Proceedings of the World Congress on Engineering and Computer Science 2007
WCECS 2007, October 24-26, 2007, San Francisco, USA

Using Database Metadata and its Semantics to Generate Automatic and Dynamic Web Entry Forms

Mohammed M. Elsheh and Mick J. Ridley

Abstract— **Automatic and dynamic generation of Web applications is the future of Web applications development. Database metadata holds a significant amount of information which can be used for this purpose. Separation between content, logic and presentation tasks cut down the cost of recoding and recompiling web applications when a minor upgrading is needed. Java and XML are among the most successful web technologies due to their portability feature. In this paper we aim to investigate how far can we go to make use of these technologies to develop an abstract, faster to develop, easier to maintain and less error prone Web application.**

Keywords— **JDBC, Metadata, Web application interface, XML**

I. Introduction

The interaction of database and Web is becoming the cornerstone of developing Web application systems. Shifting legacy data which was held in stand alone systems to be used in Web application systems was an expensive and time consuming chore for many corporations. HTML [1] and some server-side technologies such as CGI [2], PERL [3], Cold Fusion [4] and ASP [5] were widely adopted to put in action the above task. Internet usage is growing exponentially, this fact forces many corporations to upgrade and maintain their web application systems frequently to be market competitive which leads to high cost Web applications. To obviate this penalty, we propose using database metadata that can be extracted from the catalogue tables in relational database systems like PostgreSQL [6] and MySQL [7] by making use of JDBC [8]. Although involving database metadata, using HTML and server-side technologies such as (Java Servlet [9] and JSP [10]) has improved the development process overall, it does mix the content, logic and presentation. As a result it was not possible to change or improve the Web user interface without re-writing or re-compiling the whole system. To avert this weakness, we propose the development of a framework (Fig. 1) that fetches database metadata via JDBC and converts it into an XML [11] document via Java Servlet. The XML document can then be transformed into HTML, XHTML [12] or any other sort of web page by making use of an XSLT [13] stylesheet. As a result, all business logic processes are performed by Java Servlet. XSLT style sheet takes the responsibility of presentation task. Any small alterations in XSLT stylesheet results in a new look Web user interface without the need to recompile the system.

Manuscript received August 10, 2007.

Mohammed M. Elsheh is with the Department of Computing, Informatics school, University of Bradford, BD7 1DP, UK (phone: +44(0)1274233659; fax:+44(0)1274235856; e-mail: M.M.Elsheh@bradford.ac.uk).

Mick J. Ridley is with the Department of Computing, Informatics school, University of Bradford, BD7 1DP, UK (phone: +44(0)1274233946; fax: +44(0)1274 233920; e-mail: M.J.Ridley@bradford.ac.uk).

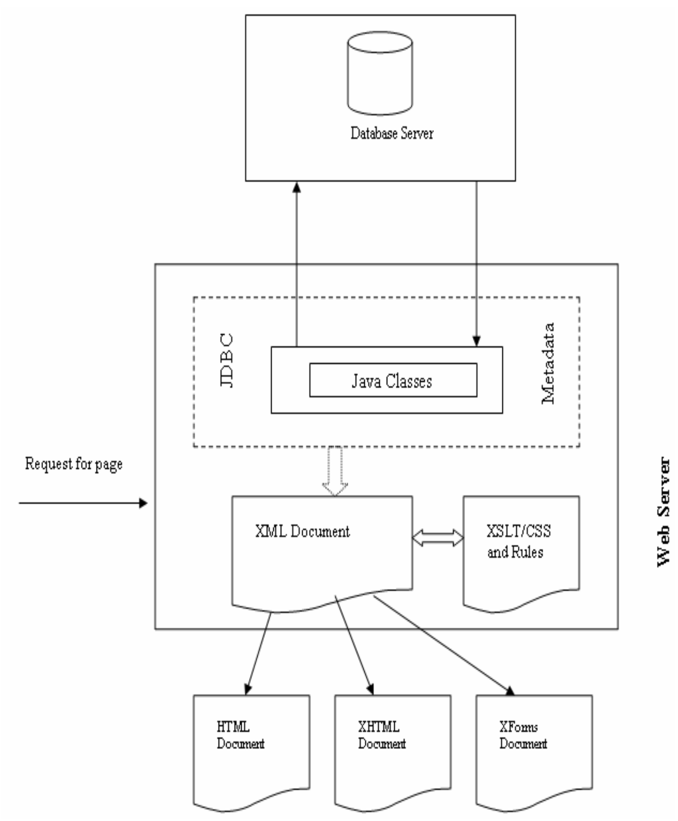


Fig. 1. Structure of the proposed framework

The remainder of this paper is organized as follow: In section II, we present some previous academic papers that are related to our work. Section III presents the framework life cycle and an example is given for demonstration purpose. Section IV describes the technique used to validate entry data. In Section V, we introduce the reason behind developing a set of rules, a sample of them is presented and this section is ended by a brief discussion of the limitations of metadata and possible solutions. Section VI concludes and gives directions for future research.

II. Previous related work

Many efforts have been made in the last few years to enhance the appearance and functionality of Web applications. This section will focus on some of these efforts. The first suggests a way that allows Web applications to access several databases. The second aims to generate dynamic Web applications based on database metadata held in database tables without making use of XML technology. The third makes use of XML technology but misses using database metadata.

Nguyen and Srinivasan [14] suggested a general purpose solution which can be used to create Web applications that can access several databases by using a page layout paradigm which encapsulates HTML and SQL [15].This approach is based on a 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. They use the mechanism mentioned above in the design and implementation of the system called DB2 WWW Connection that enables quick and easy construction of application that access relational DBMS data from the Web.

Elbibas and Ridley [16] proposed a method to generate dynamic Web entry forms based on database metadata. A database was queried and the extracted metadata was used to build the web entry form. Form elements had been constructed based on the metadata. For instance, size of the text field is determined based on the length of that column. Also help messages were generated by using Java and metadata. This approach presented a good way to produce a dynamic user interface, however, using a Java Servlet to produce HTML forms directly led to the impossibility of altering the user interface appearance without re-coding and re-compiling the source code.

Kirda, E., C. Kerer, and G. Matzka in [17] describe how XML/XSL can be deployed in creating adaptable database interfaces using WebCUS, 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 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, columns, rows and relations between tables in the database.

III. Framework life cycle

The life cycle of the proposed framework consists of several stages. These stages are demonstrated as follow:

- A connection to a database system is made and metadata about this database and its tables are extracted via JDBC.
- Java Servlet class converts this metadata into an XML document.
- This document is transformed into an XHTML document by using an XSLT stylesheet in conjunction with a set of rules (section V). This 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.
- XHTML document is returned back to the client as a Web form which enables the user to fill in. JavaScript functions are invoked for validation purpose. Entered data is submitted back to Java server to re-validate and store it into database system.

This methodology unites the best characteristics of the above methods (section II). Dynamic use of metadata via Java and JDBC eliminates problems of separate storage of database information but use of XML to describe database and form elements allows much more flexibility in the user interface.

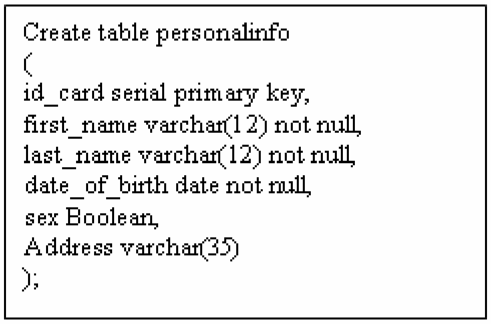
Example:

For demonstration purpose, the following tasks were performed:

1. A database table was created as in (Fig. 2).
2. Metadata was retrieved via JDBC and converted into XML document via Java code. As shown in Fig. 3, metadata for each column in the database table is presented as a single node in the XML document which is <Form_Element>. Although JDBC fetches a significant amount of metadata information, only required information for generating and validating a Web form entry is placed in the XML document. Applying an XSLT stylesheet to this document resulted in XHTML Web form as shown in Fig. 4.
3. At this stage the Web application allows the user to fill in the form. Entered data is validated on client side against the metadata in order to minimise the trips between the server and the client. Finally this data will be sent back to Java Servlet for extra validation to maintain data integrity then it is stored into database.

As we can see in Fig. 2 database table consists of six columns. Each column is mapped to a specific user interface element based on a set of rules (Section V). However, Fig. 4 displays a snapshot of an XHTML web form generated on the fly based on database metadata. The web form contains only five Web form elements due to the fact that the first column in database table (*id_card*) is a *Serial* data type which means it is not mapped to any form element and its value will be incremented automatically when entered data is stored to database (section V.1 Rule1).

However, *first_name and last_name* columns are mapped into input text boxes with length size equal to 12 characters where as *address* column is mapped to a text area form element since its length is greater than 30 characters (section V.1 Rule3). Database metadata is also used to give an indication of what data type each field should accept (in the case of data types other than strings) and whether it allows for null values or not. Furthermore, metadata is used in conjunction with JavaScript to display error messages informing the user of the name of the field and its data type.



```
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)
);
```

Fig. 2. Database table's structure

IV. 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 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 metadata to include the required information to validate the form element in the name of the element, as shown in the following program segment.

<input type="text" name="address" size="35" idtype="String" idnull="False">.

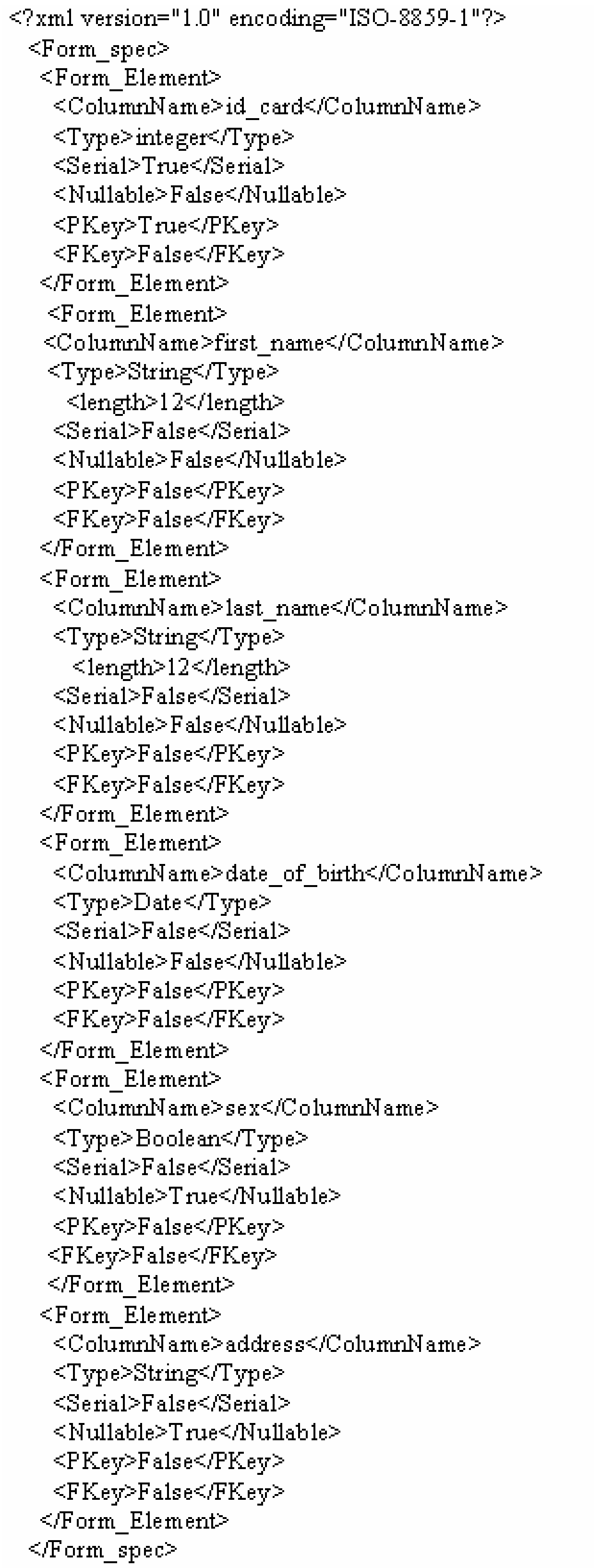
Then a generic JavaScript function is invoked to loop through the form elements and parse the names to validate the data in the element.

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 the 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 is produced. The second, it moves a lot of load to the client side whereas the validation code production is loaded on the server-side in the first approach.

V. Developing the rules or heuristics

Clean separation between content, logic and presentation tasks is the key of success for developing automatic and dynamic Web applications. To achieve this goal 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 the data itself. Since any Web user 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 user



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

Fig. 3. XML document created from database metadata

interface following changes to database tables or 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

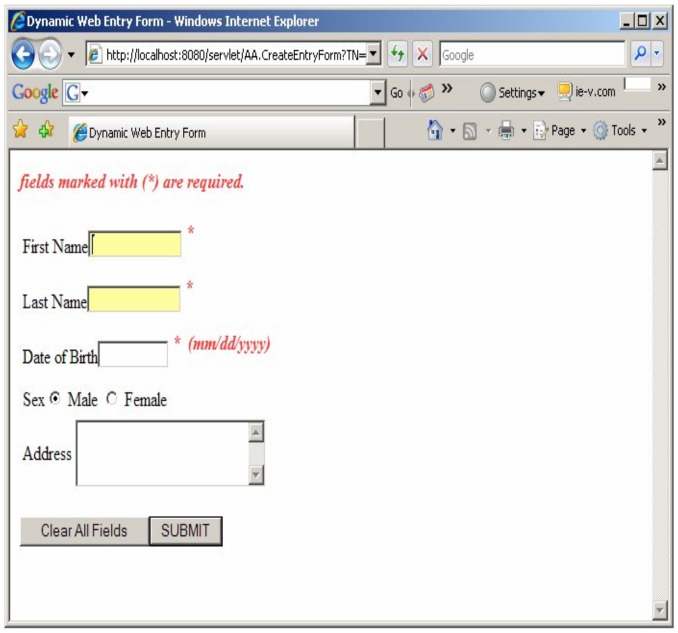

Dynamic Web Entry Form - Windows Internet Explorer
http://localhost:8080/servlet/AA.CreateEntryForm?TN=
fields marked with (*) are required.
First Name *
Last Name *
Date of Birth * (mm/dd/yyyy)
Sex Male Female
Address
Clear All Fields
SUBMIT

Fig. 4. A snapshot of an XHTML Web Entry Form generated on the fly

in a database table to 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.

V.1. Examples of these rules

The set of rules shown below is built based on the columns' data type. However, more rules can be built based on the semantics of columns' name.

- Rule1: If a column is a serial, then it should not be mapped to any form element.
- Rule2: 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 making use of the data itself for possible values.
- Rule3: If a column is character type and less than 30 characters (for example), then it should be mapped to text box. If longer than or equal to 30 characters it could be mapped to textarea control.
- Rule4: If a column is a Boolean then refer it to a choice and 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 example in XForms where the browser has more flexibility to render Web form elements.
- Rule5: If a column is a string and its length (l=x) then a JavaScript function is called to check that the length of input is acceptable.

This set of rules can be easily extended since there is a clear methodology to implement such basic rules.

V.2. Limitation of database metadata

In the proposed framework, each database column is mapped to a specific Web form element based on the metadata and set of rules. Although JDBC retrieves a significant amount of information about each column in database, the most reliable piece of information that can give indication of how to perform the mapping task is a column data type. However, this piece of information is insufficient in some cases to generate a perfect Web form entry. This raises interesting questions about the limits of automation particularly in the context of internationalization. For example, there is no fact that confirms that a specific column is intended to be a password, there is no password data type, but there may be a strong clue that we require a password text box from its name. This leads us to consider the issue of the semantic of columns' names.

In our approach we have considered to tackle this issue, another rule is developed and added to the rules' list. For instance, the following assumptions could be used to map a specific database column into password text box instead of input text box.

If a column *x* is a *String* data type and its 6<= length <= 12 and its name is password (or similar see below), then this column should be mapped into a password text box.

To make this assumption more general and less of an ad-hoc solution, it is suggested that a list of "password" words could be populated in an XML configuration file. This file could contain non-English words such as French or Spanish words for internationalization purposes.

Another example, as we can see in (Fig. 2) the *sex* column is a Boolean and as a general rule, any Boolean column is mapped to a group of radio buttons with a label True/False. However, to give a meaningful label for this column, its name can be interpreted and then the label is derived i.e. if a column's name is a *sex* or *gender* then the label should be presented as Male/Female assuming that Male is associated with True and Female with False. As in the password example, this may be further internationalized via data in XML configuration files.

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 words in upper case. For instance, if a column's name is a *first_name* it could be presented as *First Name* (See Fig. 4).

VI. Conclusion and Future Developments

The proposed approach in this work suggests that generating an abstract Web user interface for Web applications can be achieved via the potential metadata that is stored in system catalogue tables and integrating it with XML technology. This offers a new solution for generating reliable Web user interface for Web applications. Clean separation between content, logic and presentation also is achieved which overcomes the existing problems with old Web technologies such as Perl, ASP and JSP that implemented in conjunction with HTML. However, more work on the semantics of metadata is needed to achieve a high percentage of web form accuracy.

XForms is a combination of two of the most successful experiments ever performed with the Web: XML and forms [18]. As a future work we aim to investigate and test for how we can make use of XForms technology and database metadata in solving browsers compatibility and how XForms features can reduce or eliminate the need to use JavaScript for data validation tasks.

References

1. *HTML Specifications.* *http://www.library.cmu.edu/Unofficial/WebCourse/specs.html*.
2. *CGI Specification.* *http://www.utoronto.ca/webdocs/HTMLdocs/NewHTML/bibliography.html#6*.
3. *The source for PERL.http://www.perl.com/*.
4. *COLDFUSION Developer's Journal*. http://coldfusion.sys-con.com/.
5. Meyer, J., *Creating Database Web Applications with PHP and ASP*. 2003: Charles River Media.
6. *PostgreSQL: The world's most advanced open source database, http://www.postgresql.org/*.
7. *MySQL:: Developer Zone, http://www.mysql.org/*.
8. Manu, K., *An Introduction to JDBC*. Linux J., 1998. **1998**(55es): p. 2.
9. Hunter, J. and W. Crawford., *Java Servlet Programming*. 1998: O'Reilly & Associates.
10. *Java Server Pages Specifications version 2.1*. 2006. *available online at*: http://java.sun.com/products/jsp/.
11. Needleman, M.H., *XML*. Standards Update, 1999. **25**(1): p. 117-121.
12. *XHTML Specifications.* *http://www.w3.org/TR/xhtml1/*.
13. Burke, E.M., *Java and XSLT*. 2001: O'Reilly & Associates, Inc., 101 Morris Street, Sebastopol, CA 95472.
14. 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.
15. *SQL Language Reference.* *http://ugweb.cs.ualberta.ca/~c391/manual/chapt6.html*.
16. 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).
17. 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.
18. Dubinko, M., *XForms essentials*. 2003, Sebastopol, Calif ; Farnham: O'Reilly. xiv, 215 p.