2008 Second IEEE International Conference on Digital Ecosystems and Technologies (IEEE DEST 2008)

Mapping Relational Data Model to OWL Ontology: Knowledge Conceptualization in OWL

Shuxin Zhao, Pornpit Wongthongtham, Tharam Dillon, Elizabeth Chang
Digital Ecosystems & Business Intelligence Institute, Curtin University of Technology, Australia
{s.zhao, p.wongthongtham, e.chang, tharam.dillon@curtin.edu.au}

Abstract

In this paper, we introduce the issues and solutions of using OWL ontology to model extra restriction on 'Properties' of 'Classes' that are not provided by OWL specifications and to represent associations amongst 'Properties' other than 'Classes'. Two specific types of knowledge that cannot be modeled directly using OWL DL elements are identified and presented. Firstly the data value range constraint for a "DatatypeProperty"; secondly the calculation knowledge representation. Our approach to such issues is to conceptualize the knowledge in OWL and map the conceptualization in an implementation. Examples for each type of the knowledge and their OWL code are provided in detail to demonstrate our approach.

1. Introduction

Driven by the growing trend of collaborations and business' needs for publishing product information, the demand for sharing the data held in traditional databases across enterprise and application boundaries and in an open environment such as the Semantic Web [1] has been constantly increasing. Ontology-based technologies that promote knowledge sharing and reuse with explicit semantics and machine understandable representation have been studied towards this issue [2]. An ontology is defined as "*a formal, explicit specification of shared conceptualization*" [3-5]. Ontology allows specially designed software agents to automatically integrate information from heterogeneous sources at the conceptual level. Many approaches have been proposed to transform the knowledge embedded in databases, particularly in relational databases, into ontologies [6-10]. The transformation process involves database reverse engineering to acquire the implicit knowledge from databases and involves mapping the acquired knowledge onto an ontology language. Ontology Web Language (OWL) [11], as the WWW consortium recommendation for the Semantic Web, has gained the popularity as the target ontology language.

Although there are many similarities between the conceptual data model of a database (e.g. UML or EER) and an ontology, it has many practical issues when mapping relational data models onto OWL ontologies. While generalization/specialization relationships from a relational data model can be easily mapped to the hierarchical relationships by using "`Class`" and "`Subclass`" in OWL, other types of relationships and knowledge cannot be modeled directly using OWL elements such as the aggregation/composition relationships, the data value range constraint and mathematic calculation knowledge. For example, there is no any elements specified in OWL can be used for representing the constraint on the `age` property of the `Employee` class that ranges from *18* to *65*. The same situation applies to a mathematic calculation formula which calculates the total amount of a purchase order based on the quantity and price of the item purchased and based on the tax surcharge. The total amount calculation can be expressed in the following formula:

TotalAmount = (itemQuantity singleUnitPrice)*(1+taxRate)*

There are many possible ways to represent this type of knowledge in OWL. In this paper, we present one approach for representing the calculation knowledge by focusing on the knowledge conceptualization in OWL. Thus OWL ontology examples of the conceptualization of mathematic calculation knowledge are presented. The rest of this paper is arranged as follows: Section 2 reviews related work on this issue; and Section 3 summarizes OWL features for knowledge representation; this is then followed by Section 4 which demonstrates the conceptualization of calculation knowledge in OWL with sample code; and last Section 5 we conclude the paper and indicate the future work.

2008 Second IEEE International Conference on Digital Ecosystems and Technologies (IEEE DEST 2008)

2. Related Work

There is not much work that has been reported on addressing the issues of the knowledge representation with OWL. Stojanovic et al. [8] mentioned that the basic data type system in a database cannot be preserved in F-logic [12] or RDF [13]. Introducing a new class in RDF for each of the types still cannot retain the operators on the basic data types. Furthermore, some database related dynamic knowledge embedded in SQL stored procedures, triggers and built-in functions cannot be mapped to RDF.

Other research considered relevant to mapping databases to ontologies is those that introduce mapping languages such as R_2O [14] and D2R MAP [15]. R_2O specifies how to populate ontology instances of an existing ontology automatically from the data stored in a relational database. One assumption, on which the proposed approach is based, is that the mapping between an ontology's elements and their correspondent database elements is somehow known already. Under this assumption, R_2O intends to be expressive and fully declarative to specify how the ontology instances of the existing ontology can be created from its correspondent database elements such as columns of a table. It, however, does not specify how the data model of a database can be represented by an ontology in RDF or OWL. The other mapping language D2R MAP [16] specifies how to transform the data stored in a relational database into RDF syntax. It requires domain experts and database experts to identify relationships amongst tables via SQL queries in "*D2R sql*" element.

Both of the approaches do not intend to analysis the semantic mappings between a relational data model and the targeting ontology, nor to identify implicit knowledge from databases. Rather, they aim to provide an agile means of wrapping the data held in existing relational databases using RDF or OWL ontology language. Note that data differ from knowledge in that they are only raw facts.

3. OWL features for knowledge modeling

OWL[11] is the WWW consortium recommendation for the Semantic Web language. It is designed based on the formal foundation of Description Logics [16]. OWL not only allows formally describing of the meaning of terminology used in web documents but also permits machine inference and reasoning upon literally presented facts. OWL is designed based on RDF [13] and extends RDF. In order to pursue the trade-off between expressiveness and efficient reasoning, OWL has three increasingly expressive sublanguages designed to serve specific levels of implementation and users' needs. They are, namely, *OWL Lite, OWL DL, OWL Full*. Each of the sublanguages is an extension of its simpler predecessor as stated in OWL specifications. *OWL Lite* provides constructs only for specifying primary needs including classification hierarchy and simple constraints. *OWL DL* supports maximum expressiveness while retaining computational completeness and decidability which means all conclusions are guaranteed to be computable and all computations can be finished in finite time. *OWL Full* supports maximum expressiveness but not computational guarantees. In this paper, we refer the knowledge representation issues with OWL to *OWL DL* as it is the more practical one to be used in Semantic Web applications. The term "construct" and "element" of OWL DL are used interchangeably to describe the building blocks specified in OWL specifications.

The basic building blocks of *OWL DL* consist of Class, Properties of Class and Individuals of Classes which can be corresponding to Entity type, attributes of Entity type and Entity occurrences of an EER model respectively. A classification or taxonomic hierarchy of classes is realized through the element "subClassOf", which corresponds to the generalization/specialization ("is-a") relationship type between two concepts. For instance the concept "Manager" is a subtype of the concept "Staff". Two types of Property can be defined in *OWL DL*: "`DatatypeProperty`" and "`ObjectProperty`". A *DatatypeProperty* relates a property to the datatypes defined by RDF literals [13] or XML Schema Datatypes [17]. An *ObjectProperty* relates a property to an individual of a Class that actually implies an association between two concepts. For example, the Class "`Order`" has an ObjectProperty called "`customer`" whose range is of the Class "`Customer`".

OWL provides powerful mechanisms to enhance reasoning about the classes defined in an ontology by specifying property characteristics such as transitive, symmetric and functional and through property restrictions such as `allValueFrom`, `someValueFrom` etc. Some simple set operations such as `unionOf`, `intersectionOf`, and `complementOf` are also supported in OWL. There restrictions are also the means for defining axioms in OWL. However, this powerful mechanism is more designed on *ObjectProperty* for reasoning and inferring relationships among Classes while constructs for representing relationships among properties are much less provided. Only the built-in datatypes from

XML Schema are supported in OWL. Restricting *datatypeProperty* and specifying relationships amongst properties cannot be directly specified using these provided constructs without supply of further information.

4. Knowledge conceptualization in OWL

As discussed in section 3, *OWL DL* provides little mechanism for represent relationships amongst properties but many powerful means for modeling relationships amongst classes. Therefore, one approach to model associations between properties and more restrictions on properties is to conceptualize them as new classes. Then the specified elements in OWL for modeling relationships among classes can be used to model the transformed properties (i.e. classes). In addition, property restrictions and associations between properties actually represent concepts that need to be defined explicitly. For example, the age limit, for recruiting new employees ranging from 18 to 65, represents the concepts: minimum age and maximum age, which are different from the concept age of a natural human being. We can define it in OWL as `EmployeeAge` class which has two properties `minAge` and `maxAge` as shown in figure 1 below.

```
<owl:Class rdf:ID="EmploymentAge" />

<owl:DatatypeProperty rdf:ID="minAge">
  <rdfs:domain rdf:resource="#EmploymentAge"/>
  <rdfs:range rdf:resource="&xsd;#integer" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="maxAge">
  <rdfs:domain rdf:resource="#EmploymentAge"/>
  <rdfs:range rdf:resource="&xsd;integer" />
</owl:DatatypeProperty>

<EmploymentAge rdf:ID="ABCEmploymentAge">
  <minAge rdf:datatype="&xsd;#integer">18</minAge>
  <maxAge rdf:datatype="&xsd;#integer">65</maxAge>
</EmploymentAge>

<owl:Class rdf:ID="Employee" />

<owl:ObjectProperty rdf:ID="age">
  <rdfs:domain rdf:resource="#Employee" />
  <rdfs:range rdf:resource="#EmploymentAge" />
</owl:ObjectProperty>
```

Figure 1 Employee age limit conceptualization in OWL

To demonstrate further the approach of conceptualization in OWL, we use the calculation knowledge as an example and with detailed OWL code.

4.1. The calculation of total amount of a purchase order

We use the calculation formula introduced in the Introduction section here as the example and we assume it is for the purchase order processing in a company randomly named as *ABC*.

*TotalAmount = (itemQuantity*singleUnitPrice)* (1+taxRate)*

This calculation states that the "`total amount`" for a purchase order is calculated based on the "`quantity`" and "`single unit price`" of the item purchased, and the surcharge of "`GST tax`" for the purchase. Therefore, the "`total amount`" calculation actually involves three sub-calculations listed as the following:

*SubTotal = itemQuantity * singleUnitPrice*
*Tax = SubTotal * GSTRate*
TotalAmount = SubTotal + Tax

Analyzing the formula we can derive all the concepts and their properties relating to this calculation knowledge which include:

- *Purchase Order: totalAmount*
- *Item: quantity, price*
- *Tax: taxRate*

In addition, we also need to define the general concept about mathematical calculation:

- *Calculation: operand, operator*

Once we have defined these general *Classes*, we can create the individual purchase order for ABC Company that contains the total amount calculation.

4.2. Conceptualize calculation knowledge in OWL

A mathematic calculation consists of operands, operators and the order of operands and operators in general. In order to simplify the calculation definition, we can divide any calculation into the basic calculation unit which contains only two operands and one operator to avoid defining the calculation order. The operand of a calculation can be derived from any property of any class defined in the same ontology and can also come from another existing calculation individual. The operator has the standard arithmetic operators which include *addition*, *subtraction*, *multiplication* and *division*. The definition of *Operand*, *Operator* and *Calculation* classes in OWL is shown in Figure 2, Figure 3 and Figure 4 as the following:

```
<owl:Class rdf:ID="Operand"/>

<owl:FunctionalProperty rdf:ID="valueFrom_property">
    <rdf:type rdf:resource="&owl;DatatypeProperty"/>
    <rdfs:domain rdf:resource="#Operand"/>
    <rdfs:range rdf:resource="="&xsd;tring"/>
    <rdfs:comment rdf:datatype="&xsd;tring">
      the name of the property in the valueFrom_class,
      from which the value of the operand is obtained.
    </rdfs:comment>
</owl:FunctionalProperty>

<owl:FunctionalProperty rdf:ID="valueFrom_class">
    <rdf:type rdf:resource="&owl;ObjectProperty"/>
    <rdfs:domain rdf:resource="#Operand"/>
    <rdfs:range>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Tax"/>
          <owl:Class rdf:about="#OrderItem"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:range>
    <rdfs:comment rdf:datatype="&xsd;string" >
     the Class that the operand property belongs.
    </rdfs:comment>
  </owl:FunctionalProperty>
```

Figure 2 Operand Class definition in OWL

```
<owl:Class rdf:ID="Operator"/>
<owl:Class rdf:ID="Arithmetic_Operator">
    <rdfs:subClassOf rdf:resource="#Operator"/>
</owl:Class>

<owl:FunctionalProperty rdf:ID="operator_name">
    <rdf:type rdf:resource="&owl;DatatypeProperty"/>
    <rdfs:domain rdf:resource="#Operator"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:FunctionalProperty>

 <owl:FunctionalProperty rdf:ID="symbol">
    <rdf:type rdf:resource="&owl;DatatypeProperty"/>
    <rdfs:domain rdf:resource="#Operator"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:FunctionalProperty>

<owl:AllDifferent>
  <owl:distinctMembers rdf:parseType="Collection">
    <Arithmetic_Operator rdf:ID="Addition">
      <symbol rdf:datatype="&xsd;string">+</symbol>
      <operator_name rdf:datatype="&xsd;string">
          addtion
     </operator_name>
    </Arithmetic_Operator>
    <Arithmetic_Operator rdf:ID="Subtraction">
      <symbol rdf:datatype="&xsd;string">-</symbol>
      <operator_name rdf:datatype="&xsd;string">
          subtraction
      </operator_name>
    </Arithmetic_Operator>
    <Arithmetic_Operator rdf:ID="Multiplication">
      <symbol rdf:datatype="&xsd;string">*</symbol>
      <operator_name rdf:datatype">
          multiplication
      </operator_name>
    </Arithmetic_Operator>
    <Arithmetic_Operator rdf:ID="Division">
      <symbol rdf:datatype="&xsd;string">/</symbol>
      <operator_name rdf:datatype="="&xsd;string">
         division
      </operator_name>
    </Arithmetic_Operator>
  </owl:distinctMembers>
</owl:AllDifferent>
```

Figure 3 Operator Class definition in OWL

```
<owl:Class rdf:ID="Calculation">
  <owl:Restriction>
    <owl:onProperty>
      <owl:ObjectProperty rdf:ID="hasOperand"/>
    </owl:onProperty>
    <owl:cardinality rdf:datatype="&xsd;int">
       2
    </owl:cardinality>
  </owl:Restriction>
 </owl:Class>

 <owl:ObjectProperty rdf:about="#hasOperand">
    <rdfs:domain rdf:resource="#Calculation"/>
    <rdfs:range>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Operand"/>
          <owl:Class rdf:about="#Calculation"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:range>
  </owl:ObjectProperty>

<owl:FunctionalProperty rdf:ID="hasOperator">
    <rdfs:range rdf:resource="#Operator"/>
    <rdfs:domain rdf:resource="#Calculation"/>
    <rdf:type rdf:resource="&owl;ObjectProperty"/>
</owl:FunctionalProperty>
```

Figure 4 Calculation Class definition in OWL

```
<Calculation rdf:ID="GSTTax">
  <hasOperand>
    <Operand rdf:ID="taxRate">
      <valueFrom_class rdf:resource="#GST"/>
      <valueFrom_property rdf:datatype="&xsd;string">
         taxRate
      </valueFrom_property>
    </Operand>
  </hasOperand>
  <hasOperand>
    <Calculation rdf:ID="orderSubTotal">
      <hasOperand rdf:resource="#itemQuantity"/>
        <valueFrom_class rdf:resource="#OrderItem_1"/>
          <valueFrom_property rdf:datatype="&xsd;string">
             quantity
          </valueFrom_property>
      <hasOperand>
        <Operand rdf:ID="itemPrice">
          <valueFrom_class rdf:resource="#OrderItem_1"/>
          <valueFrom_property rdf:datatype="&xsd;string">
             price
          </valueFrom_property>
        </Operand>
      </hasOperand>
      <hasOperator rdf:resource="#Multiplication"/>
    </Calculation>
  </hasOperand>
  <hasOperator rdf:resource="#Multiplication"/>
</Calculation>

<Calculation rdf:ID="oderTotalAmount">
  <hasOperator rdf:resource="#Addition"/>
  <hasOperand rdf:resource="#GSTTax"/>
  <hasOperand rdf:resource="#orderSubTotal"/>
</Calculation>
```

Figure 5 Individuals definition for total Amount calculation

According to the class definitions about calculation knowledge, now we create the individual of the calculation for the total amount for ABC Company purchase order. Three individuals of the *Calculation* class are created, namely: `orderSubtotal`, `GSTTax` and `orderTotalAmount` (Shown as in

Figure 5). Then the ABC Company purchase order individual can be defined as in Figure 6 below.

```
<Order rdf:ID="ABCPurchaseOrder">
  <orderItem rdf:resource="#OrderItem_1"/>
  <totalAmount>
    <Calculation rdf:ID="oderTotalAmount">
      <hasOperator>
        <Arithmetic_Operator rdf:ID="Addition">
          <symbol rdf:datatype="&xsd;string">+</symbol>
          <operator_name rdf:datatype="&xsd;string">
              addtion
          </operator_name>
        </Arithmetic_Operator>
      </hasOperator>
      <hasOperand rdf:resource="#orderSubTotal"/>
      <hasOperand>
        <Calculation rdf:ID="GSTTax">
          <hasOperator rdf:resource="#Multiplication"/>
          <hasOperand rdf:resource="#orderSubTotal"/>
          <hasOperand>
            <Operand rdf:ID="taxRate">
              <valueFrom_class rdf:resource="#GST"/>
              <valueFrom_property rdf:datatype="&xsd;string">
                 taxRate
              </valueFrom_property>
            </Operand>
          </hasOperand>
        </Calculation>
      </hasOperand>
    </Calculation>
  </totalAmount>
</Order>
```

Figure 6 The individual definition of ABC Company purchase order

5. Conclusion

In this paper we discussed the problems associated with knowledge representation in OWL, specifically the representation of associations between properties. OWL provides powerful mechanism for defining relationships among classes but not for properties. Our solution to this issue is to conceptualize the property restrictions and the associations between properties as OWL Classes. Then we can utilize the elements specified in OWL for classes' axioms to represent these types of knowledge. Detailed OWL code of the calculation knowledge of a purchase order is given to demonstrate our approach. The further work needs to be done towards this approach is to map the OWL definitions into an implementation, which will test the performance and efficiency of the approach.

6. References

[1] W3C, "Semantic Web," vol. 2006, 2006.

[2] 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.

[3] T. R. Gruber, "A Translation Approach to Portable Ontology Specifications," *Knowledge Acquisition*, vol. 5, pp. 199-220, 1993.

[4] W. Borst, "Construction of Engineering Ontologies," University of Twente, Enschede, 1997.

[5] R. Studer, V. R. Benjamins, and D. Fensel, "Knowledge engineering: Principles and methods," *Data & Knowledge Engineering*, vol. 25, pp. 161-197, 1998.

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

[7] R. Meersman, "Ontologies and Databases: More than a Fleeting Resemblance," presented at OES/SEO Workshop Rome, Rome, 2001.

[8] L. Stojanovic, N. Stojanovic, and R. Volz, "Migrating data-intensive Web Sites into the Semantic Web," presented at the 17th ACM symposium on applied computing (SAC),, SAC, 2002.

[9] I. Astrova, "Reverse engineering of relational database to ontologies," presented at First european Semantic Web symposium, ESWS, Heraklion, Crete, Greece, 2004.

[10] M. LI, X.-Y. DU, and S. WANG, "Learning ontology from relational database," presented at the Fourth International Conference on Machine Learning and Cybernetics, Guangzhou, China, 2005.

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

[12] wikipedia, "F-logic," vol. 2006, 2006.

[13] W3C, "RDF," vol. 2006, 2006.

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

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

[16] F. Baader, "The Description Logic Handbook: Theory, Implementation and Application," F. Baader, Ed.: Cambrigde University Press, 2002.

[17] W3C, "XML Schema Datatypes," vol. 2006, 2004.