Georgia State University
Digital Archive @ GSU

Computer Science Theses | Department of Computer Science

5-3-2007

An Automated XML-Based Webform Management System

Piyaphol Phoungphol
ppiyaphol@gmail.com

Follow this and additional works at: http://digitalarchive.gsu.edu/cs_theses

Recommended Citation
Phoungphol, Piyaphol, "An Automated XML-Based Webform Management System" (2007). *Computer Science Theses.* Paper 41.

This Thesis is brought to you for free and open access by the Department of Computer Science at Digital Archive @ GSU. It has been accepted for inclusion in Computer Science Theses by an authorized administrator of Digital Archive @ GSU. For more information, please contact digitalarchive@gsu.edu.

AN AUTOMATED XML-BASED WEBFORM MANAGEMENT SYSTEM

by

PIYAPHOL PHOUNGPHOL

Under the Direction of Rajshekhar Sunderraman

ABSTRACT

In a web application, “webform” plays an important role in providing interactions between users and a server. To develop a webform in conventional method, developers have to create many files including HTML-JavaScript, SQL script, and many server-side programs to process to data.

In this thesis, we propose a new language, Webform Language (WFL). WFL can considerably decrease developing time of webform by describing it in XML and a parser will automatically generate all necessary files. In addition, we give an option for user to describe a webform in another language, called Simple Webform Language (SWFL). The syntax of a SWFL is simple and similar to the “CREATE TABLE” statement in SQL. When a parser parse webform description in SWFL, it translated the description to WFL first, and then processed it by as normal WFL.

INDEX WORDS: web application, Webform Language, Simple Webform Languge, XML, SQL, webform

AN AUTOMATED XML-BASED WEBFORM MANAGEMENT SYSTEM

by

PIYAPHOL PHOUNGPHOL

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Science

in the College of Arts and Sciences

Georgia State University

2007

Copyright by

Piyaphol Phoungphol

2007

AN AUTOMATED XML-BASED WEBFORM MANAGEMENT SYSTEM

by

PIYAPHOL PHOUNGPHOL

Major Professor: Rajshekhar Sunderraman

Committee: Ying Zhu

A. P. Preethy

Electronic Version Approved:

Office of Graduate Studies

College of Arts and Sciences

Georgia State University

May 2007

ACKNOWLEDGEMENTS

I wish to record my sincere thanks to my advisor, Dr. Raj Sunderraman, for his guidance insightful suggestions and support. It is great honor for me to be able to work with Dr. Sunderraman and I am thankful for having an advisor like him. I would also like to thank Dr. A.P. Preethy, and Dr. Ying Zhu for serving as my advisory committee and taking precious time to review my thesis.

I wish to use this opportunity to express my thanks to my wife, Inthira Srivrunyoo, for her support for without it I would have never been able to achieve this today. I would also like to thank my father, Anek Phoungphol, and my mother Orasa Phoungphol, fro encouraging me to reach this far.

I would like to thank the Computer Science Department for this support during my gradate studies. I also thank all my friends and instructors at GSU for making these three years an enjoying experience.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iv

LIST OF TABLES vii

LIST OF FIGURES viii

LIST OF ABBREVIATIONS x

CHAPTER

1. INTRODUCTION 1

2. BACKGROUND 3

Webform 3

Application for Graduation 4

Webform Processes 7

3. WEBFORM LANGUAGE 12

Complexity in Creating Webform 12

Webform Language 15

Webform Database Schema 16

Webform Main View 17

4. WEBFORM TAGS AND SYSTEM IMPLEMENTATION 19

Webform Tags 19

Application for Graduation in WFL 28

System Implementation 30

5. SIMPLE WEBFORM LANGUAGE 35

Create Webform Syntax 35

Create Webreport Syntax 39

Create Webreport Syntax 39

Application for Graduation in SWFL 40

6. WEBFORM EXAMPLES 41

Country Survey Form 41

Conference Survey Form 43

7. CONCLUSION AND FUTURE WORK 48

REFERENCES 49

APPENDICES 50

A. Webform Parser Source Code 50

B. Lexical Specification and Grammar of SWFL 93

LIST OF TABLES

Table 4.1 All tags in WFL 19

Table 4.2 Attributes of <webform> tag 20

Table 4.3 Attributes of <textfield> tag 20

Table 4.4 Attributes of <textarea> tag 21

Table 4.5 Attributes of <password> tag..... 22

Table 4.6 Attributes of <selectOne> tag 22

Table 4.7 Attributes of <selectMany> tag 23

Table 4.8 Attributes of <radio> tag 23

Table 4.9 Attributes of <checkbox> tag 24

Table 4.10 Attributes of <optionItem> tag 25

Table 4.11 Attributes of <submit> tag 25

Table 4.12 Attributes of <reset> tag 25

Table 4.13 Attributes of <if> tag 26

Table 4.14 Attributes of <when> tag 26

Table 4.15 Attributes of <resultTable> tag..... 27

Table 4.16 Attributes of <resultTexte> tag..... 27

Table 4.17 Comparisons of data types and constraints between WFL and MySQL 31

Table 4.18 Special keywords in webform properties file 32

LIST OF FIGURES

Figure 2.1 Application for Graduation Webform Example 4

Figure 2.2 Application for Graduation webform: a student selects to attend ceremony 5

Figure 2.3 A sample html file of Application for Graduation 6

Figure 2.4 Relationships between webform processes 7

Figure 2.5 JavaScript for Application for Graduation 8

Figure 2.6 PHP script to process Application for Graduation webform..... 11

Figure 3.1 JavaScript to validate required fields of Application for Graduation..... 13

Figure 3.2 SQL script for Application for Graduation 14

Figure 3.3 PHP script to show report for Application for Graudation..... 15

Figure 3.4 A database schema for Application for Graduation 17

Figure 3.5 A main view for Application for Graduation 18

Figure 4.1 Application for Graduation webform in WFL..... 29

Figure 4.2 System implementation of a webform parsers 30

Figure 4.3 A properties file of Application for graduation..... 32

Figure 4.4 A webpage to show a database of Application for Graduation..... 33

Figure 4.5 A webreport of Application for Graduation 34

Figure 4.6 A webreport for Application for Graduation in WFL 34

Figure 5.1 Create webform Syntax 36

Figure 5.2 Create webreport syntax 39

Figure 5.3 Application for Graduation in SWFL 40

Figure 6.1 Country Survey webform when a user selects other countrys 41

Figure 6.2 Country Survey webform Figure when a user selects USA 42

Figure 6.3 Country Survey webform in SWFL 42

Figure 6.4 A webreport described in SWFL to find students from China 42

Figure 6.5 Conference 2006 Survey webform 43

Figure 6.6 Conference webform: a student and selects to attend a banquet 44

Figure 6.7 Conference webform: a full-time registration attendee buys extra tickets 44

Figure 6.8 Conference 2006 Survey webform in WFL 46

Figure 6.9 Statistic page of Confenrece 2006 Survey 46

Figure 6.10 A statistic report for Conference 2006 Survey in WFL 47

LIST OF ABBREVIATIONS

HTML	HyperText Markup Language
LALR	Lookahead LR
PHP	PHP: HyperText Proprocessor
SQL	Structured Query Language
SWFL	Simple Webform Language
WFL	Webform Language
XML	Extensible Markup Language

1. INTRODUCTION

A webform is one of the most common and necessary elements of a webpage. It allows users to enter requested information and then submit it for processing at a server. Webform appears in almost every html pages in a form of login page, survey form, registration form, sign-up form, ect. There is two type of webform: one is a form that process data by just storing it in the database such as registration form and survey form. The other one is a form that process data without saving it into database such as login form and query form.

However, processes to creating a webform is not simple. First, we have to create html form to display form and take inputs from users. Sometimes a form requires dynamic html that will change form elements depend on the values user enters. For example, if the age user enters is greater that 20, a form will ask his/her metrical status, otherwise this question will not be displayed. Moreover, most html form usually need JavaScript to validate user inputs. Next, we also need server-side script such as Java Servlet, ASP, PHP, CGI scripts to process data that a user submitted. In addition, we might need database to stored what user enters. To create this database, we must create tables that match with the elements of webform.

In this thesis, we concentrate in developing webform to receive inputs from users and store it into database. We propose a new approach to create webform by using Webform Language (WFL). WFL uses xml language to describe elements and constraints of a webform. Then we develop a parser program to convert a form's description in WFL to all necessary files to implement that webform including html/JavaScript form, SQL script to create tables in the database, server-side script to save all data submit in a form to the databases, and sever-side script to show data in the database.

The outline of this thesis is as follows: In chapter1, we give an introduction to this thesis. In chapter 2, we introduce a webform, and processes relating to the webform to the reader. In Chapter 3, we show complexity problem in creating webform and introduce a new technique to create webform by using Webform Language(WFL). In Chapter 4, we explain about special tags in WFL and how webform parser is implemented. In chapter 5, we describe a Simple Webform Language(SWFL), which is a simpler version of WFL for a simple form. In the Chapter 6, we show two webform examples that are described by SWFL and WFL. In Chapter 7, the conclusion and possible future works are listed. Chapter 8 provides the references and chapter 9 the code for our system

2. BACKGROUND

In this chapter, we give a background of a webform and its elements. We also introduce Application for Graduation webform, which we will use as an example webform through this thesis. Then, we explain all processes relating to a webform and webform development.

2.1 Webform

A webform is a web page that mimics the usage of paper form by allowing a user to enter data online and send it to a server for processing [1]. Forms can be used to submit data to save on a server (e.g., ordering a product) or can be used to retrieve data (e.g., searching on a search engine).

A form in XHTML or HTML [2] is by far the most common way to use a form online. The following elements can make up the user-inputting portion of a form:

- input field
    - text – a simple textfield that allows input of a *single* line of text
    - checkbox – a check box
    - radio – a radio button
    - submit – a button that tells the browser to send data to a server
    - reset - a button that tells the browser to reset all user input to its default value.
- textarea – much like the text input field except a textarea allows for multiple rows of data to be shown and entered
- select – a drop-down list that displays a list of items a user can select from

2.2 Application for Graduation

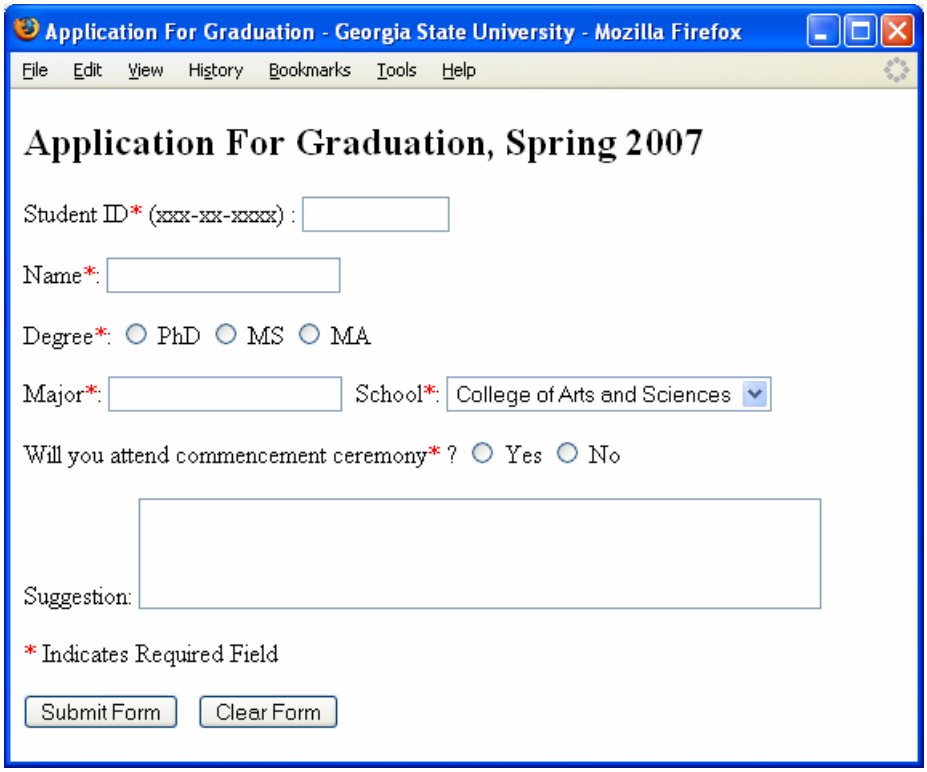


Figure 2.1 Application for Graduation Webform Example

Application for Graduation webform, as shown in Figure 2.1, is a webform example that we will use through this thesis. It is a simple form to collects students information, their plans to attend commencement ceremony and suggestions. Below are requirements of this webform.

1. The student ID, name, degree, major, schools, the number of tickets, and plan to attend the ceremony fields are required.
2. The student ID must be number in xxx-xx-xxxx format
3. If a student plans to attend commencement ceremony, the form will also ask the number of tickets and ceremony accessories that they want to order. By default, the

form should hide these questions and show them only if a student selects to attend the ceremony.

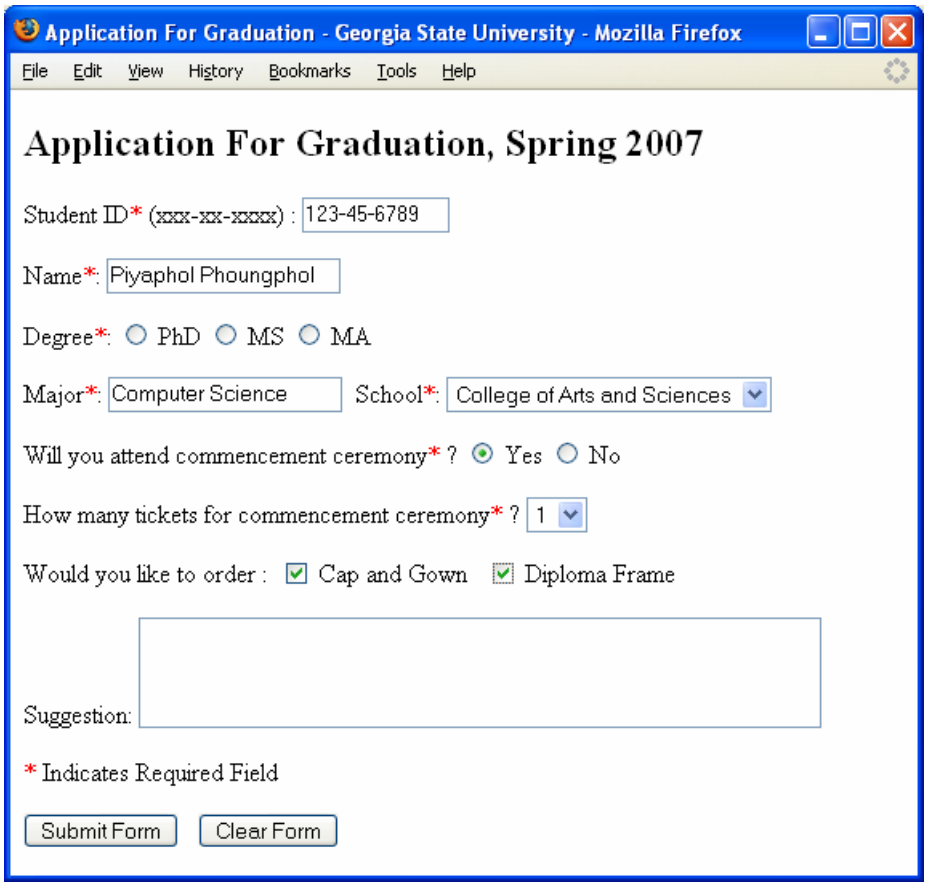


Figure 2.2 Application for Graduation webform: a student selects to attend ceremony

In the form, it contains many html elements including:

- **Text fields** : asking student for their student id, name, and major
- **Radio buttons** :asking student to pick his/her degree and plan to attend ceremony
- **Select list** : asking student's school name, and the number of ceremony tickets
- **Check boxes** : asking student to click on if student want to order Cap and Gown or diploma frame.
- **Textarea** :asking student suggestions

- **Submit button** : to send the form to the server
- **Reset button** : to clear the form

Below is html code for the Application for Graduation webform.

```
<html>
   <head>
         <title>Application For Graduation - Georgia State University</title>
   </head>
   <body>
   <h2>Application For Graduation, Spring 2007 </h2>
   <form action="processForm.php">
       <p>Student ID* : <input name="studentid" type="text" size="11"
          maxlength="11" /></p>
      <p>Name*: <input type="text" name="name" /></p>
      <p> Degree*:
         <input name="degree" type="radio" value="phd" /> PhD
         <input name="degree" type="radio" value="ms" />  MS
         <input name="degree" type="radio" value="ma" />  MA
      </p>
      <p> Major*:
         <input type="text" name="textfield3" />
         School*:
         <select name="school">
               <option value="artScience">College of Arts and
         Sciences</option>
               option value="education">College of Education</option>
               <option value="law">College of Law</option>
         </select>
      </p>
      <p>Will you attend  commencement ceremony* ?
         <input name="willAttend" type="radio" value="yes" />  Yes
         <input name="willAttend" type="radio" value="no" />  No
      </p>
         <div id="accessoryQuestion">
               <p>How many tickets for commencement ceremony ?
                     <select name="numTicket">
                           <option value="1">1</option>
                           <option value="2">2</option>
                           <option value="3">3</option>
                     </select>
               </p>
               <p>Would you like to order :
                     <input type="checkbox" name="accessories"
                           value="cap" /> Cap and Gown
                     <input type="checkbox" name="accessories"
                           value="frame" />Diploma Frame
               </p>
         </div>
         <p>Suggestion:  <textarea name="textarea" cols="50"
               rows="3"></textarea></p>
               <p> * Indicates Required Field</p>
         <input type="submit" value="Submit Form" /> 
         <input type="reset" value="Clear Form" />
   </form>
   </body>
</html>
```

Figure 2.3 A sample html file of Application for Graduation

2.3 Webform Processes

In this section, we will explain all processes relating to html webform. These processes are shown in Figure 2.4.

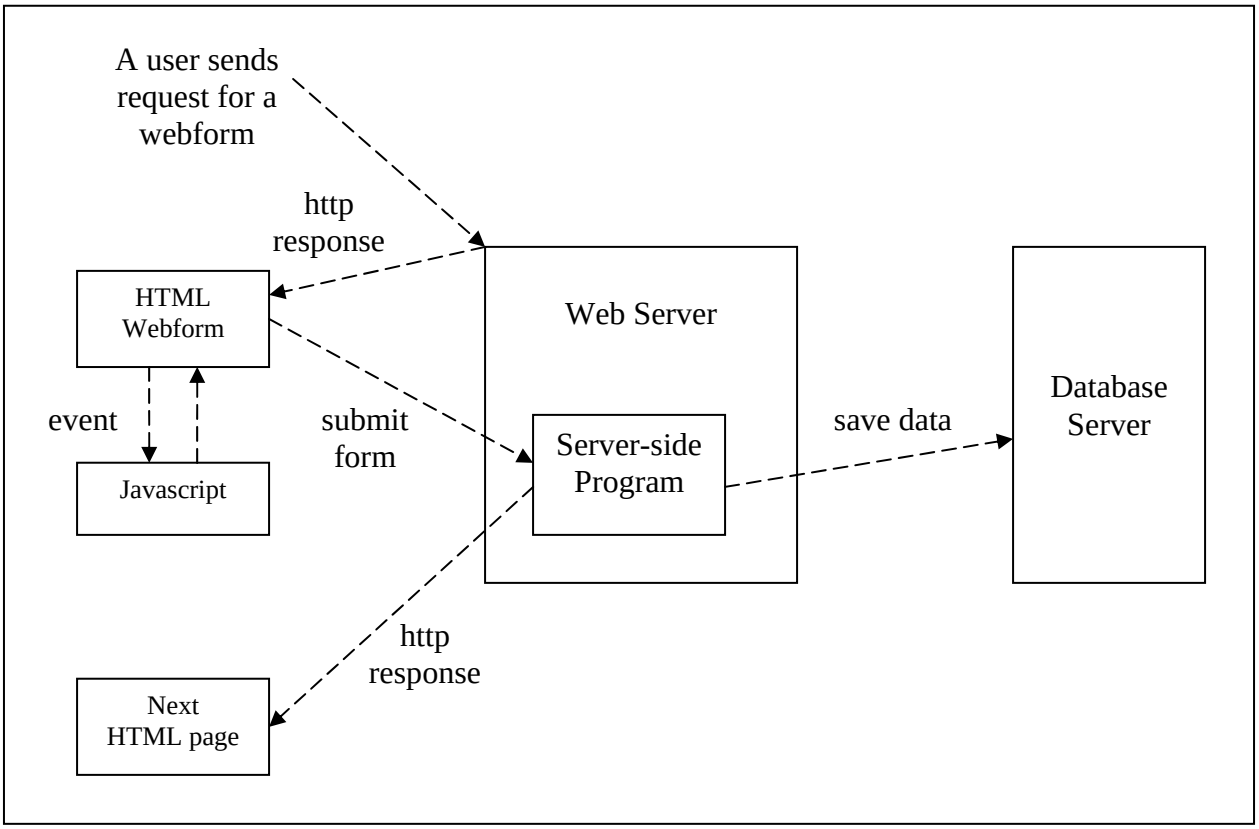


Figure 2.4 Relationships between webform processes.

2.3.1 Display HTML webform

When a user make a request to webform page by typing webform URL directly, or click a link from other pages, a web browser will send HTTP request to a web server. Then the web server receiving the request will send respond packet containing webform information in html language back to the user. Finally, the browser will render that html message as webform page to a user.

2.3.2 Modify and validate webform by client side script

After a webform is displayed, a user can enter the requested information to a webform. Sometimes entering values or selecting an options in webform might fire an event, which triggers a java script to handle that event. For example in html code of application for graduation webform shown below, an onchange event will fire when a user select or change a value of "willAttend" radio box element. The onchange event will call "addQuestion" function in Figure 2.5 to modify webform by adding the number of tickets and ceremony accessory questions.

```
<p>
       Will you attend commencement ceremony* ?
       <input name="willAttend" id="willAttend" type="radio" value="yes"
              onchange="addQuestion( )"/>Yes
       <input name="willAttend" id=" willAttend" type="radio" value="no"
              onchange="addQuestion( )"/>No
       <div id="accessoryQuestion"/>
</p>

. . .

<script type="text/javascript">
   function addQuestion ( ){

       var accessoryQuestionHTML = '<p>How many tickets for commencement ceremony ? '
              + '<select name="numTicket">'
                     + '<option value="1">1</option>'
                     + '<option value="2">2</option>'
                     + '<option value="3">3</option>'
              + '</select>'
              '</p><p>Would you like to order : '+
                     '<input type="checkbox" name="accessories" '
                            + 'value="cap" />Cap and Gown '+
                     '<input type="checkbox" name="accessories"
                            + 'value="frame" />Diploma Frame </p>';

       var answer = document.getElementById("willAttend");
       var accessoryQuestionDiv = document.getElementById("accessoryQuestion ");

       if(answer=="yes"){
              // add question
              accessoryQuestionDiv.innerHTML = accessoryQuestionHTML;
       }else{
              // remove question
              accessoryQuestionDiv.innerHTML =  ' ';
       }
   }
</script>
```

Figure 2.5 JavaScript for Application for Graduation

Utilizing JavaScript on the Document Object Model (DOM) leads to the method of Dynamic HTML that allows dynamic creation and modification of a web page within the browser. Moreover, JavaScript usually can validate user inputs before sending it to the server. As in the Application for graduation webform, the student id, name, degree, major, schools, the number of tickets and plan to attend the ceremony fields must be entered or selected before the form can be submitted, otherwise the form will show alert message to let a user knows.

2.3.3 Submit webform

Webform will be send to server to process when a user clicks on submit buttons. All user inputs will be sent as parameters in HTTP request to server-side program that are specified in the *action* attribute action of the webform. If the *method* attribute of a form is POST method, all parameters and their value will be sent within the http request packet, otherwise a browser will append all parameters and their value with URL. After a web server receives a http request packet, it will parse that http request to specified server-side program located inside the web server.

2.3.4 Process data by server-side program

Server-side programs can do a vast assortment of tasks to create dynamic web sites such as authenticating a login, checking out shopping carts, making a reservation. Anyway, most of services provided by server-side program usually interact with database by retrieving and storing data in a database. Server side program can be a scripting language, which embeds source code in html such as PHP, Perl, and Cold Fusion, or it can be compiled programs such as C, Java Servlet.

In this thesis, we will use PHP [3] to implement server-side script because it is an Open Source general-purpose scripting language and widely used in web application. Much of its

syntax is borrowed from C, Java and Perl with a couple of unique PHP-specific features. The goal of PHP language is to allow web developers to write dynamically generated pages quickly. All PHP script must be contained within <?php … ?> tag and it will be interpreted by PHP interpreter at a web server.

For database server, we use MySQL [4], which is the most popular open source database server, to stored a user input. Furthermore, MySQL is very commonly used in conjunction with PHP scripts to create dynamic and powerful server applications. MySQL has been criticized in the past because it does not have all the features of other Database Management Systems. However, MySQL continues to improve significantly, with each major upgrade, and has great popularity because of these improvements.

PHP script as shown in Figure 2.6 is some part of PHP script for Application for Graduation to read all student information from HTTP request, then save it to a database. All student information except number of tickets and accessories parameter will be stored in main table named GRADUATE_APP. The number of ticket parameter will be saved in TICKET table and accessories parameter will be saved in CEREMONY_ACCESSORY table. Both TICKET and CEREMONY_ACCESSORY tables will have studentid column, which references to other student information in the main table.

```
<?php
       . . .

   // insert student information to GRADUATE_APP table
   $query = "insert into GRADUATE_APP values ('$_POST[studentid]',"
             + "'$_POST[name]', '$_POST[degree]', '$_POST[major]', '$_POST[school]', "
             + "'$_POST[willAttend]', '$_POST[suggestion]' ) ";
    mysql_query($query);

   // insert the number of tickets to TICKET table
   If(isset($_POST['numTicket'])){
      $query = "insert into TICKET values ('$_POST[studentid]', _POST[numTicket]);
   mysql_query($query);
   }

   // if student select to attend commencement ceremony
```

```
    If(isset($_POST['accessories'])){

        // add each accessory to CEREMONY_ACCESSORY table
        foreach($_POST['accessories'] as $accessory) {
               $query = "insert into CEREMONY_ACCESSORY values ("
                      + "'$_POST[studentid]',  '$accessory' ) ";
               mysql_query($query);
        }
    }

        . . .
?>
```

Figure 2.6 PHP script to process Application for Graduation webform

3. WEBFORM LANGUAGE

In this chapter, we will show how to develop the Application for Graduation example introduced in the last chapter by conventional method. After that, we introduce a new approach in creating webform by using Webform Language(WFL) that will significantly reduce time and effort of developer. At the end of this chapter, we also explain about database schema and main view that will use to store data from webform.

3.1 Complexity in Creating Webform

As described in the previous chapter, in order to developing a webform, even a simple one, it relates to many processes. To develop a webform, we have to create many files/program to implement each of those processes. Below are general files we usually need in creating a webform that collects data from a user and then saves it to a database:

3.1.1 HTML file

First, we need an html file to display a webform to a user with a JavaScript to dynamically modify webform, and validate user inputs. Creating a simple HTML [2] page is quite easy to create by text editor or any html editor such as front page, and dream weaver. However, creating dynamic html webform is difficult task. We have to modify DOM tree of html page, or validate user inputs by JavaScript. Writing a JavaScript is time-consuming job and tends to have many errors because it does not have efficient tools or software to help in developing JavaScript code. In an Application for graduation example, JavaScript to show extra questions when student select to attend ceremony are shown in Fig. Below is a sample JavaScript code to validate all field of Application for Graduation before sending the form to the server.

```
//function to validate required fields before sending a form
function validateForm(){
	// validate student id
	if(document.getElementById('studentid')==''){
		alert('Student ID is required');
		return false;
	}else if(!document.getElementById('studentid').value.match(/^\d{3}-\d{2}-\d{3}$/)){
		alert('Student Id must be in xxx-xx-xxx format');
		return false;
	}

	// validate name
	if(document.getElementById('name')==''){
		alert('Name is required');
		return false;
	}

	// validate degree
	if(!isRadioSelected('degree')){
		alert('Degree is required');
		return false;
	}

	// validate major
	if(document.getElementById('major')==''){
		alert('Major is required');
		return false;
	}

	// validate will attend
	if(!isRadioSelected('willAttend')){
		alert('Please select to attend ceremony or not');
		return false;
	}

	return true;
}

// function to checked if radio button is selected
function isRadioSelected(elementId){
	var radioElement = document.getElementById(elementId);
	var isSelected = false;
	for (int i=0;i<radioElement.length;i++)
		if(radioElement[i].checked)
			isSelected = true;

	return isSelected;
}
```

Figure 3.1 JavaScript to validate required fields of Application for Graduation

3.1.2 SQL script

Next, We need SQL script file to create tables in a database for storing user inputs. Those tables must have their columns and data type corresponding to a webform elements. Below is sample SQL script to create tables in MySQL [4] for the Application for Graduation example.

```
CREATE TABLE GRADUATE_APP (
       studentID    VARCHAR(11) NOT NULL,
       name                VARCHAR(255) NOT NULL,
       degree       VARCHAR(255) NOT NULL,
       major        VARCHAR(255) NOT NULL,
       school       VARCHAR(255) NOT NULL,
       willAttend   VARCHAR(255) NOT NULL,
       suggestion   VARCHAR(255) NOT NULL,
       PRIMARY KEY(studentID)
) TYPE=InnoDB;
CREATE TABLE TICKET (
       studentID    VARCHAR(11) NOT NULL,
       numTicket    INT NOT NULL,
       FOREIGN KEY(studentID) REFERENCES GRADUATE_APP (studentID)
       ON UPDATE CASCADE ON DELETE CASCADE,
       PRIMARY KEY(studentID)
) TYPE=InnoDB;
CREATE TABLE CEREMONY_ACCESSORY (
       studentID    VARCHAR(11) NOT NULL,
       accessory    VARCHAR(255) NOT NULL,
       FOREIGN KEY(studentID) REFERENCES GRADUATE_APP (studentID)
       ON UPDATE CASCADE ON DELETE CASCADE,
       PRIMARY KEY(studentID, accessory)
) TYPE=InnoDB;
```

Figure 3.2 SQL script for Application for Graduation

3.1.3 PHP program to process webform

To read data submitted by a user, we need PHP [3] Script to process data and then store it into database. The PHP script will be read parameters from a HTML request and insert those data to the database. This process is easy and general for a webform that collects a user information such as a survey form, a registration form, and ect. However, a PHP script and data schema still depends on webform elements. PHP script file to process data of the Application for Graduation webform as shown in Figure 2.6.

3.1.4 PHP program to show report of a webform

Sometime, looking a webform data directly in a database is not convenient. We usually PHP program to show report or result of a webform to administrator. The report can be general report to show all data in databases or specific requested information from webform data. This PHP script will execute a query to retrieve data from database and generate html page to show its result to a user. Below is sample PHP script to show a list of student in College of Law who will attend commencement ceremony for Application for Graduation webform.

```
<?php

        . . .

        $query = "select name from GRADUATE_APP " +
                "where school = 'law' and willAttend = 'yes' order by name";
        $result = mysql_query($query) or die(mysql_error());

        print "<ol>";
        while($row = mysql_fetch_array($result)){
               print "<li>$row[name]</li>";
        }
        print "</ol>";

        . . .

?>
```

Figure 3.3 PHP script to show report for Application for Graudation

3.2 Webform Language

As explained in a previous section, it is complicated that we need to create many files to create only one simple webform. In this section, we propose a new technique to solve the complex problem of creating webform. We will use Web Form Language(WFL), which is Extensible Markup Language (XML) [5] to describe all requirements of a form and then use xml parsers to generate each files to implement the webform. The obvious advantages of describing a webform in xml is that it is self-describing and easy to understand. Moreover, most programming languages have application-programming interface (API) to support processing of

xml document. With WFL, we only need to create one xml describing a webform and then the other files will be automatically generated by xml parsers.

WFL is general html document with special tags to describe form elements. It also supports a condition that allows webform to dynamically hide or show its elements. Moreover, we can use WFL to describe the report that an administrator want to know about web form. A webform and report described in web form language will be parsed by xml parsers and then those parsers will generate corresponding html file with javascript, SQL script file , and PHP scripts to implement the webform. Tags in WFL consist of three types of special tags beside from normal html tags: special inputs tags, conditionals tags, and form result tags. We will discuss more details of these special tags later in next chapter.

3.3 Webform Database Schema

All user inputs of a webform will be stored in database schema that matches with webform elements. All normal webform elements will be stored in MainTable, except multi-valued elements and conditional elements. Each Multi-valued element such as multi-select list or checkboxes must be stored in another table, which has a key of Main table and multi-values element as primary keys. For each condition, all elements in that condition except other condition element or multi-valued element will also be stored together in another table, and use a key of main table as a primary key for that table. All multi-valued element in condition will be stored separate from other elements as mention early. For Application for Graduation example, all fields except the number of tickets and ceremony accessories will be stored in main table name GRADUATE_APP. The number of ticket and ceremony accessories are in the same condition, however ceremony accessories is multi-valued element. Consequently, both element

will stored separately. The database schema for Application for Graduation webform are show in Figure 3.4

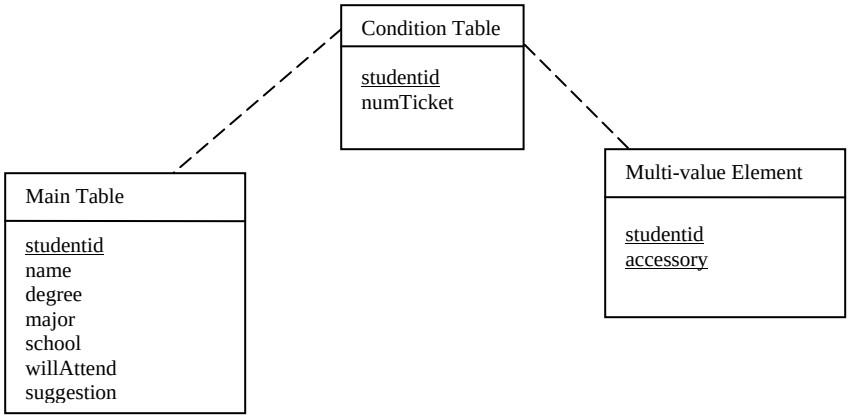


Figure 3.4 A database schema for Application for Graduation

3.4 Webform Main View

When a web form is complex or contains many conditions, its database schema will be complicated and it is difficult for administrator to remember all tables' names. For simplicity, we will create a main view of a webform which corresponds with original form structure by joining all tables except multi-values tables together with primary key of the webform. With a main view, administrator can easily query or access all single-valued element of a webform. In Application for Graduation webform example, the following SQL statement will generate a main view.

```
CREATE Main_View AS
SELECT MainTable.*, numTicket
FROM MainTable LEFT JOIN ConditionTable
       ON MainTable.studentid = ConditionTable. studentid
```

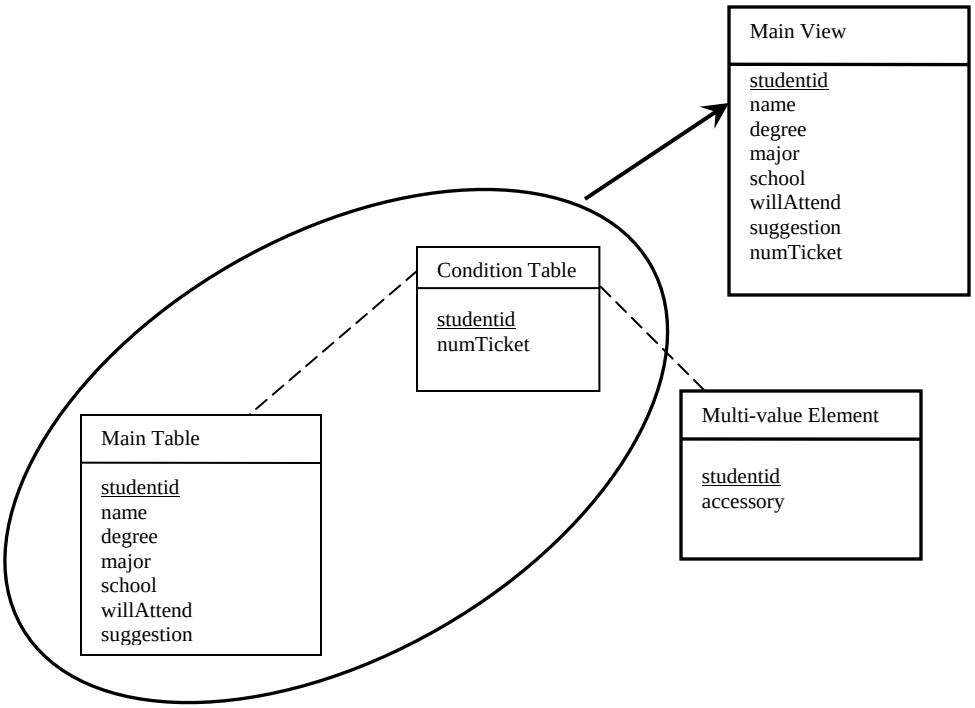


Figure 3.5 A main view for Application for Graduation

4. WEBFORM TAGS AND SYSTEM IMPLEMENTATION

In WFL, we can describe webform and its report by using normal html tags special webform tags. In this chapter, we give more details about special tags in WFL and show how to describe Application for Graduation webform example in WFL. At the end of this chapter, we demonstrate how webform parser can generate all necessary files for a webform.

4.1 Webform Tags

Special tags in web form languages are categorized into three groups: input form tags, conditional tags, and report tags.

Table 4.1 All tags in WFL

Tag Type	Tag Name	Descriptions
Input Tags	webform	A webform Form element
	textField	A Input Field element
	textArea	A textarea element
	password	Render A Password Input Field
	selectOne	A Select Element without the "multiple" attribute.
	selectMany	A Select Element with multiple selections
	radio	A group of Radio button Elements
	checkbox	A group of Checkbox Elements
	optionItem	Sub tag of <selectOne>, <selectMany>, <radio>, and <checkbox>. It represents a option of multi-choice elements
	submit	A Submit Button
	reset	A Reset Button Input Field

Report tags	webreport	A form report
	resultTable	A result table of query
	resultText	A result text of query
Conditional Tags	if	Simple conditional tag that evaluates its body only if the Boolean expression is true
	choose	Multi-conditional tag that uses in conjunction with <when> and <otherwise>.
	when	Sub tag of <choose> that includes its body if its condition is evaluated to true
	otherwise	subtag of <choose> that follows <when> tags and runs only if all of the prior conditions evaluated to false

4.1.1 Special input tags

<webform> tag

The <webform> tag describes a webform page, which can content other element tags such as normal html tags, special input tags, and conditional tags. This tag will be rendered as html page with a form containing other elements of a webform.

Table 4.2 Attributes of <webform> tag

Attribute	**Description**
id*	id of webform
key*	list of webform primary key

* indicates required attribute.

<textField> tag

The <textField> tag renders as HTML input element of the type text. This tag can validate user input with data type or regular expression.

Table 4.3 Attributes of <textfield> tag

Attribute	**Description**

id*	identification of this element.
type	data type of this element in a database(integer, decimal, string). The default data type is string.
size	size of text field
maxlength	maximum number of characters for text input
required	specify whether the user input for this text field is required or not (‘true’ or ‘false’). The default value is false
pattern	regular expression to validate user input

* indicates required attribute.

<textarea> tag

The <textarea> tag renders as HTML textarea. It can validate user input with data type or regular expression.

Table 4.4 Attributes of <textarea> tag

Attribute	**Description**
id*	identification of this element.
type	data type of this element in a database(integer, decimal, string). The default data type is string.
cols	the number of columns visible in the text-area
rows	the number of rows visible in the text-area
required	specify whether the user input for this text field is required or not (‘true’ or ‘false’). The default value is false.
Pattern	regular expression to validate user input

* indicates required attribute.

<password> tag

The <password> tag renders as HTML input element of the type password. This tag can also validate user input with data type or regular expression.

Table 4.5 Attributes of <password> tag

Attribute	Description
id*	identification of this element.
type	data type of this element in a database(integer, decimal, string). The default data type is string.
size	size of password field
maxlength	maximum number of characters for password field
pattern	regular expression to validate user input

* indicates required attribute.

<selectOne> tag

The <singleOne> tag renders as HTML select element without the "multiple" attribute. A user can specify data type or required attributes.

Table 4.6 Attributes of <selectOne> tag

Attribute	Description
id*	identification of this element. The value must be unique.
type	data type of this element in a database(integer, decimal, string). The default data type is string.
size	the number of visible items in the drop-down list
required	specify whether the selection is required

	(‘true’ or ‘false’). If it is ‘true’, one of options must be selected. The default value is false

* indicates required attribute.

<selectMany> tag

The <selectMany> tag renders as HTML select element with multiple selections. This tag can validate the minimum number and maximum number of selected options. User inputs for the multi-select element will stored in a separated table with composite keys of webform primary keys and the multi-select element.

Table 4.7 Attributes of <multiSelect> tag

Attribute	Description
id*	identification of this element.
type	data type of this element to be stored in database(integer, decimal, string). The default data type is string.
size	the number of visible items in the drop-down list
minSelected	the minimum number of options that must be selected by user.
maxSelected	the maximum number of options that can be selected by user.

* indicates required attribute.

<radio> tag

The <radio> tag renders a table of HTML "input" elements of the type "radio".

Table 4.8 Attributes of <radio> tag

Attribute	Description
id*	identification of this element. The value must be unique.

Type	data type of this element to be stored in database(integer, decimal, string). Default data type is string.
required	specify whether the selection is required (‘true’ or ‘false’). If it is ‘true’, one of options must be selected. The default value is false

* indicates required attribute.

<checkbox> tag

The <checkbox> tag renders a list of HTML checkboxes. User input for the checkbox element will stored in a separated table with composite keys of webform primary keys and the checkbox element.

Table 4.9 Attributes of <checkbox> tag

Attribute	**Description**
id*	identification of this element.
Type	data type of this element to be stored in database(integer, decimal, string). The default data type is string.
minSelected	the minimum number of options that must be selected by user.
maxSelected	the maximum number of options that can be selected by user.

* indicates required attribute.

<optionItem> tag

The <optionItem> tag is subtag of <selectOne>, <selectMany>, <radio>, and <checkbox>. It render as option of the corresponding multi-choice element. If a label of an option is not specified, it will show values as its label.

Table 4.10 Attributes of <optionItem> tag

Attribute	Description
value*	a value of the option to be sent to the server
label	a Label of this option
selected	Specifies whether an option should be selected when it is first appear (true or false). The default value is false.

* indicates required attribute.

<submit> tag

A <submit> tag renders a HTML input button of type submit.

Table 4.11 Attributes of <submit> tag

Attribute	Description
value	A label of a submit button
test	a boolean expression that must be evaluated to be true before a webform will be submitted.

<reset> tag

A <reset> tag renders a HTML input button of type reset.

Table 4.12 Attributes of <reset> tag

Attribute	Description
value	the text that is displayed on a reset button

4.1.2 Conditional tags

<if> tag

<if> tag is simple conditional tag, which renders its body only if the supplied condition is true. The inputs of any elements in the body of if tag except checkbox and multi-select will be stored in a new conditional table which has a primary key of a webform as primary key.

Table 4.13 Attributes of <if> tag

Attribute	Description
test*	boolean expression that determines whether the body content should be rendered.

* indicates required attribute.

<choose> tag

<choose> tag is multi-conditional tag that establishes a context for mutually exclusive conditional operations, marked by <when> and <otherwise>

<when> tag

<when> tag is subtag of <choose> that renders its body if its condition evaluates to 'true'. The inputs of any elements in the body of if tag except checkbox and multi-select will be stored in a new conditional table which has a primary key of a webform as primary key.

Table 4.14 Attributes of <when> tag

Attribute	Description
Test*	The test condition that determines whether the body content should be processed.

* indicates required attribute.

<otherwise> tag

<otherwise> tag is subtag of <choose> that follows <when> tags and renders its contents only if all of the prior conditions in when tags evaluated to 'false'. The inputs of any elements in

the body of if tag except checkbox and multi-select will be stored in a new conditional table which has a primary key of a webform as primary key.

4.1.3 Report tags

<webreport> tag

The <webreport> tag defines a webreport page to show result of a webform, which contents other report tags

<resultTable> tag

A <resultTable> tag queries information from database and renders the result as HTML table.

Table 4.15 Attributes of <resultTable> tag

Attribute	Description
query*	a SQL query to get result from databases
border	a border width of a table.
Default	a default text to show when there is no result matching query statement

* indicates required attribute.

<resultText> tag

A <resultText> tag queries information from database and renders the result as single-value text.

Table 4.16 Attributes of <resultTexte> tag

Attribute	Description
query*	a SQL query to get result from databases
Default	A default text when there is no result matching query statement

* indicates required attribute.

4.2 Application for Graduation in WFL

In this section, we show the code of Application for Graduation webform in WFL. The code is in xml language, so it is self-explained and we can use attributes to describe all requirements of the webform.

```
<webform id="graduateApp" key="studentid"
       title="Application For Graduation - Georgia State University">
       <h2>Application For Graduation, Spring 2007</h2>
       <p>
              Student ID*:<textField id="studentid" size="11"
                           pattern="^\d{3}-\d{2}-\d{3}$" required="true" />
       </p>
       <p>
              Name*:<textField id="name" required="true" />
       </p>
       <p>
              Degree*:<radio id="degree" layout="vertical" required="true">
                           <optionItem value="phd" label="PhD" />
                           <optionItem value="ms" label="MS" />
                           <optionItem value="ma" label="MA" />
                       </radio>
       </p>
       <p>
              Major*:<textField id="major" required="true" />
              School*:
              <selectOne id="school" required="true">
                     <optionItem value="artScience"
                            label="College of Arts and Sciences" />
                     <optionItem value="education" label="College of Education" />
                     <optionItem value="law" label="College of Law" />
              </selectOne>
       </p>
       <p>
              Will you attend commencement ceremony* ?
              <radio id="willAttend" required="true" layout="vertical">
                     <optionItem value="yes" label="Yes" />
                     <optionItem value="no" label="No" />
              </radio>
       </p>
       <if test="willAttend=='yes'">
              <p>
                     How many tickets for commencement ceremony ?
                     <selectOne id="numTicket" type="integer" required="true">
                            <optionItem value="1" />
                            <optionItem value="2" />
                            <optionItem value="3" />
                     </selectOne>
              </p>
              <p>
                     Would you like to order :
                     <checkbox id="accessories" layout="vertical">
                            <optionItem value="cap" label="Cap and Gown" />
                            <optionItem value="frame" label="Diploma Frame" />
                     </checkbox>
              </p>
       </if>
       <p>
```

```
            Suggestion:<br />
            <textArea id="suggestion" cols="50" rows="3" />
      </p>
      <p>* Indicates Required Field</p>
      <submit value="Submit Form" />
      <reset value="Clear Form" />
</webform>
```

Figure 4.1 Application for Graduation webform in WFL

In the code of Application for Graduation, we use the following attributes to describe the webform requirements.

- **key** attribute of webform tag: the webform use studentid element as its primary key
- **required** attribute of textField, radio, and selctOne tag: all required field set its required attribute to be true
- **pattern** attribute of studentid element: student id must be in xxx-xx-xxx format.
- **test** attribute of if tag: if student will attend a commencement ceremony, the form will render extra questions.
- **type** attribute of numTicket element: the number of ticket will be stored as integer in a databae

4.3 System Implementation

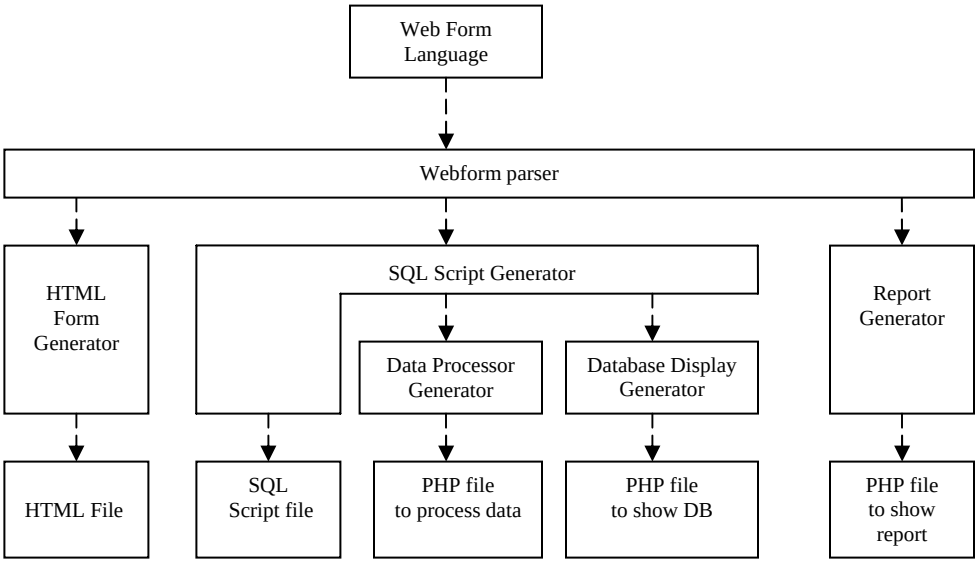


Figure 4.2 System implementation of a webform parsers

To implement the Webform Language(WFL), an xml webform parser use five modules to generate each files for a webform. Five modules include:

- HTML form generator
- SQL script generator
- PHP data processor generator
- PHP database display generator
- PHP report generator

4.3.1 HTML form generator

Html form generator uses Java XML API to parse webform described in WFL by and then generates html form to display web form to user. This module composes of two parts. The first parser will generate html form elements and JavaScript to display a webform, which

dynamically evaluates element's values that a user enters. The second parser will generate JavaScript to validate user input before the form will be submitted to a server.

4.3.2 SQL script generator

SQL Script generator will parse a form described in WFL to create table schema need to store data when a user submits the form. It will generate database schema and write SQL script. Data type and constraint of database schema must match webform elements and their constraint that a user specifies in WFL. The table below compares data type and constraints between WFL and corresponding values in MySQL.

	WFL	**MySQL**
Data types	string	VARCHAR(255)
	integer	INT
	decimal	DOUBLE
Constraints	required	NOT NULL
	key	PRIMARY KEY

Table 4.17 Comparisons of data types and constraints between WFL and MySQL

4.3.3 Data processor generator

This module will generate script PHP file to receive user inputs from HTTP request, save them to a database and then forward user to the next page. This script file will use database schema from SQL script generator and checks whether all primary keys of tables are sent in the HTTP request or not. If all primary keys for any table are sent in HTTP request, the script will save inputs to that table.

Moreover, Data Process Generator will also generate a sample webform properties file which allows to change database and webform parameters such as database host, user id,

password, database name, and the location of forwarded page later. In the properties file, parameters will be defined by using the following keywords.

Table 4.18 Special keywords in webform properties file.

Keyword	Description
dbUsername	username of database account
dbPassword	Password of database account
dbHostname	Location of database server
dbname	Database name
forwordURL	A url of the forwarded page after a form is processed.

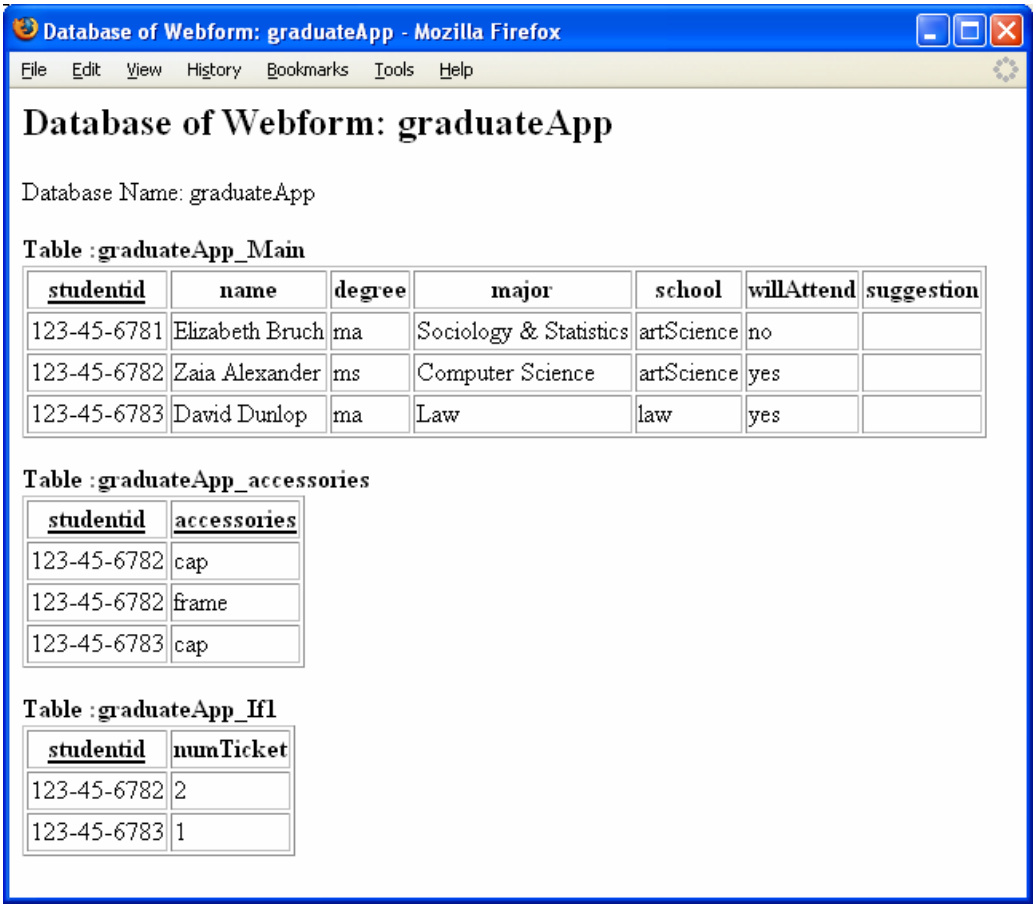
Figure 4.3 below shows the graduateAppProperties.php, the properties file of Application for Graduation webform.

```
<?php

        $dbUsername = "username";
        $dbPassword = "password";
        $dbHostname = "tinman.cs.gsu.edu";
        $dbname = "graduateApp";
        $forwardURL = "nextpage.html";

?>
```

Figure 4.3 A properties file of Application for graduation

4.3.4 Database display generator

This generator will generate PHP script file to display the whole database, which correspond with schema generated from SQL script generator. All primary key of tables will be underlined. Figure4.4 shows a database of Application for graduation, which contains three tables.



Database of Webform: graduateApp - Mozilla Firefox

File Edit View History Bookmarks Tools Help

Database of Webform: graduateApp

Database Name: graduateApp

Table :graduateApp_Main

studentid	name	degree	major	school	willAttend	suggestion
123-45-6781	Elizabeth Bruch	ma	Sociology & Statistics	artScience	no	
123-45-6782	Zaia Alexander	ms	Computer Science	artScience	yes	
123-45-6783	David Dunlop	ma	Law	law	yes	

Table :graduateApp_accessories

studentid	accessories
123-45-6782	cap
123-45-6782	frame
123-45-6783	cap

Table :graduateApp_If1

studentid	numTicket
123-45-6782	2
123-45-6783	1

Figure 4.4 A webpage to show a database of Application for Graduation

4.3.5 Result display generator

This generator will read webreport that described in WFL and then generates PHP script file to display result of a webreport. Users can specific the result they want by writing their own query to get result from database and choose to display it as single-value text or table. All database information will be read from the properties file that was generated by Data Processor Generator. Below is webreport and its code that described in WFL to find a list of student in College of Law who will attend commencement ceremony for Application for Graduation webform.

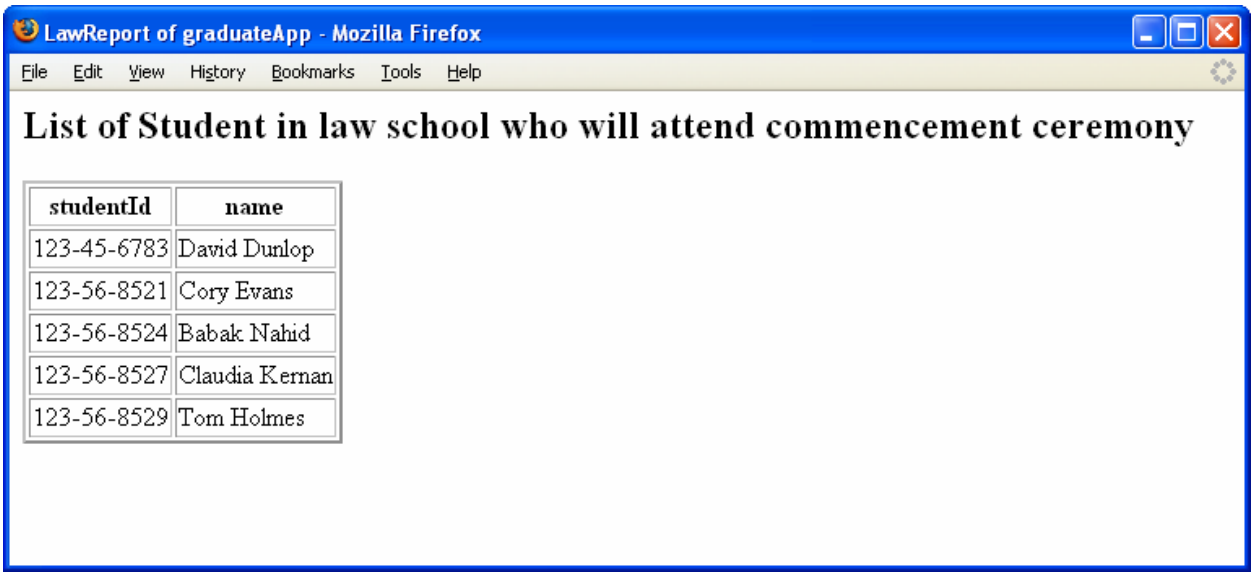


Figure 4.5 A webreport of Application for Graduation

```
<webreport id="Report1" webform="graduateApp">
      <h2>List of Student in law school who will attend commencement ceremony</h2>
      <resultTable query="select studentId, name from graduateApp_main
                   where school='Law'"/>
</webreport>
```

Figure 4.6 A webreport for Application for Graduation in WFL.

5. SIMPLE WEBFORM LANGUAGE

Web Form Language is easy and self-describing; however, it might be too long or awkward to describe a very simple webform by WFL. In this section, we propose another language called a Simple Webfrom Language(SWFL) with an objective to make webform describing easy. SWFL is a short language to describe web from and webreport by simulating create table statement of SQL. When the SWFL is processed, it will be translated to WFL first and then webform parser will generate all necessary files for webform. Below is syntax in creating webform and webreport in SWFL.

5.1 Create Webform Syntax

CREATE WEBFORM *webform_id*
(*definition , …)*

definition: *text_definition* | *element_definition* | *constraint_definition*

element_definition:
textfield_id TEXTFIELD [(*size*)] [*textinput_option* , …]
| *textarea_id* TEXTAREA [(rows [, *cols*]) [*textinput_option* , …]
| *password_id* PASSWORD [(*size*)] [*textinput_option* , …]
| *selectOne_id* SELECTONE (*optionItem* , …) [*single_choice_option* , …]
| *selectMany_id* SELECTONE (*optionItem* , …) [*multiple_choice_option* , …]
| *radio_id* RADIO (*optionItem* , …) [*single_choice_option* , …]
| *checkbox_id* CHECKBOX (*optionItem* , …) [*multiple_ choice _option* , …]
| SUBMIT [(*'button_label'*)]
| RESET [(*'button_label'*)]

text_definition : TEXT (‘*static_text*’) | HEADING SIZE (‘*heading_text*’) | TITLE (‘*form_title*’)

constraint_definition :

HIDE (*element_id* , …) CONDITION (*boolean_expression*)

| SUBMIT CONDITION (*boolean_expression*)

| KEY (*element_id* , …)

textinput_option:

NOT NULL | KEY | PATTERN(‘*regular_expression*’)

| TYPE (*data_type*) | LABEL (‘*text_label*’)

single_choice_option:

TYPE (*data_type*) | LABEL(‘*text_label*’) | NOT NULL

multiple_choice_option:

TYPE (*data_type*) | LABEL (‘*text_label*’)

| MIN (*min_num_select*) | MAX (*max_num_select*)

data_type:

integer | *decimal* | *string*

Figure 5.1 Create webform Syntax

In SWFL, webform can be defined by “create webform” statement, form ID, optional form title and a list of form definitions. Definitions consist of element definitions, static text definition, and constraint definitions.

5.1.1 Form element definition

TextFiled: textfield can be defined by its ID name, “textField” keywords and an optional size. We can specify textfield options by the following keywords

“not null” user input is required, textfield cannot be empty

“key”	textfield is primary key of a form
“pattern”	user input must be in the specified pattern
“type”	data type of the textfield
“label”	label of the textfield

TextArea: textarea can be defined by its ID name, “textArea” keyword and optional row size and column size. However, when column size are specified, row size must be specified. Textarea has the same options as Textfield. We can specify by its option using “not null”, “key”, “pattern”, “type”, and “label” keywords.

Password: password can be defined by its ID name, “password” keyword and an optional size. Its has the same options as Textfield. We can specify by password option using “not null”, “key”, “pattern”, “type”, and “label” keywords.

SelectOne: selectOne is normal html select list, but it allow user to select only one option. It can be defined by it ID name, “selectOne” keyword, and followed by a list of string options. We can specify option for selectOne by the following keywords:

“not null”	one of options must be selected
“type”	data type of the selectOne element
“label”	label of the selectOne element

SelectMany: selectMany is normal html select list which allow user to select more than one options. We defines selectMany by its ID name, “selectMany” keyword, and a list of string options. Below is the option of selectMany element

“type”	data type of the selectOne element
“label”	label of the selectOne element
“minSelected”	the minimum number of options that must be selected

“maxSelected” the maximum number of options that can be selected

Radio: radio is a group of html radio buttons. It can be define by its ID name, “radio” keyword, and followed by a list of string options. Radio element has the same options as SelectOne element which includes “required”, “type”, and “label”.

Checkbox: checkbox is a group of html checkboxes. It can be define by its ID name, “checkbox” keyword, and followed by a list of string options. Checkbox element has the same options as MultiSelect which are “type”, “label”, “minSelected”, and “maxSelected”.

Submit and **Reset** button: they can be defined by “submit” or “reset” keywords and option button label.

5.1.2 Static text definitions

Text: text element will be render as normal text in html form.

Heading: heading element will be rendered as <h> tag in html form. The size form 1 to 6 must be specified. The smaller number, the bigger heading size is.

5.1.3 Constraint definition

Hide Condition: Hide condition allows user to hide webform elements when the specified condition is true. We can create dynamic webform by using the hide condition. Hide condition can be define by “hide” keyword , a list of element that will be hidden, a “condition” keyword, and then a boolean expression. In an example below, we will hide marital status question when age user enter is less than 20.

HIDE(maritalStatus) CONDITION (age < 20)

Submit Condition: submit condition specify the condition that must be true before the form will be submitted. We define submit condition by submit condition keyword and boolean expression.

Key : we can specify a webform keys by using a “key” keyword and follow by a list of primary key element.

5.2 Create Webreport Syntax

Webreport can be described by the following syntax. It is similar to syntax in defining webform which starts by “create webreport” statement, form result ID, optional report title, and then follow by a list of report elements. The report element can be Result Table, Result Text, text, and Heading.

CREATE WEBREPORT *webresult_id* OF WEBFORM *webformt_id*

(*result_element_definition , …)*

result_element_definition:

RESULTTABLE (*SQL_select_statement*)

| RESULTTEXT (*SQL_select_statement*)

| *text_definition*

Figure 5.2 Create webreport syntax

Result Table: Result table is the result of executing the specified SQL query against the database that stored webform data. It will be rendered as html table.

Result Text: Result Text is a single value of executing the specified SQL query against the database that stored webform data. It will be rendered as static text.

5.3 A Simple Webform Parser

This module translate high-level web from language to xml Dom tree, which can write to xml file or parse to other parsers. We use Jflex [6], a lexical analyzer generator developed by Elliot Berk at Princeton University, and JCup LALR parser [7] generator from Georgia tech to generate web form language. By using terminal token and lexical grammar as shown in Appendix 9.2, we can translate SWFL to xml DOM tree.

5.4 Application for Graduation in SWFL

Below is code of Application for Graduation which is described in SWFL. It looks simple and shorter than the one described in webform language.

```
create webform graduateApp
(
       title        ("Application for Graduation Georgia State University"),
       heading2     ("Application for Graduation, Spring 2007"),
       studentid    textField(11) pattern("^\d{3}-\d{2}-\d{3}$") key
                    label("Student ID*")
                    not null,
       name         textField not null label("Name*"),
       degree       selectOne ("PhD","MS","MA") label("Degree*") not null,
       major        textField not null label("Major*"),
       school       selectOne("ArtScience","Education","Law") not null
                    label("School"),
       willAttend   radio("Yes", "No") not null
                    label("Will you attend commencement ceremony* ?"),
       numTicket    selectOne("1","2","3") not null type(integer)
                    label("How many tickets for commencement ceremony ?"),
       accessories  checkbox("Cap and Gown", "Diploma Frame")
                    label("Would you like to order :"),
       submit("Submit Form"),
       reset("Clear Form"),
       hide(numTicket, accessories) condition(willAttend="Yes"),
       key(studentid),
)
```

Figure 5.3 Application for Graduation in SWFL

6. WEBFORM EXAMPLES

With WFL, we can simply create any webform and webreport in minutes. This chapter gives two examples weform described by WFL: Country Survey Form and Conference 2006 Survey Form. In the Country Survey Form shows how to use multi-conditional tag in web form language, while Conference 2006 Survey Form uses many simple tags to solve complex condition. In the end of this section, we shows example of webreport to query data from a Conference 2006 Survey Form

6.1 Country Survey Form

Country Survey Form is a simple survey form that asks users about their name, country, and state that they came from. For a country's name, a user can select USA, Canada, India, China, or Others. If a user selects USA, this form will shows a select list of states. On the other hand, if you select others country that is not USA, a form will show text filed for the state instead. In this form, all fields are required.

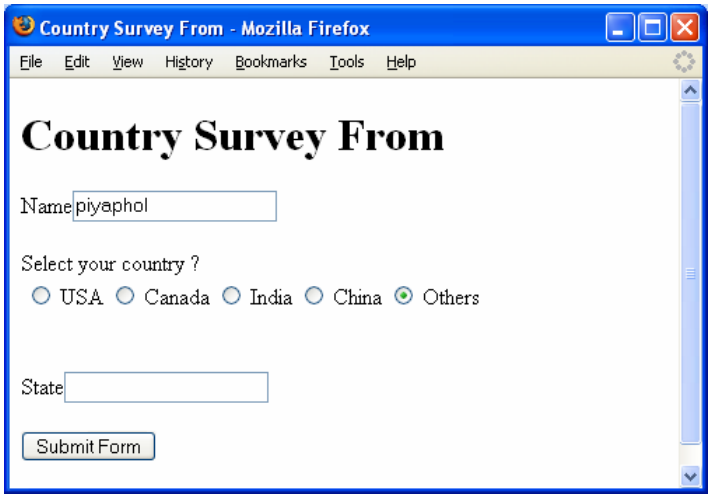


Figure 6.1 Country Survey webform when a user selects other countrys

Figure 6.2 Country Survey webform Figure when a user selects USA

Below is code to describe this form in simple web form language

```
create webform sample1
(
       title         ("Country Survey From"),
       heading1      ("Country Survey From"),
       name          textField not null label("Name"),
       country       radio("USA","Canada","India", "China", "Others") not null
                     label("Select your country ?"),
       stateSelect   selectOne("GA","FL", "CA") label("State"),
       stateText     textField label("State"),
       submit("Submit Form"),
       hide(stateSelect)          condition(country<>"USA"),
       hide(stateText)                   condition(country="USA"),
       hide(stateText, stateSelect)      condition(country=undefined)
)
```

Figure 6.3 Country Survey webform in SWFL

Figure 6.4 shows a sample webreport described in SWFL to find a list of students who are from China

```
create webreport report1 of webform countryForm
(
       title         ("Report of Country Survey Form"),
       heading2      ("List of Student from China"),
       resultTable   ("select name from countryFormMainView where country='China'")
)
```

Figure 6.4 A webreport described in SWFL to find students from China

6.2 Conference Survey Form

This survey webform will collect from attendees of the conference. A user can select to attend a banquet and to visit Aquarium. Below is summary complex requirement of this form

1. Name, E-mail, Type of Registration fields are required. The banquet Aquarium attendance questions are optional.
2. If attendee selects to attend a banquet, the form will dynamically ask a user the meal type question. Moreover, if attendee is full-time registration , he/she can buy up to three extra tickets and extra meal type questions should be populated for each ticket.
3. If a user selects to attend a banquet or buy any extra tickets, all meal for each ticket must be selected.
4. The form should verify that email address to contain '@' character

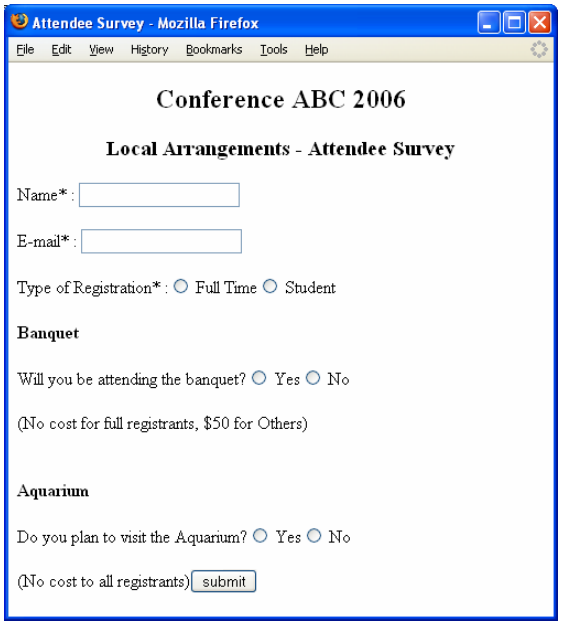


Figure 6.5 Conference 2006 Survey webform

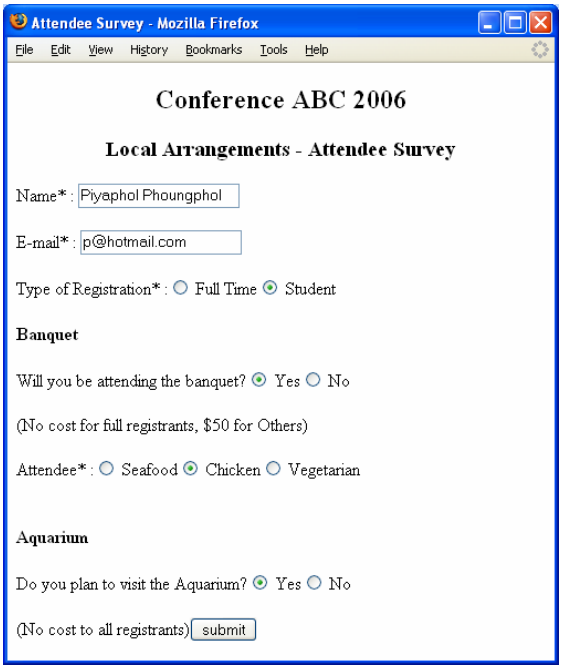


Figure 6.6 Conference webform: a student and selects to attend a banquet

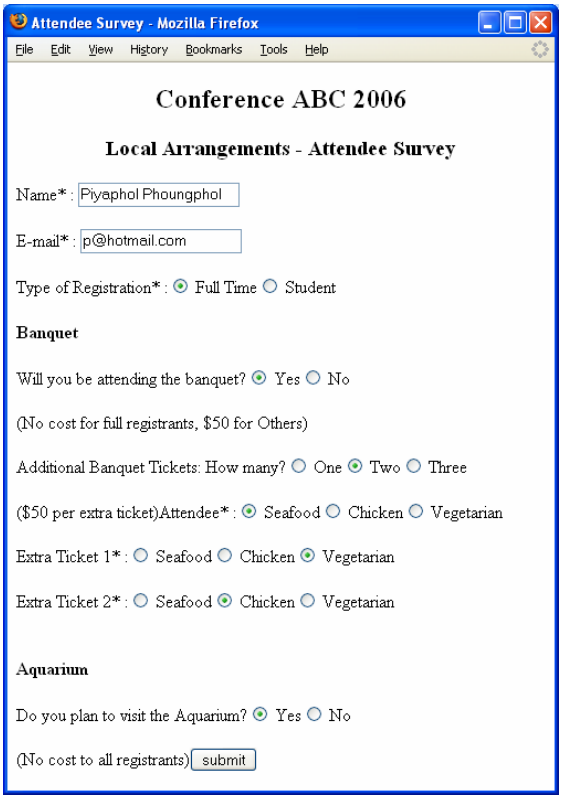


Figure 6.7 Conference webform: a full-time registration attendee buys two extra tickets

Because it is difficult to describe this complex form using a simple condition of SWFL, we will describe this form by WFL. Code below describes the Conference survey form in WFL.

```
<webform id="Conference" key="email" title="Attendee Survey">
	<center>
		<h2>Conference ABC 2006</h2>
		<h3>Local Arrangements - Attendee Survey</h3>
	</center>

	Name* : <textField id="name" required="true"/>
	<br/>

	E-mail* : <textField id="email" required="true"
				pattern="^[a-zA-Z0-9]+@[a-zA-Z0-9.]+$"/>
	<br/>

	Type of Registration* :
	<radio id="registrationType" required="true">
		<optionItem value="full" label="Full Time"/>
		<optionItem value="student" label="Student"/>
	</radio>

	<h4>Banquet</h4>
	Will you be attending the banquet?
	<radio id="willAttendBanquet">
		<optionItem value="y" label="Yes"/>
		<optionItem value="n" label="No"/>
	</radio>
	<br/>(No cost for full registrants, $50 for Others)
	<br/>

	<if test="(registrationType=='full') &amp;&amp; (willAttendBanquet=='y')">
		Additional Banquet Tickets: How many?
		<radio id="extraMeal" type="int">
			<optionItem value="1" label="One"/>
			<optionItem value="2" label="Two"/>
			<optionItem value="3" label="Three"/>
		</radio>
		<br/>($50 per extra ticket)
	</if>

	<if test="willAttendBanquet=='y'">
		Attendee* :
		<radio id="meal0" required="true">
			<optionItem value="SF" label="Seafood"/>
			<optionItem value="CK" label="Chicken"/>
			<optionItem value="VEG" label="Vegetarian"/>
		</radio>
		<br/>
	</if>

	<if test="extraMeal>=1" >
		Extra Ticket 1* :
		<radio id="meal1" required="true">
			<optionItem value="SF" label="Seafood"/>
			<optionItem value="CK" label="Chicken"/>
			<optionItem value="VEG" label="Vegetarian"/>
		</radio>
		<br/>
	</if>
```

```
        <if test="extraMeal>=2" >
                Extra Ticket 2* :
                <radio id="meal2" required="true">
                        <optionItem value="SF" label="Seafood"/>
                        <optionItem value="CK" label="Chicken"/>
                        <optionItem value="VEG" label="Vegetarian"/>
                </radio>
                <br/>
        </if>

        <if test="extraMeal>=3" >
                Extra Ticket 3* :
                <radio id="meal3" required="true">
                        <optionItem value="SF" label="Seafood"/>
                        <optionItem value="CK" label="Chicken"/>
                        <optionItem value="VEG" label="Vegetarian"/>
                </radio>
                <br/>
        </if>

        <h4>Aquarium</h4>
        Do you plan to visit the Aquarium?
        <radio id="willAttendAquarium">
                <optionItem value="y" label="Yes"/>
                <optionItem value="n" label="No"/>
        </radio>
        <br/>(No cost to all registrants)

        <submit value="submit"/>
</webform>
```

Figure 6.8 Conference 2006 Survey webform in WFL

Beside from a webform, we can also create a report to show the survey statistic by using WFL. The example code in figure 6.10 shows the number of people attend the survey, total number of meal of each type, and the number of people who plan to visit aquarium.

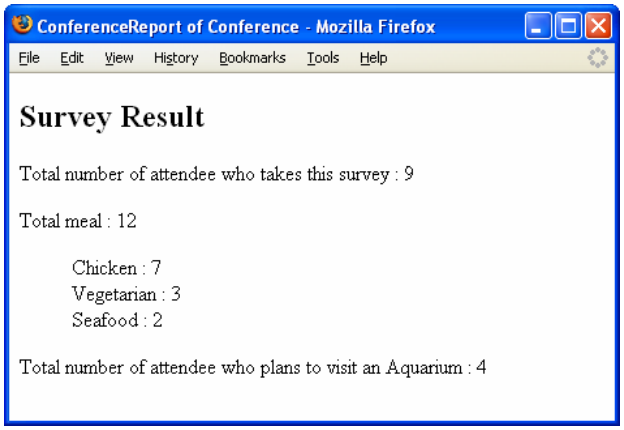


Figure 6.9 Statistic page of Confenrece 2006 Survey

```
<webreport id="report2" webform="Conference" title="Conference Report">
  <h2>Survey Result</h2>
    <p>
      Total number of attendee who takes this survey :
      <resultText query="select count(*) from Conference_Mainview"/>
    </p>
    <p>
      Total meal:<resultText query="select count(meal0)+count(meal1)+ count(meal2)+
                 count(meal3) from Conference_Mainview"/><br/>
      <blockquote>
        ck:<resultText query="select sum(if(meal0='CK',1,0) + if(meal1='CK',1,0) +
            if(meal2='CK',1,0) +  if(meal3='CK',1,0)) from Conference_Mainview"/><br/>
        Veg:<resultText query="select sum(if(meal0='VEG',1,0) + if(meal1='VEG',1,0) +
            if(meal2='VEG',1,0) + if(meal3='VEG',1,0)) from
            Conference_Mainview"/><br/>
        Seafood:<resultText query="select sum(if(meal0='SF',1,0) + if(meal1='SF',1,0)+
             if(meal2='SF',1,0) +  if(meal3='SF',1,0)) from
            Conference_Mainview"/><br/>
      </blockquote>
    </p>
    <p>
      Total number of attendee who plans to visit an Aquarium :
      <resultText query="select count(*) from Conference_Mainview where
                  willAttendAquarium= 'y'"/><br/>
    </p>
</webreport>
```

Figure 6.10 A statistic report for Conference 2006 Survey in WFL

7. CONCLUSION AND FUTURE WORK

In this thesis, we present a new technique in creating webform and webreport by using a new language called Webform Language(WFL). WFL is self-described, easy to understand because it is based on xml language, which is standard language in describing information and most programming language has supported API in parsing xml. Comparing with conventional ways, this technique can considerably reduce a complexity, and also decrease developing time used in creating webform. In addition, we also propose another language, a Simple Webform Language(SWFL), which is based on WFL and CREATE TABLE statement of SQL. SWFL lists all elements and constraints of a webform within a CREATE statement.

Despite we success in describing webform and webreport in WFL to reduce developing time, there are still many unresolved issues to improve and increase benefits of webform language in future.

1. Graphic User Interface: A graphic program to help user in design a webform/ webreport, and then the program automatically generate their description and requirements in WFL for a user.
2. Limitation of SWFL: There is some limitation in describing a complex webform by using SWFL. As shown in Chapter 7, the requirements of Conference Survey Webform Example is too complex and it is difficult to describe this form by a SWFL. The syntax for SWFL can be improved so that it can describe every webform as WFL while keeping the same level of simplicity.

REFERENCES

[1] Wikipedia, “Form (web),” 2007, http://en.wikipedia.org/wiki/Form_(web)

[2] W3C-HTML, “HyperText Markup Language(HTML4.01),” 1999, http://www.w3.org/TR/html4/

[3] PHP Documentation Group, “PHP Manual”, 2007, http://us3.php.net/manual/en/index.php

[4] MySQL AB, “MySQL 5.0 Reference Manual,” 2007, http://dev.mysql.com/doc/refman/5.0/en/

[5] W3C-XML, “Extensible Markup Language(XML),” 2006, http://www.w3.org/XML/

[6] G. Klein, “JFlex - The Fast Scanner Generator for Java,” 2004, http://www.jflex.de/

[7] S. Hudson, “Cup LALR Parser Generator in Java,” 1999, http://www2.cs.tum.edu/projects/cup/

[8] Apple Computer, Inc., Mozilla Foundation, and Opera Software ASA, “Web Forms 2.0,” 2006, http://www.whatwg.org/specs/web-forms/current-work/

[9] Apache Software Foundation, “JSF HTML tag library.,” 2006, http://myfaces.apache.org/impl/tlddoc/h/tld-summary.html

[10] JasperSoft Corporation, “JasperReports: one of the world’s most popular reporting systems,” 2007, http://www.jaspersoft.com/JasperSoft_JasperReports.html

APPENDICES

A. WEBFORM PARSER SOURCE CODE

A.1 Package WebformParser

class Parser.java

```
package webformParser;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.OutputStreamWriter;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.transform.OutputKeys;
import javax.xml.transform.Result;
import javax.xml.transform.Source;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import org.w3c.dom.Document;
import org.w3c.dom.Element;

import webformParser.html.HTMLFormGenerator;
import webformParser.php.processForm.PHPProcessFormGenerator;
import webformParser.php.report.PhpShowDBGenerator;
import webformParser.php.report.PhpShowReportGenerator;
import webformParser.simpleParser.Lexer;
import webformParser.simpleParser.simpleWebformParser;
import webformParser.sql.SQLScriptGenrator;

/**
 * Webform parser class
 *
 * @author Piyaphol Phoungphol
 *
 */
public class Parser {
        /**
         * main parser
         *
         * @param argvs
         */
        public static void main(String[] argvs) {

                if (argvs.length != 2) {
                        String fileName = argvs[0];
                        File xmlFile = new File(fileName);
                        String extension = fileName
                                        .substring(fileName.lastIndexOf(".") + 1);
```

```
                if (extension.equals("wfl")) {
                        parseWebformLanguage(xmlFile);
                } else if (extension.equals("swf")){
                        parseSimpleWebFormLanguage(xmlFile);
                } else {
                        System.out.println("Invalid file extension.");
                }
        } else {
                System.out.println("Invalid Argrument");
                System.out.println("java webformParser.Parser filename");
        }

}

/*
 * general method to write filename
 */
private static void writeFile(String text, String fileName)
                throws IOException {

        // Create file
        FileWriter fstream = new FileWriter(fileName);
        BufferedWriter out = new BufferedWriter(fstream);
        out.write(text);
        // Close the output stream
        out.close();
}

/**
 * parse webform alnguage
 *
 * @param xmlFile
 */
public static void parseWebformLanguage(File xmlFile) {

        try {
                DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
                DocumentBuilder db = dbf.newDocumentBuilder();
                Document xmlDoc = db.parse(xmlFile);
                Element root = xmlDoc.getDocumentElement();

                if (root.getTagName().equals("webform")) {
                        generatedWebform(xmlDoc);
                } else if (root.getTagName().equals("webreport")) {
                        gernerateWebreport(xmlDoc);
                } else {
                        System.out
                                        .print("Document is not either webform or
webreport.");
                }

        } catch (Exception e) {
                System.out.print("Problem parsing the file.");
                e.printStackTrace();
        }
}

/**
 * method to generate all nesscessary files for a webform
 *
 * @param xmlFile
 */
public static void generatedWebform(Document xmlDoc) throws Exception {

        // getwebform id
        Element root = xmlDoc.getDocumentElement();
        String id = root.getAttribute("id");

        // generate html files
        String htmlText = HTMLFormGenerator.getHTML(xmlDoc);
        writeFile(htmlText, id + ".html");
```

```
        System.out.println("Write html file: " + id + ".html");

        // generate sql script
        SQLScriptGenrator sqlGenrator = new SQLScriptGenrator();
        String sqlText = sqlGenrator.getSQL(xmlDoc);
        writeFile(sqlText, id + "Script.sql");
        System.out.println("Write SQL script file: " + id + "Script.sql");

        // generate phpscript to process form
        String phpText = PHPProcessFormGenerator.getPHP(xmlDoc);
        writeFile(phpText, id + "ProcessForm.php");
        System.out.println("Write php script to process form: " + id
                        + "ProcessForm.php");

        // generate properties file
        String propertiesText = PHPProcessFormGenerator.getProperties(id);
        writeFile(propertiesText, id + "Properties.php");
        System.out.println("Write webform properties file: " + id
                        + "Properties.php");

        // generate phpscript to show db
        String phpDBText = PhpShowDBGenerator.getPHP(xmlDoc);
        writeFile(phpDBText, id + "DB.php");
        System.out.println("Write php script to show DB: " + id
                        + "Properties.php");

}

/**
 * method to generate webreport
 *
 * @param xmlFile
 */
public static void gernerateWebreport(Document xmlDoc) throws Exception {

        // getwebreport id
        Element root = xmlDoc.getDocumentElement();
        String webReportId = root.getAttribute("id");

        String phpReportText = PhpShowReportGenerator.getPHP(xmlDoc);
        String fileName = webReportId + "Report.php";
        writeFile(phpReportText, fileName);
        System.out.println("Write webreport file: " + fileName);

}

/**
 * method to parse simple webform language
 *
 * @param xmlFile
 */
public static void parseSimpleWebFormLanguage(File xmlFile) {
        try {
                simpleWebformParser p = new simpleWebformParser(new Lexer(
                                new FileReader(xmlFile)));
                Document xmlDoc = (Document) p.parse().value;
                Element root = xmlDoc.getDocumentElement();
                String id = root.getAttribute("id");
                String fileName = id + ".wfl";

                printXML(xmlDoc, fileName);
                System.out.println("Write webform language file: " + fileName);

                parseWebformLanguage(new File(fileName));

        } catch (Exception e) {
                e.printStackTrace();
        }
}

/**
```

```
	 * method to write xml doc to file
	 *
	 * @param doc
	 * @param fileName
	 */
	private static void printXML(Document doc, String fileName) {
		try {
			// Prepare the DOM document for writing
			Source source = new DOMSource(doc);

			FileOutputStream out = new FileOutputStream(fileName);

			Result result = new StreamResult(new OutputStreamWriter(out,
				"utf-8"));

			// Write the DOM document to the file
			TransformerFactory factory = TransformerFactory.newInstance();
			factory.setAttribute("indent-number", new Integer(4));

			Transformer xformer = factory.newTransformer();
			xformer.setOutputProperty(OutputKeys.INDENT, "yes");
			xformer.setOutputProperty(OutputKeys.OMIT_XML_DECLARATION, "yes");

			xformer.transform(source, result);

		} catch (Exception e) {
			e.printStackTrace();
		}
	}
}
```

A.2 Package WebformParser.html

Class HTMLFormGenerator

```
package webformParser.html;

import org.w3c.dom.Document;
import org.w3c.dom.Element;

/**
 *
 * @author Piyaphol Phoungphol class to generate HTML form with javascript
 */
public class HTMLFormGenerator {
	/**
	 * method to generate html String from xml
	 *
	 * @param xmlForm
	 * @return
	 */
	public static String getHTML(Document xmlForm) throws Exception {
		String htmlText = "";

		Element root = xmlForm.getDocumentElement();

		if (root.getTagName().equals("webform")) {

			String id = root.getAttribute("id");

			String title = root.getAttribute("title");
			if (title == null) {
				title = "";
			}

			FormWriter formWriter = new FormWriter();
			FormValidator formValidator = new FormValidator();

			htmlText += "<html>\n" + "<head>\n" + "<title>" + title
				+ "</title>\n" + "<script type=\"text/javascript\">\n";

			// generate writeform function
			htmlText += formWriter.parseWebForm(root);

			htmlText += "\n\n\n";

			// generate validation form function
			htmlText += formValidator.parseWebForm(root);

			htmlText += "</script>\n"
				+ "</head>\n"
				+ "<body onload=\"writeForm('')\">\n"
				+ "<form action=\""
				+ id
				+ "ProcessForm.php"
				+ "\" onsubmit=\"return validateForm()\" name=\"myform\" " +
				+ "method=\"post\">\n"
				+ "<div id=\"root\" />\n" + "</form>\n" + "<body>\n"
				+ "</html>";
		}

		return htmlText;
	}
}
```

Class FormWriter.java

```
package webformParser.html;

import org.w3c.dom.Attr;
import org.w3c.dom.Element;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

/**
 *
 * @author Piyaphol Phoungphol
 * class to generate dynamic html form
 *
 */
class FormWriter {

	// all constants
	private final String SELECT = "select";

	private final String RADIO = "radio";

	private final String CHECKBOX = "checkbox";

	// parsing parameter
	private String _parameterName = null;

	private boolean _isMultiSelect = false;

	private String _parameterType = null;

	/**
	 * parse webform tag
	 *
	 * @param e
	 * @return
	 * @throws Exception
	 */
	public String parseWebForm(Element e) throws Exception {

		String htmlText = "function writeForm(sourceElementName){\n\n"
				+ "var root = document.getElementById('root');\n"
				+ "var htmlText = '';\n\n";

		NodeList children = e.getChildNodes();
		for (int i = 0; i < children.getLength(); i++) {
			Node child = children.item(i);

			if (child.getNodeType() == Node.TEXT_NODE) {

				// remove all tab, new line, and space
				String text = child.getNodeValue().replaceAll(
						"(\\n|\\s{2}|\\t)", "");

				if (!text.equals("")) {
					htmlText += "htmlText += '" + text + "';\n";
				}

			} else if (child.getNodeType() == Node.ELEMENT_NODE) {

				htmlText += parseTag((Element) child);
			}
		}

		htmlText += "\n//alert(htmlText);\n"
				+ "root.innerHTML = htmlText;\n"
				+ "// set focus back if source event is textbox\n"
				+ "try{\n"
				+ "// IE function: createTextRange\n"
```

```
                                + "var sourceTextBoxRange =
document.getElementById(sourceElementName).createTextRange();\n"
                                + "sourceTextBoxRange.collapse(false);\n"
                                + "sourceTextBoxRange.select();\n" + "}catch(err){\n" + "try{"
                                + "// firefox\n"
                                + "document.getElementById(sourceElementName).focus();\n"
                                + "}catch(err){}\n" + "}\n" + "}\n";

                return htmlText;
        }

        /**
         * parse xml tags
         *
         * @param e
         * @return
         * @throws Exception
         */
        private String parseTag(Element e) throws Exception {

                String tagName = e.getTagName();
                String htmlText = "";

                if (tagName.equals("textField")) {
                        htmlText = parseTextField(e);
                } else if (tagName.equals("textArea")) {
                        htmlText = parseTextArea(e);
                } else if (tagName.equals("selectOne")) {
                        htmlText = parseSelectOne(e);
                } else if (tagName.equals("selectMany")) {
                        htmlText = parseSelectMany(e);
                } else if (tagName.equals("radio")) {
                        htmlText = parseRadio(e);
                } else if (tagName.equals("checkbox")) {
                        htmlText = parseCheckbox(e);
                } else if (tagName.equals("password")) {
                        htmlText = parsePassword(e);
                } else if (tagName.equals("optionItem")) {
                        htmlText = parseOptionItem(e);
                } else if (tagName.equals("submit")) {
                        htmlText = parseSubmit(e);
                } else if (tagName.equals("reset")) {
                        htmlText = parseReset(e);
                } else if (tagName.equals("if")) {
                        htmlText = parseIf(e);
                } else if (tagName.equals("choose")) {
                        htmlText = parseChoose(e);
                } else if (tagName.equals("outputText")) {
                        htmlText = parseOutputText(e);
                } else {
                        htmlText = parseHtmlTag(e);
                }

                return htmlText;
        }

        /**
         * parse normal html tags
         *
         * @param e
         * @return
         * @throws Exception
         */
        private String parseHtmlTag(Element e) throws Exception {
                String htmlText = "";
                String tagName = e.getTagName();

                htmlText += "htmlText += '<" + tagName;

                // Get all the attributes
                NamedNodeMap attrs = e.getAttributes();
```

```
        for (int i = 0; i < attrs.getLength(); i++) {
                Attr attr = (Attr) attrs.item(i);

                String attrName = attr.getNodeName();
                String attrValue = attr.getNodeValue();

                htmlText += " " + attrName + "=\"" + attrValue + "\"";
        }
        htmlText += ">';\n";

        // parse all children
        NodeList children = e.getChildNodes();
        for (int i = 0; i < children.getLength(); i++) {
                Node child = children.item(i);

                if (child.getNodeType() == Node.TEXT_NODE) {

                        // remove space, tab, new line
                        String text = child.getNodeValue().replaceAll(
                                        "(\\n|\\s{2}|\\t)", "");
                        if (!text.equals("")) {
                                htmlText += "htmlText += '" + text + "';\n";
                        }

                } else if (child.getNodeType() == Node.ELEMENT_NODE) {
                        htmlText += parseTag((Element) child);
                }
        }

        htmlText += "htmlText += '</" + tagName + ">';\n";

        return htmlText;
}

/*
 * parse radio tag
 */
private String parseRadio(Element e) throws Exception {
        String htmlText = "";

        // read attributes
        String id = e.getAttribute("id");
        String radioName = id + "Radio";

        _parameterName = id;
        _parameterType = RADIO;

        // read value
        htmlText += "\n";
        htmlText += "//read " + id + " value\n";
        htmlText += "var " + id + ";\n" + "var " + radioName
                        + " = document.myform." + id + ";\n" + "try{\n" + "for (i=0;i<"
                        + radioName + ".length;i++){\n" + "if (" + radioName
                        + "[i].checked){\n" + id + " = " + radioName + "[i].value;\n"
                        + "break;\n" + "} else {\n" + id + " = null;\n" + "}\n" + "}\n"
                        + "}catch(err){\n" + id + " = null;\n" + "}\n";

        // write html

        NodeList children = e.getChildNodes();
        for (int i = 0; i < children.getLength(); i++) {
                Node child = children.item(i);
                if (child.getNodeType() == Node.TEXT_NODE) {

                        // remove tab, space, newline
                        String text = child.getNodeValue().replaceAll(
                                        "(\\n|\\s{2}|\\t)", "");
                        if (!text.equals("")) {
                                htmlText += "htmlText += '" + text + "';\n";
                        }
                } else if (child.getNodeType() == Node.ELEMENT_NODE) {
```

```
                        htmlText += parseTag((Element) child);
                }
        }

        _parameterName = null;
        _parameterType = null;

        return htmlText;
}

/*
 * parse text field tag
 */
private String parseTextField(Element e) throws Exception {

        String htmlText = "";

        // read attributes
        String id = e.getAttribute("id");
        String textFieldName = id + "TextField";
        Attr sizeAttr = e.getAttributeNode("size");
        String size = (sizeAttr == null) ? "" : " size=\""
                        + sizeAttr.getValue() + "\"";
        Attr maxlengthAttr = e.getAttributeNode("maxlength");
        String maxlength = (maxlengthAttr == null) ? "" : " maxlength=\""
                        + maxlengthAttr.getValue() + "\"";

        // read value
        htmlText += "\n";
        htmlText += "//read " + id + " value\n";
        htmlText += "var " + id + ";\n" + "var " + textFieldName
                        + " = document.getElementById('" + id + "');\n" + "try{\n" + id
                        + " = " + textFieldName + ".value;\n" + "}catch(err){\n" + id
                        + " = '';\n" + "}\n";

        // write html
        htmlText += "htmlText += '<input type=\"text\" name=\"" + id
                        + "\" id=\"" + id + "\"" + size + maxlength + " value =\"'+"
                        + id + "+'\"" + " onKeyUp=\"writeForm(\\'" + id
                        + "\\')\"/>';\n";

        return htmlText;
}

/*
 * parse password tag
 */
private String parsePassword(Element e) throws Exception {

        String htmlText = "";

        // read attributes
        String id = e.getAttribute("id");
        String passwordName = id + "Password";
        Attr sizeAttr = e.getAttributeNode("size");
        String size = (sizeAttr == null) ? "" : " size=\""
                        + sizeAttr.getValue() + "\"";
        Attr maxlengthAttr = e.getAttributeNode("maxlength");
        String maxlength = (maxlengthAttr == null) ? "" : " maxlength=\""
                        + maxlengthAttr.getValue() + "\"";

        // read value
        htmlText += "\n";
        htmlText += "//read " + id + " value\n";
        htmlText += "var " + id + ";\n" + "var " + passwordName
                        + " = document.getElementById('" + id + "');\n" + "try{\n" + id
                        + " = " + passwordName + ".value;\n" + "}catch(err){\n" + id
                        + " = '';\n" + "}\n";

        // write html
        htmlText += "htmlText += '<input type=\"password\" name=\"" + id
```

```
                        + "\" id=\"" + id + "\"" + size + maxlength + " value =\"'+"
                        + id + "+'\"" + " onKeyUp=\"writeForm(\\'" + id
                        + "\\')\"/>';\n";

        return htmlText;

}

/*
 * parse textArea tag
 */
private String parseTextArea(Element e) throws Exception {

        String htmlText = "";

        // read attribute
        String id = e.getAttribute("id");
        String textAreaName = id + "TextArea";
        Attr colsAttr = e.getAttributeNode("cols");
        String cols = (colsAttr == null) ? "" : " cols=\""
                        + colsAttr.getValue() + "\"";
        Attr rowsAttr = e.getAttributeNode("rows");
        String rows = (rowsAttr == null) ? "" : " rows=\""
                        + rowsAttr.getValue() + "\"";

        // read value
        htmlText += "\n";
        htmlText += "//read " + id + " value\n";
        htmlText += "var " + id + ";\n" + "var " + textAreaName
                        + " = document.getElementById('" + id + "');\n" + "try{\n" + id
                        + " = " + textAreaName + ".value;\n" + "}catch(err){\n" + id
                        + " = '';\n" + "}\n";

        // write html
        htmlText += "htmlText += '<textarea name=\"" + id + "\" id=\"" + id
                        + "\"" + cols + rows + " onKeyUp=\"writeForm(\\'" + id
                        + "\\')\">" + "</textarea>';\n";

        return htmlText;

}

/*
 * parse selectOne tag
 */
private String parseSelectOne(Element e) throws Exception {
        String htmlText = "";

        // read attributes
        String id = e.getAttribute("id");
        String selectName = id + "Select";
        Attr sizeAttr = e.getAttributeNode("size");
        String size = (sizeAttr == null) ? "" : " size=\""
                        + sizeAttr.getValue() + "\"";

        _parameterName = id;
        _parameterType = SELECT;

        // read value
        htmlText += "\n";
        htmlText += "//read " + id + " value\n";
        htmlText += "var " + id + ";\n" + "var " + selectName
                        + " = document.getElementById('" + id + "');\n" + "try{\n" + id
                        + " = " + selectName + ".options[" + selectName
                        + ".selectedIndex].value;\n" + "}catch(err){\n" + id
                        + " = null;\n" + "}\n";

        // write html
        htmlText += "\n";
        htmlText += "htmlText += '<select name=\"" + id + "\" id=\"" + id
                        + "\"" + size + " onChange=\"writeForm(\\'\\')\">';\n";
```

```
        NodeList children = e.getChildNodes();
        if (children != null) {
                for (int i = 0; i < children.getLength(); i++) {
                        Node child = children.item(i);
                        if (child.getNodeType() == Node.ELEMENT_NODE) {
                                htmlText += parseTag((Element) child);
                        }
                }
        }

        htmlText += "htmlText += '</select>';\n";
        _parameterName = null;
        _isMultiSelect = false;
        _parameterType = null;

        return htmlText;
}

/*
 * parse selectMany tag
 */
private String parseSelectMany(Element e) throws Exception {

        String htmlText = "";

        // read attributes
        String id = e.getAttribute("id");
        String selectName = id + "Select";
        Attr sizeAttr = e.getAttributeNode("size");
        String size = (sizeAttr == null) ? "" : " size=\""
                        + sizeAttr.getValue() + "\"";

        _parameterName = id;
        _parameterType = SELECT;

        // read value
        htmlText += "\n";
        htmlText += "//read " + id + " value\n";
        htmlText += "var " + id + " = new Array();\n" + id
                        + ".numSelected=0;\n" + "var " + selectName
                        + " = document.getElementById('" + id + "');\n" + "try{\n"
                        + "for(var i=0;i<" + selectName + ".options.length;i++){\n"
                        + id + "[i] = " + selectName + ".options[i];\n" + "if("
                        + selectName + ".options[i].selected)\n" + id
                        + ".numSelected += 1;\n" + "}\n" + "}catch(err){\n" + id
                        + " = null;\n" + "}\n";

        // write html
        htmlText += "\n";
        htmlText += "htmlText += '<select name=\"" + id + "\" id=\"" + id
                        + "\"" + size
                        + " multiple=\"multiple\" onChange=\"writeForm(\\'\\')\">';\n";

        NodeList children = e.getChildNodes();
        if (children != null) {
                for (int i = 0; i < children.getLength(); i++) {
                        Node child = children.item(i);
                        if (child.getNodeType() == Node.ELEMENT_NODE) {
                                htmlText += parseTag((Element) child);
                        }
                }
        }

        htmlText += "htmlText += '</select>';\n";
        _parameterName = null;
        _isMultiSelect = false;
        _parameterType = null;

        return htmlText;
}
```

```
/*
 * parse checkbox tag
 */
private String parseCheckbox(Element e) throws Exception {
        String htmlText = "";

        // read attribute
        String name = e.getAttribute("id");
        String checkBoxName = name + "CheckBox";

        _parameterName = name;
        _parameterType = CHECKBOX;

        // read value
        htmlText += "\n";
        htmlText += "//read " + name + " value\n";
        htmlText += "var " + name + " = new Array();\n" + name
                        + ".numChecked=0;\n" + "var " + checkBoxName
                        + " = document.myform." + name + ";\n" + "try{\n"
                        + "for(var i=0;i<" + checkBoxName + ".length;i++){\n" + name
                        + "[i] = " + checkBoxName + "[i];\n" + "if(" + checkBoxName
                        + ".checked)\n" + name + ".numChecked += 1;\n" + "}\n"
                        + "}catch(err){\n" + name + " = null;\n" + "}\n";

        NodeList children = e.getChildNodes();
        for (int i = 0; i < children.getLength(); i++) {

                Node child = children.item(i);

                // remove newline, tab, space
                if (child.getNodeType() == Node.TEXT_NODE) {
                        String text = child.getNodeValue().replaceAll(
                                        "(\\n|\\s{2}|\\t)", "");
                        if (!text.equals("")) {
                                htmlText += "htmlText += '" + text + "';\n";
                        }

                } else if (child.getNodeType() == Node.ELEMENT_NODE) {
                        htmlText += parseTag((Element) child);
                }
        }

        _parameterName = null;
        _parameterType = null;

        return htmlText;
}

/**
 * parse optionItem tag
 *
 * @param e
 * @return
 * @throws Exception
 */
private String parseOptionItem(Element e) throws Exception {

        String htmlText = "";

        if (_parameterType.equals(SELECT)) {
                htmlText = parseSelectOption(e);
        } else if (_parameterType.equals(CHECKBOX)) {
                htmlText = parseCheckBoxOption(e);
        } else if (_parameterType.equals(RADIO)) {
                htmlText = parseRadioOption(e);
        }

        return htmlText;
}
```

```
/*
 * write checkbox option
 */
private String parseCheckBoxOption(Element e) throws Exception {

        String htmlText = "";
        String value = e.getAttribute("value");
        Attr checkedAttr = e.getAttributeNode("selected");
        String checked = (checkedAttr != null && checkedAttr.getValue().equals(
                        "true")) ? " checked=\"checked\"" : "";
        Attr labelAttr = e.getAttributeNode("label");
        String label = (labelAttr != null) ? labelAttr.getValue() : value;

        htmlText += "if(" + _parameterName + " != null){\n" + "for(var i=0;i<"
                        + _parameterName + ".length;i++){\n" + "if(" + _parameterName
                        + "[i].value =='" + value + "'){\n" + "if(" + _parameterName
                        + "[i].checked){\n"
                        + "htmlText += '<input type=\"checkbox\" name=\""
                        + _parameterName + "\" value =\"" + value
                        + "\" checked onClick=\"writeForm(\\'\\')\" />" + label
                        + "';\n" + "} else {\n"
                        + "htmlText += '<input type=\"checkbox\" name=\""
                        + _parameterName + "\" value =\"" + value
                        + "\" onClick=\"writeForm(\\'\\')\" />" + label + "';\n"
                        + "}\n" + "break;\n" + "}\n" + "if (i==" + _parameterName
                        + ".length-1){\n"
                        + "htmlText += '<input type=\"checkbox\" name=\""
                        + _parameterName + "\" value =\"" + value + "\"" + checked
                        + " onClick=\"writeForm(\\'\\')\" />" + label + "';\n" + "}\n"
                        + "}\n" + "}else{\n"
                        + "htmlText += '<input type=\"checkbox\" name=\""
                        + _parameterName + "\" value =\"" + value + "\"" + checked
                        + " onClick=\"writeForm(\\'\\')\" />" + label + "';\n" + "}\n";

        return htmlText;
}

/*
 * write select option
 */
private String parseSelectOption(Element e) throws Exception {

        String htmlText = "";
        String value = e.getAttribute("value");
        Attr selectedAttr = e.getAttributeNode("selected");
        String selected = (selectedAttr != null && selectedAttr.getValue()
                        .equals("true")) ? " selected=\"selected\"" : "";
        Attr labelAttr = e.getAttributeNode("label");
        String label = (labelAttr != null) ? labelAttr.getValue() : value;

        if (_isMultiSelect) {
                // multiple select
                htmlText += "if(" + _parameterName + " != null){\n"
                                + "for(var i=0;i<" + _parameterName + ".length;i++){\n"
                                + "if(" + _parameterName + "[i].value =='" + value
                                + "'){\n" + "if(" + _parameterName + "[i].selected){\n"
                                + "htmlText += '<option value=\"" + value + "\" selected >"
                                + label + "</option>';\n" + "} else {\n"
                                + "htmlText += '<option value=\"" + value + "\" >" + label
                                + "</option>';\n" + "}\n" + "break;\n" + "}\n" + "if (i=="
                                + _parameterName + ".length-1){\n"
                                + "htmlText += '<option value=\"" + value + "\"" + selected
                                + ">" + label + "</option>';\n" + "}\n" + "}\n"
                                + "}else{\n" + "htmlText += '<option value=\"" + value
                                + "\"" + selected + ">" + label + "</option>';\n" + "}\n";
        } else {
                // single select
                htmlText += "if(" + _parameterName + " != null && "
                                + _parameterName + " == '" + value + "'){\n"
                                + "htmlText += '<option value=\"" + value + "\" selected >"
                                + label + "</option>';\n" + "}else{\n"
```

```
                                + "htmlText += '<option value=\"" + value + "\"" + selected
                                + ">" + label + "</option>';\n" + "}\n";

        }

        return htmlText;
}

/*
 * write radio option
 */
private String parseRadioOption(Element e) throws Exception {

        String htmlText = "";
        String value = e.getAttribute("value");
        Attr checkedAttr = e.getAttributeNode("selected");
        String checked = (checkedAttr != null && checkedAttr.getValue().equals(
                        "true")) ? " selected=\"checked\"" : "";
        Attr labelAttr = e.getAttributeNode("label");
        String label = (labelAttr != null) ? labelAttr.getValue() : value;

        htmlText += "if(" + _parameterName + " != null && " + _parameterName
                        + " == '" + value + "'){\n"
                        + "htmlText += '<input type=\"radio\" name=\"" + _parameterName
                        + "\" value=\"" + value + "\" checked"
                        + " onClick =\"writeForm(\\'\\')\" /> " + label + "';\n"
                        + "}else{\n" + "htmlText += '<input type=\"radio\" name=\""
                        + _parameterName + "\" value=\"" + value + "\"" + checked
                        + " onClick =\"writeForm(\\'\\')\" /> " + label + "';\n"
                        + "}\n";

        return htmlText;
}

/*
 * parse submit tag
 */
private String parseSubmit(Element e) throws Exception {

        Attr valueAttr = e.getAttributeNode("value");
        String value = (valueAttr == null) ? "" : " value=\""
                        + valueAttr.getValue() + "\"";

        return "htmlText += '<input type=\"submit\"" + value + "/>';\n";
}

/*
 * parse reset tag
 */
private String parseReset(Element e) throws Exception {

        Attr valueAttr = e.getAttributeNode("value");
        String value = (valueAttr == null) ? "" : " value=\""
                        + valueAttr.getValue() + "\"";

        return "htmlText += '<input type=\"reset\"" + value + "/>';\n";
}

/*
 * parse if tag
 */
private String parseIf(Element e) throws Exception {

        String test = e.getAttribute("test");
        String htmlText = "\ntry{\n" + "if(" + test + "){\n";

        NodeList children = e.getChildNodes();
        for (int i = 0; i < children.getLength(); i++) {
                Node child = children.item(i);

                if (child.getNodeType() == Node.TEXT_NODE) {
```

```
                        // remove tab, newline, space
                        String text = child.getNodeValue().replaceAll(
                                        "(\\n|\\s{2}|\\t)", "");
                        if (!text.equals("")) {
                                htmlText += "htmlText += '" + text + "';\n";
                        }
                } else if (child.getNodeType() == Node.ELEMENT_NODE) {
                        htmlText += parseTag((Element) child);
                }
        }

        htmlText += "}\n" + "}catch(err){}\n";

        return htmlText;
}

/*
 * parse choose tag
 */
private String parseChoose(Element e) throws Exception {
        String htmlText = "\ntry{\n";
        int numWhen = 0;

        NodeList children = e.getChildNodes();
        for (int i = 0; i < children.getLength(); i++) {
                Node child = children.item(i);
                if (child.getNodeType() == Node.ELEMENT_NODE) {
                        Element eChild = (Element) child;
                        if (eChild.getTagName().equals("when")) {
                                numWhen++;
                                htmlText += parseWhen((Element) child, numWhen);

                        } else if (eChild.getTagName().equals("otherwise")) {
                                htmlText += parseOtherwise((Element) child);
                                break;
                        }
                }
        }

        htmlText += "}catch(err){}\n";

        return htmlText;
}

/*
 * parse outputText tag
 */
private String parseOutputText(Element e) throws Exception {
        String htmlText = "";
        Attr valueAttr = e.getAttributeNode("value");
        String value = (valueAttr == null) ? "" : valueAttr.getValue();

        htmlText += "htmlText +=  " + value + ";\n";

        return htmlText;
}

/*
 * parse when tag
 */
private String parseWhen(Element e, int numWhen) throws Exception {
        String htmlText = "";

        String test = e.getAttribute("test");
        if (numWhen == 1) {
                htmlText += "if(" + test + "){\n";
        } else {
                htmlText += "else if(" + test + "){\n";
        }
```

```
		NodeList children = e.getChildNodes();
		for (int i = 0; i < children.getLength(); i++) {
			Node child = children.item(i);
			if (child.getNodeType() == Node.TEXT_NODE) {

				// remove tab, newline, space
				String text = child.getNodeValue().replaceAll(
								"(\\n|\\s{2}|\\t)", "");
				if (!text.equals("")) {
					htmlText += "htmlText += '" + text + "';\n";
				}
			} else if (child.getNodeType() == Node.ELEMENT_NODE) {
				htmlText += parseTag((Element) child);
			}
		}

		htmlText += "}\n";

		return htmlText;
	}

	/*
	 * parse otherwise tag
	 */
	private String parseOtherwise(Element e) throws Exception {
		String htmlText = "else {\n";

		NodeList children = e.getChildNodes();
		for (int i = 0; i < children.getLength(); i++) {
			Node child = children.item(i);
			if (child.getNodeType() == Node.TEXT_NODE) {
				String text = child.getNodeValue().replaceAll(
								"(\\n|\\s{2}|\\t)", "");
				if (!text.equals("")) {
					htmlText += "htmlText += '" + text + "';\n";
				}
			} else if (child.getNodeType() == Node.ELEMENT_NODE) {
				htmlText += parseTag((Element) child);
			}
		}

		htmlText += "}\n";

		return htmlText;
	}

}
```

Class FormValidator.java

```
package webformParser.html;

import org.w3c.dom.Attr;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

/**
 * Class to generate javascript to validate form
 *
 * @author Piyaphol Phoungphol
 *
 */
class FormValidator {

	private String _parameterName = null;

	private String _submitCondition = null;

	private final String reqxInteger = "^\\d+$";

	private final String reqxDouble = "^(\\d+|\\d+\\.\\d+|\\.\\d+)$";

	private final String reqxString = "^\\w+";

	private final String INTEGER = "integer";

	private final String DECIMAL = "decimal";

	private final String STRING = "string";

	/**
	 * parse webform tag
	 *
	 * @param e
	 * @return
	 * @throws Exception
	 */
	public String parseWebForm(Element e) throws Exception {
		String htmlText = "function validateForm(){\n" + "var result = '';\n";

		NodeList children = e.getChildNodes();
		for (int i = 0; i < children.getLength(); i++) {
			Node child = children.item(i);
			if (child.getNodeType() == Node.ELEMENT_NODE) {
				htmlText += parseTag((Element) child);
			}
		}

		htmlText += "\n //check all validation\n" + "if(result!= ''){\n"
				+ "alert(result);\n" + "return false;\n" + "}\n";

		if (_submitCondition != null) {

			htmlText += "\n //check submit condition\n" + "if("
					+ _submitCondition + "){\n"
					+ "alert('The form cannot be submitted');\n"
					+ "return false;\n" + "}\n";
		}

		htmlText += "return true;\n" + "}\n";

		return htmlText;
	}

	/*
	 * parse xml tags
	 */
```

```
private String parseTag(Element e) throws Exception {

        String tagName = e.getTagName();
        String htmlText = "";

        if (tagName.equals("textField")) {
                htmlText = validateTextField(e);
        } else if (tagName.equals("textArea")) {
                htmlText = validateTextArea(e);
        } else if (tagName.equals("selectOne")) {
                htmlText = validateSelectOne(e);
        } else if (tagName.equals("selectMany")) {
                htmlText = validateSelectMany(e);
        } else if (tagName.equals("radio")) {
                htmlText = validateRadio(e);
        } else if (tagName.equals("checkbox")) {
                htmlText = validateCheckbox(e);
        } else if (tagName.equals("password")) {
                htmlText = validatePassword(e);
        } else if (tagName.equals("if")) {
                htmlText = parseIf(e);
        } else if (tagName.equals("choose")) {
                htmlText = parseChoose(e);
        } else if (tagName.equals("submit")) {
                parseSubmit(e);
        } else {
                htmlText = parseHtmlTag(e);
        }

        return htmlText;
}

/*
 * parse submit tag
 */
private void parseSubmit(Element e) throws Exception {
        Attr testAttr = e.getAttributeNode("test");
        _submitCondition = (testAttr != null) ? testAttr.getValue() : null;
}

/*
 * parse html tags
 */
private String parseHtmlTag(Element e) throws Exception {
        String htmlText = "";

        NodeList children = e.getChildNodes();
        for (int i = 0; i < children.getLength(); i++) {
                Node child = children.item(i);
                if (child.getNodeType() == Node.ELEMENT_NODE) {
                        htmlText += parseTag((Element) child);
                }
        }

        return htmlText;
}

/*
 * parse radio tag
 */
private String validateRadio(Element e) throws Exception {
        String htmlText = "";
        String id = e.getAttribute("id");
        String radioName = id + "Radio";
        Attr requiredAttr = e.getAttributeNode("required");
        boolean required = (requiredAttr != null && requiredAttr.getValue()
                        .equals("true")) ? true : false;

        // read value
        htmlText += "\n";
        htmlText += "//read " + id + " value\n";
```

```
        htmlText += "var " + id + "='';\n" + "var " + radioName
                        + " = document.myform." + id + ";\n" + "for (i=0;i<"
                        + radioName + ".length;i++){\n" + "if (" + radioName
                        + "[i].checked){\n" + id + " = " + radioName + "[i].value;\n"
                        + "break;\n" + "}\n" + "}\n";

        // validate radio box
        if (required) {
                htmlText += "//validate" + id + "\n" + "if(" + id + "==''){\n"
                                + "result += '" + id + " is required.\\n';\n" + "}\n";
        }

        NodeList children = e.getChildNodes();
        for (int i = 0; i < children.getLength(); i++) {
                Node child = children.item(i);
                if (child.getNodeType() == Node.ELEMENT_NODE) {
                        htmlText += parseTag((Element) child);
                }
        }

        return htmlText;
}

/*
 *
 */
private String validateTextField(Element e) throws Exception {

        String htmlText = "";
        String id = e.getAttribute("id");
        String textBoxName = id + "TextField";

        Attr typeAttr = e.getAttributeNode("type");
        String type = (typeAttr == null) ? STRING : typeAttr.getValue();
        Attr requiredAttr = e.getAttributeNode("required");
        boolean required = (requiredAttr != null && requiredAttr.getValue()
                        .equals("true")) ? true : false;
        Attr reqxAttr = e.getAttributeNode("pattern");
        String reqx = (reqxAttr == null) ? "" : reqxAttr.getValue();

        // regular expression is not specify, appy type to regular expression
        if (reqx.equals("")) {
                if (type.equals(INTEGER)) {
                        reqx = reqxInteger;
                } else if (type.equals(DECIMAL)) {
                        reqx = reqxDouble;
                } else {
                        reqx = reqxString;
                }
        }

        // read value
        htmlText += "\n";
        htmlText += "//read " + id + " value\n";
        htmlText += "var " + id + "='';\n" + "var " + textBoxName
                        + " = document.getElementById('" + id + "');\n" + id + " = "
                        + textBoxName + ".value;\n";

        // validate form
        htmlText += "if(" + id + "==''){\n" + "\t// empty\n" + "if(" + required
                        + ")\n\t//required \n result += '" + id
                        + " is required.\\n';\n" + "} else {\n" + "\t// not empty\n"
                        + "if(!" + id + ".match(/" + reqx + "/))\n result += '" + id
                        + " is invalid format\\n';\n" + "}\n";

        return htmlText;
}

/*
 * parse password tag
 */
```

```
private String validatePassword(Element e) throws Exception {

        String htmlText = "";
        String id = e.getAttribute("id");
        String passwordName = id + "Password";

        Attr typeAttr = e.getAttributeNode("type");
        String type = (typeAttr == null) ? "string" : typeAttr.getValue();
        Attr requiredAttr = e.getAttributeNode("required");
        boolean required = (requiredAttr != null && requiredAttr.getValue()
                        .equals("true")) ? true : false;
        Attr reqxAttr = e.getAttributeNode("pattern");
        String reqx = (reqxAttr == null) ? "" : reqxAttr.getValue();

        // regular expression is not specify, appy type to regular expression
        if (reqx.equals("")) {
                if (type.equals(INTEGER)) {
                        reqx = reqxInteger;
                } else if (type.equals(DECIMAL)) {
                        reqx = reqxDouble;
                } else {
                        reqx = reqxString;
                }
        }

        // read value
        htmlText += "\n";
        htmlText += "//read " + id + " value\n";
        htmlText += "var " + id + "='';\n" + "var " + passwordName
                        + " = document.getElementById('" + id + "');\n" + id + " = "
                        + passwordName + ".value;\n";

        // validate form
        htmlText += "if(" + id + "==''){\n" + "\t// empty\n" + "if(" + required
                        + ")\n\t//required \n result += '" + id
                        + " is required.\\n';\n" + "} else {\n" + "\t// not empty\n"
                        + "if(!" + id + ".match(/" + reqx + "/))\n result += '" + id
                        + " is invalid format\\n';\n" + "}\n";
        return htmlText;

}

/*
 * parse textArea tag
 */
private String validateTextArea(Element e) throws Exception {

        String htmlText = "";
        String id = e.getAttribute("id");
        String textAreaName = id + "TextArea";

        Attr typeAttr = e.getAttributeNode("type");
        String type = (typeAttr == null) ? "string" : typeAttr.getValue();
        Attr requiredAttr = e.getAttributeNode("required");
        boolean required = (requiredAttr != null && requiredAttr.getValue()
                        .equals("true")) ? true : false;
        Attr reqxAttr = e.getAttributeNode("reqx");
        String reqx = (reqxAttr == null) ? "" : reqxAttr.getValue();

        // regular expression is not specify, appy type to regular expression
        if (reqx.equals("")) {
                if (type.equals("integer")) {
                        reqx = reqxInteger;
                } else if (type.equals("double")) {
                        reqx = reqxDouble;
                } else {
                        reqx = reqxString;
                }
        }

        // read value
```

```
        htmlText += "\n";
        htmlText += "//read " + id + " value\n";
        htmlText += "var " + id + "='';\n" + "var " + textAreaName
                        + " = document.getElementById('" + id + "');\n" + id + " = "
                        + textAreaName + ".value;\n";

        // validate form
        htmlText += "if(" + id + "==''){\n" + "\t// empty\n" + "if(" + required
                        + ")\n\t//required \n result += '" + id
                        + " is required.\\n';\n" + "} else {\n" + "\t// not empty\n"
                        + "if(!" + id + ".match(/" + reqx + "/))\n result += '" + id
                        + " is invalid format.\\n';\n" + "}\n";

        return htmlText;

}

/*
 * parse selectOne tag
 */
private String validateSelectOne(Element e) throws Exception {
        String htmlText = "";
        String id = e.getAttribute("id");
        String selectName = id + "Select";
        Attr requiredAttr = e.getAttributeNode("required");
        boolean required = (requiredAttr != null && requiredAttr.getValue()
                        .equals("true")) ? true : false;

        // read value
        htmlText += "\n";
        htmlText += "//read " + id + " value\n";
        htmlText += "var " + id + ";\n" + "var " + selectName
                        + " = document.getElementById('" + id + "');\n" + id + " = "
                        + selectName + ".options[" + selectName
                        + ".selectedIndex].value;\n";

        // validate form
        htmlText += "if(" + id + "==''){\n" + "\t// empty\n" + "if(" + required
                        + ")\n\t//required \n result += '" + id
                        + " is required.\\n';\n" + "}\n";

        return htmlText;
}

/*
 * parse selectMany
 */
private String validateSelectMany(Element e) throws Exception {
        String htmlText = "";
        String id = e.getAttribute("id");
        String selectName = id + "Select";
        Attr minSelectedAttr = e.getAttributeNode("minSelected");
        String minSelected = (minSelectedAttr == null) ? "0" : minSelectedAttr
                        .getValue();
        Attr maxSelectedAttr = e.getAttributeNode("maxSelected");
        String maxSelected = (maxSelectedAttr == null) ? "10000"
                        : minSelectedAttr.getValue();

        // read value
        htmlText += "\n";
        htmlText += "//read " + id + " value\n";
        htmlText += "var " + id + " = new Array();\n" + id
                        + ".numSelected=0;\n" + "var " + selectName
                        + " = document.getElementById('" + id + "');\n"
                        + "for(var i=0;i<" + selectName + ".options.length;i++){\n"
                        + id + "[i] = " + selectName + ".options[i];\n" + "if("
                        + selectName + ".options[i].selected)\n" + id
                        + ".numSelected += 1;\n" + "}\n";

        // validate multiple selectlist.
        htmlText += "// validate multiple selectlist. \n";
```

```
        htmlText += "if(!(" + id + ".numSelected >=" + minSelected + " && "
                        + id + ".numSelected <=" + maxSelected + ")){\n"
                        + " result += 'the number of selected " + id
                        + " is invalid.\\n';\n" + "}\n";

        return htmlText;
}

/*
 * parse checkbox tag
 */
private String validateCheckbox(Element e) throws Exception {
        String htmlText = "";
        String id = e.getAttribute("id");
        String checkBoxName = id + "CheckBox";
        Attr minSelectedAttr = e.getAttributeNode("minSelected");
        String minSelected = (minSelectedAttr == null) ? "0" : minSelectedAttr
                        .getValue();
        Attr maxSelectedAttr = e.getAttributeNode("maxSelected");
        String maxSelected = (maxSelectedAttr == null) ? "10000"
                        : minSelectedAttr.getValue();

        // read value
        htmlText += "\n";
        htmlText += "//read " + id + " value\n";
        htmlText += "var " + id + " = new Array();\n" + id + ".numChecked=0;\n"
                        + "var " + checkBoxName + " = document.myform." + id + ";\n"
                        + "for(var i=0;i<" + checkBoxName + ".length;i++){\n" + id
                        + "[i] = " + checkBoxName + "[i];\n" + "if(" + checkBoxName
                        + "[i].checked)\n" + id + ".numChecked += 1;\n" + "}\n";

        // validate check box.
        htmlText += "// validate check box. \n";
        htmlText += "if(!(" + id + ". numChecked >=" + minSelected + " && "
                        + id + ". numChecked <=" + maxSelected + ")){\n"
                        + " result += 'the number of checked " + id
                        + " is invalid\\n';\n" + "}\n";

        NodeList children = e.getChildNodes();
        for (int i = 0; i < children.getLength(); i++) {
                Node child = children.item(i);
                if (child.getNodeType() == Node.ELEMENT_NODE) {
                        htmlText += parseTag((Element) child);
                }
        }

        return htmlText;
}
/*
 * parse if tag
 */
private String parseIf(Element e) throws Exception {

        String test = e.getAttribute("test");
        String htmlText = "\ntry{\n" + "if(" + test + "){\n";

        NodeList children = e.getChildNodes();
        for (int i = 0; i < children.getLength(); i++) {
                Node child = children.item(i);
                if (child.getNodeType() == Node.ELEMENT_NODE) {
                        htmlText += parseTag((Element) child);
                }
        }

        htmlText += "}\n"
                        + "}catch(err){\n result += 'some thing wrong';\n }\n";

        return htmlText;
}

/*
```

```
	 * parse choose tag
	 */
	private String parseChoose(Element e) throws Exception {
		String htmlText = "\ntry{\n";
		int numWhen = 0;

		NodeList children = e.getChildNodes();
		for (int i = 0; i < children.getLength(); i++) {
			Node child = children.item(i);
			if (child.getNodeType() == Node.ELEMENT_NODE) {
				Element eChild = (Element) child;
				if (eChild.getTagName().equals("when")) {
					numWhen++;
					htmlText += parseWhen((Element) child, numWhen);

				} else if (eChild.getTagName().equals("otherwise")) {
					htmlText += parseOtherwise((Element) child);
					break;
				}
			}
		}

		htmlText += "}catch(err){}\n";

		return htmlText;
	}

	/*
	 * parse when tag
	 */
	private String parseWhen(Element e, int numWhen) throws Exception {
		String htmlText = "";
		String test = e.getAttribute("test");
		if (numWhen == 1) {
			htmlText += "if(" + test + "){\n";
		} else {
			htmlText += "else if(" + test + "){\n";
		}

		NodeList children = e.getChildNodes();
		for (int i = 0; i < children.getLength(); i++) {
			Node child = children.item(i);
			if (child.getNodeType() == Node.ELEMENT_NODE) {
				htmlText += parseTag((Element) child);
			}
		}

		htmlText += "}\n";

		return htmlText;
	}

	/*
	 * parse otherwise tag
	 */
	private String parseOtherwise(Element e) throws Exception {
		String htmlText = "else {\n";

		NodeList children = e.getChildNodes();
		for (int i = 0; i < children.getLength(); i++) {
			Node child = children.item(i);
			if (child.getNodeType() == Node.ELEMENT_NODE) {
				htmlText += parseTag((Element) child);
			}
		}

		htmlText += "}\n";

		return htmlText;
	}
}
```

A.3 Package webformParser.php.processForm

Class PHPProcessFormGenerator

```
package webformParser.php.processForm;

import org.w3c.dom.Document;
import org.w3c.dom.Element;

import webformParser.sql.Column;
import webformParser.sql.SQLScriptGenrator;
import webformParser.sql.Table;

/**
 * Class to generate php script to process webform
 *
 * @author Piyaphol Phoungphol
 *
 */
public class PHPProcessFormGenerator {

	/**
	 * get php script String to process webform from xml document
	 *
	 * @param xmlForm
	 * @return
	 * @throws Exception
	 */
	public static String getPHP(Document xmlForm) throws Exception {
		String phpText = "";

		Element root = xmlForm.getDocumentElement();

		if (root.getTagName().equals("webform")) {

			String webformId = root.getAttribute("id");

			phpText += "<?php\n\n";
			phpText += createConnection(webformId);
			phpText += writePhpInsertData(root);
			phpText += "mysql_close();\n";
			phpText += forward();
			phpText += "?>\n";

		}

		return phpText;

	}

	/*
	 * write javascipt to forward when finished
	 */
	private static String forward() {

		String phpText = "";
		phpText += "echo '<script type=\"text/javascript\">';\n";
		phpText += "echo \"location.href='$forwardURL';\";\n";
		phpText += "echo '</script>';\n";

		return phpText;
	}

	/*
	 * include database connection from webform properties file
	 */
	private static String createConnection(String webformId) {
		String phpText = "";

		phpText += "// read all db parameters\n";
```

```
                phpText += "include '" + webformId + "Properties.php';\n\n";

                phpText += "//make a connection\n";
                phpText += "mysql_connect($dbHostname, $dbUsername, $dbPassword) or
die(mysql_error());\n";
                phpText += "mysql_select_db($dbname) or die(mysql_error());\n\n";

                return phpText;
        }

        /*
         * write SQL insert statement to execute
         */
        private static String writePhpInsertData(Element e) throws Exception {
                String phpText = "";

                SQLScriptGenrator sqlReader = new SQLScriptGenrator();
                sqlReader.parseWebForm(e);

                Table[] tables = sqlReader.getTables();

                for (int i = tables.length - 1; i >= 0; i--) {
                        String tableName = tables[i].getName();
                        Column[] columns = sqlReader.getColumns(tableName);

                        phpText += "//insert to table: " + tableName + "\n";
                        phpText += "if(";

                        for (int j = 0; j < columns.length - 1; j++) {
                                phpText += "isset($_POST['" + columns[j].getName() + "']) && ";
                        }
                        phpText += "isset($_POST['" + columns[columns.length - 1].getName()
                                        + "'])){\n";

                        if (!tables[i].isMultipleChoice()) {
                                phpText += "  $query = \"INSERT INTO " + tableName
                                                + " VALUES (";
                                for (int j = 0; j < columns.length - 1; j++) {
                                        phpText += "'$_POST[" + columns[j].getName() + "]',";
                                }
                                phpText += "'$_POST[" + columns[columns.length - 1].getName()
                                                + "]')\";\n";
                                phpText += "  mysql_query($query) or die(mysql_error());\n";

                        } else {
                                String multiChoice = columns[0].getName();
                                phpText += "  foreach (" + "$_POST['" + columns[0].getName()
                                                + "'] as $" + multiChoice + "){\n";

                                phpText += "    $query = \"INSERT INTO " + tableName
                                                + " VALUES (";
                                phpText += "'$" + multiChoice + "'";
                                for (int j = 1; j < columns.length; j++) {
                                        phpText += ",'$_POST[" + columns[j].getName() + "]'";
                                }
                                phpText += ")\";\n";
                                phpText += "    mysql_query($query) or die(mysql_error());\n";

                                phpText += "  }\n";
                        }
                        phpText += "}\n\n";

                }

                return phpText;
        }

        /**
         * method to generate properties template for webform
         *
         * @param webFormId
```

```
	 * @return
	 */
	public static String getProperties(String webFormId) {
		String phpText = "";

		phpText += "<?php\n\n" + "\t//Properties file for " + webFormId
				+ " webform\n" + "\t$dbUsername = 'username';\n"
				+ "\t$dbPassword = 'password';\n"
				+ "\t$dbHostname = 'dbhostName'\n;" + "\t$dbname = 'dbname';\n"
				+ "\t$forwardURL = 'nextpage.html';\n\n" + "?>";

		return phpText;
	}

}
```

A.4 Package webformParser.sql

Class SQLScriptGenrator

```
package webformParser.sql;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;

import org.w3c.dom.Attr;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

/**
 * Class to generate SQL script and database schema
 *
 * @author Piyaphol Phoungphol
 *
 */
public class SQLScriptGenrator {

	private ArrayList _tables = new ArrayList();

	private int _ifCounter = 0;

	private int _chooseCounter = 0;

	private int _selectCounter = 0;

	private int _checkboxCounter = 0;

	private String _id = "";

	private Set _key = new HashSet();

	private Table _mainTable = null;

	/*
	 * function to remove table that doesn't have any column and add foreign key
	 * to every table
	 */
	private void organizeTables() {

		// delete empty table
		for (Iterator it = _tables.iterator(); it.hasNext();) {
			Table table = (Table) it.next();
			if (table.getColumns().length == 0) {
				// delete table
				// System.out.println("delete table:"+table.getName());
				it.remove();
			}
		}

		// System.out.println("total tables after deleted:"+_tables.size());

		// add foreign key to every table
		for (int i = 0; i < _tables.size() - 1; i++) {
			Table table = (Table) _tables.get(i);
			// System.out.println("set foreign key table"+ i);
			table.setForeignKey();
		}

	}
```

```
/*
 * return array of all tables
 */
public Table[] getTables() {

        return (Table[]) _tables.toArray(new Table[_tables.size()]);
}

/*
 * return all columns of the given table name
 */
public Column[] getColumns(String tableName) {
        Column[] cols = null;

        for (int i = 0; i < _tables.size(); i++) {
                Table table = (Table) _tables.get(i);
                if (table.getName().equals(tableName)) {
                        cols = table.getColumns();
                        break;
                }
        }

        return cols;
}

/*
 * function to parse xml data form and return sql script string using in
 * creating databse
 */
public String getSQL(Document xmlForm) throws Exception {
        String sqlSriptText = "";

        Element root = xmlForm.getDocumentElement();
        parseWebForm(root);
        sqlSriptText = writeSQLSript();

        return sqlSriptText;
}

/*
 * generate sql script from tables arraylist
 */
private String writeSQLSript() {
        String sqlSriptText = "";

        sqlSriptText += "DROP TABLE IF EXISTS ";
        for (int i = 0; i < _tables.size(); i++) {
                Table table = (Table) _tables.get(i);
                sqlSriptText += table.getName() + ",";
        }
        if (sqlSriptText.endsWith(",")) {
                sqlSriptText = sqlSriptText.substring(0, sqlSriptText.length() - 1);
        }

        sqlSriptText += " CASCADE;\n\n";

        for (int i = _tables.size() - 1; i >= 0; i--) {
                Table table = (Table) _tables.get(i);
                sqlSriptText += table.getSQl() + "\n";
        }

        sqlSriptText += "\n";
        sqlSriptText += getView();

        return sqlSriptText;
}

/**
 * get the webform mainView String
 *
 * @return
```

```
 */
public String getView() {

        ArrayList SingleChoiceTable = new ArrayList();
        for (int i = 0; i < _tables.size(); i++) {
                Table table = (Table) _tables.get(i);
                if (!table.isMultipleChoice()) {
                        SingleChoiceTable.add(table);
                }
        }

        String sqlSriptText = "";
        sqlSriptText += "CREATE OR REPLACE VIEW " + _id + "MainView AS \n";
        sqlSriptText += "( SELECT ";
        for (Iterator it = _key.iterator(); it.hasNext();) {
                String key = (String) it.next();
                sqlSriptText += " " + _mainTable.getName() + "." + key + " AS "
                                + key + ",";
        }

        for (int i = SingleChoiceTable.size() - 1; i >= 0; i--) {
                String tableName = ((Table) SingleChoiceTable.get(i)).getName();
                Column[] cols = getColumns(tableName);
                for (int j = 0; j < cols.length; j++) {
                        if (!cols[j].isKey()) {
                                String colName = cols[j].getName();
                                sqlSriptText += " " + tableName + "." + colName + " AS "
                                                + colName + ",";
                        }
                }
        }

        if (sqlSriptText.endsWith(",")) {
                sqlSriptText = sqlSriptText.substring(0, sqlSriptText.length() - 1);
        }
        sqlSriptText += "\n";

        sqlSriptText += " FROM ";
        sqlSriptText += _mainTable.getName();

        for (int i = SingleChoiceTable.size() - 2; i >= 0; i--) {
                String tableName = ((Table) _tables.get(i)).getName();
                sqlSriptText += " LEFT JOIN " + tableName + " ON (";
                for (Iterator it = _key.iterator(); it.hasNext();) {
                        String key = (String) it.next();
                        sqlSriptText += " " + _mainTable.getName() + "." + key + " = "
                                        + tableName + "." + key + " AND";
                }

                if (sqlSriptText.endsWith("AND")) {
                        sqlSriptText = sqlSriptText.substring(0,
                                        sqlSriptText.length() - 3);
                }
                sqlSriptText += " ) ";

        }

        sqlSriptText += ") ";

        return sqlSriptText;
}

/**
 * parse xml data form from its root element and generate sql script
 *
 * @param e
 * @throws Exception
 */
public void parseWebForm(Element e) throws Exception {

        if (e.getTagName().equals("webform")) {
```

```
                _id = e.getAttribute("id");
                String keyString = e.getAttribute("key");

                String[] keysArray = keyString.split(",");

                for (int i = 0; i < keysArray.length; i++) {
                        _key.add(keysArray[i].trim());
                        // System.out.println("KEY:"+keysArray[i].trim());
                }

                _mainTable = new Table(_id + "_Main");

                NodeList children = e.getChildNodes();
                for (int i = 0; i < children.getLength(); i++) {
                        Node child = children.item(i);
                        if (child.getNodeType() == Node.ELEMENT_NODE) {
                                parseTag((Element) child, _mainTable);
                        }
                }

                _tables.add(_mainTable);
                // System.out.println("total tables after parsed:"+_tables.size());
                organizeTables();
        }
}

/*
 * parse xml tags and parent table
 */
private void parseTag(Element e, Table table) throws Exception {

        String tagName = e.getTagName();

        if (tagName.equals("textField") || tagName.equals("textArea")
                        || tagName.equals("Password")) {
                parseTextBox(e, table);
        } else if (tagName.equals("selectOne")) {
                parseSelectOne(e, table);
        } else if (tagName.equals("selectMany")) {
                parseSelectMany(e, table);
        } else if (tagName.equals("radio")) {
                parseRadio(e, table);
        } else if (tagName.equals("checkbox")) {
                parseCheckbox(e, table);
        } else if (tagName.equals("if")) {
                parseIf(e);
        } else if (tagName.equals("choose")) {
                parseChoose(e);
        } else {
                parseHtmlTag(e);
        }
}

/*
 * parse html tags
 */
private void parseHtmlTag(Element e) throws Exception {

        NodeList children = e.getChildNodes();
        for (int i = 0; i < children.getLength(); i++) {
                Node child = children.item(i);
                if (child.getNodeType() == Node.ELEMENT_NODE) {
                        parseTag((Element) child, _mainTable);
                }
        }
}

/*
 * parse radio tag
 */
```

```
private void parseRadio(Element e, Table table) throws Exception {

        String name = e.getAttribute("id");
        String type = e.getAttribute("type");
        Attr requiredAttr = e.getAttributeNode("required");
        boolean isRequired = (requiredAttr != null && requiredAttr.getValue()
                        .equals("true")) ? true : false;

        if (isKey(name)) {
                table.addKeyColumn(new Column(name, type, isRequired));
        } else {
                table.addColumn(new Column(name, type, isRequired));
        }

        // System.out.println("read radio:"+name);

        NodeList children = e.getChildNodes();
        for (int i = 0; i < children.getLength(); i++) {
                Node child = children.item(i);
                if (child.getNodeType() == Node.ELEMENT_NODE) {
                        parseTag((Element) child, table);
                }
        }

}

/*
 * parse textBox type textField, textArea, password Tag
 */
private void parseTextBox(Element e, Table table) throws Exception {
        String name = e.getAttribute("id");
        String type = e.getAttribute("type");
        Attr requiredAttr = e.getAttributeNode("required");
        boolean isRequired = (requiredAttr != null && requiredAttr.getValue()
                        .equals("true")) ? true : false;

        if (isKey(name)) {
                table.addKeyColumn(new Column(name, type, isRequired));
        } else {
                table.addColumn(new Column(name, type, isRequired));
        }
        // System.out.println("read textbox:"+name);
}

/*
 * parse selectOne tag
 */
private void parseSelectOne(Element e, Table table) throws Exception {

        String name = e.getAttribute("id");
        String type = e.getAttribute("type");
        Attr requiredAttr = e.getAttributeNode("required");
        boolean isRequired = (requiredAttr != null && requiredAttr.getValue()
                        .equals("true")) ? true : false;

        _selectCounter++;

        // single select
        if (isKey(name)) {
                table.addKeyColumn(new Column(name, type, isRequired));
        } else {
                table.addColumn(new Column(name, type, isRequired));
        }

        // System.out.println("read select:"+name);
}

/*
 * parse selectMany tag
 */
private void parseSelectMany(Element e, Table table) throws Exception {
```

```
        String name = e.getAttribute("id");
        String type = e.getAttribute("type");
        _selectCounter++;

        // multiple select
        Table multiSelectTable = new Table(_id + "_Select" + _selectCounter);
        multiSelectTable.setMainTable(_mainTable);
        multiSelectTable.addKeyColumn(new Column(name, type));
        multiSelectTable.setMultipleChoice(true);
        _tables.add(multiSelectTable);

        // System.out.println("read select:"+name);
}

/*
 * parse checkbox tag
 */
private void parseCheckbox(Element e, Table table) throws Exception {

        String name = e.getAttribute("id");
        String type = e.getAttribute("type");

        _checkboxCounter++;
        Table checkboxTable = new Table(_id + "_Checkbox" + _checkboxCounter);
        checkboxTable.setMainTable(_mainTable);
        checkboxTable.setMultipleChoice(true);
        checkboxTable.addKeyColumn(new Column(name, type));

        // System.out.println("read checkbox:"+name);

        NodeList children = e.getChildNodes();
        for (int i = 0; i < children.getLength(); i++) {
                Node child = children.item(i);
                if (child.getNodeType() == Node.ELEMENT_NODE) {
                        parseTag((Element) child, table);
                }
        }

        _tables.add(checkboxTable);

}

/*
 * parse if tag
 */
private void parseIf(Element e) throws Exception {

        _ifCounter++;

        Table ifTable = new Table(_id + "_If" + _ifCounter);
        ifTable.setMainTable(_mainTable);
        // System.out.println("read if:"+_ifCounter);

        NodeList children = e.getChildNodes();
        for (int i = 0; i < children.getLength(); i++) {
                Node child = children.item(i);
                if (child.getNodeType() == Node.ELEMENT_NODE) {
                        parseTag((Element) child, ifTable);
                }
        }
        _tables.add(ifTable);

}

/*
 * parse choose tag
 */
private void parseChoose(Element e) throws Exception {

        _chooseCounter++;
```

```
        int whenCounter = 0;
        // System.out.println("read choose:"+_chooseCounter);

        NodeList children = e.getChildNodes();
        for (int i = 0; i < children.getLength(); i++) {
                Node child = children.item(i);
                if (child.getNodeType() == Node.ELEMENT_NODE) {
                        Element eChild = (Element) child;
                        if (eChild.getTagName().equals("when")) {
                                whenCounter++;
                                parseWhen((Element) child, _chooseCounter, whenCounter);

                        } else if (eChild.getTagName().equals("otherwise")) {
                                parseOtherwise((Element) child, _chooseCounter);
                                break;
                        }
                }
        }

}

/*
 * parse when tag
 */
private void parseWhen(Element e, int chooseNo, int whenNo)
                throws Exception {

        Table chooseTable = new Table(_id + "_CHOOSE" + chooseNo + "_WHEN"
                        + whenNo);
        chooseTable.setMainTable(_mainTable);
        // System.out.println("read when:"+whenNo);

        NodeList children = e.getChildNodes();
        for (int i = 0; i < children.getLength(); i++) {
                Node child = children.item(i);
                if (child.getNodeType() == Node.ELEMENT_NODE) {
                        parseTag((Element) child, chooseTable);
                }
        }

        _tables.add(chooseTable);
}

/*
 * parse otherwise tag
 */
private void parseOtherwise(Element e, int chooseNo) throws Exception {

        Table otherwiseTable = new Table(_id + "_CHOOSE" + chooseNo
                        + "_OTHERWISE");
        otherwiseTable.setMainTable(_mainTable);
        // System.out.println("read otherwise choose:"+chooseNo);

        NodeList children = e.getChildNodes();
        for (int i = 0; i < children.getLength(); i++) {
                Node child = children.item(i);
                if (child.getNodeType() == Node.ELEMENT_NODE) {
                        parseTag((Element) child, otherwiseTable);
                }
        }

        _tables.add(otherwiseTable);
}

/*
 * check if colname is primary key of a form
 */
private boolean isKey(String name) {
        boolean containKey = false;

        for (Iterator it = _key.iterator(); it.hasNext();) {
```

```
                String key = (String) it.next();
                if (key.equals(name)) {
                        containKey = true;
                        break;
                }
        }

        return containKey;

    }

}
```

Class Table

```
package webformParser.sql;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;

public class Table {

	private ArrayList _columns = null;

	private String _name = "";

	private boolean _isMultipleChoice = false;

	private Table _mainTable = null;

	public Table(String name) {
		this._name = name;
		_columns = new ArrayList();
	}

	public void addColumn(Column col) {
		_columns.add(col);
	}

	public void addKeyColumn(Column col) {
		col.setKey(true);
		// System.out.println("Table:"+_name+" KEY:"+_columns);
		_columns.add(col);
	}

	public String getSQl() {
		String sqlText = "CREATE TABLE " + _name + " (";

		// list all columns
		for (int i = 0; i < _columns.size(); i++) {
			Column col = (Column) _columns.get(i);
			sqlText += "\n   " + col.getSQL() + ",";
		}

		sqlText += "\n   PRIMARY KEY(";
		for (int i = 0; i < _columns.size(); i++) {
			Column col = (Column) _columns.get(i);
			if (col.isKey()) {
				sqlText += col.getName() + ",";
			}
		}
		if (sqlText.endsWith(",")) {
			sqlText = sqlText.substring(0, sqlText.length() - 1);
		}
		sqlText += "),";

		// list all foreigh key
		if (_mainTable != null) {
			String mainTableName = _mainTable.getName();
			Column[] foreignKeys = _mainTable.getKey();

			// foreign key string
			String keyString = "";
			for (int i = 0; i < foreignKeys.length; i++) {
				String keyName = foreignKeys[i].getName();
				keyString += keyName + ",";
			}
			if (keyString.endsWith(",")) {
				keyString = keyString.substring(0, keyString.length() - 1);
			}

			sqlText += "\n   INDEX " + _name + "_INDX(" + keyString + "),";
```

```
                sqlText += "\n   FOREIGN KEY(" + keyString + ") REFERENCES "
                                + mainTableName + "(" + keyString + ")";
                sqlText += " ON UPDATE CASCADE ON DELETE CASCADE,";
        }

        if (sqlText.endsWith(",")) {
                sqlText = sqlText.substring(0, sqlText.length() - 1);
        }

        sqlText += "\n)TYPE=InnoDB;\n";

        return sqlText;
    }

    public String getName() {
        return _name;
    }

    public Column[] getColumns() {

        return (Column[]) _columns.toArray(new Column[_columns.size()]);
    }

    public Column[] getKey() {
        Set keys = new HashSet();

        for (Iterator it = _columns.iterator(); it.hasNext();) {
                Column col = (Column) it.next();
                if (col.isKey()) {
                        keys.add(col);
                }
        }

        return (Column[]) keys.toArray(new Column[keys.size()]);
    }

    public void setMainTable(Table table) {
        _mainTable = table;
    }

    public void setForeignKey() {
        if (_mainTable != null) {
                Column[] foreignKeys = _mainTable.getKey();
                for (int i = 0; i < foreignKeys.length; i++) {
                        foreignKeys[i].setKey(true);
                        _columns.add(0, foreignKeys[i]);
                }
        }
    }

    public boolean isMultipleChoice() {
        return _isMultipleChoice;
    }

    public void setMultipleChoice(boolean isMultipleChoice) {
        _isMultipleChoice = isMultipleChoice;
    }

}
```

Class Column

```
package webformParser.sql;

/**
 * Class Column present column in a table
 *
 * @author Piyaphol Phoungphol
 *
 */
public class Column {

	public final static String integerType = "integer";

	public final static String stringType = "string";

	public final static String doubleType = "decimal";

	private String _type = Column.stringType;

	private String _name = null;

	private boolean _isRequired = false;

	private boolean _isKey = false;

	/**
	 * construnctor with column name and data type
	 *
	 * @param name
	 * @param type
	 */
	public Column(String name, String type) {
		_name = name;
		if (type != null && !type.equals("")) {
			_type = type;
		}
	}

	/**
	 * construnctor with column name, data type, required condition
	 *
	 * @param name
	 * @param type
	 * @param isRequired
	 */
	public Column(String name, String type, boolean isRequired) {
		_name = name;
		if (type != null && !type.equals("")) {
			_type = type;
		}
		_isRequired = isRequired;
	}

	/**
	 * get name of a column
	 *
	 * @return
	 */
	public String getName() {
		return _name;
	}

	/**
	 * get SQL for this column
	 */
	public String getSQL() {
		String sqlText = "";
		if (_type.equals(Column.doubleType)) {
			sqlText = _name + "   DOUBLE";
			;
```

```
		} else if (_type.equals(Column.integerType)) {
			sqlText = _name + "   INT";
		} else {
			sqlText = _name + "   VARCHAR(255)";
		}

		if (_isKey || _isRequired) {
			sqlText += " NOT NULL";
		}

		return sqlText;
	}

	/**
	 * check if a column is primary key
	 *
	 * @return
	 */
	public boolean isKey() {
		return _isKey;
	}

	/**
	 * set a column as primary or not
	 *
	 * @param isKey
	 */
	public void setKey(boolean isKey) {
		_isKey = isKey;
	}

}
```

A.5 Package webformParser.php.report

Class PhpShowDBGenerator.java

```
package webformParser.php.report;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import webformParser.sql.Column;
import webformParser.sql.SQLScriptGenrator;
import webformParser.sql.Table;
/**
 * Class to generate php script to show database
 * @author Piyaphol Phoungphol
 *
 */
public class PhpShowDBGenerator{
        /**
         * get PHP Script String to show whole database
         * @param xmlFile
         * @return
         */
        public static String  getPHP(Document xmlForm) throws Exception{
                String phpText = "";

                Element root = xmlForm.getDocumentElement();

                if (root.getTagName().equals("webform")) {

                        String webformId = root.getAttribute("id");

                        phpText += "<?php\n\n";
                        phpText += createConnection(webformId);
                        phpText += writePhpShowData(root);
                        phpText += "mysql_close();\n";
                        phpText += "?>\n";
                }

                return phpText;

        }
        /*
         * include database connection from webform properties file
         */
        private static String createConnection(String webformId){
                String phpText ="";

                phpText += "// read all db parameters\n";
                phpText += "include '"+webformId+"Properties.php';\n\n";

                phpText += "//make a connection\n";
                phpText += "mysql_connect($dbHostname, $dbUsername, $dbPassword) or
die(mysql_error());\n";
                phpText += "mysql_select_db($dbname) or die(mysql_error());\n\n";

                return phpText;
        }
        /*
         * write SQL to show all data from database
         */
        protected static String writePhpShowData(Element e) throws Exception {

                String phpText="";
                String id =  e.getAttribute("id");

                phpText += "echo '<h2>Data Form ID: "+id+"</h2>';\n";
```

```
		SQLScriptGenrator sqlReader = new SQLScriptGenrator();
		sqlReader.parseWebForm(e);

		Table[] tables = sqlReader.getTables();

		// write main table first
		phpText += writeTable(sqlReader, tables[tables.length-1]);
		// write others table
		for(int i=0;i<tables.length-1;i++){
			phpText += writeTable(sqlReader, tables[i]);
		}

		return phpText;
	}
	/*
	 * write html table to show each table in database
	 */
	private static String writeTable(SQLScriptGenrator sqlReader, Table table){
		String phpText= "";

		String tableName = table.getName();
		phpText += "echo '<p>';\n";
		phpText += "echo '<b>Table :"+tableName+"</b>';\n";
		phpText += "echo '<table border=\"1\"><tr>';\n";
		Column[] columns = sqlReader.getColumns(tableName);

		for(int j=0;j<columns.length;j++){
			if(columns[j].isKey()){
				phpText += "echo '<th><u>"+columns[j].getName()+"</u></th>';\n";
			}else{
				phpText += "echo '<th>"+columns[j].getName()+"</th>';\n";
			}
		}

		phpText += "echo '</tr>';\n";

		phpText += "$query = 'SELECT * FROM "+tableName+"';\n";
		phpText += "$result = mysql_query($query) or die(mysql_error());\n";
		phpText += "while($row = mysql_fetch_array($result)){\n";
		phpText += "	echo '<tr>';\n";
		for(int j=0;j<columns.length;j++){
			phpText += "	if($row['"+columns[j].getName()+"']!= null){\n";
			phpText += "		echo
'<td>'.$row['"+columns[j].getName()+"'].'</td>';\n";
			phpText += "	}else{\n";
			phpText += "		echo '<td> </td>';\n";
			phpText += "	}\n";

		}
		phpText += "	echo '</tr>';\n";
		phpText += "}\n";

		phpText += "echo '</table>';\n";
		phpText += "echo '</p>';\n\n";


		return phpText;
	}

}
```

Class PhpShowReportGenerator.java

```
package webformParser.php.report;

import org.w3c.dom.Attr;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
/**
 * Class to generate php script to show webreport
 * @author Piyaphol Phoungphol
 *
 */
public class PhpShowReportGenerator{
	/*
	 * parse xml tags
	 */
	private static String parseTag(Element e) throws Exception {

		String tagName = e.getTagName();
		String phpText = "";

		if (tagName.equals("resultTable")) {
			phpText = parseResultTable(e);
		} else if (tagName.equals("resultText")) {
			phpText = parseResultText(e);
		} else {
			phpText = parseHtmlTag(e);
		}

		return phpText;
	}
	/*
	 * parse resultTable tag
	 */
	private static String parseResultTable(Element e) throws Exception {

		String phpText = "";
		String query = e.getAttribute("query");
		Attr borderAttr = e.getAttributeNode("border");
		String border = (borderAttr == null) ? "" : " border=\"="+borderAttr.getValue()+"\" ";
		Attr defualtAttr = e.getAttributeNode("default");
		String defauleValue = (defualtAttr == null) ? "" : defualtAttr.getValue();

		phpText += "\n";
		phpText += "$query = '"+query+"';\n";
		phpText += "$result = mysql_query($query) or die(mysql_error());\n";
		phpText += "$no_of_fields = mysql_num_fields ($result);\n";
		phpText += "$no_of_records = mysql_num_rows ($result);\n";
		phpText += "$j = 0;\n";

		phpText += "echo '<table "+border+">\\n<tr>';\n";
		phpText += "while ($j < $no_of_fields){\n";
		phpText += "	$fieldname = mysql_field_name ($result, $j);\n";
		phpText += "	echo \"<th>$fieldname</th>\";\n";
		phpText += "	$j++;\n";
		phpText += "}\n";
		phpText += "echo '</tr>';\n";

		phpText += "if($no_of_records==0){\n";
		phpText += "		echo \"<tr rowspan='$fieldname' align='center'>"+defauleValue+"</tr>\";\n";
		phpText += "}\n";

		phpText += "$i = 0;\n";
		phpText += "while ($i < $no_of_records) {\n";
		phpText += "	echo '<tr>';\n";
		phpText += "	$j = 0;\n";
```

```
		phpText += "	while ($j < $no_of_fields){\n";
		phpText += "		echo '<td>'.mysql_result($result,$i,$j).'</td>';\n";
		phpText += "		$j++;\n";
		phpText += "	}\n";
		phpText += "	echo '</tr>';\n";
		phpText += "	$i++;\n";
		phpText += "}\n";
		phpText += "echo '</table>';\n";
		phpText += "mysql_free_result ($result);\n\n";

		return phpText;
	}
	/*
	 * parse resultText
	 */
	private static String parseResultText(Element e) throws Exception {
		String phpText = "";
		String query = e.getAttribute("query");
		Attr defualtAttr = e.getAttributeNode("default");
		String defauleValue = (defualtAttr == null) ? "" : defualtAttr.getValue();

		phpText += "\n";
		phpText += "$query = \""+query+"\";\n";
		phpText += "$result = mysql_query($query) or die(mysql_error());\n";
		phpText += "$no_of_records = mysql_num_rows ($result);\n";

		phpText += "if ($no_of_records>0) {\n";
		phpText += "		echo mysql_result($result,0,0);\n";
		phpText += "}else{\n";
		phpText += "		echo '"+defauleValue+"';\n";
		phpText += "}\n";
		phpText += "mysql_free_result ($result);\n\n";

		return phpText;
	}
	/*
	 * parse html tag
	 */
	private static String parseHtmlTag(Element e) throws Exception {
		String phpText = "";
		String tagName = e.getTagName();

		phpText += "echo '<" + tagName;

		// Get all the attributes
		NamedNodeMap attrs = e.getAttributes();
		for (int i = 0; i < attrs.getLength(); i++) {
			Attr attr = (Attr) attrs.item(i);

			String attrName = attr.getNodeName();
			String attrValue = attr.getNodeValue();

			phpText += " " + attrName + "=\"" + attrValue + "\"";
		}
		phpText += ">';\n";

		NodeList children = e.getChildNodes();
		for (int i = 0; i < children.getLength(); i++) {
			Node child = children.item(i);
			if (child.getNodeType() == Node.TEXT_NODE) {
				String text = child.getNodeValue().replaceAll(
								"(\\n|\\s{2}|\\t)", "");
				if (!text.equals("")) {
					phpText += "echo '"+text+"';\n";
				}
			} else if (child.getNodeType() == Node.ELEMENT_NODE) {
				phpText += parseTag((Element) child);
			}
		}
```

```
		phpText += "echo '</" + tagName + ">';\n";

		return phpText;
	}
	/**
	 * get PHP Script String to show webreport from xmlDocument
	 * @param xmlFile
	 * @return
	 */
	public static String getPHP(Document reportXML) throws Exception{
		String phpText = "";

		Element root = reportXML.getDocumentElement();

		if (root.getTagName().equals("webreport")) {

			String webformId = root.getAttribute("webform");

			phpText += "<?php\n\n";
			phpText += createConnection(webformId);
			phpText += parseTag(root);
			phpText += "mysql_close();\n";
			phpText += "?>\n";

		}

		return phpText;

	}
	/*
	 * include database connection from webform properties file
	 */
	private static String createConnection(String webformId){
		String phpText ="";

		phpText += "// read all db parameters\n";
		phpText += "include '"+webformId+"Properties.php';\n\n";

		phpText += "//make a connection\n";
		phpText += "mysql_connect($dbHostname, $dbUsername, $dbPassword) or die(mysql_error());\n";
		phpText += "mysql_select_db($dbname) or die(mysql_error());\n\n";

		return phpText;
	}
}
```

B. LEXICAL SPECIFICATION AND GRAMMAR OF SWFL

B.1 Lexical Specification

webform.flex

```
package webformParser.simpleParser;

import java_cup.runtime.*;

%%

%class Lexer
%unicode
%cup
%line
%column

%{
  StringBuffer s = new StringBuffer();

  private Symbol symbol(int type) {
    return new Symbol(type, yyline, yycolumn);
  }

  private Symbol symbol(int type, Object value) {
    return new Symbol(type, yyline, yycolumn, value);
  }
%}

LineTerminator = \r|\n|\r\n
InputCharacter = [^\r\n]
WhiteSpace     = {LineTerminator} | [ \t\f]

/* comments */
Comment = {TraditionalComment} | {EndOfLineComment} | {DocumentationComment}

TraditionalComment   = "/*" [^*] ~"*/" | "/*" "*"+ "/"
EndOfLineComment     = "//" {InputCharacter}* {LineTerminator}
DocumentationComment = "/**" {CommentContent} "*"+ "/"
CommentContent       = ( [^*] | \*+ [^/*] )*


DecIntegerLiteral = 0 | [1-9][0-9]*

Compare_operator = ">="  | ">" | "=" | "<=" | "<" | "!=" | "<>"

Operator       = "+"  | "-" | "*" | "/" | "mod"

Binary_operator        = "and" | "or"

Data_type      = "string" | "integer" | "decimal"

Heading        = "heading" [1-6]

Identifier = [:jletter:] [:jletterdigit:]*

%state STRING

%%

/* keywords */
<YYINITIAL> "create"                 { return symbol(sym.CREATE); }
```

```
<YYINITIAL> "webform"                { return symbol(sym.WEBFORM); }
<YYINITIAL> "webreport"              { return symbol(sym.WEBREPORT); }
<YYINITIAL> "not"                    { return symbol(sym.NOT); }
<YYINITIAL> "null"                    { return symbol(sym.NULL); }
<YYINITIAL> "key"                     { return symbol(sym.KEY); }
<YYINITIAL> "type"                    { return symbol(sym.TYPE); }
<YYINITIAL> "condition"               { return symbol(sym.CONDITION); }
<YYINITIAL> "hide"                    { return symbol(sym.HIDE); }
<YYINITIAL> "textField"               { return symbol(sym.TEXTFIELD); }
<YYINITIAL> "textArea"                { return symbol(sym.TEXTAREA); }
<YYINITIAL> "password"                { return symbol(sym.PASSWORD); }
<YYINITIAL> "selectOne"               { return symbol(sym.SELECTONE); }
<YYINITIAL> "selectMany"              { return symbol(sym.SELECTMANY); }
<YYINITIAL> "radio"                   { return symbol(sym.RADIO); }
<YYINITIAL> "checkbox"                { return symbol(sym.CHECKBOX); }
<YYINITIAL> "submit"                  { return symbol(sym.SUBMIT); }
<YYINITIAL> "reset"                   { return symbol(sym.RESET); }
<YYINITIAL> "("                       { return symbol(sym.LPAREN); }
<YYINITIAL> ")"                       { return symbol(sym.RPAREN); }
<YYINITIAL> ","                        { return symbol(sym.COMMA); }
<YYINITIAL> "selected"                { return symbol(sym.SELECTED); }
<YYINITIAL> "pattern"                 { return symbol(sym.PATTERN); }
<YYINITIAL> "label"                   { return symbol(sym.LABEL); }
<YYINITIAL> "text"                    { return symbol(sym.TEXT); }
<YYINITIAL> "title"                   { return symbol(sym.TITLE); }
<YYINITIAL> "resultText"               { return symbol(sym.RESULTTEXT); }
<YYINITIAL> "resultTable"              { return symbol(sym.RESULTTABLE); }
<YYINITIAL> "of"                      { return symbol(sym.OF); }

<YYINITIAL> {

  /* literals */
  {DecIntegerLiteral}         { return symbol(sym.NUMBER, yytext() ); }

  \"                          { s.setLength(0); yybegin(STRING); }

  {Binary_operator}           { return symbol(sym.BINARY_OPERATOR, yytext() ); }

  /* operators */
  {Compare_operator}          { return symbol(sym.COMPARE_OPERATOR, yytext() ); }


  {Operator}                  { return symbol(sym.OPERATOR, yytext() ); }

  /* data type */
  {Data_type}                 { return symbol(sym.DATATYPE, yytext() ); }

  {Heading}                   { return symbol(sym.HEADING, yytext() ); }

  /* comments */
  {Comment}                   { /* ignore */ }

  /* whitespace */
  {WhiteSpace}                { /* ignore */ }

  /* identifiers */
  {Identifier}                { return symbol(sym.NAME, yytext() ); }
}

<STRING> {
  \"                          { yybegin(YYINITIAL);
                                    return symbol(sym.STRING, s.toString()); }
  [^\"\\]+                    { s.append( yytext() ); }
  \\\"                        { s.append('\"'); }
  \\                          { s.append('\\'); }


}

/* error fallback */
.|\n                          { throw new Error("Illegal character <"+
```

```
yytext()+">"); }
```

B.2 SWFL Grammar

webform.cup

```
package webformParser.simpleParser;

import java_cup.runtime.*;
import java.util.*;
import org.w3c.dom.*;
import javax.xml.parsers.*;

action code {:

        private Set _keySet = new HashSet();

        private Map _conditionMap = new HashMap();

        private Document _doc = createNewDoc();

        private String _title = null;

        private String _submitCondition = null;


        private Document createNewDoc() {

                Document doc = null;

                try{
                        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
                        DocumentBuilder builder = factory.newDocumentBuilder();
                        doc = builder.newDocument();

                } catch (Exception e){
                        e.printStackTrace();
                }

                return doc;
        }


        private Element showLableAndText(Element element){

                NodeList children = element.getChildNodes();
                for (int i = 0; i < children.getLength(); i++) {
                        Node childNode = children.item(i);
                        if(childNode.getNodeType()== Node.ELEMENT_NODE) {
                                Element childElement = (Element) childNode;
                                if(childElement.hasAttribute("label")){

                                        String label = childElement.getAttribute("label");

        childElement.getParentNode().insertBefore(_doc.createTextNode(label),childElement);
                                        childElement.removeAttribute("label");


                                } else if(childElement.getTagName().equals("text") ){

                                        String text = childElement.getAttribute("value");
                                        Node textNode = _doc.createTextNode(text);
                                        childElement.getParentNode().replaceChild(textNode,
childElement);

                                } else {
                                        childElement = showLableAndText(childElement);

                                }
```

```
                    }
            }

            return element;

        }


:};

parser code {:

        public void report_error(String message, Object info) {

            /* Create a StringBuffer called 'm' with the string 'Error' in it. */
            StringBuffer m = new StringBuffer("Error");

            /* Check if the information passed to the method is the same
               type as the type java_cup.runtime.Symbol. */
            if (info instanceof java_cup.runtime.Symbol) {
                /* Declare a java_cup.runtime.Symbol object 's' with the
                   information in the object info that is being typecasted
                   as a java_cup.runtime.Symbol object. */
                java_cup.runtime.Symbol s = ((java_cup.runtime.Symbol) info);

                /* Check if the line number in the input is greater or
                   equal to zero. */
                if (s.left >= 0) {
                    /* Add to the end of the StringBuffer error message
                       the line number of the error in the input. */
                    m.append(" in line "+(s.left+1));
                    /* Check if the column number in the input is greater
                       or equal to zero. */
                    if (s.right >= 0)
                        /* Add to the end of the StringBuffer error message
                           the column number of the error in the input. */
                        m.append(", column "+(s.right+1));
                }
            }

            /* Add to the end of the StringBuffer error message created in
               this method the message that was passed into this method. */
            m.append(" : "+message);

            /* Print the contents of the StringBuffer 'm', which contains
               an error message, out on a line. */
            System.err.println(m);
        }

        // method  to report fatal_error
        public void report_fatal_error(String message, Object info) {
            report_error(message, info);
            System.exit(1);
        }

:};



/* Terminals (tokens returned by the scanner). */
terminal String     LPAREN, RPAREN, COMMA, COMPARE_OPERATOR, BINARY_OPERATOR, OPERATOR ;
terminal String            CONDITION, NAME, NUMBER, STRING, DATATYPE, HIDE, TYPE;
terminal String            TEXT, HEADING, TITLE, OF;
terminal String            CREATE, WEBFORM, WEBREPORT, TEXTFIELD, TEXTAREA, PASSWORD, RADIO;
terminal String            SELECTONE, SELECTMANY, CHECKBOX, SUBMIT, RESET;
terminal String            SELECTED, PATTERN, NOT, NULL, KEY, LABEL;
terminal String            RESULTTEXT, RESULTTABLE ;


/* Non terminals */
```

```
non terminal String    condition_def, cond_term, hide_condition, key_condition, submit_codition,
choice_require ;
non terminal String    expr, term, cond_expr ;
non terminal Document          webform_doc, webform_desc, webreport_desc ;
non terminal Element           statement, elem_def, named_elem_def, unnamed_element_def ;
non terminal Element           reset_def,submit_def, textfield_def, radio_def, checkbox_def ;
non terminal Element   selectone_def, selectmany_def, heading_def, text_def, title_def;
non terminal Element   textfield, textbox, textarea,password, radio, checkbox, selectone,
selectmany ;
non terminal Element   report_element, ruseltText_element, resultTable_element;
non terminal List      statement_lists, string_list, id_list, report_lists;
non terminal Map       textfield_options, textfield_option;
non terminal Map       single_choice_options, single_choice_option;
non terminal Map       multi_choice_options, multi_choice_option;


/* The grammar */

webform_doc ::= webform_desc:document
                       {:
                              RESULT = document;
                       :}
               | webreport_desc:document
                       {:
                              RESULT = document;
                       :}
               ;

webreport_desc ::= CREATE WEBREPORT NAME:webReportId OF WEBFORM NAME:webFormId LPAREN
report_lists:list RPAREN

                       {:
                              Element webReport = _doc.createElement("webreport");
                              webReport.setAttribute("id",webReportId);
                              webReport.setAttribute("webform",webFormId);

                              if(_title!=null){
                                     webReport.setAttribute("title",_title);
                              }

                              /* add all element to web form */
                              for(int i=0;i<list.size();i++){
                                     Element element = (Element) list.get(i);
                                     webReport.appendChild(element);

                                     if(!element.getTagName().matches("h[123456]")){
                                            webReport.appendChild(_doc.createElement("br"));
                                     }

                              }
                              _doc.appendChild(webReport);

                              RESULT = _doc;

                       :}
               ;


report_lists ::= report_lists:list COMMA report_element:element
                              {:
                                     if(element!= null){
                                            list.add(element);
                                     }

                                     RESULT = list;
                              :}
                       |report_element:element
                              {:
                                     List list = new ArrayList();
                                     if(element!= null){
                                            list.add(element);
```

```
                                        }
                                        RESULT = list;
                                :}
                ;


report_element ::= ruseltText_element:element
                                {: RESULT = element; :}
                        | resultTable_element:element
                                {: RESULT = element; :}
                        | heading_def:element
                                {: RESULT = element; :}
                        | text_def:element
                                {: RESULT = element; :}
                        | title_def
                ;

ruseltText_element ::= RESULTTEXT LPAREN STRING:s RPAREN
                        {:
                                Element resultText = _doc.createElement("resultText");
                                resultText.setAttribute("query",s);

                                RESULT = resultText;
                        :}
                        ;


resultTable_element ::= RESULTTABLE LPAREN STRING:s RPAREN
                        {:
                                Element resultTable = _doc.createElement("resultTable");
                                resultTable.setAttribute("query",s);

                                RESULT = resultTable;
                        :}
                        ;


webform_desc ::= CREATE WEBFORM NAME:id LPAREN statement_lists:list RPAREN
                        {:

                                Element webForm = _doc.createElement("webform");
                                webForm.setAttribute("id",id);

                                if(_title!=null){
                                        webForm.setAttribute("title",_title);
                                }


                                /* add key attribute */
                                String keyString = "";
                                for (Iterator it=_keySet.iterator(); it.hasNext(); ) {
                                        String key = (String)it.next();
                                        keyString += key+",";
                                }
                                if(keyString.endsWith(",")){
                                        keyString = keyString.substring(0, keyString.length()-1);
                                }

                                if(!keyString.equals("")){
                                        webForm.setAttribute("key", keyString);
                                }


                                /* add all element to web form */
                                for(int i=0;i<list.size();i++){
                                        Element element = (Element) list.get(i);
                                        webForm.appendChild(element);

                                        if(!element.getTagName().matches("h[123456]")){
                                                webForm.appendChild(_doc.createElement("br"));
                                        }
```

```
                                }
                                _doc.appendChild(webForm);


                                /* add submit condition */
                                NodeList submitList=_doc.getElementsByTagName("submit");
                                if( submitList.getLength() >= 0 && _submitCondition!=null){
                                        ((Element)submitList.item(0)).setAttribute("test",
_submitCondition);
                                }


                                /* apply condition */
                                try{
                                        for (Iterator it=_conditionMap.keySet().iterator();
it.hasNext(); ) {
                                                String expKey = (String)it.next();
                                                List id_list = (List)_conditionMap.get(expKey);

                                                for (Iterator idIterator=id_list.iterator();
idIterator.hasNext(); ) {

                                                        String elementId =
(String)idIterator.next();
                                                        Element element =
_doc.getElementById(elementId);

                                                        Element ifElement =
_doc.createElement("if");

        ifElement.setAttribute("test","!("+expKey+")");


        element.getParentNode().replaceChild(ifElement, element);
                                                        ifElement.appendChild(element);

        ifElement.appendChild(_doc.createElement("br"));


        ifElement.getParentNode().removeChild(ifElement.getNextSibling());

                                                }

                                        }

                                } catch(Exception e){
                                        e.printStackTrace();
                                }

                                /* modify text node and label of element */
                                webForm = showLableAndText(webForm);

                                RESULT = _doc;
                        :}
                        ;

statement_lists ::=     statement_lists:list COMMA statement:element
                                {:
                                        if(element!= null){
                                                list.add(element);
                                        }

                                        RESULT = list;
                                :}
                        |statement:element
                                {:
                                        List list = new ArrayList();
                                        if(element!= null){
                                                list.add(element);
                                        }
```

```
                                        RESULT = list;
                                :}
                        ;

statement ::=  elem_def:element
                        {: RESULT = element; :}

               | condition_def
                        {: RESULT = null; :}
               ;

condition_def ::=  hide_condition | key_condition | submit_codition;


hide_condition ::= HIDE LPAREN id_list:elementList RPAREN CONDITION LPAREN cond_expr:exp RPAREN
                        {:
                                _conditionMap.put(exp,elementList);

                        :}
                  ;

key_condition ::=  KEY LPAREN id_list:elementList RPAREN
                                {:
                                        _keySet.addAll(elementList);
                                :}
                   ;

submit_codition ::= SUBMIT CONDITION LPAREN cond_expr:exp RPAREN
                                {:
                                        _submitCondition = exp;
                                :}
                        ;



elem_def ::=   NAME:id named_elem_def:element
                        {:
                                element.setAttribute("id",id);
                                element.setIdAttribute("id", true);

                                if(!element.getAttribute("key").equals("")){

                                        _keySet.add(id);
                                        element.removeAttribute("key");

                                }

                                RESULT = element;
                        :}
               | unnamed_element_def: element
                        {: RESULT = element;  :}
               ;

unnamed_element_def ::= submit_def:element
                                        {: RESULT = element; :}
                                | reset_def:element
                                        {: RESULT = element; :}
                                | heading_def:element
                                        {: RESULT = element; :}
                                | text_def:element
                                        {: RESULT = element; :}
                                | title_def
                                ;

heading_def ::=        HEADING:heading LPAREN STRING:s RPAREN
                                {:
                                        String headingSize = heading.substring(7);
                                        Element headingElement =
_doc.createElement("h"+headingSize);
                                        headingElement.appendChild(_doc.createTextNode(s));
```

```
                                        RESULT = headingElement;
                                :}
                        ;

text_def ::=            TEXT LPAREN STRING:s RPAREN

                                {:      Element text = _doc.createElement("text");
                                        text.setAttribute("value",s);

                                        RESULT = text;
                                :}
                        ;

title_def ::=           TITLE LPAREN STRING:s RPAREN

                                {:
                                        _title = s;
                                :}
                        ;


named_elem_def ::=      textfield_def:element
                                {: RESULT = element; :}
                        | selectone_def:element
                                {: RESULT = element; :}
                        | selectmany_def:element
                                {: RESULT = element; :}
                        | radio_def:element
                                {: RESULT = element; :}
                        | checkbox_def:element
                                {: RESULT = element; :}
                        ;

reset_def ::=  RESET
                        {: RESULT = _doc.createElement("reset"); :}

                | RESET LPAREN STRING:s RPAREN
                        {:
                                Element reset = _doc.createElement("reset");
                                reset.setAttribute("value", s);

                                RESULT = reset;
                        :}
                ;

submit_def ::=          SUBMIT
                        {: RESULT = _doc.createElement("submit"); :}

                | SUBMIT LPAREN STRING:s RPAREN
                        {:
                                Element submit = _doc.createElement("submit");
                                submit.setAttribute("value", s);

                                RESULT = submit;
                        :}
                ;


textfield_def ::= textfield:element
                        {: RESULT = element;   :}

                | textfield:element textfield_options:attrMap
                        {:

                                for (Iterator it=attrMap.keySet().iterator(); it.hasNext(); ) {
                                        String key = (String)it.next();
                                         String value = (String)attrMap.get(key);
                                         element.setAttribute(key, value);
                                }

                                RESULT = element;
```

```
                        :}
                ;

textfield ::=   textbox:element
                        {: RESULT = element; :}
                | textarea:element
                        {: RESULT = element; :}
                | password:element
                        {: RESULT = element; :}
                ;

textbox ::=     TEXTFIELD
                {: RESULT = _doc.createElement("textField"); :}

                | TEXTFIELD LPAREN NUMBER:maxlength RPAREN
                {:

                        Element element = _doc.createElement("textField");
                        element.setAttribute("maxlength",maxlength);

                        RESULT = element;
                :}
                ;

textarea ::=    TEXTAREA
                        {: RESULT = _doc.createElement("textArea"); :}


                | TEXTAREA LPAREN NUMBER:row RPAREN
                        {:
                                Element element = _doc.createElement("textArea");
                                element.setAttribute("rows",row);

                                RESULT = element;
                        :}
                | TEXTAREA LPAREN NUMBER:row COMMA NUMBER:col RPAREN
                        {:
                                Element element = _doc.createElement("textArea");
                                element.setAttribute("rows",row);
                                element.setAttribute("cols",col);

                                RESULT = element;
                        :}
                ;

password ::=    PASSWORD
                        {: RESULT = _doc.createElement("password"); :}

                | PASSWORD LPAREN NUMBER:maxlength RPAREN
                        {:
                                Element element = _doc.createElement("password");
                                element.setAttribute("maxlength",maxlength);

                                RESULT = element;
                        :}
                ;


radio_def ::= radio:element
                        {: RESULT = element; :}
                | radio:element single_choice_options:attrMap
                        {:
                                for (Iterator it=attrMap.keySet().iterator(); it.hasNext(); ) {
                                        String key = (String)it.next();
                                         String value = (String)attrMap.get(key);
                                         element.setAttribute(key, value);
                                }

                                RESULT = element;
                        :}
                ;
```

```
radio ::=      RADIO LPAREN string_list:optionList RPAREN
                       {:
                               Element radiobox = _doc.createElement("radio");
                               for (Iterator it=optionList.iterator(); it.hasNext(); ) {
                                       String optionLabel = (String)it.next();

                                       Element radioOption = _doc.createElement("optionItem");
                                       radioOption.setAttribute("value", optionLabel);

                                       radiobox.appendChild(radioOption);

                               }

                               RESULT = radiobox;
                       :}
               ;

selectone_def ::=      selectone:element
                       {: RESULT = element; :}

               | selectone:element single_choice_options:attrMap
                       {:
                               for (Iterator it=attrMap.keySet().iterator(); it.hasNext(); ) {
                                       String key = (String)it.next();
                                        String value = (String)attrMap.get(key);
                                        element.setAttribute(key, value);
                               }

                               RESULT = element;
                       :}
               ;


selectone ::=  SELECTONE LPAREN string_list:optionList RPAREN
                       {:
                               Element selectList = _doc.createElement("selectOne");
                               for (Iterator it=optionList.iterator(); it.hasNext(); ) {
                                       String value = (String)it.next();

                                       Element selectOption = _doc.createElement("optionItem");
                                       selectOption.setAttribute("value", value);

                                       selectList.appendChild(selectOption);

                               }
                               RESULT = selectList;

                       :}
               ;

selectmany_def ::= selectmany:element
                       {: RESULT = element; :}

               | selectmany:element multi_choice_options:attrMap
                       {:
                               for (Iterator it=attrMap.keySet().iterator(); it.hasNext(); ) {
                                       String key = (String)it.next();
                                       String value = (String)attrMap.get(key);
                                        element.setAttribute(key, value);
                               }

                               RESULT = element;
                       :}
               ;

selectmany ::=   SELECTMANY LPAREN string_list:optionList RPAREN
                       {:
                               Element selectList = _doc.createElement("selectMany");
                               for (Iterator it=optionList.iterator(); it.hasNext(); ) {
```

```
                                        String optionLabel = (String)it.next();

                                        Element selectOption = _doc.createElement("optionItem");
                                        selectOption.setAttribute("value", optionLabel);

                                        selectList.appendChild(selectOption);

                                }

                                selectList.setAttribute("multiple","true");

                                RESULT = selectList;

                        :}
                ;



checkbox_def::= checkbox:element
                        {: RESULT = element; :}
                | checkbox:element multi_choice_options:attrMap
                        {:
                                for (Iterator it=attrMap.keySet().iterator(); it.hasNext(); ) {
                                        String key = (String)it.next();
                                         String value = (String)attrMap.get(key);
                                         element.setAttribute(key, value);
                                }

                                RESULT = element;
                        :}
                ;


checkbox ::=   CHECKBOX LPAREN string_list:optionList RPAREN
                        {:
                                Element checkbox = _doc.createElement("checkbox");
                                for (Iterator it=optionList.iterator(); it.hasNext(); ) {
                                        String optionLabel = (String)it.next();

                                        Element checkboxOption = _doc.createElement("optionItem");
                                        checkboxOption.setAttribute("value", optionLabel);

                                        checkbox.appendChild(checkboxOption);

                                }

                                RESULT = checkbox;

                        :}
                ;



textfield_options ::=  textfield_option:optionMap
                                {: RESULT = optionMap; :}

                        | textfield_options:attrMap textfield_option:optionMap
                                {:
                                        attrMap.putAll(optionMap);
                                        RESULT = attrMap;
                                :}
                        ;


textfield_option ::=   NOT NULL
                                {:
                                        Map optionMap = new HashMap();
                                        optionMap.put("required", "true");

                                        RESULT = optionMap;
                                :}
```

```
                        | KEY
                                {:
                                        Map optionMap = new HashMap();
                                        optionMap.put("key", "key");

                                        RESULT = optionMap;
                                :}
                        | PATTERN LPAREN STRING:regx RPAREN
                                {:
                                        Map optionMap = new HashMap();
                                        optionMap.put("regx", regx);

                                        RESULT = optionMap;
                                :}
                        | TYPE LPAREN DATATYPE:s RPAREN
                                {:
                                        Map optionMap = new HashMap();
                                        optionMap.put("type", s);

                                        RESULT = optionMap;
                                :}

                        | LABEL LPAREN STRING:s RPAREN
                                {:
                                        Map optionMap = new HashMap();
                                        optionMap.put("label", s);

                                        RESULT = optionMap;
                                :}

                        ;


string_list ::=         string_list:list COMMA STRING:s
                        {:
                                list.add(s);
                                RESULT = list;
                        :}
                |STRING:s
                        {:
                                List list = new ArrayList();
                                list.add(s);

                                RESULT = list;
                        :}
                ;

id_list ::=     id_list:list COMMA NAME:id
                        {:
                                list.add(id);
                                RESULT = list;
                        :}
                |NAME:id
                        {:
                                List list = new ArrayList();
                                list.add(id);

                                RESULT = list;
                        :}

                ;


single_choice_options ::= single_choice_option:optionMap
                                {: RESULT = optionMap; :}

                          | single_choice_options:attrMap single_choice_option:optionMap
                                {:
                                        attrMap.putAll(optionMap);
                                        RESULT = attrMap;
```

```
                                :}

                          ;

single_choice_option ::= NOT NULL
                                {:
                                        Map optionMap = new HashMap();
                                        optionMap.put("required", "true");

                                        RESULT = optionMap;
                                :}


                        | TYPE LPAREN DATATYPE:s RPAREN
                                {:
                                        Map optionMap = new HashMap();
                                        optionMap.put("type", s);

                                        RESULT = optionMap;
                                :}
                        | KEY
                                {:
                                        Map optionMap = new HashMap();
                                        optionMap.put("key", "key");

                                        RESULT = optionMap;
                                :}
                         | LABEL LPAREN STRING:s RPAREN
                                {:
                                        Map optionMap = new HashMap();
                                        optionMap.put("label", s);

                                        RESULT = optionMap;
                                :}

                        ;

multi_choice_options ::= multi_choice_option:optionMap

                                {: RESULT = optionMap; :}

                          | multi_choice_options:attrMap multi_choice_option:optionMap
                                {:
                                        attrMap.putAll(optionMap);
                                        RESULT = attrMap;
                                :}
                          ;



multi_choice_option ::= TYPE LPAREN DATATYPE:s RPAREN
                                {:
                                        Map optionMap = new HashMap();
                                        optionMap.put("type", s);

                                        RESULT = optionMap;
                                :}

                        | NOT NULL
                                {:
                                        Map optionMap = new HashMap();
                                        optionMap.put("required", ">=1");

                                        RESULT = optionMap;
                                :}



                        | SELECTED LPAREN choice_require:s RPAREN
                                {:
                                        Map optionMap = new HashMap();
```

```
                                        optionMap.put("required", s);

                                        RESULT = optionMap;
                                :}
                        | KEY
                                {:
                                        Map optionMap = new HashMap();
                                        optionMap.put("key", "key");

                                        RESULT = optionMap;
                                :}
                        | LABEL LPAREN STRING:s RPAREN
                                {:
                                        Map optionMap = new HashMap();
                                        optionMap.put("label", s);

                                        RESULT = optionMap;
                                :}

                        ;

choice_require ::=      NUMBER:number
                                {: RESULT = number; :}

                        |COMPARE_OPERATOR:op NUMBER:number
                                {: RESULT = op+number; :}
                        ;


expr ::=        expr:l OPERATOR:op term:r
                         {: RESULT = l+op+r; :}

                  | term:t
                        {: RESULT = t ; :}
                  ;

term ::=   LPAREN expr:exp RPAREN
               {: RESULT = "("+exp+")"; :}
             |
             NUMBER:n
               {: RESULT = n; :}
             |
             NAME:n
               {: RESULT = n; :}
             |
             STRING:s
               {: RESULT = "'"+s+"'"; :}
             ;

cond_expr ::=  cond_expr:l BINARY_OPERATOR:op cond_term:r
                        {: RESULT = l+op+ r; :}
               | cond_term:t
                        {: RESULT = t; :}
                         ;

cond_term ::= expr:l COMPARE_OPERATOR:op expr:r
                        {:
                                if(op.equals("=")){
                                        op = op+"=";
                                }else if (op.equals("<>")){
                                        op = "!=";
                                }

                                RESULT = l+op+r;
                        :}

              ;
```