式会社の登録商標である．

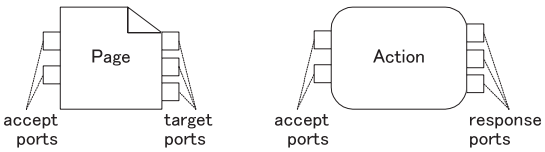

図 3 ページとアクション
Fig. 3 Page and Action.

- アプリケーション拡張・保守
  アプリケーションの拡張や保守を行う際に，ソースコード実装の工程よりも上流の工程での設計を促進することができる．そのため，設計時のバグを減らしたり，実装の自動化を進めたりすることができる．
- プラットフォーム移行
  現在 Web アプリケーションが実装されているプラットフォームから異なるプラットフォームに移行したりする際に，モデルを介在することにより，個別の移行プログラムを作らなくてもよい．
- アプリケーションのデバッグのサポート
  指示先が存在しないアンカや，どのページからも遷移が行われることのないページを検出することが容易である．
- ユーザビリティの向上
  あるページから他のページに遷移する際に，どれくらいの画面遷移を必要とするかが分かるため，ユーザビリティの向上に役立てることができる．
- テストのサポート
  モデルがプラットフォーム非依存であるため，ソースコード実装よりも上流レベルでのテストケースの導出や自動投入が可能となる．

3. WebWare 意味モデル

前章の考察から，本章では，WebWare 意味モデルを検討する．モデル化にはコンポーネント，ポート，コネクタ[5] による方法を用いる．なお，これは UML 2.0 ドラフトのアクティビティ図のサブセットにもなっている．

3.1 Page と Action

Model2 アーキテクチャに対応するため，“Page” と “Action” の 2 つの概念をモデル化する．

Page と Action はそれぞれ 0 個以上の入力ポートと 0 個以上の出力ポートを持つコンポーネントとしてモデル化される（**図 3**）．

- Page は論理的なページを表す．Page はブラウザにおいて 1 つのフレームに表示されるものを 1 つとする．

- Action は論理的なサーバのロジックである．WebWare 意味モデルでは，サーバロジックを入出力で定義しており，具体的なコンポーネント（Servlet など）を規定しない．これは，フレームワークにより，サーバコンポーネントととらえるべき単位が異なるためである．

ポートは 4 種類あり，それぞれ次のとおりである．

- Page の入力ポート（Accept）
  → Page を表示するために必要な指定情報を表す．たとえば，HTML における URL などである．
- Page の出力ポート（Target）
  → Action または Page に対するリクエストを示す．
- Action の入力ポート（Accept）
  → Action を起動するために必要な指定情報を表す．
- Action の出力ポート（Response）
  → Action からのレスポンスを示す．

それぞれのポートは，フレームワークごとに異なる属性のセットを持つ．これは，フレームワークごとに Page の表示や Action の起動などに必要な情報が異なるためである．たとえば，Struts の場合，Action の Accept は Struts の場合は URI で表されるが，Apcoordinator の場合は Form に対応するデータ型と uji.verb リクエストパラメータで示される．

3.2 遷　　移

前節で導入した Page と Action の定義を使い，Page と Action 間などの遷移を構築する手法を示す．遷移は Page または Action の出力ポートと，Page または Action の入力ポートをコネクタによって接続したものである．接続は出力ポートと入力ポートをネゴシエーションさせることによって導出する．ネゴシエーションの結果，以下の 4 種類のうちの 1 つの遷移が得られる．

- Page から Page への直接の遷移．
- Page から Action へのリクエスト．
- Action から表示する Page の選択．
- Action から Action の呼び出し．これは Page にかかわりのない遷移となるため，導出方式についての議論は本論文では対象外とする．

本方式ではポートの属性がフレームワークにより異なるため，ネゴシエーション方式もフレームワークごとに定義される．以下にページ駆動型，Struts と Apcoordinator の例をあげる．

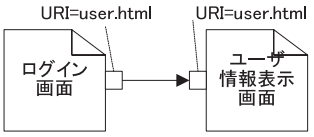

図 4 ネゴシエーション（静的ページ遷移）
Fig. 4 Negotiation between static pages.

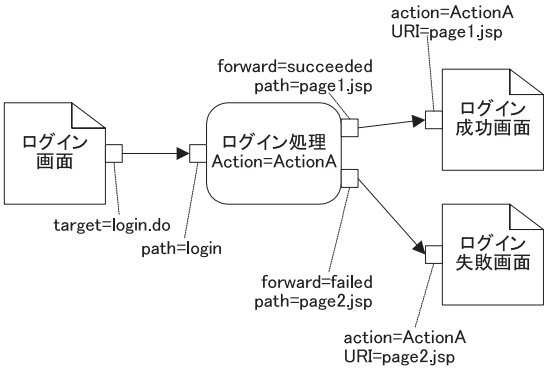

図 5 ネゴシエーション（Struts）
Fig. 5 Negotiation in Struts framework.

3.2.1 ページ駆動型

HTML を利用する場合，すなわち Page 間の静的な遷移の場合はネゴシエーションは単純で，遷移元 Page の出力ポートが指す URL と遷移先 Page の入力ポートが指す URL が合えばよい．

例を**図 4** にあげる．これは，ログイン画面からのリクエストが user.html を指定し，ユーザ情報表示画面は user.html を accept することを意味している．

Model1 アプローチの Web アプリケーションも，Page 内に直接次の Page へのリンクが埋め込まれるため，同様の方法で遷移を導出することが可能である．

3.2.2 Struts

Struts は，Model2 のフレームワークであるため，ネゴシエーションを Page から Action の間と Action から Page の間で定義する．

Page から Action への遷移導出は次のとおりである．Page の出力ポートは遷移先の URI を持つ．Action の入力ポートは自分自身のパスである path 属性を持つ．この二者が “.do” を除いて一致する場合に遷移する．

Action から Page に遷移する際は次のとおりである．Action の出力ポートは処理結果となる forward 属性と遷移先の path 属性を持つ．一方 Page の入力ポートは対象となる Action を示す action 属性と自分自身のパスである URI 属性を持つ．双方の Action と，path 属性と URI 属性の組の両方が一致する場合に遷移が定義される．

これらの様子を**図 5** に表す．

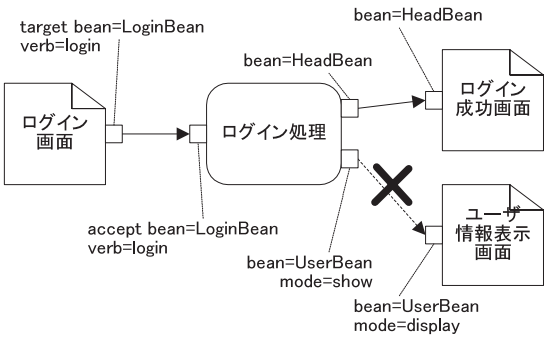

図 6 ネゴシエーション（Apcoordinator）
Fig. 6 Negotiation in Apcoordinator framework.

以上は Struts アプリケーションに含まれる定義ファイルである struts-config.xml を解析して得ることができる．

3.2.3 Apcoordinator

Apcoordinator も Struts 同様，Model2 のフレームワークであるため，ネゴシエーションを Page から Action の間と Action から Page の間で定義する．

Page から Action へは，前述のように Form に対応するデータ型と uji.verb リクエストパラメータ（以下 verb と省略）によってネゴシエーションが行われる．Page は Form に対応するデータ型についての情報と verb を送出する．サーバではその 2 つの情報から実行すべき Action を決定することができる．この仕組みにより Page から Action へのネゴシエーションが行える．この情報は Apcoordinator アプリケーションに含まれる定義ファイルであるコマンドマップを解析して得ることができる．

これを例示したものを**図 6** にあげる．ログイン画面からは LoginBean 型に login という verb をともなってデータが送出される．一方ログイン処理は LoginBean 型と login という verb の組合せを受信できる入力ポートが存在する．そのため，ログイン画面からログイン処理に対し遷移を行うことができる．

Action から Page へ遷移する際もほぼ同様で，Action は表示すべきデータ型と mode パラメータ（以下 mode）の 2 つの情報を出力する．それに対し，データ型と mode の 2 つの情報から Page のパスをマッピング表から導出することができる．これにより Action から Page のネゴシエーションが行える．この情報は Apcoordinator アプリケーションに含まれる定義ファイルであるページマップを解析して得ることができる．

図 6 の場合，ログイン処理は HeadBean 型と mode なしという出力ポートと，UserBean 型と mode が show という出力ポートが存在する．一方ログイン成功画面は HeadBean 型で mode なしの入力ポートを持つ．そのためログイン処理からログイン成功画面に対して遷移を行うことができる．一方ユーザ情報表示画面が持つ入力ポートは，型は UserBean だが mode が display である．このためネゴシエーションは失敗し，ログイン処理から遷移することはできない．

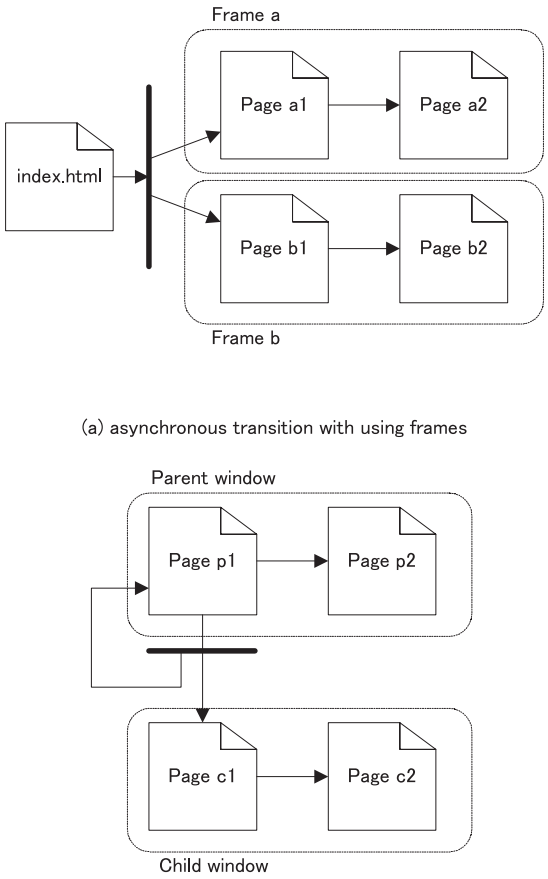

図 7 並行状態（状態の分割）
Fig. 7 Fork of a status.

3.3 並行状態

Page と Action の間の関係をサーバ側から見た場合には，1 つのリクエストで返せるレスポンスは 1 つであるため，並行状態は発生しない．

その一方で，ブラウザは以下の方法で Web ページを複数表示，操作することが可能である．

(a) 1 つのウィンドウをフレーム分割する．
(b) あるウィンドウから別のウィンドウをオープンする．

この様子を**図 7** に示す．簡単化のため，ポートの表記は省略している．太線が状態の分割を示し，それ

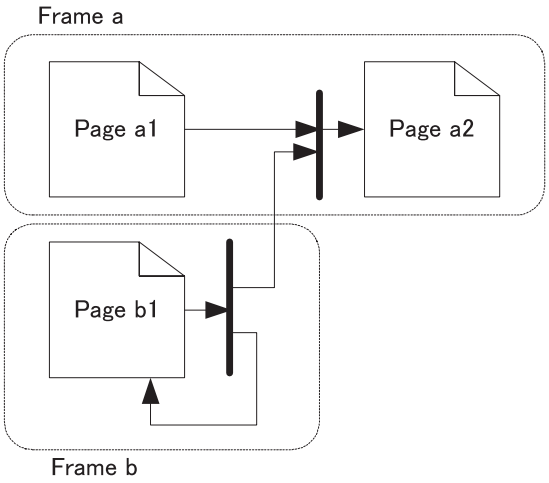

図 8 並行状態（状態の同期）
Fig. 8 Join of statuses.

ぞれのフレームまたはウィンドウが並行状態を構成する．(a) はフレームを使った場合である．index.html からフレーム分割された Page a1 と Page b1 が表示され，それぞれのフレームは独立に遷移する．(b) はある Page から他のウィンドウをオープンした場合である．Page p1 から別のウィンドウとして Page c1 をオープンし，親ウィンドウと子ウィンドウは独立に遷移する．

また，あるフレームまたはウィンドウの操作によって別のフレームやウィンドウが遷移することがある．この様子を**図 8** に示す．Frame a に示されている太線が状態の同期を示している．Frame a においては，Page a1 から Page a2 への遷移は，Page b1 からのリクエストが必要となる．

4. 評　　価

本章では，前章で構築した WebWare 意味モデルの有効性を示すために，実際に解析器を試作して既存のアプリケーションからモデルが導出できること，そのモデルからアプリケーションを構築することによってラウンドトリップエンジニアリングが可能になることを示す．また，産学連携の効果と，関連する技術について議論する．

4.1 リバースエンジニアリング

今回の解析器の試作では，対象の Web アプリケーションを Struts フレームワークに設定した．

Struts では，画面と Action をつなぐマッピング情報として struts-config.xml が定義されるため，この情報を利用する．struts-config.xml からは

1. ページからビジネスロジックを呼び出すための条件，
2. ビジネスロジックの処理結果からどのページを表示させるか，

という 2 つの情報を得ることができる．この 2 つはそれぞれ Action の Accept ポートと Response ポート，そして Page の Accept ポートに利用することができる．

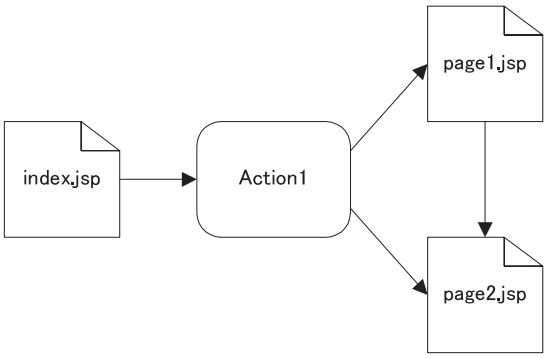

図 9 解析対象アプリケーション
Fig. 9 A sample target application.

ページの Target ポートは struts-config.xml からは取得できないため，JSP ファイルの ⟨a⟩ タグもしくは ⟨form⟩ タグより取得する．

なお，struts-config.xml で定義されていても，実際に Action の Response ポートが使われるかどうかは Action の Java コードを解析しないと分からない．ただし，簡単化のため，この部分については将来の課題とし，今回は実装は見送った．

今回の試作では解析結果を XML で出力する．**図 9** のアプリケーションを解析器に適用させて生成されたモデルの例を**図 10** に示す．

4.2 ラウンドトリップエンジニアリング

ラウンドトリップエンジニアリングに実際に効果があることを試すために，前節で試作した解析器により生成されたモデルから Web アプリケーションを構築することを試みた．

Web アプリケーションをモデルから生成するために，共同研究を行っている富士通研究所の自動生成技術[6]（以下 Apmodeler）を使った．この技術ではプロファイル付きの UML を Web アプリケーションに変換するため，WebWare 意味モデルと形式が異なる．そのため，WebWare プロジェクト内において変換器を作成した．

全体の処理のフローは**図 11** のようになる．

```
<page-transition>
 <page type="jsp" url="/index.jsp">
  <targets>
   <target event="user" url="/Action1.do" type="submit" method="GET" />
  </targets>
 </page>
 <page type="jsp" url="/pages/page1.jsp">
  <accepts>
   <accept type="url">
   <url-accept>/Action1</url-accept>
   </accept>
  </accepts>
  <targets>
   <target event="user" url="page2.jsp" type="submit" method="GET" />
  </targets>
 </page>
 <page type="jsp" url="/pages/page2.jsp">
  <accepts>
   <accept type="url">
    <url-accept>/Action1</url-accept>
   </accept>
  </accepts>
 </page>
 <action resource="com.fujitsu.pfu.pal.test.Action1">
  <accepts>
   <accept type="url">
    <url-accept>/Action1</url-accept>
   </accept>
  </accepts>
  <responses>
   <response type="struts" version="1.1">
          <struts-forward name="success" path="/pages/page1.jsp" />
   </response>
   <response type="struts" version="1.1">
   <struts-forward name="fail" path="/pages/page2.jsp" />
   </response>
  </responses>
 </action>
</page-transition>
```

図 10 解析例（XML）
Fig. 10 An example of analysis in XML form.

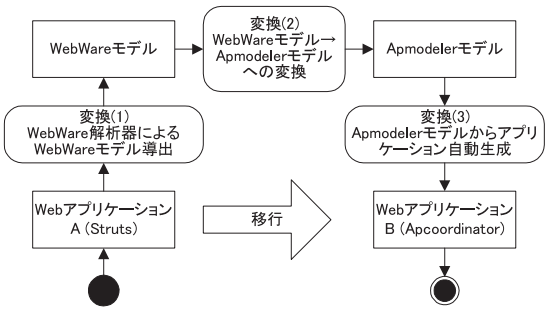

図 11 ラウンドトリップエンジニアリング
Fig. 11 Round-trip engineering.

その結果，WebWare 意味モデルが Apmodeler モデルに変換でき，そこから Web アプリケーションの生成ができることが検証できた．

また，Web アプリケーション A は Struts 準拠のアプリケーション，Web アプリケーション B は Apcoordinator 準拠のアプリケーションである．WebWare 意味モデル，Apmodeler モデルとも特定の実装技術に非依存であるため，このようなプラットフォームの移行がモデルを通して可能であることも分かった．

この段階までは，アプリケーションやモデルに手を入れることなく自動変換が可能であったが，モデルを改変することで Web アプリケーションの仕様を変更

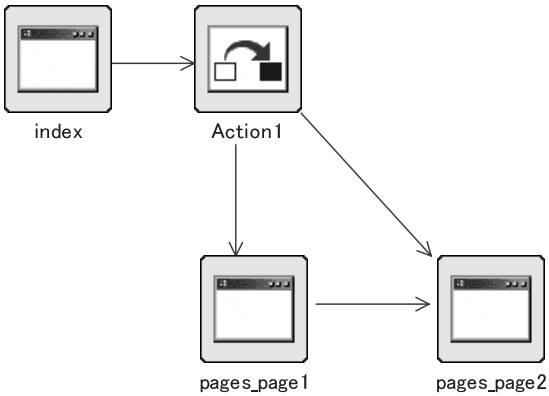

図 12 Apmodeler モデルへの変換例
Fig. 12 An example of translation to an Apmodeler model.

することができた．これは，保守作業をコード上ではなくモデル上で可能になったということである．

図 10 の解析で得られたモデルを Apmodeler モデルに変換した例を**図 12** に示す．ウィンドウが描かれたアイコンがページ，矢印が描かれたアイコンが Action を表す．

4.3 産学連携の効果

今回のプロジェクトでは，産学共同プロジェクトの枠組みで研究が行われている．そこでは，以下のような連携を基本とした．

リバースエンジニアリング技術は基礎的で地道な作業が多いため，企業ではあまり深い研究が行われていない．そのため，大学で以前から行われている言語解析などの研究成果をもとに，Web アプリケーションに必要な機能を追加した．

逆にフォワードエンジニアリングは，短納期化が求められる現場では，自動生成器を作るという観点から企業における研究が進んでいる．そのため，企業の成果物を利用することにした．

実際のプロジェクト遂行では，互いがこれらの技術を持ち寄り，企業側で両方の技術を統合してラウンドトリップエンジニアリングを実現する作業を行った．

企業で実際のアプリケーションにリバースエンジニアリング技術を適用する作業を行ってみたところ，モデルの構成に不十分な点が見つかった．そこで，それらの点をフィードバックする作業を行った．これは，単に大学の研究結果を企業で応用するだけではなく，応用結果を大学側にフィードバックできたということで，産学連携による効果であると考える．

4.4 データの取扱い

今回のモデルは，画面とアクションの間の遷移に焦

点を当て，できるだけ抽象度の高い，簡潔なものを目指した．現在の試行においては，画面遷移についてのラウンドトリップエンジニアリングにおいては現在のモデル要素で十分であるという結果を得ることができた．

一方，画面とアクションの間で流れているデータについては，各データ要素ごとに Web ページ上，ビジネスロジック内，データベースを通して扱うことが必要と考えている．

これには，従来の JavaBean などのような大粒度のデータを扱うだけでは不十分であり，細粒度解析が必要であると考えている．たとえば，遷移の原因になっているデータが，どの入力データをもとに決まっているか，などの情報は JavaBean 内の個別データの解析情報を必要とする．

本モデルの場合，データフローは遷移，つまりコントロールフロー上に載せることができるため，現在のモデルを拡張することで容易に対応が可能と考えている．

4.5 関連研究

モデルをベースとして Web アプリケーションを構築する方式については，以下に示すようなアプローチが提案されている．

1) ReWeb

既存の Web アプリケーションを解析，モデル化し，テストに応用しようとするアプローチとして，Ricca らによる ReWeb[7] がある．ReWeb では，動的ページの遷移条件を，遷移リンク上に設定された条件により決定する．それに対し，本論文による方式ではページの Action のポートどうしのネゴシエーションによって遷移を決定するという相違がある．

ReWeb は Web アプリケーションに外部からアクセスして情報を抽出する．そのため動的遷移決定には人間の介在が必要となっている．それに対し，本方式では Web アプリケーションのソース解析を行う．またフレームワークごとの拡張定義を行うことで動的遷移の決定を自動化することが可能となっている．

2) Web Application Descriptor

Web Application Descriptor による方式[1] では，画面遷移情報をモデルとして定義し，そこからマッピング情報やページなどを自動出力することでアプリケーションを作る枠組みである．この方法は新規に作成するアプリケーション，すなわちフォワードエンジニアリングに関しては有効だが，今回提案しているラウンドトリップエンジニアリングを行う枠組みが用意されていないため，既存のアプリケーションを改変，拡張する場合にはそのまま適用することは難しい．特に，MVC 型でないアプリケーションをモデル化することができないため，現在も稼動している古いタイプのアプリケーションを移行していくためには十分ではないと考えている．

ただ，今回提案した方式を利用し，4.2 節と同じ方法でモデルを WAD モデルに変換することにより，フォワードエンジニアリングに適用することは可能であると考える．

なお，WAD に関してはソースコードを参照せず，動的解析によりリバースエンジニアリングを行う手法が提案されている[8]．我々の手法はソースコードが存在することを前提としているため，方針が異なる．しかし，我々の手法も動的解析を組み合わせることにより，より詳細な情報を抽出することが可能になると考える．

3) MDA

前節と似たアプローチとして，OMG の提唱する MDA[9] がある．MDA では，相対的により上流のモデルである PIM（Platform Independent Model）とより下流のモデルである PSM（Platform Specific Model）という 2 つの概念を提唱している．PIM と PSM の間の相互変換を実現することで開発効率化と上流モデルの再利用を行う．

本方式を MDA にあてはめると，WebWare 意味モデルや富士通モデルを PIM とすれば，ソースコードが PSM に相当し，4.2 節（図 11）で論じた変換はそれぞれ次のようになる．

変換 (1) PSM→PIM 変換
変換 (2) PIM→PIM 変換
変換 (3) PIM→PSM 変換

本方式は特定のプラットフォームに依存せず，上記のようにそれぞれの変換が MDA の概念と整合するため，MDA 技術として応用することが可能である．

5. おわりに

本論文では，Web アプリケーションの意味を記述するためのモデルである WebWare 意味モデルを提案した．このモデルの特徴は，特定の実装技術に依存しないことと，ラウンドトリップエンジニアリングを目標としていることである．これにはリバースエンジニアリングが含まれ，現在広く存在する Web アプリケーションの意味モデルを導出することができ，保守作業や再構築において品質の向上に貢献することが可能となる．

その実効性を示すため，解析器を試作し，実際に既

存のアプリケーションから WebWare 意味モデルが導出できることを示した．さらに，導出により得られた WebWare 意味モデルから再び Web アプリケーションが生成できるラウンドトリップエンジニアリングが可能になることを示した．

5.1 産学連携について

今回の研究は名古屋大学を中心とした産学連携の WebWare プロジェクトで行った．そのため，大学の研究成果であるリバースエンジニアリング技術と企業の研究成果である自動生成技術を融合し，ラウンドトリップエンジニアリングの実現性と有効性の確認を早期に行うことができた．また，大学の研究結果を企業で応用する際に，発生した不具合を大学にフィードバックし，より完成度の高いモデルを作成することができた．

5.2 今後の課題

今回提案したモデルは，Page と Action，その間の遷移については含んでいるが，そこで扱われるデータについては含まれない．データの扱いについては今後の課題である．特に，Web ブラウザに表示されるデータとサーバ内で扱われているデータがどのように連携しているかを解析することは，従来のプログラムのデータフロー解析では得られない興味深い話題である．

また，今回の試作では，比較的単純な Struts アプリケーションをターゲットとしていた．今後は，解析器の品揃えを増やし，さらに複雑なアプリケーションを解析できるようにする必要がある．

謝辞 本研究は，文部科学省リーディングプロジェクト「e-Society 基盤ソフトウェアの総合開発」の支援により行われた．また，WebWare プロジェクトのメンバには貴重な議論をいただいた．この場を借りて謝意を表したい．

参 考 文 献

1) 堀 雅洋，田井秀樹：モデルに基づく Web アプリケーション開発，情報処理学会誌，Vol.45, No.1, pp.16–21 (2003).
2) Apache Software Foundation: Struts in Apache Jakarta Project. htttp://jakarta.apache.org/struts/
3) Java Community Process: JavaServer Faces. http://www.jcp.org/en/jsr/detail?id=127
4) 松塚貴英，野村佳秀：JSP を用いた Web アプリケーション構築のためのフレームワーク UJI，情報処理学会研究報告 (2000).
5) Garlan, D. and Shaw, M.: *Software Architecture*, Prentice Hall (1995).
6) Matsutsuka, T.: Model-Driven Development Approach to Web Applications, *The IASTED International Conference on Software Engineering 2005* (Feb. 2005).
7) Ricca, F. and Tonella, P.: Analysis and Testng of Web Applications, *ICSE2001*, pp.25–34 (May 2001).
8) 安部麻里ほか：動的逆解析による Web アプリケーション・モデルの抽出，日本ソフトウェア科学会第 20 回大会（2003 年度）論文集 (2003).
9) Object Management Group: Model Driven Architecture. http://www.omg.org/mda/

(平成 16 年 8 月 30 日受付)
(平成 17 年 2 月 1 日採録)

松塚 貴英（正会員）

1971 年生．1994 年東京工業大学電気電子工学科卒業，1996 年東京工業大学情報理工学研究科計算工学専攻修士課程修了．同年富士通株式会社に入社．現在，株式会社富士通研究所 IT コア研究所ソフトウェアイノベーション研究部所属．分散企業システム，Web アプリケーションフレームワーク，ソフトウェアアーキテクチャ等の研究，開発に従事する．2001〜2002 年，米 Carnegie Mellon University 客員研究員．

阿草 清滋（正会員）

1947 年生．1970 年京都大学工学部電気工学第二学科卒業．1972 年同大学院工学研究科電気工学第二専攻修士課程終了．同博士課程へ進学．1974 年より同情報工学科助手．同講師，助教授を経て 1989 年より名古屋大学教授．現在，同大学院情報科学研究科教授．同研究科長．工学博士．専門分野はソフトウェア工学，ソフトウェア開発方法論，知的開発環境，ソフトウェアデータベース，仕様化技法，再利用技法，マンマシンインタフェース．電子情報通信学会，ソフトウェア科学会，IEEE，ACM 各会員．

山本晋一郎（正会員）

1987 年名古屋大学工学部卒業後．同大学大学院に進学，1991 年同大学助手，1996 年講師．1998 年愛知県立大学情報科学部助教授．プログラミング言語処理系，ソフトウェアの形式的開発手法，ソフトウェア開発環境に関する研究に従事．近年は，細粒度のソフトウェア・リポジトリに基づいた CASE ツール・プラットフォームに関する研究を進めている．電子情報通信学会，情報処理学会，日本ソフトウェア科学会各会員．