A Rule Framework for Automatic Generation of Web Forms

Atia M. Albhbah and Mick J. Ridley

Abstract—**This paper proposes the use of common sense rules and domain specific rules rulebases using Reaction RuleML format in conjunction with database metadata rule that can be used to support the development of automated web forms. Database metadata can be extracted from system catalogue tables in typical relational database systems by several tools depending on Web application environment. The results show that this mechanism successfully insulates application logic from code.**

Index Terms—**Web forms, database metadata, XML, reaction ruleML.**

I. Introduction

Many web based applications, in commercial and scientific areas used forms to enter data for storing or querying database systems. These database systems contain information about data stored as a set of system catalogues which are known as metadata [1]. In addition, metadata consists of information such as; list of database tables, column names, and all integrity constraint rules, which will be used to control data that is saved and manipulated in the database tables. If applications are built by hand all this information in a database should be embedded into the application. This can be time consuming and may require ongoing maintenance.

Automatic and dynamic generation of Web applications is moving into the mainstream in the web development field nowadays, because many business agencies are looking to change their database applications into on-line systems, and the growth of technologies like XML[2], XHTML[3] and many others, have pushed them to update their Web applications using existing databases to take advantages of these technologies. Separation between content, logic and presentation of Web applications has become an important issue for faster building and easy maintaining, as CSS applied on the client side to let Web developer control the overall presentation of their Web applications. In addition CSS uses rules to control the appearance of the document [4]

In this paper we develop the initial use of RuleML in conjunction with database metadata originally proposed in [5] to a more general framework that allows "common sense" and "domain specific" rules to be included in the system. The aim is to extend the automation of web forms so that more semantic information is used in a consistent fashion. We investigate the use of Reaction RuleML on the server side to give a consistent use of variables and therefore a consistent look and feel to forms across pages within applications using a database. We know that web site maintenance is a problem and just as use of CSS on the client side can give consistency to the appearance of pages generally; use of a set of rules can give a similar consistency to the appearance and operation to any set of forms that interact with the same database.

We illustrate our approach with the development of a banking based example later but introduce it with a simple UK car registration example. If a database column reg_no was defined as char (8) and not null we would use the database metadata to produce a form of suitable size, which would be required for data entry. However a textbox can have additional data put into it, with the addition of domain specific knowledge we could supplement the behaviour of the form to enforce a maximum of 7 characters and that only numbers, spaces and uppercase letters (excluding I and O) were permitted. This behaviour should be applied to any form using the reg_no field.

Therefore we propose a framework as structured in Fig. 1 where we supplement database metadata rules with common sense rules and introduce a second rulebase of domain specific rules.

The common sense rules add functionality not limited to a specific domain but also not supported by database metadata which is often limited by factors such as the type system of the database itself [6]. In this category are rules like those mapping a column called password to a password type form input.

In this paper we investigate the use of Reaction RuleML on server side to give consistent use of variables and therefore a consistent look to forms across pages within applications using database. Therefore we use Reaction RuleML format to store database metadata rules and save it as rulebase, In addition, we propose a framework as in Fig. 1 that divides the rules into two types. The first one is to save the common sense rules and the second to save domain specific rules which will help to develop a prototype system that can generate automatic and dynamic web entry forms for Web applications.

Rules on the Web have become an increasingly important issue during the last couple of years in both industry and academia areas. It has been concluded that when rules are embedded in application code it becomes difficult to locate and change the logic [6], and each modification requires recompiling the application code. Hence, separating rules from application code allows easily manipulation of the rules. RuleML (Rule Markup Language) is an XML-based markup language which allows rules to be expressed as modular components using standard XML tags [7]. RuleML format can be used to represent metadata retrieved from database [8]. In this case RuleML could be used to save rules retrieved

Manuscript received May 29, 2012; revised June 30, 2012.

The authors are with the Department of Computing, School of Computing informatics and media, University of Bradford, Richmond Road, Bradford, BD7 1DP, UK (e-mail: A.Albabah@ student.bradford.ac.uk, Tel.: +441274 233946, fax: +441274 234282; M.J.Ridley@Bradford.ac.uk).

from database metadata as a rulebase which in turn separates the rules from the application code to improve accessibility, provide more flexibility, and control.

This paper is organised as following: in Section II we review related work on interfaces and provide an introduction to RuleML and Reaction RuleML. Section III illustrates the use of RuleML implemented on a prototype using PostgreSQL accessed via PHP. Conclusions and suggestions for future development are presented in Section IV.

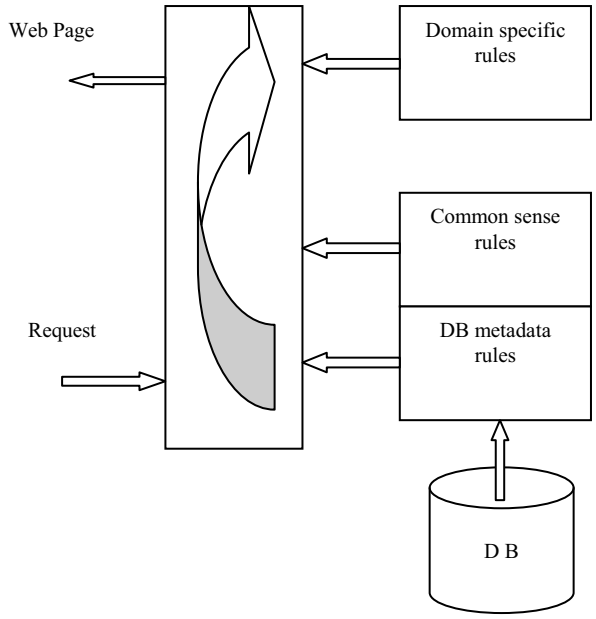

Fig. 1. Structure of the proposed framework.

II. Previous Related Work

A. Metadata Driven Interfaces

Weiner et al. [9] describe an approach of dynamically generate web based database interfaces. This was using a manually developed metadata table, which contains information about the elements required to represent a data model (table names, field names, data type, and links between tables). In the described model since the metadata is built by hand it is possible that the web interface will not be an accurate representation of the database and it needs more effort.

Elsheh and Ridley [6] proposed a model which aims to generate dynamic Web entry forms based on metadata extracted from system tables. They used the java servlet class to convert the extracted metadata via JDBC into an XML document. A set of rules has been developed and applied to database metadata which is used to map each column to specific user interface controls. In addition, the XML document is transformed into an XHTML document using XSLT stylesheet, which is returned back to the user as Web entry form. The set of rules of this scheme is embedded in the application code where it is difficult to locate and change their logic.

Mgheder and Ridley [10] suggested an approach that uses metadata stored in system tables in databases (columns name, type, size etc.) to develop generic user interface elements. They used PHP as the server script and the database abstraction library ADOdb to achieve their goal. The metadata is extracted from the database by using the ADOdb metadata methods. This metadata information combined with a developed set of rules is used to automatically map each column in the database table to a specific user interface control. The proposed model uses a set of rules which are extracted from the database to build the Web form; these rules are built within the application code, where it is not easy to maintain them.

Bertossi and Jayaraman [8] describe a methodology that uses metadata for a virtual and relational data integration system under the local-as-view (LAV) approach. They used a standard format based on XML and RuleML for representing metadata. This design allows data sources to be added to the system or removed without affecting any other data sources. Albhbah and Ridley [5] proposed a new way where a rulebase is used in conjunction with database metadata to develop automatic Web forms. They described how to develop an abstract way to achieve the goal. In their approach, the separation of the rules from the application code using Reaction RuleML format to represent metadata retrieved from database, which saved as a rule base has improved accessibility, provided more flexibility, and control.

B. RuleML

Rules on the Web are becoming a more mainstream topic these days and are expected to play an important job in the success of the Semantic Web [7]. Rule Markup Languages (RuleML) will be the vehicle for using rules on the Web [11]. RuleML has been defined by the Rule Markup Initiative to express a family of Web rules to support both forward (bottom-up) and backward (top-down) rules in XML for deduction, rewriting, transformation and reaction[7]. It is used to create a basis for a universal rule Markup Language using standard XML tags, which helps to specify rules, and allows exchange, manipulation and analysis of rules. RuleML is a family of sublanguages which was launched in August 2000 and it is now at version 1.0, which was released in 2010 [7]. The initiative is very flexible as it uses XML and it is not limited only to proposing a language but also it can be translated to some targeted rules engines (e.g. RuleML to JESS). Before executing RuleML rules, the rules need to be translated to an inference engine language, such as Java Expert System Shell (JESS) or Prolog to be executed. But in our approach we focus on how to use RuleML format to save rules as readable base, details are provided in section III.

In Fig. 2, RuleML specifies different types of rules [12] which are described as follows:

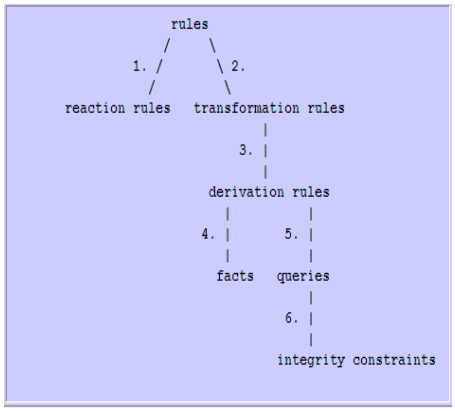

Fig. 2. A graphical view of RuleML rules [12].

- Reaction Rules (event-condition-action rules) can only be applied in the forward direction in natural, observing/checking events/conditions and performing an action if and when all events/conditions have been recognized/fulfilled as in the example *"When a share price drops by more than 5% and the investment is exempt from tax on profit, then sell it"* [11]. The reaction rule specifies the reactive behavior of a system in response to events.
- Transformation Rules (functional-educational rules).
- Derivation Rules (implication-inference rules) can be applied in both forward and backward directions as in the example *"A gold customer is a customer with more than $1Million on deposit"* [11].
- Facts as in the examples *"Tom sells TV to Landa"*, *"A module is option"*.
- Queries *"provide 10% fees discount for students attending all the lectures"*.
- Integrity Constraints (consistency-maintenance rules) as in the example *"A driver of a rental car must be at least 25 years old"* [11].

Example of the general form of RuleML 1.0 syntax is given below [13]:

<!-- Implication Rule 1: Backward notation of 'then' and 'if' roles, as in Logic Programming, and forward notation using natural 'if' ... 'then' order, as in textbook logic, with exact same meaning

"The discount for a customer buying a product is 5.0 percent if the customer is premium and the product is regular."

Notice that the ternary discount relation is applied via an Atom. Furthermore, a Data constant can syntactically be an entire phrase like "5.0 percent". It will unify only with variables and with Data having exactly the same spelling (incl. spaces)

-->

```
<Implies>
        <if>
        <And>
        <Atom>
        <Rel>premium</Rel>
        <Var>cust</Var>
        </Atom>
        <Atom>
        <Rel>regular</Rel>
        <Var>prod</Var>
        </Atom>
        </And>
        </if>
        <then>
        <Atom>
        <Rel>discount</Rel>
        <Var>cust</Var>
        <Var>prod</Var>
        <Data>5.0 percent</Data>
        </Atom>
        </then>
</Implies>
```

C. Reaction RuleML

Reaction RuleML is a branch of the RuleML family; it is described as a general language and rule interchange for the family of reaction rules [14]. Reaction RuleML introduced different types of production, action and reaction rules into the native RuleML syntax. The design aim of Reaction RuleML is to be easy to learn and facilitate fast maintenance with less risk [15].

The general syntax for Reaction RuleML has been updated to be easy to use, where more tags have been added to the new version (0.2) [16]. The general form of the Reaction RuleML0.2 syntax is given below:

```
<Rule style="active" evaluation="strong">
<label><!-- metadata --></label>
<scope><!-- scope --></scope>
<qualification><!-- qualifications --></qualification>
<oid><!-- object identifier --></oid>
<on><!-- event --></on>
<if><!-- condition --></if>
<then><!-- conclusion --></then>
      <do><!-- action --></do>
<after><!-- postcondition --></after>
<else><!-- else conclusion --></else>
<elseDo><!-- else/alternative action --></elseDo>
<elseAfter><!-- else postcondition --></elseAfter>
</Rule>
```

The building blocks of the Reaction RuleML 0.2 general form is contain one general rule form "<Rule>", three execution styles "active/messaging/reasoning" and two evaluation "weak/strong". This kind of rules suits the type of rule we are concerned with (see section III) and will be used in the prototype development.

III. Application Development

This section introduces a prototype development system which aims to design a framework of using the RuleML format to save and implement constraint rules to overcome the limitations that database metadata information is only used when data enters the database and use the rules with the metadata to generate automatic and dynamic Web forms.

The general idea of the prototype implementation was the creation of a Web form to evaluate to what extent we can use the relational database metadata and build the rule base using a RuleML format to save the rules as two types the first one is common sense Rulebase and the second one is domain specific Rulebase, which will be used to build adaptable dynamic database interfaces. The metadata will be extracted using a number of PHP's PostgreSQL functions. The following objectives are intended to be achieved.

- Extract metadata.
- Apply domain specific Rulebase.
- Apply common sense Rulebase.
- Generate Web form element.

A. Building RuleML Metadata Rule Bases

In this section we introduce two types of rules that can be applied to the information that exists in database metadata; the first rule base is to save all common sense rules as:

- Rule 1: if a column is integer type, then it should be mapped to textbox Web form control.

- Rule 2: if a column is character type and its length is less than or equal to 30 (for example), then it should be mapped to textbox Web form control.
- Rule 3: if a column is character type and its length is more than 30, then it should be mapped to textarea Web entry form.
- The condition on the length of the field in Rule 2 and Rule 3 could be more or less and could be changed.
- Rule 4: if a column is Boolean type, then it can be implemented as a group of radio buttons or drop down menu. But in this framework radio buttons will be used. In addition in this rule we can use the name of the column to make the Web form more clear.
- Rule 5: if a column is date type, then it could be mapped to a textbox and the format of the date is presented as label.
- Rule 6: if a column name is password or sub set of these words, then this column should be mapped into a password textbox.

Rules 1 – 5 are generic rules, based on data type information held in database metadata.

Rule 6 is common sense rule based on column name information held in database metadata.

Some of the above developed rules using RuleML format are illustrated in Fig. 3 as:

The second rule base is to save all domain specific rules, using our example which is customer bank information we develop set of rules as domain specific rules as:

- Rule 1: if a column name is card number and its size is 16, then it should be mapped to textbox Web form control with the exact size of 16 digits.
- Rule 2: if a column name is sort code and its size is 6, then it should be mapped to textbox Web form control with the exact size of 6 digits.
- Rule 3: if a column name is account number and its size is 8, then should be mapped to textbox Web form control with the exact size of 8 digits.
- Rule 4: if a column name is security number and its size is 3, then should be mapped to textbox Web form control with the exact size of 3 digits.
- Rule 5: if column name start date or end date and its size is 7, then should be mapped to textbox and the format of the date is presented as label. These dates are not specified as of date type in the database and we would want a month and year representation normally in the same format as actually used on a card e.g. 02/2013 not Feb/2013 etc. and not a JavaScript calendar tool specifying a single day as can be seen on some sites.

Some of the above developed rules using RuleML format are illustrated in Fig. 4 as:

B. Retrieving Metadata from Database

We start implementing our approach by creating a database table which contains customer bank information, to illustrate a range of data types and other conditions (nullability) as:

```
CREATE TABLE bankinf (
account_number integer NOT NULL,
sort_code integer NOT NULL,
card_number integer NOT NULL,
security_number integer NOT NULL,
start_date integer NOT NULL,
end_date integer NOT NULL);
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Rulebase>

<Rule>
        <if><type>integer</type></if>
        <then>textbox</then>
</Rule>
<Rule>
        <if>
              <type>varchar</type>
              <size>lessthan30</size>
        </if>
        <then>textbox</then>
</Rule>
 <Rule>
        <if>
              <type>varchar</type>
              <size>morethan31</size>
        </if>
        <then>textarea</then>
</Rule>
<Rule>
        <if>
              <type>character</type>
              <size>lessthan30</size>
        </if>
        <then>textbox</then>
</Rule>
 <Rule>
        <if>
              <type>character</type>
              <size>morethan31</size>
        </if>
        <then>textarea</then>
</Rule>
<Rule>
        <if>
              <type>bpchar</type>
              <size>lessthan30</size>
        </if>
        <then>textbox</then>
</Rule>
<Rule>
        <if>
              <type>bpchar</type>
              <size>morethan31</size>
        </if>
```

Fig. 3. Metadata rule base in RuleML format as common sense rules.

A number of PHP's PostgreSQL functions are used to make a connection to the database and retrieve the metadata, examples are shown below:

- To get the column metadata information as an array. we use the function '*pg_meta_data*'as follows:

```
$meta = pg_meta_data($conn,'bankinf');
```

Where '$conn' is the connection handle and 'bankinf' is the table's name [5].

There is another method to retrieve information about each field such as column's name, type and so on, by using specific functions as explained below. The output generated from these functions [5].

- pg_field_name( ) to return the column's name.
- pg_field_type( ) to return the column's type.
- pg_field_prtlen( ) to return the column size.
- pg_field_size() to return the internal storage size in bytes.
- pg_field_is_null() to test if a field is SQL null or not allows null.

- pg_num_fields() to return the number of columns in result resource.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Rulebase>

<Rule>
        <if><name>card_number</name></if>
        <then><size>16</size></then>
</Rule>
<Rule>
        <if><name>sort_code</name></if>
        <then><size>6</size></then>

</Rule>
<Rule>
        <if><name>account_number</name></if>
        <then><size>8</size></then>
</Rule>
<Rule>
        <if><name>secority_number</name></if>
        <then><size>3</size></then>
</Rule>
<Rule>
        <if><name>start_date</name></if>
        <then><size>7</size></then>
</Rule>
<Rule>
        <if><name>end_date</name></if>
        <then><size>7</size></then>
</Rule>
</Rulebase>
```

Fig. 4. Metadata rule base in RuleML format as domain specific rules.

C. Generate the Web forms

A general purpose PHP script was written which loops through all the metadata for each column and uses the RuleML rulebases. It tests to see which rules apply and then uses those rules to build the form elements on the fly as shown in Fig. 5 where for example the account_number is mapped to a textbox of the appropriate size (found from the metadata) and marked as required since it is specified as non null, its label is formatted as described below. Every column in the database table is mapped to a specific Web form control element. The label of each control element is the actual column's name in the database, retrieved from the database table metadata. PHP functions can used to produce a user friendly label. For instance, functions are used to replace underscores which separate words in a column's name by spaces and change the first character of all words to upper case. As a guide to the user and to make the form simpler we have used (*) for the required fields (columns that are primary key or specified as not null). This can be supplemented with JavaScript to ensure a value is provided.

IV. Conclusion and Further Work

Automatic and dynamic generation of Web forms is entering the mainstream in web development for supporting developing online systems. Rules extracted from database metadata and used to generate the Web forms when embedded in the application code are not efficient due to the difficulty of locating and changing the logic. In this paper we proposed an approach which separates the rules as independent entity from the application code, by using Reaction RuleML0.2 format as rulebase. The system evaluation is carried out using Reaction RuleML0.2 format to save the developed rules, PostgreSQL as a DBMS, PHP for server side programming, HTML and JavaScript for client side programming. As a result a Web form for user interface is generated dynamically. This approach aims to produce common sense rules and introduce a second rulebase of domain specific rules using Reaction RuleML0.2.

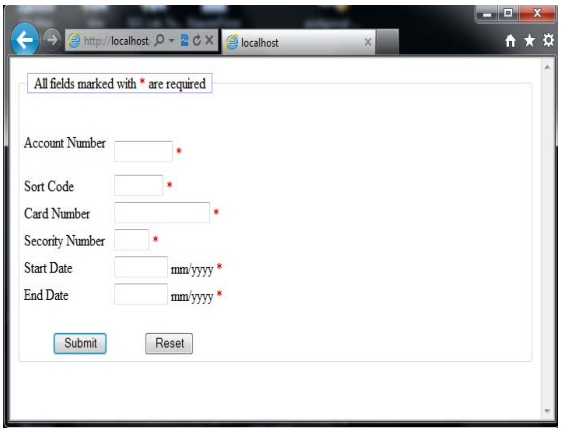

Fig. 5. User interface form generated automatically using metadata and RuleML.

No performance issues have been noted yet but we will investigate further when we have larger, more realistic sets of rules working with real databases in place. Additionally the size of database's metadata is usually small compared to the actual data, and only increases with the number of tables rather than the volume of data per table this has lead to small RuleML files which are not complex to parse.

There are a number of areas for further development which we hope to pursue:

- The current example only shows labeling for required fields this can be developed so that the rule involves applying a suitable piece of JavaScript code to enforce a non null value.
- Further development of rules that involve code to implement more semantics, development of rules to order and or group form elements (e.g. if we had end_month and end_year as separate fields instead of end_date keep those together).
- In our current implementation we have used database metadata which in this case is derived from relational database system catalogs but could be obtained from other sources such as XML Schema and used in conjunction with XML data.

References

[1] System Catalogs. [Online]. Available: http://www.postgresql.org/docs/8.1/static/catalogs.html
[2] Introduction to XML. [Online]. Available: http://www.w3schools.com/xml/xml_whatis.asp
[3] W3C XHTML Activities. [Online]. Available: http://www.w3schools.com/w3c/w3c_xhtml.asp
[4] "Cascading Style Sheets. [Online]. Available: http://www.w3.org/Style/CSS/
[5] A. M. Albhbah and M. J. Ridley, "Using RuleML and database metadata for automatic generation of web forms," in *Proceedings of the*

10th International Conference on Intelligent Systems Design and Applications (ISDA), 2010, pp. 790-794.
[6] M. M. Elsheh and M. J. Ridley, "Using Database Metadata and its semantics to Generate Automatic and Dynamic Web Entry Forms," in *Proceedings of the World Congress on Engineering and Computer Science (WCECS 07)*, 2007, pp. 654-658.
[7] The Rule Markup Initiative. [Online]. Available: http://ruleml.org/
[8] L. Bertossi and G. Jayaraman, "Designing, Specifying and Querying Metadata for Virtual Data Integration Systems," *Globe 2009, LNCS 5697*, Springer, 2009, pp. 72-84.
[9] M. S. M. Weiner and A. Cohen, "Metadata tables to enable dynamic data modeling and web interface design: the SEER example," *International Journal of Medical Informatics*, vol. 65, no. 1, pp. 51-58, April 2002.
[10] M. A. Mgheder and M. J. Ridley, "Automatic Generation of Web User Interfaces in PHP Using Database Metadata," in *Proceedings of the 3rd International Conference on Internet and Web Applications and Services*, ICIW '08, IEEE Computer Society, 2008, pp. 426-430.
[11] W. Gerd, A. Grigoris, T. Said, and B. Harold, "The Abstract Syntax of RuleML-Towards a General Web Rule Language Framework," in *Proceedings of the 2004 IEEE/WIC/ACM International Conference on Web Intelligence: IEEE Computer Society*, 2004, pp. 628-631.
[12] RuleML Design. [Online]. Available: http://ruleml.org/indesign.html
[13] Schema Specification of RuleML 1.0. [Online]. Available: http://ruleml.org/1.0/
[14] Reaction RuleML. [Online]. Available: http://ibis.in.tum.de/research/ReactionRuleML/
[15] A. K. A. Paschke, H. Boley, M. Kifer, S. Tabet, M. Dean, and K. Barrett. (2006). Reaction RuleML. [Online]. Available: http://ibis.in.tum.de/research/ReactionRuleML/
[16] Primer Reaction RuleML 0.2. [Online]. Available: http://ibis.in.tum.de/docs/docs_research/docs_ReactionRuleML/0.2/docs/ReactionRuleML-v0.2-Primer.pdf