Integration of Relational Database Metadata and XML Technology to Develop an Abstract Framework to Generate Automatic and Dynamic Web Entry Forms

Mohammed Mosbah Elsheh

PhD

2009

Integration of Relational Database Metadata and XML Technology to Develop an Abstract Framework to Generate Automatic and Dynamic Web Entry Forms

An investigation into the development of an abstract framework for producing automatic and dynamic web entry forms based on database metadata and deploying XML technology

Mohammed Mosbah Elsheh

Submitted for the degree of Doctor of Philosophy

Department of Computing

School of Informatics

University of Bradford

2009

Acknowledgements

First of all, I am very grateful to my Lord Almighty ALLAH who helped me and guided me throughout my life and made it possible. I could never have done it by myself!

I would like to express my deep thanks and appreciations to my supervisor: **Mr. Mick.J.Ridley** for his invaluable guidance and insightful direction throughout this research work. This work is the result of inspiration, advice and motivation that were extended to me by him. I am indebted to him. His most prominently directed guidance helped me overcome many problems which otherwise have taken more efforts and time.

Finally, I deeply thank my wife for her support and encouragement throughout my studies.

Abstract

Developing interactive web application systems requires a large amount of effort on designing database, system logic and user interface. These tasks are expensive and error-prone. Web application systems are accessed and used by many different sets of people with different backgrounds and numerous demands. Meeting these demands requires frequent updating for Web application systems which results in a very high cost process. Thus, many attempts have been made to automate, to some degree, the construction of Web user interfaces. Three main directions have been cited for this purpose. The first direction suggested of generating user interfaces from the application's data model. This path was able to generate the static layout of user interfaces with dynamic behaviour specified programmatically. The second tendency suggested deployment of the domain model to generate both, the layout of a user interface and its dynamic behaviour. Web applications built based on this approach are most useful for domain-specific interfaces with a relatively fixed user dialogue. The last direction adopted the notion of deploying database metadata to developing dynamic

user interfaces. Although the notion was quite valuable, its deployment did not present a generic solution for generating a variety of types of dynamic Web user interface targeting several platforms and electronic devices.

This thesis has inherited the latter direction and presented significant improvements on the current deployment of this tendency. This thesis aims to contribute towards the development of an abstract framework to generate abstract and dynamic Web user interfaces not targeted to any particular domain or platform. To achieve this target, the thesis proposed and evaluates a general notion for implementing a prototype system that uses an internal model (i.e. database metadata) in conjunction with XML technology. Database metadata is richer than any external model and provides the information needed to build dynamic user interfaces. In addition, XML technology became the mainstream of presenting and storing data in an abstract structure. It is widely adopted in Web development society because of its ability to be transformed into many different formats with a little bit of effort. This thesis finds that only Java can provide us with a generalised database metadata based framework. Other programming languages apply some restrictions on accessing and extracting database metadata from numerous database management systems. Consequently, JavaServlets and relational database were used to implement the proposed framework. In addition, Java Data Base Connectivity was used to bridge the two mentioned technologies.

The implementation of our proposed approach shows that it is possible and very straightforward to produce different automatic and dynamic Web entry forms that not targeted at any platform. In addition, this approach can be applied to a particular domain without affecting the main notion or framework architecture. The implemented

approach demonstrates a number of advantages over the other approaches based on external or internal models.

List of Author's Publications Related to this Thesis

1. Elsheh, M.M. and M.J. Ridley. *DEVELOPING AN ABSTRACT REPRESENTATION FOR USER INTERFACE ELEMENTS BASED ON DATABASE METADATA in IADIS International Conference WWW/Internet 2007 October* 2007. Vila Real, Portugal
2. Elsheh, M.M. and M.J. Ridley. *Using Database Metadata and its semantics to Generate Automatic and Dynamic Web Entry Forms*. In *WCECS 2007 World Congress on Engineering and Computer Science 2007*. San Francisco, USA: IAENG, International Association of Engineers.
3. Elsheh, M.M. and M.J. Ridley. *A Generic Approach to Generate an Abstract Web Entry XForms Using Database Metadata*. In *the Ninth Informatics Workshop for Research Students*. 2008. University of Bradford, UK.

Contents

List of Figures

List of tables

List of listing

Chapter 1

Introduction:

1.0 Overview

Nowadays the Internet is ubiquitous. Dozens of websites, blogs and Web applications systems are deployed everyday. Websites are the first kind of Web systems appeared during the 1970's. They offer information to the end-users in a static way. The interaction between them and end-users is unidirectional. However, blogs are considered as some sort of web sites. They allow users to post their comments on a specific topic. The most powerful and beneficial Web systems are the web application systems. They allow the end-user to fill in web forms in order to post several types of data such as text, images and audio/video files.

The interaction of a database and the Web is becoming the cornerstone of developing Web application systems. Shifting legacy data that held in stand-alone systems to be used in Web application systems is an expensive and time consuming chore for many corporations.

Building Web application systems involves several phases and needs a lot of work to integrate several tasks with each other. They can be developed by a team where each member has a specific role and need not work on common files.

The automatic and dynamic generation of Web application systems has become the mainstream in Web development sector. Many efforts have been made over the last two decades in order to address the difficulties faced by this approach. The majority of these efforts concentrated on the external data model such as [1], [2] and [3] whereas a few of them make use of internal data model but they failed to tackle the problem in an optimal approach since they did not achieve the separation between the logic, content and presentation such as in [4] and [5] .

1.1 Motivations

The explosive growth and variety of Web technologies has pushed many business organisations to turn their database applications into on-line systems. From another side, as new technologies such as XML and its surrounding children such as XHTML and XSLT have gained a wide acceptance in the Internet society, updating current Web applications to take advantages of these technologies is becoming a top priority for many Web applications' developers and owners. Developing interactive Web applications that meet most clients' demands in terms of the portability and compatibility of the existence soft/hard ware is a complicated task and has challenged Web developers in many phase including:

- Time needed to develop, maintain and amend Web application systems.

- High cost. The longer it takes to build up, maintain and amend the Web application the higher cost is paid.
- Taken in account the variety of platforms of end-users.
- Building Web application considering the separation between the content, logic and presentation.

1.2 Aims and objectives

The overall aim of this study is to build and test a generic and flexible approach that can be used to generate automatic and dynamic Web application systems, using database metadata that held in system catalogue tables in relational databases management system (RDBMS) in conjunction with XML technology and its related technologies. To achieve this aim, the follows objectives are considered:

- To carry out an extensive literature review about the existing web technologies. This literature mainly focuses on the limitations of the most used technologies in Web application development.
- To carry out an extensive study to investigate and discover the potential resources of database metadata in order to determine for how extent this resource can provide us with sufficient information to achieve the main aim.
- To explore the features of Web browsers and to what extent they support our approach. This mainly focuses on how Web browsers deal with Xforms technology and to what degree they can support it.

- To investigate XML technology and its surrounding technologies in order to determine for how level it can be combined with database metadata to build a very high-level abstract representation for user interface elements.

1.3 Thesis Outline

The rest of this thesis is organised as follows:

- Chapter 2: This chapter describes the state-of-the-art in Web technologies. The main purpose of this chapter is conducting a survey of available technologies and asses the most suitable ones to implement the proposed approach.

- Chapter 3: In this chapter, a comprehensive survey was carried out to understand the current state-of-the-art in database field in order to asses the suitability of different technologies for implementation of the prototype system to be developed throughout this study. Database developments and Java DataBase Access gained the most focus in this chapter.

- Chapter 4 : The philosophy of academic research is to build up on what others have done, improving what already has been made or discovering new directions and methods to push forward the wheel of life. For this reason, this chapter surveys a number of academic research papers and commercial products dealing with issues that relate to Web user interface to databases. Three main sections

are considering in this chapter. The first concentrates on issue of building Web user interfaces to databases in general. The second deals with the issue of employing metadata to producing user interfaces. The last is concerning on the notion of using XML technology to developing dynamic Web user interface.

- Chapter 5: This chapter begins with an overview of the characteristics of the prototype system. A general notion of how the framework constructed is introduced. System architecture and its layers are demonstrates in details. How does the system work and how it is employed database metadata in conjunction with XML technology to improve the notion of building an abstract framework to generate different automatic and dynamic Web entry forms is illustrated and discussed.

- Chapter 6: Concludes the thesis and points out possible future work.

Capter 2

Web Technologies

2.0 Introduction

Nowadays, the main method of providing interaction between end users and Web application systems are Web forms. They are used in all sorts of Web application systems, from basic customer support services, to more sophisticated use such as stock-sharing systems. At present, such web form based applications are primarily implemented using HTML [6], complemented with client-side scripting such as VBScript [7] and JavaScript [8], that is used for basic functionality such as data validation, and by server-side technologies such as PHP [9], .Net and JSP [10] that offer more complicated business processing and functionality. Lately, many Web developers have adopted XML [11] and its surrounding technologies for building more extensible and interactive Web application systems. As a result, XForms [12] technology that is

compliant with XML technology is introduced to solve the shortcomings of the current HTML Web forms.

Since the main focus of this study is to find out to what extent can relational database metadata in conjunction with XML technology fit in generating automatic and dynamic Web forms, the main focus of this chapter will be on those Web technologies that most fit our approach.

The chapter is spilt into three main sections. The first section discusses the most commonly used Mark-up language in building the concrete part of Web forms. The second section surveys the commonly used server-side technologies, mainly Javaservlet. The last section introduces the client-side technologies, mainly JavaScript.

2.1 Mark-up languages

Although the term “Mark-up” has several different means, the basic idea behind it in general, is to take plain text web content and add some indications of how this text should be formatted. One of the common purposes of Mark-up languages is to structure information in a proper way for display reasons. Nowadays, the common language widely used to mark-up text for web site and web applications is HyperText Mark-up Language (HTML). However, to enhance the appearance and formatting different HTML elements, Cascading Style Sheets (CSS) [13] can be used along with HTML. XML (or Extensible Mark-up Language) is another mark-up technology originating on the web and its main usage is not in creating web pages but to interchange data over the internet between different systems having different software and different platforms.

Lately, XForms was introduced and intended to replace traditional HTML, it built on a new concept, which separate the look and feeling of Web forms.

Similar to CSS in some respects, XSLT [14] is a tool that can be used to transform XML documents into different forms such as HTML, XHTML, XForms and so forth. This chapter introduces the commonly used mark-up languages used in Web applications development.

2.1.1 SGML

Standard Generalized Mark-up Language (SGML) was created in the middle of the 1970's to be used as a tool to automate document processing. SGML is an international standard (ISO8879) for the definition of platform and system independent methods of representing texts in electronic form [15]. SGML is not a document language itself but it is a Meta language that defines a description of how to specify a document language. HTML, for instance is defined in SGML.

2.1.2 HTML

HyperText Markup Language (HTML) is the most commonly used Markup language for creating websites and web applications. It has been in use on the World-Wide Web (WWW) since the early days of the last decade. HTML has been reviewed and many releases came out with many enhancements. These versions are introduced as follows:

- **HTML 1.0**

This version was the first release of HTML to the public. Since the number of people involved in website development was very limited, this release was very basic and did not offer any sort of interactivity between the end users and websites.

- **HTML 2.0**

This release is the first definitive one of the HTML family. It came with most HTML elements are that still in use until now such as the INPUT types, PASSWORD, RADIO, CHECKBOX, RESET, SUBMIT and so forth.

- **HTML 3.0**

Although this version of HTML provided several additional capabilities over HTML 2.0 such as tables, page alignment for block structuring elements and text flow around figures, it did not survive, even it did not complete. The reason is that W3C was forced to produce HTML 3.2 as an attempt to formalise all the bits that Microsoft and Netscape had implemented.

- **HTML 3.2**

It became an official standard in 1997. It introduced new HTML elements such as TABLES, IMAGE, HEADING and other element ALIGN attributes. The obvious drawback in this version is that missing some of the Microsoft/Netscape extensions such as APPLET, FRAME and EMBED.

- **HTML 4.0**

This version includes support for most of the proprietary extensions. In addition, it supports new features including support for Cascading Style Sheets, internationalised documents, extra FORM, TABLE and ECAMScript [16] enhancements. A very short time after this version has come out, it was revised and corrected in a few slight ways and was entitled HTML 4.1.

- **HTML 5.0**

It is the latest proposed version of the HTML family and still in the draft stage. It is formed by a group called Web Hypertext Application Technology Working Group (WHATWG) [17]. It adds some convenient elements for more detailed semantic structures within HTML. It also removed legacy-formatting elements such as (center, font, frame, frameset and many others) and suggested using CSS to deal with layout issues. Beside the existing features in HTML 4.01 and XHTML 1.0, the proposed release introduces new items, including:

- New layout elements that reflects typical usage on modern Web sites. A number of of these elements are technically similar to earlier elements such as <*span*> and <*div*> tags, but have a semantic meaning, for instance <*footer*> and <*nav*> (website navigation block). Other elements offer new functionality via standardised interface, like the <*video*> and <*audio*> elements.
- Server-sent DOM events which intended to improve native, cross-browser streaming. Server-sent events defines a data format for streaming events to

web browsers, and an related DOM API for accessing those events, via attaching call-back functions to specific named event types.

- Dynamic graphic capabilities.
- Programming changes to the Document Object Model (DOM). The existing DOM interfaces are extended and new APIs are introduces such as drag-and-drop and timed media playback.

2.1.2.1 HTML Document Structure

As shown in Listing 1, HTML documents are structured into two main parts, the HEAD, and the BODY. The former contains information about the document that is not generally displayed with the document, such as its title. The later contains the body of the text. All form elements are placed in this part of document. Elements allowed inside the HEAD, such as TITLE, are not allowed inside the BODY, and vice versa.

```
<HTML>
  <HEAD>
    <TITLE> Title comes here </TITLE>
    <STYLE> style if needed </STYLE>
    <SCRIPT> script if needed </SCRIPT>
  </HEAD>
  <BODY>
     HTML elements that construct the layout
     of the document.

  </BODY>
  </HTML>
```

Listing 1 HTML document structure

2.1.2.1.1 HTML Form Controls

HTML is very rich in controls and attributes that are used to construct the user interface. The most commonly used controls and attributes in building interactive Web forms are listed as follows:

- **Input Control** is the most used form control. This control can accept any data type specified by the *type* attribute. Among a dozen input types, the list below shows the most commonly used ones.

1. **Text Fields:** used to allow the end user to enter any values from the key board including numeric digits, letters and special characters. This attribute is constructed as: *<input type="text" name="any name">*. Entered data type can vary based on the value of *type* attribute and can include *int, float, date, url, hidden,* etc.
2. **Radio Buttons:** used to allow users to select one among a limited number of choices. The layout of this attribute is as:

   <input type="radio" name="any name" value="any value">.
3. **Checkboxes:** allow users to select one or more than one options among a limited number of choices. Checkboxes are constructed as:

   <input type="checkbox" name="name1" value="value1">

   *
*

   <input type="checkbox" name="name2" value="value2">

   *
*

   <input type="checkbox" name="name3" value="value3">

   The list could hold tens or hundreds of values based on the browser capability.

4. **Submit Button and Action attribute:** used to submit entered data to a specific URL. The layout of these attributes are as follow:

   <input type=”submit” value =”any text”>.

   The URL is specified by *Action* attribute as follows:

   <form name=”input” action=”any URL”>

2.1.2.2 HTML Limitations

HTML forms are considered the most significant features of HTML that facilitate the interaction between users and Web applications. However, although HTML has had some important developments since its first appearance, it still has some limitations that make it reasonable to look for another alternative tool to create and deploy Web applications. A commonly cited drawback of HTML forms is their dependency on scripting languages such as JavaScript, JScript and Visual Basic Script. Real-world HTML forms are reliant on scripts to accomplish many common tasks such as marking controls as required, performing validations and calculations, displaying error messages, and managing dynamic layout. This dependency results in complex documents, which are expensive and time consuming to maintain [18].

Another limitation is that the initialisation of data is control specific and complex to manage [19]. Since each form control has its own unique way of defining initial data this means that in order to process a blank form into a filled one, either a new document

needs to be constructed piece by piece or an existing document needs to be patched in numerous places. Constructing such forms is CPU resource consuming and can lead to bottlenecks on servers.

The third limitation is that HTML Forms can only represent flat data structures or name/value pairs [19]. In the real world, most business documents such as purchase orders needed to benefit from a complex data such as audio representation where HTML fails to deliver for this requirement. However, XML provides a better foundation for most business documents than a flattened set of names and values by offering a very rich data representation.

The fourth limitation is that HTML forms have a very simple one step process from client to server and back again. However, today's business processes are becoming ever more automated and the ability to define complex processes are typical within forms based applications. This simple limitation means that more complex back end processing and workflow management needs to be introduced which again adds complexity and cost of maintenance for these systems. One goal of introducing AJAX (Asynchronous JavaScript And XML) is to deal with the above limitation. AJAX allows processing on a client computer (in JavaScript) with data taken from the server. It can selectively modify a part of a page displayed by the browser, and update it without the need to reload the whole document with all images, menus and so forth. For instance, fields of forms, choice of user, may be processed and the result displayed immediately into the same page. To clarify this concept, let us consider this example adopted from [20] . A pizza order and delivery system holds the customers’ data such as name, phone

number, address and so forth. The order reservation page uses the phone number as access key. The order needs to be delivered to the customer address and the order itself needs to be specified by the customers every time they make a new order. Following AJAX approach, as soon as the system gets the phone number and cursor left the input field, the page should automatically fill in the customer's address. This allows customers to work on their order while the server is getting their address. However, AJAX lacks in support small electronic devices. More information about AJAX in [21].

The final drawback with HTML is the fact that there is poor separation between data and presentation, which leads to importable web forms. In other words, if a form is designed to be deployed on a desktop browser another form is needed for deployment on a mobile or handheld device [19].

Because of the above drawbacks of current HTML Web forms, XForms that is an XML based technology was introduced to tackle these limitations and make Web forms more extendable and interactive.

2.1.3 XHTML

The Extensible Hypertext Markup Language or XHTML is a W3C recommendation for Web developers [22]. Unlike HTML that is an application of SGML, a very flexible Markup language, XHTML is an application of XML, a more restrictive subset of SGML. Since XHTML is a reformulation of HTML 4.01 in XML, it can be thought of as a hybrid of HTML and XML.

The first generation of XHTML (i.e. XHTML 1.x) has not brought significant changes to HTML 4 apart from some features to achieve conformance with XML. The most important change that was introduced in this generation is the requirement that the document must be well formed and that all elements must be explicitly closed as required in XML. However, the most important improvements are introduced in XHTML 2.0 which is still in draft stage, these new features include:

- Unlike the first generation of XHTML, XHTML 2.0 is not backward compatible with its predecessors (HTML x.x and XHTML 1.x).
- HTML forms are replaced by XForms which allow forms to be displayed in appropriate ways on different devices.
- XFrames will replace HTML frames.
- DOM events will be replaced by XML Events, which use the XML Document Object Model.

2.1.4 XML

XML (Extensible Mark-up Language) became a W3C recommendation in February 1998 [11]. XML sits between the two technologies (SGML and HTML), not as a complex as SGML, whereas is vastly more powerful than HTML. XML is a meta language which allows the definition of multiple Mark-up languages [23] . In addition, XML it is not only designed to be used as a text encoding tool, but also to be used as a main vehicle to exchange information between different kinds of computers, different

Web application systems and different organisations without needing to go through several layers of conversion. Furthermore, XML has numerous specifications for mathematical formula (MathML) [24], Molecular Information [25] and many others. Unlike HTML, XML tagging can be used to define the logical content of a document rather than its physical structure that makes it possible for different applications to format the same document in different ways using numerous tools such as XSL.

The main motivation behind XML is the need to support Web applications that can not be accomplished within the limitations of HTML (see section 2.1.2.2). These applications are divided into four categories:

- Applications that require the Web client to mediate between two or more heterogeneous databases.
- Applications that attempt to distribute a significant proportion of the processing load from the Web server to the web client.
- Applications that require the Web client to present different views of the same data to different users.

2.1.4.1 XML Document Structure

As W3C states, XML has both physical and logical structure. Physically, it is comprised of storage units called entities. Entities are pieces of information that refer to other data and pages as shortcuts. Logically, an XML document consists of declarations, comments, elements, references, characters and processing instructions.

As shown in Figure 1, an XML document consists of three main parts that are Prolog, Body and Epilog.

- Prolog is an optional part and not required in order for an XML document to be well-formed. XML declaration, comments, processing instructions and document type declaration are the most pieces of information are embedded in this part. In addition, if XML declaration is included, it must be the first line of the XML document.
- A document body or instance is the bulk of the information content of the document.
- Epilog is an optional part. Comments and processing instructions could be included in this part.

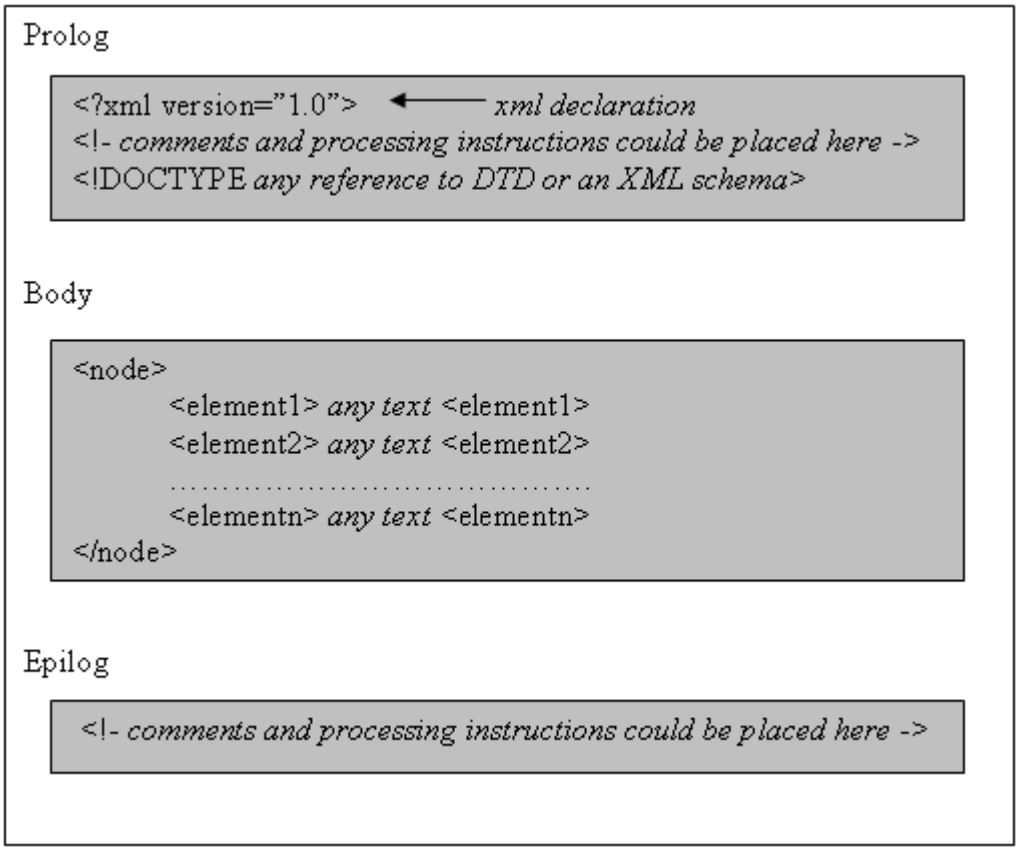


Figure 1 The layout of an XML document

2.1.4.2 Well-formed or valid XML document

The nominal requirement for any XML document is to be well-formed. A well-formed XML document must adhere to the following set of conventions:

- XML document must have consistent, well-defined structure.
- Apart from empty elements, all start tags must have matching end tags.
- All attributes values must be quoted. Either by single or double quotes.
- Every XML document has only one root element and must contain all others.
- Elements must be well nested and not overlapping.
- Apart from root element, each element must have exactly one parent element that contains it.
- Attributes and elements names are case-sensitive.
- Keywords such as ENTITY and DOCTYPE must always written in uppercase.

Despite adherering to all the above rules, an XML document maybe still an invalid document. Valid document must be both, well-formed and stick to the constraints imposed by a DTDs or XML schema.

2.1.4.3 XML validation

It is well known that every XML document must be well-formed. Beside that in many cases, XML document is required to be validated in order to meet specific requirements.

Among of several techniques used to validate XML document, DTDs and XML schema are the most used ones.

- **DTD**

It stands for Document Type Definition [26]. It uses formal grammar to identify the structure and permitted values of XML documents. DTDs can be declared as an external reference, or inline inside an XML document. Despite that fact that DTDs still in use and operates in meaningful matter, it comes with several drawbacks that make it reasonable to look for a new technique to replace it. Among of these shortcomings are:

- It dose not support a range of data types.
- It has no support for extensions or inheritance of declaration.
- Lack of namespaces, which make it not possible for an XML document to reference more than one external DTD.

- **XML schema**

XML schema is a language that used to describe the structure of an XML document [27]. It is written in XML and adheres to all XML rules. It can be seen as extension to DTDs with a restricted form of specialisation. It was introduced to tackle the shortcomings of the existing DTDs, so it comes with new features including:

- Support for primitive data types such as (xsd:date, xsd:integer, xsd:string, and so on).
- The ability to define custom data types.
- It is compatible with other XML technologies such as XSLT and XQuery.

2.1.5 XForms

This section introduces the Xforms technology, discussing its characteristics, structure and elements and how it differs from the current HTML.

2.1.5.1 Definition

XForms 1.0 became a W3C recommendation on October, 2003 [12]. It is intended to replace the current HTML form. XForms is built on the concept of Extensible Markup Language (XML). They are intended to be platform and device independent that makes Web forms reusable on many electronic devices such as PC's, handhelds, cellular phones and other systems. XForms is not a stand-alone technology, but a XML based technology. A host language is an essential requirement for XForms to be deployed. Among many XML based technologies such as SVG, XHTML, and WML, XHTML is the most common technology associated with XForms.

2.1.5.2 XForms Features

Unlike HTML, XForms is designed to separate action from intention, meaning from presentation. This philosophy resulted in many new features that were missed in traditional HTML. These features are listed as follows:

- **Minimise round trips**

In Web applications environment, the performance bottleneck is often resulted either from the network connection between the client and server or from the server when it is exhausted by processing a load of connections. To overcome this issue, it would be a great achievement if part of these processes were shifted to the client side. For example, many calculation's tasks can be made on the client side by integrating HTML and JavaScript. JavaScript's implementation results in a massive number of lines of code and requires a lot of work for maintenance. However, XForms presents an elegant solution by adding calculation fields that rely on the content of other fields.

- **Input validation**

Filling out forms is an error-prone task due to the humanity nature. In most current Web applications, many complex JavaScript codes are used to verify essential constraints such as a birthday being in the future, rather than in the past or credit card number having the correct number of digits. XForms provides built-in form validation on both the client and on the server that eliminate the need of using any scripting language to validate the entry data against specific constraints.

- **Multiple environments**

As stated earlier, XForms is intended to run in numerous environments including typical desktop Web browsers like Microsoft Internet Explorer and Firefox; audio browsers and mobile phone browsers. XForms says nothing about how the form will be displayed on the browsers or how the user will interact with it. Presentation task is totally left to the browser's processor. In addition, XForms does not make any postulation about what

sort of machine is used to fill out the form. For instance, there is no XForms control element called *date*, but binding an *input* control element with data type of date type maybe returned in some Web browsers as a date picker.

- **Accessibility**

Since XForms is capable of running in many different browsers and on many different electronic devices, this allows it to work for any person who has manual, visual, or any other disabilities. For instance, an audio interface can be used by people to fill out XForms while they are driving.

2.1.5.3 XForms Limitations

Currently, only X-Smiles Web browser supports XForms natively, some other Web browsers provide plug-in software that must be installed on the client machine in order for Web browsers to return XForms documents in proper way. Other technologies suggested generating XForms on the server-side before sending them back to a browser. The following section outlines some of these technologies:

- FormsPlayer, extends Internet Explorer 6 to include not only support for the full XForms standard, but also provides DOM 2 events, DOM 3 XPath, XML Events and the DOM 3 implementation Registry [28].
- FormFaces is a pure JavaScript processor. This means that XForms+HTML can be sent directly to the browser where JavaScript translates XForms controls into regular HTML form controls and processes the bindings directly within the browser without any need to server-side processing or plugins. This goal can be achieved in a very simple way by adding the following tag into Xforms+HTML documents: <script type=”text/javascript” src=”formfaces.js”> </script> [29].

- Firefox is supposed to support XForms via an extension and it is in use for versions 1.x and 2.x.
- X-Smiles is a client-side implementation. It is an open source XML browser, developed at Telecommunications Software and Multimedia Laboratory of the Helsinki University of technology [30]. It supports XForms as well as most current XML languages, such as XHTML and XSLT.

- **Server-side XForms engine**

  In [31], Erik Bruchez concluded that implementing a server-side XForms engine could be the promising solution to bring the capabilities of XForms to web browsers deployed today. He argued that many benefits could be obtained when this approach is adopted. These benefits include:

  1. Eliminate the need to install plug-ins or upgrading the existing browsers such MS Internet Explorer.
  2. Web pages developers can use XForms without paying any attention to the difference between web browsers.
  3. Improving latency, saving bandwidth, and providing enhanced confidentiality.

- **Ajax-Based XForms:**

  In [32], it is argued that by using Ajax-Based XForms technology, XForms will reach the web at large and fulfil its initial promise. As it is reported in this work, the above technology has solved the main issue with regards to XForms which is XForms deployment and it provided additional benefits compared to browser plug-in and native implementations, including:

1. Improved security. This is achieved by entirely keeping XForms engine state on the server and prevents end users from accessing the XForms engine internally.

2. Performance optimizations. By keeping very large XForms instances on the server instead of sending them to the client, this leads to improved page startup time and latency.
3. Ease of upgrade. By upgrading one server-side component, bug-fixes, new widgets, and extensions are deployed to all clients.

2.1.5.4 Xforms Document Structure

XForms is an application of XML and is intended to be used in conjunction with other XML vocabularies. XForms is not stand-alone technology. A host language is needed to accommodate XForms. XHTML is the most common used host language in use to host XForms. Since XForms is not standard part of XHTML 1.0, a namespace of XForms is essential to declare all XForms elements. The official namespace of XForms is *http://www.w3.org/2002/xforms.*

As Listing 2 shows the XForms: prefix is used to declare the *model* element. Yet there is no restrictions of choosing XForms namespaces prefix.

```
<html xmlns:xforms="http://www.w3.org/2002/xforms">
  <head>
    <title> document's title comes here </title>
      <xforms:model>
        The rest of model elements placed here.
      </xforms:model>
  </head>
```

Listing 2 usage of XForms namespace in declaring XForms elements.

XForms document is built up from two main parts. The first part describing the model of XForms document and the second part describing the form controls, Listing 3 illustrates a very simple example of an XForms document.

```
<html>
  <head>
    <xforms:model>
      <xforms:submission action="URL" method="post" id=”submitName”/>
      <xforms:instance xmlns="">
        <login>
          <id> </id>
          <password> </password>
        </login>
    </xforms:instance>
    <xforms:bind nodeset="id" required="true()" type="xs:string"/>
    <xforms:bind nodeset="password" required="true()" type="xs:string"/>
   </xforms:model>
  </head>
 <body>
       <xforms:input ref="id">
       <xforms:label> Enter your ID </xforms:label>
       </xforms:input>
       <xforms:secret ref="password">
       <xforms:label> Enter password </xforms:label>
       </xforms:secret>
    <xforms:submit submission=”submitName”>
    <xforms:label> Click here to login </xforms:label>
    </xforms:submit>
</body>
</html>
```

Listing 3 a simple example of XForms document

- **XForms Model**

XForms model consists of three main parts, XForms instance, bind elements and submission elements. One or more XForms instance can be defined in the XForms model. These instances can be defined either inline or by externally defined initial instance data. In our implemented prototype this instance is constructed automatically and dynamically from database metadata; and fed automatically to XForms model.

Binding elements are declared in the XForms model, they contain item properties. A model item property describes characteristics of each node of the XForms instance on

which it operates. Model item properties enforce each single node to meet numerous conditions. These conditions can depict data type, necessity of data (allow null values or not) and so forth. In our work, similar to XForms instance, binding elements are generated automatically based on database metadata.

Submission element determines which part of the XForms instance data should be serialised and how that should be made. Data can be submitted in several formats ranging from pure XML to XML with embedded data such as Word documents or images. In addition, for compatibility reasons XForms can submit data in the format of name/values pairs to allow XForms to communicate with server-side systems that were built to work with traditional HTML forms. In contrast with traditional HTML forms, XForms has no restrictions to submit all data provided by the end user in a single process. Numerous submission elements can be defined in XForms model which allow different parts of XForms instance to be submitted in several ways to different locations.

- **User Interface**

Since XForms does not depict how form controls should look, XForms processor in use is responsible for how these controls will be presented. Similarly to HTML, XForms offers several controls that allow users to input textual data or choose a single or multi items among several choices.

In the reminder section, brief description of the most used XForms controls are introduced citing the fundamental differences comparing to HTML controls.

- Input

Input control is quite similar to that one used with HTML forms. It allows users to enter any character data. Since XForms controls have the ability to use an XML Schema data type, user experience has been improved significantly. The input control may be returned in different shapes based on associated XML Schema data type. Listing 4 shows sample markup for two input controls with different data types, which resulted in different style of input controls, as shown in Figure 2 and Figure 3.

```
<xforms:input ref="name"> <!- - bound to node with XML Schema type
xs:string - ->
<xforms:label>First Name</xforms:label>
</xforms:input>

<xforms:input ref="age"> <!- - bound to node with XML Schema type
xs:date - ->
<xforms:label>Birthday</xforms:label>
</xforms:input>
```

Listing 4 sample of mark up of XForms input control

Figure 2 Xforms input control associate with XML Schemata data type xs:string

Figure 3 XForms input control associate with XML Schemata data type xs:date

The most cited difference comparing to traditional HTML forms controls is that all XForms controls must be associated with ***label*** element. The content of label element can be text, image or an external file. This richness of data type of label element can be very useful when XForms targeted people with visual disabilities.

- Output

This control cannot accept any user input. It renders data from an XForms model as inline text. The syntax of this control is shown in Listing 5 and its appearance is shown in Figure 4.

```
<xforms:output ref="fname">
    <xforms:label>Output Control= =></xforms:label>
</xforms:output>
```

Listing 5 sample markup for XForms output control

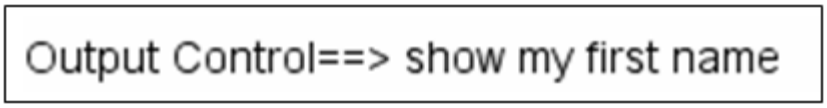


Figure 4 XForms output control.

- Secret

This form control is quite similar to its HTML forms equivalent. It does not offer any sort of encryption, rather offer only a cursory level of security. The syntax of this control is shown in Listing 6 and its appearance is shown in Figure 5.

```
<xforms:secret ref="password">
    <xforms:label>Enter password</xforms:label>
</xforms:secret>
```

Listing 6 sample markup for XForms secret control

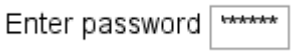


Figure 5 XForms secret control.

- Textarea

This form control has the same function of input control; the difference is that it is optimised for larger stretches of text. The syntax of this control is shown in Listing 7 and its appearance is shown in Figure 6.

```
<xforms:textarea ref="address">
<xforms:label>Address</xforms:label>
</xforms:textarea>
```

Listing 7 sample of markup for XForms textarea control

Figure 6 XForms textarea control.

- Select1

This control allows users to select only one item among several items. The rendered control in not bounded to a specific shape such as (radio buttons or checkboxes), rather it depends on XForms processor in use. The select control is supported by appearance attribute that may be used to meet specific demands of Web users. This attribute has several values, which are:

Appearance="minimal". This attribute value renders a minimal list.

Appearance="full". This attribute value renders all available choices.

Appearance="compact". This attribute value renders more compact list.

The syntax of this control is shown in Listing 8 and its appearance is shown in Figure 7.

```
<xforms:select1 ref="sex" appearance="full">
   <xforms:label> Gender </xforms:label>
    <xforms:item> <xforms:label> Male </xforms:label>
                  <xforms:value>m </xforms:value>
    </xforms:item>
    <xforms:item> <xforms:label> Female </xforms:label>
                  <xforms:value>f </xforms:value>
    </xforms:item>
</xforms:select1>
```

Listing 8 sample markup for XForms select1 control

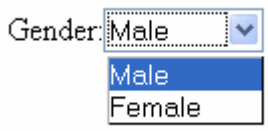


Figure 7 XForms select1 control.

- Select

This form control is similar to *select1* control; in addition, it allows users to select more than one item among several items. As *select1* control, the returned control relies on XForms processor to shape it. Listing 9 shows sample markup for XForms select

control and this sample is represented in two different shapes according to the web browsers used to invoke it as shown in Figure 8 and Figure 9.

```
<xforms:select ref="colours" appearance="full">
        <xforms:label> Select colors</xforms:label>
    <xforms:item>  <xforms:value>Red </xforms:value> </xforms:item>
    <xforms:item> <xforms:value>Orange </xforms:value>  </xforms:item>
    <xforms:item> <xforms:value>Yellow </xforms:value>  </xforms:item>
    <xforms:item> <xforms:value>Green </xforms:value>  </xforms:item>
    <xforms:item> <xforms:value>Blue </xforms:value>  </xforms:item>
</xforms:select>
```

Listing 9 sample markup for XForms select control

Figure 8 XForms select1 control invoked in X-Smiles Web browser.

Figure 9 XForms select1 control invoked in Mozilla firefox Web browser.

- Range

This control is very new and was not introduced in any form in traditional HTML. It provides as spontaneous method to enter a bounded value. This control is associated

with three attributes. *Start* and *end* which used to determine the upper and lower bounder. The suggested interval is determined by the attribute *step*. The syntax of this control is shown in Listing 10 and its appearance is shown in Figure 10.

```
<xforms:range ref="age1" start="1" end="5" step="1">
      <xforms:label> Grade </xforms:label>
</xforms:range>
```

Listing 10 sample markup for XForms range control

Figure 10 XForms range control.

- Submit

Submit control is quite similar to its HTML equivalent. However, since XForms document might contains more than one submit control, the submit parameters are taken from the element that match the IDREF specified on the attribute submission [18].

All XForms controls are supported by several elements that might be used to enhance the user experience. These elements are:

- ***Help***

  It contains a message which is presented based on an explicit request.

- ***Hint***

It contains a message which is shown at the discretions of the Xforms processor, for example, if the user exceeds the given time of hovering the mouse over a form control.

- ***Alert***

It contains a message that presented to the user in case of error condition, for instance, if a form control fails the validation.

2.1.6 XSLT

This section outlines the main features, structure and the mechanism of how XSLT works.

2.1.6.1 Introduction

Nowadays, a huge amount of data is stored, transported and exchanged in a form of XML. Having data in a form of XML does not mean this data becomes immediately useful. Data needed to be manipulated, stored, retrieved and presented to the Web users in a meaningful and friendly look. XML data intended to be viewed as HTML pages on the Web, as PDF [33] documents for printing and as WML [34] pages on WAP mobile phones [35]. Thus, a tool for transforming XML documents into different formats running on several platforms is needed.

Although XML documents can be transformed using general programming languages such as Java, C++ and C, XSLT has the advantages of being more lightweight than these languages. It allows Web developers to code programs much smaller than in traditional programming languages.

Since our proposed prototype relies heavily on using XSLT as described in (5.5.3.3), the reminder of this section introduces XSLT and its mate, XPath, in brief considering only features that used in implementing the proposed prototype.

2.1.6.2 XSLT document's syntax and structure

The main purpose of XSLT is to transform XML document into another XML document or into another format such as HTML, PDF and etc. in order for the transformation process to succeed, three components must be integrated with each other: an XML document, a XSLT stylesheet and XSLT processor as Figure 11 demonstrates.

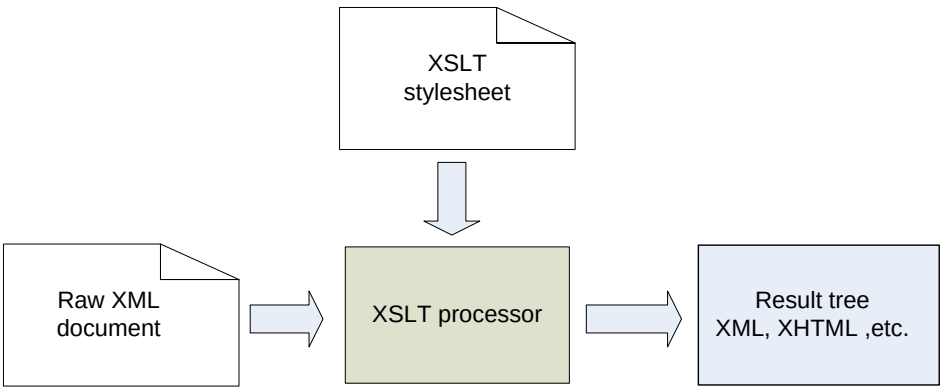


Figure 11 XSLT transformation process

XSLT document obeys normal rules of XML document, which means it must be a well-formed XML document that conform to W3C standards of syntax. As XML document, XSLT document must have a single document element, which in this case is *xsl:stylesheet*. Although the prefix *xsl:* is widely used, it is not obligatory to stick with it.

The mechanism of how XSLT works to transform an XML document into other formats is introduced in (chapter 5). A brief introductory of XSLT structure and most used statements are described in this section.

2.1.6.2.1 XSLT structure

As shown in Listing 11, XSLT consists of three main parts: document element, output element and main body.

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="text"/>
<!-- comments can be placed anywhere between document element -->
<!- - main body of XSLT placed here - - >
</xsl:stylesheet>
```

Listing 11 general structure of XSLT stylesheet

- ***Document element*:** `<xsl:stylesheet version="1.0" xmlns:xsl=http://www.w3.org/1999/XSL/Transform>`
  it has two attributes :

  version which indicates the version of XSLT in use.

  xmlns:xsl which declares the XML namespace, defining the meaning of the xsl:prefix.

- ***Output element:*** The *xsl:output* specifies the type of target document.
- ***XSLT's main body:*** XSLT's main body is made up of a number of templates, each of which matches a specific part of the source XML document and process it according to what it instructed. Templates are rules, which define how a specific part of the raw XML document maps on to the desired result.

2.1.6.2.2 XSLT's Elements

XSLT comes with many useful elements that facilitate the transformation process. The following elements are the most used in our implemented approach:

- *<xsl:template>:* Defines a reusable template for generating the wanted output for nodes of a specific type and context. This element is always associated with the *match* attribute that identifies the source node or nodes to which the rule applies. The syntax of this element is: *<xsl:template match="/">*. The value of attribute *match* could be any node on XML document specified by XPath expression.
- *<xsl:apply-templates>*: Directs the XSLT processor to find the suitable template to apply, based on the type and context of each selected node. This element is associated with two attributes *select* and *mode.* The syntax of this elements looks like *<xsl:apply-templates select="Expression ">*.

  Expression could be used to select nodes for processing and specifying conditions for several ways of processing a node.
- *<xsl:copy>:* Copies the current node from the source document to the output.
- *<xsl:copy-of>:* Inserts sub trees and result tree fragments into the result tree. It always associated with *select* attribute. The syntax of this element looks like *<xsl:copy-of select = Expression />*
- *<xsl:value-of>:* Inserts the value of a particular node as text. The common used attribute with this element is *select.* The syntax of this element is *<xsl:value-of select = Expression />*

- *<xsl:variable>:* Specifies a value bound in an expression. It is associated with two variables: *name* and *select.* The syntax of this element is *<xsl:variable* name= *Qualified name* select=*Expression*/>. Where qualified name is constructed of prefix and a local part.

In addition to above elements, XSLT as other general programming languages offers a set of elements that can be used in looping, branching and taking decisions. These elements are listed as follows:

- *<xsl:for-each>:* Applies a template repeatedly. It associated with required attribute *select.* The syntax of this element looks like *<xsl:for-each* select=*Expression*>I. where the expression is evaluated on the current context to decide the set of nodes to loop over.
- *<xsl:if>:* Allows simple conditional template fragments. It associated with required attribute *test.* The syntax of this element is *<xsl:if* test= *Boolean expression />*.
- *<xsl:choose>:* Tests compound conditions in conjunction with *<xsl:when>* and *<xsl:otherwise>* attributes. The syntax of this element is shown in Figure 12.

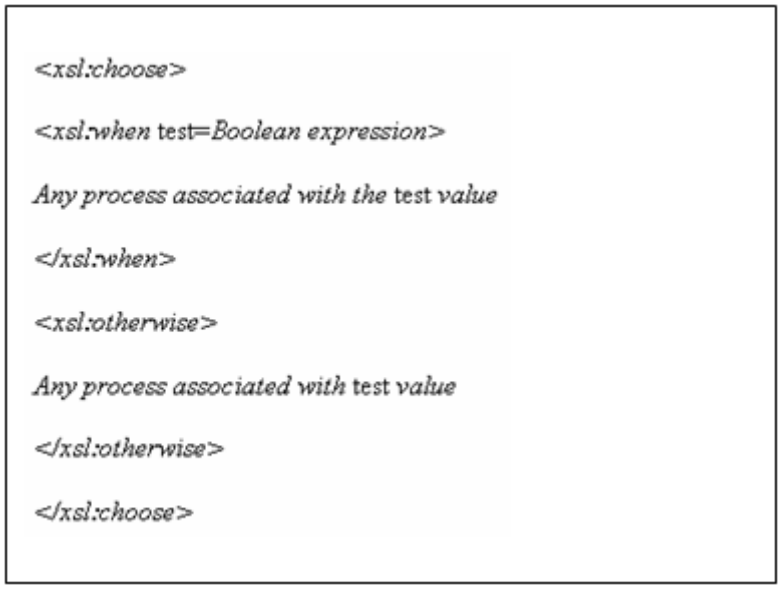


Figure 12 Syntax of *choose element*

2.1.7 XPath

The first version of XPath has become a W3C recommendation on November 1999 [36]. XSLT depends heavily on XPath to identify subsets of the source XML document tree. XPath uses path expressions to navigate in XML document similar to those used with traditional computer systems. XPath comes with plenty of built-in functions to deal with all sorts of data types. Since the main use of XPath in XSLT is to identify a specific XML nodes with specific characteristics, selecting a particular node might be performed by using a suitable path expression. The code fragment below illustrates of using an XPath expression in conjunction with an XSLT statement for accessing and retrieving a particular XML node value.

<xsl:value-of select="data/node1" />

Table 1 lists the most common used expressions in XPath regarding to XSLT.

Expression	Description
nodename	Selects all child nodes of the named node
.	Selects the present node
..	Selects the parent of the present node
/	Selects from the root node
//	Selects nodes in the document from the current node that match the selection no matter where they are
@	Selects attributes
*	Matches any element node
@*	Matches any attribute node
Node()	Matches any node of any kind

Table 1 Functions of XPath expressions. Captured from [37].

2.2 Client-side Scripts

The term “Scripts” refers to a general programming term for short, text-based software programs. Client-side Scripts are embedded in HTML WebPages and executed on the client side of the client/server architecture, rather than on the Web server. Client-side Scripts languages are light weight programming languages and do not have the complete functionality that is available to full-fledged programming languages such as Java, C, and C++. For instance, JavaScript does not support a database access. These scripting languages are used to perform some tasks on the client-side such as data validation, do some simple computations, change text style and so forth. An available Scripting languages include VBScript [38], Jscript [39], JavaScript [8]. The latter one is the most widely used scripting language and was chosen to implement our prototype.

2.2.1 JavaScript

JavaScript is an interpreted scripting language originally developed by Netscape. Although it shares many features and structures of the full Java language, it was

developed independently. JavaScript allows executable content to be embedded in Web browsers for client side execution. It is not intended to draw graphics, or perform networking or file I/O but to control browser behaviour and content.

The core JavaScript language was standardised in the ECMA-262 standard [40]. JavaScript interpreters are embedded in the most popular Web browsers such as Internet Explorer, Mozilla firefox and Netscape. JavaScript interpreters in these browsers provide many features beyond those in the ECMA specifications, which concentrates more on the core of the language syntax.

Browser compatibility is the widely cited problem with JavaScript. Mainly, there are two significant problems with JavaScript and browsers:

- Numerous JavaScript versions in different browsers. Despite the fact that the names and the numbers of JavaScript's versions supported by Microsoft and Netscape, the languages themselves are broadly, but not entirely, the same. Small and subtle differences can result in broken web pages.
- Browser programmability. Access to the browser's elements and features is determined by its own level of programmability.

Despite the fact that JavaScript can also be implemented at server-side, this approach has many drawbacks including the intrinsic limitations of the language and the requirement for the JavaScript runtime engine on the web server.

Despite all JavaScript compatibility issues, Web developers are left with no choice but to make use of it or its equivalents in order to perform basic processes on the client side such data input validations. However, XForms is a new trend which meant to present an elegant solution for these issues. Theoretically, JavaScript's role will be gradually

decremented as much as investments adopted to the major Web browsers that will allow them to natively support XForms and as many people as getting familiar with this technology.

2.3 Server-side technologies

Although there are large numbers of server-side technologies available to Web developers, this section focuses on the most commonly used technologies and the main focus will be on JavaServlets chosen to implement our prototype system.

2.3.1 CGI

CGI stands for Common Gateway Interface, it is a protocol which allows a web server to obtain data from or send data to databases, documents and other programs and present that data to end users via the Web. A CGI program receives input from the client's browser, via interaction with the Web server, by reading environment variables and or standard input, and produces HTML page output via interaction with the Web server, by writing to standard output. Although CGI programs can be written in any programming language such as (C, C++, Perl, Visual Basic), Perl is one of the most popular and widely used languages.

Despite the fact that CGI has made a big impact in the development of dynamic Web pages and has driven this sector for many years because it has a number of benefits including ease of implementation, the use of standard Web browsers as clients and the existence of a wealth of existing tools and sample code, it has many drawbacks which

are making CGI technology fade over time and let competitor technologies such as ASP, JSP, PHP, and JavaServlets overtake its role. The following are the most cited CGI drawbacks:

- Each request to a CGI application spawns a new process which leads to resources problems when the server is heavily visited.
- CGI programs when compiled become device dependant and if interpreted may lead to efficiency problems.
- Since each CGI process is shutdown after each request, this makes maintaining state too difficult.
- CGI has security vulnerabilities which make many ISPs restrict access to CGI scripts.

2.3.2 Fast CGI

FastCGI is a fast, secure Web server interface, and open extension to CGI that provides better performance without the limitation and complexity of server specific APIs [41]. Fundamentally, FastCGI is very similar to CGI, with a major difference: FastCGI processes are persistent; they need to be started prior to serving any requests, and after finishing one, they wait for a new request instead of terminating. A process manager is needed to control all processes created by FastCGI. It works with heuristics to estimate when new instances of applications need to be created. This makes FastCGI applications difficult to develop, because special directives need to be included in the web server configuration files. In addition to the previous problem, FastCGI is not implemented for some of the most popular Web Servers such as Microsoft's IIS [42].

2.3.3 Java Servlets

Java Servlets are small, platform independent server-side programs that programmatically extend the functionality of a Web server. The Servlet API was developed to take advantages of the Java platform to solve the issue of CGI and proprietary APIs.

2.3.3.1 How it works

Since a Servlet is a Java class, it needs to be executed in a Java Virtual Machine (JVM) by a service called a Servlet engine. When a Servlet is invoked via an HTTP GET or POST request from a client the web server directs the HTTP request to the Servlet engine. If the desired class is not already in memory, the Servlet engine loads and initialises it and it stays in memory until it is unloaded or the Servlet engine is stopped, consequently it handling more than one request without having to reload it. The Servlet engine then encapsulates the HTTP request into a class called HTTPServletRequest and delivers it to doGet or doPost methods. The Servlet responds by writing HTML into HTTPServletResponse which is sent back to the web server and delivered back to the HTTP client.

2.3.3.2 Servlets Web Servers

Nowadays, most web servers support Servlets in some way. Some of them provide a full-featured web server and possibly many other capabilities such as Sun

Microsysystem Java Server Web Development Kit (JSWDK) [43], W3C's Jigsaw [44] and Apache Tomcat [45]. Others operate as a add-on for other web servers, Netscape's Enterprise Server [46], Microsoft Internet Information Server (IIS) [47] and Apache [48] are some examples of the second type of Web servers.

The central abstraction in the servlet API is the Servlet interface, which is implemented by all servlets, either directly, or more commonly by extending a class that implements it such as HTTPServlet and GenericServlet. Developers are more likely to override HTTPServlet to implement their servlet [49].

When a servlet is invoked, it is loaded by the server and runs the init method. Initialisation is allowed to complete before client requests are handled or the Servlet is destroyed. Since the server calls init method once when it loads the servlet and will not call it again unless it is reloading the servlet, the concurrency issue is already solved. Once the init method is loaded and initialised client requests are ready to be handled, the service method is responsible for processing the requests. Since Servlets are capable of running multiple service methods at a time, the SingleThreadModel interface is needed to be implemented to overcome the concurrency issue. Servlets run until they are removed from service, the destroy method is called to terminate the Servlet.

In functionality, Servlets lie somewhere between Common Gateway Interface (CGI) programs and proprietary server extensions such as the Netscape Server API (NSAPI). However, Servlets have many advantages over other server extension mechanisms which are listed as following:

- They are faster than CGI scripts and Fast CGI scripts since they use a different process model. Since there is only a single instance which answers all requests concurrently, this saves memory and allows a Servlet to easily manage persistent data.
- They use a standard API that is supported by many web servers.
- They inherit all the advantages of the Java language, including ease of development and platform independence.
- They have the ability to access the large set of APIs available to the Java platform such as JDBC API to access enterprise database, Enterprise JavaBeans and others.
- Inexpensive, there are a number of free or very inexpensive Web servers, which support servlets available that are good for personal use or low-volume web sites.

For all above features, JavaServlets technology was chosen to implement the proposed prototype. Using JavaServlets makes our approach portable across operating systems and Web servers. Moreover, it allowed access to the full range of Java APIs such as JDBC to access enterprise databases.

2.4 Server-side scripting technologies

This section surveys the commonly used server-side technologies in use for developing web applications. These technologies are similar in concept but vary in term of performance and reliability.

2.4.1 Active server pages (ASP)

Active Server Pages is a Web server scripting language developed by Microsoft used for embedding dynamic content into HTML Web pages. ASP is language independent. Active Server Pages enables server side scripting for IIS with native support for both Jscript and VBScript. The ASP code is executed on the server and the content is translated into HTML before sending it to a Web browser.

ASP is implemented as an ISAPI (Internet Server API) application integrated into Microsoft Information Server (IIS), which implies that ASP technology is basically restricted to Microsoft Windows based platforms. However, ASP technology can be targeted to other platforms via third-party porting products.

2.4.2 ASP.NET

The philosophy behind the .NET Framework is to create globally distributed software with internet functionality and interoperability. The .NET Framework consists of many class libraries, includes multiple language support and a common execution platform. ASP.NET is built into this framework. ASP.NET pages execute on the server and generate mark-up such as HTML, WML, or XML that is sent to a desktop or

mobile browser [50]. In comparison to ASP, many differences between these two technologies are reported including:

- Unlike ASP, ASP.NET code is a compiled CLR (Common Language Runtime) instead of interpreted code. Since the ASP.NET code is complied the first time the page is requested and a copy of the compiled page is saved for the next time it is requested. This makes the latter technology faster than the former. This concept is similar to that one adopted in Java technologies. For instance, JavaServlet code is compiled to JVM at the first time the page is invoked and remains available to next requests. Similarly, when JSP is invoked, it is automatically converted to Servlet and compiled to JVM code.
- ASP.NET makes for easy deployment. There is no need to register components because the configuration information is built-in.
- Unlike ASP, ASP.NET supports a fully separation between logic and presentation. It consists of two main layers, .NET language such as (C#, C++, VB, etc.) and presentation layer. The later consists of WebForms and WebControls. WebForms can bee seen as a workspace where you draw controls. Building Web user interfaces employ placing selected WebControls onto WebForm. WebForms controls are created and run on the server-side. After executing whatever operation they are intended to do, they render the suitable HTML and send that HTML into output stream. WebForm Controls do not require mapping to any particular mark-up language. For instance, *DropDownList* control will be rendered as <*select*> and <*option*> tags when sent to a browser. Yet, if the target is a portable phone, the same control might

render WML. Although ASP.NET is very rich in built-in WebForm Controls such as (*TextBox, Button,* etc.), it is possible to Web applications' developers to build their own custom controls. In addition to previous Controls, ASP.NET framework introduces *Field Validator Controls* that can be used to validate data on the client browser to minimise the round trip between the client and the server. requiredFieldValidator, CompareValidator and RangeValidator are example of these controls.

2.4.3 PHP

PHP stands for **PHP**: **H**ypertext **P**re-processor. It is an open-source, server-side scripting language designed for creating dynamic Web pages [9] . It similar to JSP and ASP i.e. its code appears in HTML pages embedded within simple delimiters. PHP provides native support for the most common used databases like MYSQL, Oracle 8, PostgreSQL, Sybase, MSSQL and Informix. In addition, databases that are not natively supported can be connected via protocol-based functions; these include DBM style, LDAP and ODBC. Although PHP has come with high level features like sessions in a nice abstract way, its database access is very DBMS specific and does not provide metadata access except via specific queries on known system catalogues [51]. For instance, connecting to PostgreSQL DBMS requires using the following function:

pg_connect("dbname=XXX user=XXX password=XXX");

While connecting to MySQL DBMS requires using a different function as shown below:

mysql_connect('localhost', 'root', 'password');

Regardless to the type or number of parameters, the used functions are totally different in syntax.

2.4.4 Perl

Perl is well known as “Practical Extraction Report Language” [52]. It is an interpreted language. It gained a wide popularity since it was the most commonly used language for writing CGI scripts to generate dynamic Web pages. Dynamic Web applications require a straightforward mechanism for connecting to many different DBMSs in such dynamic way. Perl provides a very powerful tool that connects Perl scripts with different DBMSs. DBI (Database Independent Interface) is an interface between an application and one or more database driver modules. It allows a Perl application to talk to several types of DBMSs using the same method, variable and convention. As shown in Figure 13, DBI locates the database driver module for a specific DBMS and dynamically loads the suitable DBD (Database Driver) which contains the required libraries to talk with a particular database.

Once the DBI is loaded, the Perl application performs the reminding steps:

- Connect to a particular database.
- Prepare a query.
- Execute the query.
- Get the results
- Finally, close the connection.

This concept is quite similar to that one adopted in JDBC. (See section 3.2).

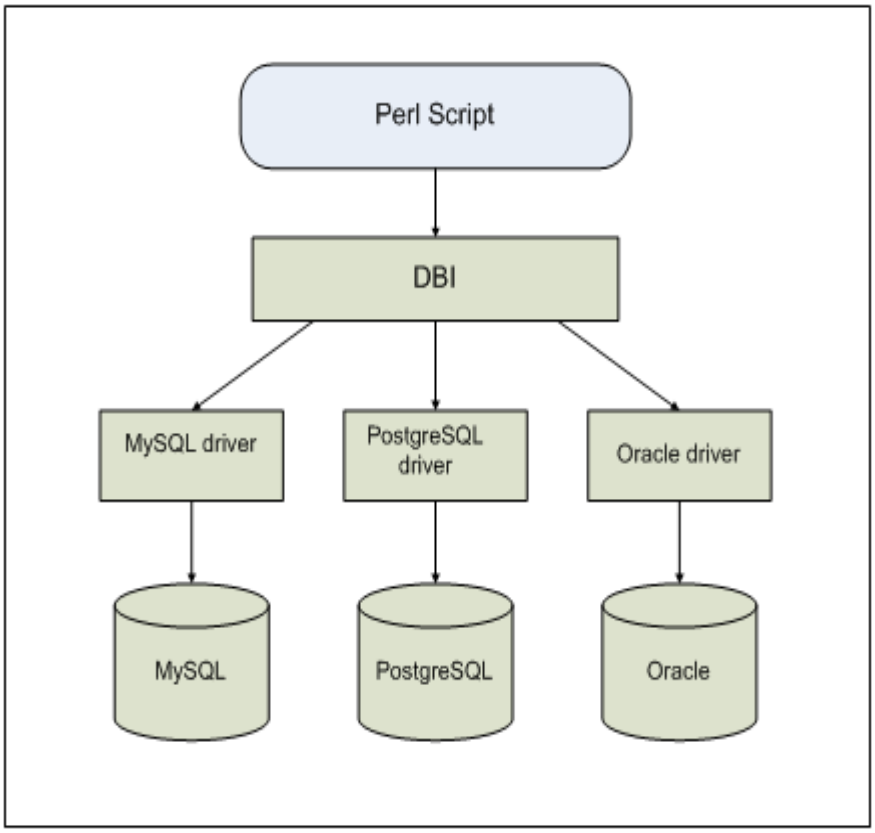


Figure 13 role of DBI for accessing database in Perl applications

2.4.5 Java Server Pages:

Java Server Pages (JSP) is a technology that enables rapid development of web-based applications that are server and platform independent [10]. JSP can be seen as a high-level abstraction of Java Servlets that is implemented as an extension of the Servlet API, this is because the JSP is compiled to Servlet by the container when it is invoked the first time. A JSP document is a text-based file that mixes JSP technology-specific and custom tags, in combination with other static (HTML or XML) tags. This document is interpreted by a JSP engine and the result is sent back to the client in a form of HTML or XML pages.

Due to the fact that our approach aims at generating different forms of Web forms by totally separating the presentation from logic and content, JSP was not considered to be used for implementation of the prototype system coded trough this study.

Chapter 3

Database Development

3.0 Introduction

The interaction of a database and the Web is becoming the cornerstone of developing Web application systems. Relational database systems are the most common form of database used for this purpose. Dynamic Web applications are the mainstream in Web development. This trend requires using database metadata. Database metadata can be extracted by several tools according to developers and users demands. JDBC is one of most used tools for this purpose. This chapter introduces a very short introduction to relational databases. Since our approach depends heavily on using JDBC for the extraction of relational database metadata, this tool is discussed in greater detail.

3.1 Relational Database, an overview

The main function of any database is storing data in such a way that this data can be retrieved from it. Databases come in many models including:

- A relational database.
- An object-oriented database.
- A hierarchical database.
- A hybrid database.

While all of these different types of databases are in use for many different purposes, it is most common to use relational databases for Web applications due to their stability and speed. The simplest description of a relational database is one that presents data in tables with rows and columns. Each column contains only one piece of data. For instance, Table 2 illustrates relational database table that consists of five columns: Employee_Number, Name, Gender, Date_of_Birth and Car_Number. This table consists of three rows, each one representing a different employee.

Employee_Number	Name	Gender	Date_of_Birth	Car_Number
12012	Adam Smith	M	12-12-1970	21
12013	Sarah King	F	11-11-1985	36
12014	Ali Khan	M	13-08-1981	null

Table 2 Employee table

Every relational database table must have a primary key that could be a very simple one consisting of a single column or a composite one consisting of more than one column. The primary key in Table 2 is *Employee_Number*. Data in a relational database is stored

in several tables instead of using a single table. The processing of organising data in several tables is called normalisation. There are two main benefits of this process:

- Eliminating redundant data (i.e. storing the same data in more than one table).
- Ensuring data dependencies (i.e. only storing related data in a table).

A distinguishing feature of a relational database is that it is possible to get data from more than one table, this process is called *join*. For instance, if we wanted to know which employee has a company car and which car the employee has, including the make, model, colour and year of the car. The cars information is stored to a particular table called *cars* and illustrated in Table 3.

Car_Number	Make	Model	Colour	year
21	Nissan	Sunny	Black	2006
36	Toyota	Crown	Red	2005

Table 3 Cars table

The above tables are joined by a common column which appears in both of them. This column, which must be the primary key in one table, is called the foreign key in the other table. As seen in Table 3, *Car_Number* is the primary key and is a foreign key in the Table 2.

Besides storing data, RDBMSs store data about the data itself in system tables or catalogs. This data is known as a metadata. RDBMSs use metadata to maintain integrity and keep data accurate and reliable all the time. RDBMSs enforce the integrity constraints each time data is inserted, updated or deleted. For this aim a set of integrity rules are applied for each relational table and are tested against the metadata. Among these rules are:

1. Duplicated rows are not allowed in a relational table.
2. Null values are permitted only under specific constrains.
3. Any column that is part of any primary key must not be null.
4. Each column can accommodate only one data type.

For example, if the car number 36 in Table 3 is no longer owned by the company and needed to be removed from the *cars* table, it must be removed from *Employee* table in order to maintain what is called referential integrity. A foreign key must either be null or equal to an existing primary key value of the table to which is refers.

A relational database must provide access to its structure through the same tools that are used to access the data. Each database includes a set of system catalog tables, which describe the logical and physical structure of the data. These tables contain information about the definitions of database objects such as user tables, views and indexes. They are created when the database is created, they can be queried by any user but can not be explicitly created or updated. Some DMBS distinguish the system tables by giving them a special prefix like “pg_” for PostgreSQL system tables [53]. Others (SQL compatible ones) put the system tables in a separate schema such as INFORMATION_SCHEMA.

In standard SQL, the data that describes the database is stored in schema called INFORMATION_SCHEMA. The information schema consists of a set of views, exposing metadata in a relational format. This allows executing SELECT statements to retrieve or to format metadata. The information schema automatically exists in all databases. We are allowed to query the information schema, but we are not allowed to change its structure or modify its data [54].

Information schema is written on top of System catalogs and written in an easy and readable manner. For instance, a query that will return the names of the fields of a table using SYSTEM CATALOGS is as follows:

```
SELECT a.attname from pg_class c, pg_attribute a, pg_type t
       WHERE c.relname=' table name '
       AND a.attnum > 0
       AND a.attrelid = c.oid
       AND a.atttypid = t.oid
```

While using INFORMATION_SCHEMA the query is as follows:

```
SELECT column_name
        FROM information_schema.columns
        WHERE table_name = 'table name'
```

3.2 JDBC and its Architecture

The commonly used technologies to access databases from Java applications are JDBC (Java API for database connection used by Java programs)[1] and ODBC (Open Database Connectivity). They are types of database access middleware. Practically, JDBC is the more powerful technology in use with a Java application due to Java's nature. JDBC is an abstraction layer defined by Javasoft that provides a standard SQL-based interface to any data source[55]. Database vendors or other third parties provide the actual implementation of these interfaces in the form of JDBC drivers. In the real

[1] JDBC is just a trademark name and Sun says it is not acronym of Java Database Connectivity

environment, JDBC is constructed from four main components: the application, driver manager, driver and data source as Figure 14 illustrates.

- The application, which invokes JDBC methods to send SQL statements to the database and retrieve results.
- Driver manger loads specific drivers for the user application.
- Driver processes JDBC methods invocations, sends SQL statements to a particular data source, and returns results back to the application that handles the database interface. JDBC Drivers are classified into four types, which are:
    1. Type 1- JDBC to ODBC Bridge: - the function of this type of JDBC Driver is to translate JDBC methods calls to ODBC function calls and makes it possible to access any ODBC data source through JDBC. Since this style of driver uses multiple levels of translation this results in a poor performance. Another problem associated with this style of driver is that, it depends on native libraries of the underlying operating system which makes it platform-dependent. This makes any application that uses this driver non-portable as the ODBC driver for the specific operating system is required in the client machine.
    2. Type 2- Native API, Partly Java Driver: This sort of driver translates JDBC into calls to a native data API. Performance is generally better than with Type1 drivers due to one less translation layer. Since this style of driver requires the database vendor's proprietary library on each client computer, the deployment problems remain unsolved.

3. Type 3 - Network Protocol, All Java Driver: This driver translates JDBC calls into a DBMS-independent network protocol that a middle-tier server translates into a DBMS-specific protocol. This flexible driver is well suited for distributed three-tier architectures. Performance issue still remains because the middle tier may itself use a Type 1 or Type 2 driver to access the database.
4. Type 4 - Native Protocol, All Java Driver: This driver converts JDBC calls directly into the network protocol used by a specific DBMS. Performance is usually good because native database calls are made directly over the network.

- Data source is a particular database where user data resides.

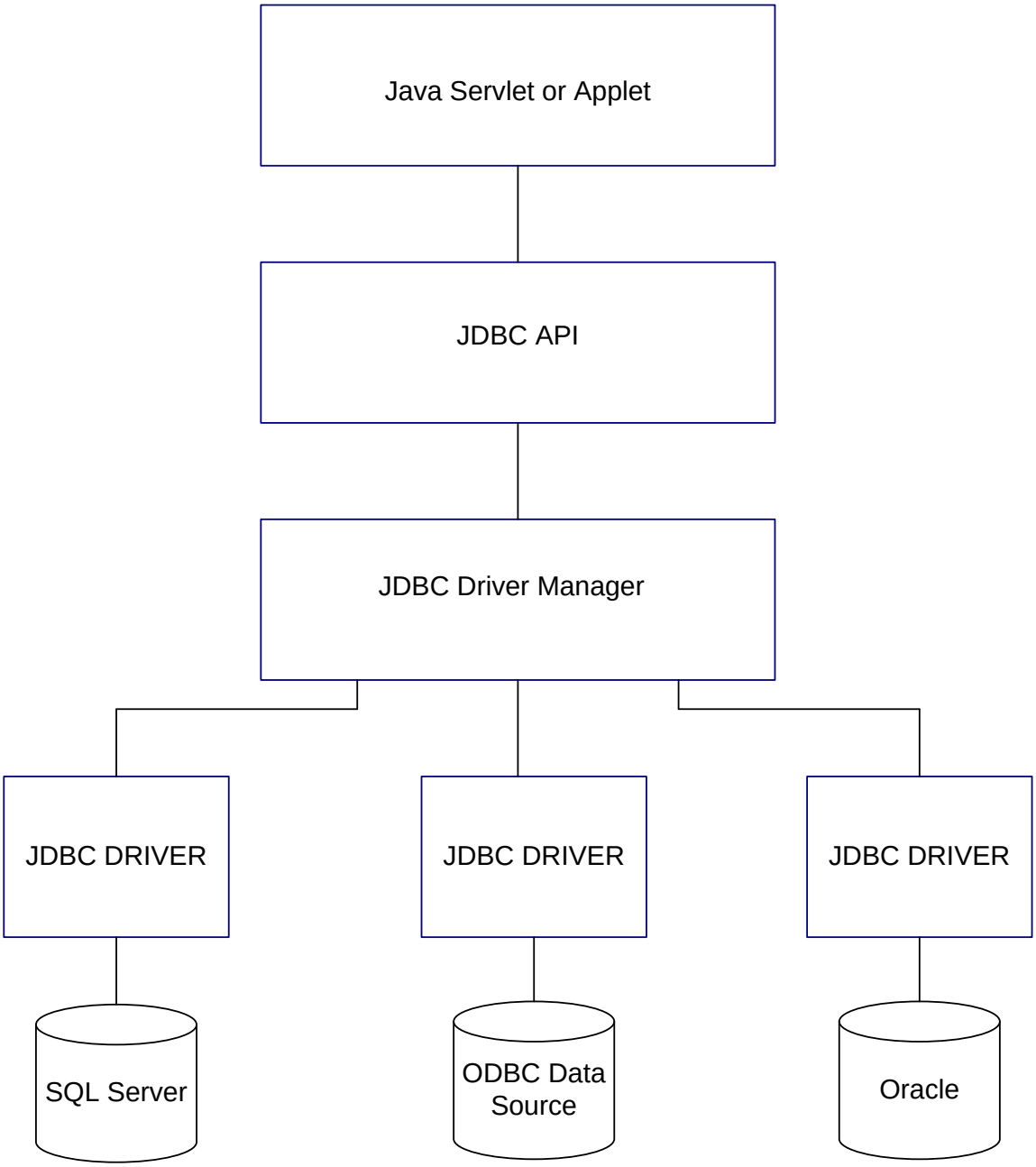


Figure 14 Components of the JDBC Architecture

3.2.1 JDBC vs. ODBC

ODBC is a Microsoft product implementing a CLI (Call Level Interface). A CLI application uses a standard set of functions to execute SQL statements and related services at runtime. Adopting a CLI approach allows Web developers to develop portable applications that are fully independent of database vendors and can be distributed in binary form. ODBC supports this tendency and allows Web developers to develop applications that are independent of any specific DBMS.

JDBC is quite similar to ODBC. It is a portable, open and published API that uses drivers to target specific databases. JDBC API is a natural Java interface to the basic SQL abstractions and concepts [56]. It takes the advantages of ODBC and builds upon it instead of starting from scratch. In fact, both interfaces are based on the X/Open SQL CLI [57] . However, the main difference between JDBC and ODBC is that, unlike the later, the former builds on and reinforces the style and virtues of Java.

Since ODBC uses a C interface, it is considered an unsuitable solution for direct use from Java. This is because, indirect usage of ODBC from Java, using calls from Java to native C code has many shortcomings in term of security, portability, implementation and robustness. However, JDBC is designed to reduce these drawbacks, and will not only allow applications, which are independent of the database product but will also allow machine independent applications to be written.

When ODBC is used, the ODBC driver manager and drivers must be manually installed on every client machine. Unlike ODBC, JDBC requires zero configuration on client

side, its code is fully automatically installable, portable, and secure on all Java platforms from network computers to mainframes [58].

As a conclusion, JDBC is a reliable tool and can be used for any database system as long as a driver exists. It is not limited to relational database but includes object relational databases and even non-relational technology such as IBM's IMS [59].

3.2.2 How JDBC works?

Accessing and retrieving information from databases using JDBC involves several processes as described below and shown in Figure 15.

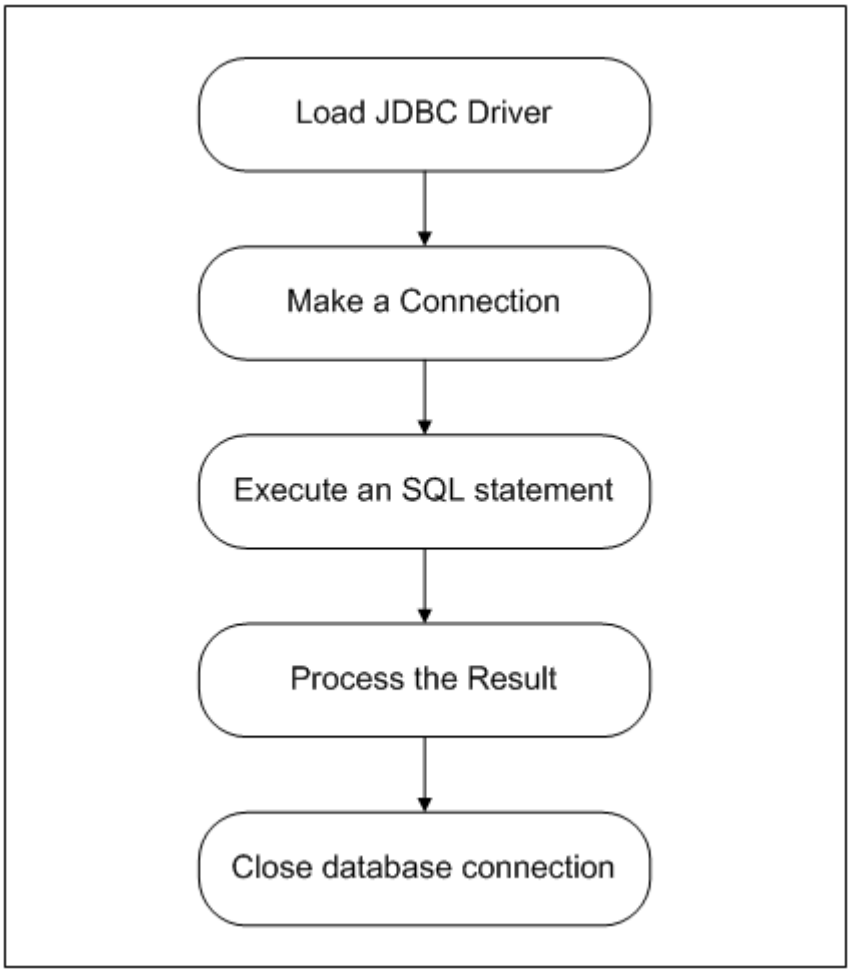


Figure 15 flows of JDBC processes

1. **Registering and loading the JDBC Driver**

   It is essential to register and load a JDBC driver with the driver manager before establishing any connection with database. The driver manager's job is to maintain a reference to all driver objects that are available to JDBC clients. The registration task is very straightforward and it is made automatically when a JDBC driver is loaded. Loading the JDBC driver requires using only one method. For instance, the following method can be called to load any selected a JDBC driver.

   ```
   Class.forName(Driver);
   ```

   Driver parameter is any desired driver.

   As an example the following Java statement

   ```
   String Driver="org.postgresql.Driver";
   ```

   can be used to load a PostgreSQL driver, where as

   ```
   String Driver="org.gjt.mm.mysql.Driver";
   ```

   can be used with MySQL driver.

2. **Establish a Database Connection**

   When the JDBC driver is loaded, establishing connection to database can be made using the following line of code.

   ```
   Connection c = DriverManager.getConnection(url, "userID", "password");
   ```

   The parameter *url* tells the driver manager which driver and data source to use. The formal syntax of database *url* is:

   ```
   jdbc:subprotocol:subname
   ```

The ***subprotocol*** is the name of a valid JDBC driver. The ***subname*** is normally a logical name or pseudonym that maps to a physical database. For instance, if ***mysql*** driver is used to access a MySQL database called ***"studentinfo"***, the connection process could be

```
Connection c = DriverManager.getConnection(jdbc:mysql://
studentinfo/ "userID", "password");
```

The returned connection is an open connection that can be used to produce JDBC statements that pass SQL statement to the DBMS.

3. Executing SQL statements

A statement is defined as an object that sends an SQL statement to the DBMS. For a statement to be executed it must be created first. It is created as shown below.

```
Statement  statement = con. createStatement();
```

The statement object provides methods to perform different operations against a database. An example of these method is ***executeQuery().*** It accepts an SQL ***SELECT*** statement and returns a ***ResultSet*** object containing the database rows extracted by the query. ***Update()*** method is another example which can be used for insertion, deleting or updating tasks. The ***ResultSet*** object can be created like this:

```
ResultSet result = statement executeQuery("select * from
mytable");
```

4. Process the Results

The whole result produced by executing an SQL statement is stored in ***ResultSet*** object. Retrieving or processing this set of result requires using several methods of the ***ResultSet*** object including ***next(), previous() and getObject()*** methods.

The Figure 16 demonstrates the flows accessing and retrieving information from ***ResultSet*** object. n***ext()*** and ***previous()*** methods are used to traverse through the ***ResultSet*** object using either the column name or the column number, while ***getObject()*** method and all of the data type-specific "***get***" methods are used to extract the actual value of data. For instance, if the first column of the ***ResultSet*** object ***resultset*** is called "student_no" and its value is associated to integer data type, second column is named "student_name" and its value is associated to string data type, either of the following statements can be used to extract the value of each column.

```
int student_no = resultset.getInt(1);
int student_no = resultset.getInt("student_no");
String student_name = resultset.getString(2);
String student_name = resultset.getString("student_name");
```

However, Java data types are not accurately isomorphic to JDBC data types and SQL data types. For instance, a Java *String* object does not precisely match any of the JDBC *CHAR* types, but it gives enough type information to represent *CHAR, VARCHAR, or LONGVARCHAR* successfully.

Nevertheless, the JDBC driver is capable of converting most of the underlying data to the particular Java type and then returning an appropriate Java value as shown in Table 4.

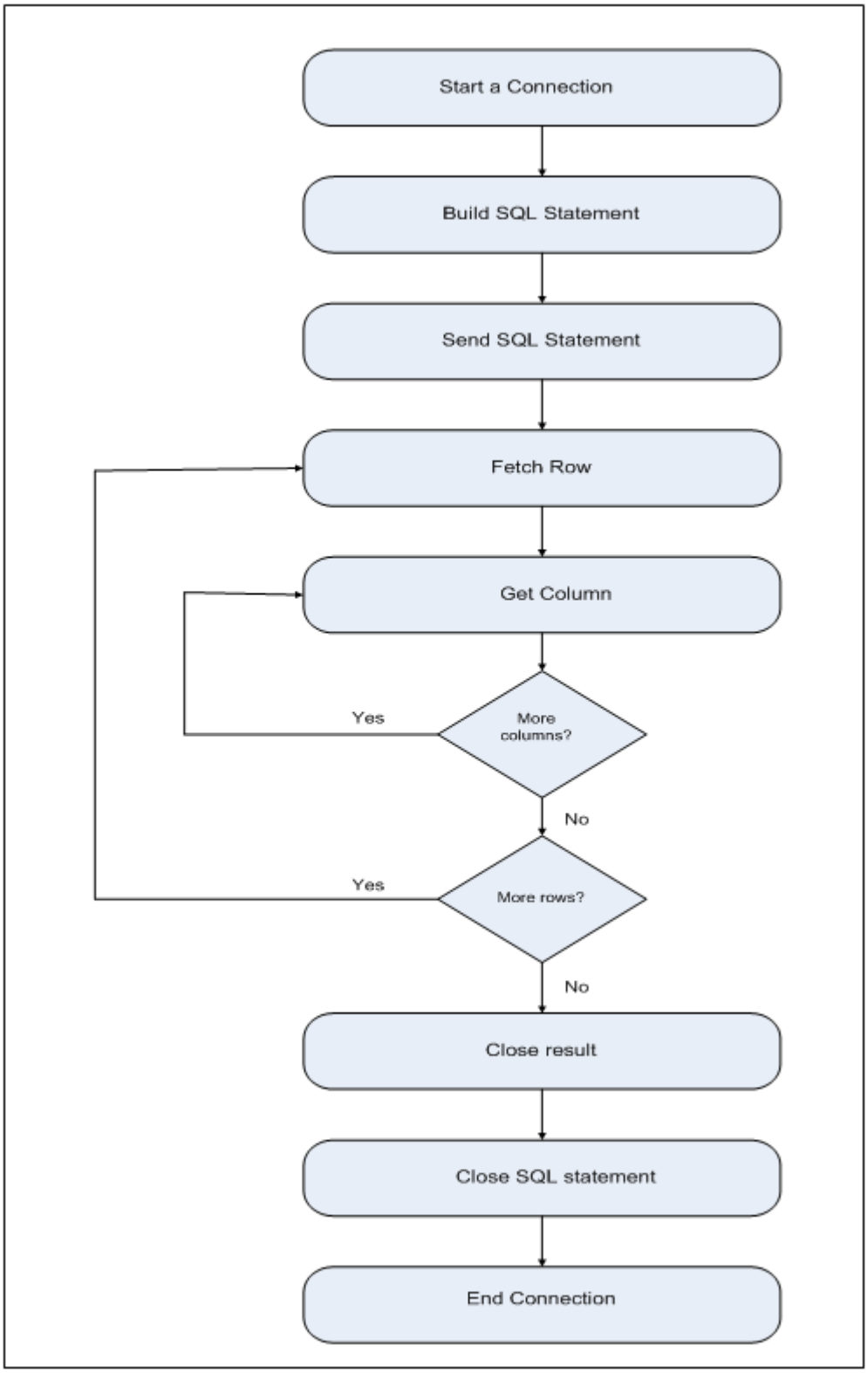


Figure 16 flow of accessing and retrieving information from ResultSet object.

JDBC Type	Java Type
CHAR	String
VARCHAR	String
LONGVARCHAR	String
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT	double
DOUBLE	double
BINARY	byte[]
VARBINARY	byte[]
LONGVARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

Table 4 JDBC Types Mapped to Java Types. Adopted from [56]

The ***ResultSet*** object is reusable and it is tied to the ***Statement*** object that created it. If ***ResultSet*** object used to execute another query, the ***ResultSet*** is closed automatically.

5. Closing Database Connection

Despite the fact that the Java virtual machine's garbage collector releases all resource that are no longer in use, it is recommended to close any connection

when processing is complete. This practise includes closing ResultSet, Statement objects and database itself. A ***close()*** can be used for all objects. The syntax of this method is as follows.

```
statement.close(); // close Statement after use
resultset.close(); // close ResultSet after use
dbconnection.close(); // close database connection in finally stage.
```

However, practically closing database after ending of each request results in overhead associated with creating a new connection for each request. A connection-pooling strategy can be obtained to solve this issue.

3.2.3 JDBC and Database Metadata

Developing dynamic Web applications requires using database metadata in an efficient way. Metadata is defined as data about data [60]. Although the scope of database metadata is very wide, one of its most useful functions is to obtain information on the database objects themselves such as tables and SQL grammar. Obtaining this information requires the use of two interfaces:

- ***DatabaseMetaData*** which provide information about the database as a whole. It contains dozens of methods for enquiring about a database. For instance, using these methods JDBC can retrieve very general information such as the database vendor. The code fragment below illustrates use of this interface.

```
DatabaseMetaData dbmd= connection.getMetaData();
System.out.println(dbmd.getDatabaseProductName());
```

Based on connected database JDBC will retrieve the database vendor such as *PostqreSQL*. More useful information can be extracted from database metadata using this interface and its methods such as number of tables in a database, their names and so forth.

- ***ResultSetMetaData*** which provides information about the types and properties of the column in ResultSet object. The information about a ResultSet object's column that is contained in a ***ResultSetMetaData*** object is accessed by invoking numerous ResultSetMetaData methods. Every piece of information can be extracted by using a specific method. The following piece of code demonstrates creating a ***ResultSetMetaData*** object.

```
ResultSet  result = statement. executeQuery("SELECT *  FROM student_table");
ResultSetMetaData  resultmd= result.getMetaData();
```

Much significant information such as primary Keys, foreign Keys, and database constraints can be retrieved using particular methods. For instance, using the previous ***ResultSetMetaData*** object the following methods can be invoked:

- `Resultmd.getColumnCount();` returns the number of columns in the *student_table.*
- `Resultmd.getColumnType(i);` returns the JDBC type for the value stored in column ith.
- `Resultmd.getColumnLabel(i);` returns the name of the ith column.

3.3 Conclusion

This chapter has surveyed the technology options that were available at the time of conducting this research. These technologies were discussed and reasons for particular choices were described. Because the proposed approach is heavily depends on database metadata, the relational database was chosen as best suited for implementing our approach. Since one gaol of our research is to come up with an abstract module that not limiting to any particular platform or any DBMS, JDBC was chosen because it allows us to develop applications at this higher level of abstraction independent of the details of a specific DBMS. Since the main scope of this research is exploring the capabilities of database metadata and investigating to what extent it can be fit in developing abstract dynamic Web forms, the issue of performance was not considering. As a result issues such as differences between driver types, database size, or rate of data exchange are ignored throughout this study.

Chapter 4

Literature Review

4.0 Introduction:

Since its first appearance in the early part of the 1990's, the Internet has improved vastly in terms of usage, equipment, data exchange and user interface design. Moving from static web pages to dynamic web applications and the ability of mobile devices to access the Internet have brought along with it many challenges to web page developers especially in terms of extracting data from databases, data exchange and user interface design.

In reality, nearly every web application consists of the following stages:

- End user initiates a connection with a server asking for a service.
- The server replies by sending back a web form for gathering data from the user.
- Collected data is sent back to the server for further processing and connection with data source, normally a database, is required for meeting the end user requirements.

- The outcome of this processing is formatted as web pages (HTML, XHTML, etc.) and sent back to the end user.

It is widely known that designing and building user interfaces based on data-model or domain model is an expensive task. The demands of users are numerous following their backgrounds and their equipment used to run or access the desired application. To address these concerns either in the Web-based or non-web based environment, many attempts have been presented in academic papers and articles, also many commercial products have been produced to address the above mentioned issues ([61], [62], [63], [64], [65], [66], [67] , [68] , [69], [70]).

This chapter surveys a number of academic research papers dealing with issues that relate to Web user interface, mainly concerned to databases. It consists of four sections. The first one discusses related work on user interface to databases. In particular, this section focus on usage of databases in Web development mainly in generating automatic user interfaces. The second section gives an overview of related work on to what extent that database metadata was used to construct web user interfaces. In the third section, a survey about the efforts that have been made in using XML and its surrounding technologies for generation web user interfaces is conducted. This chapter concludes with giving a brief glance at Web application frameworks. As an example, Ruby on Rails is discussed.

4.1 Related work on user interface to databases

It is important how much information you have, but the most important factor and the key to success in Web application development is how to organise, store, protect, access, query, update and browse this information. Integrating database with web application systems has become the cornerstone of web development community. The rest of this section introduces a brief overview of some academic papers related to usage of databases in Web development and its binding to user interface generation.

4.1.1 Accessing relational databases from the World Wide Web

Preceding the existing of ODBC and JDBC, many efforts had been made to bridge the relational database and Web applications. To illustrate the significant and ease of use of the ODBC and JDBC, one of the preceding techniques is discussed in this section. The authors in [71] described a general purpose solution which can be used to build web applications that can access numerous databases by using a page layout paradigm which encapsulates HTML [72] and SQL [73]. In order to enable web applications developers from using the full potential capabilities of HTML for building of query forms and reports, and SQL for querying and updating relational databases it suggested a technique to bridge the gap between these technologies. This technique based on flexible, general purpose variable substitution mechanism that provides cross-language variable substitution between HTML input and SQL query strings as well as between SQL result rows and HTML output. The above mechanism is used in design and implementation of the system called DB2 WWW Connection as demonstrated in Figure 17. This system enables swift and straightforward building of application that

access relational DBMS data from the Web. In addition, since the mentioned system uses native HTML and the SQL language, several visual tools could be used for building HTML forms and for production of the SQL query.

However, nowadays many powerful tools can be used to bridge web applications with RDBMS such as ODBC and JDBC. The later is chosen to implement our approach. (JDBC structure and reasons of why chosen it, see section 3.2)

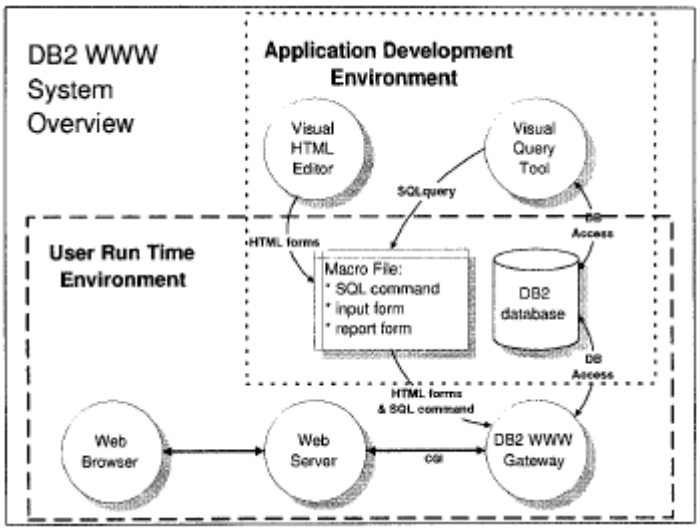


Figure 17 DB2 WWW System overview captured from [71].

4.1.2 Automatically generating World-Wide Web interfaces to relational databases

The authors in [74] argue that reformatting the information currently hosted in databases into HTML pages to deploy on the World-Wide Web is time and money consuming. To avoid this cost, they proposed a system that supports both direct querying of the database and browsing via dynamic Hyper Text links. These links are constructed from referential integrity constraints defined in the metadata.

This approach is close to what are proposing in this thesis in terms of using database metadata directly. However, a remarkable difference between the two approaches is noted. In [74] the proposed system does not use a general method of accessing the metadata; it is DBMS specific via DB2. In addition, the types of metadata and the ways that they were used are also limited since their primary focus seems to be *database browsing*. On the other hand, our approach is more generic since the metadata tables are automatically generated on the fly from any given database using the JDBC Metadata classes. So the need for manual update of metadata tables or the web interface code will be eliminated.

As a conclusion, this approach is still closer to those using an external representation rather than directly using the database metadata.

4.1.3 An Improved Method for Creating Dynamic Web Forms Using APL

The author in [75] argues that Dyalog APL (Array Programming Language) is a very powerful tool to create and deploy sophisticated web applications. In addition, it is argued that relying on a Web server model will minimise the issue of hardware and software incompatibilities on the client machines. To overcome the issue of updating and maintaining the look and feel of web pages without involving in recoding or recompiling chores, it suggested using an existing approach of creating a template of a HTML page. A server program updates this file before being rendered to the client browser. However, since this approach adds special non-standard tags to the HTML in the template document, the source document cannot conveniently be developed and

maintained using visual editing tools. To solve this matter, the author suggested using an HTML template that contains only standard HTML tags. This page can be pre-processed by APL on the server to dynamically modify HTML controls on the page as required.

The proposed model is required for using a commercial software product that allows users to configure a web server so that client requests for files having a specific extension.

In comparison to our approach, the proposed model generates only HTML web forms, whereas ours is capable to produce several types of web forms with very little effort. In addition, the model described in this work does mix the main component of the application which are content, logic and presentation. This makes it not possible to update or maintain the application without recoding and compiling the source program. Moreover, the mentioned model is meant to be using an external representation rather than directly using the database metadata.

4.2 Related work on user interface to metadata

Metadata is data about data. From this definition it is obvious that the actual data held in database tables or in other sort of data store is not the only source of information available to Web application developers or to computer programmers in general to create automated user interface. Based on this fact, a couple of efforts targeted this source of information in order to take advantages from the potential capabilities of metadata to build an automated user interfaces for Web applications. This section

discusses two academic papers related to user interface and metadata with two different aspects of metadata.

4.2.1 Metadata tables to enable dynamic data modelling and web interfaces design: the SEER example

In [1] the authors present an approach of dynamically generating web interfaces. Instead of recoding the source code of the interface, altering the metadata table can result in a new look and feel web user interface. They describe the development of a web-based interface engine whose content and structure are generated via interaction with a manually developed metadata table as shown in Table 5, which contains information about the main basic elements needed to represent a data model: table names, field names, field data type, and linkage among tables. They argue that this model should facilitate the presentation of data from several data sources via a common web interface.

The similarity between the above approach and what we have proposed in this thesis is that both models allow the web interface to be constructed automatically and dynamically from the metadata. However, the described approach needs more effort since the metadata is built by hand, whereas, we suggest using database metadata that can be retrieved dynamically on the fly using JDBC. Therefore, we conclude that the described approach is considered as using an external representation rather than directly using the database metadata.

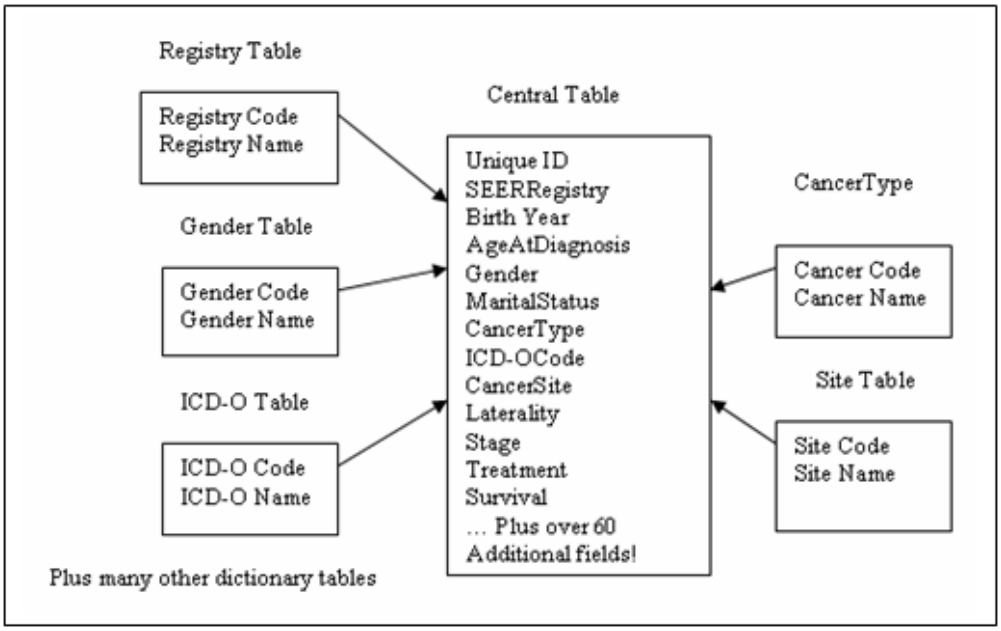


Figure 18 A subset of the SEER data model, adapted from [1]

Table Name	Field Name	Field Data Type	Linked Table	Linked Table Field
Central Table	Unique ID	Number		
Central Table	SEERRegistry	Foreign	SEERRegistry	Register Code
Central Table	Birth Year	Number		
Central Table	CancerType	Foreign	CancerType	CancerCode
Central Table	ICD-OCode	Foreign	ICD-OCode	ICD-OCode
Central Table	SiteCode	Foreign	SiteCode	SiteCode
↓	↓	↓	Continues for each field in the Central Table Table	
Registry Table	RegistryCode	Number		
Registry Table	RegistryName	String		
Registry Table	GenderCode	Number		
Registry Table	GenderName	String		
Registry Table	CancerCode	Number		
Registry Table	CancerName	String		
↓	↓	↓	Continues for each field in each lookup table	

Table 5 The metadata table that represents the SEER data model shown in Figure 18, adapted from [1]

4.2.2 Developing Web Entry Forms Based on Metadata

The authors in [76] proposed an approach that can be applied to develop automatic and dynamic HTML web forms based on metadata extracted from system catalogue tables. They suggest making use of Java and mainly Java DataBase Connectivity (JDBC), which provide a very general method for addressing databases and also includes metadata features. By overriding the advantages of metadata and JDBC, dynamic HTML forms were produced at runtime. In addition to the previous task, database metadata was used to produce user help messages. Moreover, the valuable information such as column data type and whether a specific column allows null values or not was extracted from database metadata and used in conjunction with JavaScript to validate entered data.

Although this approach and what we are proposing in this thesis target the same goal and use the same information resource that is database metadata, obvious dissimilarities between them have been cited. First, the proposed approach mainly intended to produce a single type of web form, that is HTML web forms, where as ours is more generic and abstract since it is capable of generating several sorts of web forms based on selecting an XSLT stylesheet (see section 5.5.3). Second, in our model the separation between the main components of web application (logic-business-presentation) is achieved, whereas the mentioned model has mixed up the three components which makes system redevelopment a very laborious task. Third, we propose to build up a stock of generic and extendable blocks of JavaScript functions that will be used to validate the entered data. Since these blocks of JavaScript are stored in an external file, they can be updated with no requirements to recompile the system.

4.2.3 GUI Generation from Annotated Source Code

The authors in [77] aim to reduce the effort needed for developing graphical user interface across different platforms with numerous requirements and restrictions. It is argued that by defining the user interface on an abstract level in the form of task model, the targeted goal will be achieved. The proposed approach is based on direct derivation of UI model from application source code enriched by abstract commands of user interaction to control the generation of the UI.

Generating a concrete GUI involves two stages. First, mapping the abstract GUI elements to concrete GUI supported by a given platform. Numerous possible mappings [77] between the data transfer abstract GUI elements and concrete platform-dependent GUI elements are illustrated in Table 6. Second, performing computations of sub optimal layout of the GUI elements including splitting the elements into groups that may not be visible simultaneously. Among several existing techniques to solve this issue, a simple *ad hoc* technique was chosen that puts elements mostly vertically and eventually splits the form into several tabs to fit the window if necessary.

Abstract item	Possible mappings
Text input	text field, text area, etc.
number input	text field, slider, scroll bar, spin, combination of elements, etc.
single item selection	radio buttons, combo box, list, text field, etc.
multiple item selection	check boxes, list, etc.
monitoring	gauge, label, status bar, etc.
responding to alerts	modal dialogs, alerts, etc.

Table 6 Possible mappings of abstract items, adapted from[77]

In comparison to our approach, the described effort aims to generate a cross platform automatic graphical user interface by using the source code of a given program. However, we propose to make use of database metadata to achieve a similar goal. In addition, to generate dynamic web-based UI, different database columns are mapped to UI controls based on characteristics of each column and on a set of supplied rules (see section), unlike in [77].

4.2.4 Automatic Generation of Web User Interfaces in PHP Using Database Metadata

The authors in [5] depict a way to generate dynamic user interface elements based on database metadata. PHP and the abstract library ADOdb were used to achieve this goal. PHP is able to extract database metadata from either information schema or system catalogues. Despite the fact that PHP is a very rich in functions that allow it to speak natively with the majority of DBMSs, every single supported DBMS has its own functions to perform the same tasks. To solve this issue and for sake of ease of program development and maintenance, it proposed to use a database abstract interface that can be used in development of across DBMSs Web application system with either no or a very small effort.

The described approach is similar to ours in many faces in which it makes use of the extracted information from database metadata to build up dynamic Web user interface. In addition, the same concept of mapping every single attribute in database's tables to a specific user interface control has been used. Our approach, in contrast, makes use of the Java and XML technologies instead of PHP. This direction gives it a very high level of abstraction. The outcome of our approach is not bound to a specific type of Web form. It can be formed in many different sorts of Web forms such as HTML, XHTML, XForms, etc. As long as XSLT is capable of transforming a XML document to any desired Web form, our approach will be the outstanding solution. Moving on an additional dissimilarity between the two approaches, in our case database metadata was used to build up and use generic and dynamic blocks of JavaScript to validate the data entry. Moreover, in case of generating XForms Web entry forms database metadata was used to construct the model layer that is responsible for data validation. However, the

authors in [5] say nothing about how database metadata can be used for the above purpose.

4.3 Related work on user interface and XML

XML is widely accepted in the Internet society as an alternative to old web technologies such as HTML. It comes with many new features that make it a promising technology to most existing web technology problems. Nowadays, XML and its surrounding technologies (XPath, XSLT, XForms, etc.) are a hot research topic in many different aspects. This section discusses some academic papers related to user interfaces to XML.

4.3.1 Using XML/XSL to Build Adaptable Database Interfaces for Web Site Content Management

The authors in [78] concluded, that maintaining a web site can become a sophisticated task by the time of the number of offered services and amount of data grow steadily. Integrating relational database with web sites eases the task of content management. Web-based forms are used by content mangers to insert and update information in the database and HTML pages are either statically or dynamically produced from database content. Despite the fact that databases offer support for content management, integration with the web user interface is an expensive chore. A considerable re-development effort is often required. In this paper the authors describe how XML/XSL can be deployed in creating adaptable database interfaces using

WebCUS (Web Content Update System), a tool they implemented to achieve their goal. WebCUS uses XML/XSL technologies and their MyXML template engine to generate Web forms from the underlying database schema description and has support for access control management.

In contrast to our approach, the database schema is represented in an external XML file separate from the database. However, the common aspect between the two approaches is that in the described model the XSLT stylesheets are used by MyXML template engine to transform the MyXML documents created by WebCUS into web-based user interface, whereas we used XSLT stylesheets to transform an XML document that was generated on the fly from the database into several formats of web forms. In WebCUS, the database schema is represented in an external XML file separate from the database. This XML information uses a special syntax to describe the tables, attributes and relations between tables in the database. The Extended Entity Relationship (EER) [79] methodology was used in their projects for modelling the database information. This model has to be manually converted into the WebcUS XML database schema description. Figure 19 illustrates a sample WEbCUS XML schema description for a table: book. For every single column of the table, XML attributes specify the name, textual representation and the data type of the column.

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCTYPE database SYSTEM ". ./ resources/myeer.dtd">
   <database name="shop">
     <table name="book">
       <column name="id"
               alias="Id"
               type="key" />
       <column name="title_name"
               alias="Book Title"
               type="text" />
       <column name="author_name"
               alias="Author"
               type="text" />
        <column name="publisher_name"
               alias="Publisher"
               type="text" />
       <column name="isbn_nr"
               alias="ISBN"
               type="text" />
    </table>
  </database>
```

Figure 19 EER description in XML, adapted from [78]

4.3.2 Generating Form-Based User Interface for XML Vocabularies

Y.S.Kuo, et al [2] propose and implement a tool called Forms-XML. It is intended to generate dynamic HTML forms to allow end users to interact with and update XML data complaint with the given schema. According to [80], XML vocabularies are classified in two main categories, data-centric and document-centric vocabularies. However, Forms-XML is targeted at the former category. The described tool is fed by four inputs: an XML schema, an XML document complaint with the XML schema, a user interface customisation (UIC) file, and a CSS style sheet. Hierarchal HTML forms as generated based on schema and UIC, which allow the user to edit the input XML

document. The layout of generated HTML is left for a given stylesheet that can be written by an interface designer. Furthermore, Forms-XML comes with custom control that can be invoked for customisation purposes.

This work is similar to what we are proposing in that it is targeted at producing dynamic web user interface. In contrast, the described model is specifically designed to deal with an external XML document as a data model rather than using an internal data model. In addition, the outcome of this model is entirely bound to HTML forms, whereas, our approach is more generic and flexible since it is capable of producing different types of web form based on a selected XSLT stylesheet.

4.3.3 A framework for automatic generation of web-based data entry applications based on XML

According to Turau in[81] , the main advantage of this framework is a clear separation between the business logic and the presentation. He argued that this model allows work on each phase to carry on in parallel along relatively independent but cooperating tracks. The framework is built according to the MVC design pattern and introduces a method for the conceptional and the navigational design based on a textual specification in the form of an XML application. This shapes the input to a code generation environment allowing for actual automated prototyping. The environment generates fully functional skeletons for the web pages. Collectively with the framework classes they can be utilized for testing and for needs review. They can also outline the starting point for the work of the presentation design. The described framework contributes a

method for a high level specification of data entry tasks in shape of an XML-document that can be validated against an external DTD. In comparison to our framework, the described model is specifically designed to deal with an external XML documents as a data model rather than using an internal data model.

4.3.4 GARP: A Tool for Creating Dynamic Web Reports Using XSL and XML Technologies

The authors in [82] describe an approach aimed at generating automatically Web reports from database scheme. They suggested creation of a set of JSP files holding all required information by the reports. The proposed software tool is built up based on two main technologies: XML and Java. JDBC is used to extract database metadata that is finally formed as an XML document. The outcomes of the proposed tool are JPS files and they are formatted using XSL templates.

The authors highlight the benefits for application developers in automated use of database metadata and platform independence that can be achieved by the use of JDBC. The tool they describe has three separate distinct phases as shown in Figure 20 and listed as follows:

- Extraction of database metadata in a basic XML format.
- Conversion of XML using XSL.
- Production of JSP files from XML.

Although the first and third stages use Java they are completely separate and the authors indicate the use of manual intervention to move files and use of separate

servers for different parts of the execution. The initial XML produced appears to be quite generic and very specific formatting instructions for the web reports are input via the XSL transformation at the second phase.

The authors argue that using this technique will minimise the effort in building and maintaining the described tool. Moreover, they argued that by embedding database scheme in an XML document it will be possible for the tool to convert the Web reports into several formats such as Word, Excel, PDF, etc.

Despite the fact that their approach has similarities to ours in that it uses queries on database metadata to produce XML output targeted to the Web, their approach is limited in that it is not interactive and only focuses on “Web Reports” . In addition, their principal motivation for the use of XML seems to be that it offers a simple route to a variety of output formats such as Word, Excel and PDF particularly via the use of XSL-FO.

Since their emphasis is on reports they appear to focus on table names, attribute names and data types but not on issues such as referential integrity (relevant to input or query forms but not to reports), nulls (important for data entry) or the cross over between metadata and data (significant for offering possible data values for query or input).

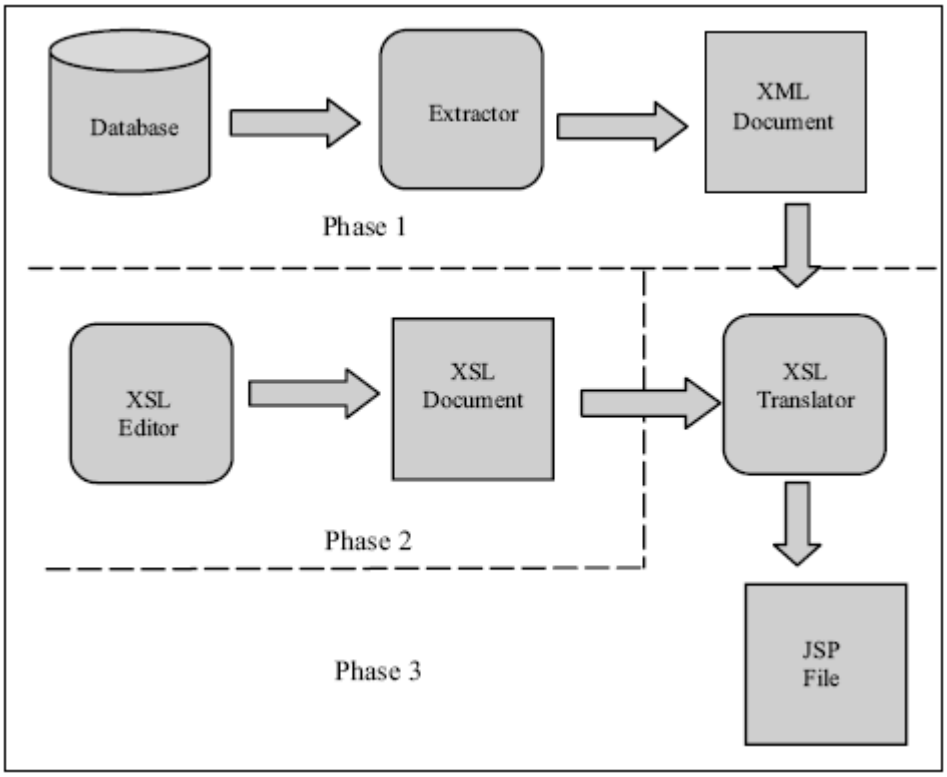


Figure 20 Reports generator architecture [GARP], adapted from [82]

4.3.5 Generic XForms-Based User Interface Generation for XML Schema

Koen De Wolf et al [3] described a way of generating automatic and dynamic XForms user interfaces from an XML schema description and XSLT stylesheet. Using XForms features it is explained how XForms Actions could be used to eliminate basic scripting tasks performed by common classic client-side scripting language such as VBScript, JScript and JavaScript. XML schema is non-deterministic and can be used to validate several instance documents. A schema is considered to be non-deterministic if the parser is unable to clearly determine the structure to validate with the schema. On the other hand, an XForms must be deterministic. The whole structure of the XForms

instance must be known and at runtime, no elements or attributes can be added or removed apart from homogenous collections. This fact makes automatic generation of XForms based on XML schema very complex task to using a single XSLT stylesheet in a generic manner.

The similarity between our approach and the described model is that both approaches target generation of automatic and dynamic Web XForms. However, the described model relies on an external representation. It makes use of an XML schema description and XSLT stylesheet to produce Web XForms user interface in generic manner. In addition, the XSLT stylesheet used in this model is capable of generating only XForms **input** elements and misses generating the other powerful controls such as **select** and **select1**. On the other side, we proposed the use of an internal data model to achieve the same goal with different techniques. (See section 5.5.3.4).

4.4 Web application frameworks

In this section, a brief introduction about the principle of web application frameworks is given. For its robust and widely recommended in web development society, Ruby on Rails is introduced and contrasted with our approach.

4.4.1 Web frameworks definition and classifications

A web application framework can be defined as a software framework that is designed to facilitate the development of dynamic web applications and web services. The main goal behind a Web application framework is to minimise the overhead linked with ordinary activities used in web development. For instance, most frameworks offer

libraries to ease database access, session management, templating frameworks and often support code reuse.

The majority of frameworks are built based on the Model View Controller (MVC) [83] architectural pattern. The remarkable feature of this approach is the separation of the data model, business rules and presentation. In addition, MVC frameworks can be classified into two categories. The first is push-based frameworks. They use actions that perform the required processing, and then push the data to the view layer to deliver the results. Ruby on Rails, Struts and Django are good examples of this architecture. The second is the pull-based frameworks architecture. They start with the view layer, which can then pull results from several controllers as required. In this model, several controllers can be involved with a particular view. Examples of frameworks using their model are Struts2 [84], JBoss [85] and Tapestry [86].

4.4.2 Ruby on Rails

Ruby on Rails, often referred to as “Rails” or “RoR” was developed by David Heinemeier Hansson and released to public in July 2004 [87]. RoR is built based on the MVC architecture and consists of several sub frameworks. The main components of RoR are shown in Figure 21 and they are:

- Active Record, which establishes the connection between the database and domain, objects. It bridges Action Controller and the database by transforming CRUD (Create, Read, Update, Delete) functions into SQL statements, sending requests to database, receiving and passing results to the Action Controller.

- Action Controller, which handles actions from forms and other user input. In addition, it establishes the connection with Active Record in order to receive and pass database data to Web Action Web Services, Action View and Action Mailer.
- Action View is responsible for facilitating templates that generate XML, HTML and other output.
- Action Mailer that provides powerful e-mailing services.
- Action Web Services which offers API creation functionality.

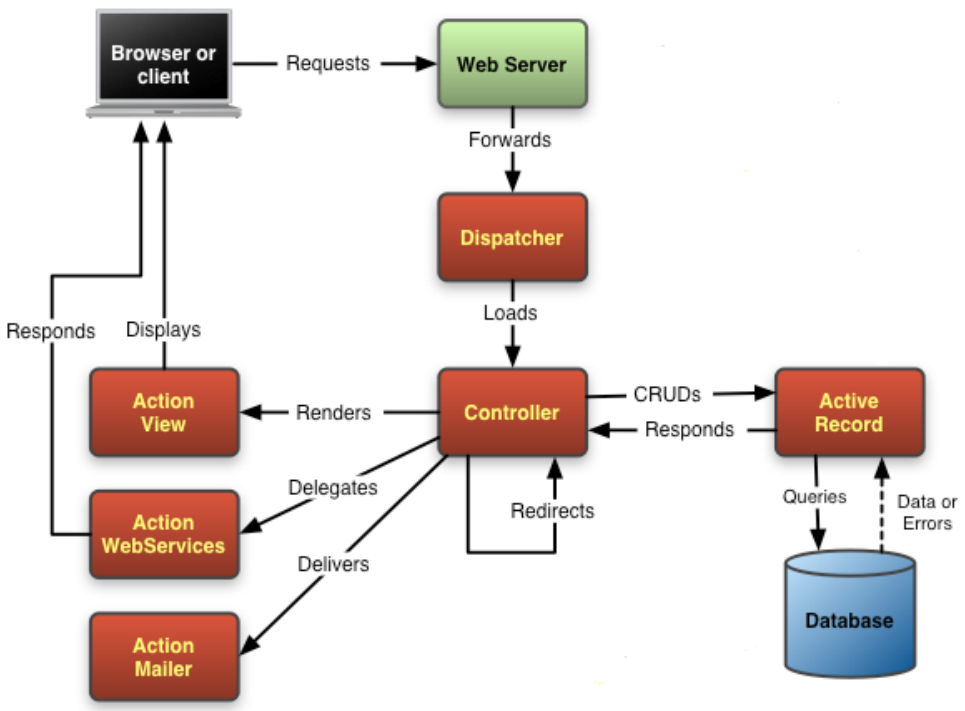


Figure 21 RoR architecture, captured from [88]

As any other technology or tool, RoR comes with some shortcomings. The main remarkable issue bounded to RoR is using a naming convention. In the RoR approach

primary keys must be of integer data type and must be identified by the use of a column whose name ends with _id. From the point of view of many developers, this seems to be an excessively restrictive model since the fact that a column is a primary key can be obtained from database metadata regardless of its name. In this thesis, we proposed an approach that allows the use of existing, or more natural, naming of database columns such as when an external real world entity like a national number provides a primary key. In addition, our approach makes it possible to work with complex primary keys and foreign keys found in existing databases since they can be retrieved from database metadata in a straightforward manner instead of relying on a name convention. Moreover, the notion of using XML technologies in conjunction with database metadata as adopted in our approach is a more independent route and more generic since it can be applied without any restriction regarding database conventions or use of particular design methods.

4.5 Conclusion

This chapter has discussed research related to our proposed approach. The majority of systems were considered targeted at creating application domain specific interfaces rather than generic interfaces. A few others did consider the notion of developing generic interfaces but they come with several shortcomings that we tried to solve them in this study. Compared all efforts discussed in this chapter to our approach, our approach is not limited by domains, because we are not developing or maintaining an

external data model or representation. Moreover, our approach makes clear separation between the content, logic and presentation; this allows the application to be developed by team independently on each other. In addition, combining database metadata with XML technology allows our approach from generating different types of Web entry forms without the need to recompile the source code or making any changes to access DBMS.

Chapter 5

Framework Implementation

5.0 Introduction

This chapter introduces our suggested approach and its implementation in detail. We aim to design a framework to support the notion of using relational database metadata to generate automatic and dynamic Web entry forms. The framework is meant to be a high abstract level, i.e. it is not bounded to any DBMSs or platform, and it is flexible and reusable with minimal effort. It begins with an overview of the features of the proposed approach. The structure of the implemented approach is demonstrated. Extracting and converting database metadata into an XML document is explained with support of code fragments. Transforming the XML document into different types of Web entry form is

illustrated in detail. Usage of database metadata in conjunction with JavaScript to validate data entry is discussed. A short conclusion is included at the end of this chapter.

5.1 Prototype Overview

The main principle of our approach and its implementation was not building a comprehensive Web framework targeting a particular field or domain. Rather it aims to study and investigate to what extent we can use the potential information held in database metadata in conjunction with XML technology and its surrounding technologies to develop an abstract representation that can be used for producing different Web entry forms that not bound to a particular platform or to a specific device. Compared with the majority of approaches and modules that we considered in the chapter 4 (Literature Review), our approach is not bound to a specific domain. Considering external data models was totally avoided in this research. Instead, the internal information in relational database metadata was considered. It is richer than the external data model. Furthermore, it is implemented the dynamic and relationship type of information. This approach aims to get the most abstract, highest level, most reusable and general solution for extracting and using database metadata. In our implemented approach, we suppose if a traditional model of data flow in a database environment is as illustrated in Figure 22, then we would like to correspond the structure with database metadata flow as demonstrated in Figure 23. Metadata can be retrieved from a logical model by several techniques at certain levels. Figure 24 shows the metadata levels in a conceptual schema. At the highest level, metadata can be extracted using JDBC metadata classes at runtime. The second level of metadata is resided in an information schema. At the bottom of the logical model metadata can be accessed by querying

system catalogs. Querying information schema and system catalogs could be restricted by the version's of DBMS. However, for reasons listed in section 3.2.1, JDBC was chosen to implement our approach. The details of how JDBC gets results at driver level by either query information schema or system catalogs; or how JDBC encapsulates getting metadata is beyond the scope of this research. In our implemented framework, metadata will be retrieved using JDBC metadata classes at runtime, converted into an XML document and then sent back to a Web browser to transform it into a desired Web entry form based on applying a set of rules.

As far as the generation of different types of Web entry forms using a single prototype system is concerned, separation between the main components of a Web application (content, logic and presentation) is considered. Adopting XML technology in our approach allows us to achieve this goal in a straightforward way. Combining the dynamic features of database metadata with the ability of XML documents to be presented in several formats was the key to this approach.

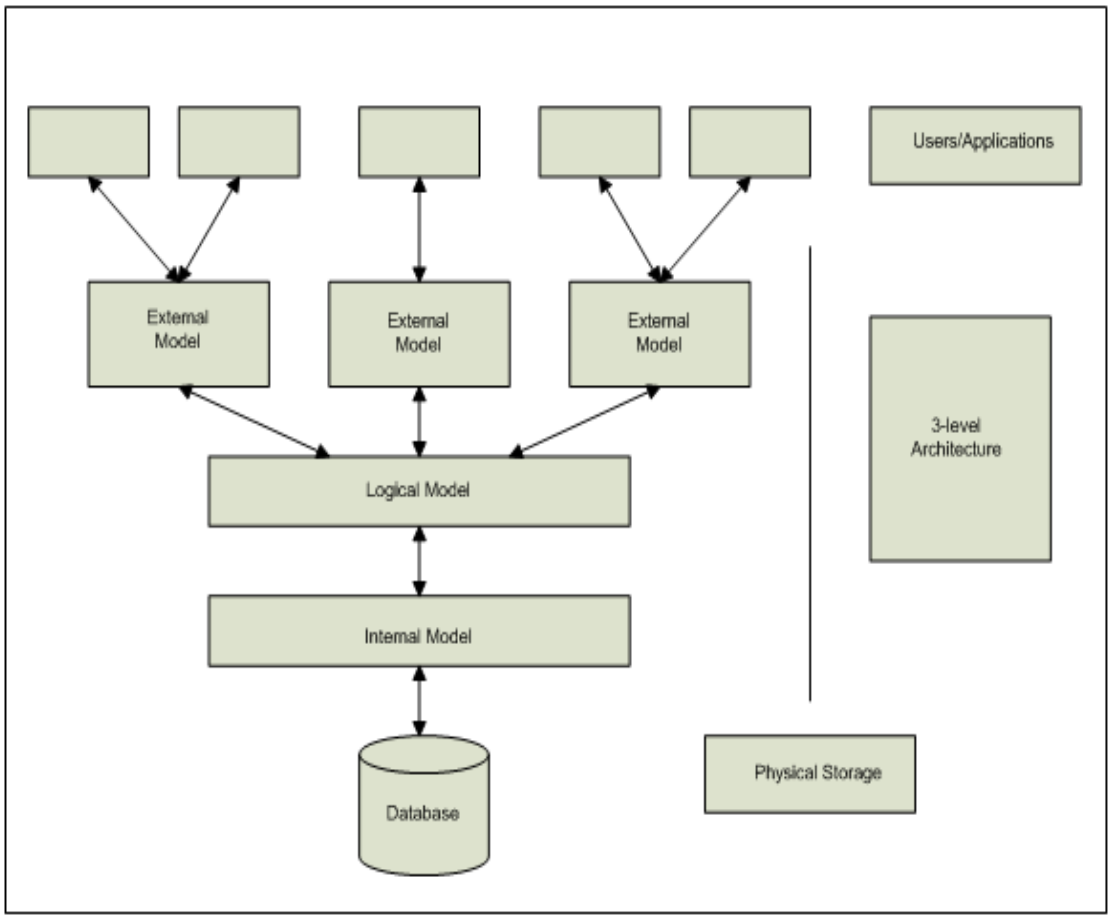


Figure 22 Three level database architecture, adapted from [89]

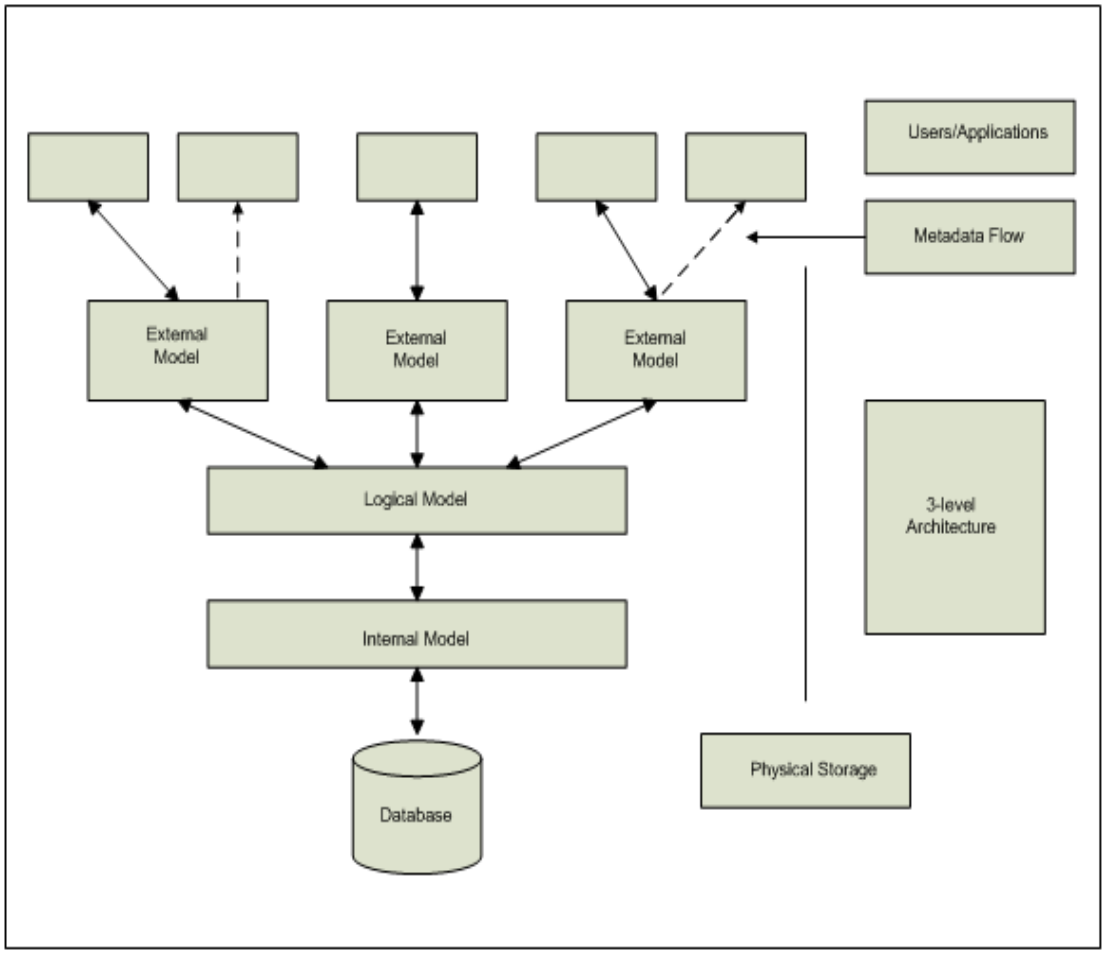


Figure 23 Three level database architecture with Metadata flow

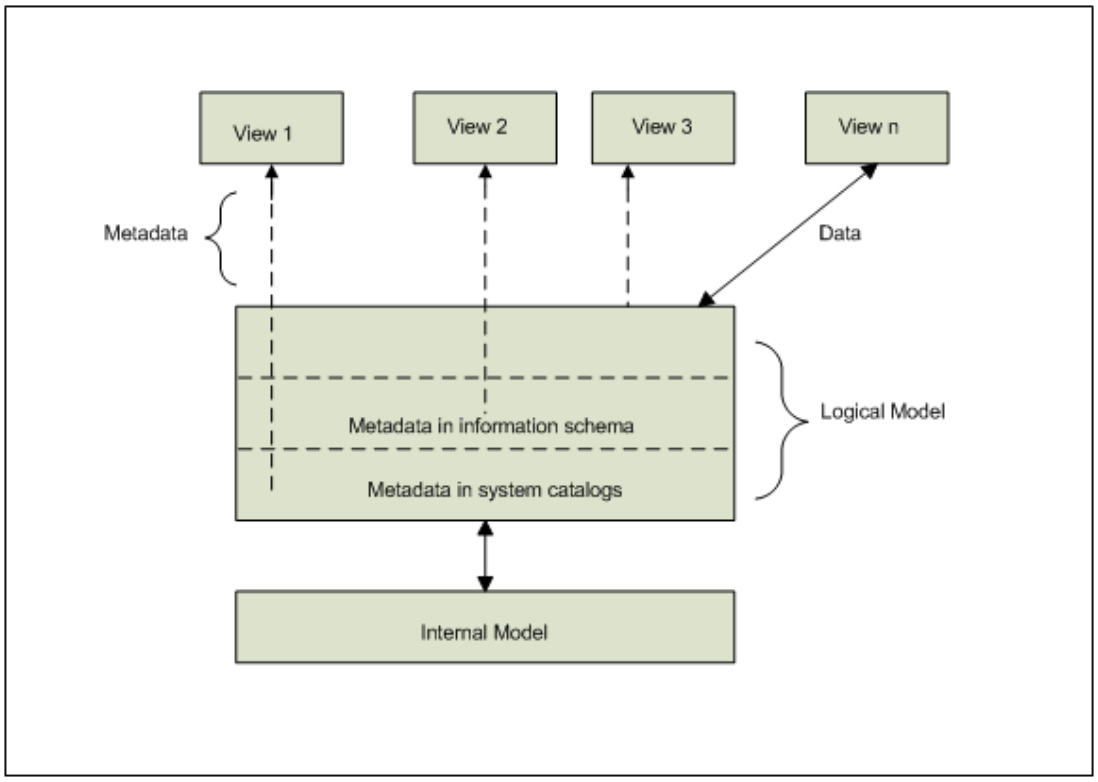


Figure 24 Metadata levels in logical model

5.2 The Prototype Characteristics

The following are features of the prototype:

- The code is reusable. No need to rewrite specific code for different databases or different platform.
- The code required for validation of entry data can be eliminated by adopting XForms technology.
- Web entry forms are generated and validated in an automatic and dynamic manner.

- Dynamic appearance of Web forms based on accessed data. Changes made to databases are reflected the next time the data is accessed.
- Legacy databases can be accessed, enquired and different Web entry forms can be generated in a straightforward way.
- Separation between content, logic and representation is complete. The code does not need to be recompiled in order to get different Web entry forms.

5.3 Three tier solution

The prototype system was implemented using a three tier solution as shown in Figure 25

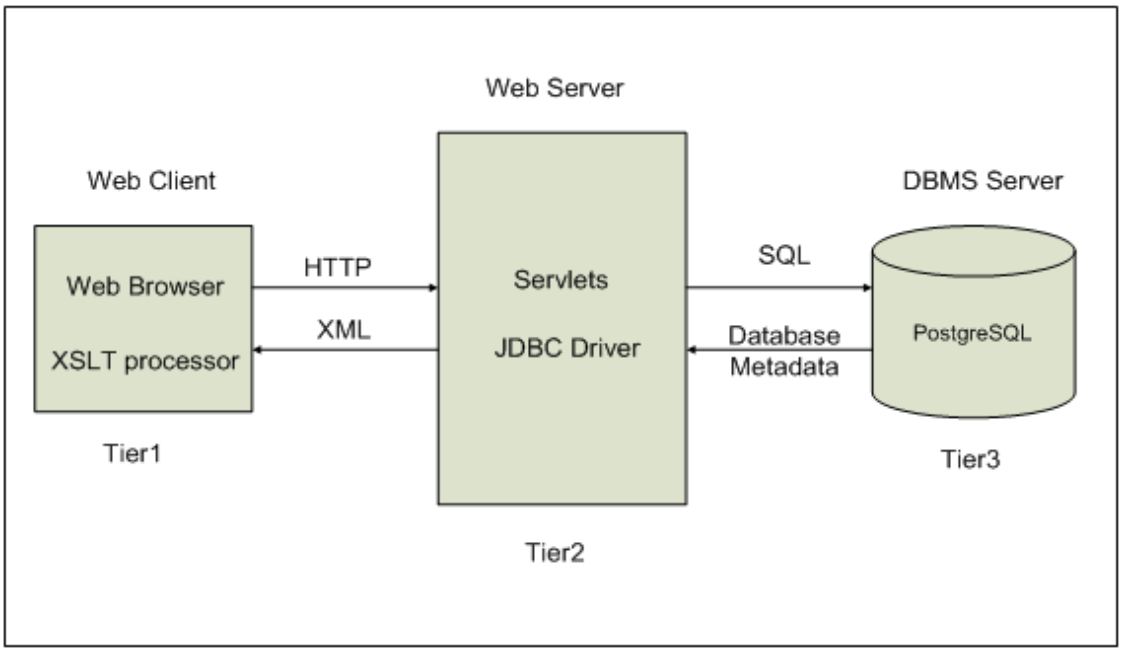


Figure 25 Three-tier solution

A 3-tier solution allows users to communicate to a middle-tier which is JavaServlet via HTTP. The JavaServlet communicates with the data tier using JDBC. This solution has

many advantages over its predecessors (1-tier and 2-tier) solutions. Among these advantages:

- **Database Connection Pools:** The middle tier can be used to pool database connections by reusing the pre-started processes that are already connected to the database. This eliminates the need to establish a new connection for each request.
- **Network traffic is minimised:** The middle tier logic executes multiple SQL queries and performs additional database processing to return only the final result to the client. This result in reducing the volume of data exchanged between the client and the server.
- **Tiers independency:** Since the three tiers are independent of each other, upgrading one tier does not require upgrading the other ones. For instance, upgrading the client tier does not require re-writing the business logic. Separation the data tier and business logic through standard database access API, makes it more flexible to upgrade or replace the underlying database.

However, there is another solution which called n-tier solution. This architecture is one which has *n* tiers, usually including client tier, database tier, and more than one tier in between.

5.4 Framework Architecture

As shown in Figure 26, the architecture of the proposed framework consists of several stages. These stages are listed as follow:

- A connection to a database management system is established using JDBC. Metadata about this database and its tables are extracted via JDBC

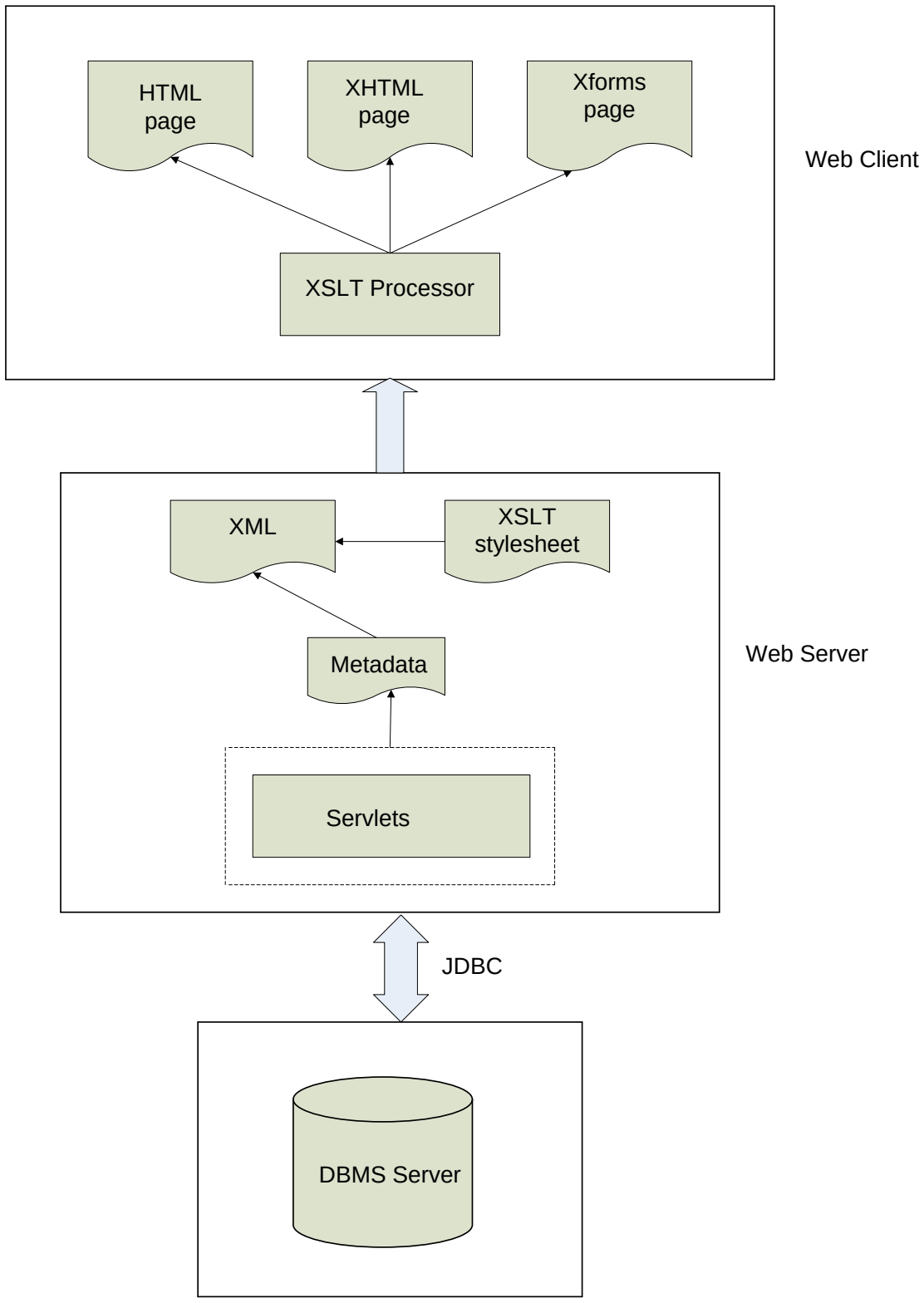


Figure 26 Architecture of the frame work

- Retrieved database metadata is converted to an XML document for further processing.
- The XML document is transformed into an HTML, XHTML, and XForms document or into any desired format of Web entry forms. This process is performed by using XSLT stylesheet in conjunction with a set of rules (5.5.3.1). However, at present the rules we have developed are very generic. In addition, it would be possible to develop and apply domain specific rules without altering the logic of the implemented approach. The implemented methodology makes a clear separation between content, logic and representation. A small alternation to a XSLT stylesheet results in a new look Web user interface without the need to re-write or re-compile the source code on Web server.
- The generated Web form is returned back to the client as a Web form which enables the user to fill in. Entered data needs to be validated against database metadata. This task can be performed via invoking proper JavaScript functions in case of generating HTML or XHTML Web forms, or can be achieved by using the built-in functions in case of XForms Web forms.

However, as shown in Figure 26 the transformation task is performed on the client-side leaving the choice of how the desired document will be rendered to Web browser. In addition, as shown in Figure 27 it would be possible in the proposed approach to perform the transformation task on the server-side and sending back the desired page into Web browser. However this choice has several disadvantages and not implemented in this study. These disadvantages including: it requires extra processing on the server and there are number of web hosts which do not have the XSLT extension.

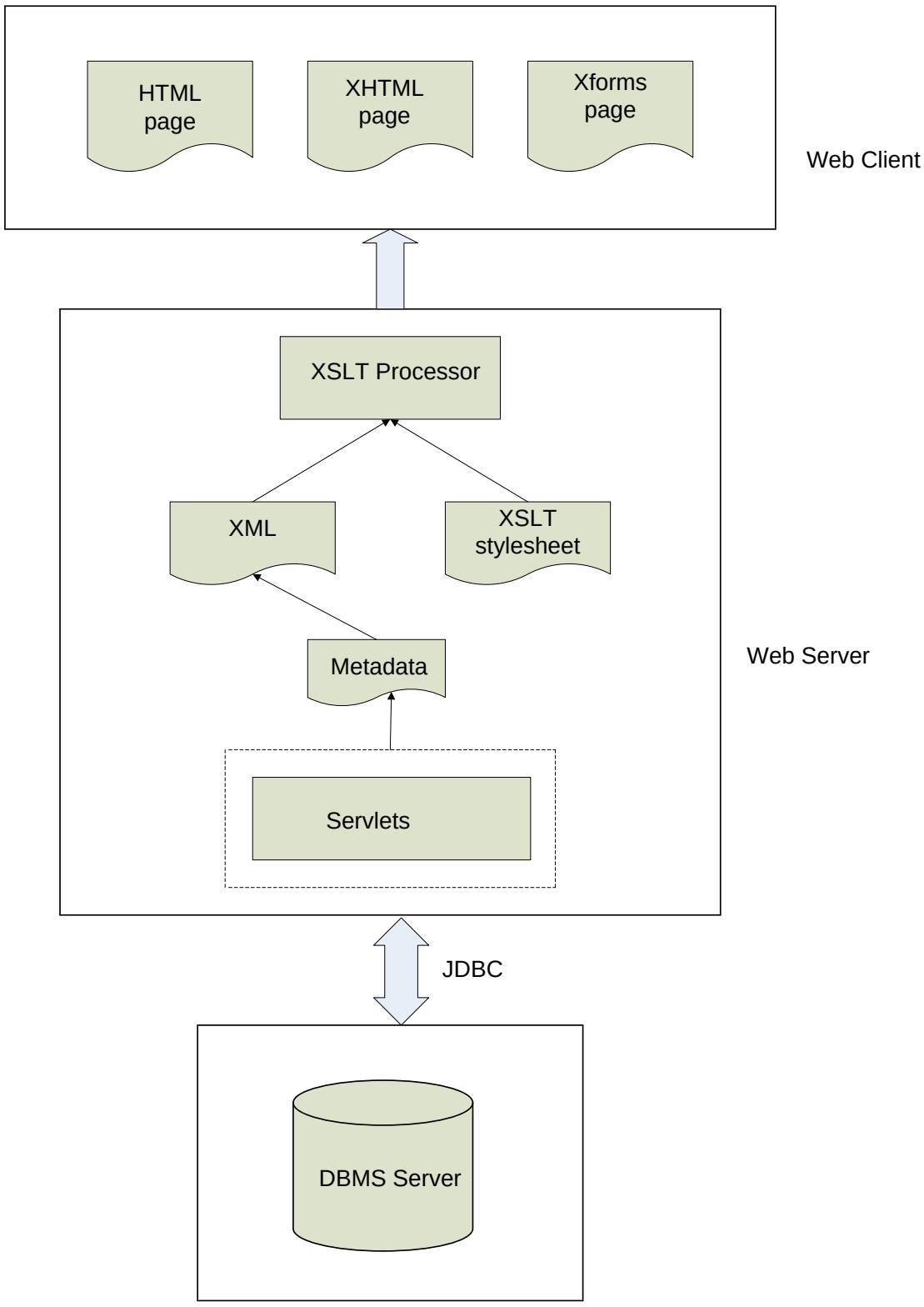


Figure 27 Framework architecture performing transformation task on server-side

5.5 The mechanism of the prototype

To illustrate this methodology and to investigate if there are any difficulties in implementing it, the following section introduces an example of the implementation of this approach.

For the sake of simplicity, this example is kept as minimal as possible. The concept is one, i.e. mapping each table's column to a specific Web entry form control based on a set of rules. The commonly used features of database metadata for this purpose are: the variety of data types, null or non null able fields, primary/foreign keys, and the semantics of metadata. The outcome of this approach is rich and determined by applying a specific XSLT stylesheet. The steps follow:

5.5.1 Connecting to database and retrieving metadata

We start with creating a database table which contains personal information as illustrated in Figure 28.

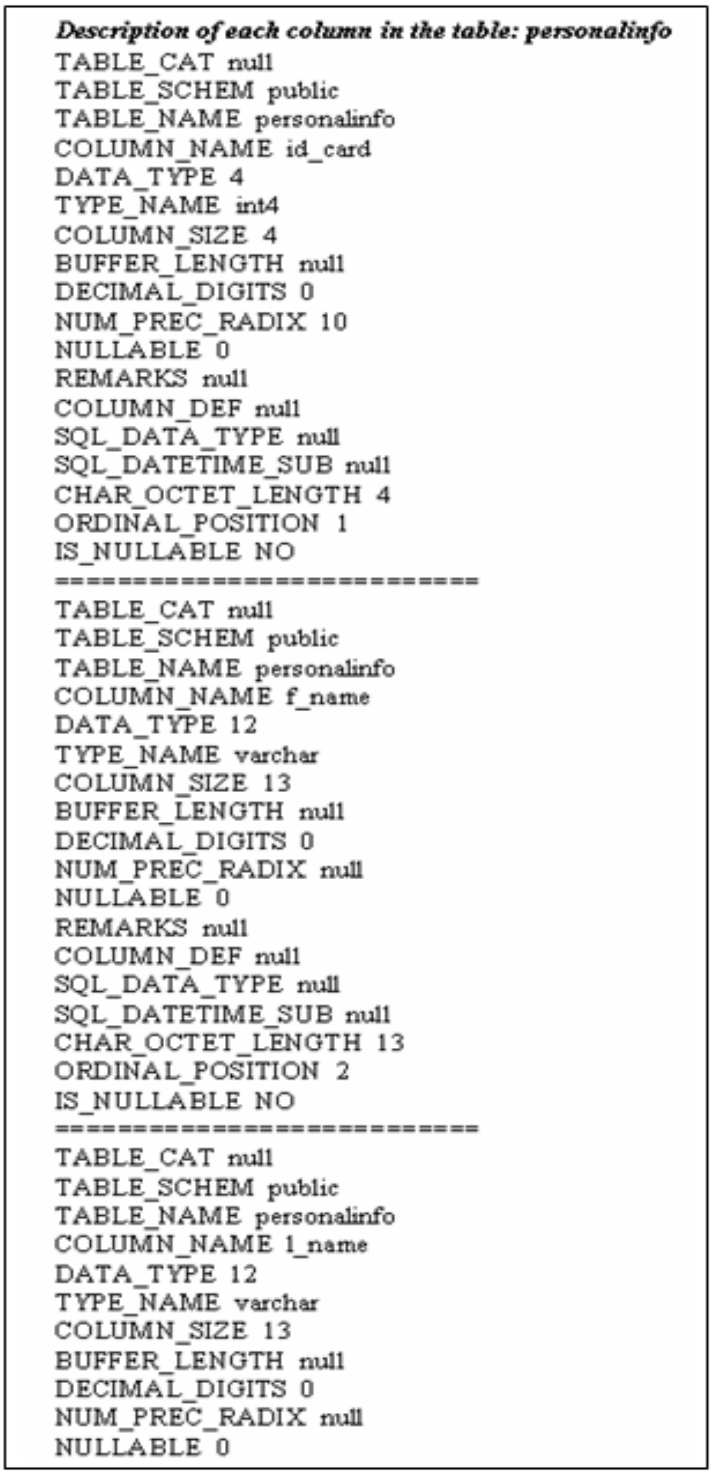
```
Create table personalinfo
(
id_card serial primary key,
first_name varchar(12) not null,
last_name varchar(12) not null,
date_of_birth date not null,
sex Boolean,
address varchar(35)
);
```

Figure 28 Database table's structure

JDBC is used to retrieve the database metadata about this table. In practice not all extracted information will be used for producing dynamic Web pages. Only the required pieces of information will be used, such as column name, data type, column size and so forth. A piece of the extracted information is shown in Figure 29.

5.5.2 Converting metadata into XML document

In order to make use of the retrieved database metadata, this information needs to be converted into an XML document. The document may contain a large amount of information. Besides the information shown in Figure 29, other pieces of information such as whether the column is primary key or not, whether the column is foreign key or not are needed to be added to the XML document in order to get sufficient information for transforming the XML document into other form.



Description of each column in the table: personalinfo

TABLE_CAT null
TABLE_SCHEM public
TABLE_NAME personalinfo
COLUMN_NAME id_card
DATA_TYPE 4
TYPE_NAME int4
COLUMN_SIZE 4
BUFFER_LENGTH null
DECIMAL_DIGITS 0
NUM_PREC_RADIX 10
NULLABLE 0
REMARKS null
COLUMN_DEF null
SQL_DATA_TYPE null
SQL_DATETIME_SUB null
CHAR_OCTET_LENGTH 4
ORDINAL_POSITION 1
IS_NULLABLE NO
==========
TABLE_CAT null
TABLE_SCHEM public
TABLE_NAME personalinfo
COLUMN_NAME f_name
DATA_TYPE 12
TYPE_NAME varchar
COLUMN_SIZE 13
BUFFER_LENGTH null
DECIMAL_DIGITS 0
NUM_PREC_RADIX null
NULLABLE 0
REMARKS null
COLUMN_DEF null
SQL_DATA_TYPE null
SQL_DATETIME_SUB null
CHAR_OCTET_LENGTH 13
ORDINAL_POSITION 2
IS_NULLABLE NO
==========
TABLE_CAT null
TABLE_SCHEM public
TABLE_NAME personalinfo
COLUMN_NAME l_name
DATA_TYPE 12
TYPE_NAME varchar
COLUMN_SIZE 13
BUFFER_LENGTH null
DECIMAL_DIGITS 0
NUM_PREC_RADIX null
NULLABLE 0

Figure 29 A portion of raw metadata retrieved from a database table shown in Figure 28

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Form_spec>
<Form_Element>
<ColumnName>id_card</ColumnName>
<Type>integer</Type>
<Nullable>False</Nullable>
<PKey>True</PKey>
<FKey>False</FKey>
<Serial>True</Serial>
</Form_Element>
<Form_Element>
<ColumnName>first_name</ColumnName>
<Type>String</Type>
<length>12</length>
<Nullable>False</Nullable>
<PKey>False</PKey>
<FKey>False</FKey>
</Form_Element>
<Form_Element>
<ColumnName>last_name</ColumnName>
<Type>String</Type>
<length>12</length>
<Nullable>False</Nullable>
<PKey>False</PKey>
<FKey>False</FKey>
</Form_Element>
<Form_Element>
<ColumnName>date_of_birth</ColumnName>
<Type>Date</Type>
<Nullable>False</Nullable>
<PKey>False</PKey>
<FKey>False</FKey>
</Form_Element>
<Form_Element>
<ColumnName>sex</ColumnName>
<Type>Boolean</Type>
<Nullable>True</Nullable>
<PKey>False</PKey>
<FKey>False</FKey>
</Form_Element>
<Form_Element>
<ColumnName>address</ColumnName>
<Type>String</Type>
<length>35</length>
<Nullable>True</Nullable>
<PKey>False</PKey>
<FKey>False</FKey>
</Form_Element>
</Form_spec>
```

Figure 30 XML document built up from a database metadata

As shown in Figure 30, each column in a database table is represented in a single XML node which is **<*Form_Element*>.** This node has seven child nodes containing information on this column including:

1. Column name.
2. Column type. If the type is string data type then it requires an extra child node which holds the length of this column.
3. Whether the column is primary key or not.
4. Whether the column is foreign key or not.
5. Whether the column is serial or not.
6. Whether the column accepts null values or not.

At this point the database metadata is extracted and converted into XML document. It is described in an abstract way. It is generated automatically and dynamically, every time a new database table(s) is queried, a new and different XML document will be generated holding the seven pieces of information about every column. If an extra piece of information is needed it can be added in a straightforward manner. For instance, the following code fragment is used to create the <*Pkey*> child in every <*Form_Element*> node as shown in Figure 30.

```
Element node5 = doc.createElement("PKey");
 Node5.appendChild(doc.createTextNode(IsPK));
 row.appendChild(node5);
```

Where the *(IsPK)* parameter in the second statement returns a boolean value of a function that checks whether a passed column name is a primary key or not.

Any number of children can be added to <***Form_Element***> node using the above code fragment.

However, the structure of the XML is flexible and can be built in many different ways, keeping the same information. Instead of presenting every single column in a database table in a single XML document containing several child nodes; it could be presented as a single XML node with several attributes holding the same information as in the previous structure Figure 30 . As shown in Figure 31, each column in the database table is presented as one XML node with several attributes. In addition, Listing 12 illustrates how this XML is built; it is possible to add as many attributes as needed.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<data>
<id_card colname="id_card" type="integer" null="false"  Pk="true" Serial="true" Fk="false" />
<first_name colname="first_name"  type="string" length="12" null="false"  Pk="false" Fk="false" />
<last_name colname="last_name"  type="string" length="12" null="false"  Pk="false" Fk="false" />
<date_of_birth colname="date_of_birth"  type="date" null="false" Pk="false" Fk="false" />
<sex colname="sex" type="Boolean" null="true" Pk="false" Fk="false" />
<address colname="address" type="string"  length="35" null="true" Pk="false" Fk="false" />
</data>
```

Figure 31 XML document built up from database metadata.

```
Element row = doc.createElement(ss);
        row.setAttribute("colname",ss);
        row.setAttribute("type",CType);
        if (str) {
         row.setAttribute("length",stringsize);
        }
        row.setAttribute("null",Nullornot);
        if (PK=="true")
         {
        row.setAttribute("Pk",PK);
        row.setAttribute("Serial",isserial);
        }
        row.setAttribute("Fk",FK);
        results.appendChild(row);
```

Listing 12 illustrates how XML document is built

Both of the above XML structures can be transformed into different Web entry forms in a straightforward manner. Since any piece of information in the XML document is accessed and retrieved using as XPath expression, the choice of using either of the above structures is left to the XSLT stylesheet developer. However, if for any reason a new desired XML's structure is needed, it would be possible to build it without altering the main concept of this approach.

5.5.3 Transforming XML document into Web entry forms

The two previous steps that were introduced in sections (5.5.1 and 5.5.2) were performed on the DBMS server and Web server. This step involved transformation of the XML document into Web entry forms and is performed on the Web client. Transformation execution is achieved by applying an XSLT stylesheet to the XML document in conjunction with a set of rules. The Web browser parsers the XSLT stylesheet and generates the desired Web entry form. Transformation is based on the idea of mapping every column in the database table which is presented as an XML node

to a specific Web form control element. As stated earlier, the proposed approach is intended to be generic and abstract to the highest possible point. Separating the content, logic and presentation allow us to generate different types of Web entry forms without the need to recompile the source code. Every type of Web forms such as HTML, XHTML or XForms can be achieved by writing and applying a particular XSLT stylesheet.

5.5.3.1 Developing the Rules or Heuristics

For developing automatic and dynamic Web entry forms by taking advantage of database metadata, we propose to develop a set of rules or heuristics which could be applied to database metadata in order to produce abstract and generic Web entry forms. Developing a set of rules is not arbitrary. It is based on potential information that resides in database metadata and data itself. Since any Web interface is built up of several user interface controls such as those currently used in HTML (radio button, drop down menu, input box, etc.), it will be more efficient if we could generate a Web dynamic user interface on the fly without the need to hard code the presentation of the user interface following changes to database tables or the data type of each column in the database. By developing a set of rules, we argue that it will be possible to automatically map each column in a database to a specific user interface control without taking into account the way the browser will render the user interface controls. Second, these rules can be used to maintain database integrity. This can be achieved, for example, instead of letting a non null Boolean column generate a database error; we

automatically generate an application that provides TRUE/FALSE must be entered options and behaviour.

Examples of developed rules

The set of rules shown below is built based on the column's data type. However, more rules can be developed based on the semantics of a column's name (section 5.5.3.2).

- Rule1: if a column is a serial, then it should not be mapped to any form element (in insert mode). Its value will be incremented *automatically* and stored in this column.
- Rule 2: if a column is a primary key, then its value must not be a null value.
- Rule 3: If a column is a string data type and its length (l=x) then a JavaScript function is called to check that the length of input is acceptable.
- Rule 4: if a column is character or string data type and less than 30 characters (for example), then it should be mapped to a text box. If longer or equal to 30 characters it could be mapped to textarea Web form control. The number of the characters could be more or less and it is left to the user interface designer to determine which length of text box suited best the user interface.
- Rule 5: if a column is a foreign key, then get the possible values from the referenced table and offer them as choices initiated as a pull down menu if the number of values is under a specific limit. This rule shows the usage of database metadata to get the fact that it is a foreign key and makes use of the data itself for possible values.

- Rule 6: if a column is a Boolean then refer it as *choice* and it is implemented as a group of radio buttons in case of HTML or XHTML Web forms. However, this column could be implemented in other ways. For instance, in XForms which has more flexibility to render Web form elements, it could be returned as radio buttons, drop down menu or any other possible type based on the client machine.

Furthermore, since there is a clear methodology to implement such basic rules, this set of rules can be easily extended when new demands are raised. The above set of rules is very generic and could be applied for any situation. Specific rules for particular cases can be produced and applied.

5.5.3.2 Limitation of database metadata

In the proposed approach each database column is mapped to a specific Web form element based on the database metadata and a set of rules. As shown in Figure 29, JDBC is able to retrieve a significant amount of information about each column in database. However, the most reliable piece of information that can give indication of how to perform the mapping task is a column data type. Nevertheless, this piece of information on its own is insufficient in some cases to generate a perfect Web entry form. For instance, the only clue tells that a particular column is intended to be a password is its name. Another example, there is no way to tell that a particular string data type is meant to be an email address so we can invoke a suitable JavaScript function to validate this string against a standard email address format. Knowing this fact leads us to look for another resource of information that can help in solving this

issue. Working on the semantic database metadata especially on semantics of columns' name was the available resource; regardless there is no guarantee that this resource will give accurate indications.

To tackle this issue, another rule is developed and added to the rules' list. For instance, the following assumption could be used to map a particular database column into password text box in HTML or into secret in XForms instead to map it into an input text box.

Rule: if a column is a String data type and its 6<=length <= 12 and its name holds the phrase (password or secret or sub set of this phrase), then this column should be mapped into password text box. To make this assumption more general, it is suggested that a list of synonyms for password words could be populated to an XML configuration file. This could contain non-English words such as French or Spanish words for internationalisation purpose.

Another example, as shown in Figure 28, the *sex* column is a Boolean and as a general rule, any Boolean column is mapped into a group of radio buttons with a label True/False. However, to give a meaningful label for this user interface element, the name of this column can be interpreted and then the label is derived i.e. if a column's name is *sex* or *gender* then the label should be presented as Male/Female assuming that Male is associated with true and Female with false.

As it is assumed that any label is the actual value of the column's name in database. For enhancing the appearance of Web forms, column's name can be processed to produce an elegant label. For example, a column's name with more than one word separated with underscore could be represented as a label with underscores replaced with space

and the first character of all words in upper case. For instance, if a column's name is *first_name* it could be presented as *First Name* as shown in Figure 32 and Figure 33.

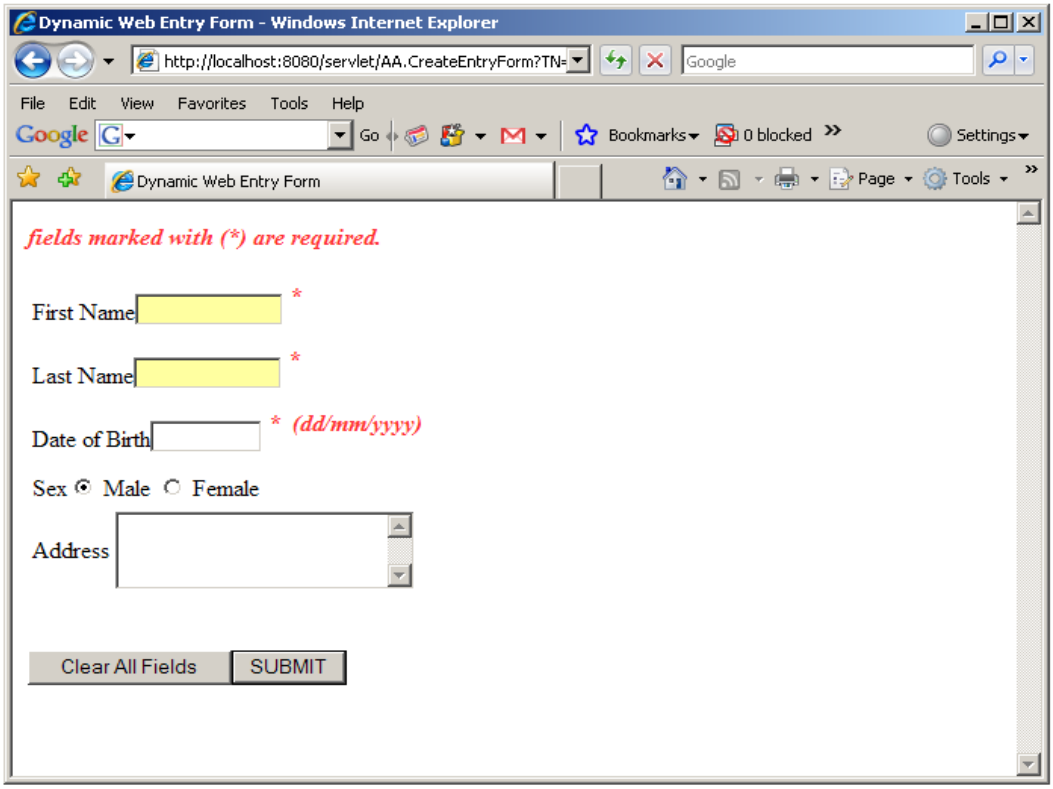


Figure 32 XHTML Web entry form generated from database metadata on the fly, invoked in IE Web browser.

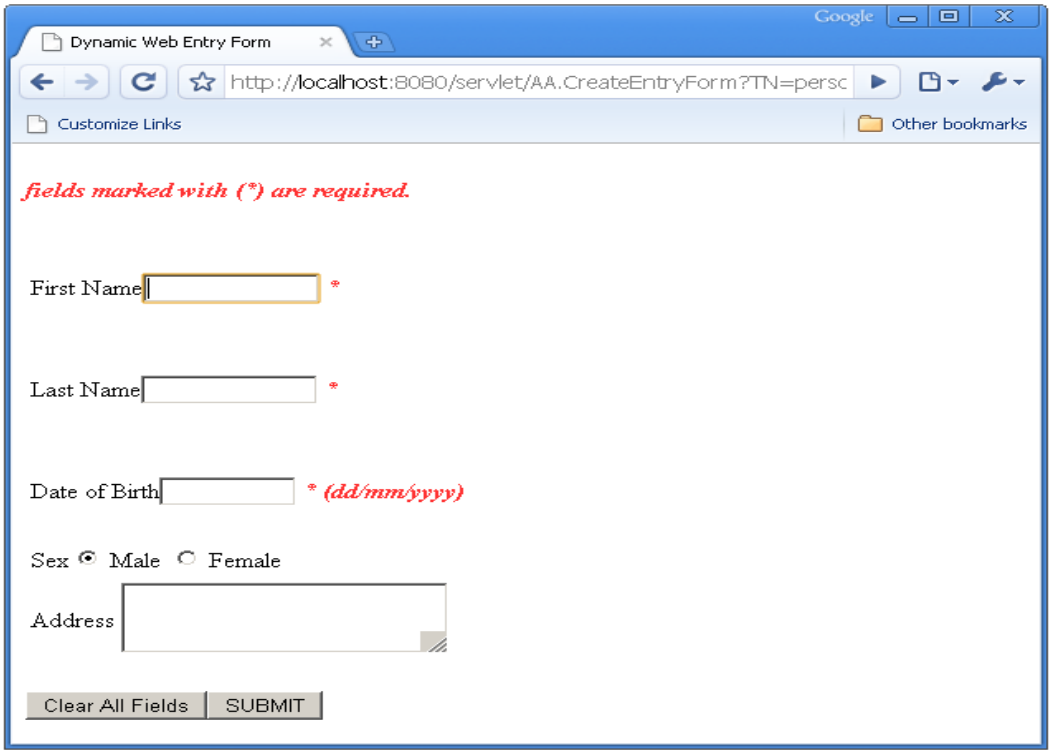


Figure 33 XHTML Web entry form generated from database metadata on the fly, invoked in Google Chrome Web browser.

5.5.3.3 Transforming the XML document

During this research we tested two different types of Web forms by writing two different style sheets.

5.5.3.3.1 Transforming the XML document into XHTML Web entry form

- The first stylesheet transforms the XML document into an XHTML document as shown in Figure 32. Since the XML document reflects a relational database table, its structure is kept very simple, i.e. as flat document displayed in Figure 30.

  The XSLT processor starts parsing the XML document by matching the root document. Then it loops through the document processing every XML node and

its children that represent a single database column. The following code demonstrates this process.

<xsl:for-each select="Form_spec/Form_Element">

Every XML document is transformed into a XHTML form control by applying a set of rules that explained in section 5.5.3.1. For instance, the Listing 13 generates the input box for any ***string*** data type.

```
<xsl:if test="Type ='String'and lenght&lt;='30'">
    <tr> <td>
     <xsl:value-of select="ColumnName" />
      <xsl:variable name="bodysize">  <xsl:value-of select="lenght" />
       </xsl:variable>
      <xsl:variable name="IdName">  <xsl:value-of select="ColumnName" />
      </xsl:variable>
  <input type="text" name="{ColumnName}" size="{$bodysize}"
  idtype="{Type}" idnull="{Nullable}" />
    </td>
      <xsl:if test="Nullable='False'">
        <td>
          <h5> <xsl:text> * </xsl:text> </h5>
        </td>
      </xsl:if>
   </tr>
</xsl:if>
```

Listing 13 illustrates creating input box using XSLT statements

The code fragment in Listing 13 generates the code fragment shown in Listing 14

```
<input type="test" name="columnname" size="columnsize" idtype="String"
idnull="False/True" />
```

Listing 14 XHTML fragment code generated by XSLT code Listing 13

As we can see in Listing 14, database metadata such as its *(name, type, whether accepts null values or not)* for the selected column is embedded in the XHTML control form. These pieces of information will be used for validation purposes. For instance, the attribute *idnull* is used to verify whether this control can accept null values or not.

5.5.3.3.2 Data validation

Since an XHTML Web form entry is generated automatically and dynamically on the fly, a generic method for data validation is needed to fit this approach.

Two possible solutions are considered to solve this issue. The first one is to use database metadata for each column located in the XML document to dynamically generate JavaScript at the same time as the form is generated. The second is to use the database metadata to include the required information in the name of element to validate the form element as shown in Listing 14. Then a generic JavaScript function as shown in Figure 34 is invoked to loop through the form elements and parse the names to validate the data in the element generating a suitable message if needed, an example is shown in Figure 35.

```
Function MainFunction(){
determine the number of elements (n);
for i= 1 to n
get the name of element (i);
get the value of element(i);
If element(i).getAttribute(idnull)=false
call subfunction1 to specify whether the value of this
attribute = null or not
return false() if the attribute value=null;
else
return true();
end if Subfunction2 // to check date data type
call the proper subfunction to verify the data type of
the element(i)
return true() or return false()
end for
 } // end MainFunction
subfunction1 // to check null values
subfunction2 // to check date data type
subfunction3 // to check integer data type
……….
……….
subfunctionn // to check any data type
```

Figure 34 Pseudocode of generic JavaScript function

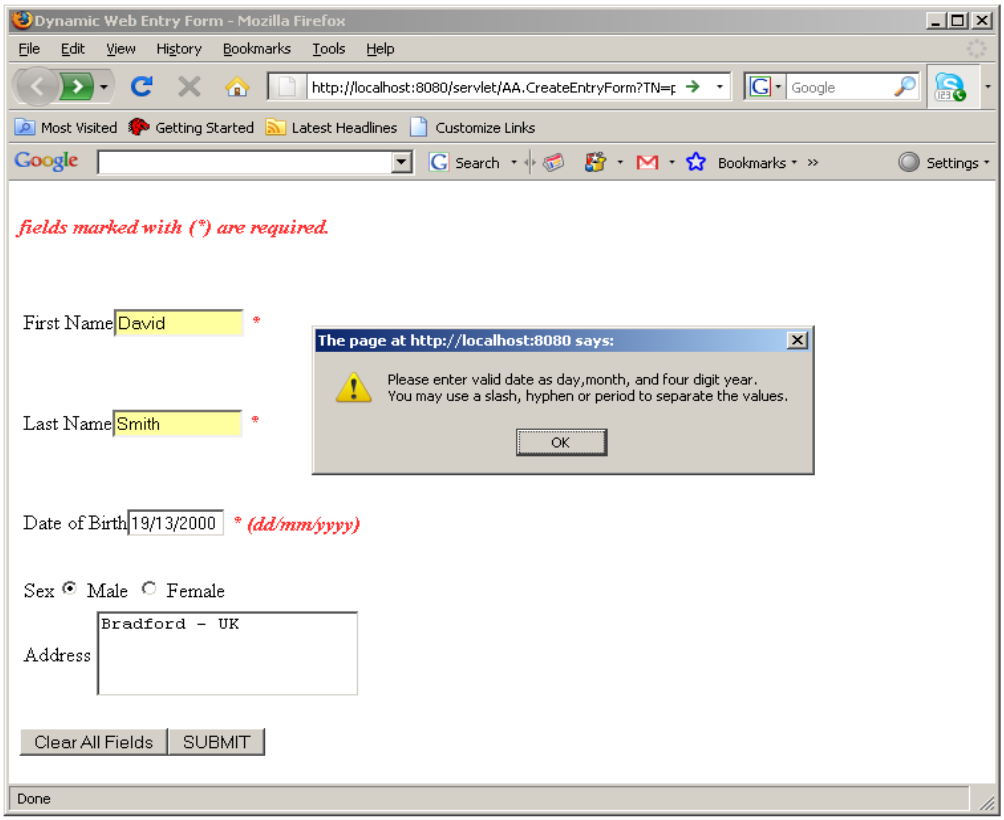


Figure 35 Data validation warning message

Although throughout this work only the second method was implemented and tested, its functionality was good enough to rely on without the need to consider the other method. However, the implemented method can be contrasted to the first one since it comes with a large amount of code on all pages, although that same block of code can be referenced by many pages efficiency. The implemented method, however, has some advantages compared to the first one. The first is that it is easier to test the validity of what it produce. The second moves a lot of load to the client-side whereas the validation code production is loaded on the server-side in the first approach. Once entered data is validated on client-side it is sent back to Web server to revalidate it and store it the database.

5.5.3.4 Transforming the XML document into XForms Web entry form

To transform the generated XML document into an XForm Web entry form, another XSLT stylesheet is needed. As stated earlier the transformation process is based on mapping every single database column into a specific Web form control according to a set of rules. Although the majority of the above rules (section 5.5.3.1) can be used in both cases (XHTML,XForms), a small number of them are not applicable with XForms where JavaScript is not involved in the process.

For demonstration purposes, let us reuse the database table shown in Figure 28 by omitting the ***serial*** property from the first column (***id_card)*** and using the second format of the XML document shown in Figure 31.

As demonstrated in (chapter 2), XForms technology is based on separation of content from presentation. It consists of two main parts; ***model*** which includes instance data elements, binding element and submission elements, and a ***user interface*** which consists of the user interface controls. XSLT transforms the XML document by building each part of the above as shown in Listing 15.

```
<xsl:stylesheet> // document root
<xsl:template match="/">
<html>
<xforms:model>
<xforms:submission /> // submit data to URL
<xforms:instance>
Copy MXL instance data by applying a particular template
</xforms:instance>
// building binding elements
Loop through the XML document
Select an XML node
Call a template to bind a binding element
End looping
</xforms:model>
<body>
Loop through the XML document
Apply rules and call the proper template to build up a user interface control
End looping
</body>
</html>
</xsl:template>
All invoked template placed here
</xsl:stylesheet> // document root
```

Listing 15 XSLT stylesheet Pseudocode that transforms XML document into XForms.

For explanation purposes, the instance data can be built by applying the template shown in Listing 16.

```
<xsl:template match="data" >
<xsl:copy>
  <xsl:copy-of select="*"/>
</xsl:copy>
</xsl:template>
```

Listing 16 illustrates building an instance data

Where as Listing 17 demonstrates creating of *binding elements.*

```
<xsl:template  name="binding">
 <xforms:bind>
 <xsl:for-each select="data/*">
 <xsl:attribute name="nodeset"><xsl:value-of select="local-
name()"/></xsl:attribute>
 <xsl:attribute name="required"><xsl:value-of
select="@null"/>()</xsl:attribute>
 <xsl:attribute name="type">xs:<xsl:value-of
select="@type"/></xsl:attribute>
 </xsl:for-each>
 </xforms:bind>
 </xsl:template>
```

Listing 17 illustrates building binding elements.

Building user interface elements is performed by accessing each XML node and getting its information associated with a set of rules. For instance, Listing 18 illustrates how textarea element can be built.

```
<xsl:if test="@type = 'string' and @length&gt;'30'">
          <xsl:call-template name="textarea" />
</xsl:if>

The invoked template is

<xsl:template  name="textarea">
   <p>
    <xforms:textarea ref="{local-name()}>
      <xforms:label><xsl:value-of select="@colname"/></xforms:label>
       </xforms:textarea>
      </p>
 </xsl:template>
```

Listing 18 illustrates building a user interface element

Applying such an XSLT stylesheet to the XML document generates an XForms Web form entry as shown in Figure 36 and segment of page source code shown in Listing 19.

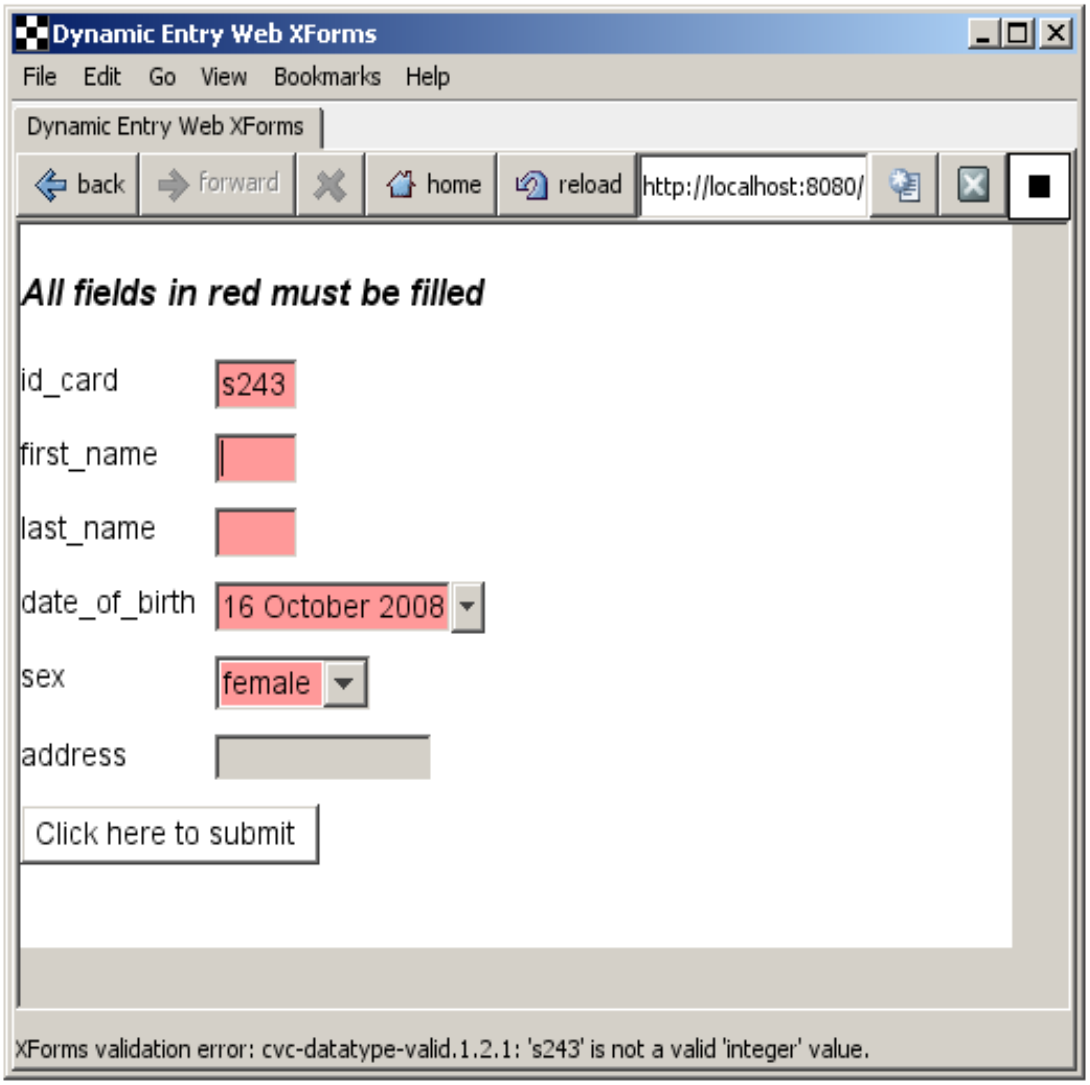


Figure 36 an XForms Web entry form generated from an XML document

Binding elements

```
<xforms:bind nodeset="id_card" required="true()" type="xs:integer"/>
<xforms:bind nodeset="first_name" required="true()" type="xs:string"/>
<xforms:bind nodeset="last_name" required="true()" type="xs:string"/>
<xforms:bind nodeset="date_of_birth" required="true()" type="xs:date"/>
<xforms:bind nodeset="sex" required="true()" type="xs:Boolean"/>
<xforms:bind nodeset="address" required="false()" type="xs:string"/>
```

User interface controls

```
<p>
<xforms:input ref="id_card">
<xforms:label>id_card</xforms:label>
</xforms:input>
</p>
<p>
<xforms:input ref="first_name" >
<xforms:label>first_name</xforms:label>
</xforms:input>
</p>
<p>
<xforms:input ref="last_name" >
<xforms:label>last_name</xforms:label>
</xforms:input>
</p>
.....

.....
```

Listing 19 segment of page source code corresponding to page shown in Figure 36

5.5.3.4.1 Data validation

One of the most interesting features in XForms is that the mechanism of data validation. JavaScript is no longer needed when working with XForms Web entry forms. As stated in chapter (2), XForms is very rich in data types and come with many built-in functions that can be used to facilitate the validation process.

Since every single user interface control is bound with associated *binding element* as shown in Listing 19, the XForms processor should apply every piece of constraint found

in the associated binding element to that particular user interface control. For instance, the first user interface control

<xforms:input ref="id_card" style="width:70pt">

<xforms:label>id_card</xforms:label>

</xforms:input>

is associated with the binding element

<xforms:bind nodeset="id_card" required="true()" type="xs:integer"/>

This means that this control can not accept any null values and its value must be an *integer* data type. So, attempting to enter any other sort of data type will generate a warning message such as that one seen on the bottom of Figure 36.

Although the concept of XForms seems to be quite clear and simple, the implementation of XForms comes with a number of overhead issues including:

- Plug-in software that enables Web browser's processor to process XForms is not always straightforward working as it is proposed to do which disallow Web browser's to render XForms pages as they should be. As mentioned in chapter 2, X-Smiles is the only Web browser that native support of XForms; however this browser is very basic and the majority of social, educational, news, entertainment and commercial companies do not consider this Web browser when upgrading their pages. For example, Figure 37 and Figure 38 illustrate the main page of the University of Bradford Web site rendered in X-Smiles and Mozilla Firefox Web browsers. This fact makes this Web browser unpopular and useable only by a few numbers of users.

- Using xml namespace requires a lot of care. For instance, the instance data element must be written in the following form

  <xforms:instance xmlns="">

  Missing the attribute ***xmlns=””*** will make the Web browser's processor behave in unpredictable way.

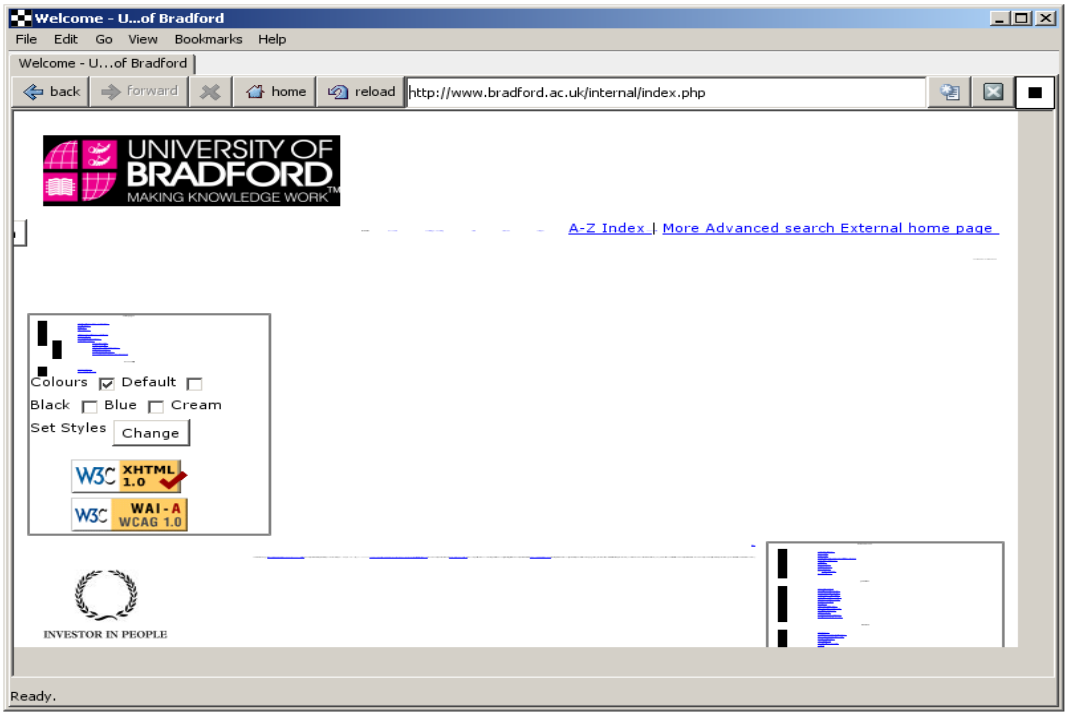


Figure 37 University of Bradford main page rendered in X-Smiles Web browsers

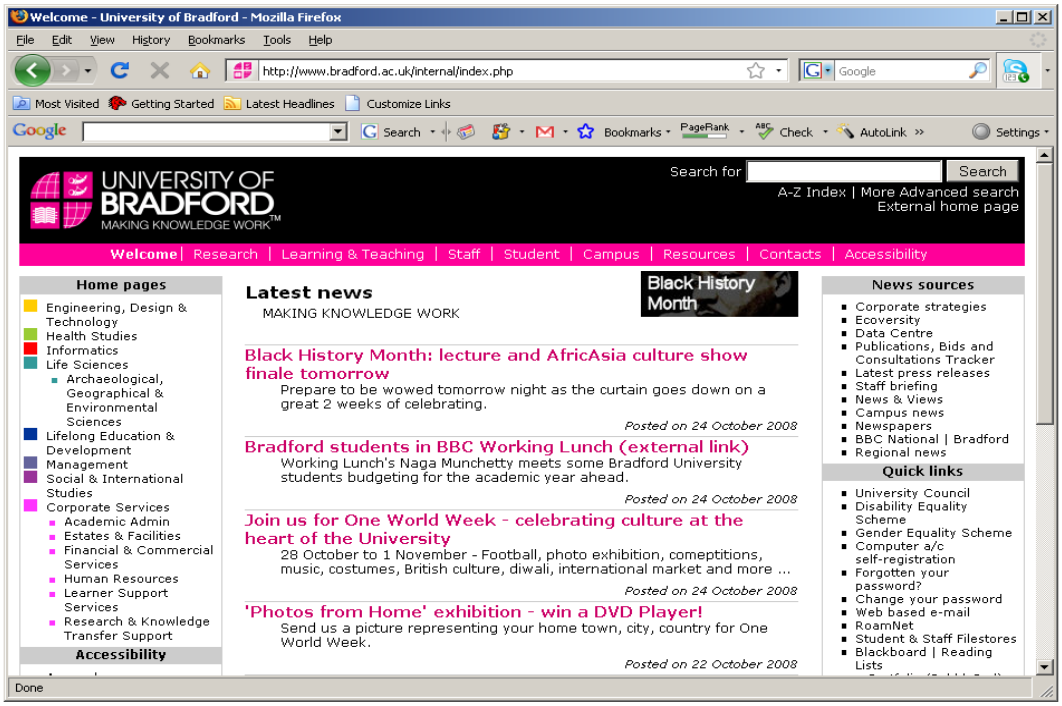


Figure 38 University of Bradford main page rendered in Mozilla Firefox Web browsers

5.6 Extended Example Scenario

The previous section (5.5) demonstrates a very generic example of how the prototype system was implemented. The implementation involved of using a single database table that contains personal information. However, for further testing of the proposed approach this section shows how this approach can be implemented in multi database table's environment. We illustrate the operation of the prototype with an extended example based on a simulation of an airplane booking scenario.

Since the mechanism and the logic is same, and to avoid redundancy, this section will be summarised and the main differences and results are mentioned. In addition, since this research is concentrated on generating of web entry forms this example shows

the generating of sequence of web entry forms that allow the user to interact with the system in a virtual environment.

As in section (5.5.1) we start with creating a portion of database tables that allow us to simulate the booking system and then extracting the database metadata from this database. The suggest database consists of six tables containing information about *flight, passengers' details* and *paying information.* These tables are shown in figures (Figure 39, Figure 40, Figure 41, Figure 42, Figure 43 and Figure 44).

```
Create table flight
(
flight_id serial primary key not null,
from varchar(25) not null,
to varchar(25) not null,
outward_date date not null,
return Boolean not null,
return_date date,
adults int2 not null,
children int2
);
```

Figure 39 flight's table

```
Create table adult
(
adult_id serial primary key not null,
title int2 REFERENCE titles,
first_name varchar(20) not null,
family_name varchar(30) not null,
address varchar(50) not null,
special_requirements varchar(100),

);
```

Figure 40 adult's table

```
Create table child
(
child_id serial primary key not null,
first_name varchar(20) not null,
family_name varchar(30) not null,
date_of_birth date not null,
address varchar (50) not null,
special_requirements varchar(100)
);
```

Figure 41 child's table

```
Create table card
(
card_id serial primary key not null,
card_issuer int2 REFERENCE issuers,
card_number char(16) not null,
security_number char(3) not null,
expiry_date date not null
);
```

Figure 42 card's table

```
Create table titles
(
title int2 primary key not null,
title_phrase varchar(4) not null
);
```

Figure 43 titles' table

```
Create table issuers
(
card_issuer int2 primary key not null,
issuers_phrase varchar(4) not null
);
```

Figure 44 issuers' table

```
<?xml version="1.0" encoding="ISO-885-1"?>
<data>
<flight>
<colname="flight_id" type="integer" null="false" Pk="true" serial="true" Fk="false" />
<colname="from" type="string" length="25" null="false" Pk="false" serial="false" Fk="false" />
<colname="to" type="string" length="25" null="false" Pk="false" serial="false" Fk="false" />
<colname="outward_date" type="date" null="false" Pk="false" serial="false" Fk="false" />
<colname="return" type="Boolean" null="false" Pk="false" serial="false" Fk="false" />
<colname="adults" type="integer" null="false" Pk="false" serial="false" Fk="false" />
 <colname="children" type="integer" null="true" Pk="false" serial="false" Fk="false" />
</flight>
<adult>
<colname="adult_id" type="integer" null="false" Pk="true" serial="true" Fk="false" />
<colname="title" type="integer" null="false" Pk="false" serial="false" Fk="true" ref="titles" />
<colname="first_name" type="string" length="20" null="false" Pk="false" serial="false" Fk="false" />
<colname="family_name" type="string" length="30" null="false" Pk="false" serial="false" Fk="false" />
<colname="address" type="string" length="50" null="false" Pk="false" serial="false" Fk="false" />
<colname="special_Requirements" type="string" length="100" null="true" Pk="false" serial="false" Fk="false" />
</adult>
<child>
<colname="adult_id" type="integer" null="false" Pk="true" serial="true" Fk="false" />
<colname="first_name" type="string" length="20" null="false" Pk="false" serial="false" Fk="false" />
<colname="family_name" type="string" length="30" null="false" Pk="false" serial="false" Fk="false" />
<colname="address" type="string" length="50" null="false" Pk="false" serial="false" Fk="false" />
<colname="date_of_birth" type="date" null="false" Pk="false" serial="false" Fk="false" />
<colname="special_Requirements" type="string" length="100" null="true" Pk="false" serial="false" Fk="false" />
</child>
<card>
<colname="card_id" type="integer" null="false" Pk="true" serial="true" Fk="false" />
```

Figure 45 XML document generated from database

As mentioned earlier in section 5.5.2, XML is flexible and can be built in different structures to accumulate as much information as needed. The produced XML document in this example as in Figure 45 is very similar to that one in Figure 31. However, since the table name is an essential piece of information in transformation stage, the XML document is built up of six nodes representing every table in database. These nodes consist of several child nodes describing every table columns.

The concept of transforming the XML document shown in Figure 45 is same to that one applied in 5.5.3 that suggest applying a set of rules and XSLT stylesheet to the XML document. However, since this example is dealing with several database tables

that representing in several XML nodes, the system logic is responsible of directing the flow of the applied XSLT stylesheet.

The system starts by passing the name of first table which is *flight table* to produce a web form that gathering data about the desired flight as shown in Figure 46 .

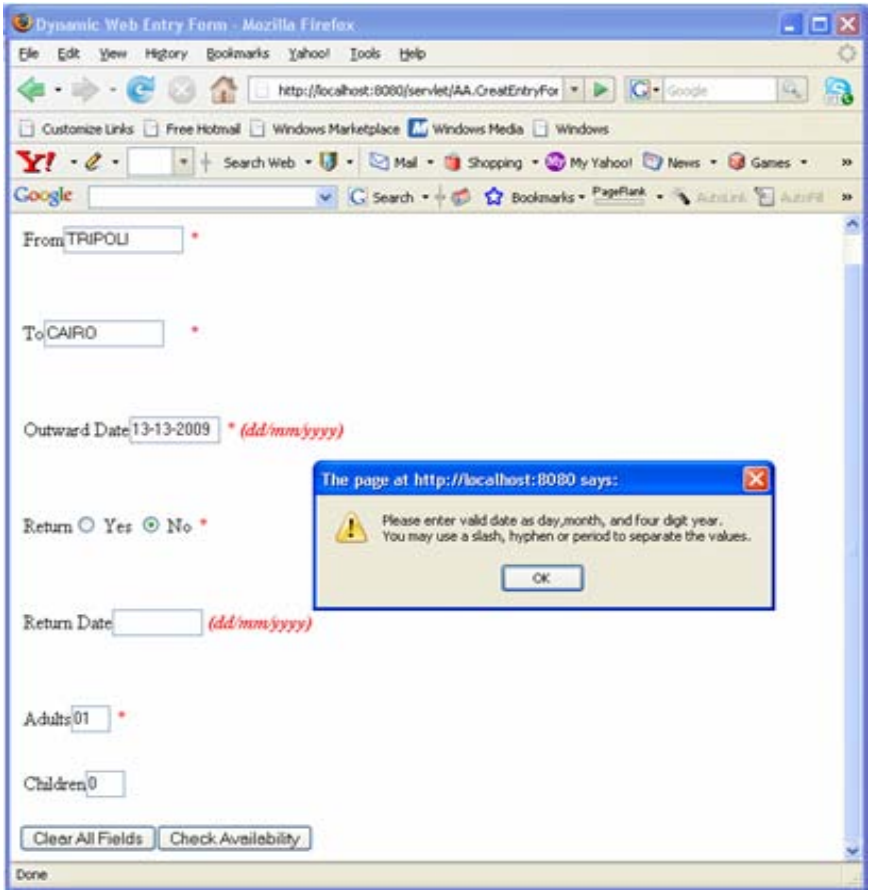


Figure 46 web entry form to collect flight's information

Assuming that all entry data was valid and the desired flight is available, then the system will pass the names of the adult and child tables with the number of adults and children in order to generate web entry forms that collect data about every passenger as shown in Figure 47 and Figure 48.

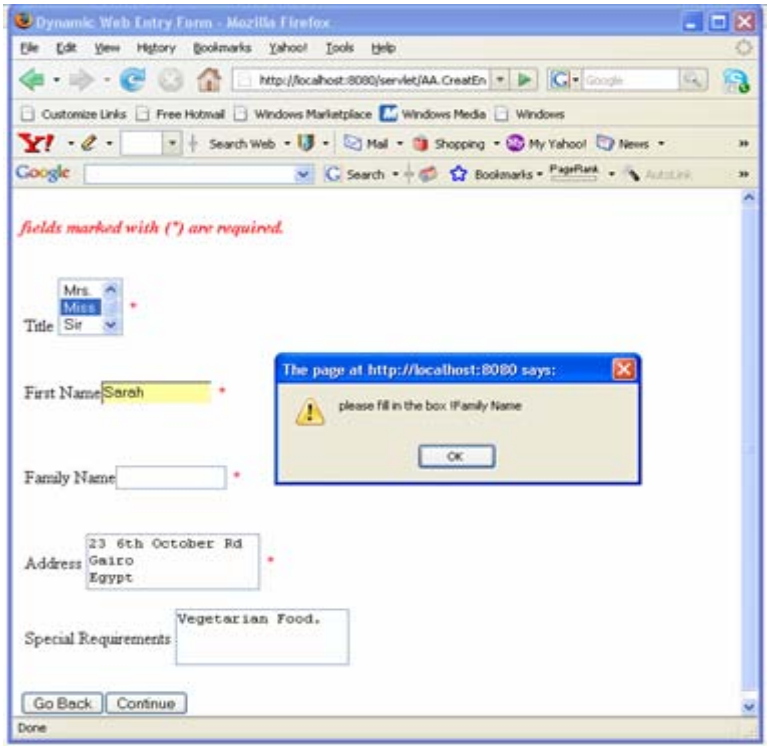


Figure 47 web entry form to collect adult's information

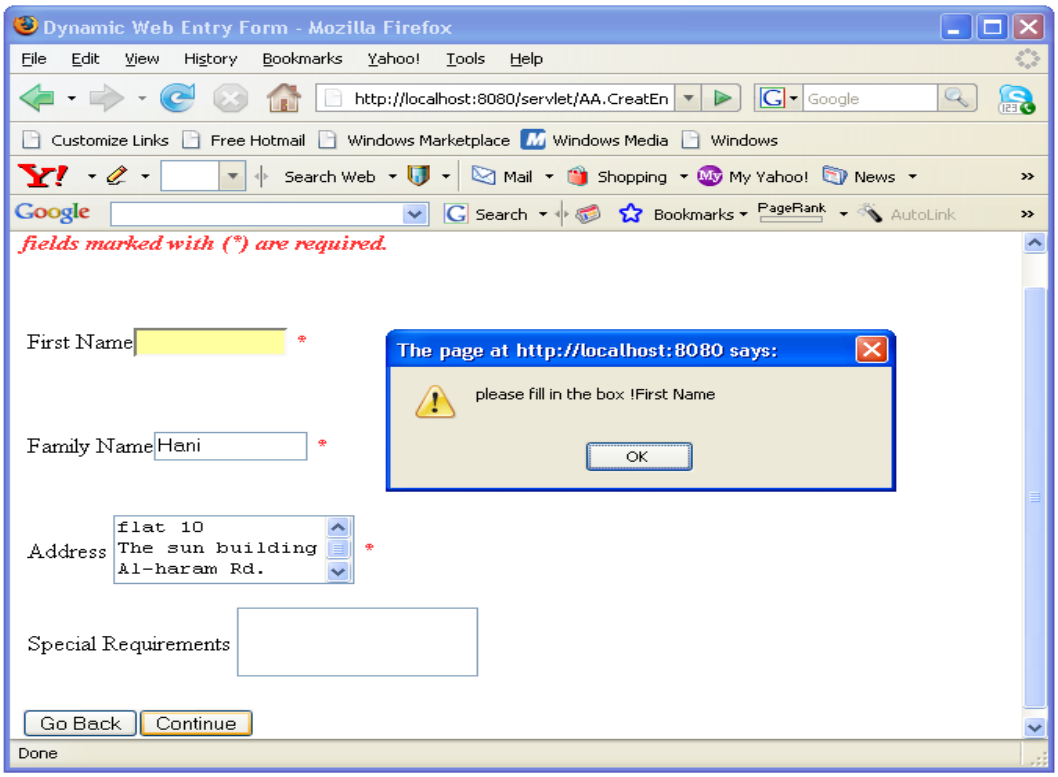


Figure 48 web entry form to collect child's information

As we can see in Figure 47 and Figure 48 the web entry form that collect child's information is similar to that one collect adult's information apart from one text field that collect the title of the passenger.

Once the user entered the whole needed information about every passenger the system will pass the name of the *card table* to the XSLT stylesheet to allow it from traversing through the XML document to transform the node that representing this table. This task produces a web entry form as shown in Figure 49.

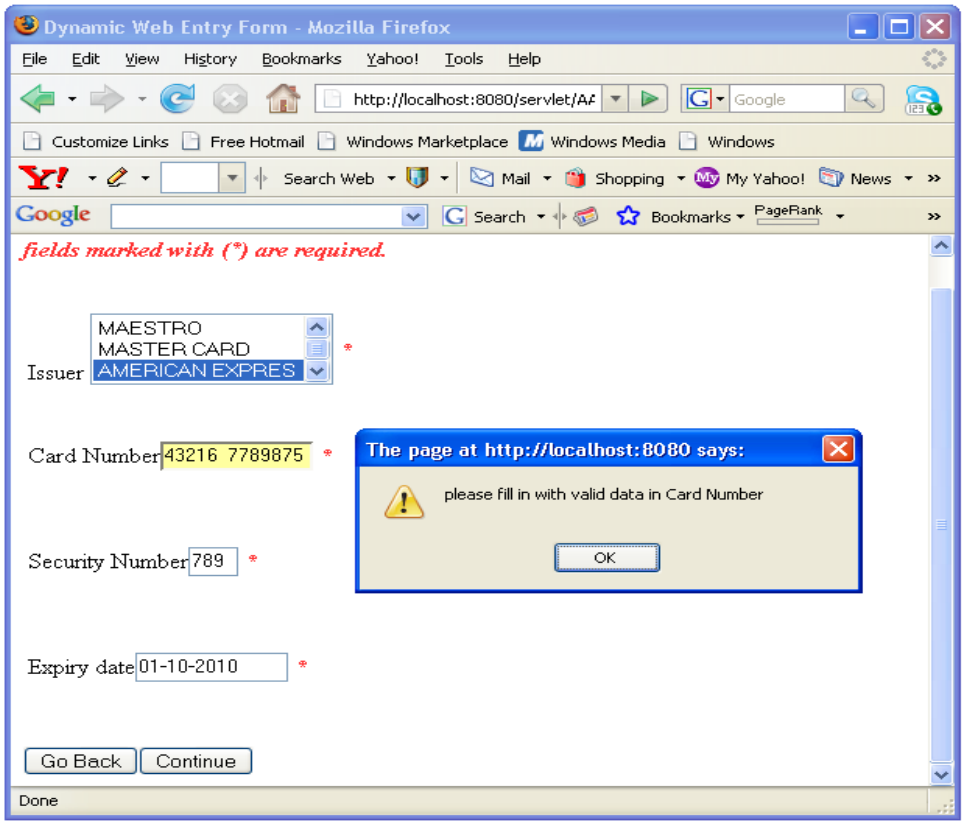


Figure 49 web entry form to collect finance information

However, the same concept of data validation in section 5.5.3.3.2 is applied in this example.

5.6.1 Arisen limitations

Through developing this example some limitation have arisen and listed as follows:

- Since the XML document is built up from database's table dynamically, the whole columns in every database table are representing in XML document in the same order that allocated in each table. In addition, the transformation task is performed in a sequential manner. As a result, the web entry form controls will appear in the same order of the columns in every table. For instance, if the *to* column in *flight's* table preceding the column *from* then the first web entry form control in Figure 46 will be *To* instead of *From.* Therefore, if the database tables were built up carelessly, the layout of web entry form controls will appear in confusable matter.
- There is no clue tells if there are some form controls should be grouped together. As a result every form control will appear in a separate line even though there are several form controls should be represented in a single line such as *title*, *first_name* and *family_name*.
- Since the transformation task is performed based on the data type of every column and a set of rules and then presented to the end user as web entry form to fill in, it is not possible to involve the actual data that entered by the end user for generating dynamic form controls. For instance, in Figure 46, if the value of *return* field was false then the *return_date* field should not appear to the user.

5.7 Conclusion

Implementing our approach shows the potential capabilities of relational database metadata that can be used to develop automatic and dynamic Web entry forms. Extracting database metadata is a straightforward task by using JDBC. Converting this metadata into XML documents in several formats makes this metadata an abstract data with the ability to transform it into many Web form types and invoked in different Web browsers as shown in Figure 32, Figure 33, Figure 35 and Figure 36. The same principle possibly could apply to producing output for not only desktop computer devices but for other devices like PDAs either by making XHTML/XForms producing a different appearance or if needed a different processing for WML (Wireless Markup Language) if that was what they needed.

Knowing and retrieving a massive amount of information from database metadata does not make it an optimal tool. A lot of work on database metadata semantics is needed in order to gain a high level of accuracy.

XForms seem to be an elegant solution for developing Web entry forms, however, the lack of native support of major Web browsers may be the big challenge for this technology.

Chapter 6

Conclusions and Future work

6.0 Conclusion

Designing and generating abstract dynamic user interfaces for web database applications and most other types of application is a very complicated task. It is often tedious, expensive and error-prone. Automatic and dynamic generation of Web user interfaces has gained a huge amount of attention in the last two decades and many efforts have been made to achieve this target.

The common feature of current tools for the automated design of user interfaces is their ability to generate the static layout of an interface from the application's data model using intelligent programs that apply design rules.

Other models and approaches adopted the use of a domain model to generate both, the layout of user interface and its dynamic behaviour. Although this approach overtakes the shortcomings of its predecessor, user interface generation based on this approach is most useful for domain-specific interfaces.

For sake of developing a dynamic Web user interface that is not targeted to any particular domain, it is argued that by making use of an existing model (i.e. the information held in database metadata) it would be possible to generate automatic and dynamic user interfaces that target any domain. Although this tendency proved the valuable notion of using database metadata in developing automatic and dynamic Web user interface, some shortcomings are associated with employing database metadata for this purpose such as mixing content, logic and presentation.

This thesis is built on the same notion, but it contributes toward a number of significant improvements for the current employment of the database metadata notion. This research proposed and evaluated an abstract framework that makes use of database metadata in conjunction with XML technology to produce different types of Web entry forms on the fly. Separation between logic, content and presentation allows user interface designers to work independently of core system developers.

Unlike other programming languages that require access to DBMSs specific system, Java can offer us a generalised database metadata based approach. Consequently, JavaServlets and JDBC were chosen to implement our approach. In addition to Java, XML technology plays a big role in designing and implementing our approach. Its ability to be transformed into many different formats of Web forms makes it the most suitable technology to implement abstract dynamic Web entry forms.

A relational database was chosen as the database technology because it provides the most needed database metadata features, such as Table/Column definition and Primary/Foreign key definition.

The developed prototype system shows a dynamic Web interface to an underlying relational database and provides automatic generation of different types of Web entry forms that facilitate swift deployment of interactive Web based applications. Automatically extracting relational database metadata and converting it to an XML document makes it possible to produce multi types of Web entry forms based on applying a particular XSLT stylesheet. This notion allows Web developers to generate Web entry forms for legacy database based systems, also it is possible to generate Web forms to target different types of platforms and small electronic devices based on applying a proper XSLT stylesheet. Moreover, since the XSLT stylesheet is responsible for converting the database metadata into Web forms and it is totally separated from the core system, generating Web forms for future types of Web forms would be a very straightforward task by writing a suitable XSLT stylesheet. The prototype system shows how database metadata can be used for data entry validation. Either by populating all required data in XHTML document elements and invoking a general JavaScript function that loops through the XHTML document verifying all data, or by using database metadata in building binding elements in the case of XForms Web forms.

In contrast to the commercial products that offer Web database integration, the implemented approach is generic because of its ability to produce any desired type of Web entry forms without the need to recode or recompile the core system. A sufficient

knowledge of XSLT language permits the generation of any particular type of Web forms.

Since the implemented approach written in Java mainly JavaServlets in conjunction with XML technology, the issues of portability and performance were overtaken. Using JDBC to bridge Web servers and database servers allows the implemented approach to communicate with any DBMS without any restrictions. Separation between the three components of the Web application: logic, content and presentation make it possible to maintain any component individually without affecting the other components. However, through this research some weaknesses were detected. Firstly, although JDBC retrieves a considerable amount of database metadata information, this information by itself is not enough in many cases to build up perfect Web entry forms. Working on database metadata semantics is very necessary to reach a considerable percentage of good quality Web forms. Database metadata semantics are very broad and even in some cases not predictable, therefore there is no guarantee that paying more attention on this issue will lead to an optimal solution. Secondly, despite the fact that XForms technology is a very elegant and promising solution, the lack of native support for this technology in major Web browsers limited its usage and implementations.

6.1 Future Work

The purpose of this research was to explore to what extent we can use database metadata in conjunction with XML technology to generate an abstract model for generation of multi type Web entry forms on the fly. As a result of this exploration the following issues are raised and can be developed in further stages.

- Investigating the capabilities of relational database metadata in building interactive Web applications that relies on using multi media elements such as working on imaging and audio data types.
- Writing XSLT stylesheets targeting small electronic devices in order to detect and resolve any shortcomings associated with this approach.
- Since the set of rules developed throughout this research is very generic, it would be possible to develop domain specific rules to support the generic rules for manipulation of semantics of database metadata. For instance, it would be possible to develop to deal with French or Spanish language domain.
- The set of rules could be re-formalised in an XML format to allow the model to be executed or adopted possibly using RuleML.

References

1. Mark, W. and C. Micah Sherr and Abigail, *Metadata tables to enable dynamic data modeling and web interface design: the SEER example.* International Journal of Medical Informatics, 2002. **65**(1): p. 51.
2. Kuo, Y.S., et al., *Generating form-based user interfaces for XML vocabularies*, in *Proceedings of the 2005 ACM symposium on Document engineering*. 2005, ACM Press: Bristol, United Kingdom.
3. Wolf, K.D., F.D. Keukelaere, and R.V.D. Walle. *GENERIC XFORMS-BASED USER INTERFACE GENERATION FOR XML SCHEMA* in *IADIS International Conference e-Society 2004*
4. Abdualrraouf, A.E. and M.J. Ridley, *Using metadata for developing automated Web system interface*, in *Proceedings of the 1st international symposium on Information and communication technologies*. 2003, Trinity College Dublin: Dublin, Ireland.
5. Mgheder, M.A. and M.J. Ridley. *Automatic Generation of Web User Interfaces in PHP Using Database Metadata*. in *Internet and Web Applications and Services, 2008. ICIW '08. Third International Conference on*. 2008.
6. http://www.w3.org/html/.
7. http://msdn.microsoft.com/en-us/library/sx7b3k7y(VS.85).aspx.
8. http://www.javascript.com/.
9. Bakken, S., A. Gutmans, and D. Rethans, *PHP 5 Power Programming*. 2004: Prentice hall.

10. *Java Server Pages Specifications version 2.1*. 2006. *available online at*: http://java.sun.com/products/jsp/.

11. W3C. *Extensible Markup Language (XML) 1.0 W3C Recommendation*. February-1998: http://www.xml.com/axml/testaxml.htm.

12. http://www.w3.org/TR/xforms/.

13. http://www.w3.org/Style/CSS/.

14. http://www.w3.org/TR/xslt.html.

15. http://etext.virginia.edu/bin/tei-tocs?div=DIV1&id=SG.

16. *ECMAScript Language Specification*. available at http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf.

17. *WHATWG community*. available at http://www.whatwg.org/.

18. Dubinko, M., *XForms essentials*. 2003, Sebastopol, Calif ; Farnham: O'Reilly. xiv, 215 p.

19. http://www.xformation.com/xforms/pr02_02.asp.

20. McLaughlin, B., *Head Rush Ajax*. 2006: O'REILLY.

21. Asleson, R., *Foundations of Ajax*. 2005: Apress.

22. *XHTML Specifications. http://www.w3.org/TR/xhtml1/*.

23. Needleman, M.H., *XML*. Standards Update, 1999. **25**(1): p. 117-121.

24. W3C. *Mathematical Markup Language (MathML™) 1.01 Specification*. http://www.w3.org/TR/REC-MathML/: 7 July 1999.

25. Gkoutos, G.V., et al. *The Application of XML Languages for Integrating Molecular ResourcesChemical Markup Language (CML)*: http://www.ch.ic.ac.uk/rzepa/xml/.

26. http://www.w3schools.com/DTD/dtd_intro.asp.

27. W3C. *XML Schema Part 1: Structures , W3C Recommendation*. May 2, 2001: *available online at:* http://www.w3.org/TR/xmlschema-1/.

28. http://www.formsplayer.com/.

29. http://www.formfaces.com/.

30. http://www.x-smiles.org/.

31. Bruchez, E. *Are Server-Side XForms Engines the Future of XForms?* in *XTech 2005: XML, the Web and beyond* 24-27 May, 2005. . Amsterdam, Netherlands.

32. Erik Bruchez, O. *XForms: an alternative to Ajax?* in *XTech 2006: Building Web 2.0*. 16-19 May 2006. Amsterdam, The Netherlands.

33. http://www.adobe.com/.

34. *WML Language References*. available online at http://developer.openwave.com/htmldoc/41/wmlref/.

35. Ruseyev, S., *WAP Technology and Applications*. 2001: Charles River Media.

36. http://www.w3.org/TR/xpath.

37. http://www.w3schools.com/XPath/xpath_syntax.asp.

38. Microsoft, *VBScript Language Reference*: *available online at:* http://msdn.microsoft.com/library/default.asp?url=/library/en-us/script56/html/ddfa5183-d458-41bc-a489-070296ced968.asp.

39. Microsoft, *JScript Language Reference*: *available online at:* http://msdn.microsoft.com/library/default.asp?url=/library/en-us/script56/html/29f83a2c-48c5-49e2-9ae0-7371d2cda2ff.asp.

40. *ECMAScript Language Specification. Standard ECMA-262 3r d Edition*. December 1999. http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf.

41. http://www.fastcgi.com/devkit/doc/fastcgi-whitepaper/fastcgi.htm.

42. Hunter, J. and W. Crawford., *Java Servlet Programming.* 1998: O'Reilly & Associates.

43. *JavaServerTM Web Development Kit, Version 1.0.1*. Available online at: http://java.sun.com/products/servlet/README.html.

44. *Jigsaw - W3C's Server, version 2.2.5*. available online at: http://128.30.52.31/Jigsaw/.

45. *Apache Tomcat*. Available online at: http://tomcat.apache.org/.

46. *Netscape Enterprise Server Release Notes.* available online at http://www.redhat.com/docs/manuals/ent-server/release-notes/es61sp2note.html.

47. *Internet Information Server*. . Available online at: http://www.microsoft.com/windowsserver2003/iis/default.mspx.

48. *Apache HHTP Server Project* Available on line at :http://httpd.apache.org/.

49. Davidson, J.D. and S. Ahmed, *Java™ Servlet API Specification*. 1998: Available onnline at: http://www.cs.helsinki.fi/u/laine/tsoha/servlet-2.1.pdf.

50. *Microsoft ASP.NET*. available on line at: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnanchor/html/anchoraspdotnet.asp.

51. Elbibas, A.A., *The use of Database Metadata to Build Adaptable Dynamic Database Interfaces for Web Applications.* ***Ph.D***. 2005, University of Bradford.

52. Schwartz, R.L., T. Phoenix, and b.d. foy, *Learning Perl*. 4th ed. 2005: O'Reilly Media, Inc.

53. *System Catalogs.* Available online at:http://www.postgresql.org/docs/8.1/static/catalogs.html.

54. *The Information Schema.* Available online at: http://www.postgresql.org/docs/7.4/static/information-schema.html.

55. Callahan, T., *So You Want a Stand-alone Database for Java.* JDJ: Java Development Journal, December 1998

56. Hamilton, G., R. Gattell, and M. Fisher, *JDBC Database Access with Java*. 1997: ADDISON-WESLEY.

57. *Data Management: SQL Call Level Interface (CLI) (CAE Specification S.)* 1995: X/OPEN Co.

58. *What Is JDBC? available online at:* http://72.5.124.55/docs/books/jdbc/intro.html.

59. *IMS, Information Management System.* available at http://www-01.ibm.com/software/data/ims/.

60. Jeffery, K.G. *Metadata the Future of Information Systems*. in *12th Conference on advanced information systems engineering*. 2000.

61. Sparenborg, J., *Dynamic User Interface Creation with ArcObjects Based on DatabaseConstraints*, in *ESRI - Professional papers*. 2004.

62. Cronin, G. *Dynamic Generation of a Database User-Interface based on Database Meta-data 'BeanBase'. available online at:* http://www.croninsolutions.com/writing/BeanBase.pdf.

63. Eliezer, K. and S. Oded, *The adaptable user interface.* Commun. ACM, 1989. **32**(11): p. 1352-1358.

64. Paulo Pinheiro da Silva, N.W.P. *User Interface Modelling with UML*. in *10th European-Japanese Conference on Information Modelling and Knowledge Representation*. May 2000. Finland: IOS Press.

65. Ralf Schweiger, A.T., et al. *Using XML for flexible data entry in healthcare example use for pathology* in *XML Europe Conference*. June 2000.

66. Polak, G. and J. Jarosz. *Automatic Graphical User Interface Form Generation Using Template Haskell*. in *TFP 2006 Seventh Symposium on Trends in Functional Programming*. 19 - 21 April, 2006. University of Nottingham, UK.

67. Cooper, M., A. Donnelly, and P. Sergeant. *User interface approaches for accessibility in complex World-Wide-Web applications- an example approach from the PEARL project*. in *6th ERCIM workshop user interfaces for all*. 25-26 October 2000. CNR-IROE, Florence, Italy.

68. ALAMN, X., et al. *Using context information to generate dynamic user interfaces.* in *10th International Conference on Human-Computer Interaction, HCI International*

2003.

69. Amihai, M., *VAGUE: a user interface to relational databases that permits vague queries.* ACM Trans. Inf. Syst., 1988. **6**(3): p. 187-214.

70. Lefer, W. *automatic graphic user interface generation for VTK*. in *The 10-th International Conference in Central Europe on Computer Graphics,*

Visualization and Computer Vision'. February 4-8, 2002. University of West Bohemia, Czech Republic.

71. Tam, N. and V. Srinivasan, *Accessing relational databases from the World Wide Web*, in *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*. 1996, ACM Press: Montreal, Quebec, Canada.

72. http://www.w3.org/MarkUp/.

73. http://www.sql.org/.

74. Papiani, M., A.N. Dunlop, and A.J.G. Hey. *Automatically Generating World-Wide Web Interfaces to Relational Databases*. in *British Computer Society Seminar Series on New Directions in Systems Development*. April 1997. University of Wolverhampton.

75. Steven, J.H., *An improved method for creating dynamic web forms using APL*, in *Proceedings of the international conference on APL-Berlin-2000 conference*. 2000, ACM Press: Berlin, Germany.

76. Elbibas, A. and M.J. Ridley. *Developing Web Entry Forms Based on METADATA*. in *International Workshop on Web Quality in conjunction with ICWE 04- International Conference on Web Engineering*. July 27, 2004. Munich (Germany).

77. Josef, J. and S. Pavel, *GUI generation from annotated source code*, in *Proceedings of the 3rd annual conference on Task models and diagrams*. 2004, ACM Press: Prague, Czech Republic.

78. Kirda, E., C. Kerer, and G. Matzka. *Using XML/XSL to Build Adaptable Database Interfaces for Web Site Content Management*. in *23rd International Conference on Software Engineering*. 2001. Toronto, Ontario, Canada.

79. Toby, J.T., Y. Dongqing, and P.F. James, *A logical design methodology for relational databases using the extended entity-relationship model.* ACM Comput. Surv., 1986. **18**(2): p. 197-222.

80. Bourret, R. *"XML and databases"*. July 2004. available at http://www.rpbourret.com/xml/xml/xml/xml/xml/xml/XMLAndDatabases.htm.

81. Volker, T., *A framework for automatic generation of web-based data entry applications based on XML*, in *Proceedings of the 2002 ACM symposium on Applied computing*. 2002, ACM: Madrid, Spain.

82. Guillen, M., et al. *GARP: a tool for creating dynamic Web reports using XSL and XML technologies*. in *Proceedings of the Fourth Mexican International Conference on Computer Science ENC 2003*.

83. Selfa, D.M., M. Carrillo, and M. Del Rocio Boone. *A Database and Web Application Based on MVC Architecture*. in *Electronics, Communications and Computers, 2006. CONIELECOMP 2006. 16th International Conference on*. 2006.

84. *Apache Struts 2 availabe at http://struts.apache.org/2.x/*

85. *JBoss community avaialbe at https://www.jboss.org/.*

86. *Tapestry framework available at http://tapestry.apache.org/.*

87. *http://en.wikipedia.org/wiki/Ruby_on_Rails.*

88. http://www.eclips3media.com/workshop/wp-content/uploads/2007/12/rails_architecture.png.

89. Eaglestone, B. and M. Ridley, *Web Database Systems*. 2001, London: McGRAW HILL.