Maintaining Web Applications by Translating Among Different RIA Technologies

Tomokazu Hayakawa, Shinya Hasegawa, Shota Yoshika, and Teruo Hikita

Abstract—**As RIA (Rich Internet Application) technologies have been widely used, the compatibility problem has arisen: they are hardly compatible with each other. To solve the problem, we have proposed and implemented an automatic RIA transformation system named Web-IR, which uses an XML-based intermediate representation with a Java-based framework. As concrete examples, Web-IR currently supports Ajax, Flex, JavaFX, and OpenLaszlo as its input/output. Our evaluations show that Web-IR can transform existing real applications into other RIA technologies keeping almost the same appearances. Finally, we conclude that Web-IR can solve the problem sufficiently.**

Index Terms—**Software design, Software maintenance, Software reusability, Web design**

I. Introduction

As RIA (Rich Internet Application) technologies have been widely used, the compatibility problem has arisen: they are hardly compatible with each other. Since there are already a large number of RIA technologies such as Ajax, Flex, Silverlight, JavaFX [11], OpenLaszlo [9], and HTML5, we can develop RIAs easily by using any of them. However, there is no common specification among RIA technologies, which means that they do not have compatibility with each other. Although the lack of the compatibility may not be considered problematic in personal use, it causes serious problems in business use; if one of RIA technologies becomes obsolete, the existing RIAs developed by the obsoleted technology will be required to be redeveloped by using another RIA technology in order to continue the support for the existing users. It obviously consumes time and costs to redevelop existing RIAs from scratch, especially in case of redevelopment of business applications, because they tend to have a large number of Web pages and UI (User Interface) components.

The purpose of this paper is to solve the problem by proposing an intermediate representation and a framework. Although it is classical to use intermediate representations in order to solve compatibility problems, few attempts have been made in this area of RIA technologies. Here, we have addressed the problem, especially focusing on UI information of RIAs. Our proposed method is the following: transforming an input RIA into an XML-based intermediate representation, and transforming the transformed intermediate representation into another RIA; both the transformations are performed by a Java-based framework that we have built. We have named the intermediate representation *IR* and our system *Web-IR*. Web-IR consists of the intermediate representation and the framework, and it aims to realize any-to-any RIA transformations. The framework of Web-IR can currently transform Ajax and Flex into the intermediate representation, and it can also transform the transformed intermediate representation into HTML5, OpenLaszlo, and JavaFX. The key ideas of Web-IR are the following: using the XML-based intermediate representation to maximize extensibility and interchangeability of the information of input RIAs, and providing the Java-based extensible framework to help developers easily transform RIAs into others through the intermediate representation.

In this paper, we mainly describe the results of our evaluations of Web-IR, because we already described in [8] an overview of the design and implementation of Web-IR and their details in [7]. The results show that Web-IR can save developers' time and costs, especially when they redevelop existing RIAs.

The rest of this paper is organized as follows. Section II introduces related work. Section III outlines an overview of Web-IR. Sections IV and V describe the intermediate representation and the framework, respectively. Section VI shows the results of our evaluations. Section VII presents the conclusion.

II. Related Work

In this section, we summarize previous works related to our study. However, there seem few studies that solve the compatibility problem by using a framework and an intermediate representation in which all of the information of an input RIA is held. Hence, we summarize relatively similar works in the next two subsections, and explain the main differences between them and our study in the subsection C.

A. RIA Related Frameworks

Yu, Kontogiannis, and Lau [13] proposed a Java-based framework, which generates input-equivalent MVC applications by applying reengineering techniques to solve porting and adaptation problems of existing Web applications. Zhang and Chen [14] proposed a Web application framework based on MVC model, which uses XML to extend its flexibility.

Manuscript received February 29, 2012.

Tomokazu Hayakawa, Shinya Hasegawa, Shota Yoshika, and Teruo Hikita are with the School of Science and Technology, Meiji University, Kawasaki, 214-8571, Japan (e-mail: {t_haya, s-hase, yoshika, hikita}@cs.meiji.ac.jp).

DOI: 10.5176_2010-2283_2.1.158

B. *Existing RIA Translation Services*

Adobe Systems has released a RIA transformation service named Wallaby [1], which transforms Adobe Flash Professional (FLA) files into HTML. In addition, Google has also released a similar RIA transformation service named Swiffy [5], which transforms Flash (SWF) files into HTML5.

C. *Differences between Previous Works and Ours*

One of the main differences between the previous works and ours is that Web-IR is designed to be highly extensible with less production costs. For example, our intermediate representation can be extended by adding arbitrary XML elements (e.g., `<calendar>`), and our framework can also be extended by adding new classes or by extending the existing classes.

III. System Overview

A. *Objectives*

One of the main objectives of our system is to provide a way of RIA transformation, especially that for UI information of RIAs, with less production costs. This is so because recent Web applications are implemented by using one or more of RIA technologies, and we will need to update such applications by using another RIA technology when a RIA technology used for existing applications becomes obsolete. We often meet such situations, especially in business. (For example, Adobe Systems has decided to discontinue the development of the Flash Player for mobile devices except for bug fixes and security updates [2].)

B. *Overview*

Fig. 1 shows an overview of our system. As the figure shows, it consists of two parts: an XML-based intermediate representation and a Java-based framework; the framework transforms an input RIA into supported RIAs as output through the intermediate representation. Web-IR can currently treat Ajax and Flex 4.5 as input, and it can also treat HTML5, JavaFX 2.0 (beta), and OpenLaszlo 4.9 as output. Moreover, Web-IR has extensibility to treat other RIA technologies, if necessary.

C. *Example of Usage*

Fig. 2 and Fig. 3 show sample Web pages before the transformation, which are rendered by different Web browsers. Fig. 4 shows the corresponding JavaFX page transformed by the framework. As these figures show, the framework can transform a Web page into another only with a slight UI difference. Fig. 5 shows a Java code fragment that performs the transformation of this example, which shows that we can transform RIAs into others with a few lines of code by using Web-IR as long as the framework supports both the input and the output RIA technologies.

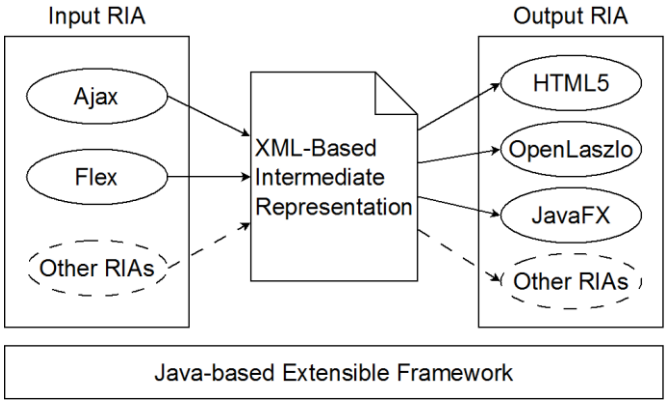

Fig. 1. Overview of Our Proposed System.

IV. Intermediate Representation

A. *The Reason for Using Intermediate Representation*

We have decided to design and use our original intermediate representation; we have named it *IR*. The reason is the following: existing RIA technologies are not seemed to be suitable for our purpose because they are not that sufficient

Fig. 2. Sample Web Page Rendered by IE 9 (before Transformation).

Fig. 3. Sample Web Page Rendered by Firefox 6 (before Transformation).

Fig. 4. Transformed JavaFX Page (after Transformation).

```
// Get an ApplicationReader to read an Ajax as input.
ApplicationReader src = ApplicationReaders.newInstance("Ajax");
// Read an Ajax from the URI passed as the argument.
Application app1 = src.read("file://...(URI)");
// Get a Translator to translate the Ajax into IR.
Translator translator1 = Translators.newInstance("Ajax", "IR");
// Translate the Ajax into IR.
Application app2 = translator1.translate(app1);
// Get a Translator to translate the IR into JavaFX.
Translator translator2 = Translators.newInstance("IR", "JavaFX");
// Translate the IR into JavaFX.
Application app3 = translator2.translate(app2);
// Get an ApplicationWriter to write the translated JavaFX as output.
ApplicationWriter dst = ApplicationWriters.newInstance("JavaFX");
// Write the translated JavaFX to the URI passed as the argument.
dst.write(app3, "file://...(URI)");
```

Fig. 5. Java Code Fragment (from Ajax to JavaFX via IR).

to represent other RIAs, since they are not designed as intermediate representation; even if we choose one of existing RIA technologies, it may lead our system to "vendor lock-in."

B. Overview

IR is based on XML to maximize extensibility and interchangeability of the information of input RIAs, and it consists of four parts: meta information, widget information, style information, and behavior information. This is so because we have considered that the information of a Web application can be classified into the four parts. In case of Ajax, for example, HTML, CSS, and JavaScript correspond to the widget information, the style information, and the behavior information, respectively; some HTML elements (e.g., `<title>`) correspond to the meta information.

Fig. 6 shows the skeleton of IR. As the figure shows, IR has `<application>` element as its root, and it also has four elements (`<meta>`, `<widget>`, `<style>`, and `<behavior>`) to store the four kinds of information separately. The elements that can be used in each part are explained in the next subsection.

C. Detail of Intermediate Representation

1) Meta Part

The meta part contains meta information of a RIA, which is not to be included in the other parts. Table I shows the currently supported elements.

2) Widget Part

Table II shows the list of the UI elements of IR (in which minor elements and attributes are omitted). These elements are chosen to be the intersection among the sets of the UI widgets of the RIA technologies that are supported by Web-IR.

3) Style Part

In the style part, IR uses CSS 2.1 as its description language, because almost all RIA technologies use CSS to define their styles. The CSS information is stored as XML to enable easy transformation of the style information.

Fig. 7 and Fig. 8 show that how the style information is treated in both format, CSS 2.1 and IR. As shown in the figures, `<rule>`, `<selector>`, `<property>`, `<name>`, and `<value>` elements are used to express the style rules.

TABLE I
META ELEMENTS OF INTERMEDIATE REPRESENTATION.

Element Name	Description
title	The title of an input RIA.
charset	The charset of an input RIA.

TABLE II
WIDGET ELEMENTS OF INTERMEDIATE REPRESENTATION.

Element Name	Description
anchor	An anchor widget (such as `<a>` in HTML).
button	A button widget.
checkbox	A checkbox widget.
hbox	An invisible box that aligns its children horizontally.
hr	A horizontal line.
image	An image widget that shows an image.
list	A list widget that shows its child widgets.
menu	A menu widget that shows its children as menu items.
radiobutton	A radio button widget.
scrollbar	A scrollbar widget.
select	A selection box widget.
slider	A slider widget.
space	An invisible widget to reserve space.
table	A table widget.
text	A read-only single-line text widget.
textarea	A text area with multiple lines.
textbox	A text area with a single line.
tooltip	A tooltip that shows a text when the cursor moves on it.
vbox	An invisible box that aligns its children vertically.

4) Behavior Part

In the behavior part, behavior information such as a click event of a button is stored as ECMAScript, because it is the most common description language among the current RIA technologies, which means that developers can easily understand and change the behavior information of IR, if necessary.

D. Examples: IR to OpenLaszlo, Flex, and JavaFX

Fig. 9, Fig. 10, and Fig. 11 show the appearances of the UI widgets of Web-IR; they are automatically transformed from the single IR shown in Fig. 12 into OpenLaszlo, HTML5, and JavaFX, respectively. We can see from these figures that we can easily develop multiple RIAs with a slight UI difference from a single source by using Web-IR.

```
<?xml version="1.0" encoding="UTF-8"?>
<application>
    <meta>
        <!-- meta information -->
    </meta>
    <widget>
        <!-- widget information -->
    </widget>
    <style>
        <!-- style information -->
    </style>
    <behavior>
        <!-- behavior information -->
    </behavior>
</application>
```

Fig. 6. Skeleton of Intermediate Representation.

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

Fig. 7. Sample CSS 2.1.

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

Fig. 8. CSS Representation Corresponding to Fig. 7 in Intermediate Representation.

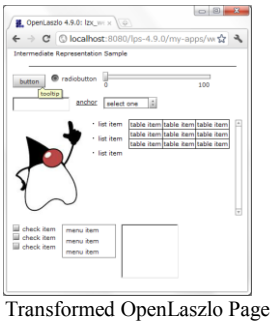

Fig. 9. Transformed OpenLaszlo Page from Intermediate Representation Shown in Fig. 12.

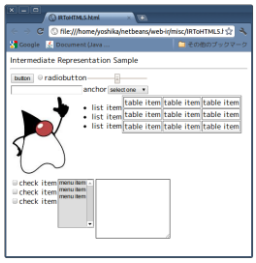

Fig. 10. Transformed HTML5 Page from Intermediate Representation Shown in Fig. 12.

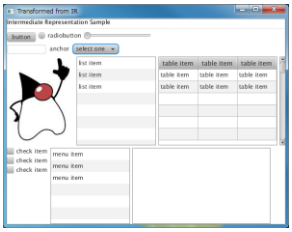

Fig. 11. Transformed JavaFX Page from Intermediate Representation Shown in Fig. 12.

```
<?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>
```

Fig. 12. Sample Intermediate Representation.

V. Framework

A. The Reason for Providing Framework

Since we have decided to use our original intermediate representation, we also need to provide a corresponding framework to ease the transformation between RIAs and the intermediate representation. We have designed our framework to be extensible by using the SPI (Service Provider Interface) pattern [10], which improves flexibility and extensibility; then we can easily change the behavior of the transformation, or add a support for a new RIA technology to the framework, if necessary.

B. Overview

In the transformation process of the framework, all of the RIA information is represented as DOM-like trees, and they are transformed by using the visitor pattern [4]. Fig. 14 (shown in Appendix) is the class diagram of the framework, which currently consists of 135 classes and 19,144 lines of code. Some of the important classes and interfaces are shown in Table III. This class design makes the framework flexible and extensible.

C. Transformation Algorithm

Fig. 13 shows an overview of the transformation algorithm used in the framework, which is based on the visitor pattern. In the figure, we assume that the input and the output are of type Ajax and IR, respectively. In the `visit` method, we can see that each element of the input RIA (in this example, `checkbox` and `text`) requires a few lines of code to perform the transformation; other kinds of information except the behavior information can also be transformed by the same fashion.

When transforming the behavior information, the current support of the framework is not that sufficient to transform the information perfectly. This is so because RIA technologies have their own features, syntaxes, and semantics. Hence, we recommend providing a library that emulates the behavior of the output RIA, as we have done for Flash; such emulation libraries increase the accuracy of the behavior transformation.

D. Other Implementations

Hasegawa, Hayakawa, and Hikita [6] have developed a system by using the framework of Web-IR, which can transform Ajax pages into Flash pages directly by using OpenLaszlo. One of the unique features of the system is that it uses a newly written Ajax emulation library to realize easy transformation with almost equivalent behavior and UI appearances. The system consists of 9 classes and 1,300 lines

TABLE III
Important Classes and Interfaces of Framework.

Class/Interface Name	Description
Application	Represents a RIA.
ApplicationReader	Represents a reader that reads an input as an instance of `Application`.
ApplicationWriter	Represents a writer that writes an instance of `Application` as an output.
Translator	Represents a translator.
VisitorTranslator	Implements `Translator` by using the visitor pattern.
Visitor	Represents visitor used in the visitor pattern.

```
/**
 * Pseudo code to describe the transformation
 * algorithm of the framework. In this example,
 * the input type is Ajax and the output type is IR.
 * @param e0 An HTML tag to be transformed.
 * @return A transformed IR element that is
 *         equivalent to e0.
 */
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;
    ...
  }
}
```

Fig. 13. Overview of Transformation Algorithm (Pseudo Code).

of code. In addition, Yoshika, Hayakawa, and Hikita [12] have developed another system by using the same framework, which can transform Flex pages into HTML5 pages directly. The system consists of 56 classes and 3,181 lines of code.

These implementations show that our framework provides an efficient way of developing RIA transformation systems.

VI. Evaluations

A. Conversion Rate of RIA Information

To evaluate our system, we measured the conversion rate between our intermediate representation and RIA technologies. First, we examined statistics of the kind of RIA technologies by choosing the top 10 sites (shown in Table IV) from Alexa [3], which ranks the top 500 sites by the traffic on the Web (as of September 5th, 2011). As a result, we found that all the top 10 sites use Ajax. Thus, we chose Ajax as input for the evaluation, and we also chose Flex for the sake of comparison. Second, we counted the number of occurrences of HTML tags and CSS properties by using our original crawler, because they represent the UI information of Ajax. Finally, we calculated the conversion rate by comparing both the number of HTML tags and CSS properties that can be transformed into IR, and vice versa.

1) Number of Occurrences of HTML Tags and CSS Properties

Tables V and VI show the number of occurrences of HTML tags and CSS properties, respectively. Table V shows that `<div>` and `<span>` are widely used; this is so because Ajax uses them to identify data areas that are manipulated by JavaScript. Moreover, Table VI shows that simple properties are more commonly used than complex ones.

2) Conversion Rate of Widget Information

Table VII shows the conversion rate of widget information between the input RIAs and the output RIAs through IR. As the table shows, considering into account the number of occurrences of the widgets in the top 10 sites, Web-IR can transform over 90% of information of widgets of the input RIAs except Flex. This is so because Flex has more than twice UI widgets than IR; we can increase the rate by adding the corresponding Flex widgets into IR, if necessary.

3) Conversion Rate of Style Information

Table VIII shows the conversion rate of style information between the input RIAs and the output RIAs through IR. As the table shows, Web-IR can transform only the part of the CSS properties of RIAs except Ajax. This is so because CSS has a large number of properties; although many of them are not commonly used. As well as the widget information, we can increase the rate by the same fashion, if necessary.

TABLE IV
Top 10 Web Sites Ordered by Traffic.

Ranking	Site Name	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/

Source: Alexa Internet (www.alexa.com)

TABLE V
Number of Occurrences of HTML Tags in Top 10 Traffic Sites.

Ranking	Tag Name	Number of Occurrences	Rate (%)
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
11	input (hidden) [a]	54	0.98
12	ul	53	0.97
13	h3	30	0.54
14	h2	29	0.53
15	td	26	0.47
…	…	…	…
		5,457	100.00

[a] `<input>` with the type named "hidden" is used to send a text to Web servers; it is not visible on the screen.

TABLE VI
Number of Occurrences of CSS Properties in Top 10 Traffic Sites.

Ranking	Property Name	Number of Occurrences	Rate (%)
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
…	…	…	…
		16,592	100.00

TABLE VII
Conversion Rate of Widget Information.

Input RIA	Output RIA	Conversion Rate (%)	Conversion Rate Considering Number of Occurrences (%)
Ajax	IR	80.6	93.8
Flex	IR	44.0	N/A [a]
IR	OpenLaszlo	100.0	100.0
IR	HTML5	90.0	N/A [b]
IR	JavaFX	100.0	100.0

[a] This cannot be calculated because we cannot count the number of occurrences, since Flex applications are compiled as binary files.
[b] This cannot be calculated because the occurrences of IR are not available.

TABLE VIII
Conversion Rate of Style Information.

Input RIA	Output RIA	Conversion Rate (%)	Conversion Rate Considering Number of Occurrences (%)
Ajax	IR	100.0	100.0
Flex	IR	33.3	N/A [a]
IR	OpenLaszlo	42.6	54.8
IR	HTML5	100.0	100.0
IR	JavaFX	62.3	95.8

[a] This cannot be calculated because we cannot count the number of occurrences, since Flex applications are compiled as binary files.

B. Comparison of Production Costs

We estimated the number of man-hours by developing prototypes in order to evaluate our system in the following two subsections.

1) Comparison of UI Development Costs

First, we estimated the man-hours with/without using Web-IR. The conditions of the test are the following: the testers developed sample UI pages that correspond to Fig. 9 and Fig. 11 from scratch by using OpenLaszlo and JavaFX, and the testers are new to the RIA technologies. The results show that JavaFX cost 1.0 man-hour as a learning cost, and 3.5 man-hours as a development cost. Likewise, OpenLaszlo cost 1.0 man-hour as a learning cost, and 9.5 man-hours as a development cost. On the other hand, by using Web-IR, the testers completed the development in fifteen minutes. Through this evaluation, we can say that even a simple development of a RIA costs not a small number of man-hours if developers are new to the RIA technology.

2) Comparison of Development Costs of UI Transformation Systems

Table IX shows the production costs of our sample development of RIA transformation systems, which indicates that the man-hours and the lines of code are reduced to approximately 2/3 by using the framework of Web-IR. We consider this reason as follows: the framework provides an efficient way for developers to develop RIA transformation systems, since it is well-designed by using the well-known design patterns such as the visitor pattern and the SPI pattern.

TABLE IX
PRODUCTION COSTS OF DEVELOPMENT OF UI TRANSFORMATION SYSTEMS WITH/WITHOUT USING FRAMEWORK OF WEB-IR.

Input RIA	Output RIA	Using Framework	Man-Hours	Lines of Code
Ajax	OpenLaszlo	No	10.0	1,450
Ajax	OpenLaszlo	Yes	5.0	960
Flex	HTML5	No	24.0	1,818
Flex	HTML5	Yes	18.0	1,231
Ajax	JavaFX	No	20.0	3,156
Ajax	JavaFX	Yes	12.0	1,627

VII. CONCLUSION

In this paper, we have proposed the method that uses the XML-based intermediate representation with the Java-based framework in order to solve the RIA compatibility problems. The results of the evaluations show that our method is sufficient to solve the problem. On the other hand, our current implementation provides limited support for the transformation of behavior information; it remains as a matter to be studied further to provide an effective way to overcome the limit.

APPENDIX

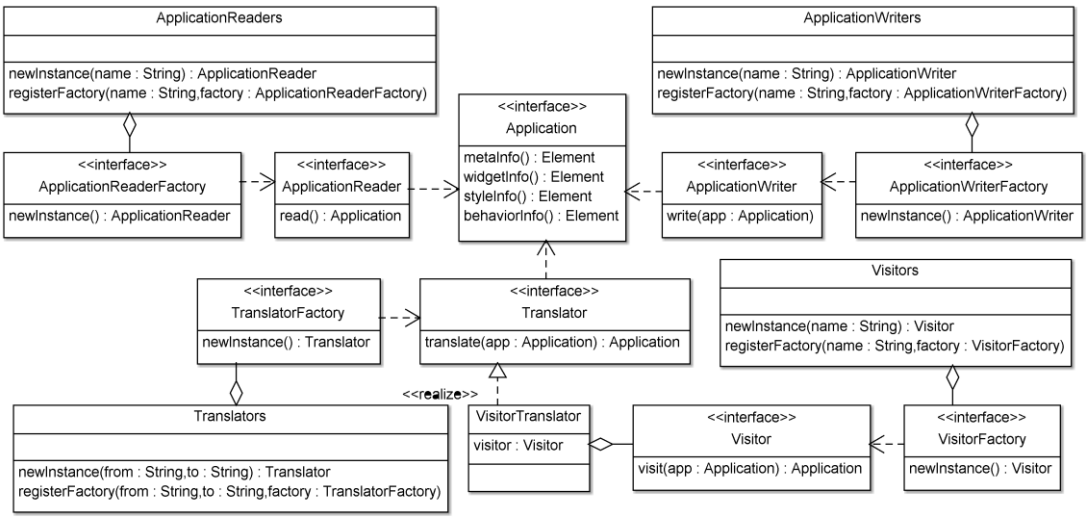

Fig. 14. Class Diagram of Framework of Web-IR.

REFERENCES

[1] Adobe Labs, "Wallaby," Available: http://labs.adobe.com/technologies/wallaby/

[2] Adobe Systems, "Flash to focus on PC browsing and mobile apps; Adobe to more aggressively contribute to HTML5," Available: http://blogs.adobe.com/conversations/2011/11/flash-focus.html

[3] Alexa the Web Information Company, "Top Sites," Available: http://www.alexa.com/topsites

[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design patterns: elements of reusable object-oriented software," Addison-Wesley Professional, 1995.

[5] Google, "Swiffy," Available: http://www.google.com/doubleclick/studio/swiffy/

[6] S. Hasegawa, T. Hayakawa, and T. Hikita, "Implementation and evaluation of Flash translation method for Ajax applications by RIA translation framework," The 74th National Convention of IPSJ, 2012 (in Japanese).

[7] T. Hayakawa, S. Hasegawa, S. Yoshika, and T. Hikita, "Design and evaluation of intermediate representation and framework for maintaining Web applications," DPS-149(12), IPSJ SIG Notes, 2011 (in Japanese).

[8] T. Hayakawa, S. Hasegawa, and T. Hikita, "Design and implementation of intermediate representation and framework for Web applications," 2nd World Congress on Computer Science and Information Engineering (CSIE 2011), Changchun, China, 2011.

[9] Laszlo Systems, "OpenLaszlo," Available: http://www.openlaszlo.org/

[10] Oracle Corporation, "Creating extensible applications with the Java platform," Available: http://java.sun.com/developer/technicalArticles/javase/extensible/

[11] Oracle Corporation, "JavaFX," Available: http://javafx.com/

[12] S. Yoshika, T. Hayakawa, and T. Hikita, "Execution of Flex applications under the iOS platform by HTML5 transformation," FIT2011, 2011 (in Japanese).

[13] P. Yu, K. Kontogiannis, and T. C. Lau, "Transforming legacy Web applications to the MVC architecture," Software Technology and Engineering Practice, 2003.

[14] P. Zhang and J. Chen, "The scheme to extending the Web application frame with XML," International Conference on Innovative Computing and Communication and Asia-Pacific Conference on Information Technology and Ocean Engineering, 2010.

Tomokazu Hayakawa was born in Ichikawa, Japan, in 1982. He earned the B.Sc. and M.Eng. degrees in computer science from Meiji University, Kawasaki, Japan, in 2004 and 2007, respectively.

He joined TG Information Network Co., Ltd., where he worked as a system engineer, as a UNIX engineer, as a software developer, as an R&D engineer, and as a technical trainer. Since 2010, he has been with the School of Science and Technology, Meiji University, where he is currently a research assistant and is pursuing his PhD in computer science. His main areas of research are system engineering, software engineering, and Web applications, especially RIA technologies.

Mr. Hayakawa is a member of the Information Processing Society of Japan.

Shinya Hasegawa was born in Kawasaki, Japan, on December 30, 1987. He earned the B.Sc. degree in 2010 and the M.Eng. degree in 2012, both in information and technology, Meiji University, Kawasaki, Japan.

His most recent publication is: Implementation and evaluation of Flash translation method for Ajax applications by RIA translation framework (Nagoya, Japan: The 74th National Convention of IPSJ, 2012). He is interested in Web technologies and software development.

Shota Yoshika was born in Yamaguchi, Japan, on November 14, 1987. He earned the B.Sc. degree in 2011 in information and technology, Meiji University, Kawasaki, Japan, where he is currently in the master's course.

His most recent publication is: Execution of Flex applications under the iOS platform by HTML5 transformation (Hakodate, Japan: Forum on Information Technology 2011). He is interested in Web technologies and their translators.

Teruo Hikita (M'79) was born in Nishinomiya, Japan in 1947. He earned B.Sc. in 1970, M.Sc. in 1972, and D.Sc. in 1978, all from University of Tokyo.

He was at University of Tokyo and Tokyo Metropolitan University, and now is at Dept. of Computer Science of Meiji University in Kawasaki, Japan for 20 years. His current research interests are in Web technologies.

Professor Hikita is a member of ACM, IPSJ, and SIAM.