情報処理学会論文誌 Vol.53 No.11 2571–2583 (Nov. 2012)

推薦論文

中間表現とフレームワークを用いたWebアプリケーションのメンテナンス法の提案と評価

早川 智一[1,a)] 長谷川 慎哉[1,b)] 吉賀 祥太[1,c)] 疋田 輝雄[1,d)]

受付日 2012年3月31日, 採録日 2012年9月10日

概要：RIA（Rich Internet Application）の移植が困難であるという問題が顕在化してきている．この移植性の欠如は，RIA の実装技術が仕様変更されたり廃止されたりした場合に問題となる．なぜならば，既存の RIA を継続利用するために他の RIA 技術への移植が必要となるが，移植にはコストがかかるためである．我々は，この問題を解決するために中間表現とフレームワークとを用いる方法を提案し，両者を総括して *Web-IR* と名付けた．中間表現は，XML をベースとする拡張可能な表現形式であり，RIA の情報を 4 種類（メタ情報・ウィジェット情報・スタイル情報・振舞い情報）に分割して保持する．フレームワークは，Java をベースとする拡張可能な部品群であり，RIA の変換処理系を実装するための機能を利用者に提供する．我々は，Web-IR を用いてプロトタイプを開発し，中間表現の表現能力とフレームワークの工数削減率とを評価した．評価の結果は，中間表現が主要な RIA の 8 割以上の UI（User Interface）情報を表現可能であり，フレームワークが RIA の変換処理系の開発工数を 2/3 以下に削減することを示した．我々は，この評価を通じて Web-IR が前述の問題解決に有用であるという結論を得た．

キーワード：Web アプリケーション，RIA（Rich Internet Application），中間表現，フレームワーク，工数

Proposal and Evaluation of Intermediate Representation and Framework for Maintaining Web Applications

Tomokazu Hayakawa[1,a)] Shinya Hasegawa[1,b)] Shota Yoshika[1,c)]
Teruo Hikita[1,d)]

Received: March 31, 2012, Accepted: September 10, 2012

Abstract: A compatibility problem, which is related to RIAs (Rich Internet Applications), has arisen: RIA technologies are hardly compatible with each other. If a RIA technology becomes obsolete, developers need to redevelop their existing Web applications by using another RIA technology, but it obviously consumes time and costs. We propose a method named *Web-IR*, which uses an intermediate representation (IR) and a framework, to solve the problem. We provide IR, which is based on XML, to represent RIA information, and it consists of the following four parts: meta information, widget information, style information, and behavior information. We also provide the framework, which is written in Java, to provide an efficient way to implement RIA translators. We developed several prototypes by using Web-IR, and evaluated the effectiveness of both IR and the framework. The result of the evaluation shows that IR can represent over 80% of UI (User Interface) information of major RIAs, and the framework can reduce production costs for translator developments to 2/3 or less. Through the evaluation, we conclude that our proposed method, Web-IR, can solve the problem sufficiently.

Keywords: Web application, RIA (Rich Internet Application), intermediate representation, framework, development cost

[1] 明治大学理工学研究科
School of Science and Technology, Meiji University, Kawasaki, Kanagawa 214–8571, Japan
[a)] t_haya@cs.meiji.ac.jp
[b)] s-hase@cs.meiji.ac.jp
[c)] yoshika@cs.meiji.ac.jp
[d)] hikita@cs.meiji.ac.jp

本論文の内容は 2011 年 11 月のマルチメディア通信と分散処理研究会にて報告され，同研究会主査により情報処理学会論文誌ジャーナルへの掲載が推薦された論文である．

1. はじめに

1.1 概要

RIA（Rich Internet Application）の普及とともに，その移植が困難であるという問題が顕在化してきている．これは，RIA の実装技術（以下，RIA 技術[*1]）間に互換性がないことに起因する [14]．しかし，この問題の解決策はまだ見当たらないようである（2 章）．そこで我々は，中間表現とフレームワークとを用いる方法を提案し，その評価結果について報告する．

1.2 RIA の非互換性が及ぼす影響

1.2.1 移植への影響

RIA の移植性の欠如は，既存の RIA の開発に使用した RIA 技術が廃れたり仕様変更されたりした場合に問題となる．なぜならば，その RIA を継続利用するために他の RIA 技術への移植が必要になるためである[*2]．移植にはコストがかかるため，大規模な RIA ほどこの問題が看過できないものとなる．

以下に，RIA の移植性の欠如に関する実例を 2 つあげる：(1) JavaFX は，言語仕様の変更によりバージョン 2.0 を境に互換性が失われた；(2) Adobe Systems は，モバイルデバイス用の Flash Player の開発を，バージョン 11.1 までで中止すると発表した [2]．前者は，同じ RIA 技術であっても仕様変更で移植が必要になる場合があることを示しており，後者は，既存の RIA 技術が実際に開発中止になる場合があることを示している．

これらのことから，既存の RIA 資産を継続利用するために，少ないコストで速やかに RIA を移植する方策が求められている．

1.2.2 新規開発への影響

RIA の移植性の欠如は，RIA の新規開発において次の問題を引き起こす：(1) 開発ごとに最適な RIA 技術の検討・選定が必要になることによるコストの増加；(2)（1）を回避するために組織内で使用する RIA 技術を統一（以下，標準化）する場合の RIA の永続性に関するリスクの増加．前者は，標準化を行っていない組織で発生する．なぜならば，開発要件によって最適な RIA 技術は異なるため，開発ごとに RIA 技術を検討・選定する必要に迫られるためである．しかし，多数の RIA 技術が存在するため，この作業にも多くのコストが必要になる．後者は，前者の問題を解決するために採られる施策である．これにより，開発ごとの RIA 技術の選定が不要となりコストの削減につながる．一方で，特定の RIA 技術を標準とすることは，RIA の永続性に関するリスクを高めることになる（1.2.1 項）．

これらのことから，RIA の新規開発においては次のことが求められている：(1)（標準化をしていない組織では）開発ごとの RIA 技術の検討・選定コストを減らすこと；(2)（標準化をしている組織では）標準とした RIA 技術で開発した RIA の永続性を高めること．

1.3 提案概要

我々は，1.2 節の問題を解決するために，中間表現（IR：Intermediate Representation）とフレームワークとを用いる方法を提案し（3 章），両者を総括して *Web-IR* と名付けた．Web-IR の核となるアイデアは次のとおりである：(1) 任意の RIA 情報を記述可能な IR（4 章）と，IR と既存の RIA 技術との変換を支援するフレームワーク（5 章）とを提供する；(2-a) 移植の場合は，フレームワークを用いて既存の RIA を IR を経由して異なる RIA に変換することでコストを低減する；(2-b) 新規開発の場合は，RIA を IR で記述してからフレームワークを用いて IR を RIA に変換することで永続性を高める．

我々は，Web-IR の有用性を評価するために，IR の表現能力とフレームワークの工数—システム開発における人的コスト—の削減率とを測定した（6 章）．評価結果は，以下のことを示した：(1) IR が，主要な RIA の 8 割以上の UI（User Interface）情報を表現できる；(2) フレームワークが，RIA の変換処理系の実装にかかる工数を 2/3 以下に削減する．

1.4 本論文の構成

2 章では，関連研究と関連技術とを紹介する．3 章では，提案手法（Web-IR）について説明する．4 章と 5 章とでは，IR とフレームワークとについて概説する．6 章では，Web-IR の評価結果を報告する．7 章では，結論と今後の展望とを述べる．なお，設計や実装の詳細は文献 [7], [8], [9], [10], [11], [12] を参照されたい．

2. 関連研究

本章では，我々の研究と関連する研究との類似性や差異について述べる．なお，我々と目的が同一の研究は見当たらなかったため，RIA のフレームワークや変換処理系などの，比較的近い研究について言及する．

2.1 RIA 関連フレームワーク

松塚 [13] は，業務アプリケーション用の Ajax フレームワークを提案し，次のように述べている：(1) 業務アプリケーションは，規模（画面数・部品数・コード量）が大きくなる傾向にあり，それらを効率良く記述・管理する機構が必要である；(2) 業務アプリケーションの UI を構成するウィジェット—ボタンなどの UI 部品—には，（たとえそ

[*1] 厳密には，「RIA 技術」は RIA の実装技術を指し，「RIA」は RIA 技術で実装したアプリケーションを指す．本論文では，誤解のおそれがない限り，双方の意味で「RIA」を用いる．

[*2] 移植せずに継続利用する選択肢もあるが，セキュリティ上の理由などから移植が必要となる場合が多い．

れが単なるテキストフィールドであっても）個々にカスタマイズ性が求められる．

これらの主張は，我々の IR とフレームワークとの提案理由に合致する．すなわち我々は，移植性・汎用性を保ちながら RIA を表現するために IR が必要であり，カスタマイズ可能な変換を実現するために柔軟に動作を変更できるフレームワークが必要であると考える．

松塚のフレームワークと我々のフレームワークとは，RIA を効率的に記述する点に類似性があるが，対象とする実装技術が異なる—松塚は Ajax のみを対象としているが，我々は Ajax を含む任意の RIA 技術を対象としている．

2.2 商用サービス

Adobe Systems は，Adobe Flash Professional（FLA）ファイルを HTML5 に変換する Wallaby [1] と呼ばれる処理系をリリースしており，これにより Flash 実行環境を備えないデバイスへの Flash コンテンツの移植が容易になると述べている．

Google は，Flash（SWF）ファイルを HTML5 に変換する Swiffy [5] と呼ばれる処理系をリリースしており，これにより iPhone や iPad などの Flash Player を持たない環境への Flash コンテンツの移植が容易になると述べている．

これらの処理系の存在は，RIA の移植手段が必要であるという我々の主張と合致する．両社は既存の Flash コンテンツを HTML5 に変換することで再利用を促しているが，我々は Flash 以外の RIA にも再利用を促す手段が必要であると考える．

これらの処理系と Web-IR とは，RIA を別形式の RIA に変換する点に類似性がある．一方で，これらが特定の RIA 技術間で直接変換を行うのに対し，Web-IR は複数の RIA 技術間で IR を経た間接変換を行う点で異なる．

2.3 その他

Ping ら [16] は，Web アプリケーションの移植性の問題を解決するために，Web アプリケーションを MVC アプリケーションに変換する Java ベースのフレームワークを提案している．

この研究と Web-IR とは，Web アプリケーションを変換し移植性を高める点に類似性があるが，対象の技術が異なる—この研究は RIA ではない Web アプリケーションのみを対象としているが，Web-IR は複数の RIA 技術を対象としている．

3. 提案手法：Web-IR

3.1 目的

Web-IR の目的は，1.2 節で述べた問題を解決する手段を提供することにある．我々は，Web-IR の想定利用者（以下，利用者）を標準的な Web 開発者—Ajax などの RIA の知識を持つが言語処理系などの専門知識は持たない開発者—と定め，Web-IR が満たすべき要件を次のように定めた：(1-a) 移植の場合は，手作業と比較して少ない工数で RIA を移植できること；(1-b) 新規開発の場合は，RIA 技術の選定工数を減らし，開発する RIA の永続性を高めることができること；(2) 言語処理系などの専門知識を持たない利用者でも使用できること；(3) 利用する組織の慣習・用途などに応じたカスタマイズができること．

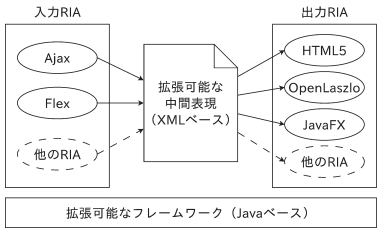

図 1 Web-IR の概要
Fig. 1 Overview of Web-IR.

3.2 提案手法

我々は，3.1 節の目的を達成するために以下の手法を提案する（**図 1**）：(1) 主要な RIA 技術が共通して備える機能を持つ拡張可能な IR を提供する；(2) IR から RIA（およびその逆方向）への変換を支援するために，デザインパターンを用いて設計した拡張可能なフレームワークを提供する；(3) 必要に応じてエミュレーションライブラリを提供する．

我々は，この手法を次のように適用することで，上述の目的を達成できると考える（適用判断については 3.3 節を参照）：(1) 移植の場合は，既存の RIA を IR を経由して別の RIA に変換する；(2) 新規開発の場合は，IR を用いて開発を行った後に，IR を RIA に変換する．ここで，変換に必要な処理系はフレームワークを用いて実装する．

我々が，Web-IR をこのように設計した理由は次のとおりである：(1) 各 RIA 技術が持つ機能には差異があるが，基本的なウィジェット（例：ボタン・テキスト）には大差がない；(2) IR の標準の表現能力を各 RIA 技術に共通する範囲に限定することで，他の RIA 技術への移植が容易になる；(3) 変換対象の RIA が独自の部品（例：カスタムウィジェット）を使用している場合でも移植が可能になる；(4) デザインパターンは広く知られており，言語処理系などの専門知識と比較してより多くの利用者が理解していると期待できる；(5) すべての機能要件に対応しようとすると，重厚長大な IR とフレームワークとになってしまい，学習工数や開発工数の増加を招く[*3]．

[*3] 我々の経験では，開発する RIA に求められる要件は組織や業務によって異なることが多く，あらかじめ機能を完備した IR やフレームワークを実装・提供することは難しい．

3.3 適用判断

我々は，Web-IR の適用は，次のいずれかの場合に特に有効であると考える：(1) すでに多くの RIA の実装に用いた RIA 技術を移植する場合；(2) 標準化（1.2.2 項）で選定した RIA 技術で新規開発を行う場合．なぜならば，これらの場合には，Web-IR を適用することで削減できる工数が，Web-IR の適用に必要な工数を上回ることが期待できるためである．

ここで重要なことは，Web-IR の部分適用も検討すべきという点である．すなわち，IR は 4 種類の情報からなるが（4.2 節），必ずしもこれらすべてを活用する必要はない．したがって，たとえば UI 情報のみを IR 化し，それ以外の情報は，IR を RIA に変換してから従来どおり手動で実装するという方法も有効である．この場合，UI 情報は汎用化されるため，将来，当該 RIA を移植する際に同一の UI を容易に再現できるという利点が得られる[*4]．

3.4 他の手法との比較

我々の目的を達成するために，RIA 技術間の直接変換を行う処理系（以下，トランスレータ）を実装・提供することも考えられるが，次の理由から現実的ではない：(1) n 種類の RIA 技術に対して，最大で $n \times (n-1)$ 通りの処理系が必要になる；(2) 変換対象の RIA 技術によっては，変換時のセマンティクスが一意に定まらず自動変換が困難である[*5]；(3) 利用者が動作をカスタマイズする場合に言語処理系の知識が必要になる．

トランスレータ方式と比較すると，我々の方式には以下のメリットがある：(1) n 種類の RIA 技術に対して処理系の数が最大で $2n$ 通りで済む；(2) 変換対象の RIA 技術のコンテキストに応じた変換が可能になる；(3) 新しい RIA 技術 X が登場した際に，IR から X への処理系を追加するだけで，すでに IR に変換されている既存の RIA を X に変換できる；(4) 利用者が，デザインパターンの知識のみで動作をカスタマイズできる．

一方で，デメリットとしては以下の点がある：(1) 変換できる情報量（例：対応ウィジェット数）が IR によって制限される；(2) 変換能力がフレームワークによって制限される．これらについては，IR とフレームワークとを拡張可能（4.4 節・ 5.5 節）とすることで対処する．

3.5 Web-IR を用いたプロトタイプの実装

我々は，移植性の評価のために，RIA の変換を行う処理系（以下，プロトタイプ）を Web-IR を用いて実装した．プロトタイプは，以下の RIA の入出力に対応する：(入力) Ajax と Flex 4.5；(出力) HTML5 と OpenLaszlo 4.9 と JavaFX 2.0．これらの選定理由は次のとおりである：(入力) RIA で最も使用されている Ajax と 2 番目に使用されている Flex とは入力として妥当である[*6]；(出力) HTML5 と OpenLaszlo と JavaFX とはマルチプラットフォームで動作し，現時点で他の技術によって置き換えられる予定がないため，ベンダロックイン—システムのライフサイクルが特定のベンダに拘束されること—を避ける観点から出力として妥当である[*7]．

図 2 Internet Explorer 9 による Web ページの表示（変換前）
Fig. 2 Web page rendered by Internet Explorer 9 (before transformation).

図 3 Firefox 6 による Web ページの表示（変換前）
Fig. 3 Web page rendered by Firefox 6 (before transformation).

3.6 プロトタイプによる変換例

図 2 と**図 3** とは，変換対象のページを Internet Explorer 9 と Firefox 6 とでそれぞれ表示したものである．このページをプロトタイプを用いて JavaFX に変換したものが**図 4** である．これらの図より，Web-IR は，ウィジェットの大きさやウィジェット間の空白などの僅かな UI の差異のみで，RIA を異なる RIA に変換する処理系を実装する能力を利用者に提供することが分かる．

図 5 は，この変換の前半部分（入力から IR）を行うプログラム片である．特徴として，Web-IR の各部品を呼び出す際に，クラス名ではなく RIA 技術名を用いるように設計した点をあげる．これにより，利用者のコードと Web-IR の部品との結合が疎になり，プログラムの拡充改善が容易になる（5.1 節）．

[*4] 業務アプリケーションの場合，UI を変更するとエンドユーザ（必ずしも計算機の専門家とは限らない）の再教育が必要になるため，なるべく UI を同一に保つことが重要である．
[*5] たとえば，Ajax では`<div>`や`<span>`が多用されるが，これらは本来は範囲を示すための汎用タグであり特別な意味を持たない．そのため，これらのタグを変換する際にはコンテキストに応じた変換が必要になる．
[*6] Google Insights for Search（http://www.google.com/insights/search/）のクエリ数調査による．
[*7] Ajax は HTML5 で代替される可能性があり，Flex は 1.2.1 項で述べたリスクがあるため出力から除外した．

図 4 Web-IR で変換した JavaFX ページ（変換後）
Fig. 4 JavaFX page transformed by Web-IR (after transformation).

```
/* Ajaxを読み込むためのApplicationReaderインタフェースのインスタンスを取得する */
ApplicationReader src = ApplicationReaders.newInstance("Ajax");
/* 引数として渡されたURIからAjaxアプリケーションを読み込む */
Application app1 = src.read("file://...(URI)");
/* 入力（Ajax）を出力（IR）に変換するTranslatorインタフェースのインスタンスを取得する */
Translator translator = Translators.newInstance("Ajax", "IR");
/* 入力（Ajax）を出力（IR）に変換する */
Application app2 = translator.translate(app1);
/* 出力（IR）を書き込むためのApplicationWriterインタフェースのインスタンスを取得する */
ApplicationWriter dst = ApplicationWriters.newInstance("IR");
/* 引数として渡されたURIにIRを書き込む */
dst.write(app2, "file://...(URI)");
```

図 5 Web-IR の使用例（Ajax から中間表現（IR）への変換）
Fig. 5 Usage of Web-IR (transformation from Ajax to Intermediate Representation).

4. 中間表現

4.1 概要

中間表現（IR：Intermediate Representation）は，RIA の情報を汎用的に表すための表現形式である．IR は，各 RIA 技術に共通する機能を持つように設計されている[*8]が（3.2 節），任意の要素を追加して機能を拡張することも可能である（4.4 節）．この拡張性は，以下の場合に有用である：(1) 標準の IR では表現できない RIA を記述する場合；(2) Web-IR の利用者の環境に固有のウィジェットなどを定義する場合．

IR の記述言語には，以下の理由から XML を採用した：(1) テキストベースであり特定のベンダに依存しない；(2) 多くの RIA 技術が記述言語に XML（またはその亜種）を用いており，利用者の再学習が不要である．

IR は内部が 4 種類に分割されており，それぞれがメタ情報・ウィジェット情報・スタイル情報・振舞い情報を表す（**図 6**）．我々が，IR をこのように分割した理由は次のとおりである：(1) 多くの RIA 技術が各情報を分割して記述する仕様になっている；(2) 各情報を分割保持することで情報の部分利用が可能になる．特に，後者は以下の理由から重要である：(1) 入力の RIA のすべての情報の変換は困難な場合でも，情報を分割することで部分変換（例：UI のみの変換）が可能になる；(2) 最初は UI 変換のみを実装し，次に振舞いの変換を実装するなどの段階的な開発が可能になる．

*8 たとえば，IR が標準で表現できない機能としてカレンダやグラフなどがある．これらは，すべての RIA に共通するウィジェットではないため，標準ではサポートされない．これらのウィジェットへの対応が必要になった場合には，IR の拡張機能（4.4 節）を用いて，当該ウィジェットのサポートを追加する必要がある．

```
<?xml version="1.0" encoding="UTF-8"?>
<application>
  <meta>
    <!-- メタ情報 -->
  </meta>
  <widget>
    <!-- ウィジェット情報 -->
  </widget>
  <style>
    <!-- スタイル情報 -->
  </style>
  <behavior>
    <!-- 振舞い情報 -->
  </behavior>
</application>
```

図 6 中間表現の概観
Fig. 6 Skeleton of intermediate representation.

表 1 中間表現のメタ要素
Table 1 Meta elements of intermediate representation.

要素名	説明
title	RIA のタイトル
charset	RIA の文字コード

表 2 中間表現のウィジェット要素
Table 2 Widget elements of intermediate representation.

要素名	説明
anchor	アンカー（HTML の<a>に相当）
button	ボタン
checkbox	チェックボックス
hbox	子要素を水平に並べる不可視の矩形領域
hr	水平線
image	画像の表示
list	子要素のリスト表示
menu	メニュー
radiobutton	ラジオボタン
scrollbar	スクロールバー
select	複数の子要素からの選択
slider	スライダー
space	空白領域
table	テーブル
text	読み取り専用テキスト
textarea	テキスト入力領域（複数行）
textbox	テキスト入力領域（単一行）
tooltip	カーソルが乗ると文字列を表示
vbox	子要素を垂直に並べる不可視の矩形領域

4.2 中間表現の構成

以下に，IR を構成する 4 種類の情報—メタ情報・ウィジェット情報・スタイル情報・振舞い情報—についてそれぞれ概説する．

4.2.1 メタ情報

メタ情報部は，RIA のメタ情報（例：タイトル・文字コード）を格納する．**表 1** に，代表的なメタ要素を示す．

4.2.2 ウィジェット情報

ウィジェット情報部は，RIA のウィジェット情報（例：ボタン・テキストボックス）を格納する．**表 2** に，主要なウィジェット要素を示す．

```
text {
   font-size: 10pt;
}
```

図 7 CSS によるスタイルの表現例
Fig. 7 Example of style representation in CSS.

```
<rule>
   <selector>text</selector>
   <property>
      <name>font-size</name>
      <value>10pt</value>
   </property>
</rule>
```

図 8 中間表現による図 7 のスタイルの表現
Fig. 8 Style representation of Fig.7 in intermediate representation.

4.2.3 スタイル情報

スタイル情報部は，RIA のスタイル情報（例：フォント・文字サイズ）を格納する．スタイル情報部は，CSS 2.1 で記述可能なすべての情報を保持できる．

我々は，以下の理由からスタイル情報の記述言語に XML 化した CSS を採用した：(1) 多くの RIA 技術がスタイルの指定に CSS を使用している；(2) CSS を XML 化することで構文解析が不要になり変換処理の実装が容易になる．**図 7** と**図 8** とに，CSS と IR のスタイル情報との対応を示す．これらの図から，CSS のルール・セレクタ・プロパティなどが，それぞれ対応する XML 要素で表現されていることが分かる．

4.2.4 振舞い情報

振舞い情報部は，RIA のイベント情報（例：ボタンやマウスのクリック動作）を格納する．振舞い情報部は，ECMAScript で記述可能なすべての情報を保持できる．

我々は，以下の理由から振舞い情報の記述言語に ECMAScript を採用した：(1) 多くの RIA 技術が振舞いの記述に ECMAScript（またはその亜種）を使用している；(2) 振舞い情報の記述に XML を用いると記述量が増えて可読性が低下する．

4.3 IR からの変換例

以下に，同一の IR から自動変換した RIA の画面を示す．**図 9** は入力の IR であり，それを OpenLaszlo・HTML5・JavaFX に変換した結果が**図 10**・**図 11**・**図 12** である．これらの図から，変換時に若干のレイアウトの相違が発生することが分かる．これは，RIA 技術ごとにレイアウトポリシが異なるためである．レイアウトの相違が問題となる場合には，IR のスタイル情報部に絶対座標値を指定する．

4.4 中間表現の拡張性

IR には，XML の文法の範囲内で任意の要素を追加することができる．たとえば，カレンダをウィジェットとして

```
<?xml version="1.0" encoding="UTF-8"?>
<application>
 ...
 <widget>
  <vbox>
   <text>
    Intermediate Representation Sample
   </text>
   <hr />
   <hbox>
    <button>
     <text>button</text>
    </button>
    <radiobutton>
     <text>radiobutton</text>
    </radiobutton>
    <slider />
   </hbox>
   <hbox>
    <textbox />
    <anchor>
     <text>anchor</text>
    </anchor>
    <select>
     ...
    </select>
   </hbox>
   <hbox>
    <image src="duke.jpg" />
    <list>
     ...
    </list>
    <table>
     ...
    </table>
    <scrollbar />
   </hbox>
   <hbox>
    <checkbox>
     ...
    </checkbox>
    <menu>
     ...
    </menu>
    <textarea />
   </hbox>
  </vbox>
 </widget>
 ...
</application>
```

図 9 中間表現のサンプル
Fig. 9 Sample intermediate representation.

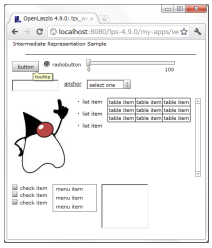

図 10 図 9 の中間表現から変換したページ（OpenLaszlo）
Fig. 10 Transformed page from intermediate representation shown in Fig.9 (OpenLaszlo).

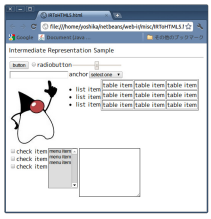

図 11 図 9 の中間表現から変換したページ（HTML5）
Fig. 11 Transformed page from intermediate representation shown in Fig.9 (HTML5).

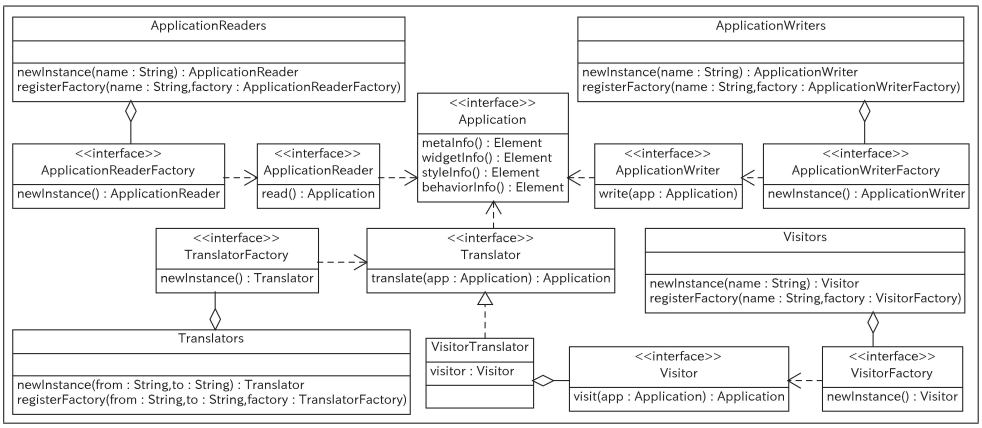

図 13 Web-IR のフレームワークの概要クラス図
Fig. 13 Class diagram of framework of Web-IR.

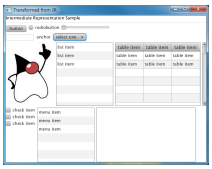

図 12 図 9 の中間表現から変換したページ（JavaFX）
Fig. 12 Transformed page from intermediate representation shown in Fig.9 (JavaFX).

追加する場合には次のようにする：(1) `<calendar>`要素をウィジェット要素（`<widget>`）の配下の任意の場所に追加する；(2) 追加した要素に対する処理をフレームワークに追加する（5.5 節）．この拡張性により，組織や業務に固有のウィジェットをあらかじめ作成して登録することが可能になり，開発時の工数を削減できる．

5. フレームワーク

5.1 概要

Web-IR のフレームワーク（以下，フレームワーク）は，IR と RIA との変換を支援するために提供される．フレームワークは，パーサやファクトリなどの RIA の変換処理に必要な部品群を利用者に提供する．利用者は，これらの部品を組み合わせるかカスタマイズして RIA の変換処理系を実装する．

我々は，以下の理由からデザインパターンを用いてフレームワークを設計した：(1) 利用者のフレームワークへの理解を容易にするため；(2) 変換機能の追加・削除などの拡充改善に対応するため．ここで，我々が採用した重要なパターンとして SPI [15]（Service Provider Interface）をあげる．SPI により，以下のメリットが得られる：(1) フレームワークとクラスとの結合が疎になり，将来新しい RIA 技術が登場しても対応が容易になる；(2) 利用者が，Web-IR が標準でサポートしない RIA への対応を追加できる；(3) 利用者が，標準で提供されるフレームワーク部品の動作をカスタマイズできる．

フレームワークの記述言語には以下の理由から Java を採用した：(1) Servlet や JSP などの Web 開発で実績のあるオブジェクト指向言語である；(2) フリーの実装（OpenJDK）もあり，ただちに言語が廃れる危険性が少ない；(3) コンパイラコンパイラなどのツール類や標準ライブラリ群が充実している．現在，フレームワークは 135 個のクラスと 19,144 行の Java のソースコードとから構成されている．

5.2 フレームワークのクラス構造

図 13 に，フレームワークの概要クラス図を示す．これらのクラスは，前述の SPI パターンに従って次の要素から構成される：(1) 機能を表すサービスインタフェース（`ApplicationReader`・`ApplicationWriter`・`Translator`・`Visitor`）；(2) サービスを生成するためのファクトリインタフェース（`ApplicationReaderFactory`・`ApplicationWriterFactory`・`TranslatorFactory`・`VisitorFactory`）；(3) 登録されたファクトリからサービスを生成するプロバイダインタフェース（`ApplicationReaders`・`ApplicationWriters`・`Translators`・`Visitors`）．すべてのサービスは，対応するプロバイダの `newInstance()` を呼び出すことで，事前に `registerFactory()` によってプロバイダに登録されたファクトリから生成される．ここで重要なことは，これらの登録・生成が，すべて文字列を用いて行われる点である．これにより，`new` 演算子を用いる場合と比較して，フレームワークとフレームワークを利用するコードとの結合が疎になり，柔軟な拡張性が実現できる．

`Application` は任意の RIA を表し，RIA の 4 種類の情報（4.2 節）の木構造のルートノードへの参照を提供するメソッドを持つ．ここで重要なことは，この木構造を Ajax で実績のある DOM（Document Object Model）と同じにすることである．これにより，利用者の学習工数の低減が期待できる．`ApplicationReader` は，RIA を `Application` として読み込むリーダを表す．`ApplicationWriter` は，`Application` を RIA

```
/**
 * Web-IRの変換を説明するための擬似コード．
 * この例では，入力と出力をそれぞれAjaxと
 * 中間表現と仮定している．
 * @param e0 変換するHTMLの要素
 * @return e0と等価な中間表現の要素
 */
Element visit(Element e0){
 switch(e0){
  case "<input type='checkbox'>":
   Element e1 = new Element("<checkbox>");
   while(e0.hasMoreChild()){
    e1.appendChild(visit(e0.nextChild()));
   }
   return e1;
  case "<input type='text'>":
   Element e1 = new Element("<textbox>");
   return e1;
  ...
 }
}
```

図 14 Web-IR の変換アルゴリズム（擬似コード）
Fig. 14 Transformation algorithm of Web-IR (pseudo code).

として書き出すライタを表す．`Translator` は，`Application` を異なる `Application` に変換するトランスレータを表す．`VisitorTranslator` は，Visitor パターン（5.3 節）による `Translator` の実装クラスである．このクラスが，前述の木構造をトラバースし，訪れているノードを引数にして `Visitor` にコールバックを行うため，利用者は変換規則を `Visitor` として実装するだけで変換を実現できる．

5.3 変換アルゴリズム

フレームワークは，入力を構文解析した木構造に Visitor パターン [4] を適用し変換を行う．変換アルゴリズムの概観を**図 14** に示す．図中の `visit` メソッドが示すように，メタ情報・ウィジェット情報・スタイル情報の変換は，引数として与えられる XML の要素 `e0` を対応する要素に再帰的に変換することで実現する．ただし，振舞い情報の変換時には，XML の要素ではなく抽象構文木（AST：Abstract Syntax Tree）のノードが変換対象となる．

なお，変換率を高めるためには，以下の要件を備えたエミュレーションライブラリの併用が望ましい：(1) 出力の RIA 技術で実装されており，(2) 入力の RIA 技術の機能を模倣するもの．これは，各 RIA 技術が多くの共通機能（例：非同期通信）を持ちながら，セマンティクスにおいては共通点が乏しく，直接変換が困難である場合があるためである．この傾向は，振舞い情報の変換時に顕著である．

5.4 フレームワークを用いた開発事例

Web-IR を用いた処理系を 2 つ紹介する．これらの処理系は，Web-IR のフレームワークのみを用いて IR を経由せずに RIA の直接変換を行う．

長谷川ら [6] は，Ajax を Flash に変換する処理系を開発した．この処理系は，IR の代わりに LZX—OpenLaszlo の記述言語—を採用し，OpenLaszlo の拡張機能を用いて Ajax のエミュレーションを行う点に特徴があり，9 個のクラスと 1,304 行のソースコードと 115 個のエミュレーションクラスとで構成される．

吉賀ら [17] は，Flex の UI を HTML5 に変換する処理系を開発した．この処理系は，Flex のソースファイル（MXML）を独自の木構造に変換してから HTML5 の DOM 木に変換する点に特徴があり，56 個のクラスと 3,181 行のソースコードとで構成される．

我々は，これらの開発で次の知見を得た：(1) RIA の変換処理系の開発では，パーサの実装に多くの工数が消費されるため，あらかじめ XML・CSS・ECMAScript のパーサを共通部品として実装・提供することで工数を削減できる；(2) フレームワークを用いることで，利用者は Visitor の実装のみに集中することができ，フレームワークを用いない場合と比較して少ない工数で処理系を実装することができる．

5.5 フレームワークの拡張性

フレームワークは，IR（4.4 節）と同様に利用者が機能を拡張することができる．具体的には，IR に要素を追加した場合には次の対応が必要である：(1) 追加した要素を処理する `Visitor` を実装する；(2) 実装した `Visitor` を生成する `VisitorFactory` を実装する；(3) 実装した `VisitorFactory` を `Visitors` に登録する．ここで重要なことは，`Visitor` を次のいずれかの方法で実装することである：(1) 既存の `Visitor` を継承して，追加した要素以外の処理は親クラスに転送する；(2) 新規に `Visitor` を実装して，追加した要素以外の処理は既存の `Visitor` に転送する．これにより，既存の変換機能を損なうことなく新しい要素への対応が可能になる．なお，新しい要素の追加ではなく既存の変換規則を変更したい場合でも，同様の方法で対応が可能である．

たとえば，カレンダをウィジェットとして追加する場合には次のようにする：(1) IR に`<calendar>`要素を追加する（4.4 節）；(2-a) `Visitor` を継承を用いて実装し，図 14 を参考に `visit()` の `switch` 文に`<calendar>`用の `case` 節を追加し，それ以外の処理（`default` 節）は親クラスの `visit()` に転送する；(2-b) `Visitor` を継承を用いずに実装し，図 14 を参考に `visit()` の `switch` 文に`<calendar>`用の `case` 節を追加し，それ以外の処理（`default` 節）は既存の `Visitor` の `visit()` に転送する．

6. 評価

我々は，Web-IR の有用性を評価するために以下の試験を行った：(1) IR が RIA の情報をどの程度表現できるか；(2) 手動による RIA 開発にどの程度の工数がかかるか；(3) フレームワークが RIA 変換システムの開発工数をどの程度削減するか．

6.1 RIA 情報の変換率による評価

6.1.1 概要

我々は，IR と各 RIA 技術の UI 情報—ウィジェット情報

とスタイル情報—との変換率を算出した（参考値として，振舞い情報の変換率にも 6.1.5 項で言及する）．評価軸に UI 情報を選択した理由は次のとおりである：(1) 大規模な RIA では画面数や UI 部品数が多くなるため，UI 情報の変換だけでも工数削減に有効である；(2) UI 情報の変換は，振舞い情報の変換と比較して実装が容易なため，少ない投資で大きな見返りを得ることが期待できる．

さらに，入力の RIA の UI 情報の出現率を重みとして考慮した場合の変換率も算出した．これは，次の理由から重要である：(1) 各 RIA 技術には多くの UI 部品が存在するが，使用頻度には偏りがあり，頻出する UI 部品への対応がより重要である；(2) 出現率を考慮せずに変換率を算出すると，UI 部品が多い RIA 技術の見かけの変換率が下がってしまい，実状とかけ離れた値になるおそれがある．

我々は，3.5 節で述べた理由から以下の RIA 技術を変換率の測定対象とした：(入力) Ajax と Flex；(出力) OpenLaszlo と HTML5 と JavaFX．これらの入力から IR への変換率と IR からこれらの出力への変換率とが評価値となる．

6.1.2 UI 情報の出現率

我々は，以下の方法で Ajax の UI 情報の出現率を測定した：(1) 独自のクローラを Java で実装し，(2) Web 上のトラフィック量の上位 [3] 10 サイト（**表 3**）を自動巡回し，(3) 使用されている HTML のタグと CSS のプロパティとの種類と個数とを計測する．なお，Flex については，バイナリファイル（SWF ファイル）にコンパイルされて配備されるという特性上，出現率を測定できなかった．

我々が，表 3 のサイトを評価に用いた理由は次のとおりである：(1) 業務アプリケーションは組織や業務によって特性が異なり，使用している UI コンポーネントの種類と出現頻度とを正確に調べることが難しい；(2) 業務アプリケーションは，利便性が特に重視されるこれらのサイトと比較して UI 要件が厳しくないことが想定できるため，これらのサイトに対して評価を行うことで，より厳しい UI 条件下での変換率を算出できると考えたため．

UI 情報の出現率の測定結果を，**表 4** と**表 5** とに示す．表 4 で注目すべき点として，`<div>`と`<span>`とが多用されている点をあげる．我々はこの理由を，これらのタグが Ajax アプリケーション中で JavaScript から操作される領域を示すために使用されているためと考える．

6.1.3 ウィジェット情報の変換率

ウィジェット情報の変換率を**表 6** に示す．表より，IR のウィジェット情報の表現能力は，入力が Ajax の場合は 8 割以上となるが，Flex の場合は 4 割強となることが分かる．これは，Flex が IR の倍以上の UI 部品を持つためである．ここで，IR の拡張機能（4.4 節）を用いると，Flex の場合でも変換率が 8 割を超えることが分かる．

6.1.4 スタイル情報の変換率

スタイル情報の変換率を**表 7** に示す．表より，IR のスタイル情報の表現能力は，入力が Ajax の場合は 10 割となるが，Flex の場合は 3 割強となることが分かる．これは，

表 3 Web 上のトラフィック量上位 10 サイト
Table 3 Top 10 web sites ordered by traffic.

順位	サイト名	URL
1	Google	http://www.google.com/
2	Facebook	http://www.facebook.com/
3	YouTube	http://www.youtube.com/
4	Yahoo!	http://www.yahoo.com/
5	Blogger.com	http://www.blogspot.com/
6	Baidu.com	http://www.baidu.com/
7	Wikipedia	http://www.wikipedia.org/
8	Windows Live	http://www.live.com/
9	Twitter	http://www.twitter.com/
10	QQ.COM	http://www.qq.com/

表 4 表 3 のサイト中の HTML タグの出現率
Table 4 HTML tag occurrences in top 10 traffic sites shown in Table 3.

順位	タグ名	出現数	出現率（%）
1	a	1,482	27.15
2	div	1,290	23.63
3	span	828	15.17
4	img	376	6.89
5	li	313	5.73
6	br	298	5.46
7	button	146	2.67
8	script	106	1.94
9	p	70	1.28
10	option	67	1.22
...	...	...	...
合計		5,457	100.00

表 5 表 3 のサイト中の CSS プロパティの出現率
Table 5 CSS property occurrences in top 10 traffic sites shown in Table 3.

順位	プロパティ名	出現数	出現率（%）
1	width	1,235	7.44
2	padding	919	5.53
3	height	902	5.43
4	display	902	5.43
5	color	842	5.07
6	background	692	4.17
7	margin	605	3.64
8	font-size	575	3.46
9	background -position	575	3.46
10	position	559	3.36
...	...	...	...
合計		16,592	100.00

表 6 ウィジェット情報の変換率
Table 6 Transformation ratio for widget information.

入力	出力	変換率（%）	変換率[*12]（%）	変換率[*13]（%）
Ajax	IR	80.6	93.8	95.0
Flex	IR	44.0	N/A	83.8[*14]
IR	OpenLaszlo	100.0	100.0	100.0
IR	HTML5	90.0	N/A[*15]	100.0[*16]
IR	JavaFX	100.0	100.0	100.0

表 7 スタイル情報の変換率
Table 7 Transformation ratio for style information.

入力	出力	変換率（%）	変換率[*12]（%）	変換率[*13]（%）
Ajax	IR	100.0	100.0	100.0
Flex	IR	33.3	N/A	83.8[*14]
IR	OpenLaszlo	42.6	54.8[*17]	88.0[*16]
IR	HTML5	100.0	100.0	100.0
IR	JavaFX	62.3	95.8[*17]	100.0[*16]

Flex がウィジェットごとに固有のスタイル情報を持つために，見かけの変換率が低くなるためである．また，IR から HTML5 以外の RIA への変換率が低い理由は，これらの RIA が CSS のサブセットのみをサポートするためである．ここでも，IR の拡張機能を用いると，変換率が 8 割を超えることが分かる．

6.1.5 振舞い情報の変換率

振舞い情報の変換率を定量的に示すことは難しい．これは，RIA の振舞いの記述に使用する言語—ECMAScript・JavaScript・ActionScript・JavaFX Script[*9]など—の間に直接的な互換性[*10]がないためである．したがって，次のように場合分けして言及する：(1) 入力 RIA から IR への変換率および IR から出力 RIA への変換率；(2) その他の考察；(3) 評価の例．

Web-IR のフレームワークは，ECMAScript のパーサを提供し，入力 RIA 中の振舞い情報を AST に変換する．この段階での変換率は，入力が ECMAScript に準拠する限りほぼ 100%である[*11]．この AST に対して，利用者が実装した `Visitor` が処理を行い IR へ変換する．この段階での変換率は，利用者の `Visitor` の実装とエミュレーションライブラリの再現精度とに依存する．これらの動作と変換率とは，IR から出力 RIA への変換においても同様である．

我々は，RIA の振舞い情報の変換のしやすさを次の 3 種類に分類できると考える：(1) 変換が容易な場合；(2) 変換が可能な場合；(3) 変換が困難な場合．(1) は，標準化 (1.2.2 項) が厳格に行われている場合に発生する．すなわち，組織の中で使用する RIA 技術をあらかじめ定めており，その RIA 技術中で使用する機能なども限定している場合には，出力 RIA 側で同等の機能を提供するエミュレーションライブラリを使用することで高精度の変換が期待できる．たとえば，標準の RIA を Ajax と定め，業務に必要なすべての機能をあらかじめ JavaScript の関数として提供し，それ以外の関数の使用を禁じている場合が該当する．(2) は，標準化が緩やかに行われている場合に発生する．すなわち，組織の中で使用する RIA 技術を定めてはいるが，その RIA 技術中で使用する機能を限定していない場合には，必要に応じて使用している機能に対して個別の対応が必要となる．たとえば，標準の RIA とライブラリとをそれぞれ Ajax と jQuery と定めたが，使用するライブラリ関数には制限を課していない場合が該当する．(3) は，標準化が行われていない場合に発生し，入力 RIA 中で使用しているすべての機能に対して個別の対応が必要となる．たとえば，標準の RIA を定めず，様々な RIA の様々な機能が実装に用いられている場合が該当する．この場合には，Web-IR のフレームワークを用いて変換処理系を実装する以外に，手作業で変換処理を行うことも検討する必要がある．

最後に，我々が実施した振舞い情報の変換率の評価例を示す．我々は，Ajax を Flash に変換する処理系において，エミュレーションライブラリを併用することで振舞い情報

[*9] JavaFX Script は JavaFX 1.x までで廃止となり，JavaFX 2.x からは Java が用いられている．
[*10] JavaScript は，ECMAScript の方言であり完全互換ではない．ActionScript は，クラスベースのオブジェクト指向機能を搭載しており，ECMAScript を拡張したものである．
[*11] ここで，AST への変換率が必ずしも RIA の振舞い情報の再現率と等価ではないことに注意されたい．これは，振舞い情報を変換するためには，スクリプト中で使用されているオブジェクトやプロパティなどの変換も必要になるためである．
[*12] 表 3 のサイトの UI 情報の出現率を重みとして考慮した変換率．
[*13] IR の拡張機能を用いた場合の変換率（UI 情報の出現率による重みは考慮していない）．
[*14] IR の拡張機能を用いる場合には，Flex のスタイル情報はウィジェット情報として扱われるため．
[*15] IR の出現率が得られないため重みを算出できない．
[*16] IR の拡張機能とエミュレーションライブラリとを併用した場合．
[*17] IR のスタイル情報は CSS であるため，表 5 の出現率を重みとして用いた．

の 41%の変換に成功した [6]．これは，上述の変換のしやすさの分類では（3）に相当する．この評価から，エミュレーションライブラリを併用すれば，変換が困難と考えられる RIA の入力に対しても，半数程度の振舞い情報を変換できるという結果を得た．

6.2 開発工数による評価

6.2.1 手動による RIA 開発の工数

我々は，手動で RIA 開発を行った場合の工数を測定した．これは，既存の RIA を新しい RIA 技術に移植する際の工数を見積もるためである．測定条件は次のとおりである：(1) 移植先の RIA 技術の初学者 3 人を被験者として用意し，(2) 指定した UI（図 12）を開発する作業を行わせ，(3) 学習工数と開発工数それぞれの平均を算出した．移植先の RIA 技術は，3.5 節と同様の理由から JavaFX と OpenLaszlo とを選択した．なお，HTML5 は Ajax と要素技術が同じであるため，移植先として不適切と判断し選択肢から除外した．

結果は次のようになった：(JavaFX) 学習工数：1.0 人時，開発工数：3.5 人時；(OpenLaszlo) 学習工数：1.0 人時，開発工数：9.5 人時．

これらの結果から，(図 12 のような単純な UI であっても) 初学の RIA 技術による UI 開発には工数がかさむことが示唆される．我々は，被験者への聞き取りから，この理由を以下のように考える：(1) RIA 技術ごとにレイアウトポリシやウィジェットの使用方法が異なるために，概要を把握するだけでも時間がかかる；(2) 多くの RIA 技術は統合開発環境を備えないため，テキストエディタとブラウザによる開発が必要となり効率が悪い．

実際の開発においては，(1) 学習工数が初回開発時にしかかからないことや，(2) 対象の RIA 技術に習熟するにつれて単位時間あたりの生産性が向上することも考慮しなければならない．一方で，(1) つねに同じ開発要員を確保できるわけではないことや，(2) 1 つの RIA だけでも多くの UI 情報を含むことを考慮すると，依然として手動に代わる移植方法が必要であると我々は考える．

6.2.2 UI 変換システムの開発工数による比較

我々は，Web-IR の使用の有無による RIA の UI 変換システムの開発工数を測定した（Web-IR が未対応の変換に対する手動対応の工数も含む）．これは，RIA の移植が必要となった際に変換システムの実装に必要な工数を見積もるためである．測定条件は次のとおりである：(1) 言語処理系を専門としない 3 人を被験者として用意し，(2) ある RIA 技術から異なる RIA 技術への変換処理系を実装させ，(3) 開発に要した工数の平均を算出した．対象の RIA 技術は，3.5 節で述べた理由から入力を Ajax とし，6.2.1 項と同様の理由から出力を JavaFX と OpenLaszlo とした．

測定結果を**表 8** に示す（参考値として，Flex から HTML5 への処理系の開発工数も示す）．表より，Web-IR の使用が，処理系の開発にかかる工数と行数とをそれぞれ 2/3 以下に削減することが分かる．我々はこの理由を，Web-IR が処理系の開発方法を強制することで，利用者が設計に煩わされることなく実装に専念できるためと考える．

表 8 Web-IR の使用の有無による開発工数の比較
Table 8 Comparison of production costs with/without using Web-IR.

入力	出力	Web-IR の使用の有無	工数（人時）	行数
Ajax	OpenLaszlo	×	10.0	1,450
Ajax	OpenLaszlo	○	5.0	960
Ajax	JavaFX	×	20.0	3,156
Ajax	JavaFX	○	12.0	1,627
Flex	HTML5	×	24.0	1,818
Flex	HTML5	○	18.0	1,231

7. おわりに

本論文では，RIA の移植性の問題を解決するために，中間表現とフレームワークとを用いる方法を提案した．プロトタイプを用いた評価は，我々の提案手法が前述の問題解決（特に UI の移植）に有用であることを示した．一方で，振舞い情報については，より効率的な変換手法の提案が必要であると考える．これについては別の機会に報告したい．

参考文献

[1] Adobe Labs.: Wallaby, Adobe Systems Inc. (online), available from ⟨http://labs.adobe.com/technologies/wallaby/⟩ (accessed 2011-12-20).

[2] Adobe Systems: Flash to Focus on PC Browsing and Mobile Apps; Adobe to More Aggressively Contribute to HTML5, Adobe Systems (online), available from ⟨http://blogs.adobe.com/conversations/2011/11/flash-focus.html⟩ (accessed 2011-12-20).

[3] Alexa Internet, Inc.: Alexa Top 500 Global Sites, Alexa Internet, Inc. (online), available from ⟨http://www.alexa.com/topsites⟩ (accessed 2011-09-05).

[4] Gamma, E., Helm, R., Johnson, R. and Vlissides, J.: *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional (1995).

[5] Google: Swiffy, Google (online), available from ⟨http://www.google.com/doubleclick/studio/swiffy/⟩ (accessed 2011-12-20).

[6] 長谷川慎哉，早川智一，疋田輝雄：RIA 変換フレームワークを用いた Ajax アプリケーションの Flash 変換手法の実装と評価，第 74 回全国大会講演論文集（分冊 1），pp.707–708，情報処理学会 (2012).

[7] 早川智一，長谷川慎哉，疋田輝雄：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，DPSWS2010 論文集，pp.187–192，情報処理学会 (2010).

[8] Hayakawa, T., Hasegawa, S. and Hikita, T.: Design and Implementation of Intermediate Representation and Framework for Web Applications, *Proc. CSIE 2011*, Changchun, China, pp.137–141 (2011).

[9] Hayakawa, T., Hasegawa, S. and Hikita, T.: Design and Implementation of Intermediate Representation and

Framework for Web Applications, *Recent Advances in CSIE 2011*, Vol.2, pp.375–384, Springer (2012).
[10] Hayakawa, T., Hasegawa, S., Yoshika, S. and Hikita, T.: Maintaining Web Applications by Translating Among Different RIA Technologies, *Proc. SEA 2011*, Singapore, pp.53–58 (2011).
[11] 早川智一，長谷川慎哉，吉賀祥太，疋田輝雄：中間表現とフレームワークを用いたWebアプリケーションのメンテナンス法の提案と評価，DPS149, 情報処理学会 (2011).
[12] Hayakawa, T., Hasegawa, S., Yoshika, S. and Hikita, T.: Maintaining Web Applications by Translating Among Different RIA Technologies, *GSTF Journal on Computing*, Vol.2, No.1, pp.250–256 (2012).
[13] 松塚貴英：業務アプリケーションに適用するAjaxフレームワーク，情報処理学会論文誌，Vol.49, No.7, pp.2360–2371 (2008).
[14] 三菱総合研究所：OSSデスクトップの普及に資するWebコンテンツ互換性向上に関する調査，情報処理推進機構オープンソフトウェア・センター（オンライン），入手先 ⟨http://www.ipa.go.jp/software/open/ossc/seika.html⟩（参照 2011-12-20）.
[15] Oracle Corporation: Creating Extensible Applications With the Java Platform, Oracle Corporation (online), available from ⟨http://java.sun.com/developer/technicalArticles/javase/extensible/⟩ (accessed 2011-12-20).
[16] Ping, Y., Kontogiannis, K. and Lau, T.C.: Transforming Legacy Web Applications to the MVC Architecture, *Proc. STEP 2003*, pp.133–142 (2003).
[17] 吉賀祥太，早川智一，疋田輝雄：Flexアプリケーションの iOS 環境での HTML5 変換による実行，FIT2011 講演論文集（第 4 分冊），pp.385–388 (2011).

付　　録

A.1 Web-IR の実装

A.1.1 実装に使用するソフトウェア

- Java SE 7（開発言語；http://www.oracle.com/technetwork/java/javase/）
- JavaCC 5.0（コンパイラコンパイラ；http://javacc.java.net/）
- Xerces2 Java Parser 2.11.0（XML パーサ；http://xerces.apache.org/xerces2-j/）

フレームワークは Java SE 7 で実装し，IR の解析には Xerces2 を用いた．CSS と ECMAScript のパーサは，JavaCC と A.1.2 節の文法規則とを用いて実装した．

A.1.2 CSS と ECMAScript の文法規則

- CSS 2.1（http://www.w3.org/TR/CSS2/grammar.html）
- ECMAScript（http://svn.dojotoolkit.org/src/trunk/tools/jslinker/src/org/dojo/jsl/parser/EcmaScript.jjt）

A.2 Web-IR の構築

A.2.1 Web-IR のコア部分の構築

(1) Java SE 7 と JavaCC とを用いて，XML・CSS・ECMAScript パーサを実装する；(2) 図 13 を参考に，フレームワークのクラス群を実装する；(3) 必要な機能や UI コンポーネントを検討し，IR の要素を決定する．

A.2.2 Web-IR に新しい RIA の入力を追加する

(1) 入力 RIA に対応する `ApplicationReader` を実装する；(2) 実装した `ApplicationReader` を生成する `ApplicationReaderFactory` を実装する；(3) 実装した `ApplicationReaderFactory` を `ApplicationReaders` に登録する；(4) 図 14 を参考に，入力 RIA と IR との変換規則を `Visitor` に実装する；(5) 実装した `Visitor` を生成する `VisitorFactory` を実装する；(6) 実装した `VisitorFactory` を `Visitors` に登録する．

A.2.3 Web-IR に新しい RIA の出力を追加する

(1) 図 14 を参考に，IR と出力 RIA との変換規則を `Visitor` に実装する；(2) 実装した `Visitor` を生成する `VisitorFactory` を実装する；(3) 実装した `VisitorFactory` を `Visitors` に登録する；(4) 出力 RIA に対応する `ApplicationWriter` を実装する；(5) 実装した `ApplicationWriter` を生成する `ApplicationWriterFactory` を実装する；(6) 実装した `ApplicationWriterFactory` を `ApplicationWriters` に登録する．

A.3 Web-IR の利用

A.3.1 RIA の移植を行う場合

(1) A.2.1 節を行う；(2) A.2.2 節を行う；(3) 図 5 を参考に，入力 RIA から IR への変換を行うメソッドを記述する；(4) A.2.3 節を行う；(5) 図 5 を参考に，IR から出力 RIA への変換を行うメソッドを記述する．

A.3.2 RIA の新規開発を行う場合

(1) A.2.1 節を行う；(2) IR を用いて RIA の開発を行う；(3) A.2.3 節を行う；(4) 図 5 を参考に，IR から出力 RIA への変換を行うメソッドを記述する．

推薦文

中間表現を用いた Web アプリケーションの変換法を提案し，この手法で異なるプラットフォーム間の変換が可能であるかを評価しているとともに，企業などが負担するコストの削減効果を評価している．本研究における手法自体は比較的古典的であるが，実際の現場の要求を反映しており，実用性の面で高い貢献が認められる．よって，本研究

会からの推薦に値する．
（マルチメディア通信と分散処理研究会主査　勝本道哲）

早川 智一　（学生会員）

1982 年生．2007 年明治大学大学院理工学研究科基礎理工学専攻博士前期課程修了．同年（株）ティージー情報ネットワーク入社．主に技術開発と UNIX サーバの構築とに従事．2010 年より明治大学理工学部助手．現在，同大学院博士後期課程で Web アプリケーションの移植に関する研究に従事．2010 年情報処理学会 DPSWS2010 学生奨励賞受賞．2012 年度山下記念研究賞受賞．

長谷川 慎哉

1987 年生．2012 年明治大学大学院理工学研究科基礎理工学専攻博士前期課程修了．Web アプリケーションの自動変換に関する研究に従事．2011 年情報処理学会第 73 回全国大会学生奨励賞受賞．

吉賀 祥太

1987 年生．2011 年明治大学理工学部情報科学科卒業．同年明治大学大学院理工学研究科基礎理工学専攻博士前期課程入学．現在，同大学院にて Web アプリケーションの自動変換に関する研究に従事．

疋田 輝雄　（正会員）

1970 年東京大学卒業．東京大学，東京都立大学を経て，1989 年から明治大学理工学部情報科学科教授．現在，ウェブ技術等に興味を持っている．2003 年から 2010 年まで情報処理学会コンピュータ科学教育委員会委員長．著書は『コンパイラの理論と実現』（共立出版）等．『情報科学こんせぷつ』シリーズ編集（朝倉書店）．