Vol.2012-SE-178 No.27
2012/11/2

コードからの抽象ユーザインタフェース情報抽出手法

井上尚紀[†1]　　岸知二[†1]

近年 MDA による GUI アプリケーションのマルチフロントエンド開発が注目されている．しかし，ソースコードのみ存在する既存アプリケーションを他のフロントエンドに移植する為には，一度リバースモデリングを行い，抽象的な UI 設計を得なければならない．そこで本研究では，テキスト情報を利用することによって，Web アプリケーションのソースコードから抽象的な UI 設計を抽出するための手法を提案する．

The technique for extracting abstract user interface information from the source code

NAOKI INOUE[†1]　TOMOJI KISHI[†1]

Recently, Multi Front-end engineering of GUI applications has been attracting attention by MDA. But, if you want to port the existing application that exist only source code to other front-end, you must obtain the abstract UI design by the reverse modeling. Then, this paper propose the technique for extraction abstract user interface model from the web application's source code by taking advantage of the text information around the UI element.

1. 研究背景

1.1 はじめに

近年，GUI アプリケーションを様々なプラットフォーム上に展開したいという要求が強まっている．例えば，同一のサービスを提供するアプリケーションを，Web・モバイル・デスクトップ上で展開している例もある．そこで注目されているのが，MDA(Model Driven Architecture：モデル駆動アーキテクチャ)によるマルチフロントエンド開発である．MDA は，標準化団体 OMG(Object Management Group)が提唱しているソフトウェア設計・開発手法であり，様々な分野での適用が検討されている．[1]

しかし，MDA による開発を行う為には抽象的なモデルが必要であり，既存アプリケーションを他のプラットフォームに移植することを狙う場合，リバースモデリングによって一度抽象的なモデルを作成する必要がある．そこで本研究は，MDA による UI の生成を考慮した，Web アプリケーションのリバースモデリングをサポートする手法を提案することを研究目的とする．今回は，特に Web アプリケーションのリバースに焦点を当て，UI の MDA 開発で必要な情報のひとつである，抽象的な UI エレメントとそのエレメントが持つ意味情報を得るための手法を提案する．具体的には，UI エレメントに付随する，もしくは近くにあるテキスト情報を利用することにより，UI エレメントが持つ意味情報を推測することを狙う．

1.2 背景知識

1.2.1 マルチフロントエンド開発

マルチフロントエンド開発は，下図 1 のように，一つのアプリケーションに対して，多様な UI 構成の体系的な設計と実装を取り扱う[2].

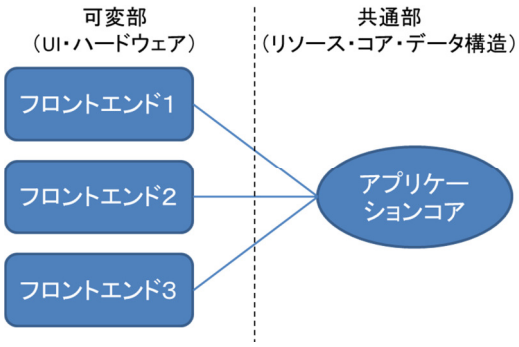

図 1 マルチフロントエンドなアプリケーションの例
Figure 1 The example of the multi front-end application

マルチフロントエンド開発では，一つのアプリケーションにおいて，全ての異なるフロントエンドが同一のアプリケーションコアにアクセスし，どのフロントエンドにおいても同様の機能を提供する．つまり，各フロントエンドにおいて共通の UI 構造を有している．また，各々のフロントエンドにおいて，各々のフロントエンドは特定の UI プラットフォームの特性に依存した形をとらなければならない．例えば異なるサイズの論理的プレゼンテーションユニットに UI エレメントをグルーピングする場合などに，構造に変更を加えなければならない．したがって，マルチフロントエンド開発における一番の困難は，いかにして共通の構造を保ちつつプラットフォーム特性に応じて構造を変更するかにある．この困難への解決策としてモデル駆動アーキテクチャによる開発が注目されている．

1.2.2 MDA

MDA は OMG が提唱しているソフトウェア開発手法である．実装技術から独立した PIM(Platform Independent

†1 早稲田大学

ⓒ2012 Information Processing Society of Japan

Model ：プラットフォーム独立モデル)を作成し，PSM(Platform Specific Model ：プラットフォーム依存モデル)へ変換し，PSM からソースコードを生成する．MDA におけるプラットフォームとは，Java，CORBA，XML のようなプログラミング言語や実現方式等の実装技術やハードウェア仕様等を指す．

MDA の利点のひとつは，プラットフォームに依存しないモデル(PIM)を作成する事により，プラットフォームの進化があったとしても PIM を再構築しなくてよい点である．また，ソースコードの自動生成によるプログラミング工程の効率化や，コーディング技術の差を埋めることによるソースコード品質の底上げ等も利点である．図 2 に MDA による開発の全体像を示す．

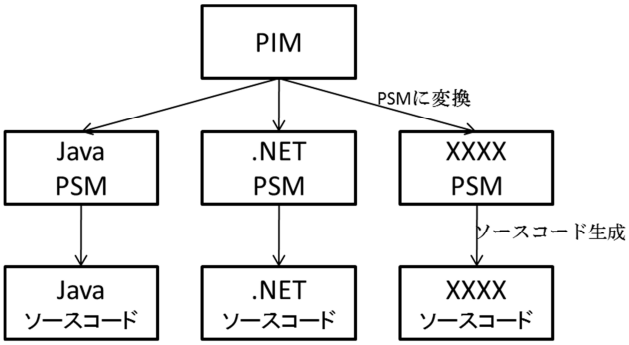

図 2 MDA による開発の全体像
Figure 2 The overview of the MDA engineering

以下，PIM, PSM の各モデルについて説明する．

A) PIM

PIM とは，OS やプログラミング言語といったプラットフォームに依存しない業務用件モデルである．手動で作られる．

B) PSM

PSM とは，Java 等の特定のプラットフォームに特化したモデルであり，プラットフォームの特性を加味したモデルである．PSM への変換ルールに従い，PSM は PIM から変換されて作成される．またプラットフォーム毎に変換ルールを定義することで，一つの PIM から複数の PSM に変換されることもある．一度変換ルールを定義すれば，プラットフォームに変化がない限りそのルールを繰り返し使用することができる．

更に，変換されて作成された PSM からソースコードを自動生成する．このモデルからコードへの自動生成の際も変換ルールを定義する必要がある．

1.2.3 リバースモデリング

リバースモデリングとは，現在使っているソースコードを基点として，より上流の成果物（モデル）を作成すること[3]である．

上記の MDA による開発を行う為には，PIM を作成しなければならない．その為，最初から MDA 開発を考慮せずに開発され，設計モデルのないアプリケーションを MDA による開発へ移行し，他のプラットフォームへ展開する為には，一度リバースモデリングを行い，プラットフォームに依存しない，抽象的な設計モデルを得る必要がある．

一般的に開発者以外が，ソースコードの情報から設計情報を導く為には，ドメイン知識やそのコードが書かれた背景等を推測しなければならなく，難しいとされる．その為，ドメインや状況に応じて様々な手法が提案されている[4][5][6]．

2. 従来研究と研究目的

2.1 MANTRA アプローチ

Goetz Botterweck は，共通性と可変性の衝突によるマルチフロントエンド開発の困難を解決する為に，モデル駆動 UI 開発を拡張した手法，MANTRA アプローチ[2]を提案した．その概要は下図 3 で示される．

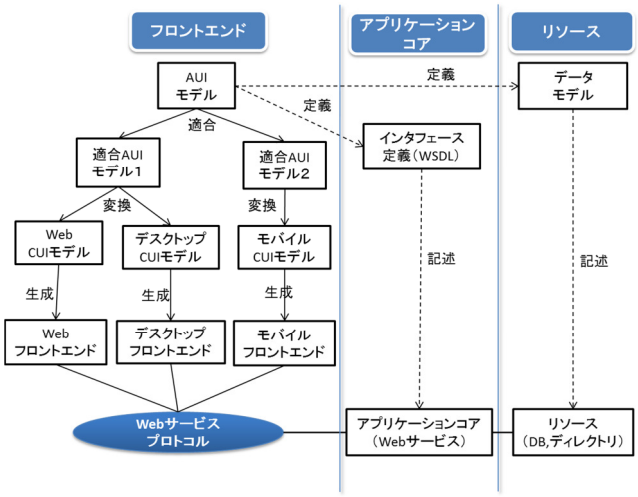

図 3 MANTRA アプローチ概要 [2]より
Figure 3 The overview of MANTRA approach from [2]

MANTRA アプローチの特徴は２つある．特徴の１つはフロントエンドのプラッフォームに依存しない段階でアプリケーションコアとのインタフェースを定義できる点にある．これにより，フロントエンドを効率良く自動生成することが可能になっている．もう１つの特徴は，PIM にあたる AUI（抽象ユーザインタフェース）モデルに，適合という処理を加えることにより，画面サイズ等の緩いプラットフォームの制限を段階的に加えられる点にある．これにより，自動生成できるフロントエンドに柔軟性を持たせることが可能になっている．この２つの特徴から，MDA によるマルチフロントエンド開発に特化したアプローチとなっている．

本研究では，この MANTRA アプローチを適用して開発していくことを前提とする．

2.2 ReGUI

Inês Coimbra Morgado ら[7]は，ソフトウェアアプリケーションの GUI の構造と振る舞いを視覚的にそして形式的

ⓒ2012 Information Processing Society of Japan

にリバースモデリングすることをサポートする為に，実際にアプリケーションを実行しながら解析する動的解析によるアプローチとそれを実現するツールを提案している．このアプローチは，ナビゲーションモデル等の視覚的な情報を自動的に取得し，ソフトウェアの検証をサポートすることを狙っている．しかし，出力されるモデル上の UI エレメントの名前付けは機械的であり，UI エレメントの意味は考慮していない．

2.3 MORE

Yves Gaeremynck らは，マルチデバイスアプリケーション開発をサポートする為に，既存の Web アプリケーションから抽象的で意味的なモデルを自動的に推論する，MORE アプローチ[8]を提案している．その概要図は図 4 に示される．

このアプローチは UI のパターンを観察し，得られた共通のパターンを利用して作られたルールとメトリクスを用いて，入力エレメントとラベルによるキャプションを対応付けることで，HTML フォームの視覚的レイアウトを意味的モデルに自動的に変換する．アプローチのフローチャートは以下の通りである．

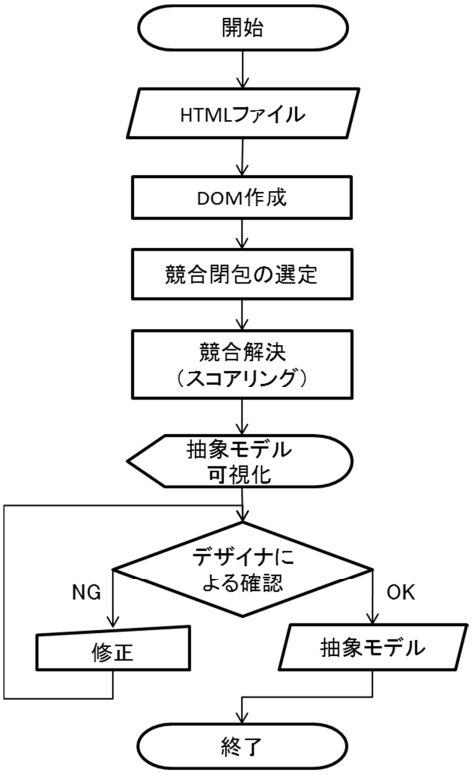

図 4 MORE アプローチのフローチャート
Figure 4 The flowchart of MORE approach

このアプローチにより，制限はあるがソースコードから UI の意味的モデルを部分的に自動的に得ることが可能になり，MDA による開発に活かすことができる．しかし，HTML<Select>エレメントやトリガー等の，<input>エレメント以外のエレメントに対応していない等の制限や，チェックボックスの解析の際に間違ったラベルと対応付けられてしまうことがあり精度が悪い，エレメントの数が増えると計算量が指数関数的に増えてしまう等の問題点がある．

本研究では，この MORE アプローチを包括・拡張し，以上の制限の緩和や計算量の問題を改善し，より実用的なアプローチを提案することを狙う．

2.4 研究目的

MANTRA アプローチは，MDA によるマルチフロントエンド開発を行う上で有用な手法である．しかし，既存のソースコードのみ存在するアプリケーションを，MANTRA アプローチを用いて他のフロントエンドに移植する場合は，一度リバースモデリングによって AUI モデルを得る必要がある．そこで，「MDA による UI のマルチフロントエンド開発を考慮した，Web アプリケーションのリバースモデリングをサポートすること」を本研究の目的とする．

また上で紹介した通り，UI 情報や抽象的なモデルを得るための手法はいくつか提案されているが，UI エレメントの意味情報は，抽象 UI モデルを作成する上で必ず必要となるが，これを自動的に適切に行なっている研究は少なかった．そこで，本研究は「UI エレメントの意味情報の抽出」にフォーカスした手法を提案する．

ここで本研究の「UI エレメントの意味情報」について説明する．UI エレメントの意味情報とは，その UI エレメントはユーザーが何をする為に存在するのかを示す情報である．例えば下図 5 では，下のフォームにはユーザー名を入力する為のテキストフィールドと，パスワードを入力する為のパスワードフィールドがある．この場合，上側のテキストフィールドは「ユーザー名を入力する」という意味情報を持ち，下側のものは「パスワードを入力する」という意味情報を持つ．この意味情報は，MANTRA アプローチの AUI モデルにおける UI コンポジットの名前，そしてその UI コンポジットはどの UI エレメントを所有しているかを知る為に必要となる．

図 5 UI エレメントの意味情報の例
Figure 5 The semantic information of UI Element

3. アプローチと提案手法

3.1 前提条件

1.1 節でも述べている通り，本研究は Web アプリケーションをリバースモデリングすることを前提としている．その理由は，今回は UI 情報を抽象化することを狙っている為，最初から Web を通して外部のアプリケーションコアに

接続にするようなアプリケーションをリバースモデリングすることが望ましく，更にそのようなアプリケーションを他のフロントエンドに展開しようとしている典型的な状況が Web アプリケーションであると考えた為である．

今回は MANTRA アプローチと同じく，様々な Web アプリケーションの中でも，HTML フォームを利用してユーザーから入力・選択された情報を引数としてアプリケーションコアへ渡し，データが返ってくるようなアプリケーションを想定している．

また，今回は幾つかの Web デザインのガイドライン[9][10][11]を参考にし，基本的にはそれらに則った，標準的な UI 構造のアプリケーションを対象とし，テキスト情報の解析にフォーカスするので，画像は扱わないこととする．

3.2 アプローチ

以下に本提案のアプローチを示す．

I. 取り扱う UI エレメントの範囲拡大

MORE アプローチでは，HTML<Select>エレメントの解析は対象外であった．それは，<Select>エレメントの解析は<input>エレメントの解析に比べて労力の割に得られる意味情報が些細なものであるという理由からであった．しかし，<Select>エレメントを解析できれば扱える UI の幅が広がり，より実用的になる．また，<input>エレメントの解析プロセスを適切に拡張すれば<Select>エレメントにも一部適用できると考え．そこで本研究では<Select>エレメントを扱うこととした．

II. 更なる UI のパターン分けによる意味付け（テーブル・フィールドセットの情報の利用）

今回想定している標準的なアプリケーションの GUI は，幾つかの典型的なパターンがあると考えられる．そこで，Web デザインのガイドラインを参考に UI 構造をパターン分けすることにより，UI エレメントの意味付けの自動化に利用する．

様々な UI を調査した結果，視認性向上の為だと思われるが，HTML フォームはテーブルやフィールドセットと組み合わせて使われていることが多いことを発見した．

MORE アプローチでは全ての入力エレメントとラベルに対して競合閉包の選択やメトリクスの計算処理を行っている為，エレメント数が増えると指数関数的に計算量が増えてしまう問題がある．

そこで，本研究ではテキスト情報と共にテーブルやフィールドセットを用いた意味付けを行えば，メトリクスの計算をしなくても意味付けを行うことができ，その為競合の数を減らすことができ，結果として計算量を減らすことができると考えた．また，テーブルやフィールドセットを用いた意味付けによって情報量が増え，精度の向上にも繋がると考えた．

3.3 提案手法

本提案手法は以下の通りである．提案手法のフローチャートを下図 6 に示す．MANTRA アプローチの AUI メタモデルにおける UI エレメントの特殊な例としてトリガーがあり，根本的にトリガー以外とトリガーでは意味情報の抽出方法が全く異なる． その為，トリガーとトリガー以外の UI エレメントの解析は同時には行われず，本節でも分けて説明する．

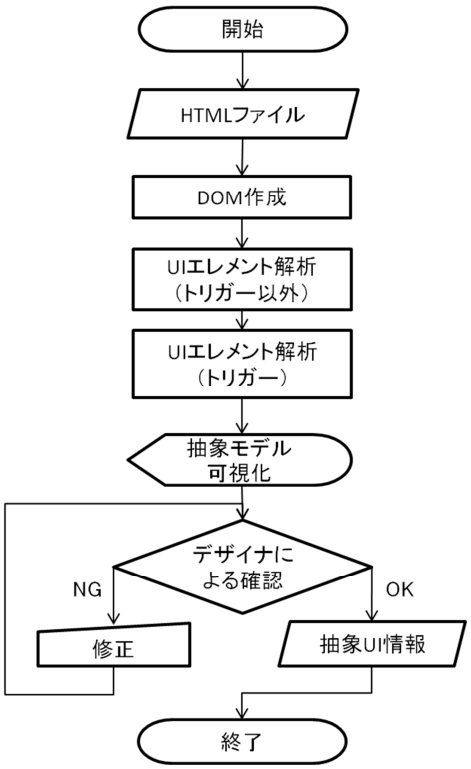

図 6 提案手法フローチャート
Figure 6 The flowchart of the proposed method

3.3.1 UI エレメント（トリガー以外）

UI エレメントは，InputField や SelectOne，SelectMany のように，ユーザーに値を入力・選択させるエレメントである．具体的エレメントとしては，テキストボックス，ドロップダウンリスト，チェックボックス等，様々なエレメントがある．UI エレメントは，エレメントの役割が明確なので，対応するラベル（[9]で定義される所の Control Label にあたる．例えば上図 5 の場合はユーザー名ラベル等）が見つかれば自動的意味付けは比較的容易に行える．しかし，入力する内容のヒントとなるラベル（[9]で定義される所の Supplemental explanation にあたる．例えば上図 5 の場合は括弧書きの補足）が含まれている場合があり，それが意味付けの自動化を難しくしている．また，チェックボックス等，対応するラベルがないエレメントも存在する可能性があるが，その場合は人手でその意味付けをする必要がある．Web デザインのガイドラインや様々な Web サイトの観察をした結果，UI 構成のパターンとしては，大きく分けて２パ

ターン存在すると考えられる．

● テーブルやフィールドセットで仕切られるパターン

下図 7 のように，会員登録等，ユーザーから複数の入力・選択を必要としている場合に見られることが多く，この場合は UI エレメントが入っているテーブルの左側のカラムや上側のローに入力・選択する内容が簡潔に書かれていることが一般的であり，右側のカラムや下側のローにフォームが存在することが一般的であり，入力・選択する内容とフォームの種類（入力・選択）を合わせたものが UI エレメントの意味情報となる（場合によっては右側のカラムや下側のローの中に更にテーブルが入っていたり，テーブルで仕切られていないパターンで複数エレメントが配置されていたりする（例えば下図 7[]の生年月日））．

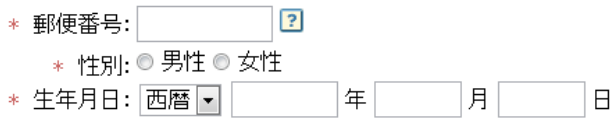

図 7 左のテーブルに対応するラベルがあるパターン
Figure 7 The UI pattern has corresponding label in the left table

また，同様のことがフィールドセットにおいても言える．フィールドセットは HTML フォームのグループ化に使われる．本手法ではこの一般的なパターンを利用する．

● テーブルで仕切られていないパターン

例えば，図 8 のような配置が考えられる．この場合は，一般的にはエレメントの左や上に対応するラベルがあることが多い（テキストボックスの場合は非フォーカス時にボックス内にヒントとなるラベルが表示されることもある）が，上のパターンと違ってテーブルで仕切られていない為エレメントに対応するラベルを探すことも上のパターンより難しく，対応するラベルとヒントとなるラベルの区別も自動的には難しい．

そこで MORE では，様々なルールを用いて対応する可能性のある UI エレメントとラベルの組み合わせを決定し，そこから独自のメトリクスによるスコアリングを行い，対応するラベルを決定している（詳細は[8]参照）．本提案でもテーブルで仕切られていないパターンにおいては，部分的に MORE のアプローチを使用する．

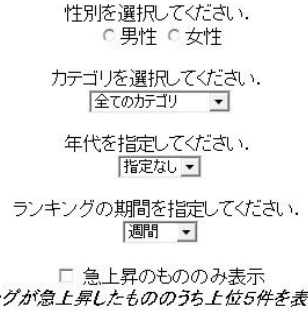

図 8 テーブルで仕切られていないパターン
Figure 8 The UI pattern which isn't divided by table

下図 9 に，トリガー以外の UI エレメントの意味付けを行う部分の本提案のアルゴリズムのフローチャートを示す．アプローチで示したように，UI 構成としてテーブルやフィールドセットを持つ場合は意味情報をそこから抽出し，場合によっては更にその下の階層の意味情報を抽出する．

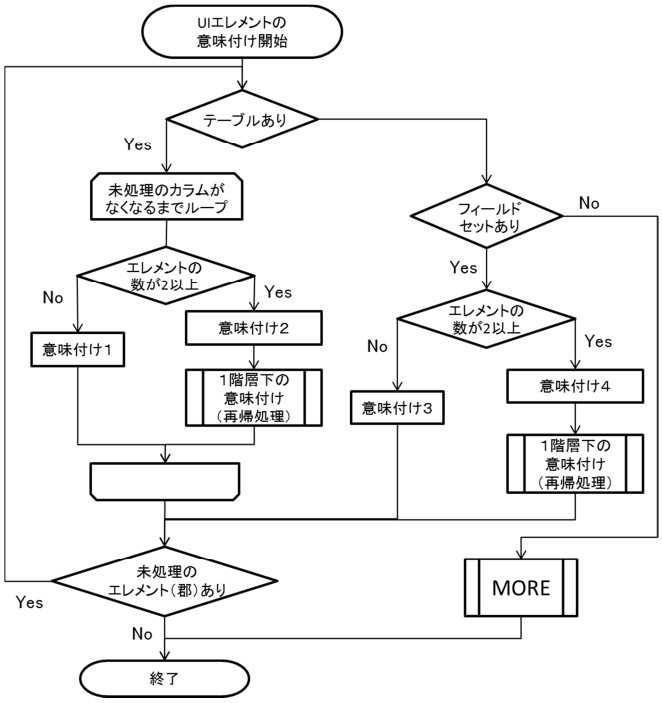

図 9 UI エレメント（トリガー以外）解析のための提案アルゴリズムのフローチャート
Figure 9 The flowchart of the proposed algorithm for analyzing UI element except the Trigger

以下フローチャートの内容を説明していく．図 9 右下に配置されているプロセス,「MORE」をみると分かるように，基本的にテーブルもフィールドセットも存在しない場合は，多少拡張はしているが基本的に MORE と同じアプローチを取る．拡張は，主にセレクトエレメントの解析についてであるが，本提案の本質ではない為，スペースの都合上，説明を割愛する．

テーブルの中にフォームがある場合は，フォームがある側のカラム（ロー）を 1 行（列）毎に処理していく．ここでカラムの中にエレメントが複数入っている場合と 1 つのみ入っている場合で意味付けのプロセスが異なってくる．

エレメントが 1 つの場合は，基本的には「対応するラベルの内容に対してエレメントの種類の処理を行う」というのが意味付け 1 プロセスで意味付けをする内容である（図 7 では「郵便番号を入力する」というのが上側のテキストフィールドの持つ意味情報である）．

エレメントが 2 つ以上の場合は，まず意味付け 2 プロセスとなる．これは UI コンポジット（UI コンポジットと UI エレメントを所有することができるエレメント）として抽象 UI 情報を抽出し，「対応するラベルの内容に対してエレ

メントの種類（一つでも入力エレメントが含まれていたら「入力」，それ以外のみだったら「選択」等）の処理を行う．」という意味付けを行うプロセスである（図 7 では「生年月日を入力する」という意味を持つ UI コンポジットが抽出される）．そして，この UI コンポジットが持つ UI エレメントとして，もしくは更なる UI コンポジットとして，再帰的にカラムの中の UI エレメントを分析していく（図 7 の場合は，生年月日 UI コンポジットの下に西暦を選択する，年を入力する，月を入力する，日を入力する，という意味情報を持った 4 つの UI エレメントが抽出される）．これはテーブルの中にテーブルやフィールドセットが入っている場合も考えられるからである．

また，フィールドセットで HTML フォームが仕切られている場合もある．その場合も<legend>タグが対応するラベルにあたり，エレメントの数によってテーブルの場合と同様の意味付けを行う．

3.3.2 UI エレメント（トリガー）

トリガーとは，ボタンやリンク等，遷移やオペレーションを開始させるきっかけとなるエレメントである．トリガーには 2 種類あり，画面の遷移のトリガーとなるものが NavigationTrigger，トランザクションのトリガーとなるものが OperationTrigger である．具体的エレメントとしては，ハイパーリンクやボタンがあるが，それらが NavigationTrigger にあたるのか，OperationTrigger にあたるのかの識別は難しく，基本的に従来手法では人手による解析を行なっていた．MORE アプローチも，トリガーの解析は対象としていない．

本手法ではここでもテキスト情報を利用して解析をサポートする．リンクやボタンのテキストとして「次へ」や「前へ」「進む」，「戻る」等が入っている場合は NavigationTrigger だと断定し，リンク先のタイトルを取得後，そこへの遷移という意味付けを行う．その他のボタンやリンクに関しては未定義とし，人手での識別を行う．

例外として，テーブルでリンクが連なっている場合はナビゲーションバーであると考えられ，それらのリンクは全て NagigationTrigger に分類される．

また，3.3.1 と 3.3.2 で得られた情報は，パターンに基づき自動的に推測された情報であり，必ずしも正しいとは限らない．その為デザイナによる修正をアプローチに挟む必要がある．

3.4 適用例

本節では下図 10 のようなユーザー登録フォームを本手法に適用する際の適用例（一部）と検討している実装を示す．

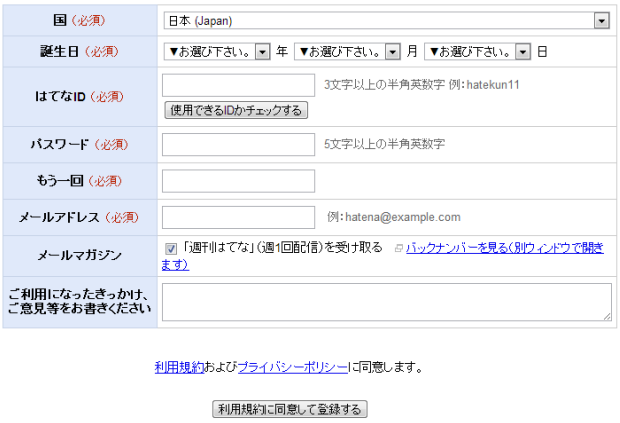

図 10　ユーザー登録フォーム（[12]より）
Figure 10 User Registration form (from [12])

現在，本手法の最終出力の形は検討中であり，定まっていない．そのため，情報抽出後に，抽出された情報を可視化し，デザイナによる修正・情報追加が行われる画面の例を下図 11 に示す．

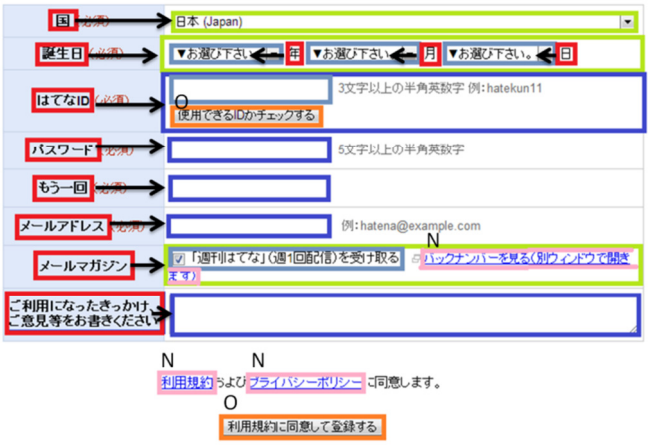

図 11　可視化・修正画面の例
Figure 11 The example of the visualization and modification screen

図 11 は意味情報を可視化し，デザイナによる修正が行われている最中の画面イメージである．デザイナはこの画面で UI エレメントとラベルの対応関係を確認し，修正する必要があれば，矢印をドラッグして修正する．例えば，図 11 の場合，本来は「はてな ID」というラベルとテキストフィールドが対応してることが望ましいが，テーブルに対応してしまっているので，矢印をドラッグし，対応関係を修正する等の作業が必要である．また，トリガーの上に N や O という文字が表示されているのは，NagigationTrigger にあたるのか OperationTrigger にあたるのかを示している．これもデザイナが修正・追加できることを想定している．

ⓒ2012 Information Processing Society of Japan

4. 議論

4.1 本研究の貢献

様々なリバースモデリングの手法が提案されているが，そのうちの大半はUIエレメントの意味付けを行わないか，もしくは手作業でエンジニアの仕事とするか，精度は気にしていない．

そこで今回は UI エレメントとその意味的な情報を結びつけることにフォーカスし，Web サイトのソースコードを解析し抽象 UI モデルを作成することのサポートをする為の手法を提案した．特に，MORE アプローチ[8]を拡張し，テーブルとフィールドセットの情報を用いることによって，計算量を減らし，情報抽出の精度を高め，制限を緩和することに貢献することを狙っている．

4.2 関連研究との差異

Yves Gaeremynck らが提案した MORE アプローチの目的は本研究の目的と似ていて，PIMA[13]と呼ばれるマルチデバイスアプリケーション開発手法において使えるような抽象モデルを，既存の HTML フォームからリバースモデリングをすることである．このアプローチでは，自動的に入力エレメントとそれに対応しているラベルを発見し，意味情報としている．このアプローチには，入力エレメント以外を扱っていないという制限や，エレメント数が増えると計算量が爆発的に増えてしまう問題，UI エレメントとラベルの対応付けを間違えてしまうことがある問題があった．本研究では，一般的な UI 構成として見られるテーブルとフィールドセットの情報を用いてこれらの問題や制限を解決することを狙っている．

L.Paganelli ら[14]は複数ページの Web アプリケーションからタスクモデルを抽出する手法を提案している．エレメント名とその種類，リンク解析のみから UI のタスクモデルを抽出している為，タスクモデルとしては，UI の MDA に使うには意味情報の解析が十分でないが，この手法では複数のページにまたがる情報を統合して一つの階層モデルを作ることができる点で大変有用である．また，タスク間の時間的制約も抽出することができ，本研究と組み合わせることで，MANTRA アプローチの AUI モデルとして使えるような抽象モデルを作成できるようになると考えられる．今後はこの手法のアイディアを本研究に取り入れていくことを考えている．

G.A.Di Lucca らは，設計モデルなしで直接コーディングされることが多く，ドキュメントが低品質もしくは不十分になりがちである，Web サイトと Web アプリケーションの理解とリエンジニアリングを狙ったリバースエンジニアリングの手法，WARE アプローチ[15]を提案している．ソースコードを静的・動的解析すると共に自動的クラスタリングを用いることで，半自動的にユースケース図とクラス図，シーケンス図を作成している．このアプローチは，大変汎用的なアプローチであり，拡張可能な為，使われている技術に関わらず様々な Web アプリケーションに適用できる．例えば，Perl によって実装されている Web アプリケーションを解析する場合には，Perl に未対応の場合でも Perl パーサのみを新しく組み込むことでアプローチを適用可能になる．しかし，動的解析においてはその作業の大部分が手作業であり，特にリンクやボタン等の UI エレメントの意味情報やリンクの遷移の解析についてはエンジニアが手動で行わなければならない為，UI の MDA による自動生成の為のリバースモデリングには向いていない．

4.3 今後の課題と展望

今後の課題や展望としては以下の内容が挙げられる．

- 従来手法との融和

今回は UI エレメントの意味付けにフォーカスし，AUI モデルを作成するのに必要な情報の一部を抽出する手法を提案した．今後は本手法の洗練と共に，従来手法と融和したアプローチを構築していくことにより，ソースコードから自動的に AUI モデルを作成することを目指していく．

- ツール化と実験的評価

本稿では方式の提案に留まるが，現在ツールとして実装の検討を進めている．動的に構築された HTML フォームにも対応できるようブラウザのエクステンションとして実装することを検討している．今後ツールを実装し，実際にソースコードから抽象 UI 情報を抽出することで，本手法の実現可能性を実装していく．

- パターン認識（画像・文字認識）技術との統合

今回は画像については取り扱わなかったが，Web や UI は画像と密接に関係している．例えば，今日ではリンクやボタンとしてや画像を使うことが一般的になっている．その為，パターン認識技術をリバースモデリングに取り入れることにより，より実用的になると考えられる．画像認識によってアイコン意味の認識や，文字認識によって画像リンクや画像ボタン上のテキスト情報を入手でき，意味付けに必要な情報の取得が促進されると考えられる．

- 非標準的な UI の意味付け

本研究では幾つかの Web・UI デザインのガイドラインを参考にし，標準的と思われる UI を対象にした手法を提案している．その為，非標準的な使い方・配置の UI を本手法に適用すると，間違った情報が抽出されてしまう可能性がある．非標準的な UI かどうかを判定し，警告を与えるか，非標準的な UI でも意味付けを行えるようなステップをアプローチに組み込むことで解決できると考えられる．

- 適用対象の拡大

本研究では UI のリバースモデリングに専念する為，最初から Web を通して外部のアプリケーションコアに接続にするようなアプリケーションを対象とした．しかし，このようなアプリケーションをリバースモデリングし，MDA を用いたマルチフロントエンド開発で他のプラットフォー

ムに移植するよりは，内部にアプリケーションコアを持つ Web アプリケーションや，Web 以外のアプリケーションを他のプラットフォームに移植する方がニーズは高いと思われる．

そこで一つの解決策として，GUI アプリケーションを Web サービスに移行する手法[16]や SOA に移行する手法[17]等と本手法を組み合わせる方法が考えられる．

また本研究では Web に特化することを前提としている為 Web に依存したテキスト情報の用い方をしているが，本手法を一部変更・拡張し，それぞれのプラットフォームに適した手法を構築することがもう一つの解決策として考えられる．

参考文献

1) OMG, OMG Model Driven Architecture http://www.omg.org/mda/
2) Goetz Botterweck : Multi Front-End Engineering, MODEL-DRIVEN DEVELOPMENT OF ADVANCED USER INTERFACES, Studies in Computational Intelligence, 2011, Vol.340/2011, pp.27-42.
3) SESSAME WG2：組み込みソフトウェア開発のためのリバースエンジニアリング,翔泳社, 2007
4) Liliana Favre ”Formalizing MDA-Based Reverse Engineering Processes”, In Proceedings of the 2008 Sixth International Conference on Software Engineering Research, Management and Applications (SERA '08), pp.153-160.
5) Artur Boronat, Jose A. Carsi, Isidro Ramos : Automatic Reengineering in MDA Using Rewriting Logic as Transformation Engine, Ninth European Conference on Software Maintenance and Reengineering (CSMR'05) , pp. 228-231
6) 市井 誠, 明神 智之, 小川 秀人: メタモデルを用いたソースコードからのモデル自動抽出手法の提案, 情報処理学会研究報告. ソフトウェア工学研究会報告 2011-SE-174(1), 1-8, 2011-10-25
7) Inês Coimbra Morgado, Ana Paiva, João Pascoal Faria : Reverse Engineering of Graphical User Interfaces, ICSEA 2011, The Sixth International Conference on Software Engineering Advances, pp.293-298
8) Yves Gaeremynck, Lawrence D. Bergman, and Tessa Lau.: MORE for less: model recovery from visual interfaces for multi-device application design. In Proceedings of the 8th international conference on Intelligent user interfaces (IUI '03), pp.69-76.
9) Microsoft, Windows User Experience Interaction Guidelines http://msdn.microsoft.com/en-us/library/aa511258.aspx
10) GNOME, GNOME Human Interface Guidelines 2.2.2 http://developer.gnome.org/hig-book/stable/index.html.en
11) usability.gov, Guidelines http://usability.gov/guidelines/index.html
12) はてな，ユーザー登録 https://www.hatena.ne.jp/register
13) Bergman, L.D, Banavar, G., Soroker, D., Sussman, J.: Combining Handcrafting and Automatic Generation of UserInterfaces for Pervasive Devices, Proceedings of CADUI III (2002), pp.155-166.
14) Scott E. Hudson, Bonnie E. John, Keith Knudsen, and Michael D. Byrne : A tool for creating predictive performance models from user interface demonstrations. In Proceedings of the 12th annual ACM symposium on User interface software and technology (UIST '99), pp.93-102
15) G.A.Di Lucca, Anna Rita Fasolino, Porfirio Tramontana : Reverse engineering Web applications: the WARE approach : Journal of Software Maintenance and Evolution: Research and Practice, 2004, vol.16, pp.71-101
16) Bo Zhang, Liang Bao, Rumin Zhou, Shengming Hu, and Ping Chen : A Black-Box Strategy to Migrate GUI-Based Legacy Systems to Web Services. In Proceedings of the 2008 IEEE International Symposium on Service-Oriented System Engineering (SOSE '08), pp.25-31.
17) Gerardo Canfora, Anna Rita Fasolino, Gianni Frattolillo, and Porfirio Tramontana. : A wrapping approach for migrating legacy system interactive functionalities to Service Oriented Architectures. J. Syst. Softw. 81, 4 (April 2008), pp.463-480.

ⓒ2012 Information Processing Society of Japan