Knowledge Extraction from Web-based Application Source Code: An Approach to Database Reverse Engineering for Ontology Development

Shuxin Zhao, Elizabeth Chang, Tharam Dillon
Digital Ecosystems & Business Intelligence Institute, Curtin University of Technology
{S.Zhao,E.Chang,Tharam.Dillon@curtin.edu.au}

Abstract

This paper presents a novel approach for extracting knowledge from web-based application source code in supplementing and assisting ontology development from database schemas. The structure of web-based application source code is defined in order to distinguish different kinds of knowledge within the source code for ontology development. The connections between the relevant parts of web application source code and the backend database schema with their various forms are explicitly specified in detail. A knowledge processing and integration model for extracting and integrating the knowledge embedded in the source code for ontology development is then proposed.

1. Introduction

Although research in ontology-based technologies has been a hot topic over the last decade, the difficulty of developing domain ontologies in an effective and automated manner still exists. The high cost of developing ontologies in terms of time, effort and resource remains high, in particular, the cost associated with knowledge acquisition has become a well-known bottleneck in the ontology development process. "*Ontology learning*" [1], has emerged and it aims to develop domain ontologies by automatically acquiring and transforming knowledge from existing resources such as text, dictionary, knowledge base, semi-structure data and structured data [1]. Amongst these ontology learning sources, relational databases have many advantages for ontology development given the large number of such databases, the richness of the data captured, the data structures and up-to-date data instances etc. Some approaches have been proposed for developing ontologies from relational databases [2-5] and their primary sources for learning ontologies are the database schemas.

However, using databases as the only source for ontology development has many limitations for the resulting ontologies. There are two main limitations: *firstly*, knowledge captured in the database is incomplete as knowledge may be lost during database implementation or DBMSs can only accommodate certain types of knowledge. For example, the knowledge about calculating the total amount of a customer purchase order, based on products information, price of products and tax, cannot be represented in a relational database. There are also many business rules associated with the data held in the database that are often beyond the definition of databases; *secondly*, databases have a severe naming problem. The terms used to denote relations and their attributes in databases are mostly abbreviations, acronyms or arbitrary variables which often make little sense. Consequently, concepts of the ontology derived from those poorly named relations and attributes will reflect little semantics.

To tackle these issues, applications that are associated with databases can be utilized for assisting and supplementing ontology development from the databases. This is because databases are not developed in isolation, rather they are designed and developed to provide a persistent data repository in the backend for a particular application built upon it. The application source code mediates the data held in the backend database and the user understandable and meaningful terms at the user interface by interpreting and transforming the data into their "intended meanings" [6].

By analyzing application source code one can extract the meaningful terms from user interface and to track down the links between the terms to the backend database. There are two ways that the application source code can be used to assist database reverse engineering for ontology development. *Firstly*, it can assist to verify and identify the semantics of concepts derived from a poorly named database schema. Application source code embeds meaningful terms of the data held in backend databases and interprets relation and attributes into their intended meanings to the user interfaces. *Secondly*, application source code itself has buried in it business rules and domain knowledge which can be further identified and added to the knowledge acquired from the database schema.

Therefore, we propose a novel approach which aims to automatically extract knowledge from web-based application source code in supplementing and assisting domain ontology development from the database schema.

978-1-4244-2660-7/08/$25.00 ©2008 IEEE

In this paper, the structure of the web-based application is examined and defined. Each part of the defined structure of the application source code is explicitly specified to explore how it integrates with the backend database schema and an initial ontology derived from the backend database schema. The rest of this paper is organized as the follows: Section 2 provides an overview of previous work on ontology development from relational databases; Section 3 defines a web-based application and its structure for the purpose of assisting ontology learning from databases; Section 4 presents our approach for extracting domain knowledge from application source code and then integrating the knowledge with the relational schema of the backend database; Section 5 concludes the paper and indicates future work.

2. Related work

The Research in ontology development or ontology learning from relational databases aims to construct ontologies through an automated process of acquiring and transforming knowledge embedded in databases. Previous approaches in this area can be categorized into two groups. The first group of approaches includes Kashyap [2], Stojanovic, Stojanovic & Volz [3], Astrova [4] which firstly create an abstract ontology model from one or more database schemas then populate ontology instances of the ontology model from the data instances of the database(s). Two common techniques are employed in this type of approach. *Firstly*, a reverse engineering technique is used to analyze the input database schema in order to extract a conceptual model in EER such as in [7]; *secondly*, a mapping technique is used to map the extracted conceptual model of the input database into an ontology language such as F-logic [8] and OWL [9]. The dominant source for identifying major concepts, properties of concepts and relationships between concepts of the target ontology is the relational schema. A relational schema specifies the structure and constraints of the data held in a relational database. It consists of the following constructs that can be used for ontology development:

- Relations (aka. Tables)
- Attributes of Relations
- Data types of attributes
- Constraints of attributes such as unique, not null
- Primary keys and Foreign keys

Key correlation of the input schema is mainly used for identifying relationships between concepts.

Although there is a clear connection between the application source code and its backend database, this important source for identifying and verifying domain knowledge is rarely considered in previous approaches for ontology development from databases [10]. Some approaches considered using user queries [2] to refine the ontology, but few have utilized the application source code. Only [11] proposed using part of the user interface, namely only the "HTML Forms", from the entire application source code to extract a "form model schema" and the extraction is manually conducted by domain experts.

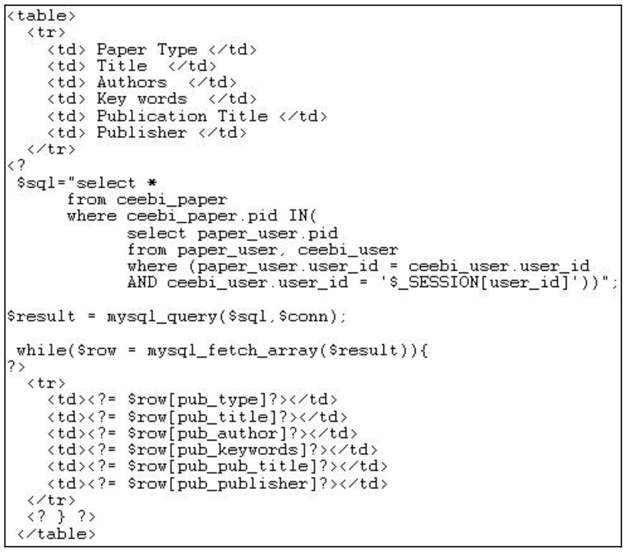
The second group of approaches for developing ontology from relational databases proposed mapping languages that directly map database instances into Semantic Web ontology syntax such as RDF [12] and OWL [9]. This group includes R2O [13] and D2R MAP [14]. The main drawbacks of this type of approaches are twofold: firstly, they ignore the fact that database instances will keep updating over time, as a result, constant synchronization between the published ontological instances and the data in the original database is required; secondly, the semantics of the original database model are less reflected in the transformed ontology instances. However, these languages can be used effectively to populate ontology instances from databases when an ontology model of that database is available.

3. The web-based application and its structure

It is important to have a well-defined notion of a web-based application in order to examine it for assisting ontology development from the database schema. In this section, we define web-based application and its structure, and examine to ascertain it can be used in conjunction with database schema for ontology development.

Web-based applications, in general, consist of a backend database and dynamic server pages which generate web content dynamically from the backend database. These server pages are implemented using web-based programming languages and are compiled and interpreted by vendor specific compilers and HTTP servers such as "Apache", "IIS" and "TOMCAT". Some of the popular web programming languages include JSP, ASP, PHP etc. These languages are encoded in a mixture with HTML tags whether they are hard coded or dynamically generated. In general in this mixed structure, HTML tags are used for rendering displaying layout, format and navigation; the web programming language is in charge of data entry/retrieval and data manipulations between the information displayed at the user interface and the backend database. *Figure 1*below illustrates this mixed structure.

In this sense, a web-based application source code can be logically divided into *User Interface* (UI for abbreviation) and *Data Manipulation Code* (DMC for abbreviation) whether they may be physically contained within one application source code file or not. Therefore, we define an abstracted structure model of web-based application source code for the purpose of knowledge extraction for ontology development and for specifying how each part of the structure can be used for the ontology learning from databases. Thus, this structure definition would differ from the structure concept of application maintainability and scalability for web-based application design such as the common 3-tier architecture. This model is depicted in *Figure 2* and is explained in the following two subsections.



```
<table>
  <tr>
    <td> Paper Type </td>
    <td> Title  </td>
    <td> Authors  </td>
    <td> Key words  </td>
    <td> Publication Title </td>
    <td> Publisher </td>
  </tr>
<?
 $sql="select *
      from ceebi_paper
      where ceebi_paper.pid IN(
            select paper_user.pid
            from paper_user, ceebi_user
            where (paper_user.user_id = ceebi_user.user_id
            AND ceebi_user.user_id = '$_SESSION[user_id]'))";

$result = mysql_query($sql,$conn);

 while($row = mysql_fetch_array($result)){
?>
  <tr>
    <td><?= $row[pub_type]?></td>
    <td><?= $row[pub_title]?></td>
    <td><?= $row[pub_author]?></td>
    <td><?= $row[pub_keywords]?></td>
    <td><?= $row[pub_pub_title]?></td>
    <td><?= $row[pub_publisher]?></td>
  </tr>
  <? } ?>
 </table>
```

Figure 1. Sample PHP code segment screenshot

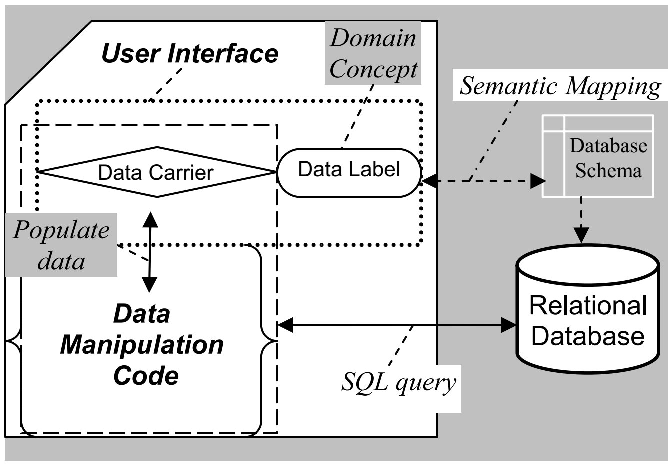


Figure 2. Abstracted structure model of web-based application source code

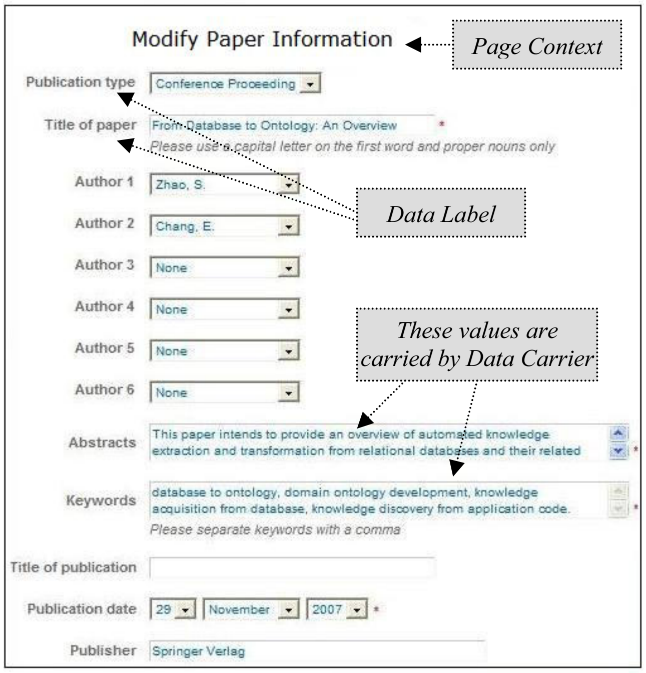
3.1. User Interface (UI)

The User Interface navigates users' interactions with an application by providing user-friendly navigations and instructions, and by accepting users' inputs and displaying user requested outputs. UIs are generally encoded in HTML. For the purpose of knowledge acquisition from application source code, we classify possible semantic sections of the user interface as the following:

- *Displaying layout*. It provides and arranges layout, format for all the contents in a web page. This section is useful when extracting knowledge contained in a web page but itself does not contain any domain knowledge. The section is encoded in HTML.
- *Content navigation among web pages*. The navigation is a set of hyperlinks encoded by HTML <a> tags. The navigation may indicate the file dependencies and the structure of the web application but is the same as displaying layout that itself does not contain any domain knowledge.
- *Page (content) context*. In general, every single web page displays content about one particular topic except the homepage(s). The topic is the page context such as "Paper List" page lists all papers that are published, "My Information" page displays personal details and "Modify Paper Information" displays all information about one paper in an editable format. The page context determines the interpretation of the content displayed in a web page. The page context is usually rendered at or near the top position of a web page as a headline or heading using a short phrase. Page context is encoded in plain text.
- *Page content*. Page content is the information and data displayed in a web page and is the major target for knowledge acquisition from application source code. We distinguish two types of page content in web-based applications: the static ones and the dynamic ones. Purely static content in a web page is disregarded as there is no connection specified between this type of content and the backend database. It is, however, a subject of information extraction and web data mining which is beyond our focus currently. What we are interested in here is the dynamic content which is generated from the backend database of a web application. This type of content can be mapped to the database schema of the backend database through *Data Manipulation Code* via SQL queries as shown in *Figure 2*. To exploit this type of content in web application source code, we first need some terminology, specifically the pair of *Data Label* and *Data Carrier*.

Data Label: is a short phrase encoded in plain text that is used to indicate the semantics of the dynamically generated content at user interface. Data Labels are usually presented as labels, headings, table headers or labels of HTML form controls at the User Interface. Data Labels are the "intended meanings" of the data stored in the backend database. Every Data Label can be mapped to one or one set of dynamically generated data values and is rendered virtually adjacent to its pair Data Carrier. In *Figure 3*, "Publication Type" and "Title of Paper" are examples of the Data Labels.

Data Carrier: We use the term Data Carrier to denote the variables defined in the web programming language meaning that it carries the actual data value to its paired Data Label. The value of Data Carriers is assigned from the backend database by other data manipulation code and is displayed as the dynamic content. The Data Carrier in the source code is populated with actual data retrieved from the backend database before the web page is sent from the web server to the requesting client browser. In *Figure 1* in the previous section, "$row[pub_type]" and "$row[pub_title]" are examples of the Data Carrier. Data Carrier is the joint of the User Interface and Data Manipulation Code as shown in the source code structure model in *Figure 2*.



Modify Paper Information

Publication type	Conference Proceeding
Title of paper	From Database to Ontology: An Overview * *Please use a capital letter on the first word and proper nouns only*
Author 1	Zhao, S.
Author 2	Chang, E.
Author 3	None
Author 4	None
Author 5	None
Author 6	None
Abstracts	This paper intends to provide an overview of automated knowledge extraction and transformation from relational databases and their related *
Keywords	database to ontology, domain ontology development, knowledge acquisition from database, knowledge discovery from application code. * *Please separate keywords with a comma*
Title of publication	
Publication date	29 November 2007 *
Publisher	Springer Verlag

Figure 3. An user interface screenshot

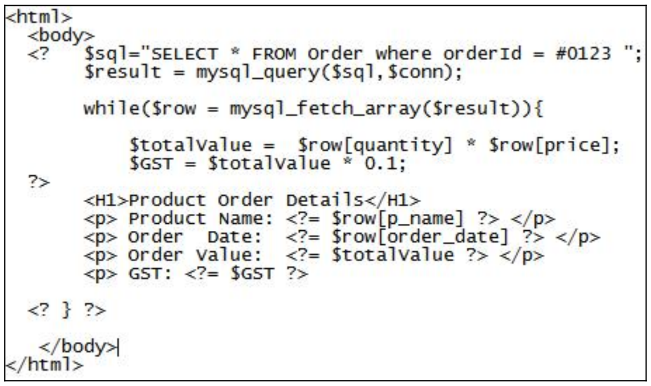
3.2. Data Manipulation Code (DMC)

The Data Manipulation Code, on the other hand, is in charge of data interpretation and transformation between the display at the user interface and the data held in the backend database. It populates the Data Carrier with data instances of the backend database or fetches user inputs from Data Carrier at the user interface updating them into the backend database. Data Manipulation Code retrieves and updates data in the database by executing hard coded SQL queries or predefined stored procedures. Attributes and relations in the backend database are explicitly specified in the SQL statement. Therefore, a connection between the Data Label and database schema can be established by tracing the DMC and analyzing SQL queries. DMC is encoded using the web programming language whose syntax can be easily distinguished from those of HTML tags.

There are basically three ways that a Data Label can be linked to the relational schema of the backend database via its pair Data Carrier in the data manipulation code. They are, namely, *direct link*, *indirect link* and *user input validation*:

3.2.1 The Direct Link. In direct links, Data Carriers are populated directly with the result of a SQL query. So that one Data Label or one set of Data Labels can be mapped to one attribute or one set of attributes of some relations of the backend database. In *Figure 4*, the Data Carrier "$row[p_name]" in the code conveys the actual value of the Data Label "Product Name" and is populated with the SQL query result. Therefore, a direct link can be built between "Product Name" and the attribute "p_name" of the relation "Order" in the database schema. The direct link can be generalized as:

{Data Label ←→ Data Carrier} ←→ SQL Query ←→ Attribute of Relation



```
<html>
  <body>
  <?   $sql="SELECT * FROM Order where orderId = #0123 ";
       $result = mysql_query($sql,$conn);

       while($row = mysql_fetch_array($result)){

           $totalValue =  $row[quantity] * $row[price];
           $GST = $totalValue * 0.1;
  ?>
       <H1>Product Order Details</H1>
       <p> Product Name: <?= $row[p_name] ?> </p>
       <p> Order  Date:  <?= $row[order_date] ?> </p>
       <p> Order Value:  <?= $totalValue ?> </p>
       <p> GST: <?= $GST ?>

  <? } ?>

   </body>
</html>
```

Figure 4. PHP source code segment showing direct link and indirect link between Data Label and backend database

3.2.2 The Indirect Link. In the indirect link, on the other hand, Data Carrier is populated by the result of extra data manipulation based on one or more SQL query results and/or with some hard coded pieces of knowledge. We call this kind of data manipulation code as Data Transformation Code (DTC). It represents knowledge transformation. This type of knowledge is usually beyond the definition of relational schemas which also need to be captured in the target ontology developed from database schema. The Data Transformation Code includes:

• *Calculation*: It is mathematical expression whose operands are derived from the attributes of the relations in the backend database which can be tracked from related SQL queries. For example, in the source code shown in *Figure 4*, in the statement "$totalValue = $row[quantity] * $row[price]", the operands "$row[quantity"] and

$row[price] are the attribute “Quantity” and “Price” of the relation “Order” respectively which are specified in the SQL query “$sql” above. Another example is the knowledge about GST tax calculation in the source code, that is “GST = totalValue * 0.1”, where “0.1” is hard coded knowledge representing tax rate.

• *Combination*: Concatenation of strings such as “Title + FirstName + LastName” or a phone number is a concatenation of “country code + area code + phone number”.

The indirect links can be illustrated as:
{Data Label ←→ Data Carrier} ←→ {DTC:Calculation;Combination of Multiple Operands} ←→ SQL Query ←→ Attributes of Relations

3.2.3 User Input Validation. There are validations of user input among DMC, in addition to the direct and indirect links, that may also indicate associations between Data Labels and the attributes of relations in the database schema. Two types of validations of user input are differentiated. One is the validation of data type such as the input of a date of birth must comply with a date data type or phone number must contain only numeric characters. However, this type of knowledge has already been specified in relational schemas as the data types of attributes. Thus we can ignore this type of validation code to avoid unnecessary duplication. The other type of validation is the validation of the value range of the input data. For instance, the age limit for employee recruitment in an organization must be between 18 and 60. The validation includes the validation code in the web programming language and also JavaScript. The links between a Data Label and an attribute of a relation via validation code can be illustrated as the following:

{Data Label ←→ Data Carrier} ←→ {Validation: Data value range} ←→ SQL Query ←→ Attribute of Relation

We distinguish between these three kinds because they will be used differently in the process of refining and improving the original database schema and the resulting ontology developed from the database schema.

4. Knowledge extraction from web-based application source code

The purpose of defining the structure of web-based application source code in the previous section is to identify and examine the relevant knowledge from the source code in order to assist and supplement ontology development from databases. In this section, we introduce our model for knowledge extraction from web-based application source code and for integrating the extracted knowledge during the process of ontology development from the database schema.

The tasks are straightforward upon the structure of the source code. Firstly, we need to extract the Data Labels and their associated Data Manipulation Code including *direct links*, *indirect links* and *validation code*; secondly we need to map the Data Labels onto the relational schema of the backend database; finally we need to refine the database schema with the extracted knowledge and to integrate the extracted knowledge into the ontology developed from the refined database schema. The entire process including the knowledge extraction from application source code and the integration with the relational schema for ontology development is illustrated in *Figure 5*.

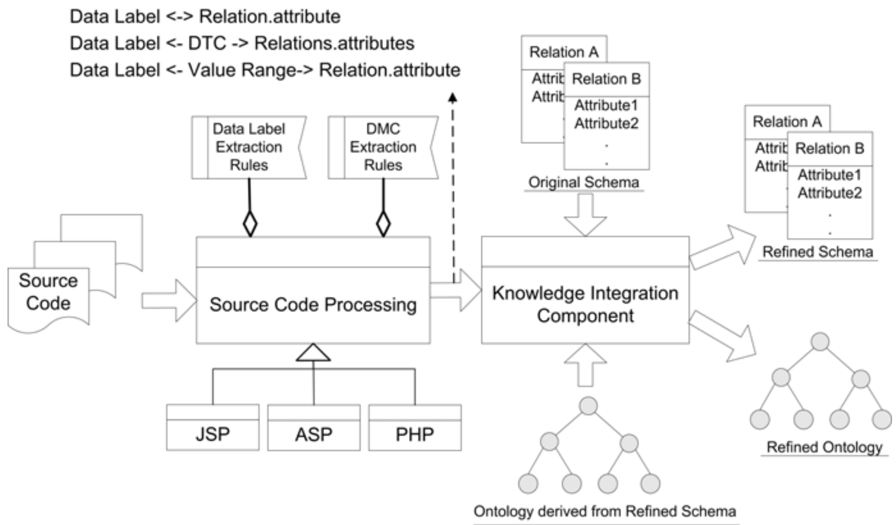

Data Label <-> Relation.attribute
Data Label <- DTC -> Relations.attributes
Data Label <- Value Range-> Relation.attribute
Data Label Extraction Rules
DMC Extraction Rules
Source Code
Source Code Processing
JSP
ASP
PHP
Relation A
Relation B
Attribute1
Attribute2
Original Schema
Knowledge Integration Component
Refined Schema
Refined Ontology
Ontology derived from Refined Schema

Figure 5. The model of using application source code to supplement ontology development from relational database

This model contains two major components: the *Source Code Processing Component* and the *Knowledge Integration Component* which are in charge of domain knowledge extraction from application source code and in charge of domain knowledge integration with the relational schema respectively.

4.1. The source code processing component

The Source Code Processing Component takes input as the source code files of a web-based application and performs code analysis including SQL analysis upon them. The source code includes all related code in the web application which could generally contain the web page source file, .Net or java Classes, Java Beans, .NET web services and business rules, JavaScript etc. The source code that is implemented in different languages will be processed by one of the corresponding language processing units shown as JSP, ASP and PHP in the model. A new unit for processing other implementation languages can be plugged into the model when needed.

The code processing is carried out based on the web-based application structure definition described in the previous section. Two libraries, i.e. Data Label Extraction Rules and DMC Extraction Rules are used. Data Label Extraction Rules is used for extracting the Data Labels and Data Carrier pairs; DMC Extraction Rules is used for extracting the direct and indirect links between Data Labels and the database schema and validation knowledge. The extraction processes are described as the following:

1. All the pairs of Data Label and Data Carrier are extracted from each of the source code files based on the Data Label Extraction Rules.

2. For each Data Carrier in the extracted pairs, the process tracks its associated Data Manipulation Code based on DMC Extraction Rules in order to map it onto the relational schema of the backend database. This code tracking procedure will lead to three scenarios which generate three types of output:

Scenario A: In this scenario, the Data Carrier is linked directly to a SQL query. Therefore, we can map one or more attributes of relations in the relational schema specified in the SQL query to the Data Label in the pair. The process is then completed for this pair of Data Label and Data Carrier. The output in this scenario is one or one set of a new entry of mapping between the Data Label and an attribute of relation in the database schema as in the form: *Data Label ←→ Attribute of Relation*

Scenario B: In the second scenario, the Data Carrier is linked to the SQL query indirectly via Data Transformation Code (DTC) in DMC such as calculation, combination. Operands in the DTC either come from a SQL query which indicate an attribute of a relation in the backend database or from hard coded knowledge. The output in this scenario is a new entry of indirect mapping in the form of: *Data Label→ DTC Code ← Attributes of Relations*. Note that the attributes may come from more than one relation.

Scenario C: In the third scenario, the Data Carrier is validated against some hard coded knowledge (note that we only need to consider the data value range validation as mentioned in 3.2.3). Then the valid Data Carrier is updated to the backend database via a SQL statement. The output form in this scenario is: *Data Label ←→ Data Range ←→ Attribute of Relation*.

There are many possible ways in each of the implementation languages that the Data Labels can be associated with Data Carriers other than a variable, i.e. Data Carrier. For example, in ASP, a *"datagrid*" can be used to display a set data value associated with some attributes of relations from a SQL query result; in JSP, a set of *"Getter"* and *"Setter"* associated with java classes or java beans may be used to fetch data from the backend database. In these cases, the code analysis process unit will define a set of forms of the Data Carriers for each of the implementation languages.

An important sub-area of this work is the creation and maintenance of Data Label Extraction Rules and DMC Extraction rules. It is, however, a challenge that will need to be addressed in our future work in depth.

4.2. The knowledge integration component

The Knowledge Integration Component synthesizes the three types of outputs, which are generated from the Source Code Processing Component, into the relational schema and into the ontology derived from the relational schema after a refinement process.

4.2.1. Integration of the Direct Link with Database Schema. The direct links are the outputs generated from the *Scenario A* in the processing component. This integration process replaces the attributes' names of relations in the relational schema with the corresponding Data Labels specified in links. One may argue that it takes too much effort for the simple term replacement. However, the real world database schemas have serious poor naming problem, which consequently makes poor sense of the ontologies created from database schemas. The original database schema is refined after this process.

4.2.2. Integration of the Indirect Link with the Derived Ontology. The indirect links are the output generated from the *Scenario B* and are the Data Transformation Code which consists of multiple operands and operators. For those Data Labels that are linked to multiple attributes of a single relation, they will be added as an attribute of that relation if they are not defined in the database schema; For the Data Labels that are linked to multiple attributes of more than one relation, they will be added as attributes of the relations with mandatory cardinality in the relationship among the involved relations. The relationship and its cardinality can be derived from the database schema.

The Data Transformation Code in the links will be added to the resulting ontology, which is developed from database schemas, as axioms. The axioms specify the calculation formula which involves multiple properties of one or more Classes in OWL DL[15]. However, the current constructs defined in OWL DL specification cannot model this type of knowledge directly. We have discussed this issue and proposed a solution based on conceptualization in OWL in a separate paper [16].

4.2.3. Integration of the Validation Knowledge. The Validation knowledge is generated from the *Scenario C*. It specifies the value range of some attributes of relations in the database schema. This will be added as a property restriction to the resulting ontology.

5. Conclusion and future work

In this paper, we have motivated the need for using database related application source code to assist and supplement the ontology development from relational databases. User friendly terms displayed at user interfaces of the application are the intended meanings of the data held in backend databases. While the names of the attributes coded in the backend databases are named arbitrarily, they can show little semantics of the data. In addition, the application embeds business rules and logic which are usually not defined in the database schema but are also important to be captured in the resulting ontologies. Therefore, we have firstly examined the structure of web-based application source code and defined Data Labels and Data Carriers which can be mapped to the database schema of a backend database. Then we proposed the process model for extracting and integrating the Data Labels, and the knowledge embedded in Data Manipulation Code to refine the backend database schema and the ontology derived from the relational schema. Future work in this research includes: defining the Data Label extraction rules for accurate Data Label extraction among the source code and the implementation of a prototype of the entire approach to evaluate this approach and the ontology developed.

6. References

[1] A. Gómez-Pérez, D. Manzano-Macho, E. Alfonseca, R. Núñez, I. Blacoe, S. Staab, O. Corcho, Y. Ding, J. Paralic, and R. Troncy, "A survey of ontology learning methods and techniques," OntoWeb Consortium 2003.

[2] V. Kashyap, "Design and creation of ontologies for environmental information retrieval," in *the 12th Workshop on Knowledge Acquisition, Modeling and Management*, Alberta, Canada, 1999.

[3] L. Stojanovic, N. Stojanovic, and R. Volz, "Migrating data-intensive Web Sites into the Semantic Web," in *the 17th ACM symposium on applied computing (SAC),*, SAC, 2002, pp. 1100-1107.

[4] I. Astrova, "Reverse engineering of relational database to ontologies," in *First european Semantic Web symposium, ESWS*, Heraklion, Crete, Greece, 2004, pp. 327-341.

[5] M. Jarrar and R. Meersman, "Formal Ontology Engineering in the DOGMA Approach," in *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE; Confederated International Conferences CoopIS, DOA, and ODBASE 2002*, 2002, pp. 1238 -1254.

[6] C. Silva and J. A. Cullell, *Knowledge Coordination*, 2003.

[7] R. H. L. Chiang, T. M. Barron, and V. C. Storey, "Reverse Engineering of Relational Databases: Extraction of an EER Model from a Relational Database," *Data & Knowledge Engineering,* vol. 12, pp. 107-142, 1994.

[8] M. Kifer, G. Lausen, and J. Wu, "Logical Foundations of Object-Oriented and Frame-Based Languages," *Journal of the Association for Computing Machinery,* pp. 741-843, 1995.

[9] W3C-OWL, "Ontology Web Language Overview." vol. 2006, M. K. Smith, C. Welty, and D. L. McGuinness, Eds.: WC3, 2004.

[10] S. Zhao and E. Chang, "From Database to Semantic Web Ontology: An Overview," in *Third International IFIP Workshop on Semantic Web & Web Semantics (IFIP SWWS 2007)*, Albufeira, Portugal, 2007.

[11] I. Astrova and B. Stantic, "An HTML Forms driven Approach to Reverse Engineering of Relational Databases to Ontologies," in *the 23rd IASTED International Conference on Databases and Applications (DBA)*, Innsbruck, Austria, 2005, pp. 246-251.

[12] W3C-RDF, "Resource Description Framework." vol. 2006, 2004.

[13] J. Barrasa, Ó. Corcho, and A. Gómez-Pérez, "R_2O, an Extensible and Semantically Based Database-to-ontology Mapping Language," in *Semantic Web and Databases, Second International Workshop, SWDB 2004*, Toronto, Canada, 2004.

[14] C. Bizer, "D2R MAP – A Database to RDF Mapping Language," in *the 12 thInternational World Wide Web*, Budapest, Hungary, 2003.

[15] W3C, "Ontology Web Language," WC3, 2006.

[16] S. Zhao, P. Wongthongtham, T. Dillon, and E. Chang, "Mapping Relational Data Model to OWL Ontology: Knowledge Conceptualization in OWL," in *2nd IEEE International Conference on Digital Ecosystems and Technologies*, Phitsanulok Thailand, 2008.