Knowledge Conceptualization and Software Agent based Approach for OWL Modeling Issues

Shuxin Zhao, Pornpit Wongthongtham, Elizabeth Chang, Tharam Dillon

Digital Ecosystems & Business Intelligence Institute, Curtin University of Technology
GPO Box U1987
Perth WA 6845, Australia
{s.zhao, p.wongthongtham, e.chang, tharam.dillon@curtin.edu.au}

Abstract. In this paper, we present the issues and solutions of using OWL ontology to model the knowledge captured in relational databases. Two specific types of knowledge, which are common to various domains, are identified that cannot be represented directly using constructs specified in OWL DL. Firstly the data value range constraint and secondly the calculation knowledge representation. The solution to the first problem is to conceptualize the data range as a new class and the solution to the second problem is proposed, based on utilizing software agent technology. Examples with OWL code and implementation code are given to demonstrate the problems and solutions.

Keywords: OWL modeling issue, Knowledge conceptualization, Software-agent, Ontology

1 Introduction

Since its advent, database technologies have been broadly applied to the development of information systems where persistent data repository and efficient data retrieval are required. This has consequently produced a vast number of databases. These databases embed extensive domain knowledge and up-to-date business information that are crucial to various domains and business processes. With the increasing trend of collaborations amongst organizations and business needs for sharing and publishing their products information, these databases are demanded to be shared and integrated without organizational and application boundaries. However, databases are enterprise and application dependant in that their design and development are subjected to a particular business problem domain of an organization. This has prevented the databases from being shared and integrated in an open environment.

Ontology-based technologies provide a feasible approach to this problem. Ontology-based technologies promote knowledge sharing and integration by formally and explicitly defining the meanings and associations of information and data. Knowledge represented by ontologies will enable machines or software agents to combine and share them from heterogeneous sources meaningfully. An ontology is defined as "a formal, explicit specification of shared conceptualization" [2-4]. Ontologies allow specially designed software agents to automatically process and

integrate information from distributed sources. Many approaches have been proposed to transform the knowledge embedded in databases, particularly in relational databases, into ontologies [5-10]. 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.

However, there is a critical issue of using OWL to fully and accurately represent the knowledge captured in relational databases. Although there are many similarities between a conceptual data model of a database, such as UML or EER model, and an ontology (some researchers classify UML as lightweight ontologies [12]), there are many practical issues when mapping the knowledge captured in a conceptual model onto an OWL ontology. For example, there are three common types of relationships between concepts we model in an UML model, namely, generalization/specialization, aggregation and composition and association. While generalization/specialization can be modeled straightforward using OWL hierarchical mechanism i.e. *Class* and *Subclass, Property* and *Subproperty,* the *aggregation/composition* relationship cannot be represented directly using OWL elements. There are also other types of knowledge captured by a relational database that we found hard to model in OWL such as the value range restrictions on an attribute, and the functional dependency among several attributes of one or more tables which capture some sort of relationships between attributes rather than concepts.

In this paper, we present two alternative solutions to tackle this OWL modeling issue, namely, conceptualization approach and software agent based approach. Two specific examples are used to demonstrate each of the approaches respectively: firstly the problems of modeling the data value range constraint; secondly, the problem of modeling mathematic calculation knowledge, whose operands are derived from attributes of one or more concepts, which represents relationships between these attributes. Our motivation is to reveal some ideas of extending the expressiveness of OWL in the mean time to retain computational completeness of the ontology model, thus to make OWL more useful and more adaptive. The rest of this paper is organized as follows: Section 2 reviews related work on these issues; Section 3 introduces some preliminary concepts necessary for understanding the foundations of the proposed solutions; then Section 4 describes the problems in details with examples; followed by Section 5 demonstrating the solutions to the problems with code example; last in Section 6, we conclude the paper and indicate future work.

2 Related Work

W3c rules is an addition for modeling knowledge in addition to OWL. Expressiveness vs. decidability. HP report OWL weakness.

There is not much work that has been reported on addressing the issues of the knowledge representation with OWL. Stojanovic et al. [6] mentioned that the basic data type system in a database cannot be preserved in F-logic [13] or RDF [14]. 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 R2O [15] and D2R MAP [16]. R2O 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, R2O aims 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.

3 Preliminary Concept

3.1 OWL Specification (reduce to an simple description)

OWL[17] is the WWW consortium recommendation for the Semantic Web language. It is designed based on the formal foundation of Description Logics [18]. 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 [14] 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 [17]. *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* [17] 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"*. *DatatypeProperty* relates a property to the data types defined by RDF literals [14] or XML Schema Datatypes [19]. *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. These 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 associations amongst properties are much less provided. Only the built-in data types from XML Schema are supported in OWL. Restricting *DatatypeProperty* and specifying relationships amongst properties cannot be directly modeled using these provided constructs without the supply of further information.

3.2 Software Agent

Software agent technology has been in extensive discussion for many years but it is perhaps recently that it has been attracting much attention of exploitation in the emergence of the Semantic Web. Basically software agents are components in an application that are characterized by among other things autonomy, pro-activity and an ability to communicate [20]. Autonomy means that agents can independently carry out complex and long term tasks. Pro-activity means that agents can take initiative to perform a given task without human intervention. Ability to communicate means agents can interact with other agents or other components to assist to achieve their goals.

In this paper we implement software agents using JADE (Java Agent Development framework), an agent-oriented middleware [21, 22]. The reason we use JADE is simply because it facilitates development of complete agent-based applications and it is written in well known object-oriented language, Java. More details of JADE can be found on its website (http://jade.tilab.com).

Basically in this paper, we utilize JADE agent technology to help define the knowledge of calculation. A JADE agent is identified under FIPA specifications [23] by an agent identifier. A task can be defined for an agent to carry out. Agent action defines the operations to be performed. Agent communication according to FIPA specifications [23] is the most fundamental feature of software agents. Format of messages is compliant with that defined by FIPA-ACL message structure.

4 Problem Description with Examples

In this section, we describe the two specific types of knowledge that cannot be modeled directly in OWL in detail with examples. Solutions to the problems are given in the following section.

4.1 Data Value Range Modeling Problem in OWL

The first type of knowledge that we mentioned in the introduction section that cannot be modeled directly using constructs defined in OWL DL is the constraint on data value range. Data value range constraint is very common to various domains. For example, a company recruitment statement contains a minimum age and a maximum age requirement and a bank product requests a minimum and a maximum amount of deposit over a period such as monthly. This refers to data value range in database development. This kind of data constraint can be obtained from database schema, application source code through validation and SQL queries. It, however, cannot be directly represented using any constructs specified in OWL DL. One example of the recruitment requirement for the employee's age constraint in a company, named ABC, can be expressed as the formula:

```
ABCEmployee (18 < age < 65)
```

In OWL DL, if we define a Class namely *Employee*, with a *DatatypeProperty* namely *age* shown as in the OWL definition below:

```
<owl:Class rdf:ID="Employee"/>
<owl:DataTypeProperty rdfID="age">
      <rdfs:domain rdfresource="#Employee" />
      <rdfs:range rdfresource="&xsd;integer" />
<owl:DatatypeProperty>
```

We may further add constraints such as the cardinality on the *age* property, but no any other elements defined in OWL for property restrictions, such as *allValueFrom* and the set operator like *unionOf* and *intersectionOf*, can be used to model the simple value range constraint. We therefore need other means to represent this kind of knowledge in OWL ontologies.

4.2 Calculation Knowledge Representation Problem in OWL

The second type of knowledge cannot be modeled directly using OWL constructs is the general calculation knowledge. An arithmetic calculation consists of operands and arithmetic operators such as addition, subtraction, multiplication and division. Operands in a calculation are often derived from columns of tables in a database or from properties of Classes in an ontology. The result of a calculation, in the mean time, is assigned to a column or a property. This represents associations amongst properties rather than classes. It may also represent the dynamic knowledge which is generated at run time in a given application. This type of knowledge is usually

defined in SQL queries such as stored procedures or application source code when validating new data entry to ensure data consistency. One example of this type of knowledge is the calculation of total cost including GST tax of a purchase. The cost is calculated based on three properties: the *"quantity"* of the product in the purchase, the *"price"* of the product excluding GST tax and current *"GST tax rate"*. It can be expressed as the following formulas:

```
SubTotal = itemQuantity * singleUnitPrice
Tax = SubTotal * GSTRate
TotalCost = SubTotal + Tax
```

In OWL, there is no constructs defined for modeling this type of associations among properties from one or more Classes.

5 Approach

For the modeling problems stated in the previous section, we propose two possible ways to tackle the issues, which are demonstrated with code in this section.

5.1 Conceptualization of Data Value Range Constraint in OWL

As OWL does not provide any constructs for restricting value range on *DatatypePropertie*, we cannot represent this constraint directly in the way that we specify it in a programming language or in a database management system. However, we can model the value range constraint by conceptualizing it into a new Class. The conceptualization actually explicitly reflects the semantics of the data restriction because the general concept *Age* of human being is different from the concept *minimum age* and *maximum age* in a company recruitment requirement. We demonstrate the solution to the first problem defined in section 4.1 in List 1.

In the OWL ontology List 1, the constraint on employee's age is conceptualized as a new Class *"EmploymentAge"*. It has two *DatatypeProperties*: *"minAge"* and *"maxAge"*. There is one individual created for ABC company recruitment requirement called *"ABCEmploymentAge"* whose *"minAge"* is 18 and *"maxAge"* is 65. The property *"age"* of the Class *"Employee"* can therefore be defined as an *ObjectProperty* whose range is of the class *"EmployeeAge"*. If there are individuals of ABC company employee, their age must be between 18 and 65.

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

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

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

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

```
 </EmploymentAge>

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

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

List 1 conceptualization of data value range constraint in OWL

5.2 Software Agent-Based Knowledge Representation Approach

Other than the conceptualization, we can also incorporate other existing technologies to represent the knowledge. In this section, we demonstrate how to represent the calculation knowledge by using software agent with OWL. For the calculation knowledge described in section 4.2, we can define it in the following formula.

```
Total Cost = Price * Quantity * (1 + GST Rate / 100)
```

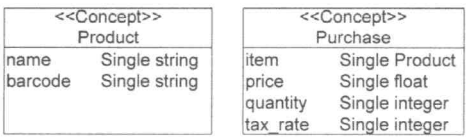
Ontologies are typically specific to a given domain. For the above formula we specify to a product trading domain which would not be the same as in a payroll system. Thus product concept could have properties of name, barcode, etc. Agents then have some shared understanding with the product concept and its properties. There may be two products named the same. In order to unequivocally identify a product, it may be necessary to specify barcode.

According to the FIPA specifications [23], when agents communicate, product information representation is embedded inside ACL messages. Because JADE agents are Java-based, the information can be represented using objects.

In order to exploit agent and ontology technology to support and allow agents to discourse and reason about facts and knowledge related to a given domain, we specify the approach into 3 steps.

- Define concepts in an ontology. In the purchase example, it includes *Product* and *Purchase* concepts.
- Develop proper Java classes for the above two concepts in the ontology.
- Define the calculation formula by hard-coding it.

In order to illustrate defined concepts of *Product* and *Purchase* in an ontology, we use Wongthongtham's notation [24] to model *Product* and *Purchase* knowledge representation. Figure 1 (A) shows *Product* concept and Figure 1 (B) shows *Purchase* concept. Ontology class *Product* has datatype properties of *name* and *barcode* both related to a string type. Ontology class *Purchase* has object properties of *item* related to the ontology class *Product*. The ontology class *Purchase* also has datatype properties of *price* related to a float type and *quantity* and *tax_rate* related to an integer type.



(A) (B)

Fig. 1 Product and Purchase concepts in Ontology Modeling

We reuse schema classes available in JADE *PredicateSchema*, *AgentActionSchema*, and *ConceptSchema* included in the *jade.content.schema* package to define the structure of each type of predicate, agent action, and concept respectively [22]. In the example, we can model the domain including one concept (Product), one predicate (Purchase – to apply to a product) and one agent action (Calculate – to calculate total cost including tax).

Since the ontology is shared among agents, *TradeOntology* class is placed in an ad-hoc package, ontology. The ontology defined in Java is shown in List 2.

```
package TradingPackage;

import jade.content.onto.*;
import jade.content.schema.*;
import jade.util.leap.HashMap;
import jade.content.lang.Codec;
import jade.core.CaseInsensitiveString;

public class TradeOntology extends jade.content.onto.Ontology  {
  //NAME
  public static final String ONTOLOGY_NAME = "Trade";
  // The singleton instance of this ontology
  private static ReflectiveIntrospector introspect = new
ReflectiveIntrospector();
  private static Ontology theInstance = new TradeOntology();
  public static Ontology getInstance() {
     return theInstance;
  }

   // VOCABULARY
    public static final String PURCHASE_ITEM="Item";
    public static final String PURCHASE_QUANTITY="Quantity";
    public static final String PURCHASE_TAX_RATE="Tax_Rate";
    public static final String PURCHASE_PRICE="Price";
    public static final String PURCHASE="Purchase";
    public static final String CALCULATOR="Calculator";
    public static final String CALCULATE="Calculate";
    public static final String PRODUCT_NAME="Name";
    public static final String PRODUCT_BARCODE="Barcode";
    public static final String PRODUCT="Product";

  /*  Constructor   */
  private TradeOntology(){
    super(ONTOLOGY_NAME, BasicOntology.getInstance());
    try {

    // adding Concept(s)
    ConceptSchema productSchema = new ConceptSchema(PRODUCT);
    add(productSchema, TradingPackage.Product.class);

    // adding AgentAction(s)
    AgentActionSchema calculateSchema = new
AgentActionSchema(CALCULATE);
    add(calculateSchema, TradingPackage.Calculate.class);

    // adding AID(s)
    ConceptSchema calculatorSchema = new ConceptSchema(CALCULATOR);
    add(calculatorSchema, TradingPackage.Calculator.class);

    // adding Predicate(s)
    PredicateSchema purchaseSchema = new PredicateSchema(PURCHASE);
    add(purchaseSchema, TradingPackage.Purchase.class);
```

```
      // adding properties
      productSchema.add(PRODUCT_BARCODE,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema. MANDATORY);
      productSchema.add(PRODUCT_NAME,
(TermSchema)getSchema(BasicOntology.STRING), ObjectSchema.OPTIONAL);
      purchaseSchema.add(PURCHASE_PRICE,
(TermSchema)getSchema(BasicOntology.FLOAT), ObjectSchema.MANDATORY);
      purchaseSchema.add(PURCHASE_TAX_RATE,
(TermSchema)getSchema(BasicOntology.INTEGER), ObjectSchema.MANDATORY);
      purchaseSchema.add(PURCHASE_QUANTITY,
(TermSchema)getSchema(BasicOntology.INTEGER), ObjectSchema.MANDATORY);
      purchaseSchema.add(PURCHASE_ITEM, productSchema, ObjectSchema.
MANDATORY);

     }catch (java.lang.Exception e) {e.printStackTrace();}
    } }
```

List 2 Trade Ontology defined in Java

```
package TradingPackage;

import jade.content.*;
import jade.util.leap.*;
import jade.core.*;

public class Product
      implements Concept {

// Barcode
 private String barcode;
 public void setBarcode(String
value){
       this.barcode=value; }
```

```
public String getBarcode() {
    return this.barcode;
}

// Name
 private String name;
 public void setName(String value) {
      this.name=value;
 }
 public String getName() {
      return this.name;
 }
}
```

List 3 Product concept defined in Java

```
package TradingPackage;

import jade.content.*;
import jade.util.leap.*;
import jade.core.*;

  public class Purchase
implements Predicate {

  // Price
   private float price;
   public void setPrice(float
value) {
      this.price=value;
   }
   public float getPrice() {
     return this.price;
   }

  // Tax_Rate
   private int tax_Rate;
   public void setTax_Rate(int
value) {
      this.tax_Rate=value;
    }
```

```
   public int getTax_Rate() {
     return this.tax_Rate;
   }

  // Quantity
   private int quantity;
   public void setQuantity(int
value) {
     this.quantity=value;
   }
   public int getQuantity() {
     return this.quantity;
   }

  // Item
   private Product item;
   public void setItem(Product
value) {
     this.item=value;
   }
   public Product getItem() {
     return this.item;
   }
}
```

List 4 Purchase concept defined in Java

The schemas for *product*, *purchase*, *calculate*, and *calculator* concepts are associated with *product.java*, *purchase.java*, *calculate.java*, and *calculator.java* classes respectively. Each property in a schema has a name and a type. For example,

in the *product* schema, *barcode* has its type as string. Every product must have barcode as declared as MANDATORY. Similarly, value for properties *item*, *price*, *quantity*, and *tax rate* cannot be null because when the purchase is made these values are mandatory. Validation is made by throwing an exception if the value of mandatory properties is null.

The *product* concept could be defined specifically to particular products e.g. books, CDs for more specific trading. Properties of the product concept i.e. *name* and *barcode* will be inherited to books and CDs. Book and CDs concepts can have their own specific properties e.g. the CDs concept might have *tracks* property and books might have *authors* property and so on.

Java classes, associated with the product concept and the purchase predicate in the example, are shown in List 3 and List 4 respectively.

Agent action associates with the agent identifier which is intended to perform action for this example to calculate total cost included tax. Calculation can be hard coded getting value from object of class purchase i.e. price, quality, and tax rate.

For example a product of $200 price, 2 quantity, and 10% tax rate would have expression as following:

```
((action (agent-identifier :name calculator) calculate
(product :name "xxx" :barcode "01211") purchase (product :name
"xxx" :barcode "01211") 360)
```

Alternatively, we can also specify in class purchase as the attribute of *TotalCost* shown as in List 5 below.

```
// Total Cost
   private float TotalCost;
   public float getTotalCost() {
     return this.price * this.quality * (1 + tax_Rate / 100);
}
```

List 5 The formula defined in Java

6. Conclusion

In this paper we addressed the problems associated with knowledge representation in OWL. OWL specifications provide many mechanisms for defining restrictions and associations among Classes but not for properties. We have presented two types of knowledge, which are common to various domains, but cannot be modeled directly using constructs specified in OWL. To tackle this knowledge presentation gap in OWL, we have proposed two alternative solutions to the problems. One is to conceptualize the knowledge such as the data value range constraints and the other is to use other existing technology such as software agents to encode and convey the knowledge. We do not intend to list all possible OWL modeling problems rather we aim to provide some hints to other likewise knowledge representation issues with OWL that have yet to be resolved.

References

1. W3C. Semantic Web. 2006 5 May 2006 [cited 2006 30 May]; Available from: http://www.w3.org/2001/sw/.
2. Studer, R., V.R. Benjamins, and D. Fensel, Knowledge engineering: Principles and methods. Data & Knowledge Engineering, 1998. 25: p. 161-197.
3. Borst, W., Construction of Engineering Ontologies. 1997, University of Twente, Enschede.
4. Gruber, T.R., A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition, 1993. 5(2): p. 199-220.
5. Kashyap, V. Design and Creation of Ontologies for Environmental Information Retrieval. in Twelfth Workshop on Knowledge Acquisition, Modelling and Management. 1999. Voyager Inn, Banff, Alberta, Canada: MCC and Telcordia Technologies, Inc.
6. Stojanovic, L., N. Stojanovic, and R. Volz. Migrating data-intensive Web Sites into the Semantic Web. in the 17th ACM symposium on applied computing (SAC),. 2002. SAC: ACM Press.
7. Meersman, R. Ontologies and Databases: More than a Fleeting Resemblance. in OES/SEO Workshop Rome. 2001. Rome: Luiss.
8. Astrova, I. Reverse engineering of relational database to ontologies. in First european Semantic Web symposium, ESWS. 2004. Heraklion, Crete, Greece: Springer.
9. LI, M., X.-Y. DU, and S. WANG. Learning ontology from relational database. in the Fourth International Conference on Machine Learning and Cybernetics. 2005. Guangzhou, China: IEEE exlorer.
10. Zhao, S. and E. Chang. Mediating Databases and the Semantic Web: A methodology for building domain ontologies from databases and existing ontologies. in SWWS'07- The 2007 International Conference on Semantic Web and Web Services. 2007. Las Vegas, Nevada, USA: CSREA Press.
11. W3C. Ontology Web Language. 2006 5 Sept 2006 [cited; Available from: http://www.w3.org/2004/OWL/.
12. Gomez-Perez, A., M. Fernandez-Lopez, and O. Corcho, Ontological engineering: with examples from the areas of knowledge management, e-Commerce and the Semantic Web. Advanced Information and Knowledge Processing, ed. X. Wu and L. Jain. 2004, London: Springer-Verlag.
13. wikipedia. F-logic. 2006 [cited 2006 21 Dec]; Available from: http://en.wikipedia.org/wiki/F-logic.
14. W3C. RDF. 2006 [cited 2006 June 2006]; Available from: http://www.w3.org/RDF/.
15. Barrasa, J., Ó. Corcho, and A. Gómez-Pérez. R2O, an Extensible and Semantically Based Database-to-ontology Mapping Language. in Semantic Web and Databases, Second International Workshop, SWDB 2004. 2004. Toronto, Canada.
16. Bizer, C. D2R MAP – A Database to RDF Mapping Language. in the 12 thInternational World Wide Web. 2003. Budapest, Hungary: ACM.
17. W3C. Ontology Web Language. 2004 5 Sept 2006 [cited 2006; Available from: http://www.w3.org/2004/OWL/.
18. The Description Logic Handbook: Theory, Implementation and Application, ed. F. Baader, et al. 2002: Cambrigde University Press.
19. W3C. XML Schema Datatypes. 2004 [cited 2006 20 Oct].
20. Wooldridge, M., Introduction to MultiAgent Systems. 1st ed. 2002: John Wiley & Sons.
21. Bellifemine, F., JADE Java Agent DEvelopment Framework. 2001, Telecom Italia Lab: Torino, Italy.
22. Bellifemine, F., G. Caire, and D. Greenwood, Developing Multi-Agent Systems with JADE. 2007: John Wiley & Sons Ltd.

23. Bellifemine, F., A. Poggi, and G. Rimassa. JADE: a FIPA2000 compliant agent development environment. in The fifth International Conference on Autonomous Agents. 2001. Montreal, Quebec, Canada: ACM Press, New York, USA.
24. Wongthongtham, P., A methodology for multi-site distributed software development, in School of Information Systems. 2006, Curtin University of Technology: Perth.