中間表現とフレームワークを用いたWebアプリケーション変換

早川 智一[†1] 長谷川 慎哉[†1] 疋田 輝雄[†1]

現在，Rich Internet Application(RIA) と呼ばれる Web アプリケーションが急速に普及してきている．RIA は利用者の操作性・利便性を著しく向上させるが，その実装技術は Ajax を筆頭に JavaFX，Flex，Silverlight など多岐にわたっており，統一的な仕様は存在しない．統一仕様の不在は，Web 開発者に特定の実装技術の選択を強制することになり悩みの種となっている．そこで，既存の RIA を別種の RIA に変換することができれば，より広いプラットフォームで動作することが可能になり，より多くのユーザに機能を提供することができるはずである．先行研究としては，Web アプリケーションの変換に関するものが存在するが，RIA 間の変換に関しては見当たらないようである．本論文では前述の課題に対処するため，特にユーザインターフェースの変換に主眼を置きながら，中間表現とフレームワークを用いて Web アプリケーションを変換する方法について提案する．

Transforming Web Applications by Using Intermediate Representation and Framework

Tomokazu Hayakawa,[†1] Shinya Hasegawa[†1] and Teruo Hikita[†1]

By the spread of Rich Internet Application (RIA), Web applications are becoming more convenient to use. Although there are many RIA technologies such as Ajax, JavaFX, Flex and Silverlight, by far there seems no unified specification technique. Transforming an existing RIA to another is one of the solutions for it. In this paper we realize this, especially focussing on that for UI, by using an intermediate representation and a framework.

1. はじめに

昨今，インターネットの爆発的な普及により，Web アプリケーションと呼ばれる Web ブラウザを実行環境とするアプリケーションが台頭してきている．従来の Web アプリケーションは，ユーザインタフェース（以下 UI）が貧弱で操作性が劣るなどの欠点があったが，Ajax (Asynchronous JavaScript+XML) と呼ばれる Rich Internet Application（以下 RIA）の実装技術の登場によりその欠点を克服しつつある．RIA の実装技術には，Ajax 以外にも，JavaFX[7] や Flex[1]，Silverlight[5] など多数が存在するが，統一的な仕様が存在しないため，いずれかの実装技術を選択しなければならないという問題がある．そこで，既存の RIA を別種の RIA に変換することができれば，より広いプラットフォームで動作することが可能になると同時に，より多くのユーザに機能を提供できるというメリットが生まれる．

Web アプリケーションの変換に関する先行研究としては，フレームワークを用いる方法などいくつかの前例が存在するが，RIA 間の変換に関するものは見当たらないようである．本研究では，特に UI 情報の変換に主眼を置きながら, 中間表現とフレームワークを用いて Web アプリケーションを変換する方法について提案する．なお，スクリプトなどの UI 情報以外の変換については今回は副次的な扱いとしている．

本論文の構成は以下の通りである．2 節では関連研究と関連技術について述べる．3 節では本研究の中間表現とフレームワークを用いた実行例や実行結果について述べる．4 節では中間表現の概要や設計について述べる．5 節ではフレームワークの概要や設計について述べる．6 節では考察と今後の展望について述べる．

2. 関連研究と関連技術

2.1 関連研究

田枝他[12] は，PC 向けに作成されたコンテンツをテンプレートとスクリプトを用いて携帯端末用に変換する手法について提案しているが，その主な変換対象は静的コンテンツである．谷他[13] は，XSLT[11] を用いて Web アプリケーションをモジュール化しながら開発する方法について提案しているが，その主な対象は新規開発時である．Yu Ping 他[8] は，既存の Web アプリケーションを MVC モデルに変換するフレームワークについて提案してお

†1 明治大学理工学研究科
School of Science and Technology, Meiji University.

り，Marchetto 他[4] は，既存の Java アプリケーションを Web サービス型アプリケーションに変換する手法について提案しているが，RIA への変換については言及していない．

2.2 関 連 技 術

(1) XUL

XUL[6] は Mozilla Foundation によって策定された，XML を基にした UI を記述するための言語である．DHTML の要素技術 (HTML，CSS，etc) を記述言語に採用していることや，アプリケーションをコンテント，スキン，ロケールの 3 部分に分割して記述する点において本研究の中間表現と類似性が見られるが，Web アプリケーションの動作・振舞いを記述することができない点において異なる．

(2) XForms

XForms[10] は W3C によって策定された，UI を定義するための XML の仕様である．外観の指定に CSS を用いる点において本研究の中間表現と類似性が見られるが，Web アプリケーションの動作・振舞いを記述することができない点において異なる．

(3) Velocity

Velocity[2] は Apache Software Foundation によって開発されている Java ベースのテンプレートエンジンである．元となるテンプレートを用意し，テンプレート中のシンボル（文字列）を置換して成果物を出力する点において本研究のフレームワークと類似性が見られるが，基本的に単純置換である点や，Web アプリケーションの動作・振舞いを記述することができない点において異なる．

(4) XSLT

XSLT は W3C によって策定された，XML 文書を他の XML 文書に変換するための仕様である．変換規則を利用者が記述し変換を行う点において本研究のフレームワークと類似性が見られるが，XSLT が XML に変換規則を記述するのに対し，本研究では Java のクラス階層を用いて変換規則を表現する点において異なる．

3. 実 行 例

図 1-**図 4** は本フレームワークと中間表現を用いた変換の例である．まず，変換前アプリケーション（図 1 と**図 3**）を見ると，ボタンとテキストと画像の 3 種類のウィジェットが存在する．これらは，Web ページにおいて頻繁に使用されるものである．ボタンにはクリックされた場合のイベントが設定されている．変換の前後（図 1 と**図 2** の間，図 3 と図 4 の間）で，空白やレイアウトにおいて若干の変化が見られるが，これは実装技術ごとに配置など

図 1 変換前 (HTML) アプリケーション
Fig. 1 Sample (HTML) Application.

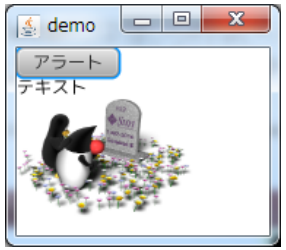

図 2 変換後 (JavaFX) アプリケーション
Fig. 2 Translated (JavaFX) Application.

に関するポリシーが異なるためである．また，図 3 と図 4 のボタンのラベル名にも相違（「OK」と「了解」）が見られるが，これはそれぞれの標準 API を用いて変換を実現していることに起因するものである．

図 5-**図 7** は変換の過程のソースコードである．ソースコードは紙面の都合上関連部分のみを抜粋・整形してある．中間表現（**図 6**）を見ると，入力がより一般化された形式で保存されていることがわかる．例えば，XHTML（図 5）におけるボタンは`<input type="button">`で表現されるが，中間表現においては`<button>`で表現されている．さらに XHTML 中の ID は一意であるという性質を利用して，中間表現の中で振舞い部からウィジェットを参照する際の識別子として使用されている．変換後の JavaFX のソース（図 7）を見ると，異なる形式に変換されていることがわかる．特徴的な部分として，`VBox` という指定が挙げられる．これは，自身の `content` 内に指定された要素を縦方向に並べるための命令であり，XHTML におけるブロック要素の配置におよそ対応している．

4. 中 間 表 現

4.1 概　　要

中間表現は，文書フォーマットとして XML を使用し，要素`<application>`をルートに持つ．内部は (1) ウィジェットに関する情報，(2) スタイルに関する情報，(3) 振舞いに関する情報に大別される（**図 8**）．ウィジェット部は要素`<widgets>`を親に持ち，テキストボックスなどの UI のコントロールに関する情報を保持している．スタイル部は要素`<styles>`を親に持ち，フォントのサイズや色などの見栄えに関する情報を保持している．振舞い部は要素`<behaviors>`を親に持ち，クリック時の動作など振舞いに関する情報を保持している．

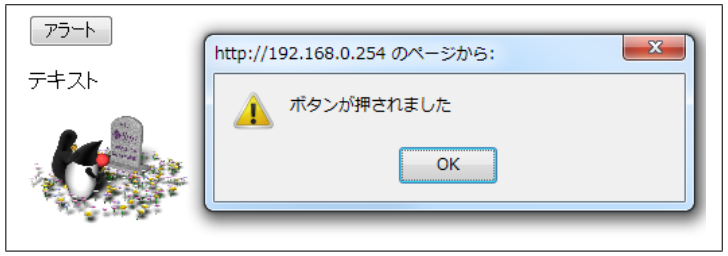

図 3 変換前 (HTML) アプリケーション（クリック後）
Fig. 3 Sample (HTML) Application after Clicking.

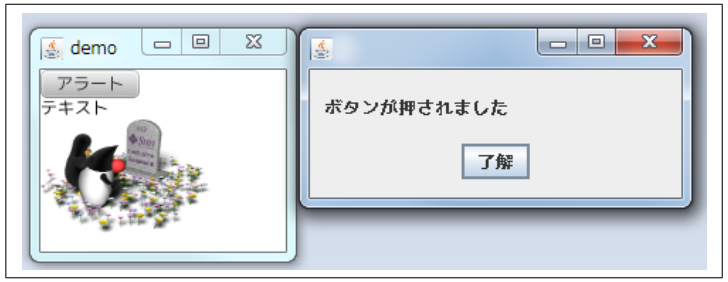

図 4 変換後 (JavaFX) アプリケーション（クリック後）
Fig. 4 Translated (JavaFX) Application after Clicking.

4.2 設　　計

中間表現を設計する際には様々な方式が考えられるが，本研究では汎用性を考慮し XML を採用した．中間表現の内部は前述の通り 3 つに分かれているが，これはコンテンツとレイアウト等の分離を促進し，後述のフレームワークにおける変換容易性を高めるためである．また中間表現は，様々な RIA への変換可能性を高めるため，主として多くの RIA の実装に共通する要素から構成される．代表的なウィジェットを**表 1** に，振舞い部でサポートされるマウスイベントとキーイベントを**表 2** と**表 3** にそれぞれ示す．なお，現時点の中間表現は基本的な表現能力しか保持していないが，この範囲においてはフレームワークに手を入れることなく容易に使用することができる．また，より高い表現能力が必要な場合には，後述の方法を用いて拡張することが可能である（5.5 節）．

```
<html>
    <head>
        <script type="text/javascript">
            function init(){
                var b1 = document.getElementById("b1");
                b1.addEventListener('click',func1,false);
            }
            function func1(){
                alert("ボタンが押されました");
            }
        </script>
    </head>
    <body onload="init();">
        <div>
            <input id="b1" type="button" value="アラート"/>
            <p>テキスト</p>
            <img src="SunRIPsmall.png" width="120" height="80"/>
        </div>
    </body>
</html>
```

図 5 変換前 (HTML) アプリケーションのソース
Fig. 5 Source of Sample (HTML) Application.

```
<application>
    <widgets>
        <button id="b1" value="アラート"/>
        <text>テキスト</text>
        <image src="SunRIPsmall.png" width="120" height="80"/>
    </widgets>
    <styles></styles>
    <behaviors>
        function init(){
            b1.addEventListener('click',func1,false);
        }
        function func1(){
            alert("ボタンが押されました");
        }
    </behaviors>
</application>
```

図 6 中間表現のソース
Fig. 6 Source of Intermediate Representation.

5. フレームワーク

5.1 概　　要

フレームワークは実装言語として Java 言語を使用し，Web アプリケーションから中間表

```
var b1: Button;
var stage: Stage = Stage{
    scene: Scene{
        content: VBox{
            content: [
                b1 = Button{
                    text: "アラート";
                    onMouseClicked: function(e:MouseEvent): Void{
                        Alert.inform("ボタンが押されました");
                    }
                },
                Text{content: "テキスト";},
                ImageView{
                    image: Image{
                        url: "{__DIR__}SunRIPsmall.png";
                        width: 120;
                        height: 80;
                    }
                }
            ]
        }
    }
}
```

図 7 変換後 (JavaFX) アプリケーションのソース
Fig. 7 Source of Translated (JavaFX) Application.

表 1 中間表現でサポートされる UI 要素
Table 1 UI Elements in Intermediate Representation.

要素名	備考
button	ボタン
radiobutton	ラジオボタン
image	画像
label	ラベル
checkbox	チェックボックス
textbox	テキストボックス
passwordbox	パスワードボックス
combobox	コンボボックス
listbox	リストボックス

表 2 中間表現でサポートされるマウスイベント
Table 2 Mouse Events in Intermediate Representation.

イベント	発生条件
onclick	マウスクリック時
onmousedown	マウス押下時
onmouseup	マウスを離した時

現への変換と，中間表現から Web アプリケーションへの変換をサポートする（**図 9**）．さらに，フレームワークは AbstractFactory や Visitor などのデザインパターン[3] を使用することで，設計と実装の分離を促進し，自身の再利用性を高めるとともに拡張性を担保している．ただし，現時点で使用できる形式は，入力としての Ajax(DHTML) と出力としての

表 3 中間表現でサポートされるキーイベント
Table 3 Key Events in Intermediate Representation.

イベント	発生条件
onkeypress	キー打鍵時
onkeydown	キー押下時
onkeyup	キーを離した時

表 4 使用している要素技術とバージョン
Table 4 Supported Technologies and Version.

要素技術	バージョン
XHTML	1.1
CSS	2.1
JavaScript	1.5
XML	1.0
Java	1.6
DOM	Level2

```
<?xml version="1.0" encoding="UTF-8"?>
<application>
    <widgets>
        <!-- UI コンポーネントに関する情報 -->
    </widgets>
    <styles>
        <!-- フォントや色などに関する情報 -->
    </styles>
    <behaviors>
        <!-- クリック時の動作など
             振る舞いに関する情報 -->
    </behaviors>
</application>
```

図 8 中間表現概要
Fig. 8 Skeleton of Intermediate Representation.

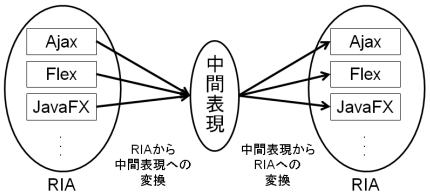

図 9 フレームワーク概要
Fig. 9 Overview of Framework.

JavaFX に留まっている．これらを選択した理由は，前者は既存の資産の多さに配慮したためであり，後者は Java の持つベンダー非依存性に注目したためと，異なる形式への変換可能性を示すためである（Ajax，Flex，Silverlight は XML とスクリプト言語を用いて構成されており類似性があるが，JavaFX は全く異なる形式であるため）．

5.2 設　　計

一般にフレームワークを設計する際には，言語処理系などに関する技術や技法が駆使されることが多いが，これらは Web アプリケーションの開発者にとって敷居が高く，現実的な選択肢とは言えない．本研究では Web アプリケーション開発者が容易に習得・利用できるよう，(X)HTML，CSS，DOM，XML，Java などの基本的な技術を軸として設計を行った．

本研究におけるフレームワークの最大の特徴は，変換過程における一連のデータ構造を DOM ツリーとして扱う点にある（**図 10**）．これにより，主に DOM の知識があれば変換や

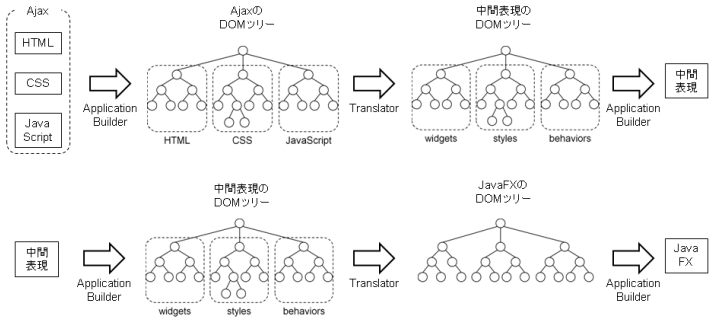

図 10　変換の過程
Fig. 10　Process of Translation.

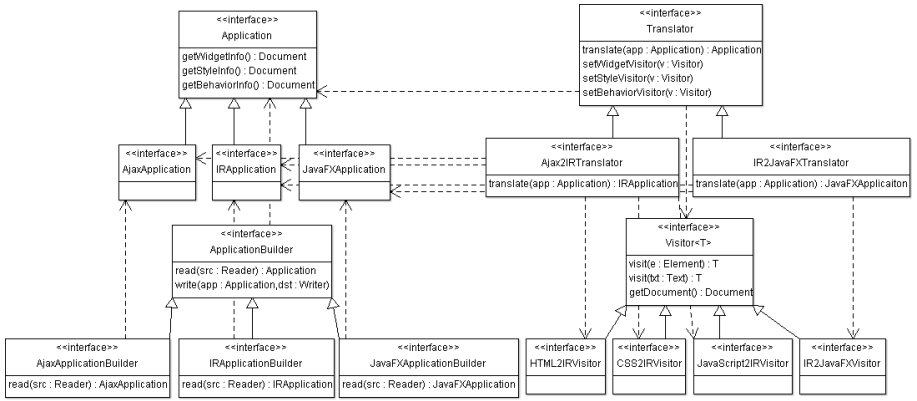

図 11　フレームワークのクラス図
Fig. 11　Class Diagram of Framework.

置換を行うことが可能になり，データ形式として XML の使用が容易になる．

図 11 にフレームワークの概要クラス図を示す．`Application` は，アプリケーションを表すインターフェースである．本実装では，それぞれ Ajax，中間表現，JavaFX に対応するサブインターフェースが存在する．`Translator` は，アプリケーション間の変換器を表すインターフェースである．`Translator` の内部は後述の 3 種類の `Visitor` で構成され，これらはそれぞれウィジェット，スタイル，振舞いの変換に対応する．本実装では Ajax から中間表現への変換を行うものと，中間表現から JavaFX への変換を行うものが存在する．`Visitor` は，DOM ツリーの中を縦断しながら処理を行う際に使用されるインターフェースである．DOM ツリーで要素が発見されるたびに対応する `Visitor` に対してコールバックがかかり，これを再帰的に繰り返すことで最終的な形式が得られる．

5.3　変　　換

フレームワークによる変換の概要について以下に述べる．

(1)　入力からの DOM ツリーの構築

入力ファイルは，フレームワークのビルダによって DOM ツリーの形で読み込まれる．中間表現は，既に述べたように 3 つの部分によって構成されているため，この DOM ツリーも内部に 3 つの部分木を保持している．Ajax アプリケーションを入力とする場合，これはそれぞれ (X)HTML，CSS，JavaScript を木構造に変換したものである．

(2)　要素の置換

構築された DOM ツリーは，フレームワークに予め実装されているルールに従って置換が行われる．DOM ツリーは 3 つの部分木で構成されているため，置換にも 3 つの `Visitor` が使用される．この置換においては，Visitor パターンを用いてデータ構造と処理の分離が実現されているため，利用者が必要に応じて置換ルールを変更したり拡張することができる．

(3)　DOM ツリーのファイルへの出力

変形が完了した DOM ツリーは，フレームワークのビルダによってアプリケーション固有の形式で出力される．フレームワークから見ると，入力形式から中間表現への変換と中間表現から出力形式への変換は，木の変形という観点において等価である．

5.4　実　　装

5.4.1　入　　力

今回実装した処理系は，Ajax(DHTML) アプリケーションを入力として与えることができ，**表 4** に示したバージョン，およびそのサブセットに対応している．主な要素技術への対応については以下の通りである．

(1)　(X)HTML

表 5 は本処理系が対応している XHTML のモジュール[9] である．紙面の都合上，要素や属性の個々の説明は割愛する．対応しているものは○，部分的に対応しているものは△，対応していないものは×で示してある．

(2)　CSS

表 5　サポートしている XHTML のモジュール
Table 5　Supported XHTML Modules.

モジュール名	対応状況
Structure Module	△
Text Module	○
Hypertext Module	△
List Module	△
Object Module	×
Presentation Module	○
Edit Module	△
Bidirectional Text Module	×
Forms Module	△
Table Module	△
Image Module	○
Client-side Image Map Module	×
Server-side Image Map Module	×
Intrinsic Events Module	△
Metainformation Module	×
Scripting Module	△
Stylesheet Module	○
Link Module	○
Base Module	×

表 6　変換後に使用される JavaFX コントロール
Table 6　Supported JavaFX Controls.

JavaFX コントロール名	対応する中間表現
Button	button
CheckBox	checkbox
ChoiceBox	combobox
Label	label
ListView	listbox
PasswordBox	passwordbox
RadioButton	radiobutton
TextBox	textbox

本処理系はおよそ任意の CSS に対応している．これは，入力を構文解析する際にセレクタ等の情報がそのまま保存されるためである．ただし，変換後の UI の再現性は，変換後のアプリケーションの実装技術の CSS への対応度に依存している．

(3)　JavaScript

現在の処理系は中間表現の振舞い部がサポートする範囲での，比較的単純な JavaScript の変換に対応している．

5.4.2 出　　力

現在の処理系は，変換結果として JavaFX アプリケーションを出力する．**表 6** に出力で使用される代表的な UI コントロールを示す．

5.5 拡 張 方 法

本フレームワークの標準の変換動作を拡張および変更する場合には，`Translator` または `Visitor` を継承し，必要なメソッドをオーバーライドすればよい．また，追加の UI 部品（例：カレンダーなど）をサポートする必要が生じた場合には，中間表現に新しい要素を追加し，`Visitor` を継承した上で必要な処理を追加すればよい．

6. おわりに

本研究の中間表現とフレームワークを用いることで，Web アプリケーション開発者が標準的に保持している技術や知識を用いて，既存の Web アプリケーション，特にテキストや画像から構成されるものを別種の RIA に変換することが可能であるという一定の成果を示すことができた．これにより，画面数や部品数の多いアプリケーションを変換する際の省力化が期待できる．一方で，UI の振舞いに関する変換についてはさらなる改良が必要で，例えば JavaScript の扱いにおいては，現時点では処理系の対応範囲や利用者の自由度が少なく，自由に変換規則を定義できるとは言えない部分がある．

今後は他の実装技術のサポートを追加するとともに，本フレームワークおよび中間表現に対する評価方法を検討し，本システムの適用可能範囲や有用性等についての評価が急務であると考えている．また，変換処理の詳細についても別の機会に報告したい．

参 考 文 献

1) Adobe: Flex, http://www.adobe.com/products/flex/.
2) Apache Software Foundation: Velocity, http://velocity.apache.org/.
3) Gamma, E., Helm, R., Johnson, R. and Vlissides, J.M.: *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional (1994).
4) Marchetto, A. and Ricca, F.: Transforming a Java Application in an Equivalent Web-Services Based Application: Toward a Tool Supported Stepwise Approach, *Web Site Evolution* (2008).
5) Microsoft: Silverlight, http://www.microsoft.com/silverlight/.
6) Mozilla Foundation: XUL, https://developer.mozilla.org/En/XUL.
7) Oracle: JavaFX, http://javafx.com/.
8) Ping, Y., Kontogiannis, K. and Lau, T.C.: Transforming Legacy Web Applications to the MVC Architecture, *Eleventh Annual International Workshop on Software Technology and Engineering Practice* (2004).
9) W3C: Modularization of XHTML, http://www.w3.org/TR/xhtml-modularization/.
10) W3C: XForms, http://www.w3.org/TR/xforms/.
11) W3C: XSLT, http://www.w3.org/TR/xslt.
12) 田枝覚，渡部聡彦，中川裕志：多様な携帯端末に適応可能なコンテンツ中間記述，技術報告，情報処理学会研究報告情報学基礎（FI） (2002).
13) 谷義人，満田成紀，鯵坂恒夫：XSLT を用いたモジュール性の高い WEB アプリケーションの開発手法とフレームワークの提案，技術報告，情報処理学会研究報告ソフトウェア工学（SE） (2004).