Smart Forms

A survey to state and test the most major electronic forms technologies that are based on W3C standards

Submitted by
Radiya Abolkhairat
0050107610

Supervised by
Dr Stijn Dekeyser

A thesis submitted for the degree of
Master of Computing Technology
In major(Software and web)

Department of Mathematics and Computing
Faculty of science
The University of Southern Queensland, Toowoomba, QLD

June 17, 2013

Acknowledgments

I would like to thank my supervisor, Dr. Stijn Dekeyser, for his guidance, encouragement, support and patience over the difficult conditions that I had to complete this research.

I also would like to say thank you to Dr. Richard Watson for his help and recommendations.

Finally, I would like to thank my husband for his support, care and love.

Abstract

Smart Forms are efficient and powerful electronic forms that could be used for the interactions between end users and web applications systems. Several electronic forms software products that use W3C technologies are presented to meet the demands of users. This thesis aims to study and test the major electronic forms technologies that are based on W3C standards. It discusses the main electronic forms features and experiments them with some essential applications. This research produces deep understanding of the most electronic forms technologies that are based on W3C standards and their important features, which make an electronic form smart form. In addition, it opens developments prospects for other researchers to develop some applications ideas that could contribute in the electronic forms domain.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

SmartForm is a dynamic electronic form that has powerful electronic features such as connecting to database, calculation and validation. The electronic form is a digital version of the paper form. Electronic forms present many advantages that addressed problems with using paper forms such as:

- The automatic validation and calculation with electronic forms makes entered data accurate and consistence. Using electronic forms reduce the chance of having errors and fast filling data.
- Routing paper forms from one person to another could be costly and time-consuming because it requires for several business processes arrangements. Electronic forms can automate a dynamic workflow for routing data form.
- Electronic forms can be connected to a network (Internat or Internet) that make the update of the forms easy and fast.
- Electronic forms use different technologies for security that make them safer for sensitive information than paper forms.

Different technologies are presented to improve electronic forms with useful features such Portable Document Format (PDF), word processors, spreadsheets forms, XForms and HTML Forms. PDF electronic forms are based on International Standard Organization (ISO) specifications and include data, logic and presentation information to be rendered as the same look on papers [5]. Word processors and spreadsheets are Standard office applications that can be used for creating electronic forms with limited features [5]. The main objective of this thesis is producing a deep study of the major electronic forms technologies and applications that are based on World Wide Web Consortium (W3C) standards such as XForms and HTML5 Forms.

Thesis structure: This thesis starts by providing background about smart forms and the technologies that are included and excluded in the study. Chapter 2 describes the most major features that will be tested later to check their availability and support by electronic forms technologies and applications. Chapter 3 reviews the primary technologies that presented by W3C standards, and discusses the related researches that conducted to develop or study electronic forms technologies. Chapter 4 describes some electronic forms applications, experiments and assesses their support for the presented features in chapter 2. Chapter 5 presents a summary that compare the functionality and form features of the experimented applications which introduced in chapter 4. The last chapter provides a conclusion and suggestions for future work.

Chapter 2

Major electronic forms features

Smart form is a part of a web application. Web application is a program that resides on the server and transported to the client across the network and displayed using a web browser. There are two kinds of web applications: client and server sides. Server-side script handles how the information could be stored or retrieved. Client-side script handles how the information could be displayed to the user. Electronic forms have many different features that make them smart and help the user to deal with data in easy, accurate and secure way.
Client-side script is used on the user's computer and executed by the web browser. When the server sends a requested web page to the user, that page would be displayed by using any scripts on the client-side. Client-side scripting is mainly used for appearance and interaction such as JavaScript. It can be viewed and copied by the user [2].
Server-side script cannot be viewed or copied by the user because it runs on the server. When the client sends a request for a web page to the server, the server will execute the script for that page and sends the results to the client to display the page. PHP is an example of server-side scripts [2]. This chapter explains the most major features of electronic forms with their client-side and server-side implementations.

2.1 Electronic forms supported features

2.1.1 Data Validation

Validation is to check the quality of the entered data in a form fields and determine if it is compatible with specific standards and rules. There are several different types of data validation such as:

- Existence validation: This check is used for required fields in a form

to make sure they are not left empty.

- Range validation: This check is to make sure that the entered data in a field is in a specific range.
- Data type validation: To check if the entered data is the correct type. For instance, to make sure there is no numbers are entered in the name field.
- Field length validation: To ensure the size of numbers or characters is entered. For example, to make sure that the entered name is not too long.
- Digit validation: To check if a code of series of numbers or characters is correct. This validation uses an extra digit append to the code and generated from the code itself using an algorithm. Examples of this validation are bank account number and International Standard Book Numbers (ISBN) validations [3] [17].
- Form completion validation: This validation is to check if a form is completed by using a formula set by the forms developer. For Example, if a form has a check box that determines if it is completed or not , the form completion validation would be by checking the value of that check box [14] .

Client-side validation

When the user enters data in a form, the input date would be validated by executing a script which is submitted to the client with the web page. If the script detects any error, an error state is displayed to the client. The input data is sent to the server if no error is detected. [39]

Server-side validation

When the user enters data in a form, the input data is sent to the server to be validated. If the server-side validation script detects any errors, an error state is sent to the client. The input data is transmitted to the next process if no error is detected. [39]

Client-side VS server-side validation

Server-side validation is more secure and reliable than client-side because the latter one is easy to be attacked, injected with other harm codes or stopped by disabling the javascript code. [46]

Using only the server-side implantations fix the problem and it is enough for

validation, but because it is harder to setup and deal with than client-side implementations. Moreover, the client-side one helps to reduce round trips to the server. So, it seems that the good solution for this problem is to use both of them. [25]

2.1.2 Dynamic form modification

The purpose of a form determines the number of fields. So, it is better to give the user the ability to modify the form such as adding new fields and remove them. For example, sometimes users need to add more than one phone number. The new form modification (including, adding, or hiding fields) would be created as a new node and then add it to the DOM. Any modifications could be removed from the form using Dom functions as well. [9].

client-side

The modification of the form is executed on client-side and then sending these modifications with data after validation to the server.

server-side

The sending modifications with data are validated to check if they meet specific requirements. For example, the server determines if the number of adding modification is not more than expected. If so, data is not stored and an error message is sent to the client.

2.1.3 Calculation feature

This feature is used to compute a form field based on a specific formula. This formula uses form fields or other external values. The formula is executed when the user finishes filling out fields and submit the form. Calculation could be written in different ways. For instance, a calculation may be performed when a certain condition is met [56]. Calculation may be performed on different types of data. For example: [37]

- Date calculation: For example, a form calculates the date of each sale in an organization. So, the sale representatives can perform their activities before or after that date.
- Form serialization calculation: For example: invoices, receipts and forms that have a unique reference number. A calculation formula provides the form with a unique number (invoice, receipt or reference number) based on previous stored form numbers.

- Numrical calculation: This calculation performs mathematical operations between numbers from form fields values or other external values. For example, calculating the total price in an invoice form.

client and server side calculation

This feature may be implemented on either client or server side. Calculation on the client is quicker, where there is no need for an internet connection to do it. However, it is not secure because the script could be turned off and the calculation may not be executed at all.

2.1.4 Access right control

Access right control refers to a set of security information that enables users from access to resources depending on their permissions and roles. Permissions are provided to users on various levels of access that depend on their roles or jobs. For example, a profile form gives the user a table-level access to store data at the first time of filling out the form, and then the same user is given a row-level access to the form to update or delete details from his/her profile.

Reference Monitor (RM)

Access control policy (rules that is taken on by a system to control access to a resource) is implemented on the ability of users to perform operations on information in a system. This access control policy is based on a set of requirements on a control validation technique called Reference Monitors (RM) [28]. Based on the residence of reference monitor, there are two kinds of access control models: client-side and server-side [59].

- Client-side RM
  In this model, Client Reference Monitor (CRM) is located in the client‘s system and controls access to and usage of data and information. Data can be either stored centrally or locally. Data cannot be customized for specific users. However, users still need to prove that they have access to limited rights to the information on the client. [59] [64]

- Server-side RM
  Server Reference Monitor (SRM) in this model is stored on the server‘s system and controls access to data. Information is allowed to be stored on the clients storage if it does not need to be controlled centrally. If access to the information has to be controlled centrally, this means that it must not be allowed to be stored on the client side. [59] [64]

- Client-side Vs server-side RM
  CRM is less reliable and credible than SRM because CSM is more likely to be manipulated and the information is vulnerable to be attacked by malicious users. However, CRM is needed when the client has to control delegation of information to other users on behalf of server. The better solution is having both SRM and CRM working together for better functionality and security. [59] [64]

Access control models

- Discretionary-access control (DAC)
  DAC is an access control mechanism that allows the user to change the access policy, where users are able to control access to their own resources. In addition, users can transfer the access rights to those resources to other users. DAC is flexible and easy to be implemented. However, it may be not secure because malicious users may easily change the access policy on behalf of the owner and make data accessible by others.

- Mandatory Access Control (MAC)
  MAC is an access control mechanism where the access to all resources is controlled by the system administrator. It is not allowed for users to change the access control policy. Mac is not easy to be implemented. Yet, it is more secure than DAC.

- Role Based access control (RBAC)
  RBAC provides a secure and effective way to control access to resources based on permissions and roles. Under RBAC Roles are formed to different job functions in a system, Users are assigned to the roles based on their responsibilities, permissions are assigned to roles and the user could be a member in more than one group. [10]

Electronic form access control

The electronic form could be involved in control access on different levels such as:

1. form-level access:
   A traditional user name and password form may be used to control access to other web pages of the application or to the database itself. Furthermore, users could have different access to different views of the form depending on their roles and their permissions.

2. database-level access:
   In this case, the roles, permissions, usernames and passwords are performed on data storage itself under the database system that is re-

sponsible to control access. The system asks the users to enter their credentials to be authorized to access to the database through the form.

2.1.5 Data Storage

Dynamic electronic form needs data storage (database) to store entered data in or retrieve data from. Electronic forms technologies use different methods and techniques for creating and dealing with data storage. This data storage could be one of the following:-

Client-side data storage

client-side data storage is a storage that is initiated by a specific web application to store data related to the user on the user‘s system. This data storage has different technologies such Cookies, IE UserData and HTML5 Web SQL database. These technologies use various techniques to create the data storage which could be a simple text based file, an xml file or binary database file. Some of these technologies are supported natively by the web browsers. Others need for plug-ins installation to be available on the browser. For example, Cookies are supported by all web browsers without need for plug-ins and Adobe Flash needs for plug-ins installation on the browser. [61]

Server-side data storage

Server-side storage is a storage that is created and managed by using a database server software called Data base management system (DBMS) or Relational Data Base Management System (RDBMS). DBMS/RDBMS allows different users and applications to access the central data storage across a network, where data can be searched and retrieved to produce reports or displayed in web pages. Examples of data base servers are MySQL, Microsoft SQL server and Oracle. [4]

2.1.6 SVG feature

Scalable Vector Graphics (SVG) is an xml-based language that is used to render graphical contents (vector graphics, texts and images). SVG could be used to capture data from the user in electronic forms, where SVG supports and captures mouse and keyboard events. Dom methods are used to handle events in SVG.

Client-side SVG

SVG on the client-side is executed and rendered on a web browser. Some of web browsers need to special plug-ins installation (i.e. internet explorer) for

SVG rendering and others are natively support SVG (i.e. firefox). When the user releases a mouse or keyboard events, SVG captures data that is sent to the server once the form is submitted.

server-side SVG

Data that sent by the client (including data which captured by SVG) is validated on server-side.

2.1.7 subforms feature

Subform is a form that is included in a main form. It contains information related to the main form. A subform could be part of more than one separate form that helps with designing and testing large-based form applications. The subform has its own controls, properties and events that can interact with the parent form. Using subforms enables the user to choose which part of the form wants to show. In addition, a secure access could be provided on some views or subforms.

client and server sides

Subforms support the same features that supported by any electronic form either client or server side.

2.1.8 Hint information feature

This feature is used to display information related to a specific form field. This information would be an ephemeral message that helps the user to have correct expectation about the data should be filled out in a form field.

client and server side

Hint information is just a message that is displayed to the user. Thus, it would be implemented on the client-side and there is no need for the server side implementation.

2.1.9 Dependency update feature

Using this feature enables controlling update changes between related form fields on each keystroke. For example, a form could have an output fields depending on entered data in one or more input fields.

client and server sides

This feature is performed on client side as there is no security requirement to be done on server-side.

2.1.10 Upload feature

The upload feature enables the client to select information from its local system to be sent to and stored on the server. The uploaded information would be attached with the form submission.

client-side upload

On the client-side, the information is validated to check if its metadata is correct. For example, to check if an uploaded file has a specific size and type.

server-side upload

Before saving the uploaded information on the server, It would be validated to check if its metadata and to make sure that it is secure.

2.1.11 spell checking

This feature is enabled in an electronic form to check if words of text data (that is filled in text form field) are spelled correctly

client and server sides

This feature could be just implemented o client-side and there is no need for server-side implementation, because it does not cause any type of security risks.

2.1.12 Autocomplete feature

This feature is provided be electronic forms to help users filling the form quickly. Autocomplete feature enables the form to give users suggestions for the words that they may want to type in (by typing a few letters of the word).

client and server sides

As this feature does not require any security implementations, it may be only performed on the client-side.

2.1.13 Summary

This chapter introduced the most significant form features that are supported by electronic forms. In addition, it demonstrated the client and server sides implementation of those features.

Chapter 3

Technology and Literature review

This chapter introduces relevant technologies and efforts that are presented and involved in the electronic forms development domain. The first section describes the major related technologies that based on W3C standards and could be used in electronic forms developing. The second section is about related work that is performed by other researchers to enhance web forms.

3.1 W3C electronic forms technologies

This section presents the most current major technologies that are based on W3C standards and used to develop electronic forms. The first section introduces the old HTML Forms. The second section is about XForms. The tird section is about HTML5 forms and the last section produces similarities and differences between XForms and HTML5 Forms and their features.

3.1.1 HTML Forms

HyperText Markup Language (HTML) is a markup language that is used for developing web applications and enables web browsers from rendering web pages. It includes many elements and tags that are used to construct a web page with texts, images, blocks, paragraphs, etc. HTML passed with different versions: HTML 2.0, HTML 3.0, HTML 3.2, HTML 4, HTML 4.01 and HTML5.

HTML forms (HTML4.01 and before) is one of W3C standards for developing electronic forms. HTML forms is defined by using **form** element. It is part of a web page and consists of different elements such as textbox, checkbox, menus, etc.

HTML forms have limitations that encourage the W3C working group to develop new electronic forms technologies. HTML forms depend on using scripting languages to implement the common tasks such as validation, calculation, handling errors. In addition, it is not easy to HTML forms to initial data and the user is not remembered by the form and has to repeat the same data each time. Furthermore, HTML forms do not separate date from presentation, where the collected data is a set of name and value pairs that are presented by the form elements. Finally, HTML forms are not useful to be used for form workflow because they need to translate the form data format at each workstation.

New electronic forms technologies were presented by W3C to address the problems that associated with the old HTML forms: HTML5 Forms and XForms (detailed later in this chapter). HTML5 Forms improve the markup by adding new input types and attributes, and reduce the need for scripting that was required in the old ones. The powerful XML-centric of XForms fixed most of the old HTML forms limitations. Moreover, XForms remove the need for using scripts to perform the common tasks such as validation and calculation that are included within the language structure.

3.1.2 XForms

Xforms is one of W3C standards. It designed to be the next generation of web forms that presents a new platform-independent markup language for user interaction, and transactional procedures between a client and server. XForms is an XML-based application that is integrated with other markup languages such HTML, XHTML or SVG. The major difference between XForms and HTML forms is the separation between the presentation of controls (user interface) and data collected by controls (data modeling), where XForms presents the form in three parts: [19]

- An explicit data model to define data type of form data.
- The user interface that is consisted of controls presentation which is bound to the data module.
- Using XML to exchange data with the server.

This section reviews The primary XForms versions, presents the main benefits of using XForms, describes browsers support for XForms and finally outlines the supported form features in XForms.

3.1.2.1 Major XForms versions

- **XForms 1.0: Data Model** was published on 6 April 2000. It is The first XForms specifications which was the initial XForms Working

Draft. This draft presented a proposal of the explicit data module that is implemented by XForms, where it described XForms datatypes and sytax that is defines in XML schema specifications [19].

- **XForms 1.0 (Third Edition)** was published on 29 October 2007. The specification of this version produced a new platform-independent markup language (XForms processor) for user interaction between a client and server [11].

- **Xforms 1.1** was published on 20 October 2009. It enhanced the XML processing platform introduced in **XForms 1.0** by improving the user interface and adding new submission capabilities, action handlers, utility functions, more helpful datatypes and powerful action processing ability. Moreover, it includes conditional, iterated and background execution, and the capability to control querying data as well as to access event context information [12]

- **Xforms 2.0** was published on 29 March 2012. The specification of this version added a number of new features such as dialogs, custom functions, variables, a pluggable expression language with extra functions (XPath 2.0), model-based switch and repeat, Attribute Value templates, and consuming as well as submitting JSON and CSV instance data [31]

3.1.2.2 The essential benefits of using XForms

The following advantages of using XForms demonstrate the primary improvements that are achieved by Xforms:

- **Strong typing**: XForms has strongly typed form data and this speeds up the validating and submitting the form.

- **Reduce scripting**: the strong typing and using XML schema in XForms reduce the need for scripts to handle events.

- **Reuse existing schema**: Same XML schema can be used more than one time in the same XForms application. This eases updating of validation constrains and avoids duplication.

- **External schema addition**: the form designer can provide additional constrains that are included within the XForms module to improve the overall usability of the resulting Web application.

- **XML submission**: this enables validating and processing the received XML instance data by the client.

- **improved accessibility**: XForms user interface is comprehensive to be accessed by different devices because XForms separates data from presentation, as well as user interface controls encapsulate all related metadata like labels.
- **Internationalization**: the Internationalization of data is ensured by using XML 1.0 for instance data.
- **Multiple device support**: XForms user interface controls are generic and suits for various devices.

3.1.2.3 XForms browsers support

XForms is not natively supported by common web browsers and it needs for plug-in installation to be displayed. However, there are some implementations of XForms that enables rendering XForms without the need for plug-ins. Client-side implementations like XSLTForms (explained later in chapter 4) and server-side implementations such as Better forms and Orbeon form (explained later in chapter ?) that transform Xforms document to JavaScript + HTML document allow supporting XForms without using any plug in installation.

3.1.2.4 supported Form features in XForms

This section describes the most major electronic forms features (these features were explained in chapter2, will be assessed in this chapter and chapter3 with the tested electronic forms technologies and applications) that are supported and not supported by XForms. The features are demonstrated with examples from XFroms code.

3.1.2.4.1 Data validation

Xforms has two types of validation rules:

1. Validation based on xml schema: the submitted data will be validated using a given xml schema. [49]
   Example:

```
<xf:model schema="schema.xsd">
   <xf:instance src="instance.xml" />
   <xf:bind id="zip" required="true()" type="ftype:zipType"

nodeset="ftype:zip" />
   <xf:bind id="zip2" required="true()" type="ftype:zip2Type"
```

```
nodeset="ftype:zip2" />
   </xf:model>
```

code source: `http://en.wikibooks.org/wiki/XForms/Validation_with_Bind`

2. validation with bind: data will be validated based on specific type and constraint. [49]
   Example:

```
<xf:model>
   <xf:instance xmlns="">
      <data>
        <PositiveDecimalAmount>1.0</PositiveDecimalAmount>
         <NegativeDecimalAmount>-1.0</NegativeDecimalAmount>
       </data>
    </xf:instance>
 <!-- note that the gt operator does not work and that the greater
than character must be escaped.  -->
     <xf:bind nodeset="PositiveDecimalAmount" type="xs:decimal"

required="false()"
constraint=". &gt; 0"/>

<!-- note that the lt operator does not work and that the less than character
must be escaped.  -->
     <xf:bind nodeset="NegativeDecimalAmount" type="xs:decimal"

required="false()"
constraint=". &lt; 0"/>
</xf:model>
```

code source: `http://en.wikibooks.org/wiki/XForms/Validate_with_schema_types`

It seems that both of those two types of validation are at the same powerful level. The difference between them may be that the validation using schema is more useful with complex forms, where the form could use more than one schema and one schema could be used by more than one form.

3.1.2.4.2 Dynamic form modification

This feature is supported by XForms. For example controls could be added to or removed from the form using **repeate** element, and then

using **insert** element to add the new element to DOM and **delete** to remove an element from DOM.

```
<repeat id="repeat" nodeset="telephones/telephone" appearance="compact">
<select1 ref="@type" appearance="minimal">
<label>Type</label>
<item>
<label>Home</label><value>home</value>
</item>
<item>
<label>Mobile</label>
<value>mobile</value>
</item>
<item>
<label>Company</label>
<value>company</value>
</item>
</select1>
<input ref=".">
<label>Number</label>
</input>
<trigger>
<label>X</label>
<delete nodeset="." at="1" if="count(//telephone) > 1"
ev:event="DOMActivate" />
</trigger>
</repeat>
<trigger>
<label>New</label>
<insert nodeset="telephones/telephone" at="index('repeat')"
position="after" ev:event="DOMActivate" />
</trigger>
```

code source: `http://www.agencexml.com/xsltforms/xf.xml`

3.1.2.4.3 Calculation

XForms provides standard ways to support calculation without the need to use JavaScript.
For example:
The following calculation is implemented using **bind** element and **calculate** attribute:

```
<xf:bind nodeset="Results/sum" calculate="sum(../../Row/A)"
```

```
type="xsd:decimal" />
```

code source: `http://www.agencexml.com/xsltforms/spreadsheet.xml` The following calculation is by using **toggle** element that call **case** which acts as a function to perfume sum or add operation.

```
<xf:toggle case="add"/>
</xf:action>
</xf:trigger>
<xf:switch>
<xf:case id="add" selected="true">
<xf:trigger>
<xf:label>=</xf:label>
<xf:action ev:event="DOMActivate">
<xf:setvalue ref="second" value="../screen"/>
<xf:setvalue ref="result" value="../first + ../second"/>
<xf:setvalue ref="screen" value="../result"/>
<xf:setvalue ref="screenbuffer" value="0"/>
</xf:action>
</xf:trigger></xf:case>
```

code source: `http://www.agencexml.com/xsltforms/calculator.xml`

3.1.2.4.4 Data Storage

Xforms supports XML database. It uses XForms data module to store data form. In addition, XForms can provide support for JSON data storage (see section 1.4).

3.1.2.4.5 Access rights control

Access rights control is not explicitly supported by XForms. The XML data is accessible by default and the form responses to the access rights that configured by the database system which is connected to the form.

3.1.2.4.6 SVG feature

SVG is supported by XForms and could be used to capture data from the user.

```
<svg xmlns=http://www.w3.org/2000/svg
xmlns:xlink="http://www.w3.org/1999/xlink" width="100%" height="100%"
viewBox="0 0 450 250" onload="init()" xml:space="preserve">
```

```
 <script type="text/ecmascript">
 <![CDATA[
   var evtText1; var evtText2; var evtText3;
var typeInitialized = false; var text = "";
function init() {
  //getting references to dynamic text elements
    evtText1 = document.getElementById("evtText1");
    evtText2 = document.getElementById("evtText2");
    evtText3 = document.getElementById("evtText3");
      }
    //this function handles the keyboard events
        function typeText(evt) {
           //handle "keypress" for all "real characters"
            if (evt.type == "keypress") {
             //some browsers support evt.charCode, some only evt.keyCode
               if (evt.charCode) {
                 var charCode = evt.charCode;
                   }
                   else {
                    var charCode = evt.keyCode;
                   }
                   //all real characters
   if (charCode > 31 && charCode != 127 && charCode < 65535) {
        text += String.fromCharCode(charCode);
                   }
          //backspace key
   if (charCode == 8) {
                   //shorten text
                text = text.substring(0,text.length-1);
                   }
                   //enter key
                   else if (charCode == 10 || charCode == 13) {
                         stopTyping(evt);}}
              //update dynamic text
              evtText1.firstChild.nodeValue = text;
     //suppress unwanted browser shortcuts. e.g. in Opera or Mozilla
              evt.preventDefault();}
            //this function adds the event listeners
            function initTyping(evt) {
                if (!typeInitialized) {
document.documentElement.addEventListener("keypress",typeText,false);
document.documentElement.addEventListener("click",stopTyping,false);
     evtText2.firstChild.nodeValue = "Typing Active";
                   typeInitialized = true;}
```

```
     //we don't want the click event on the document level to
      //immediately stop the typing mode
          evt.stopPropagation();
            }
function stopTyping(evt) {
document.documentElement.removeEventListener("keypress",typeText,false);
document.documentElement.removeEventListener("click",stopTyping,false);
                    typeInitialized = false;
                    evtText2.firstChild.nodeValue = "Typing Inactive";
                    evtText3.firstChild.nodeValue = "You typed: "+text;
            }
        ]]>
    </script>
    <rect id="bgRect" x="-500" y="-500" width="1500" height="1500"
fill="yellow" stroke="none" />
    <rect x="10" y="50" width="150" height="20" fill="white" stroke="black"
onclick="initTyping(evt)" />
    <g font-size="10px" font-family="Arial">
<text x="10" y="130">Click in the white rectangle to start typing text.
<tspan x="10" dy="15">Click outside the textbox or press enter to stop writing
text.</tspan>
<tspan x="10" dy="15">Press backspace to delete the last character when typing
is active</tspan></text>
 <text x="15" y="65" id="evtText1" pointer-events="none"> </text>
 <text x="15" y="90" id="evtText2">Typing Inactive</text>
 <text x="15" y="105" id="evtText3">You Typed: </text>
    </g>
</svg>
```

code source: `http://www.carto.net/svg/eventhandling/`

3.1.2.4.7 Subforms feature

Subforms are implemented in XForms using the element **load** that has the attributes **show** with the value **embed** and the attribute **target** to determine the position of the subform in the main form.

```
<xforms:load if="@selected != 'true'" show="embed" targetid="subform"
resource="books.xml"/>
```

code source: `http://www.agencexml.com/xsltforms/writers.xml`

3.1.2.4.8 Hint information feature

This feature is implemented in XForms using **hint** element.

```
<xf:hint>Also known as given name.</xf:hint>
```

code source: `http://www.agencexml.com/xsltforms/input.xml`

3.1.2.4.9 Dependency update feature

This feature is supported by XForms and implemented using **incremental** attribute with the **input** element.

```
<xf:input ref="PersonSurName" incremental="true">
```

code source: `http://www.agencexml.com/xsltforms/input.xml`

3.1.2.4.10 Upload feature

Xforms provides **upload** element to attach a file with the form. The uploaded file is stored in the XForms data module with a specific type which is indicated with **ref** attribute.

```
<xforms:model>
<xforms:instance xmlns="">
<data>
<attachment3 xsi:type="xsd:base64Binary"/>
</data>
</xforms:instance>
</xforms:model>

<xforms:upload ref="attachment3" incremental="true">
<xforms:label>Upload for base64Binary: </xforms:label>
</xforms:upload>
```

source code: `http://www.agencexml.com/xsltforms/uploads.xml`

3.1.2.4.11 Spell checking

XForms does not provide spell checking. However, there is an implementation of XForms (**Orbeon Forms**) that supports this feature (see Orbeon Forms section in chapter4).

3.1.2.4.12 Autocomplete feature

Although this feature is not supported by XForms, it is provided by the server-side implementation of XForms **Orbeon Forms** (see Orbeon Forms section in chapter4).

3.1.3 HTML5 Forms

HTML5 developed by W3C and Web Hypertext Application Technology Working Group (WHATWG). First, WHATWG group created HTML5 specification in 2004. It was titled web 2.0 on that time (HTML5 forms was called web form 2.0 as well). In 2006, W3C started working on developing HTML5 and published the first draft for HTML5 specification in 2008 [36]. HTML5 forms adds more features (new input types and new attributes) that address the practical limitations in HTML4 forms. For example, one of these limitations in HTML4 form that the validation is performed using a script associated with the form, while HTML5 forms has a built-in validation feature which cancels the need for a script [35]

This section introduces a brief description of the new input types and new attributes that are included in HTML5 forms, and then demonstrates the browser support of HTML5 forms, and finally discusses the supported form features in HTML5 forms.

3.1.3.1 New HTML5 forms attributes and input types

This section presents several new attributes and input types that are developed in HTML5 forms. Those improvements add more powerful features to the form and avoid the need to write lots of scripts. [43]

(a) **HTML5 forms attributes**

- **Placeholder**: this attribute allows to display a short description of text when the field is empty or does not receive a focus.
  example:
  ```
  <input type="text" name="user-name" id="user-name"
  placeholder="at least 3 characters">
  ```
- **Autofocus**: is used to give the field an automatic focus when the page is rendered. Autofocus is a Boolean attribute and could be implemented in the following ways in HTML5:
  Autofocus
  autofocus=""
  autofocus="autofocus"

  example:
  ```
  <input type="text" name="first-name" id="first-name" autofocus>
  ```
  However, In XHTML5, this attribute is not Boolean and has to be implemented using **autofocus="autofocus"** style.
- **Autocomplete**: enables the user to complete the form based on earlier input. By default, this attribute is set to **on** and it could be disabled by set its state to **off**
  example:
  ```
  <input type="text" name="tracking-code" id="tracking-code"
  autocomplete="off">
  ```

- **Required**: this attribute is added to a form field to inform the user that data must be entered in that field before submitting the form. An error message is displayed automatically to the user if a required field is left empty.
  example:
  ```
  <input type="text" id="given-name" name="given-name" required>
  ```
- **Pattern**: is used to implement a JavaScript regular expression (regex or regexp is predefined symbols that are used to create patterns to be checked if match texts or strings of texts.) to a field's value. Data entered in that field is checked based on the pattern that performed to its value.
  example:
  ```
  <label> Product Number:
  <input pattern="[0-9][A-Z]{3}" name="product" type="text"
  title="Single
  digit followed by three uppercase letters."/>
  </label>
  ```
- **List and datalist element**: **list** attribute enables the user to display a list of options for a form field based on what the user typed in. List attribute is associated with **datalist** element by using the same ID and they should reside in the same document. The new HTML5 datalist element is used to defined list of options for form fields.
  example:
  ```
  <label>Your favorite fruit:
  <datalist id="fruits">
  <option value="Blackberry">Blackberry</option>
  <option value="Blackcurrant">Blackcurrant</option>
  <option value="Blueberry">Blueberry</option>
  <!--  -->
  </datalist>
  If other, please specify:
  <input type="text" name="fruit" list="fruits">
  </label>
  ```
- **Multiple**: this Boolean attribute is used to enable the user enter more than one value of **datalist** options
  example:
  ```
  <label>Your favorite fruit:
  <datalist id="fruits">
  <select name="fruits">
  <option value="Blackberry">Blackberry</option>
  <option value="Blackcurrant">Blackcurrant</option>
  <option value="Blueberry">Blueberry</option>
  <!--  -->
  </select>
  If other, please specify:
  </datalist>
  <input type="text" name="fruit" list="fruits" multiple>
  </label>
  ```

Apart from using with the datalist element, just one more implementation for multiple attribute to enable the user send or attach files with the e-mail.
example:

```
<label>Upload files:
<input type="file" multiple name="upload"></label>
```

- **novalidate and formnovalidate**: these Boolean features are used to disable validation when a form is submitted. Novalidate could be only used with the form element, while formnovalidate could be used with a form field (submit or image input types).
formnovalidate example:

```
<form action="process.php">
<label for="email">Email:</label>
<input type="text" name="email" value="gordo@example.com">
<input type="submit" formnovalidate value="Submit">
</form>
```

novalidate example:

```
<form action="process.php" novalidate>
<label for="email">Email:</label>
<input type="text" name="email" value="gordo@example.com">
<input type="submit" value="Submit">
</form>
```

- **Form**: could be used with input, select or textarea elements. Form attribute is used to specify an element associated with one or more form, where the element may be written from anywhere in the source and does not need to be a child to a form.
example:

```
<form action="form.php" id="apply">
 First name: <input type="text" name="fname"><br>
 <input type="submit" value="Submit">
</form>

Last name: <input type="text" name="lname" form="apply">
```

- **Formaction**: is used to specify the url of a file or script that processes the form when is submitted. This attribute is similar to **action** attribute. However, the browser first checks the formaction attribute if is presented. Formaction is used with submit button (type=‘submit‘) or image button (type=‘image‘).
example:

```
<input type="submit" value="Submit" formaction="process.php">
```

- **Formenctype**: this attribute indicates how form data is encoded when the form is submitted by using the **POST** method type. It is used with type=‘submit‘ or type=‘image‘ of input element. Formenctype is similar to the **enctype** attribute and it has the default value **application/x-www-formurlencoded**. **example:**

```
<input type="submit" value="Submit" formenctype="application
/x-www-form-urlencoded">
```

- **formmethod**: determines the HTTP method (GET, POST, PUT and DELETE) that will submit the form data. It is similar to **method** attribute and can be used with submit or image input types.
  example:
  ```
  <input type="submit" value="Submit" formmethod="POST">
  ```
- **formtarget**: indicates where to display the target window that receives the result from submitting the form. It is similar to the **target** attribute and is used only with submit and images input types.
  example:
  ```
  <input type="submit" value="Submit" formtarget="_self">
  ```

(b) **HTML5 forms input types**

- **Search**: is used to define a search field. It works as google search or any other search web site.
  example:
  ```
  <input type="search" name="search">
  ```
- **email**: this type is for defining an input type field to enter an email address that is validated automatically.
  example:
  ```
  <input type="email" name="email">
  ```
- **url**: is to allocate an input field that is used to enter a url addresss. This field is automatically validated by the browser when the form is submitted.
  example:
  ```
  <input type="url" name="url">
  ```
- **tel**: is used for input types fields that allow the user to enter telephone numbers. This field is validated automatically. However, there is no a specific syntax is enforced for validation because phone numbers are different around the world.
  example:
  ```
  <input type="tel" name="tel" id="tel">
  ```
- **number**: specifies an input fields enter numerical values. This input type field may used with additional attributes that are used to apply restrictions to the entered number. These attributes are: min- indicates the minimum value, max- indicates the maximum value, step- indicates the default step value of the spinbox control and value- indicates the default value. **example:**
  ```
  <input type="number" min="5" max="18" step="0.5" value="9"
  name="shoe-size" >
  ```
- **range**: defines an input field that represents a numerical value within a given range. It is similar to a number type but simpler and more specific. Range type is displayed as a slider control and uses the attributes min, max, step and value (see the number type) for specifications.
  example:
  ```
  <input id="skill" type="range" min="1" max="100" value="0">
  ```

- **date**: is used to display a date picker that allows the user to select a certain date.
  example:
  ```
  <input id="dob" name="dob" type="date">
  ```
  It could use min and max attributes to apply a specific range
  ```
  <input id="startdate" name="startdate" min="2012-01-01"
  max="2013-01-01" type="date">
  ```
- **week**: is used to display a date picker that allows the user to select a certain week and year.
  example:
  ```
  <input id="vacation" name="vacation" type="week">
  ```
- **month**: is used to display a date picker that allows the user to select a certain month and year.
  example:
  ```
  <input id="expiry" name="expiry" type="month" required>
  ```
- **time**: is used to render a time control that enables the user to enter or select a specific time.
  example:
  ```
  <input id="exit-time" name="exit-time" type="time">
  ```
- **datetime**: this type renders date and time picker (with time zone) that enables the user to select a time on a specific day.
  example:
  ```
  <input id="entry-day-time" name="entry-day-time" type="datetime">
  ```
- **datetime-local**: this type renders date and time picker (with a local time zone) that enables the user to select a time on a specific day.
  example:
  ```
  <input id="arrival-time" name="arrival-time " type="datetime-local">
  ```
- **color**: this input type displays a color picker that allows the user to select a color and returns its hex value. The user can either type the hex value of the color or select from the color picker.
  example:
  ```
  <input id="color" name="color" type="color">
  ```

3.1.3.2 Browsers support for HTML5 forms

The previous HTML5 forms are not supported natively by old versions of web browsers. In addition, not all modern browsers support all of them. If a feature is not supported by a browser, it would be backed to the default type (type = 'text'). This section describes different ways that are used to check if a feature is supported by a browser or not and demonstrates how to make a non-supported browser works as a native supported browser for a feature. [43, 57]

3.1.3.2.1 Form detection

- **Mark Pilgrims detection techniques**: they are four techniques that are used to detect browser support for a particular feature:
  (a) check if a specific property exists on a global object such as a window or navigator.
  (b) create an element and check if a specific property exists on it.
  (c) create an element, check if a specific method exists on that element ,and then call that method to check its returned value.
  (d) create an element, set a property to a specific value, and then check if the property has saved that value.

  The following example uses detection technique number 4 to check if an input type **email** is supported:

```
var input = document.createElement('input');
input.setAttribute('type','email');
return input.type !== 'text';
```

- **Modernizr**: is an open source, MIT-licensed small JavaScript library that is used to check the supported native implementations for next-generation web technologies- is used to detect if a browser has a native support for a certain feature or not. Modernizr is implemented in the web page by using **script**element at the top of the page:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Dive Into HTML5</title>
  <script src="modernizr.min.js"></script>
</head>
<body>
  ...
</body>
</html>
```

3.1.3.2.2 Form Polyfills

Mark Pilgrim techniques and Modernizr are just to check the availability of the native implementation of HTML5 features in the browser and they do not add any functionality that not exist. Polyfill is a plugin that is written using JavaScript. It is used to add a support of a feature in HTML5 to browsers (older browsers) that are not native support.

The following figures represent the current browsers that support and not support the new attributes and input types in HTML5 forms.

	Trident					Gecko		WebKit				Presto	
	IE 6	IE 7	IE 8	IE 9	IE 10	Firefox 4	Firefox 6	Safari 4	Safari 5	Chrome 4	Chrome 5-13	Opera 9.5+	Opera 11 beta
autocomplete	.	.	.	.	.	✓	✓	.	.	.	.	✓	✓
autofocus	.	.	.	.	✓	✓	✓	✓	✓	✓	✓	✓	✓
list	.	.	.	.	✓	✓^	✓	.	.	.	.	✓	✓
max	.	.	.	.	✓		.	.	✓	✓	✓	✓	✓
min	.	.	.	.	✓		.	.	✓	✓	✓	✓	✓
multiple	.	.	.	.	✓	✓^^	✓	✓	✓	✓	✓	.	✓*
pattern	.	.	.	.	✓	✓	✓	.	.	✓	✓	✓	✓
placeholder	.	.	.	.	✓	✓	✓	✓	✓	✓	✓	.	✓
required	.	.	.	.	✓	✓	✓	.	✓	✓	✓	✓**	✓**
step	.	.	.	.	✓		.	.	✓	.	✓	✓	✓

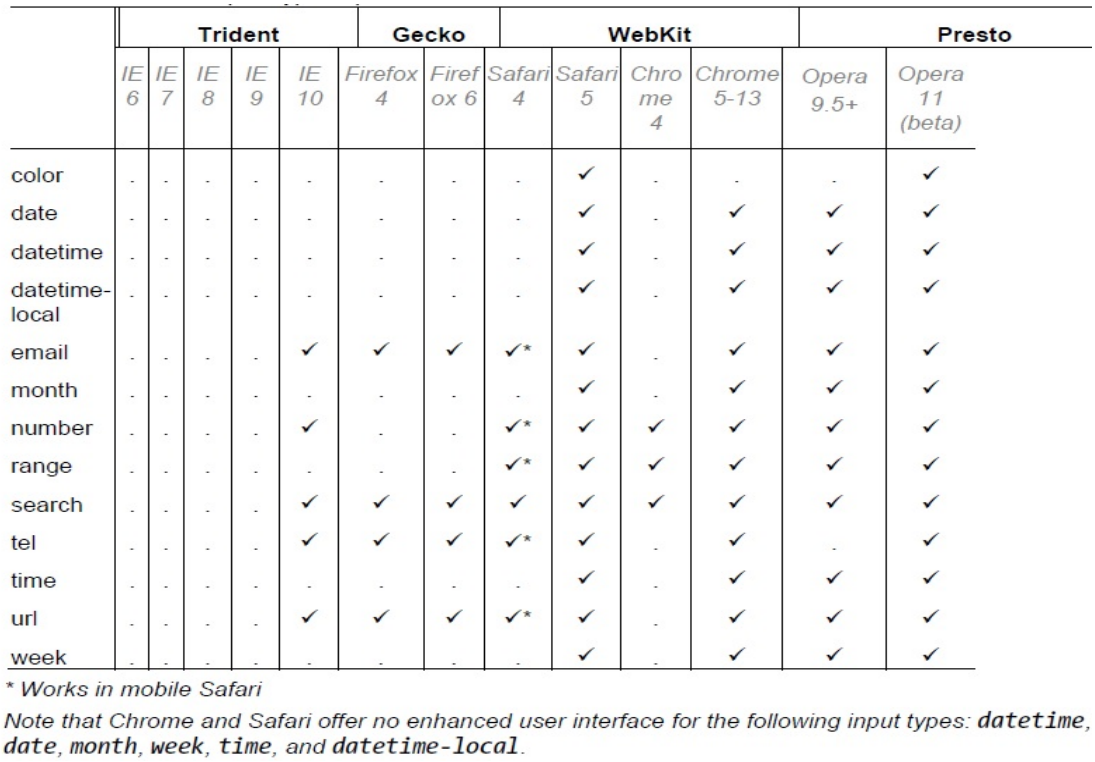
^ *Partial implementation, list renders permanently*

^^ *Currently not implemented for* `type="email"`

* *For* `<input type="file">`

** *No support for select elements*

Figure 3.1: HTML5 Form Attribute Implementation, sourced from [43]



	Trident					Gecko		WebKit				Presto	
	IE 6	IE 7	IE 8	IE 9	IE 10	Firefox 4	Firefox 6	Safari 4	Safari 5	Chrome 4	Chrome 5-13	Opera 9.5+	Opera 11 (beta)
color	.	.	.	.	.	.	.	.	✓	.	.	.	✓
date	.	.	.	.	.	.	.	.	✓	.	✓	✓	✓
datetime	.	.	.	.	.	.	.	.	✓	.	✓	✓	✓
datetime-local	.	.	.	.	.	.	.	.	✓	.	✓	✓	✓
email	.	.	.	.	✓	✓	✓	✓*	✓	.	✓	✓	✓
month	.	.	.	.	.	.	.	.	✓	.	✓	✓	✓
number	.	.	.	.	✓	.	.	✓*	✓	✓	✓	✓	✓
range	.	.	.	.	.	.	.	✓*	✓	✓	✓	✓	✓
search	.	.	.	.	✓	✓	✓	✓	✓	✓	✓	✓	✓
tel	.	.	.	.	✓	✓	✓	✓*	✓	.	✓	.	✓
time	.	.	.	.	.	.	.	.	✓	.	✓	✓	✓
url	.	.	.	.	✓	✓	✓	✓*	✓	.	✓	✓	✓
week	.	.	.	.	.	.	.	.	✓	.	✓	✓	✓

Figure 3.2: HTML5 Form input types, sourced from [43]

3.1.3.3 Form supported features in HTML5 Forms

This section explains the most essential electronic forms features that are supported and not supported by HTML5 Forms. These features are verified and illustrated using examples from available references.

3.1.3.3.1 Data validation

HTML5 Forms implement client- side validation and provide a live object **ValidityState** for each element created in a form. A reference to this object is used to check the validity status of form controls.
The following command captures a reference to the **ValidityState** of a form control named **myInput** [36].

```
var valCheck = document.myForm.myInput.validity;
```

The following command returns a Boolean value that demonstrates if validation constrains of a form element are met or not.

```
valCheck.valid
```

There are eight different status or constrains are used to implement form validation in HTML5 forms. These validity status are checked by using references to their **ValidityState** objects [36].

- using **required** attribute: is used to make sure that the form field is not empty (see section1.1).
- using **input types** feature: is used to ensure that the value of a form field matches the value of type attribute such as number, email, url, etc (see section 1.1).
- using **pattern** attribute: garentees that the value of a form field matches the pattern rule of the pattern value attribute (see section 1.1).
- using **maxLength** attribute: ensures that the value of a form field is not too long (see section1.1).
- using **min** attribute: to enforce the minimum value of a numerical field (see secion 1.1).
- using **max** attribute: to enforce the maximum value of a numerical field (see section1.1).
- using **step** attribute: enforces the incremental step between **min** and **max** values of a numerical field (see section 1.1).
- **setCustomValidity**: is used to handle errors that are set by the application code and put the form control into the customError state. For example: display an error message if a password confirmation field does not match.

```
passwordConfirmationField.setCustomValidity("Password values do not
match.");
\end{itemize}
```

3.1.3.3.2 Validation attributes and functions

HTML5 forms provide a few attributes and functions that help the developer to improve validation process [36].

- **willValidate** attribute: renturns true if a form control will be validated, otherwise false.
- **validationMessage** attribute: is used to display en error message to inform the user that a certain form control will be validated. For example, if a **required** has left empty, an error message (This field requires a value.) will be displayed by the browser (once supported) to the user.
- **checkValidity** Function: returns true if the form control‘s value is valid, otherwise false.
  Disable validation in HTML5 Forms

  Validation checking in HTML5 form could be turned off using **novalidate** and **formonvalidate** attributes as explained in section 1.1.

3.1.3.3.3 Dynamic form modification

This feature could be implemented using JavaScript in HTML5 Forms. The following part of code is to dynamically add more fields to an HTML5 form: [58]

```
function moreFields() {
        counter++;
        var newFields = document.getElementById('readroot')
.cloneNode(true);
        newFields.id = '';
        newFields.style.display = 'block';
        var newField = newFields.childNodes;
        for (var i=1;i<newField.length;i++) {
            var theName = newField[i].name
            if (theName)
                newField[i].name = theName + counter;
        }
        var insertHere = document.getElementById('writeroot');
        insertHere.parentNode.insertBefore(newFields,insertHere);
    }
```

The following part of code is to dynamically remove fields from an HTML5 form: [58]

```
<input type="button" value="Remove review"
onclick="this.parentNode.parentNode.removeChild(this.parentNode);" />
```

3.1.3.3.4 Calculation

This feature is supported by HTML5 Forms. The **output** element is used to perform and display the result of any calculation. The output element is supported in Opera 9.5+, Firefox 4+, Chrome, Safari 5+, and InternetExplorer 10 [43].
example:

```
<form onsubmit="return false" oninput="o.value = a.valueAsNumber +
b.valueAsNumber">
<input name="a" id="a" type="number" step="any"> +
<input name="b" id="b" type="number" step="any"> =
<output name="o" for="a b"></output>
</form>
```

3.1.3.3.5 Data Storage

HTML5 Forms support two kinds of data storage: **web storage** and **web SQL databases**.

(a) **Web storage**: is similar to cookies. However, it is faster and more secure because data is used just when is requested and is not sent

with each server request. As well as it enables to store large amount of data, where data is stored in key/value pairs. There are two types web storage objects: **sessionStorage**and**localStorage** [63, 36].

- **SessionStorage**: this object stores data to be available for only one window and it is expired when that window is closed.

```
if (sessionStorage.clickcount)
 {
 sessionStorage.clickcount=Number(sessionStorage.clickcount)+1;
 }
else
 {
 sessionStorage.clickcount=1;
 }
document.getElementById("result").innerHTML="You have clicked
the button " + sessionStorage.clickcount + " time(s)
in this session.";
```

- **localStorage**: data that is stored in this object is not expired even when the browser is closed. Stored data are available for all windows that are related to the same domain.

```
if (localStorage.clickcount)
 {
 localStorage.clickcount=Number(localStorage.clickcount)+1;
 }
else
 {
 localStorage.clickcount=1;
 }
document.getElementById("result").innerHTML="You have clicked
the button " + localStorage.clickcount + " time(s).";
```

3.1.3.3.6 Browser support for web storage

Browser support for web storage should be checked before using:

```
function checkStorageSupport() {
//sessionStorage
if (window.sessionStorage) {
alert('This browser supports sessionStorage');
} else {
alert('This browser does NOT support sessionStorage');
}
//localStorage
if (window.localStorage) {
alert('This browser supports localStorage');
} else {
alert('This browser does NOT support localStorage');
}
}
```

The following table shows browsers that have support for web storage:

Browser	Details
chrome	Supported in version 3.0 and greater
Firefox	Supported in version 3.0 and greater
Internet Explorer	Supported in version 8.0 and greater
Opera	Supported in version 10.5 and greater
Safari	Supported in version 4.0 and greater

Table 3.1: HTML5 web storage browser support, reprinted from [36]

(b) **Web SQL database**: allows HTML5 forms to store and access data in an indexed database [36]. The following table shows browsers support for web SQL databases:

Browser	Details
chrome	Supported in version 3.0 and greater
Firefox	Not supported
Internet Explorer	Not supported
Opera	Supported in version 10.5 and greater
Safari	Supported in version 3.2 and greater

Table 3.2: HTML5 web SQL database browser support, reprinted from [36]

3.1.3.3.7 JSON Object Storage JavaScript Object Notation (JSON) is an open standard for storing and exchanging data as texts. It could be used to represent objects as strings and vice-versa. HTML5 forms allows using JSON to serialize complex objects in and out of web storage, where some browsers limit values to be text string data types and modern versions of browsers contain built-in support for JSON [36]. **example:**

```
<script>
var data;
function loadData() {
data = JSON.parse(sessionStorage["myStorageKey"])
}
function saveData() {
sessionStorage["myStorageKey"] = JSON.stringify(data);
}
window.addEventListener("load", loadData, true);
window.addEventListener("unload", saveData, true);
</script>
```

3.1.3.3.8 Access rights control

A traditional log in form could be implemented using HTML5 forms.

The form can access data based on the roles and permission that performed on the server database. I found a technical report (http://www.script-tutorials.com/access-control-with-bit-masks/) that represents an example to perform user access rights using bit masks. This example is implemented with PHP script that defines create, read, edit and delete permissions.
The following code is part of PHP script that is used to define access rights to users:

```
<?php
// define bit mask for access rights
define('CAN_READ', 1 << 0);// 000001
define('CAN_CREATE',1 << 1);// 000010
define('CAN_EDIT_OWN', 1 << 2);// 000100
define('CAN_DELETE_OWN', 1 << 3);// 001000
define('CAN_EDIT_ANY', 1 << 4);// 010000
define('CAN_DELETE_ANY', 1 << 5);// 100000

...

// different sets of permissions
$sUserPerm= CAN_READ;
$sWriterPerm= CAN_READ | CAN_CREATE | CAN_EDIT_OWN | CAN_DELETE_OWN;
$sModeratorPerm= CAN_READ | CAN_EDIT_ANY | CAN_DELETE_ANY;
$sAdminPerm= CAN_READ | CAN_CREATE | CAN_EDIT_OWN | CAN_DELETE_OWN |
CAN_EDIT_ANY | CAN_DELETE_ANY;

...

functionisCanReadCreate($sRule) {
return($sRule& (CAN_READ | CAN_CREATE));
}
...

// check functions
functionisCanRead($sRule) {
return($sRule& CAN_READ) ?'You can Read<br />':'';
}
...
```

3.1.3.3.9 SVG feature

HTML5 support inline SVG. The same example in section 2.1.4.6 In HTML5 Forms is used to demonstrate how SVG could be used to capture data from the user.

3.1.3.3.10 Subforms feature

HTML5 forms do not provide any element or type to implement sub-

forms. However, there is a tool called **Zend Framewoek**(is an open source framework and object-oriented application. It is used for developing web applications using PHP5) can be used to embed subforms in a HTML5 form (http://webdeveloper.eu5.org/2013/04/22/subforms-example-in-zend-framework-1-x/).

3.1.3.3.11 Hint information feature

This feature is provided in HTML5 forms and implemented with **placeholder** attribute (see section 1.1).

3.1.3.3.12 Dependency update feature

I could not find a reference related to this feature in HTML5 forms. However, in my opinion, it seems that HTML5 forms do not provide an element or attribute to support this feature but it could be implemented using JavaScript.

3.1.3.3.13 Upload feature

This feature is available in the previous version of HTML and it is supported in HTML5 forms.

```
<p>
Please select a file:<br>
<input type="file" name="info_file" size="40">
</p>
```

3.1.3.3.14 Spell checking

HTML5 forms support spell checking feature with **spellcheck** attribute. Spellchecking is enabled when **spellcheck** attribute is true and when it is false, the spell checking is disabled. This feature is supported in Internet Explorer 10, Firefox, Opera, Chrome, and Safari [62].

```
<input type="text" size="30" spellcheck="true">
<textarea spellcheck="false"></textarea>
```

3.1.3.3.15 Autocomplete feature

HTML5 forms provide **autocomplete** (see section 1.1) attribute that is used to enable this feature.

3.1.4 XForms VS. HTML5 forms

This section produces the following tables that outline similarities and differences between XForms and HTML5 Forms:

XForms	**HTML5 Forms**
Built-in validation (no need for a script).	Bult-in validation (no need for a script).
Non-supported natively by web browsers (need for plug in installation.)	supported nativelt by modern web browsers (Old browser and non-supported browsers need for Polyfill plug in to add the support for HTML5 froms).
XForms is an XML-based. It cannot be standalone and has to be integrated with other markup language such as XHTML or SVG.	HTML5 Forms is a standalone technology and it could be embedded with XML or SVG.
XForms is more powerful and suitable for designing complex forms with XML data.	HTML 5 Forms is simpler and more suitable for designing simple forms.

Table 3.3: XForms VS. HTML5 Forms

Form feature	**Electronic form technology**	
	XForms	**HTML5 Forms**
Data validation	supported (built-in validation)	supported (built-in validation)
Dynamic form modification	supported	not directly supported and could be implemented with JavaScript
Calculation	supported (by using **output** element)	supported (by using standard elements)
Data storage	supported (XML database and JSON support may be added)	supported (web storage and web SQL database)
Access rights control	not supported (XForms considers XML data is accessible and relies on access rights in a connected database system)	not directly supported (could be performed with a script or relies on access rights in a connected database system)
SVG	supported and could be used to capture data	supported and could be used to capture data
Subform	supported and is implemented by using **load** element	not directly supported (could be add to HTML5 Form by using a tool called **Zend Framewoek** that uses PHP5 scripting)
Hint information	supported with **hint** element	supported with **placeholder** attribute
Dependency update feature	supported with **incremental** attribute	not supported
Upload	supported with **upload** element	supported with **type** attribute
Spell checking	not directly supported (XForms implementation **Orbeon Forms** provides this feature)	supported with **spellcheck** attribute
Autocomplete	not directly supported (XForms implementation **Orbeom Forms** provides this feature)	supported with **autocomplete** attribute

Table 3.4: Xfroms and HTML5 Forms supported and non-supported features

3.2 Related research

This section reviews some research conducted to study and improve electronic forms and related technologies. The first [20] proposed a prototype system that uses metadata database and XML technologies to automate a dynamic electronic form. The second [27] produced a demonstration system to explore the effectiveness of using XForms technology in communication in public health. In addition, it discusses the advantages of XForms, its advanced features, and its suitability for using in public health. The third [18] focused on presenting two concepts: enabling creation of forms that include access rules which control read and update rights to data objects and allowing an automatic derivation of workflow process from access rules that are contained in the form. The fourth [15] studied the possibility of adding JSON support in XForms data instances, where implementation is performed using a client-side implementation of XForms (XSLTForms) and an external JSON API in XForms page. The fifth [26] presented an MVC web framework based on XML technologies. The next [33] research designed a markup declarative language called **XFormsDB** to implement **XFormsDB** framework that is used for developing interactive multi-user applications. Finally, a research paper [32] introduced a new semi-automatic transformation method to generate XForms + XHTML document from an input XML schema.

3.2.1 Generation and validation of web forms using database metadata and xforms.

Elsheh and Ridley [20] proposed a system that generates automatically an XML document using a metadata SQL database, and then transforming that XML document into XHTML on the server side or XForms on the client side using XSLT stylesheets.

The system was developed using Java Servelt and XML technologies such as XSLT, XPath and CCS. The database was created with PostgreSQL and MySQL DBMSs, and JDBC was used to connect the database and express database metadata. Each XML node is generated from database metadata for each single column in the database tables. The generated XML document is transformed to a dynamic XForms document (each XML node is transformed to a specific electronic form element) using two templates of XSLT. One template is used to copy the XML document to the instance data in the produced XForms model, and the other one is used for XForms element binding. The limitation of this research is that it was implemented for generating a simple XML schema, which is transformed to a simple XForms.

This work presented an important improvement in electronic forms technologies by using W3C standards such as XML, XSLT and XPath. Thus, this research is considered one of the major contributions that related to my research.

3.2.2 Applying the xforms standard to public health case reporting and alerting

A demonstration project [27] is implemented to illustrate the convenience of using XForms for information exchange between provider and public health systems for notifiable condition reporting and alerting. The developed project benefits of using Retrieve Forms for Data Capture RFD (a method that is used for gathering data within one system to meet the requirements of an external system and enabling interoperability with other implemented RFD systems) with XForms, where XForms has an XML feature that is used to define form data, store and transport form instance data. Moreover, it has advanced form controls that ease form development and reduce the need for scripting.

However, the project had limitations in implementing XForms. The first limitation is that XForms does not have native support for web browsers and need for plug-in installation for the common browsers such as Internet Explorer and Mozilla Firefox. The second is that hand-coding XForms is considered time-consuming where there is no an editor providing sufficient support for XForms.

The research claimed that XForms is still immature to be used for notifiable condition reporting and patient-specific public health alerting. It debated that HTMl5 does not replace XForms, while HTML5 provides simpler support for more complex multimedia form elements. Both of them (HTML5 and XForms) support advanced form validation and controls.

This research described how XForms can be used to improve the electronic form system in public health. In addition, it produced a contributed discussion about Xforms that is related to my research.

3.2.3 Peer-to-peer form based web information systems

A research paper [18] produced a type of peer-to-peer web information system that enables users to create electronic forms and use them to capture data to be stored in a distributed database. Users also with this system are able to generate reports from their own forms or their peers‘ forms if they have right to access them. In addition, the system constructs a graph that represents a workflow to allow the system automating the workflow process depending on the access rights which associated with the forms.

XForms is used in developing the proposed system. XForms has many features such as using XML output, client-side validation and Model-View-Controller (MVC) that helped to address the practical problems that are associated with using HTML in designing the system. The pro-

posed system added some fundamental features that were not present in XForms:

(a) **Database connection**: The proposed system enables users to create an XML schema or extend other forms's schema, as well as allowing read and write access to the database, while XForms require an existed XML schema to read and write data to and from the database.

(b) **Access rules**: access rules in the proposed system are included explicity in the form definition, whereas XForms does not include any access rules to control access rights to data stored in an XML document or a database.

(c) **Workflow design**: The proposed system allows derivation of a workflow process using the access rules that are included in the form definition, whilst the idea of workflow is not associated with XForms.

The research approached an electronic form system with an improved workflow and access rules. One of the major W3C standards (XForms) is used to develop the system. Furthermore, it suggested some features that could be added to XForms, which is one of the related technologies that is explored in my research.

3.2.4 Json for xforms

Couthures [15] presented an implementation using XSLTForms to access JavaScript Object Notation (JSON) data on different domains via JSONP (JSONwith padding is a script that is used to request data from a server on various domains) without cross-domain limitations. This proposed implementation enables storing JSON objects into XML documents and allowing the use of XPath by adding a minimal number of extra attributes and elements that should be named without conflicting with ordinary JSON names.

JSONP support is implemented in twofold:

- **Request submission**: using the **script** element within an HTML document to execute and specify the **src** attribute that allows adding parameters. The unlimited cross-domain allows loading external JavaScript libraries. A function with **Callback** name is sent to the server to be integrated it in its response.
- **Response processing**: the response is the parameter of the **callback** function call which is performed by the received JavaScript source. This function is responsible to store the received JSON data object into its internal XML document.

This work has a contribution to the electronic forms development field by adding JSON support to the XForms, which is a fundamental state of electronic forms technologies in my research.

3.2.5 MVC web framework based on eXist application server and XRX architecture

This research paper [26] discussed the creation of a prototype MVC web framework application (named **iu5**) based on using a native XML database (eXist-db software), XForms REST XQuery (XRX) architecture and a client-side implementation of XForms (XSLTForms) that is embedded in eXistdb. Moreover, it developed a simple XML data module for data storing and an XQuery generator for the application infrastructure.

XSLTForms was used in the proposed application because it is flexible and simple to work with dynamic created XForms documents. XSlT-Forms XForms processor receives XForms document from a web server and converts it to HTML + CSS + JavaScript document. It performs data validation and then sends an XML-fragment back to the web server. The XQuery script runs on the web server and is responsible to send XForms documents to the XSLTForms processor on eXist-db and also sends and receives XML data to and from XSLTForms.

The research is relevant to my work because it contributed to the electronic forms developing domain by presenting an MVC framework application. This prototype system is developed by using eXist-db system, which is detailed later in this thesis and based on W3C standards such as XSLTForms (client-side implementation of XForms), XRX and XML.

3.2.6 XFormsDBAn XForms-Based Framework for Simplifying Web Application Development

A Master‘s Thesis [33] proposed developing an extension of XForms called **XFormsDB** declarative markup language, implementing a prototype and extensible XRX framework called **XFormsDB framework** which is powerful and useful for designing multi-user applications, and finally validating the developed language and the implemented framework by designing two web applications samples. This project is released as an open source application and can be found on the web site (https://code.google.com/p/xformsdb/).

The XFormsDB markup language is a declarative programming language. This language has a syntax and processing model that are similar to XForms with including to a few additions. For example, XFormsdb language supports authentication (by adding login and logout elements, and user requests) and access control (by adding xformsdb:secview and xformsdb_users.xml document).

The XFormsDB framework is a comprehensive web server that is implemented to demonstrate and prove the feasibility and capabilities of

the XFormsDB markup language. It is developed with several open-source application software and libraries: Apache ANT, Apache Tomcat, Eclipse IDE for Java EE Developers, eXist-db, Orbeon Forms and 3DM 0.1.5beta1. XFormsDB framework web pages are designed using different Web standards and technologies: XHTML, XForms, XFormsDB, XML, CSS, XQuery, XPath and JavaScript. The XFormsDB framework supports three major types of web browsers by providing each one of them with a devoted web page at the same time: a web browser with XForms1.1, a web browser with AJAX support and a web browser with a (X)HTML support. The persistent data is stored in two various types of XML-based: XML document and eXist-db Native XML Data (NXD). XFormsDB framework performs two separate server-side transformation process. The first is for transforming XHTML+XFormsDB to XHTML+XForms 1.1 and performed by the XFormsDBTransformer class that is divided to seven basic steps and consists of four XSLT transformations. The second transformation process is performed using a server-side XForms implementation (Orbeon Forms), it transforms the output of the first transformation process XHTML + XForms1.1 document to XHTML + CSS +JavaScript. XFormsDB framework uses the declarative XFormsDB markup language to help web developers and non-programmers users to develop interactive web application.

Two sample multi-user web applications are designed with XFormsDB framework. Those applications are used to verify the capabilities of the XFormsDB. The first application sample is a Personal Information Management (PIM): **Contacts** which is a simple web application that allows users to store, view and managing their personal information. This web application is developed to test the ease of using specific user interfaces functions such as storing and update data. The second application sample is Personal Information Management (PIM): **Blog** which is an online journal for publishing personal information, such as thoughts, comments, and experiences. This web application is developed to verify the capability of XFormsDB for designing current real-life web applications, including multiple web pages and complex data source queries.

There are a few limitations that indicated with this thesis. Firstly, synchronized updates issue in the XFormsDB framework. Performing simple insert and delete operations in a large XML document requires submitting back and forth, as well as updating an XML document may disclose sensitive information to the users. Secondly, the update query request of the XFormsDB may result in a large XML fragment because the used XPath expression must reference to the root element. Finally, transactions cannot be grouped and are a bit slow, where submitting any request depends on the successful submitting of the previous one.

This thesis presented an XRX framework application that is registered as an open source project, which has an essential contribution in elec-

tronic forms developing. It developed a declarative markup language (XFromsDB) by using some of the technologies that are stated in my research such as XForms, eXist-db and Orbeon Forms.

3.2.7 Generating XForms from an XML schema

This research [32] implemented an XForms editor that automates a translation of a given XML schema (XSD) to an XForms form. It performs a semi-automatic transformation algorithm that allows the form developer to decide which elements and attributes declarations will be included in the form and how they will be displayed, collected together and validated. This algorithm is implemented using an object-oriented language that based on XML parsing technologies such as DOM.

The proposed semi-transformation algorithm consists of two phases. The first algorithm phase traverses the input XML schema from the root to the leaves to derive an XML document, and then generating an XForms instance node that includes all the possible XML elements and attributes from the given XML schema. The second algorithm phase traverses the derived XForms instance in the first phase from the root to the leaves. It creates the binding parts and UI components based on types of nodes in the given XML schema whether simple types or complex types.

This work is an algorithm that contributed in the electronic forms improvements by automating XForms form based on a given XML schema. Xforms is an electronic form technology which one of W3C recommendations based a W3C standard XMl, and explored with different implementations in my research.

3.3 Summary

This chapter involved technologies and researches that are related to the electronic forms development and based on W3C Standards. Firstly, it reviewed the primary related technologies (XForms and HTML5 Forms) that are based on W3C standards and have an essential contribution to electronic forms developing. Secondly, it discussed related work about electronic forms enhancement.

Chapter 4

Electronic forms applications

This chapter describes some major software that use electronic forms technologies based on W3C standards such as XForms and XML technologies. These software are experimented and tested. The following sections represent the software, their characteristics and functionalities, details of the installation and the supported form features. The following software are introduced respectively: AJAXForms, eXist-db (XSLTForms and betterForm), Orbeon Forms and finally InfoPath.

4.1 AJAXForms Project

AJAXForms is a server-side implementation of XForms 1.0. It is a JAVA program that transforms XHTML/Xforms documents to HTML/JavaScrip and it is supported by just Firefox, Opera and IE browsers. The transformation is made at compile time and transforms in the client, where the client and server exchange the model data trough XML documents using AJAX [1].

AJAXForms is mainly based on XSLT 1.0 transformation in addition to using an XPath engine (Jaxen) to perform XPath expressions [6]. This program consists of two components: [1]

- A compiler that performs the transformation.
- JavaScript library to be used by the compiled files.

4.1.1 Installation

AJAXForms requires the following software:

- **(JRE 5.0).** Java Runtime Environment version 5.
- **Apache Ant**:
  It is an open source Java and command-line tool that based on XML. It uses an XML file (that has a default name buld.xml) to

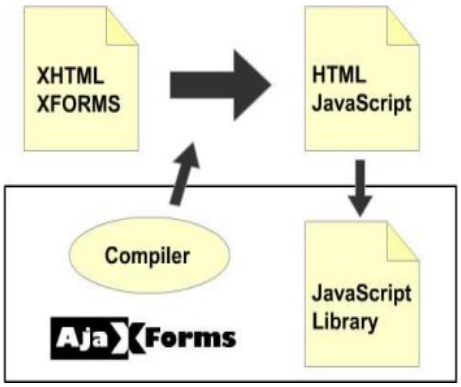


Figure 4.1: sourced from [1]

build, compile and run Java applications. Ant requires Java Development Kit (JDK), Where the required version of Java depending on the Ant version. It supports Linux, MacOS X, Windows XP and Unix platforms.
I installed apache-ant-1.8.4 that requires JDK 1.4 or higher on Windows7 machine.

Installing Ant:

(a) The downloaded file is extracted into the directory (apache-ant-1.8.4.

(b) setting the required environmental variables: JAVA_HOME to **JDK1.5** directory, ANT_HOME to **apache-ant-1.8.4** directory and then adding ANT_HOME/bin to the current Path.

(c) Some Ant tasks requires the library dependencies that is obtained by running ant -f fetch.xml -Ddest=system.

(d) Building Ant in the directory apache-ant-1.8.4 by running: build -Ddist.dir= -ant-1.8.4 >dist

Installing AJAXForms
I downloaed the zipped template from (`http://ajaxforms.sourceforge.net/docs/comousar.html`), and then unzipped into **ajaxform** directory in **apache-ant-1.8.4 directory**.

Runing AJAXForms with Ant
Build.xml and build.properties are used by the apache ant to build the application and they reside in ajaxforms directory. Lib directory contains the required libraries.
Any web content or XForms files are stored in web directory.
XForms documents are transformed using **ant** command that uses build.xml by default.
The compiled files are built in the **build** directory.

I had an issue with using AJAXForms application. Where apache ant works well but when running **ant** command, I have an error with the transformation. As a result, the XForms content in the resulted web pages are not displayed. I could not fix this issue and I could not find any support to fix it, because there is no activity or support for AJAXForms application since 2006.

4.1.2 Form supported features in AJAXForms

Form features in AJAXForms are tested based on the online samples that are running on the AJAXForms web site (http://ajaxforms.sourceforge.net/index.html) and investigating the code of the examples that are shipped with AJAX-Forms template. The following sections demonstrate the form feature that are provided and not provided by AJAXforms.

4.1.2.1 Data validation

This feature is tested using an online form sample **bind**, and it seems that AJAXForms supports validation.

```
<xf:model>
<xf:instance>
<root type="mal">
<value type="value2">value1</value>
</root>
</xf:instance>
<xf:bind id="text" nodeset="value" required="true()">
<xf:bind id="type" nodeset="@type" required="true()"/>
</xf:bind>
</xf:model>
```

code source: `http://ajaxforms.sourceforge.net/samples/bind.xform`

4.1.2.2 Dynamic form modification

Based on testing the online form samples **Balance and Bookmarks**, It seems that this feature is supported by AJAXForms.

```
<xf:repeat id="repeatBookmarks" nodeset="bookmark" appearance="compact">
...
<xf:trigger>
<xf:label>Delete</xf:label>
<xf:delete nodeset="." at="1" ev:event="DOMActivate"
if="count(//section) > 1" />
</xf:trigger>
</xf:repeat>
<xf:trigger>
<xf:label>New section</xf:label>
<xf:insert nodeset="//section" at="last()" position="after"
ev:event="DOMActivate" />
```

```
</xf:trigger>
```

code source: `http://ajaxforms.sourceforge.net/samples/bookmarks.xform`

4.1.2.3 calculation

Calculation feature is tested using **calculator** sample on the web site. However some operations in this example are not working properly, it seems that is supported by AJAXForms.

```
<xf:trigger>
<xf:label>=</xf:label>
<xf:action ev:event="DOMActivate">
<xf:setvalue ref="/equation/second" value="/equation/screen"/>
<xf:setvalue ref="/equation/result" value="/equation/first - /equation/second"/>
<xf:setvalue ref="/equation/screen" value="/equation/result"/>
<xf:setvalue ref="/equation/screenbuffer" value="0"/>
</xf:action>
</xf:trigger>
</xf:case>
<xf:case id="multiply" selected="false">
<xf:trigger>
<xf:label>=</xf:label>
<xf:action ev:event="DOMActivate">
<xf:setvalue ref="/equation/second" value="/equation/screen"/>
<xf:setvalue ref="/equation/result" value="/equation/first * /equation/second"/>
<xf:setvalue ref="/equation/screen" value="/equation/result"/>
<xf:setvalue ref="/equation/screenbuffer" value="0"/>
</xf:action>
</xf:trigger>
```

code source: `http://ajaxforms.sourceforge.net/samples/calculator.xform`

4.1.2.4 Data storage

AJAXForms template is shipped with **index.xform** example. This example enables to get and put in an XML file **data.xml**. Thus, AJAXForms supports XML data storage.

4.1.2.5 Access Rights control

I could not find any example or reference to verify which type of access control is supported by AJAXForms.

4.1.2.6 SVG feature

There is no example or reference is found to verify if this feature is supported or not by AJAXForms.

4.1.2.7 subforms feature

AJAXforms web site has an example **Repeat** that shows how to implement subforms in AjAXForms.

```
<xf:model>
<xf:instance>
<data select="colors">
<items name="colors">
<item value="red">red</item>
<item value="green">green</item>
<item value="blue">blue</item>
</items>
<items name="numbers">
<item value="one">one</item>
<item value="two">two</item>
<item value="three">three</item>
<item value="four">four</item>
</items>
</data>
</xf:instance>
</xf:model>
...
<xf:repeat nodeset="items[@name = /data/@select]/item" id="repeat"
appearance="compact">

<xf:switch>
<xf:case id="input">
<xf:input ref="."><xf:label>Input</xf:label></xf:input>
<xf:trigger>
<xf:label>Output mode</xf:label>
<xf:toggle case="output" ev:event="DOMActivate"/>
</xf:trigger>
</xf:case>
<xf:case id="output">
<xf:output ref="."><xf:label>Output</xf:label></xf:output>
<xf:trigger>
<xf:label>Input mode</xf:label>
<xf:toggle case="input" ev:event="DOMActivate"/>
</xf:trigger>
</xf:case>
</xf:switch>
```

code source: `http://ajaxforms.sourceforge.net/samples/repeat.xform`

4.1.2.8 Hint information feature

The online sample **Actions** demonstrates that hint information feature is provided by AJAXForms.

```
<xf:help>
This action causes the processing of xforms-rebuild
to happen, bypassing the normal event flow.
<ul> <li>model</li></ul>
</xf:help>
```

code source: `http://ajaxforms.sourceforge.net/samples/actions.xform`

4.1.2.9 Dependency update feature

The online sample form **Incremental** illustrates that this feature is implemented using **incremental** attribute.

```
<xf:label>Incremental</xf:label>
<xf:input ref="text" incremental="true"><xf:label>Input</xf:label></xf:input>
<xf:select ref="select" incremental="true">
```

code source: `http://ajaxforms.sourceforge.net/samples/incremental.xform`

4.1.2.10 Upload feature

No example on the web site or shipped with the AJAXForms, and I could not find another reference to verify if this feature is provided by AJAXForms.

4.1.2.11 Spell checking

I could not find if this feature is provided by AJAXForms because the project is not working properly as mentioned in section 1.1. As well as there is no samples on the web site or references about this feature and AJAXForms.

4.1.2.12 Autocomplet feature

Even this feature is not verified if supported or not supported by AJAX-Forms, where no samples are provided on the website and I could not find any reference about that.

4.1.3 Why is AJAXForms involved in this research

AJAXForms has no development and activity since 2006. However, it was a starting point to develop another application called XSLTForms project. XSLTForms is implemented as client-side in Exist-db that is detailed in the next section. AJAXForms is the project ancestor of XSLTForms, where XSLTForms adds the following changes: [7]

(a) There is a unique XSLT stylesheet for each Java source code, html documents have IDS that generated based on their tag name and position.

(b) XSLT must transform Xpath expressions in JavaScript.

(c) Full xforms1.1 support.

(d) Full XML schema support while AJAXForms only support simple types of XML schema [1].

4.2 eXist-db

eXist-db is an open source xml database management system(native xml database). It is built based on Java and it can be embedded inside other applications or run as stand-alone server. [38] Exit-db is a powerful environment that supports web2.0 technology standard such as: XSLT, XPath, XQuery, and HTTP interfaces. It supports two types of xforms implantations: client-side implementation (xsltForms) (see section 1.4) and server-side implantation (Betterform) (see section 1.5). [21]

4.2.1 eXist-db data storage

eXist-db has hierarchical collections of schema-less storage of xml documents. Users can query various parts of these collections or all documents that are stored in the database using an extended XPath syntax. Most of path expression queries are only processed using index information Based on path join algorithms. [21]

Using eXist database system is by logging in to the eXist database system administration from the local eXist homepage (web-based interface) and then select Admin from Administration section to enter the appropriate username and password. In addition, Exist-db provides a GUI admin client based on Java which allows users to perform administrative tasks such as security management. eXist-db system includes some simple XQuery web applications that help users to experiment and query the eXist-db database using sample XML documents. [22]

4.2.2 eXist-db security

eXist-db database system supports access control management for users and groups based on UNIX permissions that are read, write and execute (which is update for exist-db) for owner, group and others. Users‘ accounts information (e.g usernames and passwords) and groups are stored in the database (in the xml file /db/system/users.xml located in collection/db/system and generated during database setup). [13] [24]

Each database user has one or more group, and each resource is owned by one user and a group (who created it). The owner, group and others may have different permissions under eXist-db database and only the owner who is able to change these permissions. [13] [24]

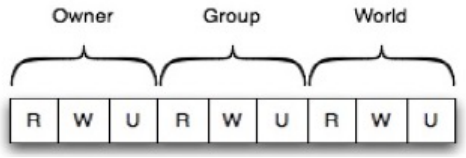


Figure 4.2: permission categories(reprinted from [13])

Category	Description
Owner	These permissions for the owner of a resource
Group	These permissions for members of the group of a resource
World	These permissions for other users

Table 4.1: reprinted from [13]

4.2.3 Installation

All recent versions of Linux, Mac OS X, and Windows are compitable with eXist-db. It needs at least 512 MB of RAM and about 200 MB of disk space and requires Java 6 or 7 either Java Runtime Environment (JRE) or Java Development Kit (JDK). [22]
I used version 1.6 of Java Development Kit (JDK) and downloaded the latest stable release of eXistdb installer from existdb web site (`http://exist-db.org`) and started installation on a Linux virtual machine.

The following command is used to start the installer on Linux:

java -jar eXist-setup-1.4.2-rev16251.jar

The installation asks the user to enter a password for the eXist adminstrator user (admin).

Before starting running eXistdb, there are some environmental variables should be set:
Export JAVA_HOME=/home/jdk1.6.0-21
Export PATH=PATH:/home/jdk1.6.0-21

After finishing eXist-db installation, it starts running using its embedded Jetty webserver (jetty is an open source HTTP server and servelt container based on Java) by using the default port number 8080.

Start running eXist-db is by launch the file (startup.sh) in (bin) eXist directory path and stop it is by launch the (shutdown.sh) in the same directory.
[22]
Entering http://localhost:8080/exist/index.xml in the web browser opens the eXist-db local homepage if the server runs correctly.

4.2.4 client-side functionality (xsltForms)

xsltForms is an open source engine (LGPL license). It is a client-side implementation of XForms 1.1 to convert xForms to XHTML + JavaScript (AJAX) using XSLT transformation, where there is no need for plugins installation. eXist-db is shipped with xsltForms package that includes a script library to generate an optimized AJAX code. Any designed xsltForms are stored in eXist/webapp/xforms path.

4.2.4.1 Supported form features by xsltForms

This section outlines the form features that are supported and not supported by XSLTForms. These features are verified and tested using some form samples that are implemented eXist-db software.

4.2.4.1.1 Data validation

This feature is tested and found that it is supported by eXist-db on the client-side (xsltForms).

XsltForms support the same validation rules in xforms (see chapter3 XForms section for tested examples) [16]. However, most xsltForms validation is using bind method because using validation based on xml schema is more complex for xsltForms [60].

4.2.4.1.2 Dynamic form modification

Dynamic adding and removing form fields are tested on eXist-db client-side (xsltForms) and found that they are supported.
For example:
The following part of code for deleting

```
<trigger>
<label>X</label>
<delete nodeset="." at="1" if="count(//telephone) > 1"
ev:event="DOMActivate" />
</trigger>
```

The following part of code for adding

```
<trigger>
<label>New</label>
<insert nodeset="telephones/telephone" at="index('repeat')"
position="after" ev:event="DOMActivate" />
</trigger>
```

code source: `http://www.agencexml.com/xsltforms/xf.xml`

4.2.4.1.3 Calculation

xsltForms support the same calculation ways that are provided by XFroms :
For example:
The following calculation example is by using **bind** element and **calculate** attribute:

```
<xf:bind nodeset="Results/sum" calculate="sum(../../Row/A)"
type="xsd:decimal" />
<xf:bind nodeset="Results/avg" calculate="avg(../../Row/B)"
type="xsd:decimal" />
<xf:bind nodeset="Results/min" calculate="min(../../Row/C)"
type="xsd:decimal" />
```

[8] code source: `http://www.agencexml.com/xsltforms/spreadsheet.xml`

The following calculation example is by using **toggle** element.

```
<xf:trigger>
<xf:label>+</xf:label>
<xf:action ev:event="DOMActivate">
<xf:setvalue ref="first" value="../screen"/>
<xf:setvalue ref="screenbuffer" value="0"/>

<xf:toggle case="add"/>
</xf:action>
</xf:trigger>
</td>
<td>
<xf:switch>
<xf:case id="add" selected="true">
<xf:trigger>
<xf:label>=</xf:label>
<xf:action ev:event="DOMActivate">
<xf:setvalue ref="second" value="../screen"/>
<xf:setvalue ref="result" value="../first + ../second"/>
<xf:setvalue ref="screen" value="../result"/>
<xf:setvalue ref="screenbuffer" value="0"/>
</xf:action>
</xf:trigger></xf:case>
```

code source: `http://www.agencexml.com/xsltforms/calculator.xml`

4.2.4.1.4 Data storage

XsltForms supports XML database. An xsltform is tested to generate, get and put data in an XML database under Exist-db database system. The tested example **hello2.xml** is shipped with eXist-db package.

```
<xf:model>
<!-- Try to load the default instance from the db. If no person exists in  the db,
the query returns an empty <person/> element. Without that we
would receive an error. -->
<xf:instance id="default" src="../rest/db?_query=&lt;person&gt;
{/person/name}&lt;/person&gt;
&amp;_wrap=no"/>

<!-- Template data: will be copied into the default instance if the db does not
yet contain a person. -->
<xf:instance id="template">
<person xmlns="">
<name></name>
</person>
</xf:instance>

<xf:submission id="put-to-db" method="put" replace="none"
  action="../rest/db/simple-xforms.xhtml">
<xf:toggle case="case-busy" ev:event="xforms-submit"/>
<xf:toggle case="case-submit-done" ev:event="xforms-submit-done"/>
<xf:toggle case="case-submit-error" ev:event="xforms-submit-error"/>
</xf:submission>

<xf:submission id="get-from-db" method="get" replace="instance"
    action="../rest/db?_query=/person&amp;_wrap=no">
<xf:toggle case="case-busy" ev:event="xforms-submit"/>
<xf:toggle case="case-submit-error" ev:event="xforms-submit-error"/>
<xf:toggle case="case-submit-done" ev:event="xforms-submit-done"/>
</xf:submission>
```

4.2.4.1.5 access rights control

Access control is supported by xsltForms weather by using a traditional username and password form or when the form accesses a secure resource in the database to get or put data. In the last case, the eXist-db asks users to prove their credentials (username and password), where eXist has a Cocoon (it is an open source project based on Java and is used to build dynamic internet applications, it relies on using XML and XSLT transformation [34]) action that presents a login interface page. This login is not required each time because the entered username and password are stored in a created HTTP session [24].

4.2.4.1.6 SVG feature

xsltForms supports using SVG to capture data in the form. The following part of code is used to test this feature.

```
//this function handles the keyboard events
```

```
function typeText(evt) {
//handle "keypress" for all "real characters"
if (evt.type == "keypress") {
//some browsers support evt.charCode, some only evt.keyCode
if (evt.charCode) {
var charCode = evt.charCode;
}else {
var charCode = evt.keyCode;
}
//all real characters
if (charCode > 31 && charCode != 127 && charCode < 65535) {
text += String.fromCharCode(charCode);}
//backspace key
if (charCode == 8) {
//shorten text
text = text.substring(0,text.length-1);}
//enter key
else if (charCode == 10 || charCode == 13) {
stopTyping(evt);}}
...
<rect id="bgRect" x="-500" y="-500" width="1500" height="1500" fill="yellow"
stroke="none" />
<rect x="10" y="50" width="150" height="20" fill="white" stroke="black"
 onclick="initTyping(evt)" />
<g font-size="10px" font-family="Arial">
<text x="10" y="30" font-size="12px" font-weight="bold">Demonstration of adding
and removing Event Listeners</text>
<text x="10" y="130">Click in the white rectangle to start typing text.
<tspan x="10" dy="15">Click outside the textbox or press enter to stop writing
text.</tspan>
<tspan x="10" dy="15">Press backspace to delete the last character when typing
is active</tspan></text>
```

code source: `http://www.carto.net/svg/eventhandling/`

4.2.4.1.7 subforms feature

Subform is supported by xsltForms and tested using the following example:

```
<h3>Writers (Subforms)</h3>
<xforms:repeat nodeset="writer" appearance="compact">
<p>
<xforms:output ref="@firstname"/> <xforms:output ref="@lastname"/> 
<xforms:trigger>
<xforms:label><xforms:output value="choose(@selected != 'true','Show','Hide')"/>
Books</xforms:label>
<xforms:action ev:event="DOMActivate">
<xforms:load if="@selected != 'true'" show="embed" targetid="subform"
resource="books.xml"/>
```

```
<xforms:unload if="@selected = 'true'" targetid="subform"/>
<xforms:setvalue ref="@selected" value=". != 'true'"/>
</xforms:action>
</xforms:trigger>
<xforms:group id="subform"/>
</p>
</xforms:repeat>
```

code source: `http://www.agencexml.com/xsltforms/writers.xml`

4.2.4.1.8 Hint information feature

This feature is tested and found that is provided by xsltForms.

```
<p>Enter your first name, and last name.</p>
 <xf:input ref="PersonGivenName" incremental="true">
 <xf:label>Input First-Name:</xf:label>
 <xf:hint>Also known as given name.</xf:hint>
 </xf:input>
```

code source: `http://www.agencexml.com/xsltforms/input.xml`

4.2.4.1.9 Dependency update feature

This feature is enabled in xsltForms using **incremental** attribute for an element. This attribute reflects any update or change in that element on another element.

```
<xf:input ref="PersonSurName" incremental="true">
 <xf:label>Input Last Name:</xf:label>
  <xf:hint>Also known as sur name or family name.</xf:hint>
   </xf:input>
   Output Last Name: <b><xf:output ref="PersonSurName"/></b>
```

code source: `http://www.agencexml.com/xsltforms/input.xml`

4.2.4.1.10 upload feature

This feature is not supported by xsltForms.

4.2.4.1.11 Spell checking

This feature is not provided by XSLTForms.

4.2.4.1.12 Autocomplete feature

Autocomplete feature is verified and found that is not supported by XSLT-Forms.

4.2.5 Server-side functionality (betterFORM)

Betterform is a declarative server-side implantation of xforms 1.1 that is compatible with all modern browsers without plugins installation. It transforms XForms documents on the server to XHTML + Javascript (AJAX) to provide the user with an attractive interface without the need for writing any script code. betterFORM is activated once the eXist-db installed. Furthermore, it is configured by default to execute XForms documents that are stored in the database weather using the web interface or the admin client interface [23].

betterFORM has add-ons that help to ease working with XForms: [23]

- **Dashboard** is a simple browser to the database. It enables the user to create new collections or upload files into the database.
- **FeatureExplorer**: It is a live documentation of XForms. It eases working with XForms by providing many working examples along with relevant links to the XForms specifications. Moreover, it offers a live CSS reference that helps with styling forms.
- **TimeTracker**: It is a small application that gives the user some hints how to write a single-page application with eXist-db and Betterform.

4.2.5.1 Supported form features in betterFORM

This section verifies which form features are provided and not provided by betterFORM using eXist-db software.

4.2.5.1.1 Data Validation

Data validation is supported by betterFORM by using the validation rules of XForms(see chapter3 XForms section). The server-side implementation of XForms (betterFORM) makes the validation more secure and reliable.

4.2.5.1.2 Dynamic form modification

betterFORM supports adding and removing form elements that are tested using the same previous example in section 3.2.4.1.2.

4.2.5.1.3 Calculation

betterFORM supports calculation form feature. This feature is tested using the same examples in section 3.2.4.1.3.

4.2.5.1.4 DataStorage

betterFORM can get, put data and generate an XML document and save it into eXist-db database.
The following example from Wikipedia is used to test this feature:

```
<xf:model>
<xf:instance id="data-instance" src="data.xml" xmlns="" />
<xf:submission id="read-from-file" method="get"
 action="data.xml" replace="instance" instance="data-instance" />
 <xf:submission id="save-to-file" method="put"
 action="data.xml" replace="instance" instance="data-instance" />
  </xf:model>
  </head>
  <body>
  <xf:input ref="Element1">
  <xf:label>Elementu 1:</xf:label>
  </xf:input>
  <br />
    ...
  <xf:submit submission="read-from-file">
  <xf:label>Reload</xf:label>
  </xf:submit>
  <xf:submit submission="save-to-file">
  <xf:label>Save</xf:label>
  </xf:submit>
```

code source: `http://en.wikibooks.org/wiki/XForms/Read_and_write_with_get_and_put`

4.2.5.1.5 Access rights control

betterFORM files are stored in a collection called (betterform) in the eXist-db database. So, any access to this collection including the access to the dashboard requires users to enter their credentials depending on their roles and groups. In this case, Cocoon(see section 1.4.5) login interface is represented by the eXist-db system. In addition, a form may developed under betterFORM to be used as a login interface (with username and password) to control access to other resources.

4.2.5.1.6 Svg feature

Many SVG with XForms examples are tested and they are not displayed. Thus, it seems that this feature is not supported by betterFORM.

4.2.5.1.7 subforms feature

betterFORM supports subformms. Contact example (shipped with Betterform package) is tested to illustrate subforms and cross-module submission in Betterform. Cross-module submission allows passing instances between subforms. It could be used to reuse the validation constraints between modules to prevent duplicating the bind constraints.

```
<xf:submission id="update-subform" resource="model:address#instance('default')
/address" method="post" replace="none" ref="address[index('addresses')]">
<!--<xf:message ev:event="xforms-submit-done" level="ephemeral">
Masterform has updated Subform.
</xf:message>-->
</xf:submission>
...

<xf:submission id="s-update-master" resource="model:master#instance('contact')
/data/address[index('addresses')]" replace="none" method="post">
<xf:message ev:event="xforms-submit-done" level="ephemeral">Subform has updated
Master.</xf:message>
</xf:submission>
```

4.2.5.1.8 Hint information

Hint information feature is provided by betterFORM and is tested using the same example in section 3.2.4.1.8.

4.2.5.1.9 Dependency update feature

Incremental attribute is used to enable this feature in betterFORM (see section 3.2.4.1.9 for tested example).

4.2.5.1.10 upload feature

betterFORM allows upload feature and is tested using the following example:

```
<xforms:instance xmlns="">
<data>
<attachment1 xsi:type="xsd:anyURI"/>
<attachment2 xsi:type="xsd:hexBinary"/>
</data>
</xforms:instance>
</xforms:model>
...
<xforms:upload ref="attachment1" incremental="true">
<xforms:label>Upload for anyURI: </xforms:label>
</xforms:upload>
<br/>
<xforms:output value="attachment1">
<xforms:label>Value: </xforms:label>
</xforms:output>
...
<xforms:upload ref="attachment2" incremental="true">
<xforms:label>Upload for hexBinary: </xforms:label>
</xforms:upload>
<br/>
```

```
<xforms:output value="attachment2">
<xforms:label>Value: </xforms:label>
</xforms:output>
```

code source: `http://www.agencexml.com/xsltforms/uploads.xml`

4.2.5.1.11 Spell checking

betterFORM does not provide any control or component that supports this feature..

4.2.5.1.12 Autocomplete feature

This feature is not supported by betterFORM.

4.3 Orbeon Forms

Orbeon PresentationServer (OPS) is an open source Ajax-based XForms engine, so no need for client-side plugins installation. Orbean form is based on XML technology and supports XForms1.1 specification. It includes an XML pipeline engine that processing XML data using XML pipeline language (XPL)(XPL is a declarative language that is built up from small components called **XML processors**. XPL is used to process XML documents via those XML processors. In addition, it is used to perform advanced functions such as document aggregation, conditionals ("if" conditions), loops, schema validation, and sub-pipelines [54]). Moreover, Orbeon Forms performs XSLT transformations using built-in components and is shipped with examples and documentation that help the user to design web forms. Electronic forms could be programmed by writing required Xforms, XML and XSLT tags under orbeon form or built with a visual web-based means that is provided by orbeon form as well. [53]

4.3.1 Orbeon Forms builder

Orbeon form builder is a web-based application that provides a visual environment for building electronic forms. The following features of form builder help with designing web forms: [52]

- **Easy layout**: form builder has a clear and easy layout (sections and grids).
- **Data validation**: form builder supports data type validation, as well as it enables to import an XML schema for validation.
- **HTML and PDF output**: Forms that built with form builder can be automatically produced in HTML or PDF format.

- **Internationalization**: Form builder could be used to full design forms in multiple languages.
- **Form Runner**: Orbeon form runner is a runtime environment that is used to test designed forms.
- **XPath expressions**: It is an advanced feature of form builder that enables query XML data using control details dialog.
- **Services and actions**: Form builder provides an editor to implement simple action and call simple services.
  (a) **HTTP service**
  FormBuilder has a built-in HTTP service editor to create simple REST services that enable the form from calling the service to send and receive xml documents.
  (b) **Database service**
  It an editor that enables the user to create a template of SQL query or update. The datasource for this service is JDNI name(Java Naming and Directory Interface (JNDI) is Java API that enables applications from accessing directory services using a name such as db in the following example).
  Example:
  SELECT * FROM orbean address book WHERE id = <sql:para type="xs:string"select=""/>

  The parameters have to have placeholders in the form of XML elements. "type" is similar to SQL type.
  This quey template runs in the database and returns data in XML format, where names of database column are changed to lowercase (if not). The elements of the returned xml document as follows:
  - Root element <response >
  - <row>element for each row in the database.
  - Under <row >element there is an element for each column in the database.

4.3.2 Installation

Orbeon Forms needs at least 1GB of RAM and it may be installed on any recent versions of Mac OS, Linux or windows. Orbeon form requires Java5 or greater and a servelt1.4 container or greater.
I installed version 1.5 of Java development Kit (JDK) and apache-tomcat-6.0.24 using windows7 machine.

Installing apache tomcat

- Downloading apache-tomcat-6.0.24 zipped file from `http://tomcat.apache.org`, and extracting into **apache-tomcat-6.0.24** directory.
- setting the environmental variable **CATALINA_HOME** to the directory **apache-tomcat-6.0.024**.

Setup Orbeon Forms

- Downloading Orbeon form 3.9 from `http://www.orbeon.com/forms/download`.
- Extracting the archive into **orbeonform** directory.
- Copying the file **orbeon.war** from **orbeon form** directory into **tomcat_home/webapps**.

Start running Orbeon form

- starting tomcat by running the file **tomcat_home/bin/startup.bat**.
- Open the URL http://localhost:8080/orbeon to display Orbeon form welcome page.
- Shutdown.bat file stops Tomcat server.

I had an issue with Orbean Forms that gives the error **java heap space** error when I run any tool in Orbean form such as Form builder. This problem is fixed by setting the variable **CATALINA_OPTS** in apache-tomcat-6.0.24/bin/**catalina.bat** file.
set CATALINA_OPTS=-Xms512m -Xmx512m [30]

4.3.3 Supported form features by Orbeon Forms

The following sections experiment form samples with Orbeon Forms. These samples are designed with electronic forms features to check their availability and support by Orbeon Forms.

4.3.3.1 Data validation

Orbeon Forms supports data validation that is implemented with the same XForms validation rules. Validation may be programmed or implemented using Form builder. It is tested using **controls form** example that shipped with Orbeon Forms package.

4.3.3.2 Dynamic form modification

This feature is tested and found that is supported by Orbeon Forms. The following part of code (from bookcast example that shipped with Orbeon 521 Forms package) demonstrates adding and removing form elements.

```
<xforms:trigger appearance="minimal">
<xforms:label><img src="/apps/xforms-bookcast/images/add.gif"/>
</xforms:label>
<xforms:insert ev:event="DOMActivate" context="instance('books-instance')"
nodeset="book" at="1" position="before" origin="instance('book-template')"/>
</xforms:trigger></td>
```

```
<td class="add-td">
<xforms:trigger appearance="minimal">
<xforms:label>Add One</xforms:label>
<xforms:insert ev:event="DOMActivate" context="instance('books-instance')"
nodeset="book" at="1" position="before" origin="instance('book-template')"/>
</xforms:trigger></td></tr>
<xforms:repeat nodeset="book" id="book-repeat">
<tr><td>
<xforms:trigger appearance="minimal">
<xforms:delete ev:event="DOMActivate" context="instance('books-instance')"
nodeset="book" at="index('book-repeat')"/>
<xforms:label><img src="/apps/xforms-bookcast/images/remove.gif"/>
</xforms:label>
</xforms:trigger></td>
```

4.3.3.3 Calculation

Calculation feature is supported by Orbeon Forms. For tested examples see section 3.2.4.1.3.

4.3.3.4 SVG feature

Orbean Forms provides SVG feature and is tested using the example in section 3.2.4.1.6.

4.3.3.5 Data sorage

Orbeon Forms uses persistence API to store form definition and data weather in the same or different place. For example, form definition may be stored in eXist and form data stored in other different. Orbeon Forms is shipped with an embedded eXist database, where data is stored by default in eXist. In addition, It provides a built-in support implementation for a number of databases (see table below) and allows users to implement their own persistence API. [51]

Storage	Notes	Availability
eXist	A A full persistence layer implemented on top of the open source eXist database. You can setup this persistence layer to either use the internal eXist database, or an external one.	Built-in
Oracle	Afull persistence layer implemented on top of the Oracle database. Forms and data are stored using OracleXMLTypecolumns.	Built-in.
MySQL	Afull persistence layer implemented on top of MySQL.	Built-in

Alfresco	This persistence layer isn't bundled with Orbeon Forms: it has been developed byAlexey Ermakovand is available as an add-on that you can download and configure with Orbeon Forms.	Add-on
MongoDB	A full persistence layer implemented on top of MongoDB. NOTE: This is experimental as of 2011-04-26.	Built-in
File system (resource)	A minimal, read-only persistence layer implemented on top of Orbeon Forms' resource manager. This allows reading form definitions stored within the Orbeon Forms WAR file.	Built-in

Table 4.2: reprinted from [51]

4.3.3.6 Access rights control

Two types of access control are provided by Orbean Forms: [55, 50]

(a) **form builder access control**: that is role-based. Permissions and role are defined through a property file: form-builder-permissions.xml. This enables specify different permissions (create, edit, or view a form) for users depending on their roles.

(b) **Access control to deployed forms**: two levels of access control could be implemented:

- **Field level**: is implemented in the form definition using $frroles. This level of access control makes if a form field is visible or read only or not for the current user. The visibility and read only XPath expressions are defined in the Form Builder control validation dialog.
- **Form level**: controle acces to a form is based on path. For example,
  - If Orbean form is deployed on /orbeon and a form report is on /orbeon/fr/ then,
  - the path /orbeon/fr/report/new is for creating a new report and,
  - the path /orbeon/fr/report/edit/id is for editing a report.
  - Giving access to authenticated users to create new report is implemented on /orbeon/fr/report/new.
  - Giving access to authenticated users to edit a report is implemented on /orbeon/fr/report/edit/*.

Orbeon form 4.0 provides an improved option for access control. It enables to control role-based permissions in form builder under **Advanced** menu and **set permissions** option.

4.3.3.6.1 username and roles

Orbeon form automatically adds two headers to deployed forms in form runner:

- Orbeon-username: is one value that specifies the current user.
- Orbeon-roles: is a list of values that each one specifies one role.

Those headers are forwarded to form runner by XForms engine. Moreover, they are forwarded to the the persistence layer to be used in database operations.

4.3.3.7 subforms feature

Subforms feature is tested and found that is supported by Orbeon Forms.

```
<xf:model>
<xf:instance id="names" src="names.xml">
   <null/>
</xf:instance>
<xf:submission id="get-places" method="get" action="names.xml"
replace="instance" instance="names"/>

<xf:instance id="spaces">
<null/>
</xf:instance>
<xf:submission id="get-places" method="get" action="spaces.xml"
replace="instance" instance="spaces"/>
</xf:model>
<h2>Names</h2>
<xf:group ref="instance('Names')">
<xf:repeat nodeset="person">
<xf:output ref="name"/><br/>
</xf:repeat>
<xf:submit submission="get-names">
<xf:label>Load names</xf:label>
</xf:submit>
</xf:group>

 <h2>spaces</h2>
 <xf:group ref="instance('spaces')">
 <xf:repeat nodeset="spaces">
 <xf:output ref="name"/><br/>
 </xf:repeat>
 <xf:submit submission="get-spaces">
 <xf:label>Load spaces</xf:label>
 </xf:submit>
 </xf:group>
```

4.3.3.8 Hint information feature

The example in section 3.2.4.1.8 is used to test this feature and found that is provided by Orbeon Forms.

4.3.3.9 Dependency update feature

Like xsltForms and Betterform, this feature is supported by Orbeon Forms with using **incremental** attribute.(See section 3.2.4.1.9)

4.3.3.10 upload feature

Orbeon Forms supports upload feature and is tested with an example sipped with Orbean Forms package (xforms-upload directory).

4.3.3.11 spell checking

Orbeon Forms provides spell checker component that is bind to the node which contains the text to be checked (for example, text area control).

```
<xforms:textarea ref="."/>
<fr:spell-checker ref="."/>
```

The spell checker component is displayed on the form page as an icon next to the form field. This icon allows the user to open the spell checker icon dialog to correct spelling mistakes. [48]

4.3.3.12 Autocomplete feature

This feature is supported by Orbeon Forms. It allows adding an auotocomplete field to the field and gives the user a list of suggestions when typing a few characters [47].

The autocomplete component in Orbeon Forms is implemented in three modes: [47]

(a) **Static mode**: in this mode, the auotcomplete field shows a static list of suggestions (it is changed or obtained from other resource) depending on what the users type in it (It will only display items which label starts with the text entered by users).

Example:

```
<fr:autocomplete ref="country-name" dynamic-itemset="false">
<xforms:label>Enter a country name: </xforms:label>
<xforms:itemset nodeset="instance('all-countries')/country">
<xforms:label ref="name"/>
<xforms:value ref="name"/>
```

```
</xforms:itemset>
</fr:autocomplete>
```

(b) **Resource**: the autocomplete field in this mode shows a list of suggestions that returned by HTTP service in an XML document.

Example:

```
<fr:autocomplete ref="country-name" labelref="country-name/@label"
resource="/xforms-controls/services/countries?
country-name={$fr-search-value}"max-results-displayed="4">
<xforms:label>Country code: </xforms:label>
<xforms:itemset nodeset="/countries/country">
<xforms:label ref="name"/>
<xforms:value ref="us-code"/>
</xforms:itemset>
<xforms:alert>Value is mandatory</xforms:alert>
</fr:autocomplete>
```

This mode of auotcomplete feature introduced after the 3.9 release, and isn't yet available in Orbeon Forms 3.9 or earlier releases.

(c) **Dynamic mode:** the autocomplete in this mode displays a list of suggestions that is updated automatically based on what users typed, and it is not retrieved from a service.

Example:

```
<fr:autocomplete ref="instance('selected-countries')/dynamic"
labelref="instance('selected-countries')/dynamic/@label"
id="dynamic-autocomplete"
dynamic-itemset="true">

<!-- React to user searching -->
<xforms:action ev:event="fr-search-changed">
<xforms:variable name="search-value" select="event('fr-search-   value')"/>
<xforms:variable name="make-suggestion"
  select="string-length($search-value) >= 2"/>
<xforms:action if="$make-suggestion">

      <!-- Update itemset -->
  <xforms:setvalue ref="instance('search-dynamic')/country-name"
  value="$search-value"/>
<xforms:send submission="update-countries"/>
</xforms:action>
<xforms:action if="not($make-suggestion)">

       <!-- Delete itemset -->
<xforms:delete nodeset="instance('searched-countries')/country"/>
</xforms:action>
```

```
  </xforms:action>

  <xforms:label>Country code: </xforms:label>
  <xforms:itemset nodeset="instance('searched-countries')/country">
<xforms:label ref="name"/>
<xforms:value ref="us-code"/>
  </xforms:itemset>
    </fr:autocomplete>
```

4.4 Microsoft Office InfoPath 2007

InfoPath is a client-side implementation that based on using XML technologies such as XSLT, XML Schema and XPath expressions. InfoPath is used to design form templates that can be filled out in InfoPath, a web browser or a mobile device.

When the user designs a form template, InfoPath automatically generates an XML schema or the user can design a form template based on an existing XML schema. InfoPath creates an (.xsn) file for the form template that is a cabinet file combined of all different and required files such as XML schema (.xsd) file and XSLT (.xsl) file. XSLT presentation will be generated automatically when designing a form template, where InfoPath will produce the XSLT structure based on mapping xml schema elements to the UI controls.

InfoPath has a built-in feature that enables the user to save form data locally, and continue working on a form template that is stored on a server by caching it locally [29].

4.4.1 Installation

Microsoft InfoPath is installed as a part of Microsoft office 2007 package. It needs for Windows Vista or at least Windows XP Service Pack 1. It could be installed on MAC operating system as well.

I used Windows 7 machine to install Microsoft office 2007 that include InfoPath.

4.4.2 connection to a web service

InfoPath enables the user to configure the form template to allow receiving or submission of data (XML data) to a web service. For example, connect a form template to a web service could be used to receive or submit data to another program or system that is not directly supported in InfoPath. In addition, fields with values that are received from a web service may be

added to the form template or bind fields to another that associated with a web service [41].

4.4.2.1 Supported web service standard in InfoPath

- **Simple Object Access Protocol (SOAP)**: is a simple protocol based on XML that allows exchange information over the intenet. InfoPath supports Microsoft SOAP Toolkit 3.0.
- **Web Service Description Language (WSDL)**: is a description language based on XML that is used to describe location, communication protocol and interfaces to a web service.
- **Universal Description, Discovery, and Integration (UDDI)** Services: is an XML-based directory service that is used for registration and search web services. InfoPath uses UDDI to search for available web services.

4.4.3 Form supported features in InfoPath

The following sections are about form features that are supported and not supported by InfoPath. These features are tested using form samples that are designed by Microsoft InfoPath 2007.

4.4.3.1 Data validation

InfoPath supports data validation and It could be done using the following rules: [65]

- Using built-in data types to validate data.
- Using the built-in feature for validation condition , or adding the code of that condition in validating event and in this case the programming will be in C or visual basic (Microsoft visual studio should be installed).
- Using validation rules to check any changed event of a field or clicked event of a button.

4.4.3.2 Dynamic form modification

InfoPath provides dynamic form modification by using **repeating section** option. For example, adding and removing form fields are tested and found that are supported.

4.4.3.3 calculation

Calculation feature is supported in InfoPath. A formula (XPath expression consists of values, fields, functions, and operators) is used to calculate a form field or control. This formula is inserted to the form element or control from **insert formula** option that can be found under properties of the field [29].

4.4.4 Data storage

InfoPath can support different data storage: [29, 40]

- **XML file**: data form is stored as XML file locally or somewhere on a network.
- **SQL and Microsoft Access databases**: InfoPath enables the user to submit data to SQL AND access databases. As well as it allows query data from them and binds it to the form controls.
- **any relational database**: for instance, an InfoPath form could receive data from and submit data to an Oracle database that is connected using a web service.

4.4.5 Access rights control

InfoPath allows classifying users into one or more groups by using **User Roles** option. This option could be used to give users different views of a form depending on their roles [42]. In addition, Information Rights Management (IRM) in Microsoft Office InfoPath 2007 helps to create a form with restricted permissions for specific users. **permission** option is used to control access to a form and give users or groups one of the following permissions: [44]

- **Read permission**: gives the user Read access to the form.
- **Change permission**: enables the user to access the form to read, edit and save changes.
- **Full access permission**: gives the user full access to the form.

4.4.6 SVG feature

InfoPath does not support SVG feature, where there is no option or control is provided in InfoPath2007 to perform SVG. However, I could not find any reference to prove that.

4.4.7 subforms feature

Subforms feature is supported by InfoPath2007. A subform is created by using **template part** that is includes in **Desin a form template** dialog box, and then inserted to the main form.

4.4.8 Hint information feature

InfoPath provides this feature (called **ScreenTip**) under a form element or control properties.

4.4.9 dependency update feature

This feature is supported by InfoPath. It is implemented via **Rules** option that is under **control properties**. This option enables to set a fields value using a formula (XPath expression) depending on a change event in another fields value.

4.4.10 Upload feature

InfoPath enables the user to upload or attach files with a form. This feature is performed using **File attachment control** that allows uploading all types of files. If the form designer limited the allowed uploaded files then the user cannot upload a file out this limited set. Moreover, by default, InfoPath considers specific types of files (such as .com, .exe and .cmd) unsafe and restricts them from attaching or uploading with a form. System administrators in InfoPath can add and remove file types from the unsafe list by setting a value of a certain registry key. This restriction prevents maliciousexecutable files from attacking users computers [45].

4.4.11 spell checking

InfoPath supports this and it is enabled by selection the option **Enable spelling checker** from the **control properties** and then **Display** tab.

4.4.12 Autocomplete feature

Autocomplete feature is provided by InfoPath. The option **Enable AutoComplete** (selected from the control **properties** and the **display** tab) enables this feature

4.5 Summary

This chapter presented the most essential software that are used for smart forms developing based on W3C standards. It described and explained their capabilities and verified the supported and not supported form features that are tested and experimented.

Chapter 5

Summarized comparison between tested applications

This chapter presents a summarized comparison for the client and server side functionality, as well as supported and non-supported form features between the experimented applications that are introduced and tested in the previous chapter. The first section presents a table that compares client and server side implementations between the applications. The second section produces the comparison in tables; each table compares one form feature between the applications.

5.1 client and server side functionality comparison

The following table displays the functionality type of each of the previous experimented applications:

Application	functionality
AJAXForms	server-side implementation of XForms 1.0
eXist-db(XSLTForms)	client-side implementation of XForms 1.1
eXist-db(Better Forms)	server-side implementation of XForms 1.1
Orbeon Forms	server-side implementation of XForms 1.1
InfoPath	client-side system from Microsoft

Table 5.1: functionality type comparison

5.2 Form features comparison

This section is for comparing supported and non-supported form features between the applications. The following tables show a summarized

comparison for each form feature in each application.

5.2.1 Data validation

The following table compares data validation feature between the previous experimented applications:

Application	Data validation feature
AJAXForms	supported(XForms validation rules)
eXist-db(XSLTForms)	supported(XForms validation rules)
eXist-db(Better Forms)	supported(XFroms validation rules
Orbeon Forms	supported(XForms validation rules, mhand-coding or using Form Builder)
InfoPath	supported(using three different rules)

Table 5.2: Data validation comparison

5.2.2 Dynamic form modification

The following table compares dynamic form modification feature between the previous experimented applications:

Application	Dynamic form modification feature
AJAXForms	supported
eXist-db(XSLTForms)	supported(adding and removing from DOM with insert and delete elements)
eXist-db(Better Forms)	supported(adding and removing from DOM with insert and delete elements)
Orbeon Forms	supported(adding and removing from DOM with insert and delete elements)
InfoPath	supported(using repeating section interface option)

Table 5.3: Dynamic form modification comparison

5.2.3 calculation

The following table compares calculation feature between the previous experimented applications:

Application	calculation feature
AJAXForms	supported(XForms standard ways)
eXist-db(XSLTForms)	supported(XForms standard ways)
eXist-db(Better Forms)	supported(XForms standard ways)
Orbeon Forms	supported(XForms standard ways)
InfoPath	supported(using XPath expressions and insert formula interface option)

Table 5.4: Calculation comparison

5.2.4 Data storage

The following table compares data storage feature between the previous experimented applications:

Application	Data storage feature
AJAXForms	XML database
eXist-db(XSLTForms)	XML database
eXist-db(Better Forms)	XML database
Orbeon Forms	Embedded eXist-db database(XML), built-in database(Oracle, MySql and MangoDB) and Add-on database(Alfresco)
InfoPath	XML, MySql, Microsoft Access and any relational databse

Table 5.5: Data storage comparison

5.2.5 Access rights control

The following table compares access rights control feature between the previous experimented applications:

Application	Access rights control feature
AJAXForms	not verified
eXist-db(XSLTForms)	relies on the server database access control
eXist-db(Better Forms)	relies on the server database access control
Orbeon Forms	form builder access control(role-based) and Access control to deployed forms(field level and form level)
InfoPath	user roles option(gives users different views of a form) and permission option(gives users different restricted permissions to a form)

Table 5.6: Access rights control comparison

5.2.6 SVG feature

The following table compares SVG feature between the previous experimented applications:

Application	SVG feature
AJAXForms	not verified
eXist-db(XSLTForms)	supported and could be used to capture data
eXist-db(Better Forms)	not supported
Orbeon Forms	supported and could be used to capture data
InfoPath	not supported

Table 5.7: SVG feature comparison

5.2.7 Subforms feature

The following table compares subforms feature between the previous experimented applications:

Application	Subforms feature
AJAXForms	supported(using repeat element)
eXist-db(XSLTForms)	supported(using repeat element)
eXist-db(Better Forms)	supported(using repeat element) with cross-module submission
Orbeon Forms	supported(using repeat element)
InfoPath	supported(template part option from design a form template dialog box)

Table 5.8: Subforms feature comparison

5.2.8 Hint information

The following table compares hint information feature between the previous experimented applications:

Application	Hint information feature
AJAXForms	supported(using help element)
eXist-db(XSLTForms)	supported(using hint element)
eXist-db(Better Forms)	supported(using hint element)
Orbeon Forms	supported(using hint element)
InfoPath	supported(one of a control properties called screenTip)

Table 5.9: Hint information comparison

5.2.9 Dependency update feature

The following table compares dependency update feature between the previous experimented applications:

Application	Dependency update feature feature
AJAXForms	supported(using incremental attribute)
eXist-db(XSLTForms)	supported(using incremental attribute)
eXist-db(Better Forms)	supported(using incremental attribute)
Orbeon Forms	supported(using incremental attribute)
InfoPath	supported(using Rules option under a control properties)

Table 5.10: Dependency update comparison

5.2.10 Upload feature

The following table compares upload feature between the previous experimented applications:

Application	Upload feature
AJAXForms	not verified
eXist-db(XSLTForms)	not supported
eXist-db(Better Forms)	supported(using upload element)
Orbeon Forms	supported(using upload element)
InfoPath	supported(using file attachment control option)

Table 5.11: Upload feature comparison

5.2.11 Spell checking

The following table compares spell checking feature between the previous experimented applications:

Application	Spell checking feature
AJAXForms	not verified
eXist-db(XSLTForms)	not supported
eXist-db(Better Forms)	not supported
Orbeon Forms	supported(fr:spell-checker)
InfoPath	supported(one of a control properties)

Table 5.12: Spell checking comparison

5.2.12 Autocomplete feature

The following table compares autocomplete feature between the previous experimented applications:

Application	Autocomplete feature
AJAXForms	not verified
eXist-db(XSLTForms)	not supported
eXist-db(Better Forms)	not supported
Orbeon Forms	supported(fr:autocomplete) in three modes: static, resource and dynamic mode
InfoPath	supported(one of a control properties)

Table 5.13: Autocomplete comparison

5.3 Summary

This chapter introduced a summary that compared the experimented applications in this thesis. The comparison involved their client and server side implementations, and assessed for features.

Chapter 6

summary, conclusion and further work

6.1 summary

This thesis focused on smart forms concept. It reported the most major electronic forms technologies that are based on W3C standards. The research explained the supported features that could be included in an electronic form and make it smart form. In addition, it presented the related researches that contributed in the electronic form developing, as well as it described the main W3C electronic forms technologies, detailed their components, benefits, and web browser support, and tested them in term of the major electronic form features that explained in this research. Furthermore, those form features are tested using four electronic forms applications: AJAXForms, eXist-db that implements two technologies (XSLTForms on client-side and betterFORMS on server-side), Orbeon Forms and Microsoft InfoPath. The research outlined the installations, features, and capabilities of those applications which use W3C technologies in their implementations. Finally, this study produced a summary that compares the experimented applications in terms of their functionalities and electronic form features.

6.2 conclusion

HTML5 forms and XForms are the frontier electronic forms technologies that are based on W3C standards. Although XForms is more complex and not natively supported by web browsers, HTML5 forms did not replace it. XForms is more powerful where it is the first choice for the most software products industries. There are different implementations of XForms that transform XForms to JavaScript and HTML to enable web browsers rendering XForms. HTML5 Forms is simpler to be understood and modified by web designers. In addition, it provides powerful new input types and attributes, as well as it has a native support for

the most major smart forms features that are explored in this research. Xforms does not support all the electronic forms features such as spell checking and autocomplete features. However, Orbeon Forms application (which is a server-side implementation of XForms) provides all the smart forms features in addition to other useful and powerful electronic forms capabilities. As a result, I believe that the Orbeon Forms is a perfect application that could be used to develop a smart form with most of the powerful features.

6.3 further work

Based on the study that presented in this thesis, there are different concepts that can be conducted in the electronic forms developing field. Firstly, developing an algorithm transforms XForms to HTML5 Forms. This could be a contributed improvement that benefits from the native support of web browsers for HTML5 Forms. Secondly, building an application that is automatically generated an electronic form (XForms or HTML5 Form), SQL database and php script from XML schema/document. This application can be implementing by using Orbeon Forms application that tested and presented in this thesis as the most powerful electronic form application because it supports the most major electronic forms features. Finally, the first research that presented in chapter 3-section3.2.1 (**Generation and validation of web forms using database metadata and xforms.**) may be enhanced to generate complex XML schema and XForms.

Bibliography

[1] Ajaxforms. http://ajaxforms.sourceforge.net/index.html.

[2] Client-side and server-side scripting. http://www.yourwebskills.com/clientserver.php.

[3] Data validation. http://www.wong-sir.com/cit/information_processing/data_validation.htm.

[4] database servers, 2002. http://www.pcworld.idg.com.au/article/191014/database_servers/.

[5] Electronic forms. White Paper, 2009.

[6] agence XML. From ajaxforms to xsltforms. http://www.agencexml.com/xsltforms-slideshow/projectancestor.htm,lastviewed22April.

[7] agence XML. From ajaxforms to xsltforms. http://www.agencexml.com/xsltforms-slideshow/ajaxforms2xsltforms.htm,lastviewed22April.

[8] agence xml. Xsltforms. http://www.agencexml.com/xsltforms.

[9] Must be build. Adding form fields dynamically with jquery. http://www.mustbebuilt.co.uk/2012/07/27/adding-form-fields-dynamically-with-jquery/.

[10] Messaoud Benantar. *Access control systems: security, identity management and trust models*. Springer, 2006.

[11] John M. Boyer. Xforms 1.0 (third edition). Technical report, W3C. http://www.w3.org/TR/2007/REC-xforms-20071029/#intro.

[12] John M. Boyer. Xforms 1.1. Technical report, W3C. http://www.w3.org/TR/xforms/.

[13] Loren Cahlander, Paul Cross, Zar Geldiyev, Adam Stallman, and Mike Turpin. Exist-db open source xml database software architecture description. December 2010. chaos management.

[14] Toon Calders, Stijn Dekeyser, Jan Hidders, and Jan Paredaens. Analyzing workflows implied by instance-dependent access rules. In *PODS*, pages 100–109, 2006.

[15] Alain Couthures. Json for xforms. *XML Prague 2011*, page 13, 2011.

[16] Alain Couthures. xsltforms-support, February 2012. http://www.mail-archive.com/xsltforms-support@lists.sourceforge.net/msg00109.html.

[17] Commercial data processing. Checking the data (verification validation). http://www.jhigh.co.uk/ComputingSG/CDP/CDP_Validation.html.

[18] S. Dekeyser, J. Hidders, R. Watson, and R. Addie. Peer-to-peer form based web information systems. In Gill Dobbie and James Bailey, editors, *The Seventeenth Australasian Database Conference (ADC2006)*, pages 79–88, Hobart, Tasmania, Australia, January 2006. Australian Computer Society, Inc. http://eprints.usq.edu.au/293/1/adhw_adc06_final.pdf.

[19] Micah Dubinko, Stacy Silvester, Sebastian Schnitzenbaumer, and Dave Raggett. Xforms 1.0: Data model. Technical report, W3C. http://www.w3.org/TR/2000/WD-xforms-datamodel-20000406/.

[20] Mohammed M. Elsheh and Mick J. Ridley. Generation and validation of web forms using database metadata and xforms. In *Symposium on progress in information communication technology*, 2010. http://spict.utar.edu.my/SPICT-10CD/papers/spict10_18.pdf.

[21] Existdb. Open source native xml database. http://http://exist-db.org.

[22] Existdb. Open source native xml database. http://cdi.uvm.edu/exist/quickstart.xml,last visited 30 March 2013.

[23] Existdb. Open source native xml database, l. http://exist-db.org/exist/apps/doc/xforms.xmll,last visited 3 April 2013.

[24] Existdb. Open source native xml database, user authentication and access control. http://cdi.uvm.edu/exist/security.xml,last visited 30 March 2013.

[25] Better Form. The xforms toolkit. http://www.betterform.de/en/xforms-toolkit.html.

[26] Yuri Gapanyuk, Egor Lakomkin, Sergey Ionkin, and Martin Davtyan. Mvc web framework based on exist application server and xrx architecture. In Denis Turdakov and Andrey Simanovsky, editors, *SYRCoDIS*, volume 735 of *CEUR Workshop Proceedings*, pages 19–25. CEUR-WS.org, 2011.

[27] Rebecca A Hills, Janet G Baseman, Debra Revere, Craig L K Boge, Mark W Oberle, Jason N Doctor, and William B Lober5. Applying the xforms standard to public health case reporting and alerting. *Online Journal of Public Health Informatics*, 3(2):414–425, June 1998. http://ojphi.org/htbin/cgiwrap/bin/ojs/index.php/ojphi/article/view/3656/3086.

[28] Trent Jaeger. Systems and internet infrastructure security lab. Technical report, Pennsylvania State University. http://ix.cs.uoregon.edu/~butler/teaching/10F/cis607/papers/jaeger-refmon.pdf.

[29] Philo Janus. *Pro InfoPath 2007*, volume 252. Apress, 2007.

[30] JavaHowTo. 6 common errors in setting java heap size, July 2006. http://javahowto.blogspot.com.au/2006/06/

6-common-errors-in-setting-java-heap.html,last visited 12 April 2013.

[31] Jr. Steven Pemberton Nick Van den Bleeken John M. Boyer, Leigh L. Klotz. Xforms 2.0. Technical report, W3C. http://www.w3.org/MarkUp/Forms/wiki/XForms_2.0#About_the_XForms_Specification.

[32] Ján Kasarda, Martin Nečaskỳ, and Tomáš Bartoš. Generating xforms from an xml schema. In *Networked Digital Technologies*, pages 706–714. Springer, 2010.

[33] Markku Pekka Mikael Laine et al. Xformsdb-an xforms-based framework for simplifying web application development. 2010.

[34] Matthew Langham and Carsten Ziegeler. *Cocoon: building XML applications*. Sams, 2003.

[35] Bruce Lawson and Remy Sharp. *Introducing html5*. New Riders, 2011.

[36] Peter Lubbers, Frank Salim, and Brian Albers. *Pro HTML5 programming*. Apress, 2011.

[37] Wayne Allan Malkin. system and method of implementing calculation fields in an electronic form, Novmber 2004.

[38] Wolfgang Meier. exist: An open source native xml database. *Web, Web-Services, and Database Systems*, pages 169–183, 2003.

[39] Antony J. Moore, Susan M. Warren, Scott D. Guthrie, and Steven A. Isaac. web controls validation, July 2005.

[40] msdn. Introduction to microsoft office infopath 2007. http://office.microsoft.com/en-us/infopath-help/introduction-to-microsoft-office-infopath-2007-HA010163577.aspx,last visited 19 April 2013.

[41] msdn. Overview of web service support in infopath 2007. http://msdn.microsoft.com/en-us/library/office/bb852081(v=office.12).aspx,last visited 19 April 2013.

[42] MssharePointTips. Using infopath 2007 user roles. http://www.mssharepointtips.com/tip.asp?id=1082,last visited 20 April 2013.

[43] Christopher Murphy, Richard Clark, Oli Studholme, and Divya Manian. *Beginning HTML5 and CSS3: The Web Evolved*. Apress, 2012.

[44] Office. Create a form that has restricted permission. http://office.microsoft.com/en-au/infopath-help/create-a-form-that-has-restricted-permission-HA010077998.aspx,last visited 20 April 2013.

[45] Office. Insert a file attachment control. http://office.microsoft.com/en-au/infopath-help/insert-a-file-attachment-control-HP010080908.aspx,last visited 20 April 2013.

[46] The open web application security project OWASP. Reviewing code for data validation. https://www.owasp.org/index.php?title=Reviewing_Code_for_Data_Validation&setlang=es#Never_Rely_on_Client-Side_Data_Validation.

[47] Orbean. Autocomplete. http://wiki.orbeon.com/forms/doc/developer-guide/xbl-components/autocomplete#TOC-Overview,last visited 27 April 2013.

[48] Orbean. Spell checker. http://wiki.orbeon.com/forms/doc/developer-guide/xbl-components/spell-checker#TOC-Overview,last visited 27 April 2013.

[49] Orbean. Xforms - validation. http://wiki.orbeon.com/forms/doc/developer-guide/xforms-validation.

[50] orbeon. Form runner access control. http://blog.orbeon.com/2012/05/forms-access-control-with-form-builder.html,last visited 14 April 2013.

[51] orbeon. Form runner/form builder persistence api. http://wiki.orbeon.com/forms/doc/developer-guide/form-runner/persistence-api,last visited 14 April 2013.

[52] orbeon. Orbeon form builder - user guide. http://wiki.orbeon.com/forms/doc/user-guide/form-builder-user-guide#TOC-Orbeon-Form-Builder,last visited 9 April 2013.

[53] orbeon. Orbeon forms faq. http://dl.lib.brown.edu:8083/ops/doc/home-faq,last visited 9 April 2013.

[54] Orbeon. Xml pipeline language (xpl). http://wiki.orbeon.com/forms/doc/developer-guide/xml-pipeline-language-xpl,last visited 27 April 2013.

[55] orbeon. Forms access control with form builder, May 2012. http://wiki.orbeon.com/forms/doc/developer-guide/form-runner/access-control#TOC-With-Orbeon-Forms-4.0,last visited 14 April 2013.

[56] Pdfscrpting.com. Calculating field values and more. http://www.pdfscripting.com/public/department45.cfm.

[57] Mark Pilgrim. Dive into html5, 2010.

[58] quirksmode.org. Extending forms. http://www.quirksmode.org/dom/domform.html,last visited 8 May 2013.

[59] Ravi Sandhu and Jaehong Park. Usage control: A vision for next generation access control. *Computer Network Security*, pages 17–31, 2003.

[60] Alain Couthures Source forge. xsltforms-support, July 2012. http://sourceforge.net/mailarchive/message.php?msg_id=29510306.

[61] Edwin Tump. The risks of client-side data dtorage. Technical report, SANS institute reading room site, 2011. http://www.sans.org/reading_room/whitepapers/storage/risks-client-side-data-storage_33669.

[62] W3schools.com. Html spellcheck attribute. http://www.w3schools.com/tags/att_global_spellcheck.asp,last visited 8 May 2013.

[63] W3schools.com. Html5 web storage. http://www.w3schools.com/html/html5_webstorage.asp,last visited 8 May 2013.

[64] Hua Wang, Yanchun Zhang, and Jinli Cao. Access control management for ubiquitous computing. *Future Generation Computer Systems*, 24(8):870–878, 2008.

[65] S.Y.M. Wong-A-Tonl. Infopath basics: 3 ways to validate data in infopath. `http://www.bizsupportonline.net/infopath2007/infopath-basics-3-ways-validate-data-infopath.htm`.