DYNAMIC WEB FORMS DEVELOPMENT USING RULEML

A. M. Albhbah

PhD

UNIVERSITY OF BRADFORD

2013

DYNAMIC WEB FORMS DEVELOPMENT USING RULEML

Building a framework using metadata driven rules to control Web Forms generation and appearance

Atia Mahmod Albhbah

Submitted for the degree of

Doctor of Philosophy

Department of Computing

School of Computing, Informatics and Media

University of Bradford

2013

Atia M. Albhbah “DYNAMIC WEB FORMS DEVELOPMENT USING RULEML”
Keywords: Web Forms, Database Metadata, XML, RuleML, Reaction RuleML.

Abstract

Web forms development for Web based applications is often expensive, laborious, error-prone, time consuming and requires a lot of effort. Web forms are used by many different people with different backgrounds and a lot of demands. There is a very high cost associated with the need to update the Web application systems to achieve these demands.

A wide range of techniques and ideas to automate the generation of Web forms exist. These techniques and ideas however, are not capable of generating the most dynamic behaviour of form elements, and make Insufficient use of database metadata to control Web forms’ generation and appearance.

In this thesis different techniques are proposed that use RuleML and database metadata to build rulebases to improve the automatic and dynamic generation of Web forms.

First this thesis proposes the use of a RuleML format rulebase using Reaction RuleML that can be used to support the development of automated Web interfaces. Database metadata can be extracted from system catalogue tables in typical relational database systems, and used in conjunction with the rulebase to produce appropriate Web form elements. Results show that this mechanism successfully insulates application logic from code and suggests that the method can be extended from generic metadata rules to more domain specific rules.

Second it proposes the use of common sense rules and domain specific rules rulebases using Reaction RuleML format in conjunction with database metadata rules to extend support for the development of automated Web forms.

Third it proposes the use of rules that involve code to implement more semantics for Web forms. Separation between content, logic and presentation of Web applications has become an important issue for faster development and easy maintenance. Just as CSS applied on the client side to control the overall presentation of Web applications, a set of rules can give a similar consistency to the appearance and operation of any set of forms that interact with the same database. We develop rules to order Web form elements and query forms using Reaction RuleML format in conjunction with database metadata rules. The results show the potential of RuleML formats for representing database structural and active semantics.

Fourth it proposes the use of a RuleML based approach to provide more support for greater semantics for example advanced domain support even when this is not a DBMS feature. The approach is to specify most of the semantics associated with data stored in RDBMS, to overcome some RDBMSs limitations. RuleML could be used to represent database metadata as an external format.

Declaration

I hereby declare that this thesis has been genuinely carried out by myself and no portion of the work referred to in this thesis has been used in any previous application for a degree. The invaluable participation of others in this thesis has been acknowledged where appropriate.

Atia Albhbah

Acknowledgements

In the Name of Allah, the Most Gracious, the Most Merciful all praise is due to Allah who helped me throughout my life, and for his glorious ability and granting me the health and ability to complete this thesis.

I would like to express my gratitude to all those who supported and encouraged me to complete this thesis.

First of all, I would like to give my sincere thanks to my supervisor *Mr Mick Ridley* for his guidance and support from the initial to the final of this research. He has offered me all the assistance to complete this thesis.

I would like to thank my mother, wife and children for their support and love.

Last but not least, I also thank *Mrs Bev Yates* for her assistance.

List of Abbreviations

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
APL	Array Programming Language
ASP	Active Server Pages
CGI	Common Gateway Interface
CSS	Cascading Style Sheets
DB	Database
DML	Data Manipulation Language
EER	Extended Entity Relationship
HTML	Hypertext Markup Language
JDBC	Java Database Connectivity
JESS	Java Expert System Shell
JSP	Java Server Pages
ODBC	Open Database Connectivity

OWL	Web Ontology Language
PHP	PHP: Hypertext Pre-processor
RDBM	Relational Database Management
RDBMS	Relational Database Management System
RIF	Rule Interchange Format
RuleML	Rule Markup Language
SDL	Schema Definition Language
SGML	Standard Generalized Markup Language
SQL	Structured Query Language
SWRL	Semantic Web Rule Language
W3C	The World Wide Web Consortium
WebCUS	Web Content Update System
WWW	World Wide Web
XHTML	Extensible HyperText Markup Language
XML	Extensible Markup Language

XPath	XML Path Language
XPointer	XML Pointer Language
XQuery	A Query Language for XML
XSL	Extensible Stylesheet Language
XSLT	Extensible Stylesheet Language Transformations

List of Author's Publications

1. 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, available on line at: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=05687166
2. A. M. Albhbah and M. J. Ridley, " A rule framework for automatic generation of Web forms," in Proceeding of the 4th IEEE International Conference on Computer Science and Information Technology (IEEE ICCSIT 2011), China, June 2011, available on line at: http://www.ijcte.org/abstract/537-A608.htm
3. A. M. Albhbah and M. J. Ridley, "An extended rule framework for Web forms: adding to metadata with custom rules to control appearance," International Journal of Machine Learning and Computing, IACSIT, Singapore, Dec 2011, 1 (5): 466-472, available on line at: http://www.ijmlc.org/papers/70-A608.pdf.

Table of Contents

List of Tables

List of Figures

Chapter 1

1 Introduction

1.1 Introduction

Can we imagine how our life would be without the Internet? How much of our daily routine would change without electronic communication like Email, Facebook and Twitter? How much harder to access information without search engines?

The Internet effectively spreads into all domains of our daily life, such as eLearning, eCommerce and eGovernment. It has changed our life, changed ways of buying goods, finding people, making travel reservations and more. Web application systems are the most powerful Web systems. In Web applications, users interact with the system by filling in Web forms to supply several types of data. Many Web based applications, in commercial and scientific areas use forms to enter data for storing or querying database systems.

Automatic and dynamic generation of Web applications is moving into the mainstream in the Web development field nowadays [1 , 2], because many agencies are looking to change their database applications into on-line systems, and the growth of technologies have pushed them to update their Web applications using existing databases to take advantages of these technologies.

In this thesis we investigate the use of RuleML (Rule Markup Language) to store database metadata rules and save it as a rulebase, which will help to develop a prototype system that can generate automatic and dynamic HTML forms for Web applications.

This chapter introduces the motivations behind this research and the objectives that are going to be studied and investigated throughout this research.

1.2 **Motivations**

Building Web applications takes a lot of time and the longer it takes to develop, amend and maintain the greater the budget required. Therefore increasingly developers are looking for ways to automate the development process and reuse information. We aim to contribute to that by developing methods of storing and using rules in combination with database metadata.

Databases contain information about data stored as a set of system catalogues which are known as metadata [3]. In addition, metadata consists of all the 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 manually all this information in a database should be embedded into the application. This can be time consuming and may require ongoing maintenance. Thus, automatic and dynamic generation is a preferable way of building or updating Web applications using the database metadata to ensure consistency.

Information retrieved from database metadata alone is not enough to be used to generate the best Web form element for each column. To use the information extracted from metadata, it needs a set of supporting rules which will enable us to map each column to the most appropriate form element.

From the start of RuleML project in 2000, rules on the Web have become an increasingly important issue 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 [4], and each modification requires recompiling the application code. A rulebase approach will also allow extensions to rules beyond those from metadata. Separation between content, logic and presentation of Web applications has become an important issue for faster development and easy maintenance [1 , 5]. Hence, separating rules from application code allows easily manipulation of the rules.

As use of CSS rules to control the appearance of the document on the client side which gives consistency to the appearance of pages generally [6]; the use of a set of rules can give a similar consistency to the appearance and operation of any set of forms that interact with the same database.

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 a database [8], in this case it could be used to save rules retrieved from database metadata as a rulebase which in turn separates the rules from the application code to improve accessibility, provide more flexibility, and control. The rules will help in designing the query forms, and in some cases for example we can invoke a suitable RuleML, which will help to map the column to the correct element control, and store all possible values for one column as a domain.

Some database systems do not support advanced features such as domains and composite attributes, for example MySQL does not support user defined domains [9 , 10] which can be created as a data type and then that type used in a table definition. To overcome this type of limitation RDBMS data can be represented by storing database metadata in a standard external format that can be used by design tools and for transforming database specifications between RDBMSs. RuleML format allows us to overcome variations between RDBMSs.

1.3 Aims and Objectives

This thesis aims to investigate how a RuleML format can be used to store database metadata rules and save it as a rulebase. This will help to develop a prototype system that can generate automatic and dynamic Web entry forms for Web applications. In order to achieve these aims, the objectives of the thesis are set as follows:

- Carry out an extensive literature review about the most commonly used database and Web technologies available for creating and developing Web sites and Web applications.
- Carry out an extensive study to investigate and discover the most strongly connected related work, and how extracted rules stored as some form of database metadata can provide us with sufficient information to achieve the main aim.
- Reformulate the rules and convert them into code and store them in RuleML (Rule Markup Language) format as rulebase to replace the hard coded rules within applications with a readable reusable format.
- Design a more general framework that includes as many rules as possible in the system. The aim is to extend the automation of Web forms so that more semantic information is used in a consistent fashion.

- Develop domain specific rules to extend the generic rules for manipulation of semantics of database metadata.
- Generate Web interface forms that access the database automatically using the rulebase.
- Create an extended Rules Framework for Web Forms by adding to the metadata with custom rules to control appearance of Web form elements in a semantic way.
- Store database metadata in an external format to maximise support for advanced features such as domains, not supported by all systems, so that more sophisticated Web entry forms can be generated dynamically and automatically.

1.4 Contributions

The thesis proposes a number of contributions to the field of automatic and dynamic generation of Web forms. These contributions can be summarized as:

- **A rulebase approach:** to make sufficient use of database metadata to control web forms' generation. The use of a RuleML format to store a set of supporting rules as a rulebase to overcome database metadata limitations and to separate rules out of the applications code. This work has been published in (A. M. Albhbah and M. J. Ridley, ISDA 2010).

- **Domain specific rulebase**: the contribution in this task was to design a more general framework that includes as many rules as possible in the system. The aim is to extend the automation of Web forms so that more semantic information is used in a consistent fashion. Rules do not support the manipulation of semantics of database metadata in some cases; a domain specific rulebase was developed to support the common sense rulebase for the manipulation of semantics of database metadata. A more general framework was designed to extend the automation of web forms. This work has been published in (A. M. Albhbah and M. J. Ridley, IEEE ICCSIT, June 2011).
- **Control appearance of Web form elements:** just as CSS is applied on the client side to aid the overall presentation of Web applications, the contribution in this task was to create an extended Rules Framework for Web Forms by adding to the metadata with custom rules in order to control the appearance of Web form elements in a semantic way. This work has been published in (A. M. Albhbah and M. J. Ridley, IACSIT, Dec 2011).
- **Extending the use of RuleML to store metadata and database semantics:** the contribution in this task was the use of RuleML to represent database metadata in an external format to maximise the support for advanced features and overcome RDBMSs limitations such

as varied support for domains and composites. Through this more sophisticated Web entry forms can be generated dynamically and automatically.

1.5 Thesis Structure

The rest of this thesis as follows:

Chapter 2: Background

This chapter presents an overview of the most commonly used database technologies, surveys Web technologies available for creating and developing Web sites and Web applications, discusses the Extensible Markup Language and its relevant sublanguages, including an overview of the family of Rule Markup Languages including RuleML the most used technique in our prototype. This chapter attempts to describe the techniques and basic concepts used throughout this thesis.

Chapter 3: Literature Review

This chapter surveys a number of academic papers and articles most strongly connected or related to the work presented in this thesis. It reviews the literature on developments in database technology and Web applications over the last few years. The overview was carried out to understand the current state-of-the-art in this area. It introduces different approaches on user interfaces to databases which focus on automatic mechanisms to be used for generating user interfaces, extracting database metadata to construct Web user interfaces, XML and Web applications, RuleML as a rulebase and using RuleML format to save rules.

Chapter 4: Using RuleML and Database Metadata for Generating Automatic and Dynamic Web Entry Forms

This chapter introduces a prototype development system which aims to test the use of the RuleML format to support the development of automated Web interfaces. This chapter begins with an overview of the proposed prototype. Then a RuleML metadata rulebase is built based on metadata rules extracted from the database catalogue. The implementation of the proposed prototype is then discussed with an example that shows how the proposed approach works.

Chapter 5: A rule framework design

This chapter introduces our suggested framework implementation. We aim to design a more general framework that includes as many rules as possible in the system. The aim is to extend the automation of Web forms so that more semantic information is used in a consistent fashion. So we investigate the use of Reaction RuleML0.2 which is a development from original 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 accessing a database. The aim is to extend the automation of Web forms so that more semantic information is used in a consistent fashion.

Chapter 6: An extended rules framework for Web forms: adding to metadata with custom rules to control appearance

This chapter proposes the use of rules that involve code to implement more semantics for Web forms. It extends the work presented in Chapter 5 by developing an additional set of rules to control the appearance of Web forms. In particular, a set of rules are proposed to control the appearance of Web form elements in a semantic way using Reaction RuleML 0.2 format in conjunction with database metadata rules.

Chapter 7: Extending the use of RuleML to store metadata and database semantics

This chapter proposes the use of a RuleML format to implement further semantics for Web forms. RuleML can be used to represent RDBMS data structures by storing database metadata in an external format for some design tools. Just as XML Schema which uses the elements and attributes to express the semantics of XML data, in principle RuleML could be a representation for RDBMS data too. In this chapter, a similar approach to the role for XML Schema is presented. The approach is tested by using it to represent domain features supported by some but not all DBMSs.

Chapter 8: Conclusion and future work

This chapter presents conclusion of the thesis and discusses limitations of the research. Finally, possible future research areas are pointed out.

Chapter 2

2 Background

2.1 Introduction

This chapter is spilt into three main sections. The first section gives an overview of the most commonly used database technologies. The second section surveys Web technologies available for creating and developing Web sites and Web applications over the last few years. The third section discusses the Extensible Markup Language (XML) and its relevant sublanguages ending this section with an overview of the Rule Markup Languages which including RuleML the most used in our prototype. So this chapter attempts to describe the techniques and basic concepts used throughout this thesis.

2.2 Database Technologies

Interaction of database and the Internet has become a cornerstone in the development of Web application systems. Databases and database technology have a significant impact on the increasing use of computers. The survey was taken to understand the current state-of-the-art in this area. It is fair to say that databases play a crucial role in almost all areas where the use of computers is central, including education, business, medicine, commerce and engineering to name a few. This Section provides an overview of Relational Databases which are the most commonly used for Web applications and which are used for implementation of the prototype system to be developed during this research. We also overview tools used to access a database, and extraction of relational database metadata.

2.2.1 Database

A database is a shared collection of structured data and its description, designed to meet the information needs of an organization [11]. It can be manipulated according to its integrity rules to ensure that the database is at least plausible and shared by application systems. The main task of any database is to store data in a way that can be used easily.

2.2.1.1 Database data models

A data model is a set of structures used to build a database, and the process that can be applied to deal with databases, and safety rules that guarantee the existence of the database. Data model quality can be judged on its ability to deal with the data and its requirements. So there are many database models such as:

- Non-relational database
    - Hierarchical data model.

  - Network data model.
- Relational database.
  - Relational data model.

The most commonly database used for Web applications is relational databases because of the stability on a large scale, and the acceptance and speed.

2.2.1.2 Relational database

The relational model was first published in 1970 by Edgar F. Codd [12], it was the first widely accepted model for database analysis and design. A relational database is a collection of tables with rows and columns used to store data; each table has a name defined by the person who created it, these tables are created, updated, read using SQL (Structured Query Language), and controlled using a Database Management System (DBMS). In a relational database tables, called relations, that consist of columns, called attributes, each column contains a set of values from the same domain which represent facts from the real world, and rows, called tuples, where each row contains all the information about one object [12]. In other words the Relational Database Management System (RDBMS) is a collection of relations (tables), containing data, which are connected by the chosen unique attributes. Furthermore, there is normalization theory which can be applied to each database design to answer these questions:

- How many tables will be in the database?
- What information is being represented?
- Which column will be in which table?
- Is there a relationship between the tables?

Normalization is the process of simplifying the database design, which will result in a good design. By following the Normal Forms rules, every table must have a primary key which uniquely identifies table rows, and foreign keys which are columns used to reference a primary key in another table. In this case each data item entered to a foreign key column should match the referential primary key data. In addition the RDBMS holds both data and a description of this data, known as metadata [11].

2.2.2 SQL

Structured Query Language (SQL) is the main language used for managing data in RDBMS, it was first defined by D. D. Chamberlin and others at IBM San Jose Research Laboratory California in the early 1970s [13] it was pronounced as its previous name Structured English Query Language (Sequel) and was later changed to SQL. It is used for creating, querying and updating relational databases. SQL has tools for summarizing, calculating, etc. Data can be combined from multiple tables using table's relationship. SQL has two branches:

- Schema Definition Language (SDL) defines database structures and its integrity constraints.
- Data Manipulation Language (DML) manipulates data in relational database.

2.2.3 Metadata

The database system contains the data and the description of the database structure and its constraints, which is stored in the DBMS catalogue and known as metadata. Metadata is defined as data about data [14], which contains information on how each table is structured, data type and storage format of each item. In addition it lists tables in each database, column names in each table and so forth. Moreover a relational database includes a set of system catalogue tables for describing the logical and physical structure of the data [3]. This information is useful for generating dynamic Web entry forms, providing such information as:

- Name of database tables.
- Number of columns in a database table.
- Column name.
- Column type, which could be a special type such as serial.
- Column size.
- Column is primary key or not.

- Column is foreign key or not.
- Column accepts null values or not.

There is more information which could be retrieved from database metadata such as check constraints which allow restrictions such as:

- State a minimum, maximum or range requirement for the value in a column for example price >= 0, Age >= 18 && Age <=60.
- Each element must respect its type and restrictions of its corresponding domain.
- A foreign key value should be retrieved from a parent table's primary key or candidate key.

Although some information can be retrieved from database metadata there are limitations. To make some information more useful we may need some supporting rules for example:

- There is no database type to tell if a column is a password. But by using some rules applied to the information retrieved from the database we may be able to determine if it is a password based on a domain definition or set of words used to name the column.
- Unless the database system supports composite types there is no way to group columns together as one block for example name information

(title*, first name, last name*), or address information, bank account information.

2.3 **Web Technologies**

This section gives an overview of languages, Client-Side scripting languages, Server-Side scripting languages, and Extensible Markup Language (XML) available for creating and developing Web sites and Web applications.

2.3.1 HTML

Hypertext Markup Language (HTML) has been used to create documents on the World Wide Web (WWW) since its first implementation in the 1990s [15]. HTML allows the user to publish online documents with structure such as heading, text, tables, lists, photos, and retrieve online information via hypertext links. HTML forms allow data to be collected from clients for processing, they enable users to create applications that include database functionality and provide access to the data, but a significant limitation of HTML forms is their dependency on scripting languages. HTML forms are reliant on scripts to accomplish many common tasks such as performing validations, marking controls, displaying error messages, and calculations [16]. HTML 1.0 was the first release of HTML to the world; it was very simple and used to put simple text onto the Web. Since the early days of the Web HTML has gone through several versions which are listed below:

- HTML 2.0

This was the first definitive one of the HTML family. It added a few new features such as password, input types, radio buttons, reset, submit, check box to forms.

- HTML 3.2

It became a W3C recommendation in January 1997 [17].It improved HTML 2.0 by supporting the use of new features such as tables, applets, fonts, superscripts, subscripts, text flow around images[18].

- HTML 4.0

It became a W3C recommendation in April 1998. The most notable additional feature was the use of Cascading Style Sheets (CSS), and it supports more multimedia options, better printing facilities, scripting languages. In December 1999 W3C recommended HTML 4.01, it was a minor update of corrections and bug-fixes from HTML 4.0 [17].

- XHTML

Extensible HyperText Markup Language (XHTML) reformulates HTML 4.01 in XML [19]. It is an application of XML, and it takes advantage of XML's strict syntax to ensure pages are well- formed. It combines the data structure and extensibility strengths of XML and the formatting strengths of HTML 4.01.

Comparison of HTML 4 and XHTML [20] is shown below:

HTML	XHTML
	Documents must be well formed
uppercase recommended to use for the standard tags and attributes.	Lowercase must be used for element and attribute names.
Some elements such as <p> (paragraph) element could omit end tags. <p>First paragraph. <p>Second paragraph.	End tags are required for non-empty elements. <p>First paragraph.</p> <p>Second paragraph.</p>
Attribute values is not quoted.	Attribute values must always be quoted.
It allows some attribute value to be minimized.	Attribute value pairs must be written in full.
No end tag for Empty elements. <hr>	Empty elements must either have an end tag or the start tag must end with />. <hr> </hr> , <hr/>

Table 2-1 Comparison of HTML 4 and XHTML [20].

- HTML 5.0

It is the successor to HTML 4 and XHTML [20] . HTML 5 defines an HTML syntax that is compatible with HTML 4 and XHTML1 documents published on the Web. It allows for MathML (Mathematical Markup Language) and SVG (Scalable Vector Graphics) elements to be used inside a document [21]. It added some new functions for embedding audio, video, graphics, client-side data storages, and interactive documents [17]. It is still in review.

2.3.2 RDF

The Resource Description Framework (RDF) is known as a framework for describing Web resources [22] that are designed to be read and understood by computer and are not supposed to be displayed on the Web to users [23]. RDF is written in XML and used to describe resources such as properties for shopping items, Web events scheduling time and Web pages information [23], and uses web identifiers to identify resources. It is used for knowledge representation generally moving on from specifically Web resources.

2.3.3 Client –Side Scripts:

A script is program code that does not need compiling or pre-processing before being executed. So when the Web page is downloaded the browser executes the script driven by events such as mouse clicks or data entry that can make Web pages more dynamic [24].

2.3.3.1 JavaScript

JavaScript is the most commonly used scripting language, it works with most browsers. JavaScript can be added to HTML Web pages using the HTML <script> tag; it can be added to the head section or the body section or both. JavaScript in the body section will be executed while the page loads, but in the head section will be executed when called. It can do many things such as validate form data before submission for processing by a server, react to events, and put dynamic text into an HTML page [25] . The core JavaScript language has been standardised in the ECMA-262 standard [26]. ECMA standard is based on JavaScript (Netscape) and JScript (Microsoft), it is known as ECMAScript. There is another widely used scripting language based on ECMAScript which is ActionScript. It uses the Adobe Flash Player platform to provide website functionality.

2.3.3.2 AJAX

Asynchronous JavaScript and XML (AJAX), is a Web development technique for creating interactive Web applications that allows data on a Web page to be dynamically updated without reloading an entire page. It is a combination of JavaScript, markup language to return the requested data and server side language to handle the request [27]. By using AJAX servers send back the requested data to the browser without any additional information or presentation [28]. AJAX can be used to create autocomplete Web forms; it is used to update the contents of the form without reloading a whole page by using an autocomplete function, dynamically loading and displaying data from the server as a list of matching options. As example of using autocomplete in AJAX the autocomplete function will autocomplete a list of suggested country names when the user start typing in the text field as in Figure 2-1 below [29]:

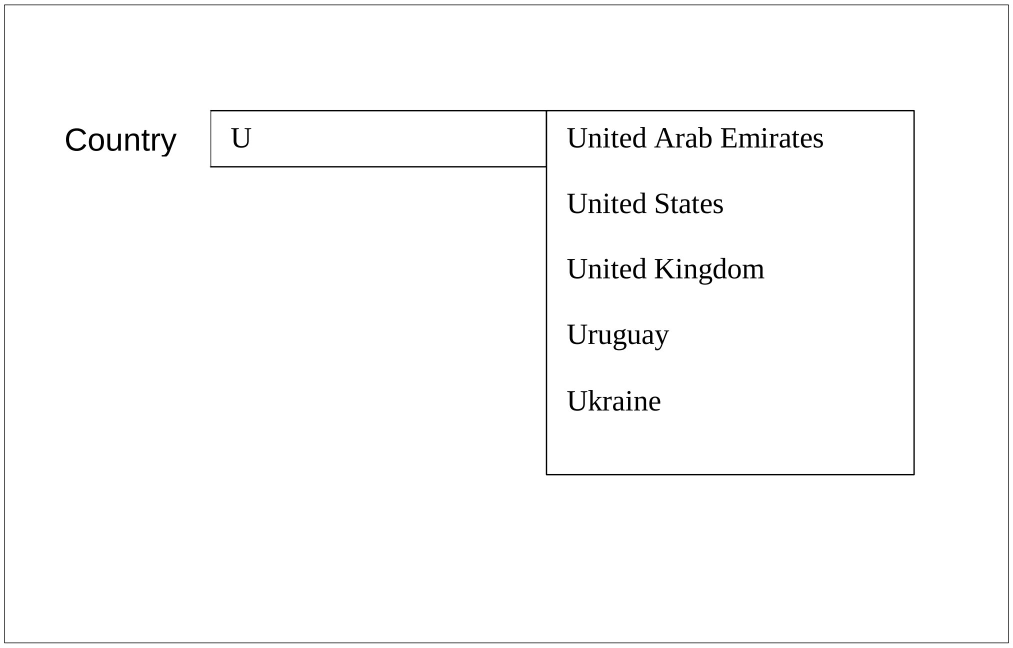


Figure 2-1 Autocomplete AJAX function Example

2.3.3.3 JSON

JavaScript Object Notation (JSON) is described as a lightweight data exchange format that is based on a subset of the JavaScript language and easy for users to read and write [30]. It is faster and easier to parse than XML [31]. JSON is sometimes seen as a lightweight alternative to XML however its inventor said "JSON is not a document format. It is not a markup language." [32] So in some ways should not been seen as a general alternative to XML which is those things but it does fit well for certain applications, a key use is for data delivery for Ajax where if the use will be in JavaScript JSON is a good and natural fit [33]. JSON is exchanging text information, much like XML and it can be translated to and from XML [34]. It is better than XML for some types of representations, e.g. object oriented data generally, arrays, but not the type of structure (rules) we are dealing with. It may be more easily readable than XML but that is not relevant to us similarly it may be more efficient with less space used for tags compared to data but that again is not relevant to us.

2.3.4 Server-Side Technologies:

This section reviews the most common used server-side technologies which are used to develop how Web servers communicate with external programs for handling a Web page request, process all necessary operations, and send the result back to the client. The main focus will be on PHP that have chosen to implement and test our framework.

2.3.4.1 CGI

The Common Gateway Interface (CGI) is a standard interface between the Web server software and the external applications. In Addition CGI is a generic interface for calling external programs to query databases. With CGI, while passing user-specific data to a program, the Web server can call up the program. The program then processes that data and the server passes the response back to the browser. CGI can be written in any language supported on the Web server host machine such as C, C++, Perl, Visual basic, and any Unix shell [35].

2.3.4.2 Perl

Practical Extraction Report Language (Perl) is a programming language originally developed by Larry Wall [36] for writing utilities that perform large amounts of string handling, text file processing, and interaction with the operating system. Perl gains its importance due to its support for a wide range of interface applications (CGI applications) , where it provides a very powerful tool that connects Perl scripts with different DBMSs in such dynamic way, such as generate dynamic Web pages and design interface between an application and one or more database driver modules. A Perl application can talk to several types of DBMSs using the same method.

2.3.4.3 PHP

PHP (PHP: Hypertext Pre-processor) is a server-side scripting language used to create dynamic Web pages for interacting with the user. Moreover, PHP can be embedded into the HTML code for serving dynamic Web pages. PHP supports many database systems such as PostgreSQL, MySQL, Oracle, so it can be used to create dynamic Web pages that are generated from information accessed from a database [37 , 38].

Amongst a wide range of libraries, PHP has facilities for parsing and accessing XML documents, SimpleXML extension provides a simple toolset to access and convert XML documents to an object that can be processed with normal property selectors and array iterators [38], introduce that there are some examples:

- Load XML file $xml = simplexml_load_file("example37.xml");

  Where example37.xml is the XML file name and $xml is a variable.

- Count XML elements

  $p_cnt = count($xml->Reaction); this function will count how many Reaction elements in the XML file.

- Counts the children of an element

  $p_cnt3 = count($xml2->table[$m]->column);

- Finds children of given node

```
$Ruletype=$xml->Reaction[$s]->event->type;
```

- Read the integer data only from XML element

```
$string="0,1,2,3,4,5,6,7,8,9";
 $Rulecolsize=strpbrk($Rulesize,$string);
```

- Gets the name of the XML element

```
echo $child->getName();
```

2.3.4.4 ASP

Active Server Pages is Microsoft's server-side scripting language that is used to develop dynamic Web-based applications. Like PHP It has the ability to embed dynamic content into HTML Web pages. In addition ASP enables server side scripting for IIS (Internet Information Server) with native support for JScript and VBScript which are executed on the server [39]. Furthermore ASP provides access to many database systems like MySQL, PostgreSQL, and Oracle, and it has similar functionality to PHP and Perl.

2.3.4.5 Java Servlets

Java Servlets are platform independent server-side Java programs used to extend Web servers, easy to use, which take advantage of the Java platform to solve the issue of CGI and proprietary APIs. The following are some of the advantages offered by servlets [40] :

- The code is executed once when the Web server loads it. Then it only calls a service method to handle a new request once the servlet is loaded and the servlet stays in memory while serving incoming requests until it is unloaded or the servlet engine is stopped.
- They are portable, so they can be moved to a new operating system without changing the source code.
- They use a standard API that is supported by several Web servers.
- They provide access to the large set of APIs available to the Java platform such as JDBC API to access a database.
- They can take advantage of the Java Security Manager.

2.3.4.6 Java Server Pages

JSP (Java Server Pages) is a Java platform technology which provides a simple way to create dynamic Web applications that are platform independent [41]. JSP combine (HTML or XML) elements with Java code to produce dynamic Web pages. Thus, a JSP document is a text-based file that mixes template data (HTML tags) with dynamic actions to generate a response to a request from a client. In addition a JSP page may contain a method to access a database by calling a JDBC (Java Database Connectivity) function which will process a requested form [42].

2.4 XML

Extensible Markup Language (XML) is a subset of Standard Generalized Markup Language (SGML); XML became a W3C Recommendation in February 1998 [43]. It is a simple and very flexible text format and a meta language used to define other languages. XML extends the power of HTML by separating data from presentation, and it is not intended to replace HTML because XML and HTML were designed with different goals. HTML was designed to display data and how data looks, but XML was designed to describe data, store data, focus on what data is, transport data [44], and exchange structured information. In the following sections we introduce a number of significant XML languages and applications.

2.4.1 XML Schema

XML Schema is an XML-based language, became a W3C recommendation in 02 May 2001[45]. It specifies XML structure, in detail it specifies the definition of each type of element in the schema and the type of the data associated with it. The Schema uses XML elements and attributes to express the semantics [11]. It replaced DTD for definitions because it is more powerful.

2.4.2 XSLT

XSLT (Extensible Stylesheet Language Transformations) became a W3C recommendation in November 1999 [46]. It is a part of Extensible Stylesheet Language (XSL) which is used to transform an XML document from one form to another, for example so that it is recognised by a browser such as (X)HTML. XSLT uses XML Path Language (XPath) to navigating nodes in an XML document for transforming to a different format like XHTML and HTML [47].

2.4.3 XPath

XML Path Language (XPath) uses path expressions to address nodes through the hierarchical structure of an XML document similar to the expressions which are used when working with a traditional computer file system [48 , 49]. XPath has many built-in functions which are used to identify XML nodes with specific characteristics which is used by XSLT to transform XML document into another XML document or (X)HTML document. Table 2-2 lists some of the path expressions for selecting XML nodes.

Functions of XPath expressions captured from [50]

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

Table 2-2 XPath expressions make up an important part of XQuery.

2.4.4 XPointer

XPointer is short for XML Pointer Language, which is built on top of the XML Path Language to allow addressing points and ranges into the internal structures of XML documents to access the content of elements or attributes [11], and used to address expressions in URI references as fragment identifiers [51].

2.4.5 XQuery

XQuery is a Query Language for XML proposed by the W3C query working group [11]. XQuery 1.0 second edition became a W3C Recommendation in December 2010 [52], and originally intended as a kind of SQL for XML data. In addition it was designed to query the XML data. It makes use of XPath expressions to navigate through XML elements in an XML document.

2.4.6 Rule Markup Languages

Rules in the Web have become a mainstream topic these days, and will play an important job in the success of the semantic Web. Rule Markup Languages will be the vehicle for using rules on the Web. In fact a Web rule Language is a concrete (XML-based) rule syntax for the Web [53]. We introduce a number of notable rule languages in a historic sequence.

2.4.6.1 RuleML

RuleML (Rule Markup Language) 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, transformational, and reaction [7]. The Rule Markup Initiative come out of RuleML to explore rule systems suitable for the Web, allow exchange of rules between different systems on the Web and interoperation between major commercial and non-commercial rules systems [54]. It is used to create a basis for a universal rule Markup Language using standard XML tags, which helps to specify rules, and allows exchanging, manipulating and analysing rules. RuleML is a family of sublanguages which was launched in August 2000 and as of 2012 is at version 1.0 [7]. The initiative is very flexible in its use of XML and it is not limited only to propose a language but also translators for some targeted rules engines (e.g. RuleML to JESS). Before executing RuleML rules, the rules have to be translated to an inference engine language, such as Java Expert System Shell (JESS) or Prolog to be executed. But in our research we focus on how to use RuleML format to save rules as readable rulebase. RuleML is used to share rule bases in XML and publish these rules on the Web [55]. It designed to be the interchange format of the most Web rules in an XML format [56]. In Figure 2-2, RuleML shows different types of rules which are described as follows:

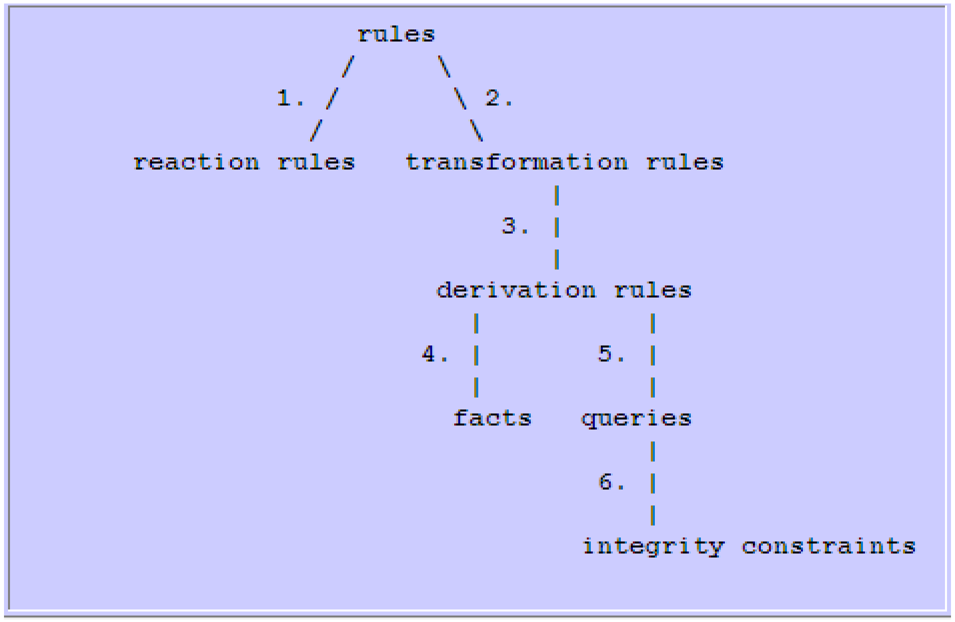


Figure 2-2 A graphical view of RuleML rules [57].

1. 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*" [58]. The reaction rule specifies the reactive behaviour of a system in response to events.
2. Transformation Rules (functional-educational rules).
3. 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"* [58].

4. Facts as in the examples *“John sells XMLBible to Mary”*,*”A Porsche is luxury”*.
5. Queries *“Give the discount amount for all customers buying any products”, to* query the rulebase for the discount amount.
6. Integrity Constraints (consistency-maintenance rules) as in the example “*A customer who rents a car must be at least 25 years old*” [58].

The Figure 2-3 below show some example of rules in version 0.7 of RuleML, this example rulebase contains four rules. The third and fourth rules are actually facts.

```
<rulebase>
<!--In English: The first rule says that a person owns an object if that person buys
the object from a merchant and the person keeps the object. -->
<if>
 <atom>
  <rel>own</rel>
  <var>person</var>
  <var>object</var>
 </atom>
 <!-- explicit 'and' -->
 <and>
  <atom>
   <rel>buy</rel>
```

```
    <var>person</var>
    <var>merchant</var>
    <var>object</var>
  </atom>
  <atom>
    <rel>keep</rel>
    <var>person</var>
    <var>object</var>
  </atom>
 </and>
</if>
<!-- In English: The next rule says that a person buys an object from a merchant if
the merchant sells the object to the person. -->
<if>
 <atom>
  <rel>buy</rel>
  <var>person</var>
  <var>merchant</var>
  <var>object</var>
 </atom>
 <atom>
  <rel>sell</rel>
  <var>merchant</var>
  <var>person</var>
  <var>object</var>
  </atom>
</if>
 <!-- The next rule is a fact that says, in English, that
John sells XMLBible to Mary. -->
 <if>
  <atom>
    <rel>sell</rel>
    <ind>John</ind>
    <ind>Mary</ind>
    <ind>XMLBible</ind>
  </atom>
  <!-- empty 'and' -->
  <and/>
</if>
<!-- The last rule is a fact that says, in English, that Mary keeps XMLBible.-->
 <if>  <atom>
    <rel>keep</rel>
    <ind>Mary</ind>
    <ind>XMLBible</ind>
  </atom>
  <and/>
</if></rulebase>
```

Figure 2-3 Example of rulebase in RuleML version 0.7 document [59]

RuleML has its key components, and its building blocks, below are some of them as [60]:

- Predicates (atoms) are n-array relations defined as <Atom> element, that include variables <Var> which will instantiated by ground values when rules are applied, and <Ind> as individual constants, and so forth.
- Derivation Rules <Implies> consist of two main parts which are body <body> and head <head>. The body part can has one or more conditions <atom> which connected by <And> or <Or>. The head part is derived from existing other rules or facts applied.

Example of the general form of RuleML 0.91 syntax is given in Figure 2-4 :

In English *"The discount for a customer buying a product is 5 percent if the customer is premium and the product is regular"* [61].

```
<Implies>
 <head>
   <Atom>
      <Rel>discount</Rel>
      <Var>customer</Var>
      <Var>product</Var>
      <Ind>5.0</Ind>
   </Atom>
 </head>
 <body>
  <And>
```

```
      <Atom>
        <Rel>premium</Rel>
        <Var>customer</Var>
      </Atom>
      <Atom>
        <Rel>regular</Rel>
        <Var>product</Var>
      </Atom>
    </And>
  </body>
</Implies>
```

Figure 2-4 Example of rule in RuleML 0.91 syntax [62].

The example in Figure 2-5 shows some of the Changes in RuleML 1.0 relative to the previous version RuleML 0.91, that <head> element is replaced with <then> and <body> element is replaced with <if>.

```
<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>
```

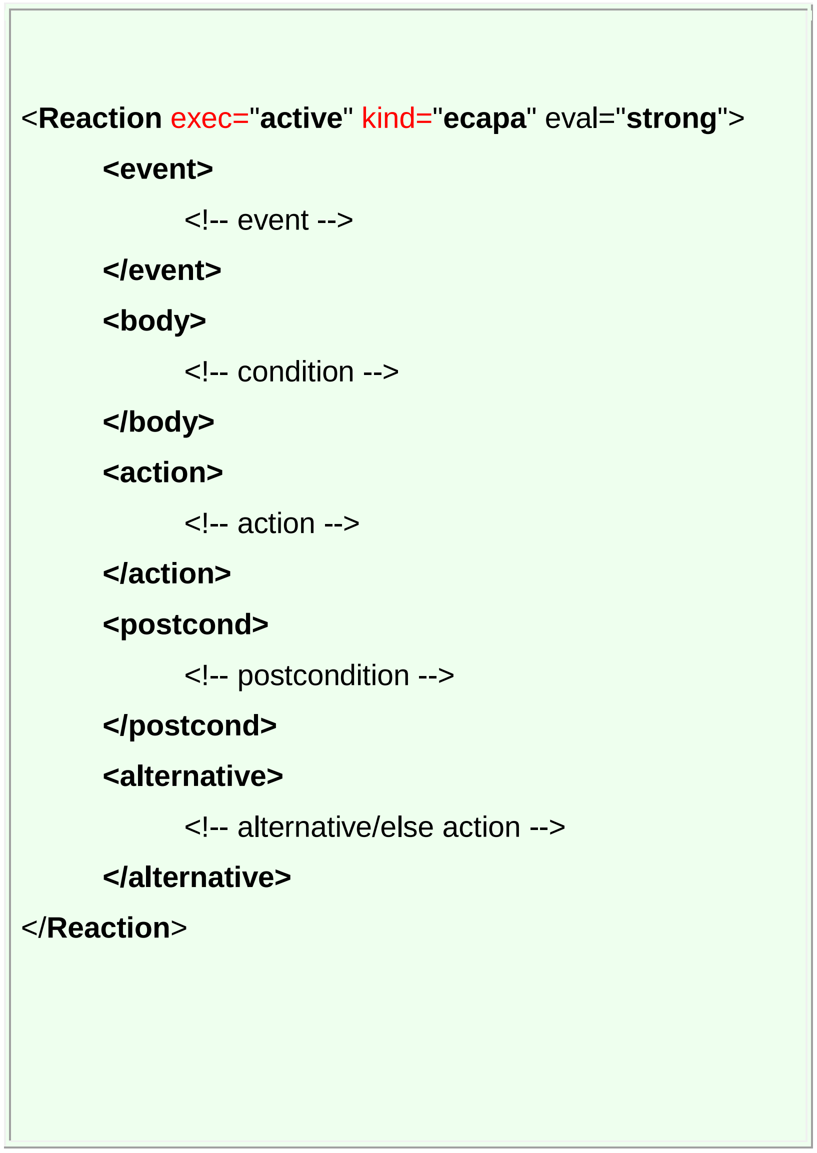
Figure 2-5 Example of rule in RuleML 1.0 syntax [63]

2.4.6.2 Reaction RuleML

Reaction RuleML (event-condition-action rules) is a branch of the RuleML family; it is described as a general language and rule interchange for the family of reaction rules [64]. Reaction RuleML introduced different types of production, action and reaction rules into the native RuleML syntax. The design of Reaction

RuleML makes it easy to learn and can be maintained faster with less risk in the opinion of [65].

The general syntax for Reaction RuleML 0.1 [66] in Figure 2-6 below:



```
<Reaction exec="active" kind="ecapa" eval="strong">
        <event>
                <!-- event -->
        </event>
        <body>
                <!-- condition -->
        </body>
        <action>
                <!-- action -->
        </action>
        <postcond>
                <!-- postcondition -->
        </postcond>
        <alternative>
                <!-- alternative/else action -->
        </alternative>
</Reaction>
```

Figure 2-6 The general syntax for Reaction RuleML 0.1[66].

In the first tag of the general syntax of Reaction RuleML 0.1 there are three attributes, they are [66]:

1. @exec (stand for execution type), this attribute contains one of the general execution styles:
   - Active: ’actively’ polls/detects occurred events by monitoring/validity time function.
   - Passive: which waits for incoming complex event message and sends outbound messages as actions which match with the defined event.
   - Reasoning: Knowledge representation derivation and event/action logic reasoning and transitions.

2. @kind attribute denotes the kind of reaction rule.
3. @eval attribute denotes the interpretation of the rule as strong or weak.

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) [67]. The general form of the Reaction RuleML0.2 syntax is shown in Figure 2-7 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>
```

Figure 2-7 The general syntax for Reaction RuleML 0.2 [67]

Furthermore, some parts are replaced and added to version 2.0 of Reaction RuleML [67], for example:

- “<Implies> has been replaced by one general <Rule>, which is used as constructor for all types of rules.
- Reaction RuleML 0.2 supports XPointer and XPath expressions as markup and query language to point into and select data from external

XML data sources and create constructive views over resource sets.” [67].

2.4.6.3 SWRL

Semantic Web Rule Language (SWRL) combines OWL (Web Ontology Language) with RuleML sublanguages of the RuleML (Rule Markup Language) [68 , 69]. OWL became a W3C recommendation in 10 February 2004 [70], it is designed for use by applications for processing the content of information instead of only presenting the information to humans.

2.4.6.4 R2ML W3C

R2ML 0.1 is an XML based rule language released in 2006, this project is about the design of integrity and derivation rules on the basis of the Rule Markup language (RuleML) and the Semantic Web Rule Language (SWRL). It defines a general markup framework for integrity rules, derivation rules, production rules and reaction rules. The current release is R2ML 0.5 which was released in August 2007 [71] .

2.4.6.5 W3C RIF

At the end of 2005, W3C chartered the Rule Interchange Format (RIF) Working Group to develop a standard for exchanging rules. It is an effort to define a standard Rule Interchange Format for facilitating the exchange of rule sets among different systems [72].

2.4.6.6 Section Summary

From the previously introduced rule languages RuleML was chosen to be used as a rulebase to store the system rules because RuleML is easy to read and understand, and also designed to be the interchange format for most Web rules in an XML format. Moreover, it works across various rule languages and platforms and it is well supported and readable by for example PHP's standard XML functions.

2.5 Chapter Summary

The aim of this chapter was to present a brief background to some key Web database technologies. The chapter started by presenting the general domain of the thesis and the database technologies. Therefore, an overview of Web technologies related to this research is presented. Finally it presents a brief description of XML and its surrounding techniques ending the last section with an overview of Rule Markup languages.

In conclusion of the previously presented technologies, PHP is chosen as server side programming language because it is:

- Open source.
- Cross platform.
- Free.
- Server scripting language.

- Allows embedding of program logic in HTML pages.
- Enables serving dynamic Web pages.
- Has a facility for parsing and accessing XML documents.
- Supports many database systems.

RuleML as rulebase to store the system rules because RuleML is:

- Easy to read and understand.
- Designed to be the interchange format of most Web rules in an XML format.
- Works across various rule languages and platforms.
- XML – based that makes it readable using PHP's standard XML functions.

HTML and JavaScript as client side programming:

- Normal Web interface.
- Allows use of forms.

PostgreSQL as DBMS:

- Free.
- Open source.
- Support for SQL standard.
- Has advanced features such as domain and composite type support.

Chapter 3

3 Literature Review

3.1 Introduction

In the literature, much work [4 , 5 , 8 , 73 , 74 , 75 , 76 , 77 , 78 , 79 , 80 , 81] has been done in different aspects of Web applications. This chapter surveys a number of academic papers and articles most strongly connected or related to the work presented in this thesis. These address many issues of Web-based applications and their solutions as moving from static Web pages to dynamic Web applications especially in terms of extracting data from databases, data exchange, user interface design, and semantic Web rules (RuleML).

This chapter is organized as follows: Section 3.2 introduces previous works related to different approaches on user interface to database which focus on automatic mechanisms to be used for generating user interfaces. Section 3.3 describes state of the art research similar to this thesis for extracting database metadata to construct Web user interfaces. Section 3.4 review some academic papers related to XML and Web applications, and gives an overview of related work on using RuleML as a rulebase and using RuleML format to save rules. Section 3.5 summarizes the chapter.

3.2 Related Work on User Interfaces to Databases

This Section gives an overview of the works related to different approaches on user interface to database which focus on automatic mechanisms to be used for generating user interfaces.

Prior to the use of ODBC and JDBC, different approaches were developed to link relational databases with Web applications [73 , 78]. For example, Nguyen et al. [73] have developed an approach that can access a database using SQL and HTML sections linked together using cross language variable substitution. This approach allows Web developers to make use of all features available in HTML and SQL for building query forms, reports, querying and updating relational databases. The cross language variable substitution bridges the gap between HTML input and SQL query as well as SQL result rows and HTML output. It was used in designing and implementing a system called DB2 WWW connection which enables the development of applications that access relational DBMS data from the Web. The end user of this system only sees the requested forms and results. The mechanism was designed and implemented as demonstrated in Figure 3-1. The disadvantage of this system was that the forms and reports were built in advance, and not in a dynamic way.

In addition the system does not use a general method of accessing the database; it is DBMS specific via DB2, however, these days many powerful tools can be used to bridge Web applications with DBMS.

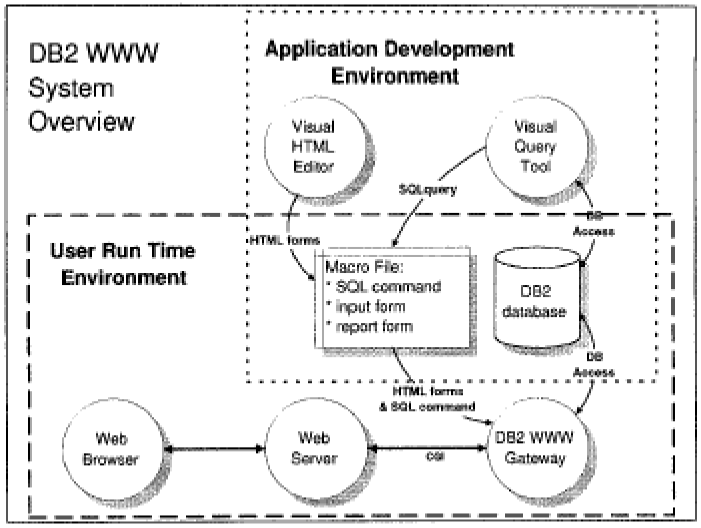


Figure 3-1 DB2 WWW System Overview adopted from [42]

An automated method for accessing relational databases from the WWW was proposed in [74 , 75]. The authors have argued that it is time consuming to reformat the information that is available in databases into HTML pages to be deployed on the WWW [74]. Thus, their proposed approach automatically generates a WWW interface to a database using the metadata available from the catalogue. The interface supports direct querying and browsing of the database based on dynamic Hyper Text links constructed from database metadata integrity constraints. This work is close to what are proposing in this thesis in terms of using database metadata. However, a remarkable difference between the two approaches is noted. The proposed system does not use a general method of accessing the metadata; it is DBMS specific via DB2. our approach is more generic since the metadata information is extracted automatically and used with the developed rules to generate the Web forms on the fly from any given database tables.

Halasz [76] presented an approach to create a template of an HTML page, which is modified by a server program before being sent to the client browser, by using APL (Array Programming Language) to minimize the amount of code and the hardware on the client machine. The approach overcomes the issue of recoding or recompiling the HTML page code by creating a template of an HTML page. The author suggested using an HTML template which contains only HTML tags, so it can be developed and maintained using APL on the server and form controls are generated and modified dynamically as required. This approach was a good idea but it has limitations of using an external representation rather than directly using the database metadata, using the metadata direct to generate Web forms is less error prone, and these days many different languages which are more commonly in use on servers can be used to bridge Web applications with DBMS rather than using APL.

3.3 Related Work on Metadata to Web Entry Forms

This section describes research in areas similar to the approach in this thesis for extracting database metadata to construct Web user interfaces.

Weiner et al. [77] describe an approach of dynamically generating Web based database interfaces. This was using a manually developed metadata table, which contains information about the database such as tables names, columns names, data types, and links between tables. In the described model since the metadata is built by hand rather than accessing existing metadata dynamically it is possible that the Web interface will not be an accurate representation of the database and it needs more effort. Figure 3-2 shows their model while the metadata table can be shown in Table 3-1. The metadata information they use is available in the database schema. So it is not the same as our approach. The proposed research generates the rules by hand since the information in the rules is not available elsewhere but the metadata is extracted automatically from the database.

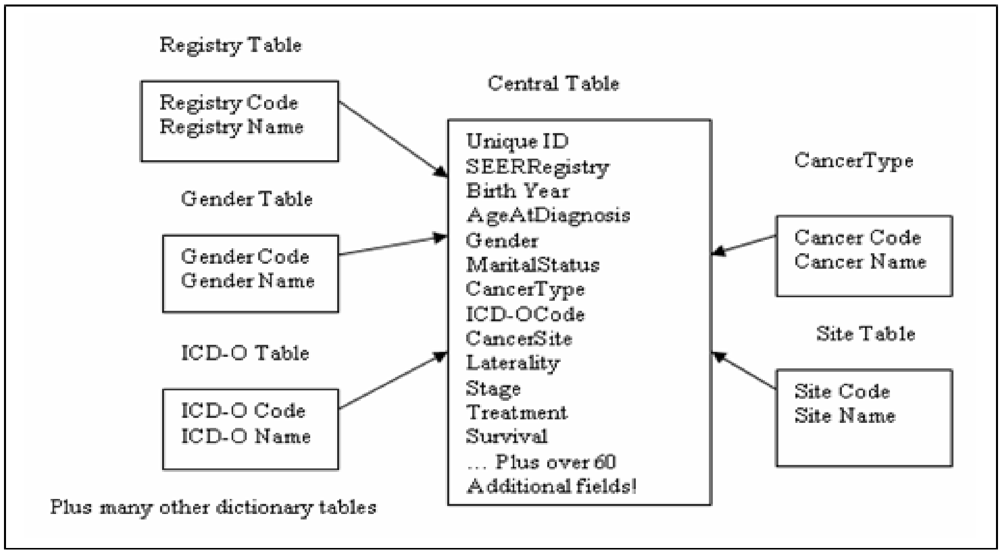


Figure 3-2 A subset of the SEER data model, adapted from [77].

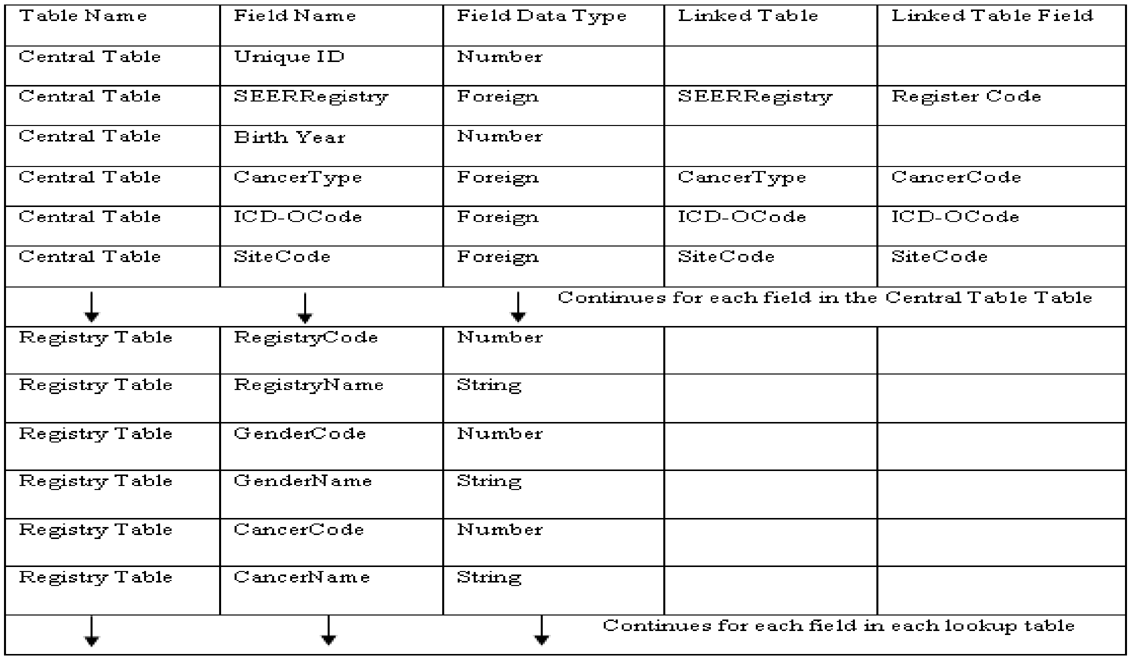


Table Name	Field Name	Field Data Type	Linked Table	Linked Table Field
Central Table	Unique ID	Number		
Central Table	SEERRegistry	Foreign	SEERRegistry	Register Code
Central Table	Birth Year	Number		
Central Table	CancerType	Foreign	CancerType	CancerCode
Central Table	ICD-OCode	Foreign	ICD-OCode	ICD-OCode
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 3-1 Metadata table that represents the SEER data model shown in Figure 3-2, adapted from [77].

Elbibas et al. in [78] proposed an approach to develop and maintain HTML forms based on metadata extracted from a database table. The authors have used Java Database Connectivity (JDBC) [82] for accessing database. It included metadata features. Their proposed approach generates dynamic HTML forms which have been generated and validated automatically. As the HTML is generated automatically on the fly, i.e., dynamic HTML, changes that are made to the database are reflected once the data is accessed again. Java and metadata were used to show help messages to the user to validate the input data. The set of rules of this scheme is embedded in the application code where it is difficult to locate and change their logic. In addition the set of rules does not support the manipulation of semantics of database metadata in some cases. So it is possible to develop domain specific rules to support the generic rules, as an example to deal with column names.

Elsheh et al. in [4] 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. Although XML is used it differs from our approach which is using RuleML. This approach has the same problems encountered by [78] where the set of rules of this scheme is embedded in the application code where it is difficult to locate and change their logic. In addition the set of rules does not support the manipulation of semantics of database metadata in some cases. So it is possible to develop domain specific rules to support the generic rules, as example to deal with columns name. In our framework the separation between the logic and presentation is achieved.

Mgheder et al. in [75] 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.

3.3.1 Section summary

From the previous approaches we summarise some points which will be taken to make our approach as generic and abstract as possible:

- Dynamic metadata VS hand created.
- Range of languages used.
- Specific to one DBMS.
- Separation of rules out of code.

3.4 Related Work on XML and Web-based Data Entry Applications, and RuleML

This section gives an overview of some academic papers related to XML and Web applications, and gives an overview of related work on using RuleML as a rulebase and using RuleML format to save rules.

XML Schema uses elements and attributes to express semantics of XML data, but XML Schema does not have active elements. Bernauer et al. in [79] proposed an approach which implemented an Active XML Schema with XML Schema that defines active behaviour to enrich XML documents. Active XML Schema specifies active behaviour by using Event-Condition-Action rules, which automatically performs an action as reaction if a given condition applied. They do not use RuleML.

Kirda et al. in [80] implemented a system to build adaptable database interface using XML/XSL and WebCUS (Web Content Update System) as in Figure 3-3. The system stores the Database schema in an external XML file to define the EER(Extended Entity Relationship) [83]. So the XML files described the database schema and the access control rules. The XML files use a special syntax to describe the tables, rows, columns and the table's relationship. The information has to be manually converted into WebCUS XML database schema description (XML-EER), and every time the database schema has to be modified manually. The system uses XSLT stylesheets to separate the layout of the updated system from the code, which is used by MyXML template engine to transform the MyXML documents into Web forms.

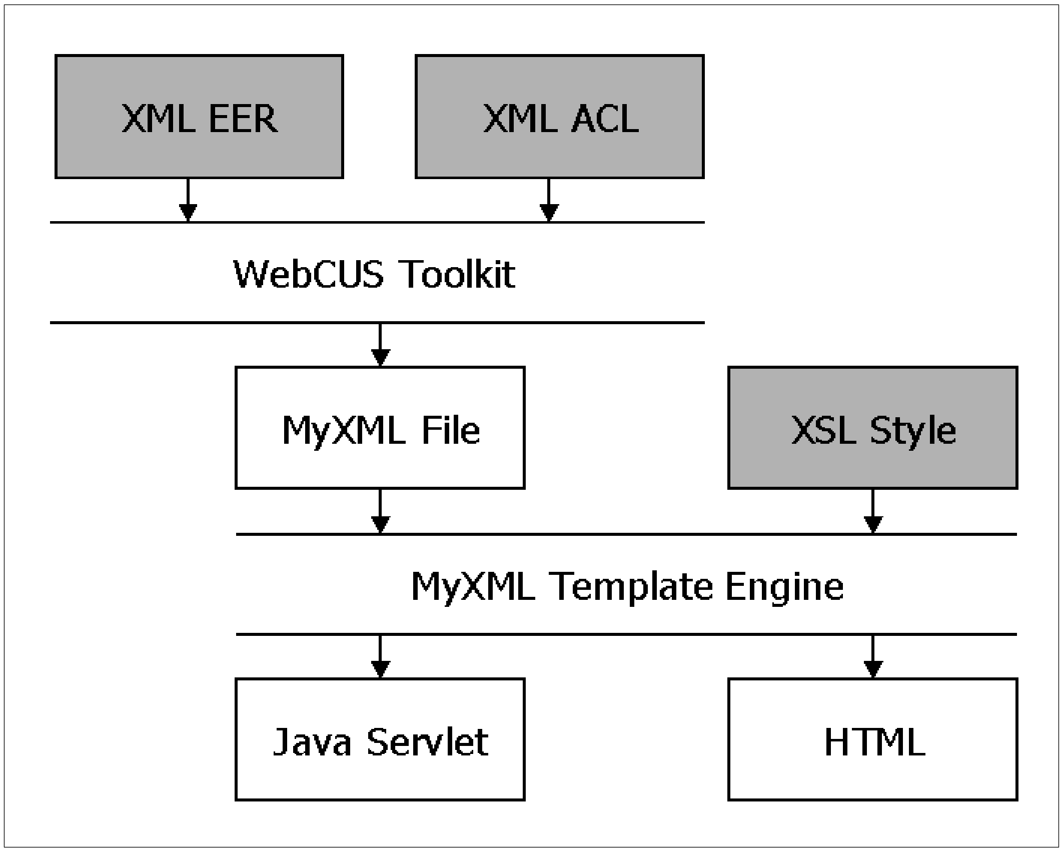


Figure 3-3 The WebCUS Architecture

Turau in [5] describes a framework that introduces a method for Web-based data entry applications based on a textual specification of XML application forms. It focuses on the presentation design by separating between presentation and business logic. The author implemented a three-tier framework called Wizard for Web based data entry application using Java Servlets and Java Server Pages to solve the separation between business rules and user interface presentation code. This used a single XML file to save the formal specification, which is used as input for a code generator to generate a system prototype. So it can be used for testing data entry process and a user interface was established. The generated views have its default design appearance.

Bertossi et al. in [8] describe a methodology that uses metadata for a virtual and relational data integration system. They used a standard format based on XML and RuleML for representing metadata. Native XML was used to represent data about the schemas, RuleML was used to represent the mapping between the global schema and the local schemas, and XQuery was used to query the metadata. This design allows data sources to be added to the system or removed without affecting any other data sources. As a conclusion, this approach is similar to ours in using RuleML to store metadata.

3.4.1 RuleML as Rulebase

RuleML provides a format for what is claimed [81] to be a natural form for human reasoning and behaviour, that is if-then-rules. However the individual rules need to be developed into a Rulebase, in a different domain to ours Schmidberger et al. in [81] have mentioned that there is no established standard rule format for industrial plant information reasoning available. They described an approach which implements rulebase engineering of automation systems. The system was created especially for the automatic instantiation of Asset Management Functionalities and the automatic creation of interlocking control code. They have used a rule format based on a combination of RuleML and MathML elements in the logic part. Thus, in the context of rulebase automation of plant engineering tasks there will be a need for common description of such rules in a format which is understood by humans and can be interpreted by a computer.

3.5 Chapter Summary

The chapter has introduced an overview of previous work related to the work presented in this thesis. It has provided a wide range of techniques and ideas related to Web development, which is the central topic in this thesis. The chapter has been divided into three main sections to organise the overview. It started with an overview of the work that has been done on the topic of user interfaces and databases, then moving to some ideas related to user interfaces and metadata, in the third section an overview of some academic papers related to XML, XML Schema and Web applications has been addressed, and different ideas of using RuleML as rulebase introduced. By reviewing different techniques that were proposed of using database metadata to improve the automatic and dynamic generation of Web entry forms; we summarise some points which will be taken into consideration to make our approach as generic and abstract as possible:

- In the past external representation was used rather using database metadata.
- The database metadata was manually created instead of being extracted dynamically from database.
- Some approaches used were specific to one DBMS; our aim is to create an application that can be adaptable to various DBMS.
- The rules were embedded in the application code whereas our approach aims to separate rules out of application code.
- Rules do not support the manipulation of semantics of database metadata in some cases, our approach can tackle this problem in two ways, first develop rulebases to support the common sense rules, for as example to deal with column name, second develop

domain specific rulebases to support the common sense rules as example to deal with column name and size.

In the next four chapters, different techniques are proposed that use RuleML and database metadata to improve the automatic and dynamic generation of Web entry forms.

Chapter 4

4 Using RuleML and Database Metadata for Generating Automatic and Dynamic Web Entry Forms.

4.1 Introduction

This chapter introduces a prototype development system which aims to test the use of the RuleML format to support the development of automated Web interfaces. The RuleML stores sets of rules to overcome database metadata limitations see Section 4.4.3 and use them to generate automatic and dynamic Web forms. The system is not bound to any platform and could be implemented in a variety of languages. Here we have implemented this in PHP as an example of a language used for Web development in a number of styles and often in an ad-hoc and unstructured style. This chapter begins with an overview of the proposed prototype. Then, building RuleML metadata rulebase, based on metadata rules extracted from the database catalogue, is introduced. The implementation of the proposed prototype is then discussed with an example that shows how the proposed approach works.

4.2 Prototype Overview

The principle idea of the prototype implementation was the creation of a Web form to evaluate to what extent we can use the relational database metadata by building the rulebase using the RuleML format to store a developed set of rules which will be discussed in section 4.4.1. The rulebase will be used as an abstract representation that can be used to build adaptable dynamic database interfaces and to produce different Web entry forms. The metadata will be extracted from system catalogue tables as typically found in relational database systems. In this case using a number of PHP's PostgreSQL functions at runtime in conjunction with the rulebase to produce appropriate Web form elements. Details of how the proposed prototype works are presented in the following sections.

4.3 General Metadata Rules

Columns in the database table have properties such as data type and column name. The properties are the metadata of the table. In practice only the required pieces of information extracted from the database metadata will be used for producing dynamic Web forms. From these metadata the required rules are described as following:

- Rules based on type definition of columns. For example
    - Column is serial or not.

  - Column type is Boolean, character or string, integer
  - Column size.
- Rules based on uniqueness. For example
  - Column is a primary key.
  - Column is a foreign key.
- Rules based on null ability (not null) to ensure all rows in the table contain a definite value for the columns specified as not null.
- Rules based on columns' names. For example:
  - Column name is password or variant e.g. password.

4.4 The Prototype Implementation

The prototype implementation consists of several processes as shown in Figure 4-1. The proposed prototype aims to achieve the following objectives:

- A connection to a database management system is created using a number of PHP's PostgreSQL functions.

- Extract metadata using specific functions to retrieve information about each field in a database table. A relational database provides access to its structure through the same tools that are used to access the data, specific PostgreSQL functions can be used as a tool. In practice not all extracted information will be used for producing dynamic Web forms but only the required pieces of information will be used for this purpose such as data type, null or not null fields, primary and foreign keys.

- Apply rulebase. The concept of mapping each table's column to a specific Web entry form element is based on a set of rules. For developing automatic and dynamic Web forms, a rulebase based on RuleML format was developed as shown in Figure 4-3. This proposed rulebase works by taking advantage of database metadata. This rulebase will be applied in conjunction with the metadata of each column to decide which form element will be created for each column.

- Generate Web form element. The generated Web form is returned back to the client so the user can fill in the required information. In the Web form many controls that have constraints are checked to make sure that the correct information is entered. This is important to avoid any missing fields.

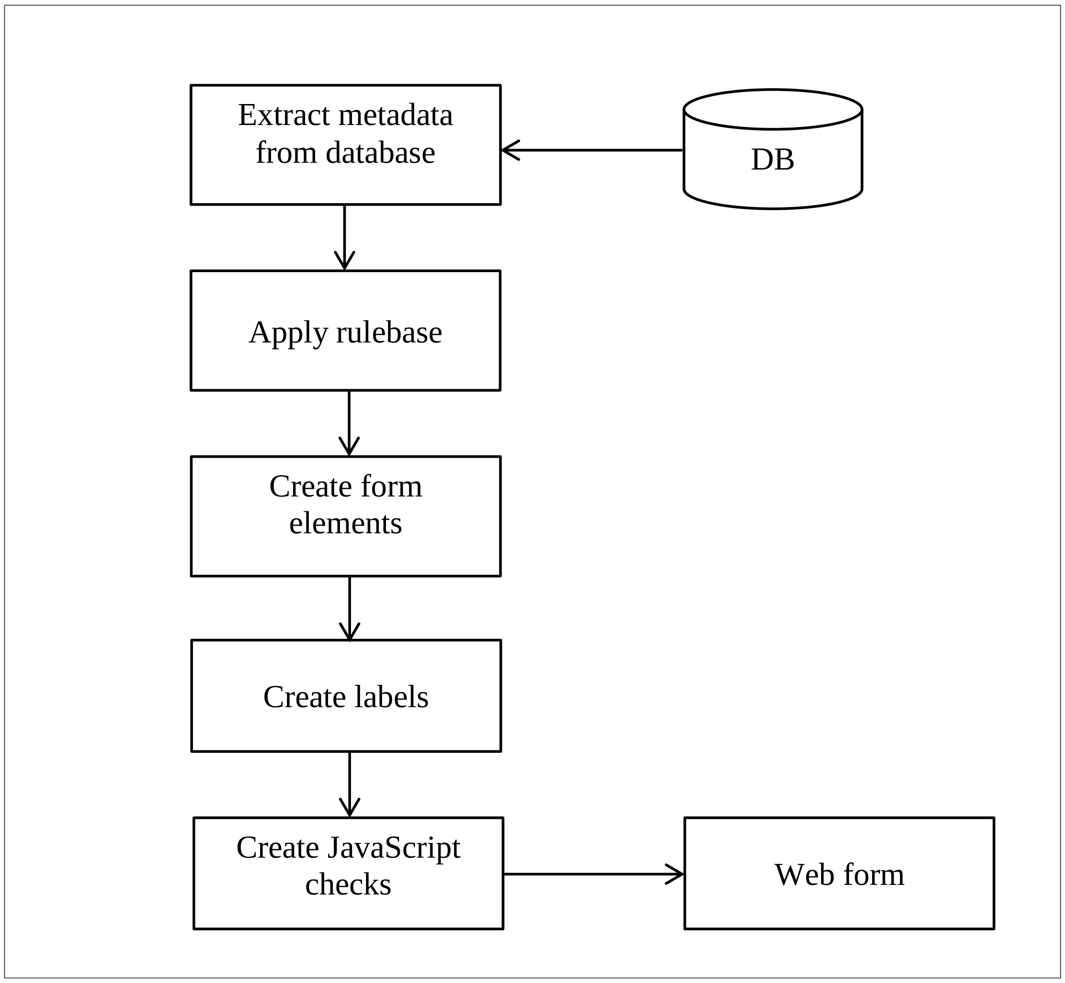


Figure 4-1 Prototype Implementation

4.4.1 Building RuleML metadata rulebase

In this section we address some common sense rules based on the information that exists in the database metadata. Any Web form consists of many form elements such as (text box, text area, drop down list, check list, radio buttons). By taking advantage of database metadata available a set of common sense rules can be addressed. These rules will be used in conjunction with the database metadata to generate a Web form dynamically on the fly. It has the advantage of avoiding the need to hard code the presentation of the Web form following any changes to the database tables or data type of each column in the database. The following common sense rules are used to build the rulebase in RuleML format and describe how to generate elements of a form automatically, as shown in Figure 4-3.

- 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 it's 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 it's length is more than 30, then it should be mapped to textarea Web control form.
- Rule 4: if a column is Boolean type, then it can be implemented as a group of radio buttons or drop down menu. So if the column is Boolean and it is not null then in this case it is pair of radio buttons, but if the column is Boolean and it is nullable then in this case it is a group of radio buttons. In some cases a default value is generated automatically which

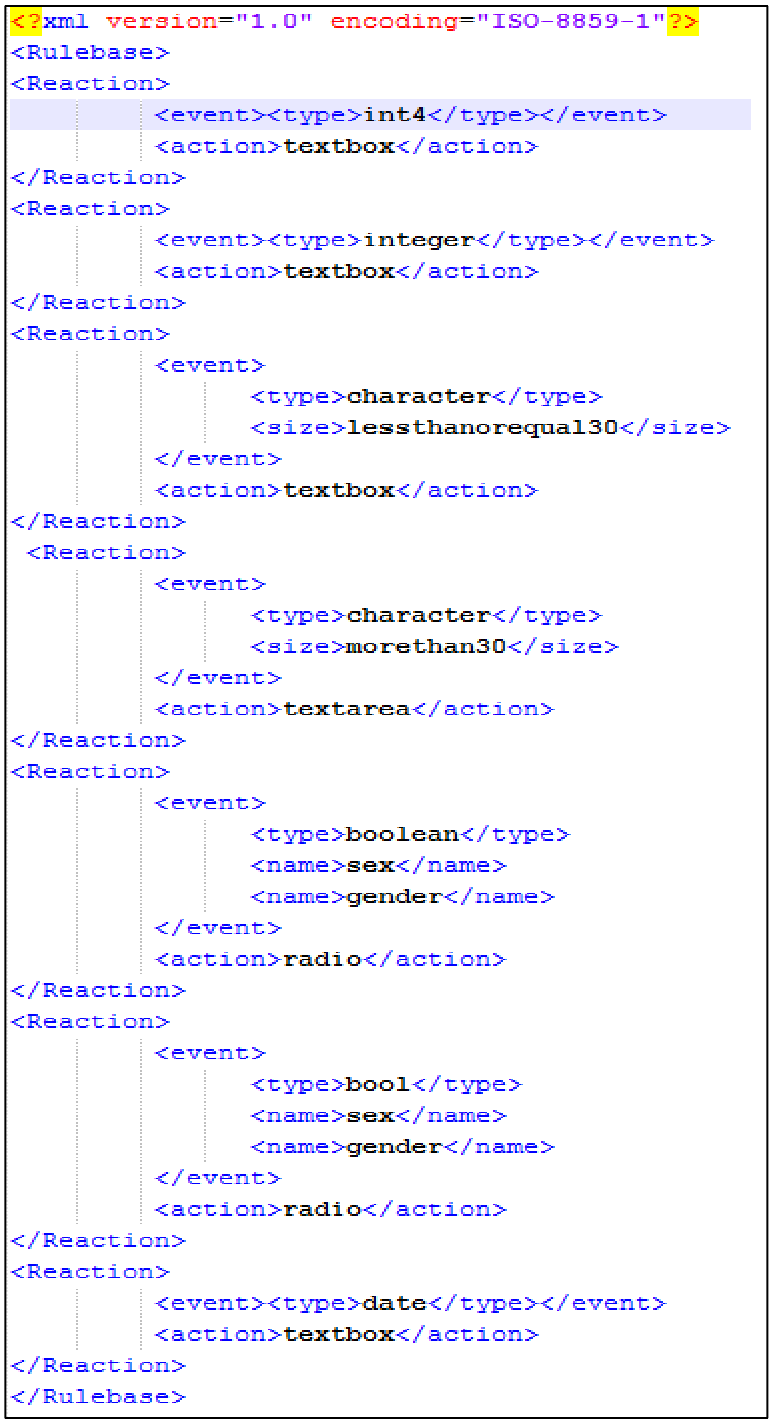
means one of the radio buttons is pre-selected by the system unless the user has chosen another one. In all these cases prototype framework implementation for radio buttons will be used. In addition in this rule we can use the name of the column to make the Web form more clear.

- Rule5: if a column is date type, then it could be mapped to a textbox and the format of the date provided as a label for this element.

The condition on the length of the field in Rule2 and Rule3 could be set at different threshold values and it could be changed by allowing it to be set as a parameter. Figure 4-2 shows the algorithm of the above developed rules.

```
BEGIN
    READ column_type, column_size, column_is_null_able
    SET  n   // value of size condition for the textbox
    CASE column_type is:
        Integer:      Action= create textbox element
        Character:    IF column_size <= n THEN
                          Action= create textbox element
                      ELSE
                          Action = create textarea element
                      ENDIF
        Boolean:      IF column_is_null_able THEN
                          Action = create group of radio buttons element
                      ELSE
                          Action = create pair of radio buttons element
                      ENDIF
        Date:         Action = create textbox element
    ENDCASE
END
```

Figure 4-2 Pseudo code for common sense rules



```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Rulebase>
<Reaction>
        <event><type>int4</type></event>
        <action>textbox</action>
</Reaction>
<Reaction>
        <event><type>integer</type></event>
        <action>textbox</action>
</Reaction>
<Reaction>
        <event>
              <type>character</type>
              <size>lessthanorequal30</size>
        </event>
        <action>textbox</action>
</Reaction>
 <Reaction>
        <event>
              <type>character</type>
              <size>morethan30</size>
        </event>
        <action>textarea</action>
</Reaction>
<Reaction>
        <event>
              <type>boolean</type>
              <name>sex</name>
              <name>gender</name>
        </event>
        <action>radio</action>
</Reaction>
<Reaction>
        <event>
              <type>bool</type>
              <name>sex</name>
              <name>gender</name>
        </event>
        <action>radio</action>
</Reaction>
<Reaction>
        <event><type>date</type></event>
        <action>textbox</action>
</Reaction>
</Rulebase>
```

Figure 4-3 Metadata rulebase in RuleML format

4.4.2 Retrieving metadata from database

After building a rulebase in a RuleML format the next step is to use any database table that contain different integrity constraint rules in conjunction with the rulebase to build a Web form automatically. The implementation started by creating a database table which contains employee information as shown in Figure 4-4 below:

```
CREATE TABLE employee (
id_no integer NOT NULL,
name character (30) NOT NULL,
date_of_birth date NOT NULL,
full_time Boolean,
address character (35),
CONSTRAINT  id_no PRIMARY KEY (id_no));
```

Figure 4-4 Database table’s structure

A number of PHP's PostgreSQL functions are used to make a connection to the database and retrieve the metadata; some of these functions are listed below:

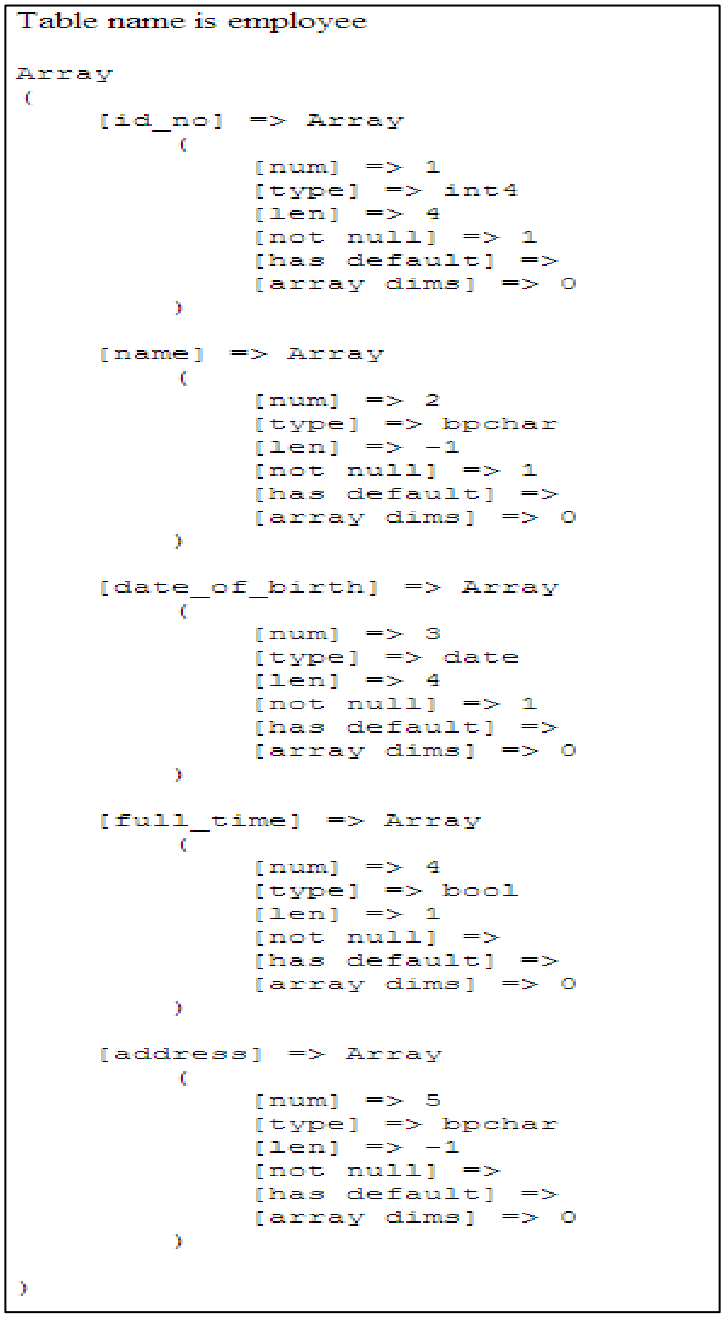
- To make a connection to the database we use the function '*pg_connect*' as follows:

$conn = pg_connect("host=localhost port=5432 dbname=postgres user=postgres password=atiaalbabah");

Where '*localhost*' is the server name, '*postgres*' is database name, second '*postgres*' is the name of the user and '*atiaalbabah*' is the password used to access the database.

- To get the column metadata information as an array as shown in Figure 4-5 we use the function '*pg_meta_data*' as follows: *$meta* = *pg_meta_data($conn,'employee');*

Where '*$conn*' is the connection handle and '*employee*' is the table's name.



```
Table name is employee

Array
(
    [id_no] => Array
        (
            [num] => 1
            [type] => int4
            [len] => 4
            [not null] => 1
            [has default] =>
            [array dims] => 0
        )

    [name] => Array
        (
            [num] => 2
            [type] => bpchar
            [len] => -1
            [not null] => 1
            [has default] =>
            [array dims] => 0
        )

    [date_of_birth] => Array
        (
            [num] => 3
            [type] => date
            [len] => 4
            [not null] => 1
            [has default] =>
            [array dims] => 0
        )

    [full_time] => Array
        (
            [num] => 4
            [type] => bool
            [len] => 1
            [not null] =>
            [has default] =>
            [array dims] => 0
        )

    [address] => Array
        (
            [num] => 5
            [type] => bpchar
            [len] => -1
            [not null] =>
            [has default] =>
            [array dims] => 0
        )

)
```

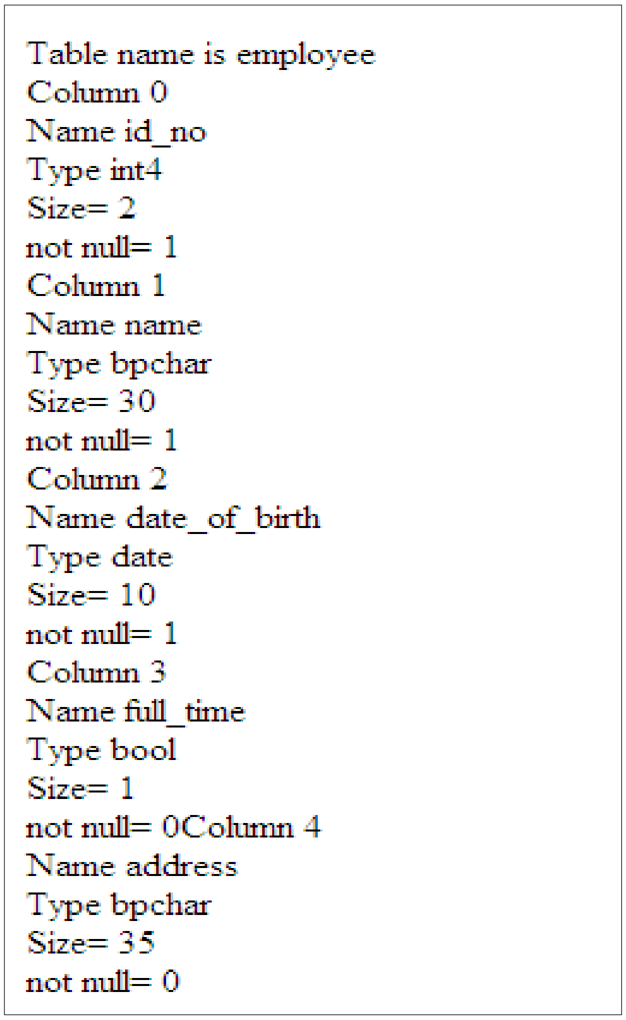
Figure 4-5 Metadata as an array extracted from database table

- There is another method to retrieve information about each table field which is used in the prototype implementation to get the specific information needed for the form implementation such as column's name, type and so on [84], by using specific functions as explained below in Table 4-1. The output generated from these functions is shown in Figure 4-6.

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

Table 4-1 Methods to retrieve information about each field

There are a number of different technologies available depending on the DBMS and the implementation language; we illustrate two methods in the language and DBMS to use for the prototype. Equivalent functionality is available in other situations; with some variations e.g. some systems may provide additional information.



```
Table name is employee
Column 0
Name id_no
Type int4
Size= 2
not null= 1
Column 1
Name name
Type bpchar
Size= 30
not null= 1
Column 2
Name date_of_birth
Type date
Size= 10
not null= 1
Column 3
Name full_time
Type bool
Size= 1
not null= 0Column 4
Name address
Type bpchar
Size= 35
not null= 0
```

Figure 4-6 Metadata from database table using direct PHP PostgreSQL functions

- Another method to retrieve information about some fields in a table to get the specific information needed for the form implementation, it is to query the database table metadata as show in the example in Figure 4-7.

```
$query = "SELECT   column_name, ordinal_position, is_nullable, data_type,

      character_maximum_length, constraint_name, constraint_type,
cons_description  FROM postgress_metadata where table_name =
'employee'";

$result = pg_query($query);

       if (!$result) {

           echo "Problem with query " . $query . "<br/>";

           echo pg_last_error();

           exit();    }    $m=0;

       while($myrow = pg_fetch_assoc($result)) {

       if ($myrow['ordinal_position']!=$m)  {

            $name= $myrow['column_name'];

            $Metacoltype=$myrow['data_type'] ;

             $Metacolsize=$myrow['character_maximum_length'];

             $colnotnull= $myrow['is_nullable'];

             $constraint_type=$myrow['constraint_type'];

            $constraint=$myrow['cons_description'];
```

Figure 4-7 Query specific information form database metadata

4.4.3 Limitation of database metadata

The only piece of information that tells us that a particular column is a password is its name; there is no distinctive database type of password, unlike the situation with a Web form where a distinct password type exists.

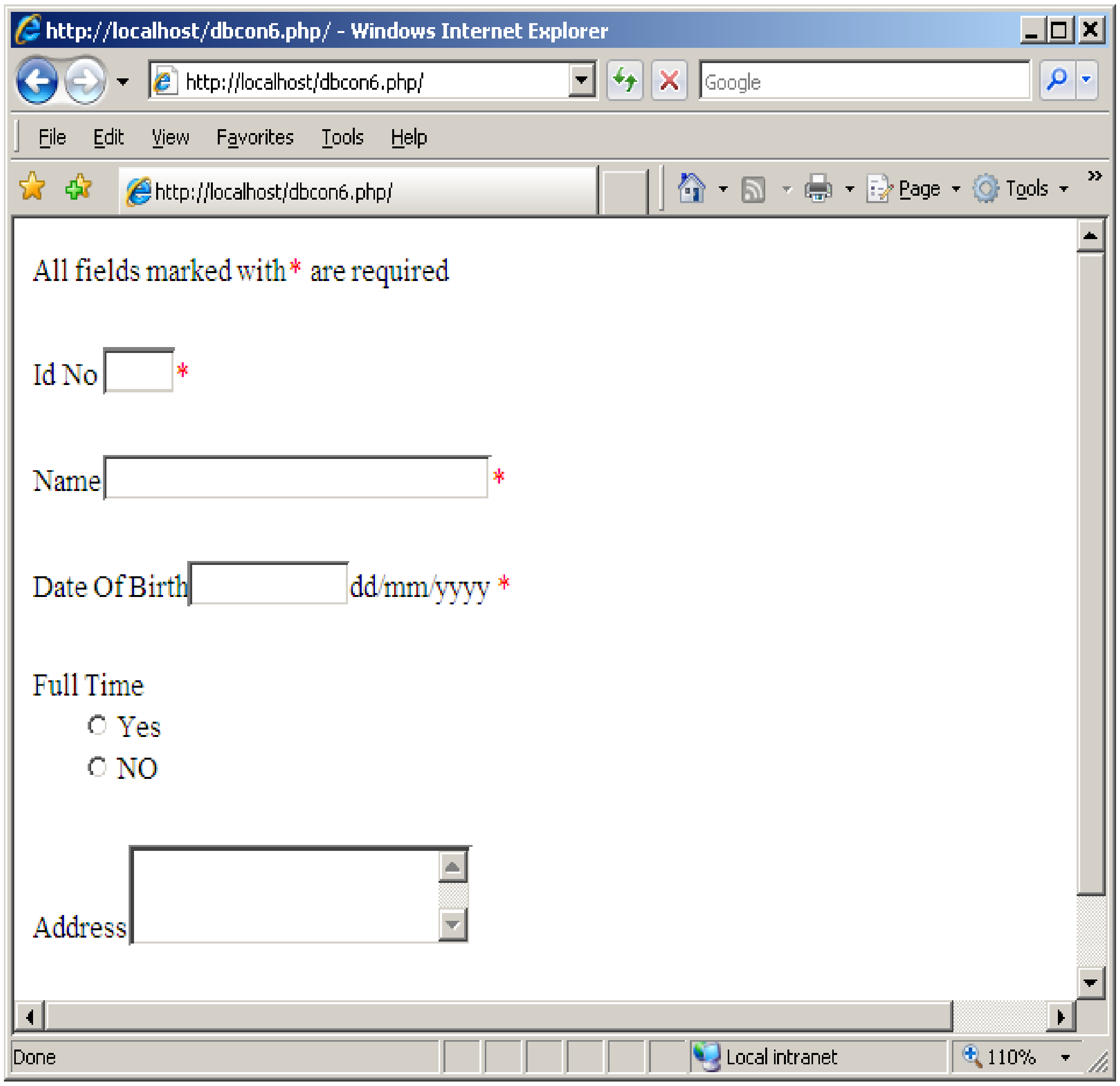
Numerous other examples exist where the type systems of typical RDBMSs are arguably not semantically rich enough.

- If a column is called email we might infer that its structure should be in the form someone@something.something but it will simply be “text”.
- Some RDBMS may support a composite type connecting several columns together but for many the only association of a number of columns as parts of an address may be naming conventions like calling the columns st_address, city_address, etc.

So to use this information to generate the correct Web form element for each column the metadata alone cannot be used, it needs some supporting rules which will help to map each column to the right form element. Therefore we can invoke a suitable RuleML, and store all names as domains which will help to map each column to the correct element control.

4.4.4 Generate the Web entry forms

A general purpose PHP script was written which loops through all the metadata for each column and uses the RuleML rulebase, the rulebase is comprehensive and has a sensible and complete order. So every column will map to something, from a rich set of features if it's a primary key column down to a plain form element for a nullable text field. The script tests to see which rules apply and then uses those rules to build the form elements on the fly as shown in Figure 4-8 where for example the column id_no 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 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. In Figure 4-9 we use functions to help the user entering the date field.


http://localhost/dbcon6.php/ - Windows Internet Explorer
http://localhost/dbcon6.php/
Google
File Edit View Favorites Tools Help
http://localhost/dbcon6.php/
Page Tools
Done
Local intranet
110%

All fields marked with * are required

Id No *

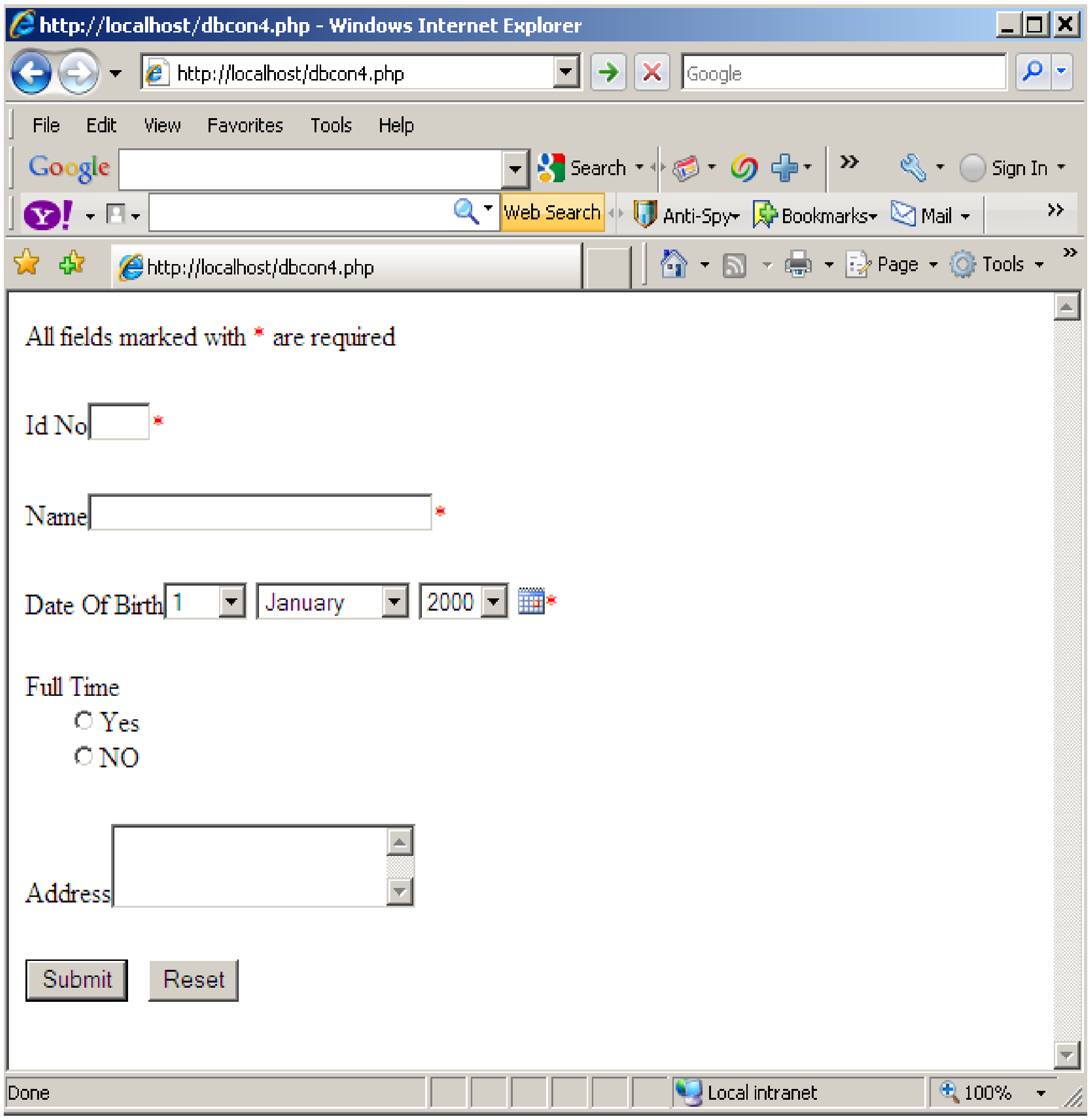
Name *

Date Of Birth dd/mm/yyyy *

Full Time
○ Yes
○ NO

Address

Figure 4-8 Screenshot showing user interface form generated automatically using metadata and RuleML.


http://localhost/dbcon4.php - Windows Internet Explorer
http://localhost/dbcon4.php
Google
File Edit View Favorites Tools Help
Search
Sign In
Web Search
Anti-Spy
Bookmarks
Mail
Page
Tools
Done
Local intranet
100%

All fields marked with * are required

Id No *

Name *

Date Of Birth 1 January 2000 *

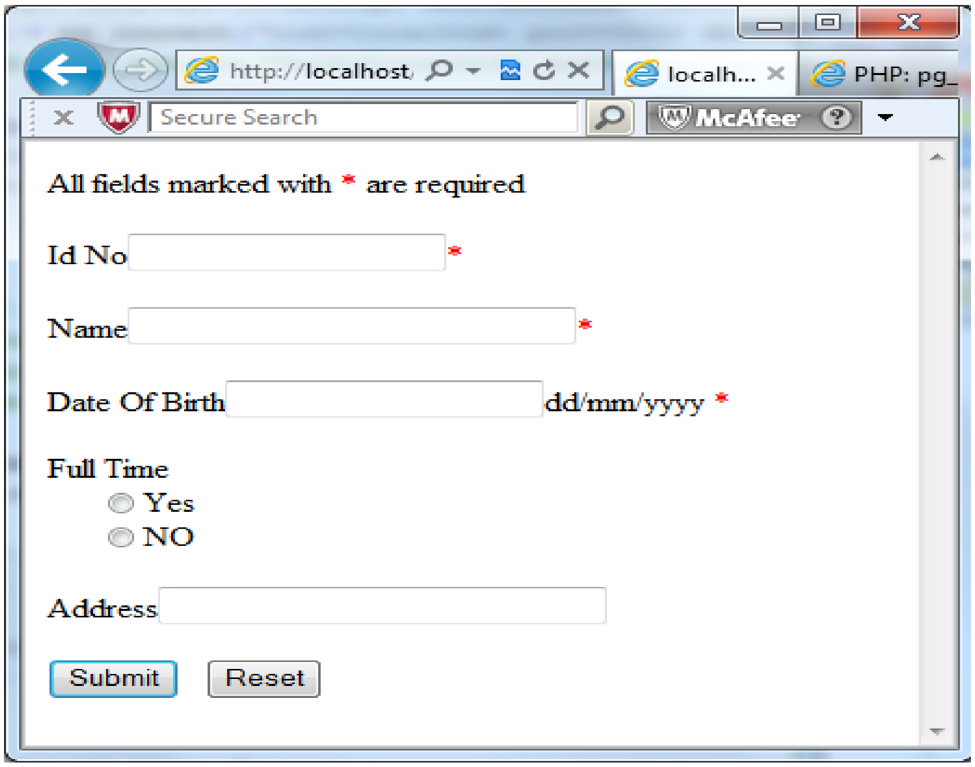
Full Time
Yes
NO

Address

Submit Reset

Figure 4-9 Screenshot showing user interface form generated automatically, and function used to generate a date field.

The written script is general. Changes to the metadata lead to changes in the generated form. As in Section 4.4.1, Rule2 and Rule3 control the generation of columns which are mapped to textbox or textarea depending on the column length metadata. So any changes applied to the table metadata are applied to the form automatically, for example when the column metadata length is changed to a length less than or equal to the conditional length in the rules, the mapping of the address column is changed automatically from a textarea to a textbox control element as shown in Figure 4-10.



All fields marked with * are required

Id No *

Name *

Date Of Birth dd/mm/yyyy *

Full Time

Yes

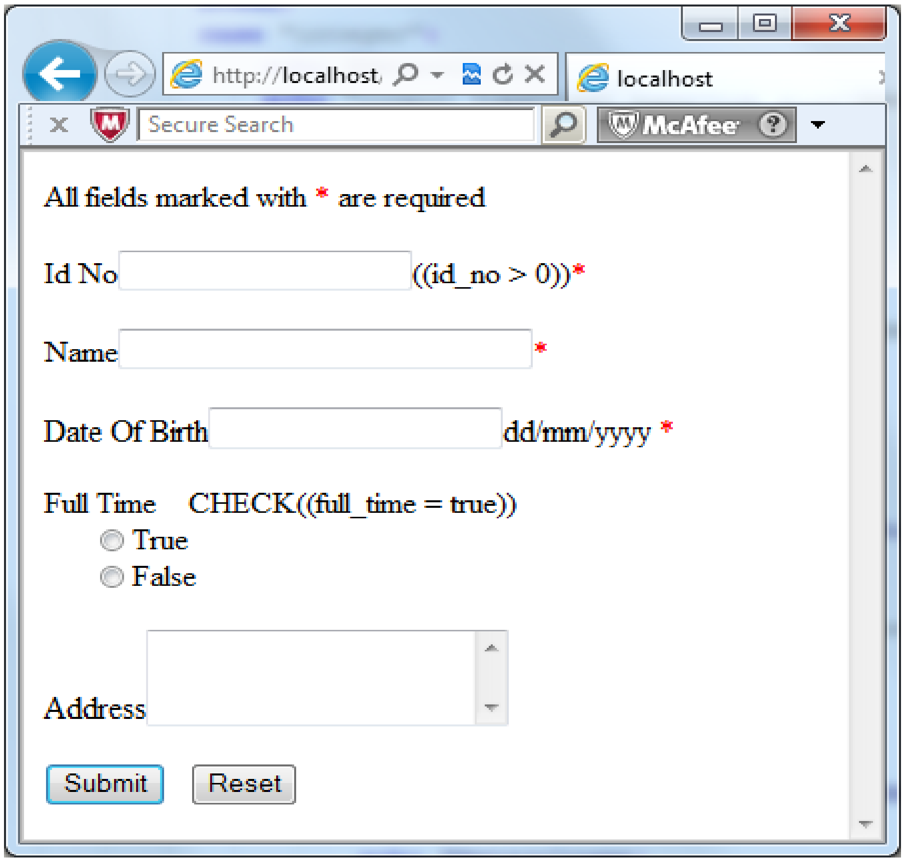
NO

Address

Figure 4-10 Screenshot showing the changes of address control element automatically generated.

Figure 4-11 presents another method to retrieve column's constraints from database metadata and shows it to the user to add more semantic to the Web forms, by using the following query in this case the table name is employee:

$query = "SELECT column_name, ordinal_position, is_nullable, data_type, character_maximum_length, constraint_name, constraint_type, cons_description FROM postgress_metadata where table_name = 'employee'";



All fields marked with * are required

Id No ((id_no > 0))*

Name *

Date Of Birth dd/mm/yyyy *

Full Time CHECK((full_time = true))

True

False

Address

Figure 4-11 Screenshot showing constraint type of some column automatically generated.

The additional text for some fields such id_no>0 is obtained from the cons_description metadata element.

4.5 Chapter Summary

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 chapter we proposed an approach which separates the rules as an independent entity from the application code, by using a RuleML format as rulebase. The system evaluation was successfully carried out using Reaction RuleML0.1 format to store 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 use as generic rulebase as possible using RuleML.

In the next chapter we propose to investigate the use of Reaction RuleML 0.2 as an improved version of RuleML 0.1. In Reaction RuleML 0.2 more detailed tags are provided with more meaningful tags naming. Another extension to our work is to develop two kinds of rules: a more complete set of common sense rules and domain-specific rules.

Chapter 5

5 A Rule Framework Design

5.1 Introduction

This chapter introduces our suggested framework implementation. We aim to design a more general framework that includes as many rules as possible in the system. The aim is to extend the automation of Web forms so that more semantic information is used in a consistent fashion. So we investigate the use of Reaction RuleML 0.2 which is a development from original 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 accessing 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 of any set of forms that interact with the same database. Use of common sense rules and domain specific rules rulebases using Reaction RuleML 0.2 format in conjunction with database metadata rules can be used to support the development of automated Web forms. The aim is to extend the automation of Web forms so that more semantic information is used in a consistent fashion. We illustrate our approach with the development of a banking based example later.

5.2 **Framework Overview**

In this chapter, we develop the initial use of Reaction RuleML 0.2 in conjunction with database metadata originally proposed in [85] to a more general framework that allows "common sense" and "domain specific" rules to be included in the system.

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 [4]. In this category rules are like those mapping a column called password to a password type form input.

As a general example of domain specific rules we offer mapping the column of landline telephone number to two separate textboxes control elements; the first one is for the area code and the second one is for the telephone number. Here we use our semantic knowledge of the structure of landline phone numbers to improve the user experience and avoid possible errors.

Common sense rules use domain specific rules for some advanced information, as the common sense rules use column type (found from database metadata) and domain specific rules support the common sense rules by providing the exact size (found from domain specific rulebase) for specific columns using column name (found from database metadata). Domain specific rules use the name and the size of database column to support mapping each column to a Web form control element.

We illustrate the limitations of database metadata alone with the following car registration example. For a single 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 HTML form 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 Figure 5-1 where we supplement database metadata rules with common sense rules and introduce a second rulebase of domain specific rules.

Reaction RuleML 0.2 format is used to store metadata rules as rulebase, In addition, we propose a framework as in Figure 5-1 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.

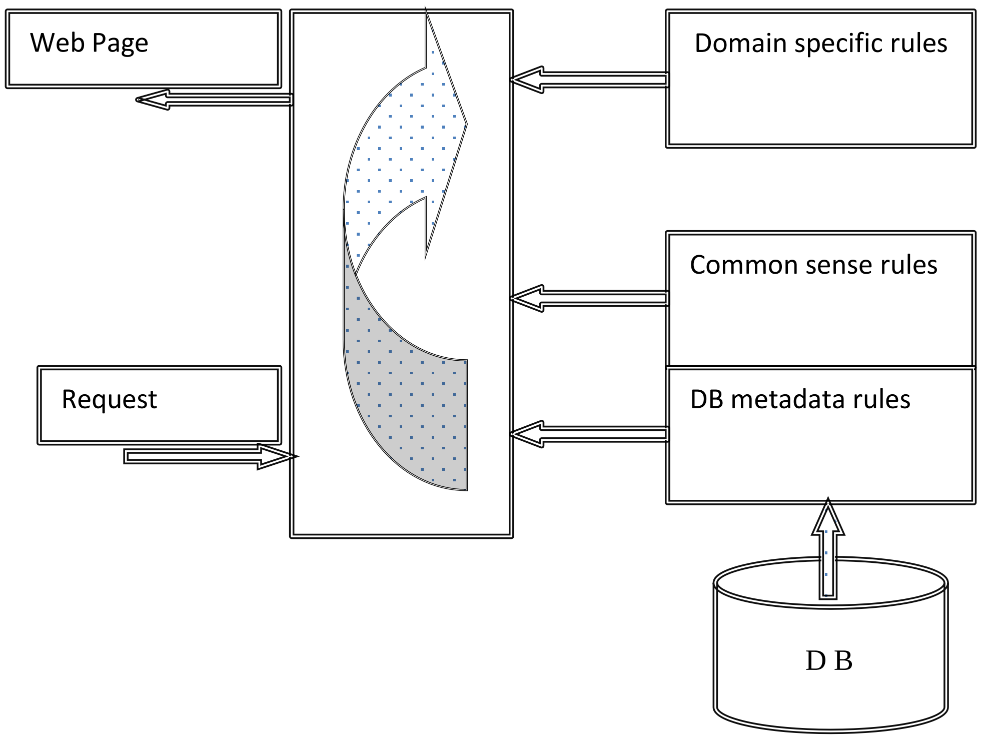


Figure 5-1 Structure of the proposed framework

5.3 Framework Implementation and Evaluation

This section introduces a prototype development system which aims to implement and test the framework, introduced in Section 5.2, of using the Reaction RuleML 0.2 format. It stores and implements constraint rules to overcome the limitations of only using the database metadata information. These rules are used with the metadata to generate automatic and dynamic Web forms.

The general idea of the prototype implementation is to create a Web form to evaluate to what extent we can use the relational database metadata and build the rulebase 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. Both types are used to build adaptable dynamic database interfaces. The metadata are extracted using a number of PHP's PostgreSQL functions. The following sub-sections present a flow diagram and implementation of the proposed framework.

5.3.1 Framework mechanism

The steps that are required to use rules and metadata for generating automatic Web forms are shown in Figure 5-2, which extends the original prototype implementation discussed in chapter 4 Figure 4-1. The following objectives are intended to be achieved.

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

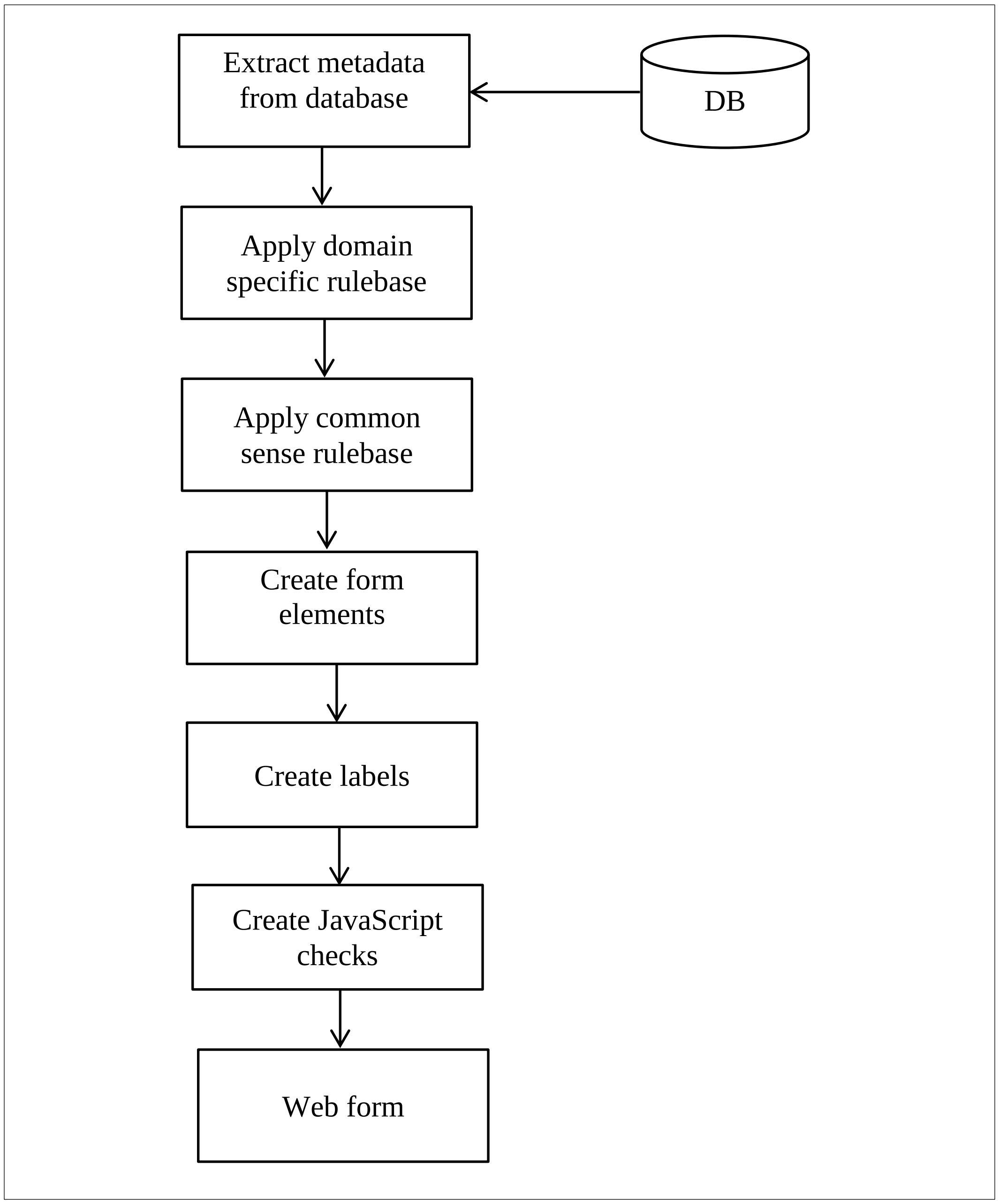
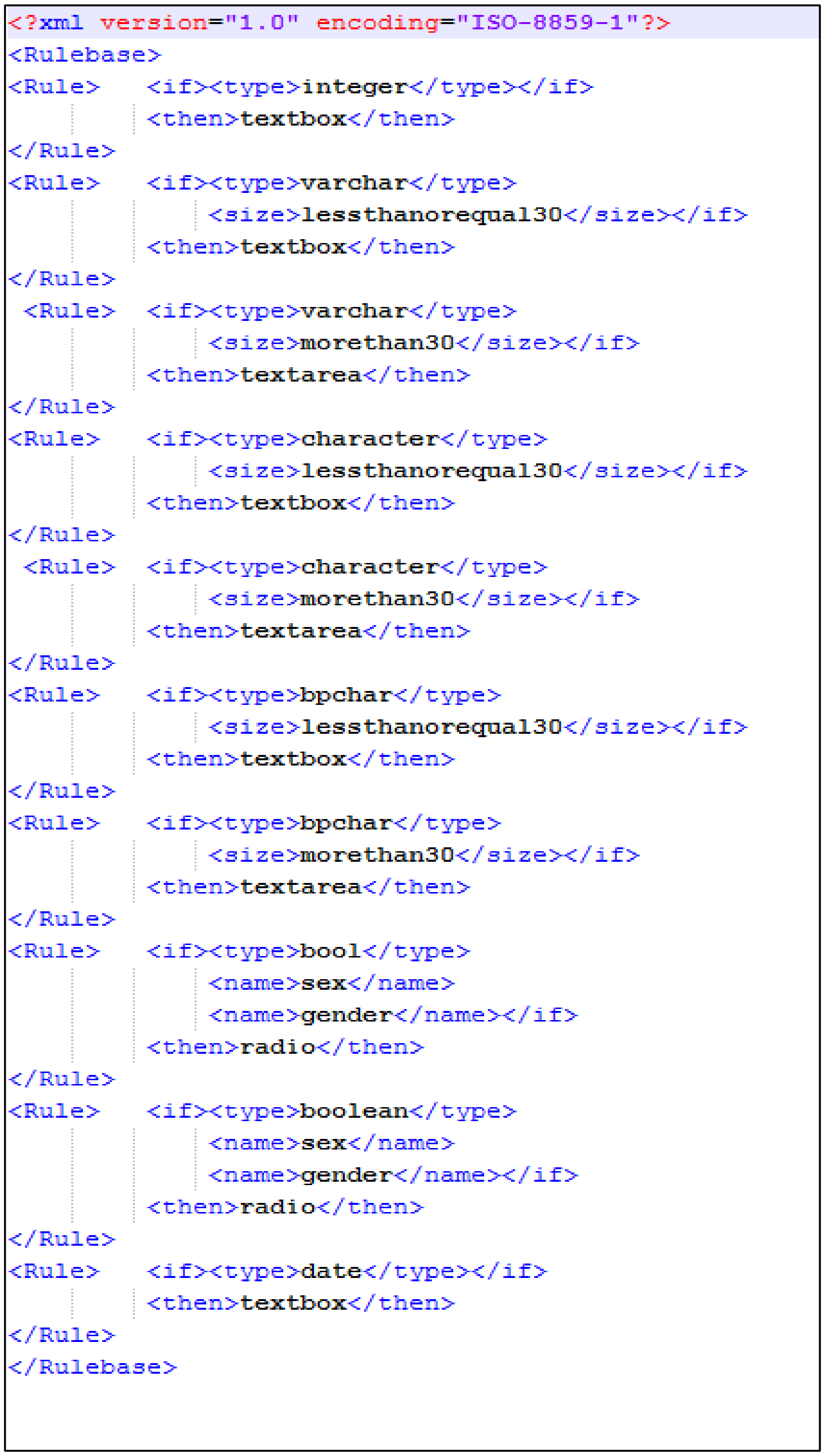

Extract metadata from database
DB
Apply domain specific rulebase
Apply common sense rulebase
Create form elements
Create labels
Create JavaScript checks
Web form

Figure 5-2 Framework mechanism

5.3.2 Building RuleML metadata rulebases

In this section we develop and implement two types of rules introduced in Section 5.3 that can be applied to the information which exists in database metadata. The first rulebase is to save all common sense rules which were originally introduced in Section 4.4.1, using Reaction RuleML 0.2 as an improved version of RuleML 0.1. In Reaction RuleML 0.2 more detailed tags are provided with more meaningful tag naming. Some of the above developed rules are illustrated in Figure 5-3 as:



```
<?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>lessthanorequal30</size></if>
         <then>textbox</then>
</Rule>
 <Rule>  <if><type>varchar</type>
             <size>morethan30</size></if>
         <then>textarea</then>
</Rule>
<Rule>   <if><type>character</type>
             <size>lessthanorequal30</size></if>
         <then>textbox</then>
</Rule>
 <Rule>  <if><type>character</type>
             <size>morethan30</size></if>
         <then>textarea</then>
</Rule>
<Rule>   <if><type>bpchar</type>
             <size>lessthanorequal30</size></if>
         <then>textbox</then>
</Rule>
<Rule>   <if><type>bpchar</type>
             <size>morethan30</size></if>
         <then>textarea</then>
</Rule>
<Rule>   <if><type>bool</type>
             <name>sex</name>
             <name>gender</name></if>
         <then>radio</then>
</Rule>
<Rule>   <if><type>boolean</type>
             <name>sex</name>
             <name>gender</name></if>
         <then>radio</then>
</Rule>
<Rule>   <if><type>date</type></if>
         <then>textbox</then>
</Rule>
</Rulebase>
```

Figure 5-3 Metadata rulebase in Reaction RuleML format as common sense rules.

The second rulebase 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:

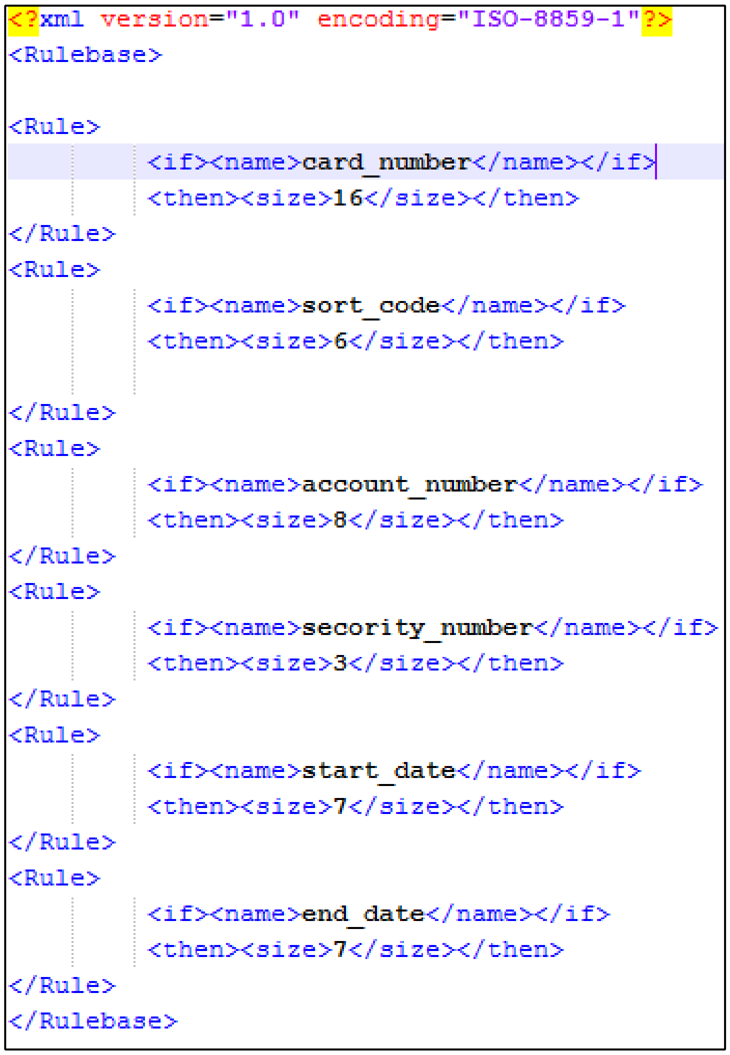
- Rule 1: if a column name is card number and its size is 16, then it should be mapped to textbox Web form control with the *exact* size of 16 digits.
- Rule 2: if a column name is sort code and its size is 6, then it should be mapped to textbox Web form control with the *exact* size of 6 digits.
- Rule 3: if a column name is account number and its size is 8, then it 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 it should be mapped to textbox Web form control with the *exact* size of 3 digits.
- Rule 5: if a column name is start_date or end_date and its size is 7, then it should be mapped to textbox and the required 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 a 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.

Normally database metadata gives the size of form element but a Web form will take extra data even if the database system subsequently truncates them and the metadata limit is a maximum but here we know shorter data is invalid too. So using the above rules to allow for the exact size only. Figure 5-4 shows the algorithm of the above developed rules.

```
BEGIN
    READ column_name, column_size
    IF column_name = card_number && column_size = 16 THEN
        Action= create textbox element
        Size = 16
    ELSEIF column_name = sort_code && column_size = 6 THEN
        Action = create textbox element
        Size = 6
    ELSEIF column_name = account_number && column_size = 8 THEN
        Action = create textbox element
        Size = 8
    ELSEIF column_name =security && column_size = 3 THEN
        Action = create textbox element
        Size = 3
    ENDIF
END
```

Figure 5-4 Pseudo code for domain specific rules

Some of the above developed rules using RuleML format are illustrated in Figure 5-5 as:



```
<?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>
```

Figure 5-5 Metadata rulebase in Reaction RuleML format as domain specific rules

5.3.3 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 (null ability) as shown in Figure 5-6 below:

```
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);
```

Figure 5-6 Database table's structure

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:

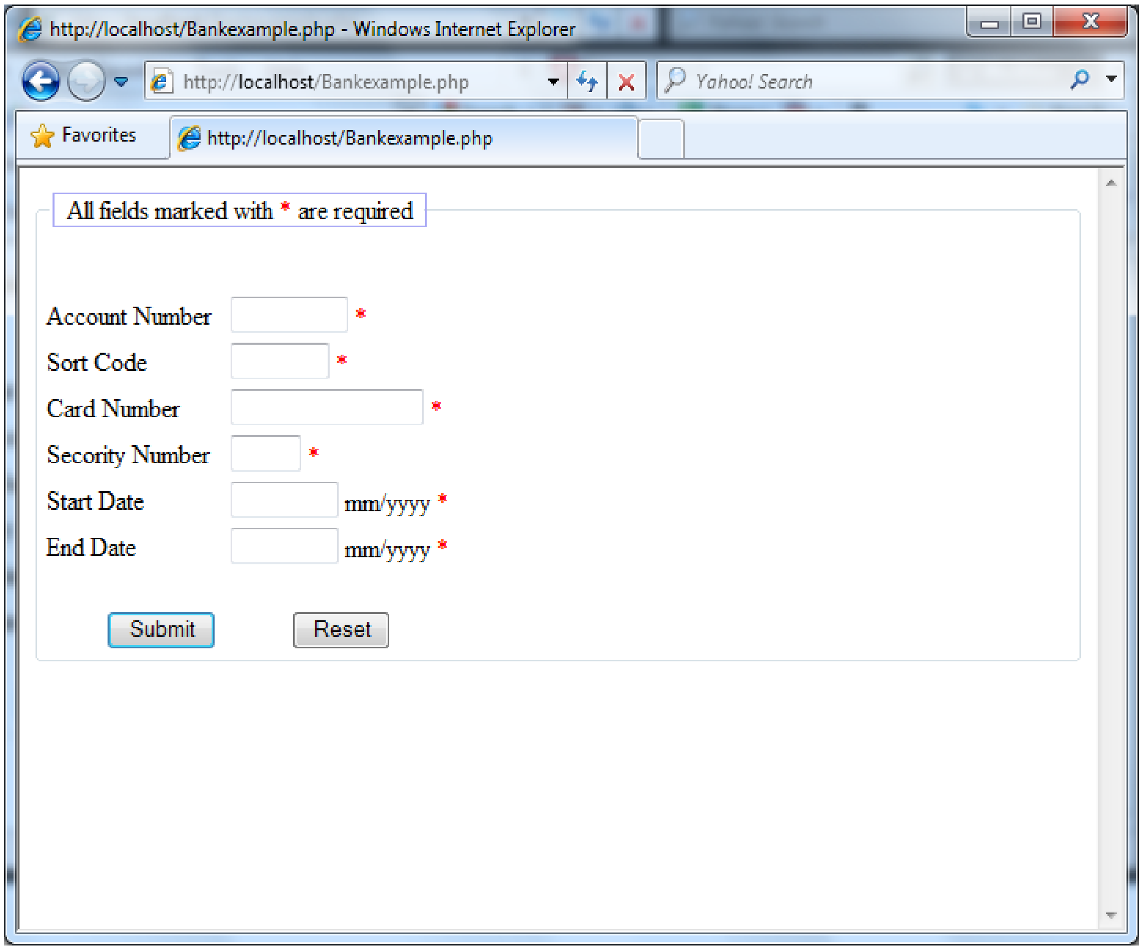
```
$meta = pg_meta_data($conn,'bankinf');
```

Where ‘$conn’ is the connection handle and ‘bankinf’ is the table’s name [84 , 85].

There is another method to retrieve information about each field such as column’s name, type and so on, by using specific functions [84 , 85] as explained in Table 4-1.

5.3.4 Generate the Web entry 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 Figure 5-7 where for example the account_number is mapped to a textbox of the appropriate size (found from the domain specific rules) 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 be 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. We note that the date fields shown use the domain specific rule overriding any default date metadata format.


http://localhost/Bankexample.php - Windows Internet Explorer
http://localhost/Bankexample.php
Yahoo! Search
Favorites
http://localhost/Bankexample.php
Submit
Reset

All fields marked with * are required

Account Number	*
Sort Code	*
Card Number	*
Secority Number	*
Start Date	mm/yyyy *
End Date	mm/yyyy *

Figure 5-7 User interface form generated automatically using metadata and RuleML.

5.4 Chapter Summary

The framework proposed in this chapter demonstrated the potential capabilities of Reaction RuleML 0.2 to provide automatic and dynamic generation of Web forms. In this chapter we proposed an approach which separates the rules as an independent entity from the application code, by using Reaction RuleML 0.2 format as rulebase. These rules are used to give consistency to the appearance of the forms. The framework used common sense rules which supplemented the original database metadata rules and introduced a second rulebase of domain specific rules using Reaction RuleML 0.2. The implementation of the proposed framework is carried out using Reaction RuleML 0.2 format to store the developed rules, PostgreSQL as a DBMS, PHP for server side programming, and HTML and JavaScript for client side programming. As a result of the implementation a Web form for user interface was generated dynamically.

Chapter 6

6 An Extended Rules Framework for Web Forms: Adding to Metadata with Custom Rules to Control Appearance.

6.1 Introduction

This chapter proposes the use of rules that involve code to implement more semantics for Web forms. Separation between content, logic and presentation of Web applications has become an important issue for faster development and easy maintenance. This chapter extends the work presented in Chapter 5 by developing an additional set of rules to control the appearance of Web forms. In particular, a set of rules are proposed to control the appearance of Web form elements in a semantic way using Reaction RuleML 0.2 format in conjunction with database metadata rules.

6.2 Prototype Overview

We know that Web site maintenance is a problem. One solution to this problem is the use of CSS on the client side which can give consistency to the appearance of pages generally. The 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, as presented in Chapter 5.

The work presented in Chapter 5, shown in Figure 5-1, employed the database metadata rules and common sense rules with another rulebase of domain specific rules which has an additional set of rules added to it.

Adding more rules to the domain specific Rulebase can be useful to control the appearance of Web form elements in a semantic way by grouping similar Web form elements in a more precise layout. Also, it helps in designing a query form for retrieving data from database table and to use form for data entry.

Therefore, in this chapter, we investigate the use of Reaction RuleML 0.2 on the server side to give a consistent use of variables and hence a consistent look to the forms across pages within an application that uses a database. In addition, we propose a framework mechanism, as shown in Figure 6-1, which divides the rules into three types. The first type stores the common sense rules and the second type stores domain specific rules which will help to develop a prototype system that can generate automatic and dynamic Web entry forms for Web applications. The third type stores query rules which involve code to implement more semantics of the form elements.

6.3 Framework mechanism

This section introduces a framework mechanism which aims to design a framework, using the Reaction RuleML 0.2 format, to save and implement rulebases in order to generate automatic and dynamic Web forms and to support a composite attribute which consists of a group of values from more than one domain [86]. Query forms can be designed and generated using a set of rules.

The proposed framework consists of several processes as shown in Figure 6-1, which develops the system shown in chapter 5 Figure 5-2. The following objectives are intended to be achieved.

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

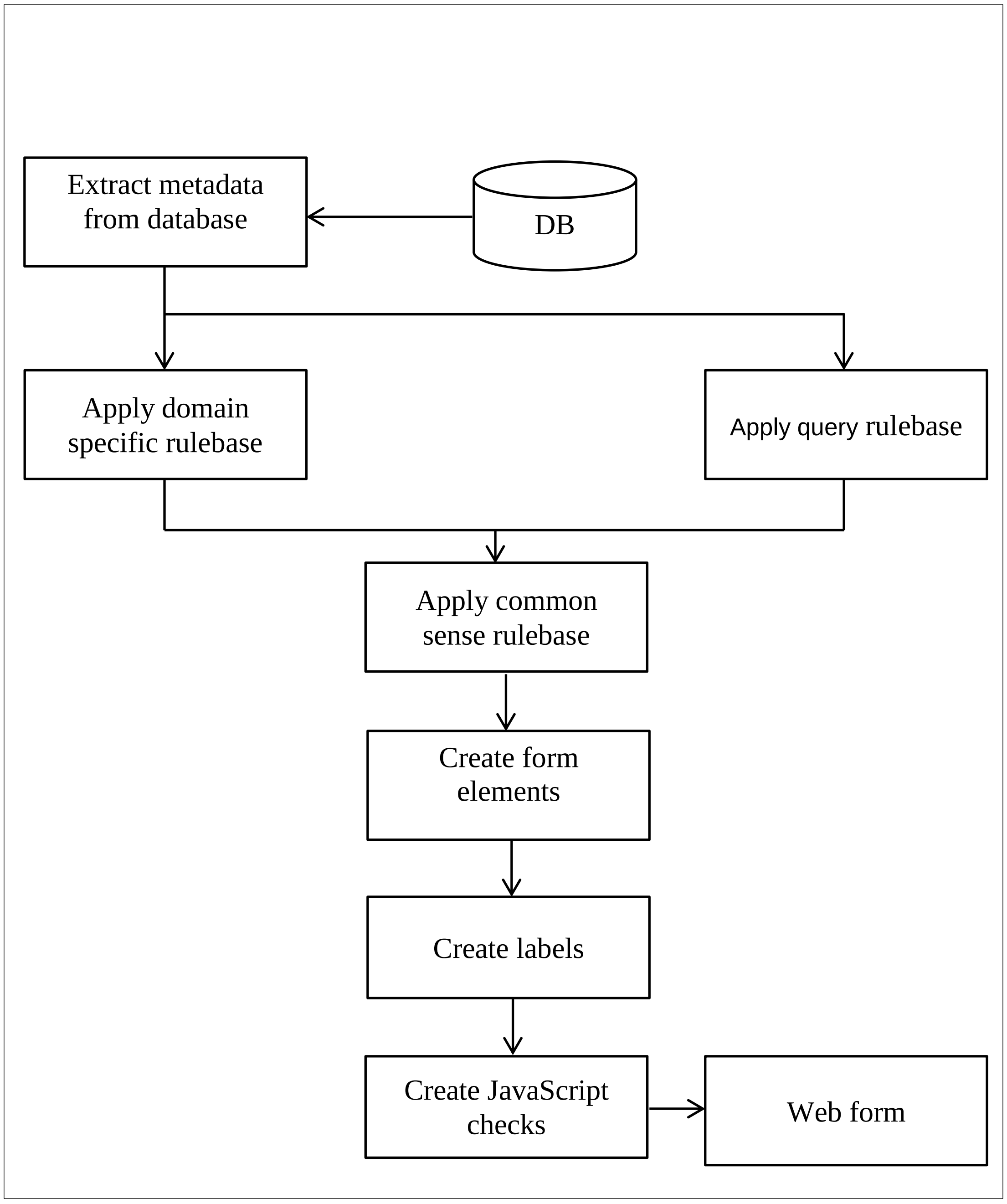

Extract metadata from database
DB
Apply domain specific rulebase
Apply query rulebase
Apply common sense rulebase
Create form elements
Create labels
Create JavaScript checks
Web form

Figure 6-1 Framework mechanism

6.3.1 Building RuleML metadata rulebases

In this section we introduce two types of rules that can be applied to the information that exists in database metadata

- The first rulebase is to save all common sense rules introduced in Section 4.4.1, some of the above developed rules using RuleML format are illustrated in Figure 5-3.
- The second rulebase is to save all domain specific rules, which can be divided to two sets of rules, using our example, which is student information, we develop two sets of domain specific rules the first one can be applied to generate input forms and the second one will be applied to generate the query forms as:

6.3.1.1 The first domain specific rule set

- Rule 1: if a column name is title then it should be grouped to block1.
- Rule 2: if a column name is first_name then it should be grouped to block1.
- Rule 3: if a column name is last_name then it should be grouped to block1.
- Rule 4: if a column name is house_no then it should be grouped to block 2.
- Rule 5: if column name is street then it should be grouped to block 2.
- Rule 6: if column name is town then it should be grouped to block 2.

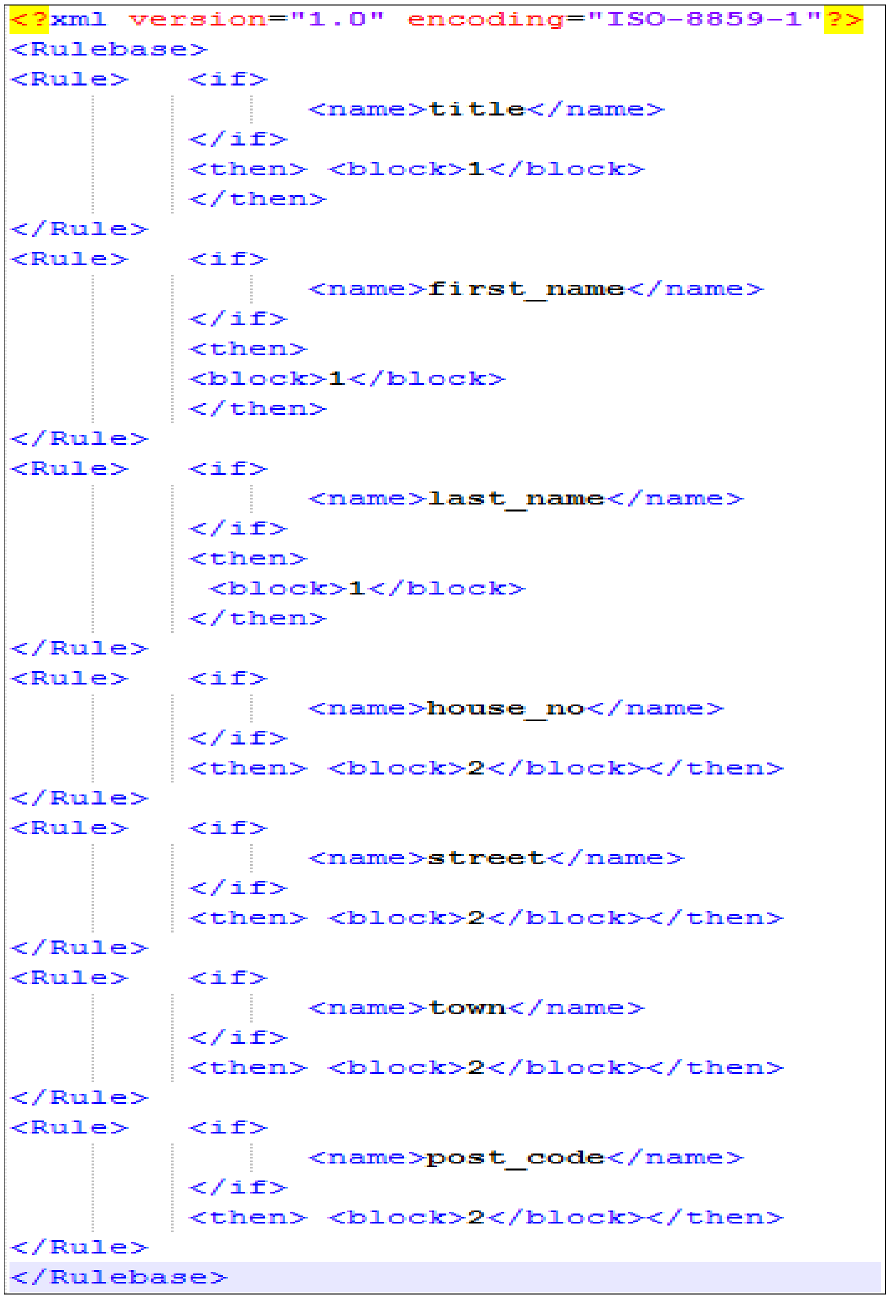
- Rule 7: if column name is post_code then it should be grouped to block 2.

Figure 6-2 shows the algorithm of the above developed rules as:

```
BEGIN
    READ column_name
    CASE column_name is:
        title:          Action= group to block1
        first_name:     Action= group to block1
        last_name:      Action= group to block1
        house_no:       Action= group to block2
        street:         Action= group to block2
        town:           Action= group to block2
        post_code:      Action= group to block2
    ENDCASE
END
```

Figure 6-2 Pseudo code for domain specific input form rules

Some of the above developed rules using RuleML format are illustrated in Figure 6-3 as:



```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Rulebase>
<Rule>    <if>
                <name>title</name>
          </if>
          <then> <block>1</block>
          </then>
</Rule>
<Rule>    <if>
                <name>first_name</name>
          </if>
          <then>
          <block>1</block>
          </then>
</Rule>
<Rule>    <if>
                <name>last_name</name>
          </if>
          <then>
           <block>1</block>
          </then>
</Rule>
<Rule>    <if>
                <name>house_no</name>
          </if>
          <then> <block>2</block></then>
</Rule>
<Rule>    <if>
                <name>street</name>
          </if>
          <then> <block>2</block></then>
</Rule>
<Rule>    <if>
                <name>town</name>
          </if>
          <then> <block>2</block></then>
</Rule>
<Rule>    <if>
                <name>post_code</name>
          </if>
          <then> <block>2</block></then>
</Rule>
</Rulebase>
```

Figure 6-3 Metadata rulebase in Reaction RuleML 0.2 format as domain specific input form rules

6.3.1.2 The second domain specific rule set

These rules will be applied to generate the query form as:

- Rule 1: if a column name is first_name then it should be grouped to block1.
- Rule 2: if a column name is last_name then it should be grouped to block1.
- Rule 3: if a column name is town then it should be grouped to block2.
- Rule 3: if a column name is post_code then it should be grouped to block2.

The rules in **this section** differ from the rules in **section 6.3.1.1**, these rules will be used to generate query form which contains elements shown in the rules only, but rules in **section 6.3.1.1** will be used to generate Web entry form to group the composite attributes which consist of a group of values from more than one domain.

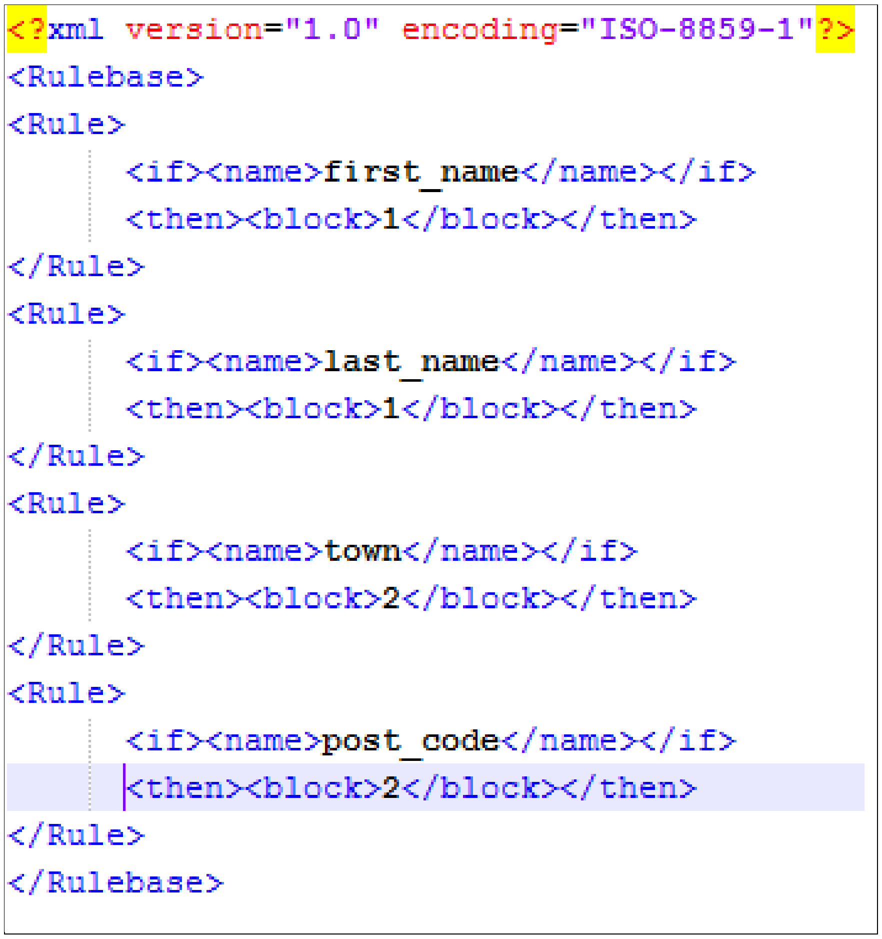
In this section we use our domain specific rules to overcome the lack of semantic content available automatically from a database. We may *know* that first_name and last_name are related items and should be grouped but this information is not available automatically. In most RDBMSs the only connection is in the similarity of the column names. In the example although the "name" elements are named similarly the “address” elements are not. Yet we *know* that the house_no, street, town etc are related in a similar way to first_name and last_name. In some DBMSs the columns could be implemented as a composite type but this is not commonly done.

Figure 6-4 shows the algorithm of the above developed rules as:

```
BEGIN
    READ column_name
    CASE column_name is:
        first_name:   Action= group to block1
        last_name:    Action= group to block1
        town:         Action= group to block2
        post_code:    Action= group to block2
    ENDCASE
END
```

Figure 6-4 Pseudo code for domain specific query form rules

Some of the above developed rules using RuleML format are illustrated in Figure 6-5 as:



```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Rulebase>
<Rule>
        <if><name>first_name</name></if>
        <then><block>1</block></then>
</Rule>
<Rule>
        <if><name>last_name</name></if>
        <then><block>1</block></then>
</Rule>
<Rule>
        <if><name>town</name></if>
        <then><block>2</block></then>
</Rule>
<Rule>
        <if><name>post_code</name></if>
        <then><block>2</block></then>
</Rule>
</Rulebase>
```

Figure 6-5 Metadata rulebase in Reaction RuleML 0.2 format as domain specific query form rules

6.3.2 Retrieving metadata from the database

We start implementing our approach by creating a database table which contains student information, to illustrate a range of data types and other conditions as shown in Figure 6-6 below:

```
    CREATE TABLE student (
student_id serial NOT NULL,
title character(6) NOT NULL,
first_name character(20) NOT NULL,
last_name character(10) NOT NULL,
house_no character(5) NOT NULL,
street character(20) NOT NULL,
town character(20) NOT NULL,
post_code character(10) NOT NULL,
CONSTRAINT student_id PRIMARY KEY (student_id))
```

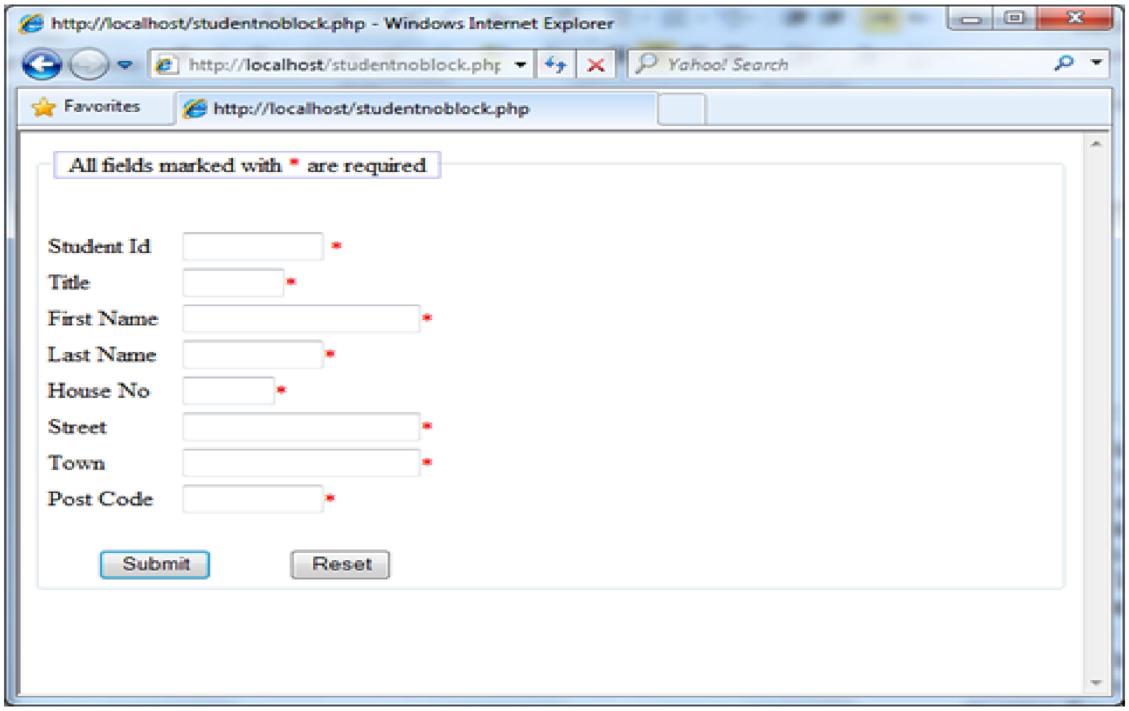
Figure 6-6 Database table's structure

A number of PHP's PostgreSQL functions are used to make a connection to the database and retrieve the metadata [85 , 87].

6.3.3 Generate the Web forms

A general purpose PHP script was written which loops through all the metadata for each column and uses the Reaction RuleML 0.2 rulebases. It tests to see which rules apply and then uses those rules to build the form elements on the fly as:

- In Figure 6-7 using the common sense rules where for example the student_id 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 be 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.



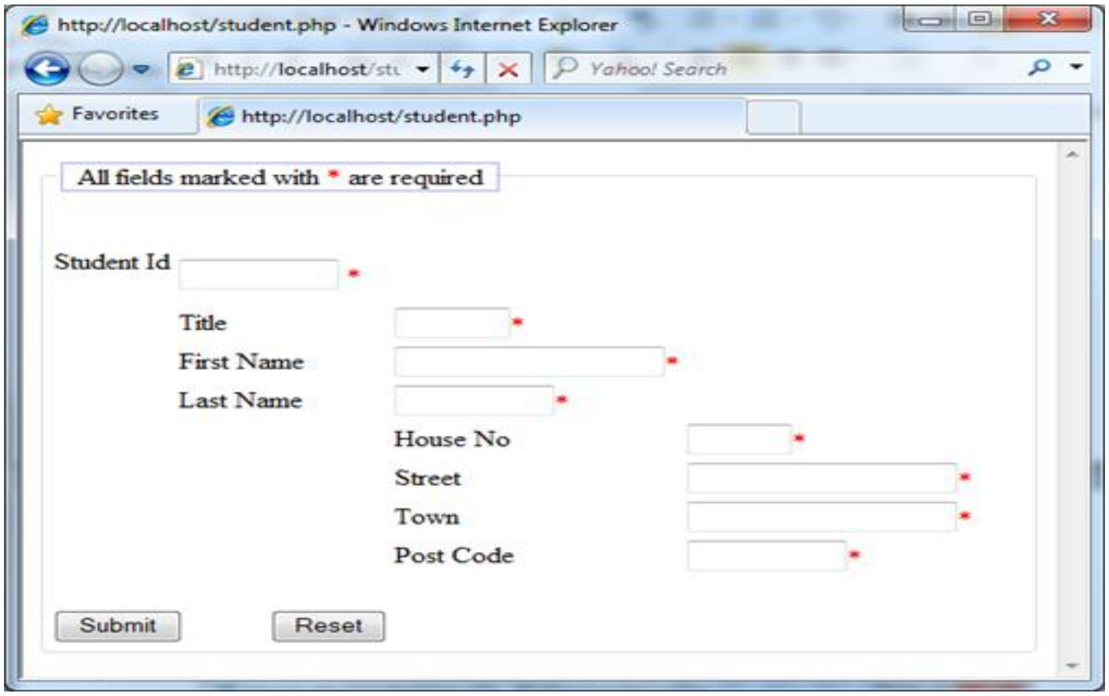
http://localhost/studentnoblock.php - Windows Internet Explorer

All fields marked with * are required

Student Id	*
Title	*
First Name	*
Last Name	*
House No	*
Street	*
Town	*
Post Code	*

Figure 6-7 User interface input form generated automatically using metadata and Reaction RuleML 0.2

- In Figure 6-8 using the first domain specific rules, which are applied to generate Web entry form to group the composite attributes which consist of a group of values from more than one domain. For example the attributes (title, first_name, last_name,) were mapped as a block to give more semantics to the form, and the attributes (house_no, street, town, post_code) were mapped as a block to group the elements together as a composite attribute. The rules represent all the semantic information which is not in the database metadata to order the Web form elements. In the example we simply indent each block; other styling could be applied using CSS.



http://localhost/student.php - Windows Internet Explorer
http://localhost/stu
Yahoo! Search
Favorites
http://localhost/student.php
Submit
Reset

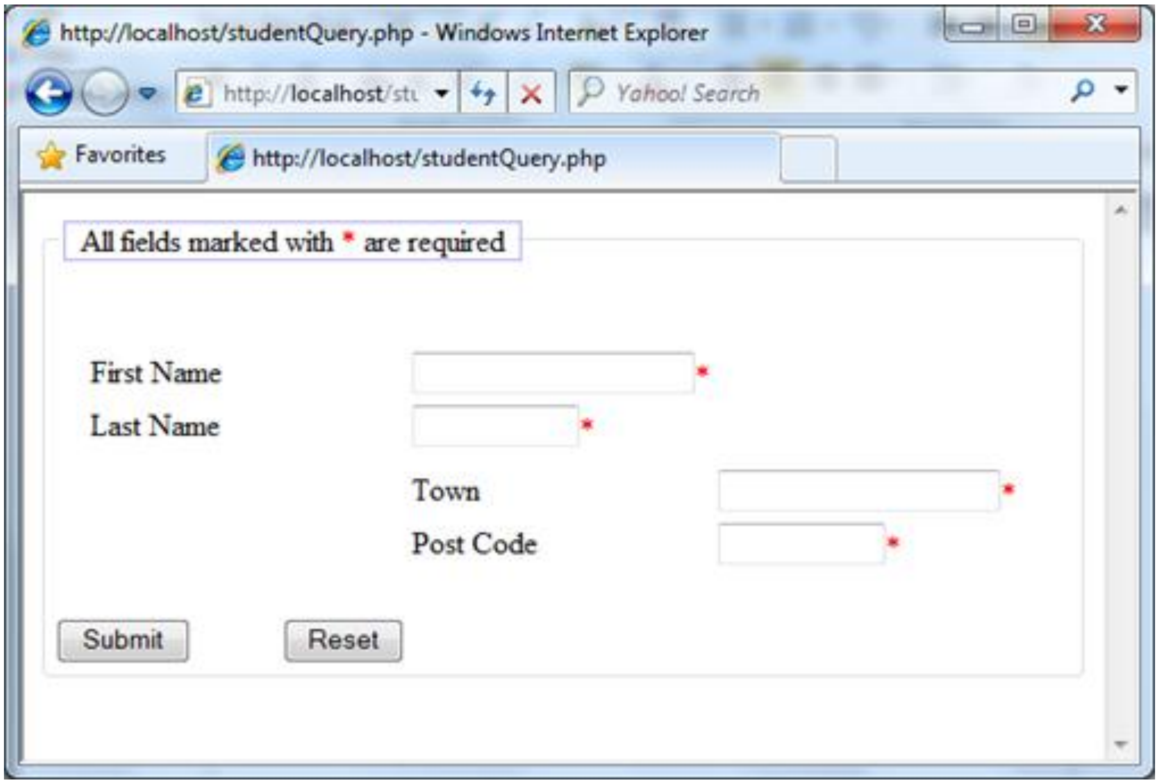
All fields marked with * are required

Student Id *

Title *
First Name *
Last Name *
House No *
Street *
Town *
Post Code *

Figure 6-8 User interface input form generated automatically using metadata and Reaction RuleML 0.2 grouped the attributes as a composite attribute.

- In Figure 6-9 using the second domain specific rules, which are applied to generate Web query form to order the required attributes. For example the attributes (first_name, last_name, town, post_code) have been ordered to generate the query form as in Figure 6-9. The rules could be used to order and generate more forms which can contain different attributes to help the user to get the needed information.


http://localhost/studentQuery.php - Windows Internet Explorer
http://localhost/stu
Yahoo! Search
Favorites
http://localhost/studentQuery.php
Submit
Reset

All fields marked with * are required

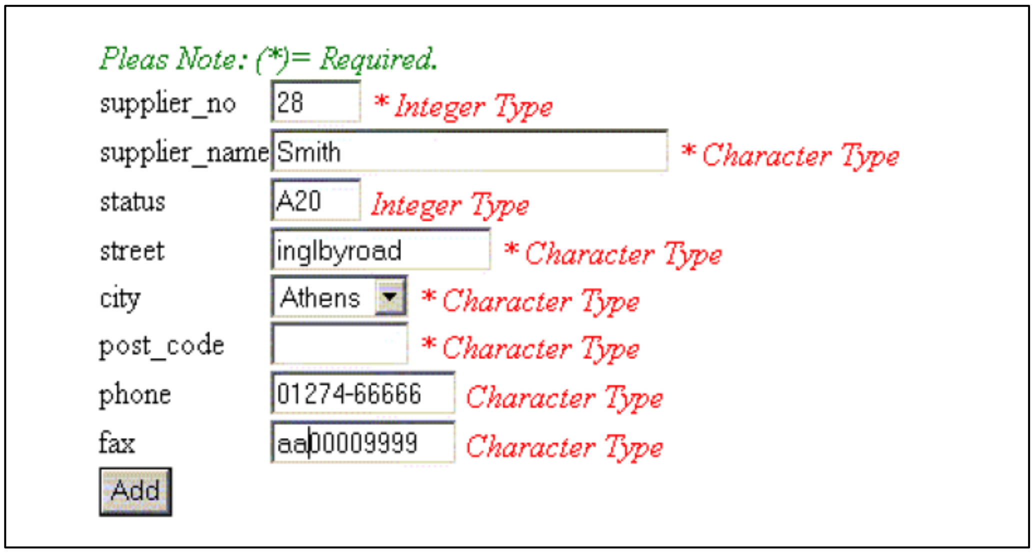
First Name *
Last Name *
Town *
Post Code *

Figure 6-9 User interface query form generated automatically using metadata and Reaction RuleML 0.2.

6.4 Evaluation

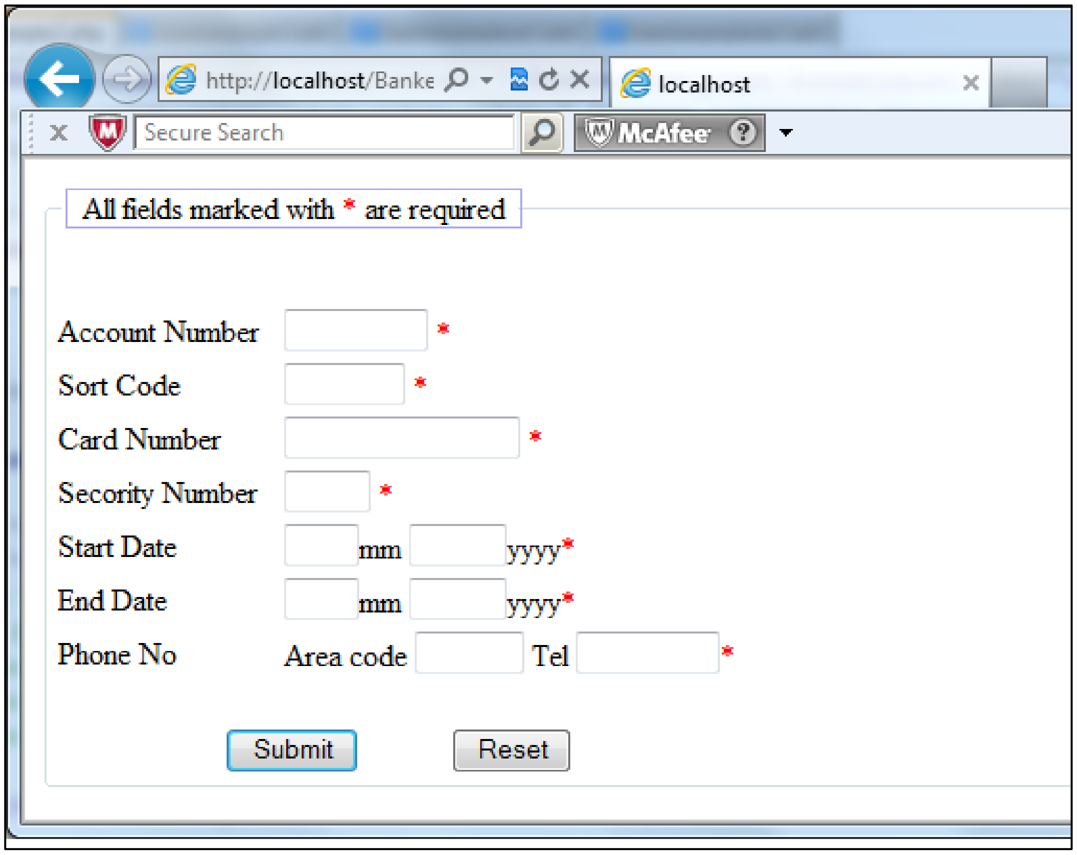
We found out that a wide range of techniques and ideas to automate the generation of Web forms does exist. These techniques and ideas however, are not capable of generating the most dynamic behaviour of form elements, and make insufficient use of database metadata to control Web forms' generation and appearance. In addition, it has been concluded that when rules are embedded in application code, it becomes difficult to locate and change the logic [4] , and each modification requires recompiling the application code.

Elbibas et al. in [78] proposed an approach to develop and maintain HTML forms based on metadata extracted from a database table. Their proposed approach generates dynamic HTML forms which have been generated and validated automatically. They use a set of supporting rules to map each column to the most appropriate form element and which are embedded in the application code where it is difficult to locate and change their logic. Moreover, the set of rules does not support the manipulation of semantics of database metadata in some cases, As shown in Figure 6-10 below, the column of phone number was mapped to one textbox including the area code; here we can use our semantic knowledge of the structure of landline phone numbers to improve the user experience and avoid possible errors. By using domain specific rules, we offer mapping the column for landline telephone numbers to two separate textboxes control elements; the first one is for the area code and the second one is for the telephone number as shown in Figure 6-11.


Pleas Note: (*)= Required.

supplier_no	28	* Integer Type
supplier_name	Smith	* Character Type
status	A20	Integer Type
street	inglbyroad	* Character Type
city	Athens	* Character Type
post_code		* Character Type
phone	01274-66666	Character Type
fax	aa00009999	Character Type

Figure 6-10 Shows suppliers html entry form generated using metadata [78].

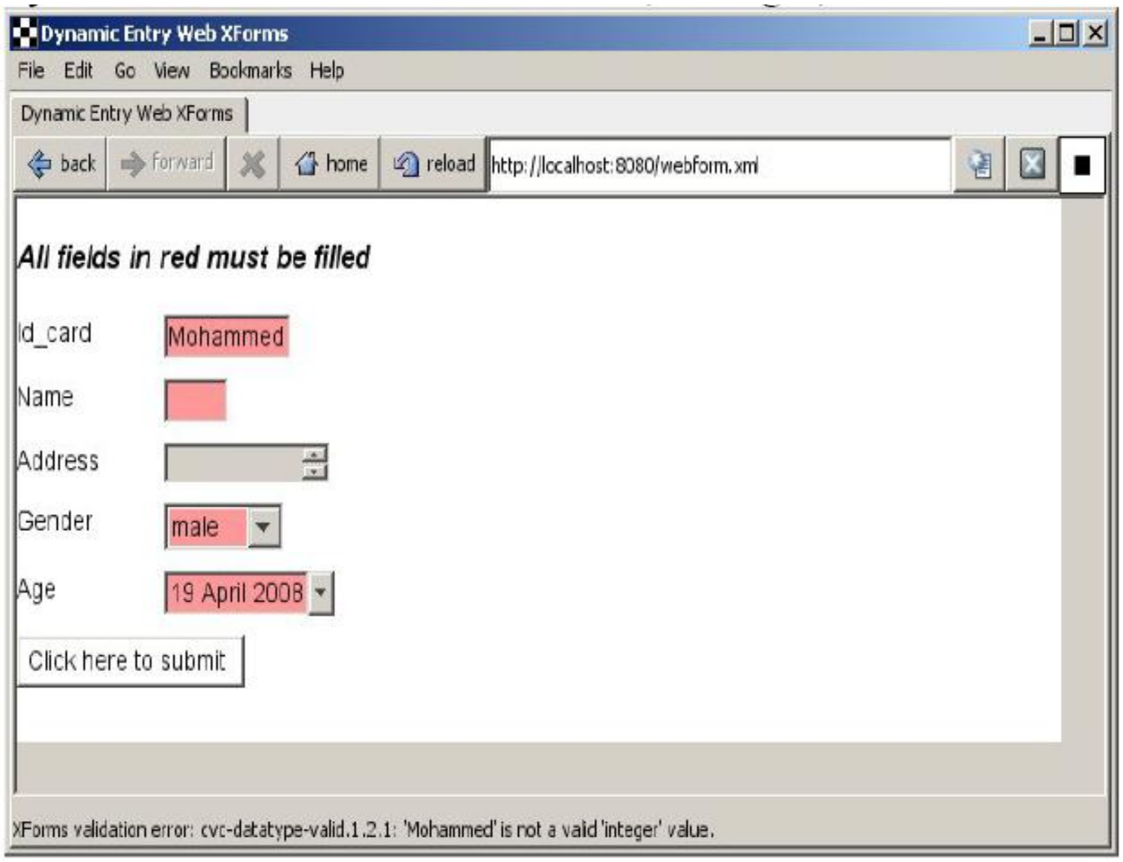


All fields marked with * are required

Account Number		*		
Sort Code		*		
Card Number		*		
Security Number		*		
Start Date	mm	yyyy*		
End Date	mm	yyyy*		
Phone No	Area code		Tel	*

Figure 6-11 User interface form generated automatically using metadata and RuleML.

Elsheh et al. in [1 , 4] proposed a model which aims to generate dynamic Web entry forms based on metadata extracted from system tables. This approach has the same problems encountered by [78] [78] when the set of rules are embedded in the application code where it is difficult to locate and change their logic. In addition, the set of rules does not support the manipulation of semantics of database metadata in some cases as in Figure 6-12 and Figure 6-13 below. The address column is mapped to one textarea control element. As an example of this: how do we know house no, street, town and postcode are related? Our approach can tackle this problem in two ways: first by developing rulebases to support the common sense rules, for example to deal with columns’ names. Secondly by developing domain specific rules to support the generic rules for example to deal with columns’ names and sizes. So domain specific rules were developed to support the common sense rules to deal with columns names as in Figure 6-14. Common sense rules use domain specific rules for some advanced information. As the common sense rules use column type (found from database metadata) and domain specific rules support the common sense rules by providing the exact size (found from domain specific rulebase) for specific columns using column name (found from database metadata). Domain specific rules use the name and the size of each database column to support mapping each column to a Web form control element.

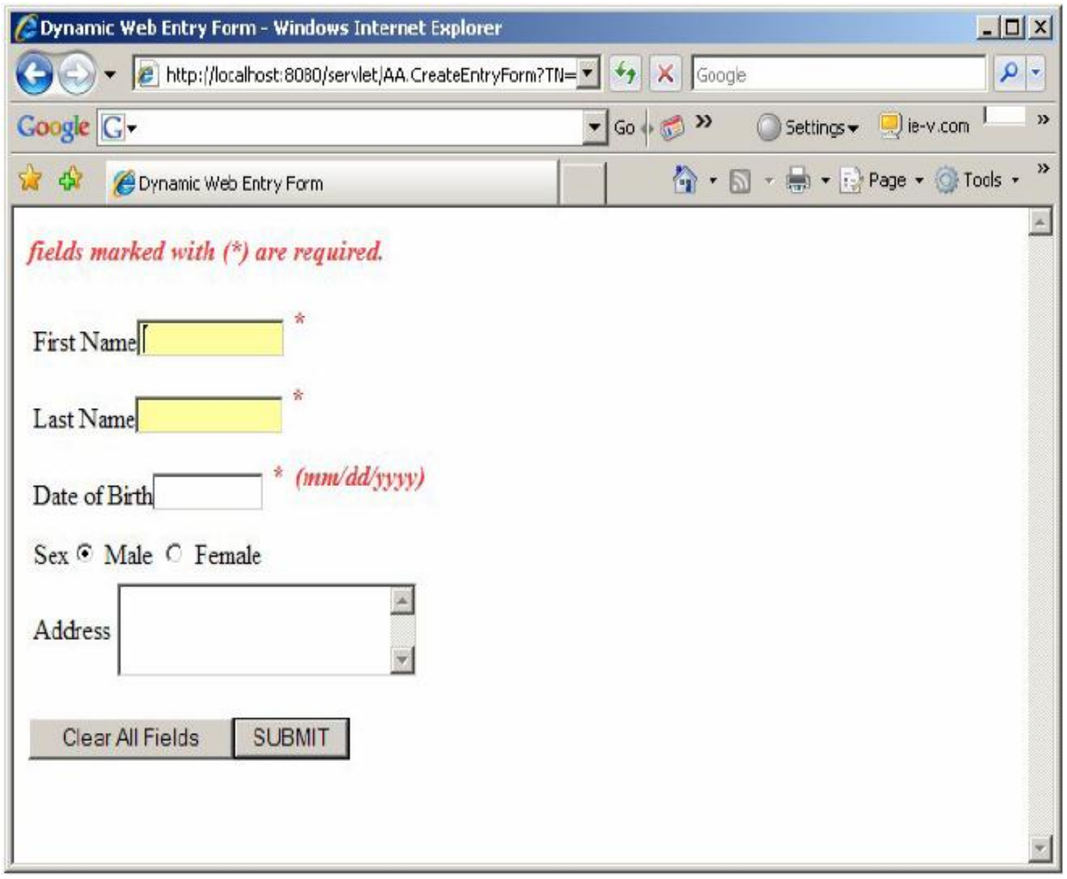


All fields in red must be filled

Id_card	Mohammed
Name	
Address	
Gender	male
Age	19 April 2008

XForms validation error: cvc-datatype-valid.1.2.1: 'Mohammed' is not a vaid 'integer' value.

Figure 6-12 A snapshot of Web XForms generated on the fly [1].


Dynamic Web Entry Form - Windows Internet Explorer
http://localhost:8080/servlet/AA.CreateEntryForm?TN=
Google
Go
Settings
ie-v.com
Dynamic Web Entry Form
Page
Tools

fields marked with (*) are required.

First Name *

Last Name *

Date of Birth * *(mm/dd/yyyy)*

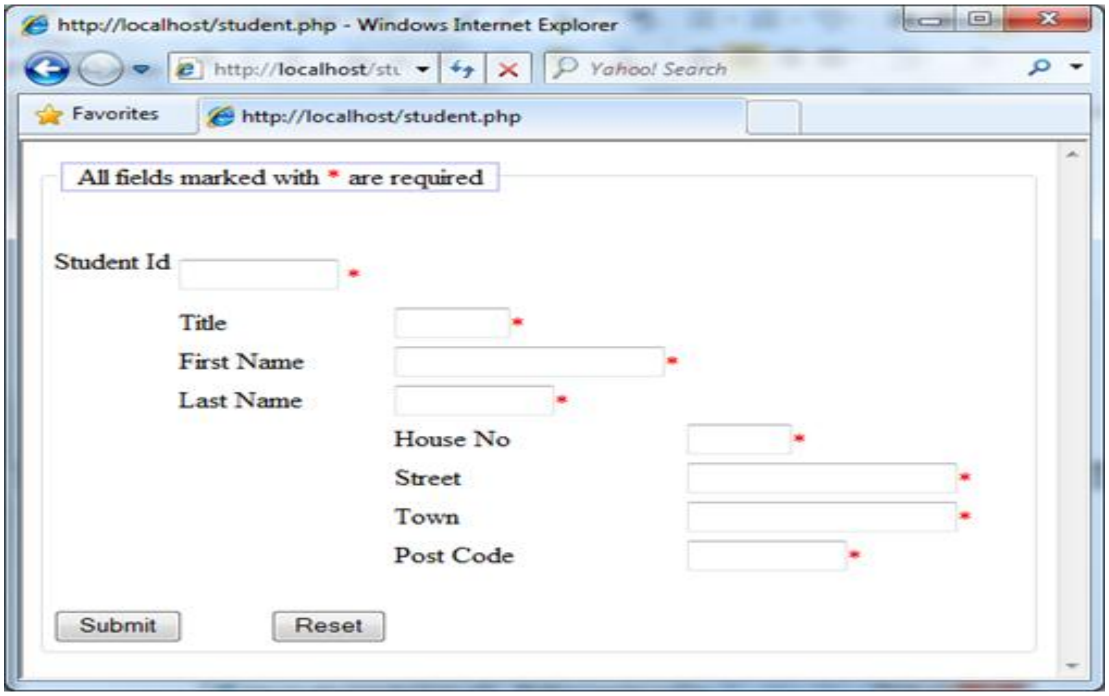
Sex [✓ Male] Female

Address

Clear All Fields SUBMIT

Figure 6-13 A snapshot of an XHTML Web Entry Form generated on the fly [4].

Domain specific rules can support a composite type to map related attributes as one block to give more semantics to a form. These are applied to generate Web entry form to group the composite attributes that consist of a group of values from more than one domain. For example the attributes (title, first_name, last_name,) were mapped as one block to give more semantics to the form, and the attributes (house_no, street, town, post_code) were mapped as separate block to group the related elements together as a composite attribute as shown in Figure 6-14. The rules represent all the semantic information which is not in the database metadata to order the Web form elements. In the example below, we simply indent each block; other styling could be applied using CSS.


http://localhost/student.php - Windows Internet Explorer
http://localhost/stu
Yahoo! Search
Favorites
http://localhost/student.php

All fields marked with * are required

Student Id *

Title *
First Name *
Last Name *

House No *
Street *
Town *
Post Code *

Submit Reset

Figure 6-14 User interface input form generated automatically grouped the attributes as a composite attribute.

6.5 Chapter Summary

As a result of using the rules a Web form for user interface is generated dynamically. This approach aims to use the produced common sense rules introduced in Section 4.4.1, Figure 5-3 shows some of these rules in Reaction RuleML0.2 format and introduce a second rulebase of domain specific rules using Reaction RuleML0.2, and a further development of rules that involve code to implement more semantics and to separate between content, logic and presentation of Web application. The development of rules to order and or group form elements was divided to two rulebases. The first one applied to generate an input form and group the related attributes as blocks, and the second one applied to generate a Web query form to order the required attributes and control the form layout. So they helped in designing the query forms and include only those useful elements, which will be used to query the database.

Chapter 7

7 Extending the use of RuleML to store metadata and database semantics

7.1 Introduction

Shifting legacy data held in stand-alone systems to be used in Web application systems can be expensive and time consuming. RuleML can be used to represent RDBMS data by storing database metadata in an external format for some design tools. Just as XML Schema which uses elements and attributes to express the semantics of XML data, but XML Schema does not have active elements [79], in principle RuleML could be used as a representation for RDBMS metadata too. This chapter proposes the use of RuleML format to implement more semantics for Web forms.

In this chapter we demonstrate how this RuleML based approach can provide support for greater semantics using the example of advanced domain support even when this is not a DBMS feature. Many database systems do not support domains and composite attributes, for example MySQL does not support user defined domains which can be created as data type and then use the type in a table definition. We present an approach which is used to specify composite types and constraints.

7.2 Prototype Overview

Domains are useful for abstracting common fields between tables into a single location for maintenance. For example, an email address column may be used in several tables, all with the same properties. This allows us to define a domain and use that rather than setting up each table's constraints individually. The benefits of domains are many [88] for example:

- A constraint placed on a domain ensures that all columns and variables intended to hold values in a range or format can hold only the intended values. For example, a data type can ensure that all credit card numbers typed into the database contain the correct number of digits.
- To make the applications and the database structure easy to understand.

Database logic is found in multiple places in RDBMSs for example type information in create table statements and create domain statements; therefore it will be helpful if we can get all rules/logic in one format. In addition if we can provide a more independent format that can help transfers from one RDBMS to another of both metadata and data itself.

Not all RDBMSs fully support advanced SQL features such as create domain. Even if they do they may or may not support further features such as constraints within create domain or composite type. We illustrate this with a typical create table statement from a system that doesn't support domains as in Figure 7-1 below:

```
create table person (

                id serial,

                name char (25),

                building_no char (5),

                street char (20),

                town char (20),

                postcode char (25));
```

Figure 7-1 Person table creation without composite type

PostgreSQL now supports the creation of more structure in create table statements as illustrated below:

- Create structured type as in Figure 7-2 below of creating address table as type of composite attributes [89]. We create an address structured type via the route of creating a table. In most advanced RDBMSs table creation is equivalent to type creation [90]:

```
create table address(

            building_no char (5),

            street char (20),

            town char (20),

            postcode char (25));
```

Figure 7-2 Address table (and type) information

- Create a table that uses the address table as in the example below. This shows how the address table can be used in another table as a type for the address column [89]:

```
create table person (
                    id serial,
                    name char (25),
                    address address);
```

Figure 7-3 Person table creation using composite type

Figure 7-1and Figure 7-3 representations may be seen as equivalent in that they both store the same data but arguably the form using the address type has greater semantics and would be preferable if this feature is supported.

Our aim is to provide rich representations in RuleML for the table information that can be used to create the richest table structure in any RDBMS, the richer structure also support the development of the semantically richer forms developed earlier.

7.3 **Framework mechanism**

This section introduces a mechanism which aims to design a framework, using an XML format, to save database table’s metadata in an external format using RuleML in order to support the creation of tables using domains as attribute types and composite attributes which consist of groups of values from more than one domain. This can be used with RuleML rulebases in order to generate automatic and dynamic Web forms. The proposed framework consists of several processes as shown in Figure 7-4. The following objectives are intended to be achieved.

- Store table's metadata in XML files. These files uses XML tags to describe the tables and it's columns information as:

```
<Rulebase><table><name>         </name>
                <column><name>    </name>
                        <type>      </type>
                        <size>      </size>
                        <isnull>     </isnull>
                        <unique>   </unique>
                        <key>        </key>
                </column>
   </table></Rulebase>
```

Each column is represented in a single XML node, and the empty tags could be included.

- Create database tables using the stored metadata for new database or reuse the existed database tables. To create the new tables a PHP script is used which reads the structure of the table stored in XML files. This script then creates the SQL script which actually creates the table in RDBMS.

- Apply Rulebase in conjunction with the metadata of each column stored in XML file to map each column to the correct Web entry control element.
- Generate Web form element.

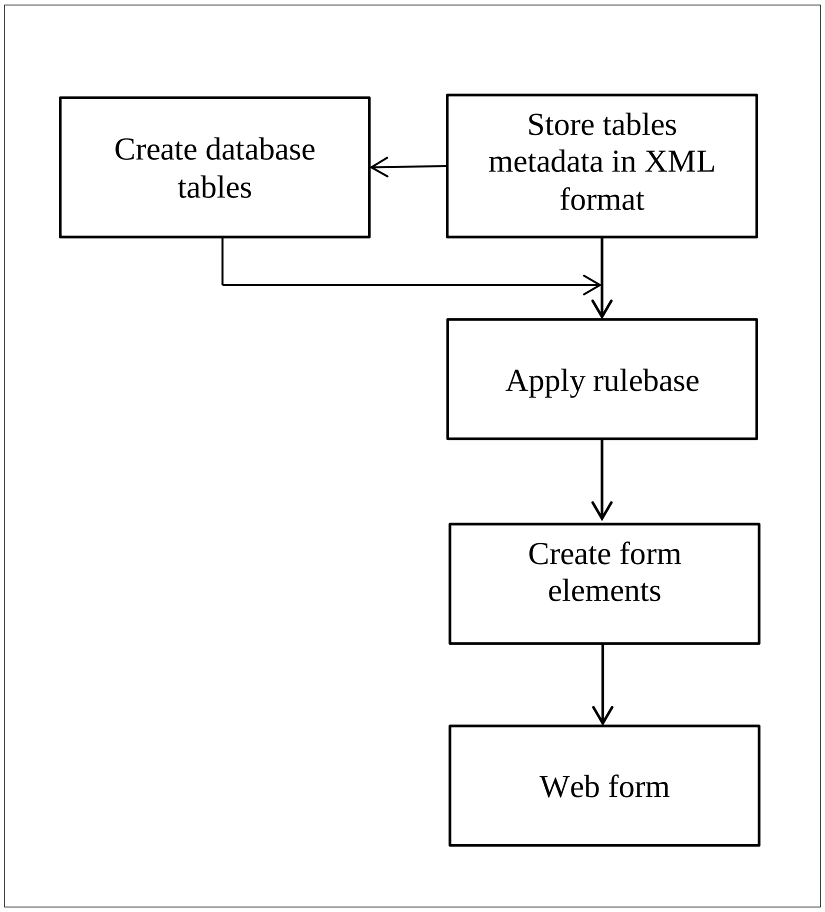


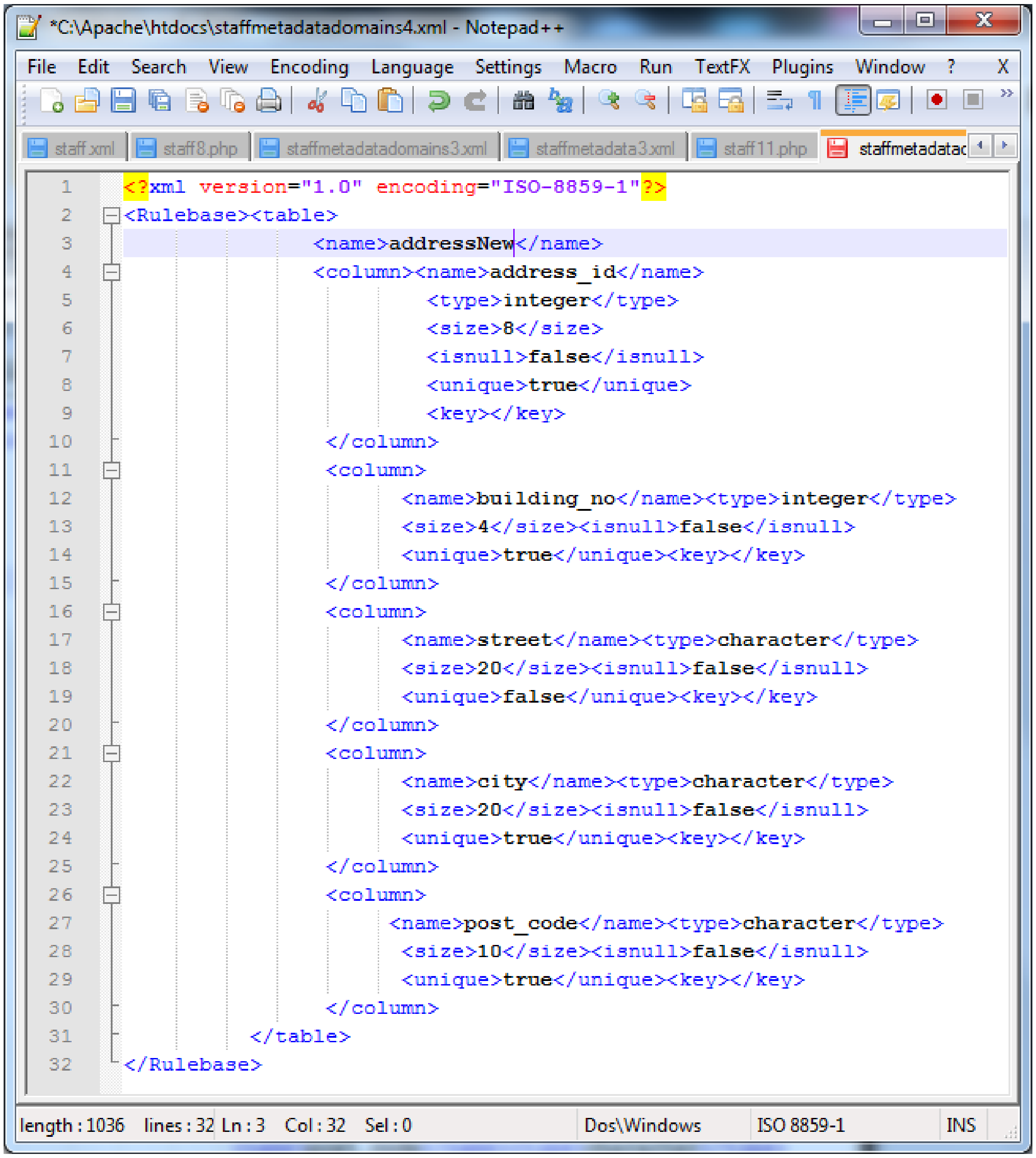
Figure 7-4 Framework mechanism

To illustrate this mechanism and investigate if there are any difficulties in implementing it, the following sections introduce an example of the implementation of this approach.

7.3.1 Table's metadata in XML files for table creation

A database schema is represented in RuleML file. This RuleML information uses XML tags to describe the tables, columns, rows as in

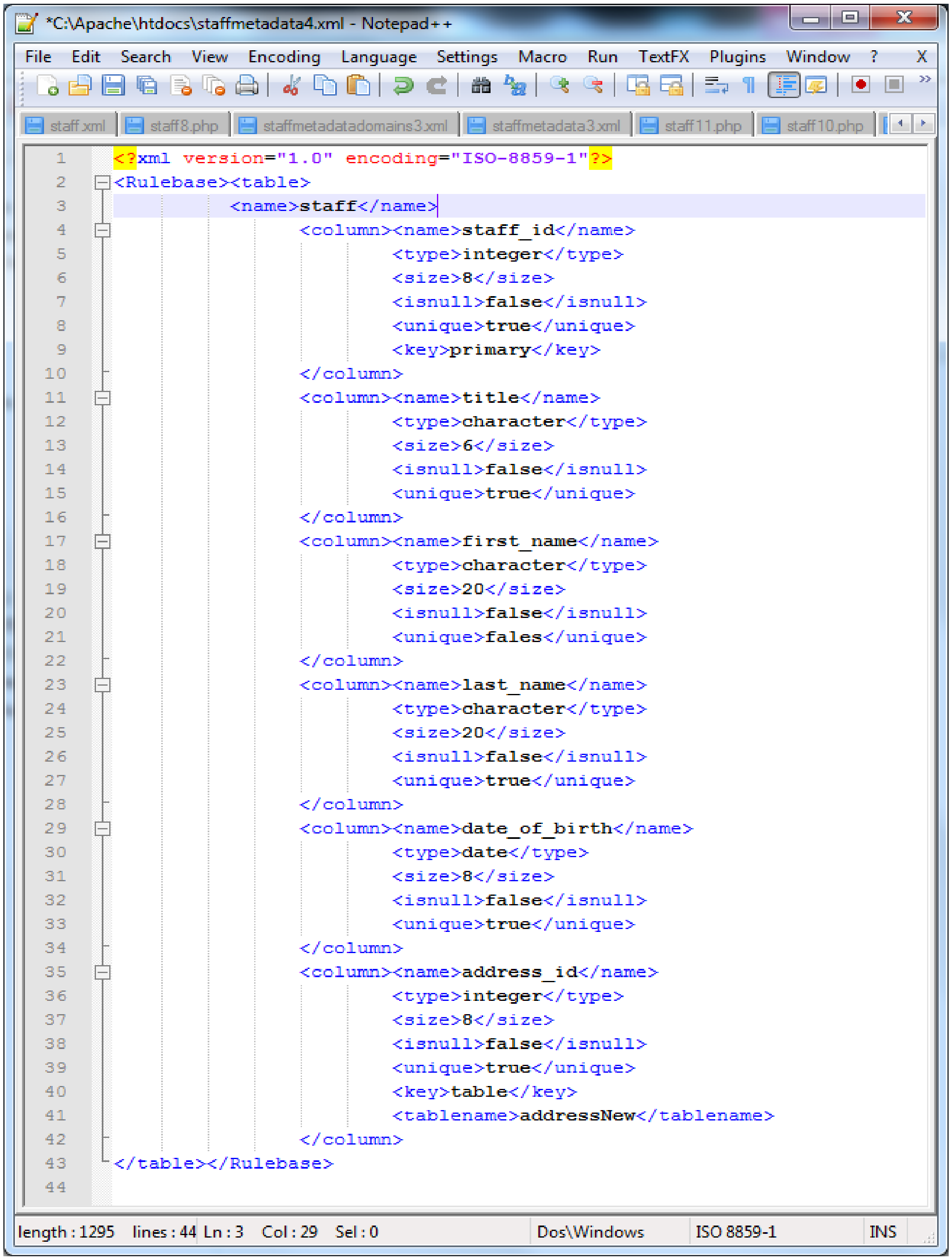
Figure 7-5, and Figure 7-6. It is used for modelling database information, so the previous structure of composite attributes or domains could be represented in XML tags as in the example below:


*C:\Apache\htdocs\staffmetadatadomains4.xml - Notepad++
File Edit Search View Encoding Language Settings Macro Run TextFX Plugins Window ? X
staff.xml staff8.php staffmetadatadomains3.xml staffmetadata3.xml staff11.php staffmetadatac

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Rulebase><table>
                <name>addressNew</name>
                <column><name>address_id</name>
                         <type>integer</type>
                         <size>8</size>
                         <isnull>false</isnull>
                         <unique>true</unique>
                         <key></key>
                 </column>
                 <column>
                       <name>building_no</name><type>integer</type>
                       <size>4</size><isnull>false</isnull>
                       <unique>true</unique><key></key>
                 </column>
                 <column>
                       <name>street</name><type>character</type>
                       <size>20</size><isnull>false</isnull>
                       <unique>false</unique><key></key>
                 </column>
                 <column>
                       <name>city</name><type>character</type>
                       <size>20</size><isnull>false</isnull>
                       <unique>true</unique><key></key>
                 </column>
                 <column>
                      <name>post_code</name><type>character</type>
                       <size>10</size><isnull>false</isnull>
                       <unique>true</unique><key></key>
                 </column>
           </table>
</Rulebase>
```

length : 1036 lines : 32 Ln : 3 Col : 32 Sel : 0
Dos\Windows
ISO 8859-1
INS

Figure 7-5 address table's metadata represented in XML tags


*C:\Apache\htdocs\staffmetadata4.xml - Notepad++
File Edit Search View Encoding Language Settings Macro Run TextFX Plugins Window ? X
staff.xml
staff8.php
staffmetadatadomains3.xml
staffmetadata3.xml
staff11.php
staff10.php

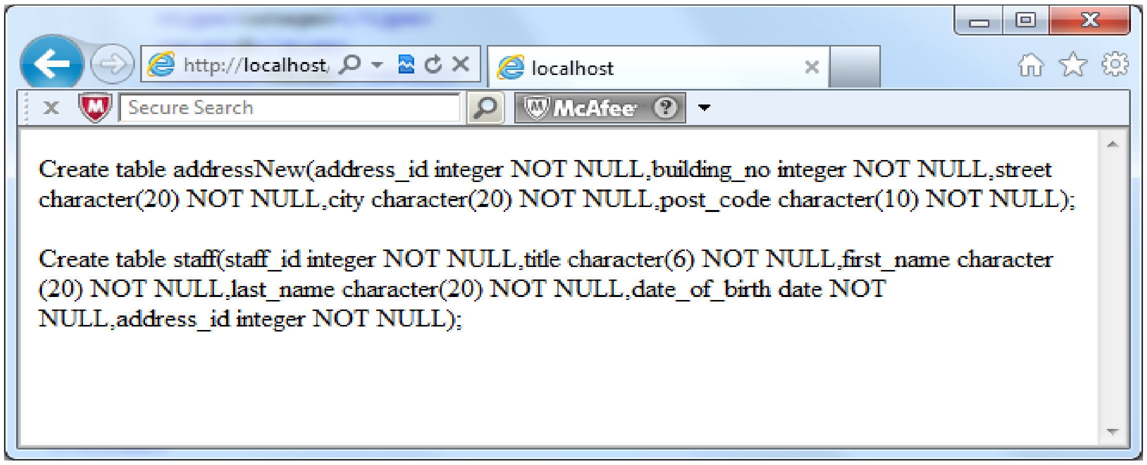
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Rulebase><table>
            <name>staff</name>
                  <column><name>staff_id</name>
                          <type>integer</type>
                          <size>8</size>
                          <isnull>false</isnull>
                          <unique>true</unique>
                          <key>primary</key>
                  </column>
                  <column><name>title</name>
                          <type>character</type>
                          <size>6</size>
                          <isnull>false</isnull>
                          <unique>true</unique>
                  </column>
                  <column><name>first_name</name>
                          <type>character</type>
                          <size>20</size>
                          <isnull>false</isnull>
                          <unique>fales</unique>
                  </column>
                  <column><name>last_name</name>
                          <type>character</type>
                          <size>20</size>
                          <isnull>false</isnull>
                          <unique>true</unique>
                  </column>
                  <column><name>date_of_birth</name>
                          <type>date</type>
                          <size>8</size>
                          <isnull>false</isnull>
                          <unique>true</unique>
                  </column>
                  <column><name>address_id</name>
                          <type>integer</type>
                          <size>8</size>
                          <isnull>false</isnull>
                          <unique>true</unique>
                          <key>table</key>
                          <tablename>addressNew</tablename>
                  </column>
</table></Rulebase>
```

length : 1295 lines : 44 Ln : 3 Col : 29 Sel : 0
Dos\Windows
ISO 8859-1
INS

Figure 7-6 staff table’s metadata represented in XML tags

7.3.2 Database tables creation

To create new tables a PHP script is used to read the structure of the table stored in XML files as in Figure 7-5, Figure 7-6. This script then creates the SQL script as shown in Figure 7-7, which actually creates the tables in the RDBMS.



```
Create table addressNew(address_id integer NOT NULL,building_no integer NOT NULL,street
character(20) NOT NULL,city character(20) NOT NULL,post_code character(10) NOT NULL);

Create table staff(staff_id integer NOT NULL,title character(6) NOT NULL,first_name character
(20) NOT NULL,last_name character(20) NOT NULL,date_of_birth date NOT
NULL,address_id integer NOT NULL);
```

Figure 7-7 SQL script created dynamically using table's metadata stored in XML files

As a result of the created SQL script the tables originally specified in the XML file will be created as below:

```
CREATE TABLE addressnew (
 address_id integer NOT NULL,
 building_no integer NOT NULL,
 street character(20) NOT NULL,
 city character(20) NOT NULL,
 post_code character(10) NOT NULL);
```

```
CREATE TABLE staff (
  staff_id integer NOT NULL,
  title character(6) NOT NULL,
  first_name character(20) NOT NULL,
  last_name character(20) NOT NULL,
  date_of_birth date NOT NULL,
  address_id integer NOT NULL);
```

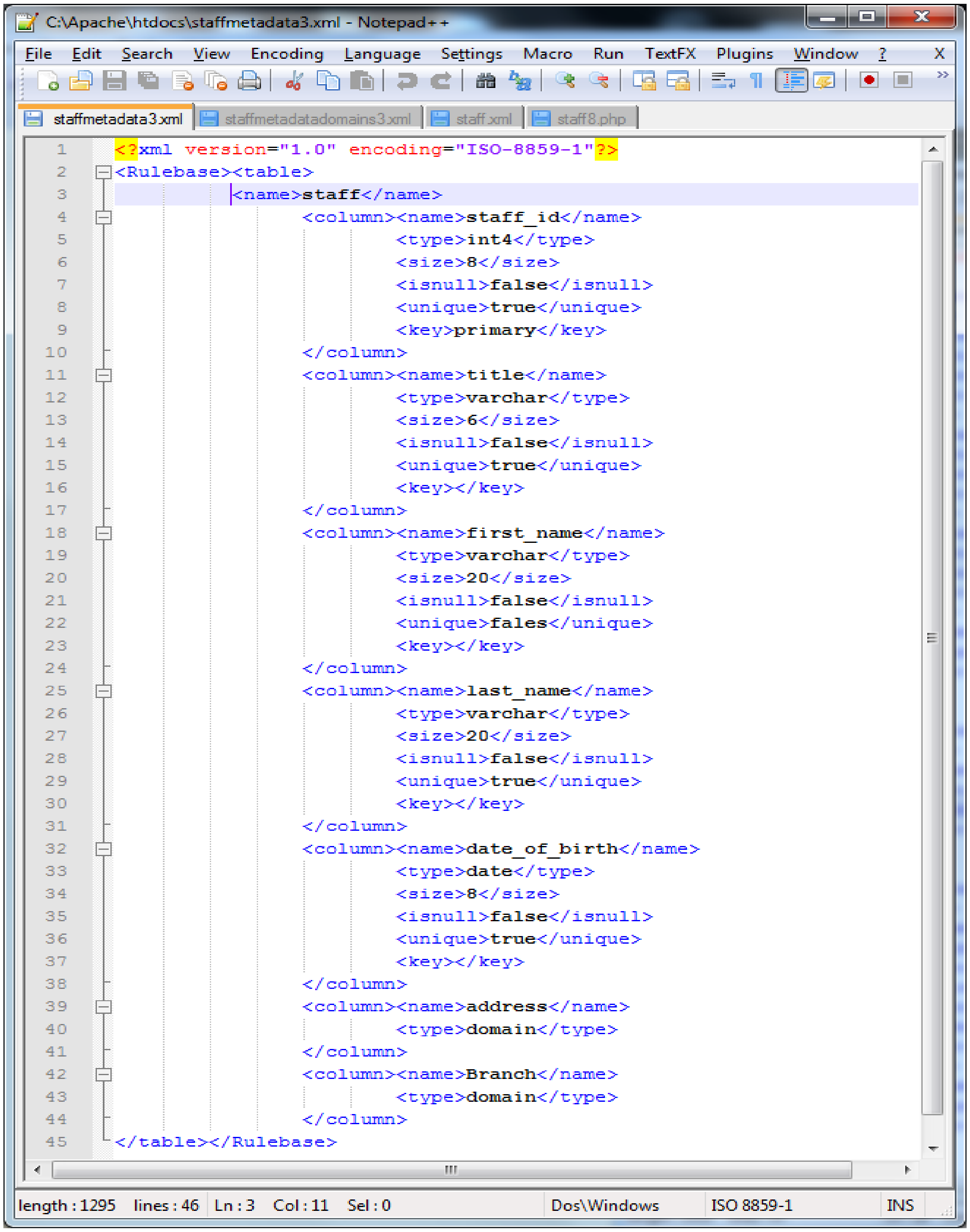
7.3.3 Existing table’s metadata stored as XML format

In this section we address how to store a table’s metadata in an XML format, particularly for systems that do not support domains and composite attributes. Database metadata can be represented in a XML file, this XML file uses XML tags to describe the tables and columns metadata, it is for modeling database information, so the metadata is stored into XML format.

7.3.3.1 Staff table metadata stored in XML format

The database metadata is stored in a XML format in separate files, as the example used in the prototype implementation the staff table metadata stored in XML file as shown in Figure 7-8. The XML file includes all the required information to (re) create the table in an RDBMSs whether it support domains or not. The tags organised to specify each column’s metadata in separate column tags. From the figure below the table staff consists of 8 columns the last two columns are created using domains, each column refers to a separate domain as below:

```
<column>
        <name>address</name>
        <type>domain</type>
</column>
<column>
        <name>Branch</name>
        <type>domain</type>
</column>
```


C:\Apache\htdocs\staffmetadata3.xml - Notepad++
File Edit Search View Encoding Language Settings Macro Run TextFX Plugins Window ?
staffmetadata3.xml
staffmetadatadomains3.xml
staff.xml
staff8.php

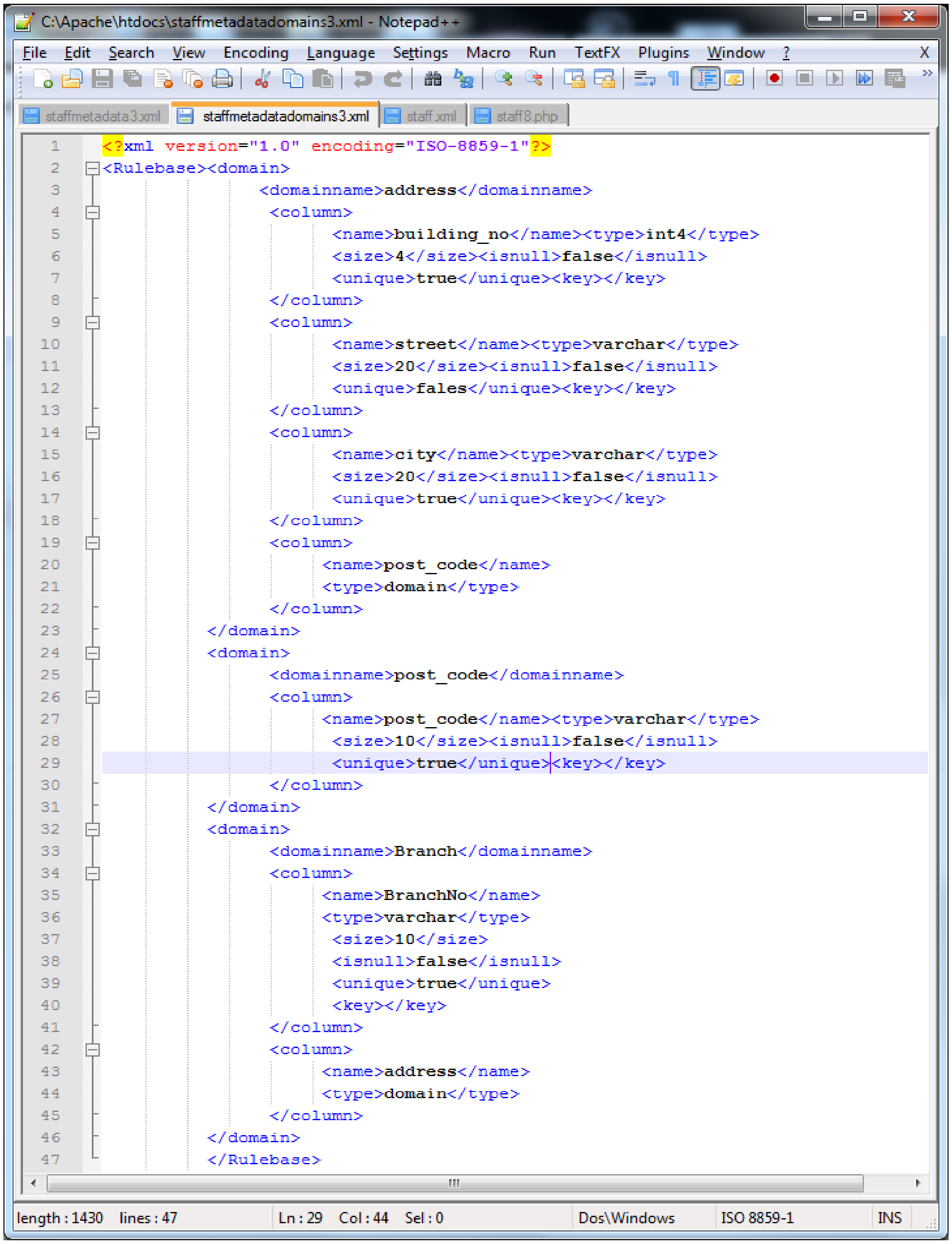
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Rulebase><table>
            <name>staff</name>
                <column><name>staff_id</name>
                        <type>int4</type>
                        <size>8</size>
                        <isnull>false</isnull>
                        <unique>true</unique>
                        <key>primary</key>
                </column>
                <column><name>title</name>
                        <type>varchar</type>
                        <size>6</size>
                        <isnull>false</isnull>
                        <unique>true</unique>
                        <key></key>
                </column>
                <column><name>first_name</name>
                        <type>varchar</type>
                        <size>20</size>
                        <isnull>false</isnull>
                        <unique>fales</unique>
                        <key></key>
                </column>
                <column><name>last_name</name>
                        <type>varchar</type>
                        <size>20</size>
                        <isnull>false</isnull>
                        <unique>true</unique>
                        <key></key>
                </column>
                <column><name>date_of_birth</name>
                        <type>date</type>
                        <size>8</size>
                        <isnull>false</isnull>
                        <unique>true</unique>
                        <key></key>
                </column>
                <column><name>address</name>
                        <type>domain</type>
                </column>
                <column><name>Branch</name>
                        <type>domain</type>
                </column>
</table></Rulebase>
```

length : 1295 lines : 46 Ln : 3 Col : 11 Sel : 0
Dos\Windows
ISO 8859-1
INS

Figure 7-8 Staff table metadata stored in XML format

7.3.3.2 Domain tables metadata in XML format

The database domain’s metadata and the structure of the composite attributes are stored in XML format as shown in Figure 7-9. Each domain in the previous XML file shown in Figure 7-8 is connected with the XML domains file shown in Figure 7-9. A domain can be used inside another one as shown in the address domain that contains a postcode column which is itself a domain. The structure of the post code column is also included in the domains file.

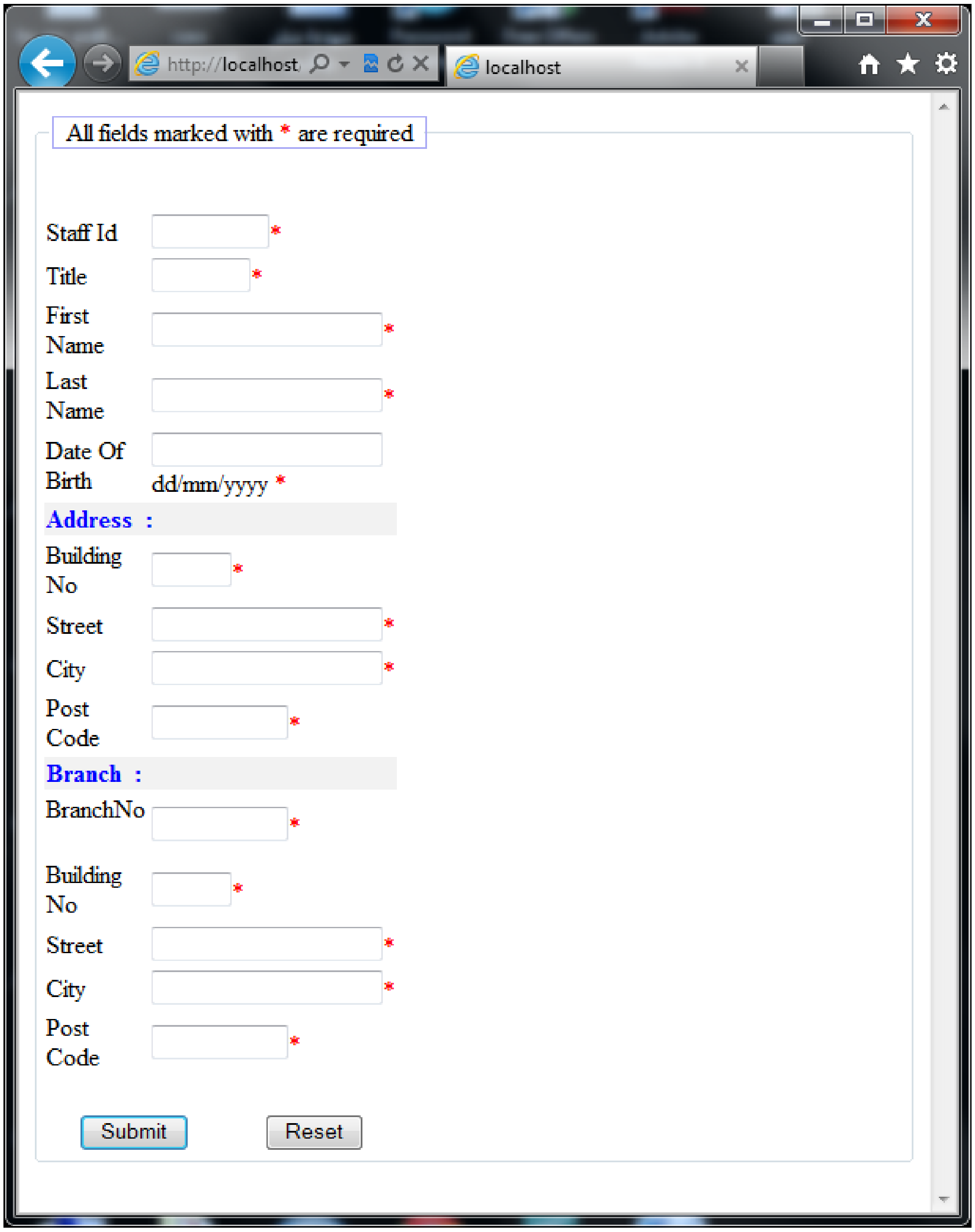

C:\Apache\htdocs\staffmetadatadomains3.xml - Notepad++
File Edit Search View Encoding Language Settings Macro Run TextFX Plugins Window ?
staffmetadata3.xml
staffmetadatadomains3.xml
staff.xml
staff8.php
length : 1430 lines : 47 Ln : 29 Col : 44 Sel : 0 Dos\Windows ISO 8859-1 INS

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Rulebase><domain>
              <domainname>address</domainname>
               <column>
                     <name>building_no</name><type>int4</type>
                     <size>4</size><isnull>false</isnull>
                     <unique>true</unique><key></key>
               </column>
               <column>
                     <name>street</name><type>varchar</type>
                     <size>20</size><isnull>false</isnull>
                     <unique>fales</unique><key></key>
               </column>
               <column>
                     <name>city</name><type>varchar</type>
                     <size>20</size><isnull>false</isnull>
                     <unique>true</unique><key></key>
               </column>
               <column>
                    <name>post_code</name>
                    <type>domain</type>
               </column>
         </domain>
         <domain>
               <domainname>post_code</domainname>
               <column>
                    <name>post_code</name><type>varchar</type>
                     <size>10</size><isnull>false</isnull>
                     <unique>true</unique><key></key>
               </column>
         </domain>
         <domain>
               <domainname>Branch</domainname>
               <column>
                    <name>BranchNo</name>
                    <type>varchar</type>
                     <size>10</size>
                     <isnull>false</isnull>
                     <unique>true</unique>
                     <key></key>
               </column>
               <column>
                    <name>address</name>
                    <type>domain</type>
               </column>
         </domain>
         </Rulebase>
```

Figure 7-9 Domain tables metadata in XML format

7.4 Generate the Web forms

We now demonstrate the use of the XML metadata format to generate the Web forms. By using the stored metadata files in conjunction with the RuleML Rulebase used in the previous chapters, as shown in Figure 5-3, a PHP script is written to loop through all the metadata for each column in every table. This and uses the RuleML rulebase to map each column to a Web form element on the fly. From the Web form generated which is shown in Figure 7-10 we can see how the composite columns' attributes are generated using the address domain table and also how the domain table can be used many times. Figure 7-10 shows the result of using address domain table twice, the first one is to generate the staffs address elements and the second one is to generate the branch address elements using the same domain. Additionally within each address the post code is itself another domain.



All fields marked with * are required

Staff Id	*
Title	*
First Name	*
Last Name	*
Date Of Birth	dd/mm/yyyy *

Address :

Building No	*
Street	*
City	*
Post Code	*

Branch :

BranchNo	*
Building No	*
Street	*
City	*
Post Code	*

Figure 7-10 User interface form generated automatically using metadata stored as XML format and rulebase as RuleML format.

7.5 Chapter Summary

We would like to specify all the semantics associated with data stored in RDBMS tables. XML Schema uses XML elements and attributes to express the structure of XML data, which may be comparable to RDBMS data, but XML Schema does not do everything. It can be used to express some limitations of data such as possible ranges of values and characteristics such as uniqueness. It does not have active elements which would allow us to express more behavior; however these can be found in an XML format in RuleML's Event-Condition-Action like elements.

To overcome some RDBMSs limitations RuleML is used to represent RDBMS data by storing database metadata in an external format, so it is also a way to overcome the differences between RDBMSs in areas such as whether they support domains and composites. Thus we propose a way to give a single syntax that can then map them to structures supported by a particular RDBMS and we test this by producing the same result for the Web form.

As a result a Web form for user interface is generated dynamically that corresponds to the database being used and at the same time maximises the use of semantics in metadata or elsewhere.

So XML Schema alone is not sufficient but by using a RuleML format we can go one stage father to implement more semantics for both database structures themselves and the Web forms built dynamically to access them.

Chapter 8

8 Conclusion and Future Work

8.1 Conclusion

Automatic and dynamic generation of Web forms is entering the mainstream in Web development for supporting developing online systems. This can be achieved by using database metadata, stored separately but preferably retrieved directly from the database. A set of rules is required to convert the facts in the metadata into information that can drive the form creation. 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.

This thesis has contributed towards the development of dynamic Web applications. The approach proposed separates the rules as an independent entity from the application code, by using a RuleML format as rulebase. A framework was proposed in this thesis to demonstrate the potential capabilities of Reaction RuleML to provide automatic and dynamic generation of Web forms.

Another extension to the work is to go beyond the basic information available from the database metadata and to develop two further kinds of rules: a more complete set of common sense rules and domain-specific rules. All these rules are then used to give consistency to the appearance of forms. The framework used common sense rules which supplemented the original database metadata rules and introduced a second rulebase of domain specific rules, which are invoking code to implement more semantics and to further separate between content, logic and presentation of a Web application. The development of rules to order and or group form elements was divided to two rulebases. The first one was applied to generate input forms and group the related attributes as blocks, and the second was applied to generate Web query forms that ordered the required attributes and controlled the form layout. So they helped in designing the query forms and include the most suitable form elements, which will be used to access the database.

XML Schema uses elements and attributes to express semantics of XML data, but XML Schema does not have active elements which RuleML has, like Event-Condition-Action elements. By using RuleML format to overcome some RDBMSs limitations, RuleML is used to represent RDBMS data by storing database metadata in an external format, so it is a way to overcome the differences between RDBMSs in areas such as whether they support domains and composites or not. Thus we propose a way to give a single syntax that maps them and produces the same result for the Web form. As a result a sophisticated Web form for user interface is generated dynamically.

The system evaluation started by using Reaction RuleML 0.1format, it then used Reaction RuleML 0.2 format as an improved version of RuleML 0.1 to store the developed rules. In Reaction RuleML 0.2 more detailed tags are provided with more meaningful tags naming.

So it was successfully implemented using Reaction RuleML format to store the developed rules, the technologies to support this were PostgreSQL as a DBMS, PHP for server side programming, HTML and JavaScript for client side programming. These are typical systems and equivalent features are available which mean that the proposed framework could also be deployed in situation that used alternative equivalent technologies such as ASP.net and Flash.

PostgreSQL was used to show both standard SQL features and the additional features available with some advanced systems.

As a result a Web forms for user interface is generated dynamically. This approach aims to use as generic rulebase as possible using RuleML.

8.2 Future Work

Although the research presented in this thesis is promising and positive, a number of related issues were raised in the course of the work which could be developed in future stages. Some of these are:

- Performance evaluation

In this work no performance issues have been noted but this is recommended as further work 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 and we also accumulate the metadata we need into a single object and obtain it once and reuse it if needed, this has lead to small RuleML files which are not complex to parse.

- User data validation

In principle we could use metadata integrity constraint rules and the RuleML rulebases to validate user input data. This could be implemented in various ways. This work does not focus on form validation; a wide range of methods have been used during previous work by many people to validate user data entered, for example using XForms or JavaScript and in the future developers will be looking at features built into HTML5.

- Rule engines

Rule engines are software systems that execute or fire rules in a runtime environment. It would be possible to develop a rule engine for metadata common sense and domain specific rulebases to support the management of the rules, execution and dynamic change. A rule engine could also be used to check for conflicts, inconsistencies or gaps.

- Automatic rule creation and update

In conjunction with a rule engine it may be possible to develop a system to allow less experienced users to create rules in an interactive fashion. In the frame work as implemented users must build some rules by hand and require an understanding of XML in general and the particular RuleML format used. A system to create rules dynamically could take control of the rules and allow users with less experience to use the system in a less error prone way.

- Forms presentation

In our work we can determine the order of the blocks but at present the order within the block depends on the order the columns are defined in the database. In a relational database the columns do not have an ordering and unless one is imposed by for example an SQL query they appear by default in the order they were defined in the database. Extension of the ordering work done could allow user to specify form element ordering in more detail.

- Development with other types of database

In our current implementation we have used database metadata which in this case is derived from relational database system catalogues, but could be obtained from other sources such as XML Schema and used in conjunction with XML data.

References

[1] M. M. Elsheh and M. J. Ridley, "GENERATION AND VALIDATION OF WEB FORMS USING DATABASE METADATA AND XFORMS," in *Symposium on Progress in Information and Communication Technology(SPICT 10)*, pp. 23-26.

[2] N. Skrupsky, M. Monshizadeh, P. Bisht, T. Hinrichs, V. Venkatakrishnan, and L. Zuck, "WAVES: Automatic Synthesis of Client-side Validation Code for Web Applications," *ASE Science Journal*, 2012.

[3] System Catalogs. Available: http://www.postgresql.org/docs/8.1/static/catalogs.html. [Accessed: 12 Dec. 2012].

[4] M. M. Elsheh and M. J. Ridley, "Using Database Metadata and its semantics to Generate Automatic and Dynamic Web Entry Forms," in *Proceedings of WCECS 2007 World Congress on Engineering and Computer Science 2007*, pp. 654-658.

[5] V. Turau, "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, pp. 1121-1126.

[6] Cascading Style Sheets home page. Available: http://www.w3.org/Style/CSS/. [Accessed: 15 Dec. 2012].

[7] The Rule Markup Initiative. Available: http://ruleml.org/. [Accessed: 12 Dec. 2012].

[8] L. Bertossi and G. Jayaraman, A. Hameurlain and A. Tjoa "Designing, Specifying and Querying Metadata for Virtual Data Integration Systems Data Management in Grid and Peer-to-Peer Systems," *Lecture Notes in Computer Science*, vol. 5697, A. Hameurlain and A. Tjoa, Eds. Berlin: Springer 2009, pp. 72-84.

[9] S. K. Cabral and K. Murphy, *MySQL administrator's bible*: Wiley, 2011.

[10] SQL: Practical Guide for Developers MySQL. Available: http://cs.ecs.baylor.edu/~donahoo/practical/sql/details-mysql.html. [Accessed: 12 Dec. 2012].

[11] T. M. Connolly and C. E. Begg, *Database systems: a practical approach to design, implementation, and management*, 5th ed. Boston: Pearson Education international, 2010.

[12] B. Eaglestone and M. Ridley, *Web Database Systems*. London: McGraw Hill, 2001.

[13] D. D. Chamberlin and R. F. Boyce, "SEQUEL: A structured English query language," in *Proceedings of the 1974 ACM SIGFIDET (now SIGMOD) workshop on Data description, access and control*, 1974, pp. 249-264.

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

[15] A history of HTML. Available: http://www.w3.org/People/Raggett/book4/ch02.html. [Accessed: 20 Dec. 2012].

[16] M. Dubinko, *XForms essentials*. Sebastopol, Calif ; Farnham: O'Reilly, 2003.

[17] HTML Versions. Available: http://www.w3schools.com/w3c/w3c_html.asp. [Accessed: 12 Dec. 2012].

[18] Introducing HTML 3.2. Available: http://www.w3.org/MarkUp/Wilbur/. [Accessed: 12 Dec. 2012].

[19] W3C XHTML Activities. Available: http://www.w3schools.com/w3c/w3c_xhtml.asp. [Accessed: 12 Dec. 2012].

[20] T. Felke-Morris, *Web Development and Design Foundations with XHTML*, 4th ed: Pearson, 2009.

[21] HTML5 differences from HTML4. Available: http://dev.w3.org/html5/html4-differences/. [Accessed: 12 Dec. 2012].

[22] J. Hjelm, *Creating the semantic Web with RDF: professional developer's guide*: John Wiley & Sons, Inc., 2001.

[23] Introduction to RDF. Available: http://www.w3schools.com/rdf/rdf_intro.asp. [Accessed: 18 Apr. 2013].

[24] Scripting and Ajax. Available: http://www.w3.org/standards/webdesign/script. [Accessed: 12 Dec. 2012].

[25] JavaScript Introduction. Available: http://www.w3schools.com/js/js_intro.asp. [Accessed: 12 Dec. 2012].

[26] ECMAScript Language Specification. Available: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf. [Accessed: 12 Dec. 2012].

[27] AJAX Introduction. Available: http://www.w3schools.com/ajax/ajax_intro.asp. [Accessed: 12 Dec. 2012].

[28] B. McLaughlin, *Head Rush Ajax*, 1st ed. Sebastopol: O'Reilly Media, 2006.

[29] AutoComplete Demonstration. Available: http://www.asp.net/ajaxLibrary/AjaxControlToolkitSampleSite/AutoComplete/AutoComplete.aspx. [Accessed: 12 Dec. 2012].

[30] Introducing JSON. Available: http://www.json.org/. [Accessed: 17 Apr. 2013].

[31] JSON Tutorial. Available: http://www.w3schools.com/json/default.asp. [Accessed: 17 Apr. 2013].

[32] JSON: The Fat-Free Alternative to XML. Available: http://www.json.org/fatfree.html. [Accessed: 30 Apr. 2013].

[33] D. Guinard and V. Trifa, "Towards the web of things: Web mashups for embedded devices," in *Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), in proceedings of WWW (International World Wide Web Conferences), Madrid, Spain*, 2009.

[34] W. Guanhua, "Improving Data Transmission in Web Applications via the Translation between XML and JSON," in *Communications and Mobile Computing (CMC), 2011 Third International Conference on*, 2011, pp. 182-185.

[35] S. Gundavaram, *CGI Programming on the World Wide Web*: O'Reilly, 1996.

[36] R. L. Schwartz, T. Phoenix, and B. D. Foy, *Learning Perl*, 4th ed: O'Reilly Media, Inc, 2005.

[37] C. Bates, *Web programming: building Internet applications*, 3rd ed. Chichester: Wiley, 2006.

[38] What can PHP do. Available: http://www.php.net/manual/en/intro-whatcando.php. [Accessed: 12 Dec. 2012].

[39] Active Server Pages Technical Articles. Available: http://msdn.microsoft.com/en-us/library/ms972347.aspx. [Accessed: 12 Dec. 2012].

[40] J. Goodwill, *Developing Java^{TM} Servlets*: Sams, 2001.

[41] JavaServer Pages Technology - Frequently Asked Questions. Available: http://java.sun.com/products/jsp/faq.html. [Accessed: 12 Dec. 2012].

[42] JavaServer Pages: A Developer's Perspective. Available: http://java.sun.com/developer/technicalArticles/Programming/jsp/. [Accessed: 12 Dec. 2012].

[43] XML1.0 W3C Recommendation. Available: http://www.xml.com/axml/testaxml.htm. [Accessed: 12 Dec. 2012].

[44] Introduction to XML. Available: http://www.w3schools.com/xml/xml_whatis.asp. [Accessed: 12 Dec. 2012].

[45] Introduction to XML Schema. Available: http://www.w3schools.com/schema/schema_intro.asp. [Accessed: 19 Dec. 2012].

[46] XSLT Introduction. Available: http://www.w3schools.com/XSL/xsl_intro.asp. [Accessed: 12 Dec. 2012].

[47] D. Deitel, Nieto, Lin & Sadhu, *XML HOW TO PROGRAM*: Prentice Hall, 2001.

[48] XML Path Language (XPath). Available: http://www.w3.org/TR/xpath. [Accessed: 12 Dec. 2012].

[49] XPath Introduction. Available: http://www.w3schools.com/XPath/xpath_intro.asp. [Accessed: 12 Dec. 2012].

[50] XPath Syntax. Available: http://www.w3schools.com/XPath/xpath_syntax.asp. [Accessed: 12 Dec. 2012].

[51] XML Pointer Language. Available: http://www.w3.org/TR/WD-xptr. [Accessed: 16 Jan. 2013].

[52] XQuery 1.0: An XML Query Language (Second Edition). Available: http://www.w3.org/TR/xquery/. [Accessed: 17 Jan. 2013].

[53] Rule Markup Languages and Semantic Web Rule Languages. Available: http://nparc.cisti-icist.nrc-cnrc.gc.ca/npsi/ctrl?action=shwart&index=an&req=18533385&lang=en. [Accessed: 12 Dec. 2012].

[54] H. Boley, S. Tabet, and G. Wagner, "Design rationale of RuleML: A markup language for semantic web rules," in *International Semantic Web Working Symposium (SWWS)*, 2001, pp. 381-402.

[55] RuleML Primer. Available: http://ruleml.org/papers/Primer/RuleMLPrimer2012-08-09/RuleMLPrimer-p0-2012-08-09.html. [Accessed: 12 Dec. 2012].

[56] H. Boley, A. Paschke, and O. Shafiq, M. Dean, J. Hall, A. Rotolo, and S. Tabet "RuleML 1.0: The Overarching Specification of Web Rules," in *Semantic Web Rules*, *Lecture Notes in Computer Science*, vol. 6403, M. Dean, J. Hall, A. Rotolo, and S. Tabet, Eds.: Springer Berlin Heidelberg, 2010, pp. 162-178.

[57] RuleML Design. Available: http://ruleml.org/indesign.html. [Accessed: 12 Dec. 2012].

[58] 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, 2004, pp. 628-631.

[59] Version History, 2001-01-25: Version 0.7. Available: http://ruleml.org/0.7/. [Accessed: 12 Dec. 2012].

[60] A. Paschke, H. Boley, A. Kozlenkov, and B. Craig, "Rule responder: RuleML-based agents for distributed collaboration on the pragmatic web," in *Proceedings of the 2nd international conference on Pragmatic web*, 2007, pp. 17-28.

[61] H. Boley, "The RuleML family of web rule languages," in *Proceedings of the 4th International Conference on Principles and Practice of Semantic Web Reasoning*, 2006, pp. 1-17.

[62] Schema Specification of RuleML 0.91. Available: http://ruleml.org/0.91/. [Accessed: 12 Dec. 2012].

[63] Schema Specification of Deliberation RuleML Version 1.0. Available: http://ruleml.org/1.0/. [Accessed: 12 Dec. 2012].

[64] Reaction RuleML. Available: http://ruleml.org/reaction/. [Accessed: 12 Dec. 2012].

[65] A. Paschke, A. Kozlenkov, H. Boley, M. Kifer, S. Tabet, M. Dean, and K. Barrett. Reaction RuleML. Available: ruleml.org/reaction/docs/RuleML06_Poster.pdf. [Accessed: 21 Dec. 2012].

[66] Reaction RuleML Tutorial. Available: http://ruleml.org/reaction/docs/Reaction-RuleM_tutorial06.pdf. [Accessed: 12 Dec. 2012].

[67] Reaction RuleML 0.2. Available: http://ruleml.org/reaction/0.2/index.htm. [Accessed: 12 Dec. 2012].

[68] SWRL RuleML. Available: http://ruleml.org/swrl/. [Accessed: 12 Dec. 2012].

[69] SWRL: A Semantic Web Rule Language Combining OWL and RuleML. Available: http://www.w3.org/Submission/SWRL/. [Accessed: 12 Dec. 2012].

[70] OWL Web Ontology Language Overview. Available: http://www.w3.org/TR/owl-features/. [Accessed: 19 Dec. 2012].

[71] R2ML -- The REWERSE I1 Rule Markup Language. Available: http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=R2ML. [Accessed: 12 Dec. 2012].

[72] RIF Overview. Available: http://www.w3.org/TR/rif-overview/. [Accessed: 12 Dec. 2012].

[73] T. Nguyen 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, pp. 229-540.

[74] M. Papiani, 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.

[75] M. A. Mgheder and M. J. Ridley, "Automatic Generation of Web User Interfaces in PHP Using Database Metadata," in *Proceedings of Internet and Web Applications and Services, 2008. ICIW '08. Third International Conference on*, 2008, pp. 426-430.

[76] S. J. Halasz, "An improved method for creating dynamic web forms using APL," in *Proceedings of the International Conference on APL-Berlin*, 2000, pp. 104-111.

[77] M. S. Mark Weiner, Abigail Cohen, "Metadata tables to enable dynamic data modeling and web interface design: the SEER example," *International Journal of Medical Informatics*, vol. 65, p. 51, 2002.

[78] A. Elbibas and M. J. Ridley, "Developing Web Entry Forms Based on METADATA," Presented at International Workshop on Web Quality in conjunction with ICWE 04- International Conference on Web Engineering. Available: http://www.pst.informatik.uni-muenchen.de/~baumeist/icwe/ws/ws1/icwe04.pdf. [Accessed: 21 Dec. 2012].

[79] M. Bernauer, G. Kappel, and G. Kramler, "Approaches to implementing active semantics with XML schema," in *Database and Expert Systems Applications, 2003. Proceedings. 14th International Workshop on*, 2003, pp. 559-565.

[80] E. Kirda, C. Kerer, and G. Matzka, "Using XML/XSL to Build Adaptable Database Interfaces for Web Site Content Management," in *Proceedings of the XML in Software Engineering Workshop (XSE 2001), 23rd International Conference on Software Engineering (ICSE 2001)*, 2001.

[81] T. Schmidberger and A. Fay, "A rule format for industrial plant information reasoning," in *Emerging Technologies and Factory Automation, 2007. ETFA. IEEE Conference on*, 2007, pp. 360-367.

[82] Understanding JDBC Metadata. Available: http://eye-on-objects.com/c_brown.htm. [Accessed: 12 Dec. 2012].

[83] T. J. Teorey, D. Yang, and J. P. Fry, "A logical design methodology for relational databases using the extended entity-relationship model," *ACM Comput. Surv.*, vol. 18, pp. 197-222, 1986.

[84] PostgreSQL Functions. Available: http://www.php.net/manual/en/ref.pgsql.php. [Accessed: 22 Mar. 2013].

[85] A. M. Albhbah and M. J. Ridley, "Using RuleML and database metadata for automatic generation of web forms," in *Proceedings of 10th International Conference on Intelligent Systems Design and Applications (ISDA)* 2010, pp. 790-794.

[86] A. M. Albhbah and M. J. Ridley, "A Rule Framework for Automatic Generation of Web Forms," in *Proceedings of 4th IEEE International Conference on Computer Science and Information Technology (IEEE ICCSIT 2011)*, 2011, vol. 5-A608, pp. 76 - 81.

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

[88] Using domains. Available: http://dcx.sybase.com/1200/en/dbusage/domains-integrit.html. [Accessed: 12 Dec. 2012].

[89] PostgreSQL extended or object relational features. Available: http://www.comp.brad.ac.uk/~postgres/postgreSQL/worksheet5.html. [Accessed: 12 Dec. 2012].

[90] PostgreSQL: The world's most advanced open source database. Available: http://www.postgresql.org/. [Accessed: 12 Dec. 2012].