Object Oriented Model for Ontology Engineering

Afzal Ahmad, Amjad Farooq and Yasir Saleem and Asim Remat
Department of Computer Science and Engineering, U.E.T., Lahore, Pakistan

***Abstract*– Ontology engineering gets vital importance after the idea of semantic web. The huge amount of data on current web makes information retrieval most difficult and complex process. This problem brings the idea of semantic web in which data will be in a structured form so that machine can understand and integrate data from different sources to retrieve required information for the end user. Ontology makes this possible to have data in structured form, so success of semantic web heavily depends on the successful development of ontology. Ontology development from scratch is most difficult and time taking process. So there is need to use existing, mature and accepted object oriented models of software engineering for development of ontology. This will reduce effort and time because constructing complex ontology required lot of effort, time and expertise in knowledge engineering. This paper investigates the development of ontology from object oriented models.**

***Keywords*–** Semantic Web, Ontology Engineering, Object Oriented Paradigm and Class Diagram

I. INTRODUCTION

Today's web contains huge amount of data but machines cannot understand, process and integrate this huge amount of data into useful information required by humans. The reason why machines are not able to understand data is that it is not structured semantically on the web.

The idea of semantic web is that the data on the web is structured in some form so that machines are able to understand and integrate data in useful information. But the success of semantic web heavily dependent on ontologies that will make the unstructured data on current web in a structural and meaningful form. Ontologies make machines able to comprehend and integrate data on behalf of humans. So the success of semantic web is not possible without ontologies [6], [15].

Ontology engineering is a field which studies method and techniques for developing ontologies. Developing ontologies from scratch is difficult and time consuming process. The structure of ontology is very similar to the structure of class diagram in object oriented design [1], [4]. These structure similarities can be fruitful for developing ontology from class diagram with less effort and time. This paper will discuss and proposed a method to transform class diagram to ontology.

II. RELATED WORK

There are many similarities between class diagram structure and ontology structure. Both ontology and class diagram contain classes and different types of relationship between classes [2].

There are the similarities and differences between ontology development languages and object oriented languages. Classes are regarded as type of instances in object oriented design and as set of individuals in ontology development languages. Compilers are used at built time and compilers errors are regarded as problems in object oriented and reasoners are used to check consistency in ontology development languages. Classes in objected oriented design can declared their members as private but on the other everything is public in ontology development languages [9], [11], [12].

The UML class diagram is mapped to DAML+OIL ontology using the mapping [13] as listed in Table 1.

Table 1: Mapping Rules

UML	DAML+OIL
Package	Ontology
Class	Class
Attributes	DatatypeProperty
Generalization/specialization	subClassOf/subPropertyOf
Association	Object Property

The UML can be used to developed ontology because of similar structure of class diagram and ontology. The UML is open standard and widely adopted by industry and taught in universities but on the other hand ontology development requires expertise of knowledge engineering [14], [15].

III. CLASS DIAGRAM

In object oriented design, a class diagram is a hierarchy of entities of a domain of interest. Each entity in a class diagram is represented as a class. A class contains attributes and method. Attributes are the properties of an entity and method represent functions of an entity. A class in class in object oriented design can contain any number of objects and each object of class is known as single instance of that class.

The elements of class diagram are classes containing attributes and methods and relationships such as association, generalization, aggregation, composition, dependency and realization. The class diagram can be defined as [7], [10].

$D = \{C, R\}$

Here, C is the set of classes and R is the set of relationships in class diagram. The set C can be defined as

$C = \{ci\ (cn, A, M) \mid i=1, 2, 3....N\}$

Here, cn is the name of class, A is the set of attributes and M is the set of method of class ci. The set A can be defined as

A= {ai (an, b) |i=1, 2, 3….N}

Here, the an is the name of attribute and b is the data type of it.

The set R can be defined as

R={ri(ci,cj,RT,m(m1,m2),r(r1,r2)),ci<->cj }

RT= {association, aggregation, composition, generalization, dependency, realization}

Here, RT is set of relationship, m (m1, m2) is the multiplicity and r (r1, r2) is the roles of association between two ci and cj. The ci<->cj defines that relationship is unidirectional or bidirectional. Association relationship is normally two way relationship.

IV. ONTOLOGY

Ontology is not simply taxonomy, which just classify data in a domain. Ontology contains taxonomy as a component. So ontology is partially taxonomy. Taxonomy only categorizes things and nothing more but on the other hand ontology categorizes things in the form of taxonomy as well as defines richer relationships between different concepts of hierarchy [3], [5].

The elements of ontology are classes with attributes, individuals, relationships, functions and axioms. The functions and axioms are constraint on relationships. The ontology can be defined as:

O= (C, R, I)
C= {ci (cn, A) | i=1, 2, 3….N}
A= {ai (an, b) |i=1, 2, 3….N}
R={ri(ci,cj,RT,m(m1,m2),r(r1,r2),ci<->cj) }
RT= {is-a, has-a, non-taxonomic}

Here, in set C, cn is the name, A is the set of attributes and M is the set of methods of class ci. In set A, a is the name and b is the data type of attribute ai. In set R, ci and cj are two classes, RT is a set of relationships, m1 represent multiplicity of ci to cj and m2 of cj to ci, r1 represent the role on ci side and r2 on cj side and the ci<->cj represent the direction of relationship ri.

V. PROPOSED TRANSFORMATION

The class diagram can be transformed to ontology by using the simple proposed transformation. The proposed transformation is

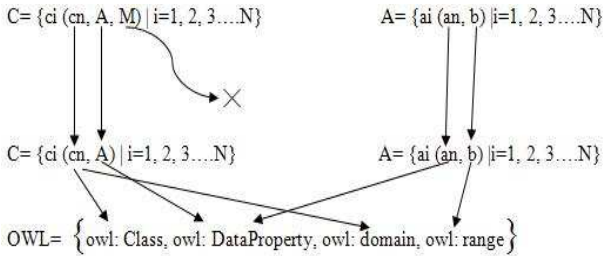

Fig. 1: Formal Representation of Rule 1

Rule 1: All classes in a class diagram will become the classes or concepts of ontology as well as attributes of classes. This rule is formally represented in Fig. 1.

Rule 2: All relationship in class diagram will transform to three types of relationships in ontology. This rule is formally represented in Fig. 2.

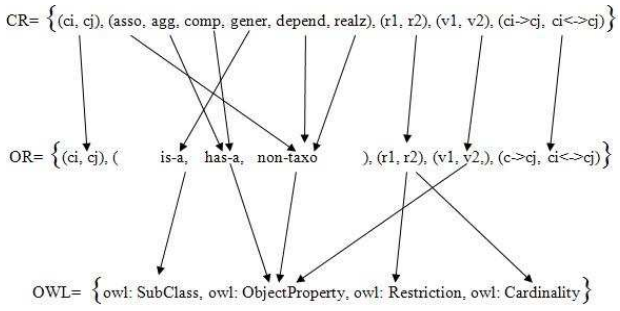

Fig. 2: Formal Representation of Rule 2

Rule 3: All instances of class will become individuals of ontology classes. By using above transformation, steps to develop ontology form class diagram are as follows [8]:

i. Define all class in class diagram using owl: class in ontology.
ii. Define attributes of each class using owl: DataProperty, owl: domain and owl: range in ontology.
iii. Define relationships between classes using owl: ObjectProperty, owl: domain, owl: range, owl: Restriction and owl: Cardinality in ontology.

Object of classes will be defines as individuals of classes in ontology.

A. Case Study: Order Processing

Order processing system class is taken as an example to apply proposed transformed. The class diagram for order processing system is given below:

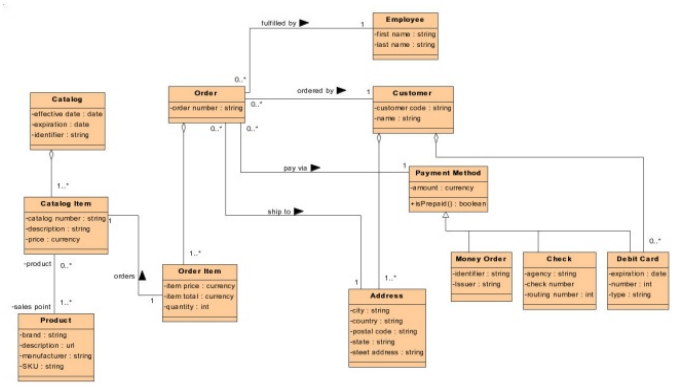

Fig. 3: Order Processing Class Diagram

The above order processing class diagram can be represented in the form of set of classes and relationship as DPurchaseOrder = (C, R) where,

C = Set of all classes in purchase order class diagram

R = Set of all relationship exist in purchase order class diagram:

C= {c1 (Order, AOrder),
c2 (OrderItem, $A_{OrderItem}$),
c3 (CatalogItem, $A_{CatalogItem}$),
c4 (Catalog, $A_{Catalog}$),
c5 (Employee, $A_{Employee}$),
c6 (Customer, $A_{Customer}$),
c7 (Address, $A_{Address}$),
c8 (PaymentMethod, $A_{PaymentMethod}$),
c9 (MoneyOrder, $A_{MoneyOrder}$),
c10 (Check, A_{Check}),
c11 (DebitCard, $A_{DebitCard}$),
c12 (Product, $A_{Product}$)}

A_{Order} = {a1 (OrderNumer, String)}
$A_{OrderItem}$ = {a1 (ItemPrice, Currency), a2 (ItemTotal, Currency), a3 (Quantity, Integer)}
$A_{CatalogItem}$ = {a1 (CatalogNumber, String), a2 (Description, String), a3 (Price, Currency)}
$A_{Catalog}$ = {a1 (EffectiveDate, Date), a2 (Expiration, Date), a3 (Identifier, String)}
$A_{Employee}$ = {a1 (FirstName, String), a2 (LastName, String)}
$A_{Customer}$ = {a1 (CustomerCode, String), a2 (Name, String)}
$A_{Address}$ = {a1 (City, String), a2 (Country, String), a3 (PostalCode, String), a4 (State, String), a5 (StateAddress, String)}
$A_{PaymentMethod}$ = {a1 (Amount, Currency), a2 (IsPaid, Boolean)}
$A_{MoneyOrder}$ = {a1 (Identifier, String), a2 (Issuer, String)}}
A_{Check} = {a1 (Agency, String), a2 (CheckNumber, String), a3 (RoutingNumber, String)}
$A_{DebitCard}$ = {a1 (Expiration, Date), a2 (Number, Integer), a3 (Type, String)}
$A_{Product}$ = {a1 (Brand, String), a2 (Description, String), a3 (Manufacturer, String), a4 (SKU, String)}
R = {r1 (c1, c5, Association, (0...*, 1), (fulfilled by, fulfilled), <->),
r2 (c1, c6, Association, (0…*, 1), (ordered by, ordered), <->),
r3 (c1, c7, Association, (0…*, 1), (ship to, is used to ship), <->),
r4 (c1, c8, Association, (0…*, 1), (pay via, used to pay), <->),
r5 (c2, c3, Association, (1, 1), (orders form, ordered), <->),
r6 (c8, c10, Generalization, Null, Null, ->),
r7 (c8, c9, Generalization, Null, Null, ->),
r8 (c8, c11, Generalization, Null, Null, ->),
r9 (c6, c11, Composition, (1, 1…*), Null, ->),
r10 (c1, c2, Composition, (1, 1…*), Null, ->),
r11 (c6, c7, Composition, (1, 1…*), Null, ->),
r12 (c4, c3, Composition, (1, 1…*), Null, ->),
r13 (c3, c12, Association, (0…*, 1…*), (product, sales point), <->) }

According to proposed transformation, the first rule is that all classes in class diagram will transform to classes and attributes will transform to attributes in ontology. So all classes in Order Processing class will be defined in ontology as:

DPurchaseOrder = (C, R) where,
C = Set of all classes in purchase order class diagram

R = Set of all relationship exist in purchase order class diagram:

C= {c1 (Order, AOrder),
c2 (OrderItem, AOrderItem),
c3 (CatalogItem, ACatalogItem),
c4 (Catalog, ACatalog),
c5 (Employee, AEmployee),
c6 (Customer, ACustomer),
c7 (Address, AAddress),
c8 (PaymentMethod, APaymentMethod),
c9 (MoneyOrder, AMoneyOrder),
c10 (Check, ACheck),
c11 (DebitCard, ADebitCard),
c12 (Product, AProduct)}

and all attributes of classes in Order Processing Class will be attributes of classes in ontology as
AOrder = {a1 (OrderNumer, String)}
AOrderItem = {a1 (ItemPrice, Currency), a2 (ItemTotal, Currency), a3 (Quantity, Integer)}
ACatalogItem = {a1 (CatalogNumber, String), a2 (Description, String), a3 (Price, Currency)}
ACatalog = {a1 (EffectiveDate, Date), a2 (Expiration, Date), a3 (Identifier, String)}
AEmployee = {a1 (FirstName, String), a2 (LastName, String)}
ACustomer = {a1 (CustomerCode, String), a2 (Name, String)}
AAddress = {a1 (City, String), a2 (Country, String), a3 (PostalCode, String), a4 (State, String), a5 (StateAddress, String)}
APaymentMethod = {a1 (Amount, Currency), a2 (IsPaid, Boolean)}
AMoneyOrder = {a1 (Identifier, String), a2 (Issuer, String)}}
ACheck = {a1 (Agency, String), a2 (CheckNumber, String), a3 (RoutingNumber, String)}
ADebitCard = {a1 (Expiration, Date), a2 (Number, Integer), a3 (Type, String)}

AProduct = {a1 (Brand, String), a2 (Description, String), a3 (Manufacturer, String), a4 (SKU, String)}
According to second rule of proposed transformation, relationships between classes will transform to three types of relationships in ontology. So after applying transformation rule 2, the relationships in Order Processing Class will be defined in ontology as
R = {r1 (c1, c5, non-taxonomic, (0...*, 1), (fulfilled by, fulfilled), <->),
r2 (c1, c6, non-taxonomic, (0…*, 1), (ordered by, ordered), <->),
r3 (c1, c7, non-taxonomic, (0…*, 1), (ship to, is used to ship), <->),
r4 (c1, c8, non-taxonomic, (0…*, 1), (pay via, used to pay), <->),
r5 (c2, c3, non-taxonomic, (1, 1), (orders form, ordered), <->),
r6 (c8, c10, is-a, Null, Null, ->),
r7 (c8, c9, is-a, Null, Null, ->),
r8 (c8, c11, is-a, Null, Null, ->),

r9 (c6, c11, has-a, (1, 1…*), Null, ->),
r10 (c1, c2, has-a, (1, 1…*), Null, ->),
r11 (c6, c7, has-a, (1, 1…*), Null, ->),
r12 (c4, c3, has-a, (1, 1…*), Null, ->),
r13 (c3, c12, non-taxonomic, (0…*, 1…*), (product, sales point), <->)
}

All objects of class in Order Processing Class will transform to individuals in ontology.

VI. RESULT, ANALYSIS AND DISCUSSION

The order processing ontology developed above can be implemented with variety of tools available but in this case it is implemented using Protégé, an open source tool for ontology development. The developed ontology is validated using following methods [2].

Firstly code generated by the tool is validated using Manchester University online validator and the report of validation is generated. The following figure shows the validation report of developed ontology.

Secondly the inferred hierarchy is produced by the Protégé-OWL tool. The following figure shows the inferred hierarchy. Thirdly the ontology graph is produced by the Protégé-OWL tool. The following figure shows the ontology graph.

VII. CONCLUSION AND FUTURE WORK

We have presented the simple transformation for developing ontology from object oriented paradigm. This transformation is very simple and ontology can be easily

Fig. 4: Manchester University Online Ontology Validator Report

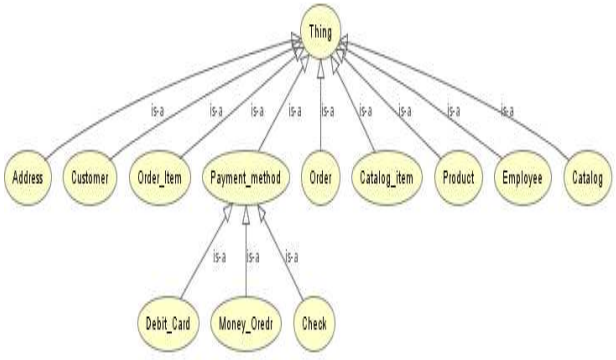

Fig. 5: Inferred Ontology produced by Protégé-OWL tool

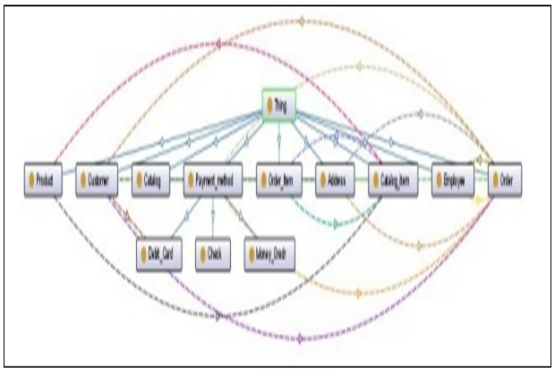

Fig. 6: Ontology graph produced by Protégé-OWL tool

developed form object oriented class diagram by extractions classes and relationships form class diagram and presented these classes and relationships in ontology.

Future work and discussion will be focused on improving this transformation and evaluating it by developing complex ontologies and development of a tool which can automatically develop ontology by taking class diagram mathematical form as input.

REFERENCES

[1] R. Mizoguchi, K. Kozaki, O. Saito, T. Kumazawa and T. Matsui. Structuring of Knowledge Based on Ontology Engineering. Hiroshi Komiyama, Kazuhiko Takeuchi, Hideaki Shiroyama and Takashi Mino (Eds.), Sustainability Science: A Multidisciplinary Approach,, Section 2-3, pp.47-68, United Nations University Press, 2011.

[2] Kouji Kozaki, Takeru Hirota, and Riichiro Mizoguchi. A Quality Assurance Framework for Ontology Construction and Refinement. Proc. of 8th Extended Semantic Web Conference (ESWC2011), pp.305-320, Heraklion, Greece, MAy 29 - June 2, 2011

[3] Barry Smith, Riichiro Mizoguchi and Sumio Nakagawa. Interdisciplinary Ontology, Vol.3, Proceedings of the Third Interdisciplinary Ontology Meeting. Keio University, Tokyo, Japan, February 27-28, 2010

[4] Ian Horrocks. Ontologies and the semantic web. Communications of the ACM, 51(12):58-67, December 2008.

[5] A Software Engineering Approach to Comparing Ontology Modeling with Object Modeling. International Symposium on Computer Science and its Applications (2008)

[6] ChiMu Corporation. Object Modeling, Foundations of O-R Mapping. Computer Science and its Applications, 2008. CSA 2008.

[7] Dr. Waralak V. Siricharoen. (2007) Ontologies and Object models in Software Engineering.

[8] W3 working Group Note. (2006). A Semantic Web Primer for object-oriented software developers.

[9] W. Vongdoiwang, .D. N. Batanov. (2004). Similarities and Differences between Ontologies and Object Model. CCCT'05 proceeding 2004. Austin, Texas.

[10] P. Mohan, C. Brooks. (2004). Learning Objects on the Semantic Web

[11] Rodrigo Bonacin, Maria Cecília Calani Baranauskas, Kecheng Liu. From Ontology Charts to Class Diagrams: Semantic Analysis Aiding Systems Design. In Proceedings of ICEIS (3)'2004. pp.389~395

[12] J. Angele, S.Staab, H. Schurr, Object Oriented Logics for Ontologies. Draft Whitepaper Series, Karlsruhe, Germany, 2003.

[13] D. E. Jenz. (2003). It is High Time for Pursuing the Ontology-Centric Approach Sujoy Paul. Using an UML Class Diagram to Model DAML+OIL Ontology, 2003

[14] Paul Kogut. UML for ontology Development. Journal The Knowledge Engineering Review, Volume 17 Issue 1, March 2002, New York.

[15] Cranefield, M. Purvis. (1999). UML as an Ontology Modeling Language. Proceeding of the IJCAI-99 Workshop on Intelligent Information Integration, Department of Information Science, University of Otago, New Zealand.