Available online www.jsaer.com

***Journal of Scientific and Engineering Research*, 2020, 7(8):41-48**

Research Article

ISSN: 2394-2630
CODEN(USA): JSERBR

An Agent-based Algorithm for Conversion of UML Models to SQL by Means of XML

Bennett E.O.[1], Onwuegbuchulem Gift[2], Sako, D. J. S.[3]

Department of Computer Science, Rivers State University, Port Harcourt, Nigeria
E-mails: [1]bennett.okoni@ust.edu.ng, [2]giftonwuegbuchulem@gmail.com, [3]sunday.sako@ust.edu.ng

Abstract The emergence of the Unified Modeling Language (UML) as a quality for modeling systems has supported the use of automated software tools that ease the conversion of UML models to software codes that make system development very simple and fast. However, these automated tools have problems of inefficient memory management and longer conversion time which causes irrelevant conversion hold up. This research is aimed at designing an agent-based tool that will have memory efficient algorithm with minimal conversion time which will convert UML classes directly to SQL code through XML as an intermediary language. The research simply adopted experimental research design approach and object-oriented methodology to generate an efficient conversion algorithm. The design was implemented using Java programming language tools. Comparing the results of our implementation with the existing conversion tools shows that our objectives were met in terms of quality and efficiency of the process.

Keywords Intelligent Agent, UML Models, SQL, XML

1. Introduction

Agent-based computing represents a novel software engineering paradigm that has emerged from merging two technologies, namely Artificial Intelligence (AI) and object- oriented distributed computing [1]. Agent-based systems aim to strike a balance between artificial intelligence and computational utility. Agents are intelligent, autonomous, software components capable of interacting with others within an application, attaining a common goal and thereby contributing to the resolution of some given problems. They are important because they inter-operate within modern applications like electronic commerce and information retrieval.

Most software engineering paradigms are unable to provide structures and techniques that make it easier to handle this complexity. Thus, there is a reason to develop a framework of software engineering that accounts for the intentional dimensions, namely intents and motivations, goals and reasons, alternatives, beliefs, and assumptions, in its methodologies. Against this background, one will argue that analyzing, designing, and implementing software as a collection of interacting intelligent agents represents a promising approach [2] to software engineering. An agent is an encapsulation of goals, know-how and resources. Agent-oriented techniques provide a natural way for modeling complex systems, by decomposing its problem space into autonomous agents and their interactions.

The Unified Modeling Language (UML), is a standardized modeling language consisting of an integrated set of diagrams, developed to help systems and software developers for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems.

Extensible Markup Language (XML) data maps best onto trees. Structured Query Language (SQL) data maps best onto arrays. These approaches are two totally different ways of looking at the world. Likewise, both relational databases and more object-oriented views of data such as XML can be applied to make sense of and

process the vast amounts of information around us. Both approaches are useful in the process of delivering information to a user. XML is great at structuring data. Relational databases are great at storing and relating data. How do you get the best out of these two different ways of looking at data? You can create a table that contains a row for each part of your XML document.

The aim of this paper is to design a mechanism for the conversion of UML models to SQL through XML as intermediate software in order to identify the correlation between the class diagrams of UML and XML.

2. Literature Review

Some database packages implement a "persistent parse tree" database schema to store XML. What this means is that the entire XML document is decomposed, element by element, into individual nodes, and each node is stored as a separate row in a database table, a table not specific to any particular XML language but able to accept any XML data.

Another method for integrating XML with a SQL database is XML decomposition. An XML decomposition database schema takes the opposite approach to persistent parse trees by creating a schema that entirely encompasses your XML documents. Every element, as well as every single attribute of each element, has a separate table. Rigid relational integrity is maintained, but at the price of flexibility. Both persistent parse trees and decomposition are misguided attempts to integrate XML and SQL tightly.

(a) Overview of Software Agent: This article describes that though there is a rapid growth in the field of agent technology, it is not satisfactory in terms of a software engineering [3].

(b) Conversion of UML models to XML Schema: This explained that the logical-level is a direct one-to-one representation of the XML schema data structures in terms of a UML profile [4].

(c) UML to SQL Present Methodology: The present methodology works with two algorithms – algorithm 1 and algorithm 2. The methodology used the UML design to generate the nested tables of Nested Normal Form (NNF) from UML class diagram by application of the algorithms [5].

(d)Unified Modeling Language (UML): This provides graphical constructs (elements) to build a model of an application, which define the various components of the application, their structure, relationships and behavior [6].

(e) Extensible Markup Language (XML): This comprises a set of tags that defines the dependent meaning of data. XML overcomes some of the disadvantages of the more popular markup language, HTML (Hypertext Markup Language) by simply permitting the user to generate and expound his own tags, this gives the user the flexibility to precisely code and search documents [7].

(f) Goal Oriented Agent Modeling: The desires of agents are completely interpreted in the agent's operations. Not long ago, there is a growing effort in labelling goal modeling publications [8]

3. System Design

This exposes the structure of the architecture, elements, coherence, and data for a system to satisfy specified requirements. Figure 3.1 shows the architecture of the proposed system and Figure 3.2 elaborates more on the sequences of our approach towards generating an SQL code from the UML class diagram.

Validator

This is the first component of our system. We present the codified class diagram in a well-defined structure in the form of mathematical formulation to the validator to validate its correctness using stored validation algorithm

Schema

Hence the validator validates the class diagram; the validated file will be imported into the Transformer component with the stored markup language schema definition (XSD).

Transformator

This component transforms the imported file into XML and further explain the shape of the class diagram using the schema and transfers the XML to the generator component

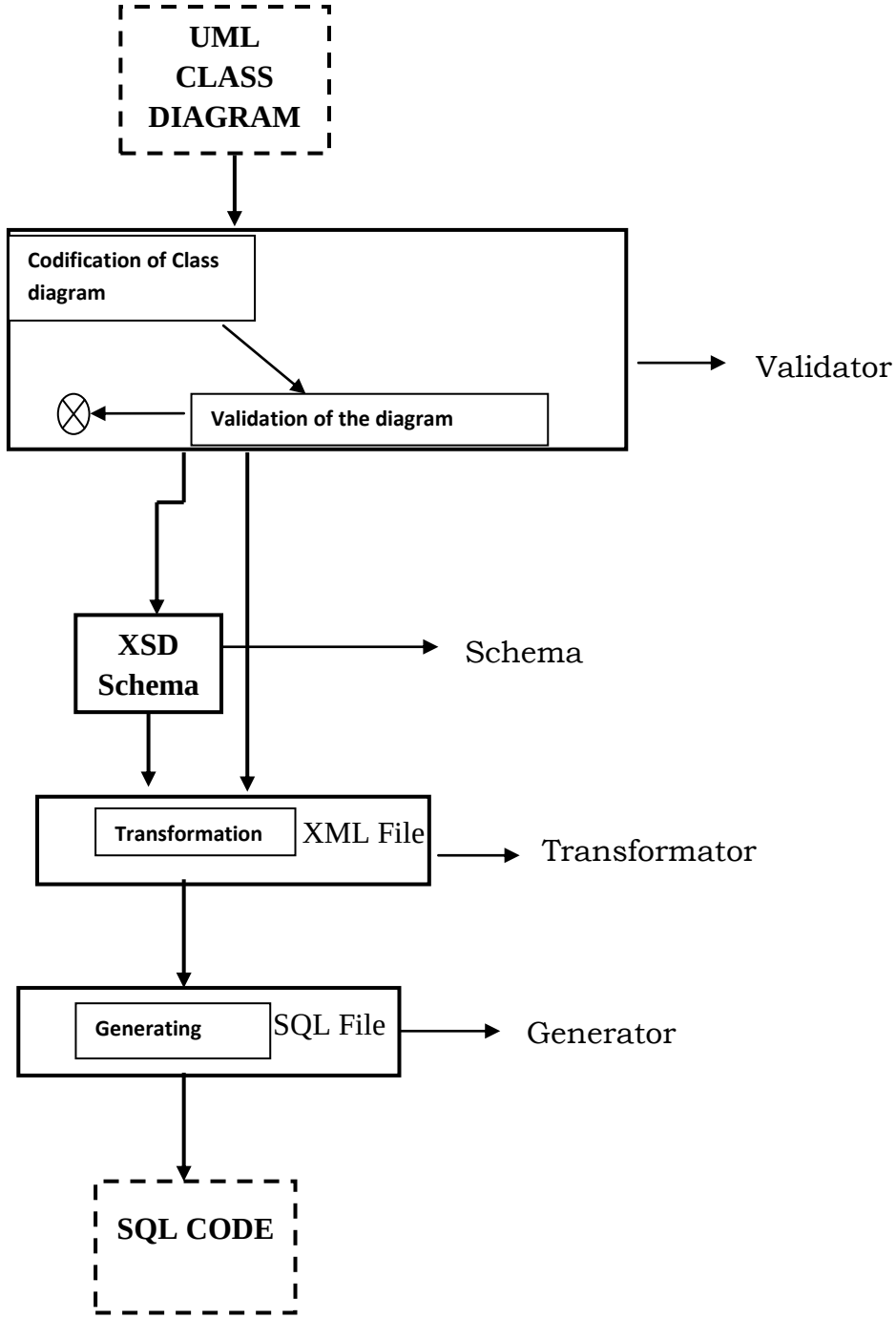

Figure 3.1: Architectural Design of the Proposed System

Generator

The generator component uses its algorithm to generate SQL code from the XML file.

(a) UML Class Diagram

This is the first step in our sequence. The Unified modeling Language (UML) class diagram is drawn using Microsoft Visio application. The application possesses all the tools for making the UML model. Tools such as Class, Member, Separator, Association, Aggregation, Composition, inheritance and separator etc are all there. All these tools are used in Microsoft Visio to create our required UML class diagram

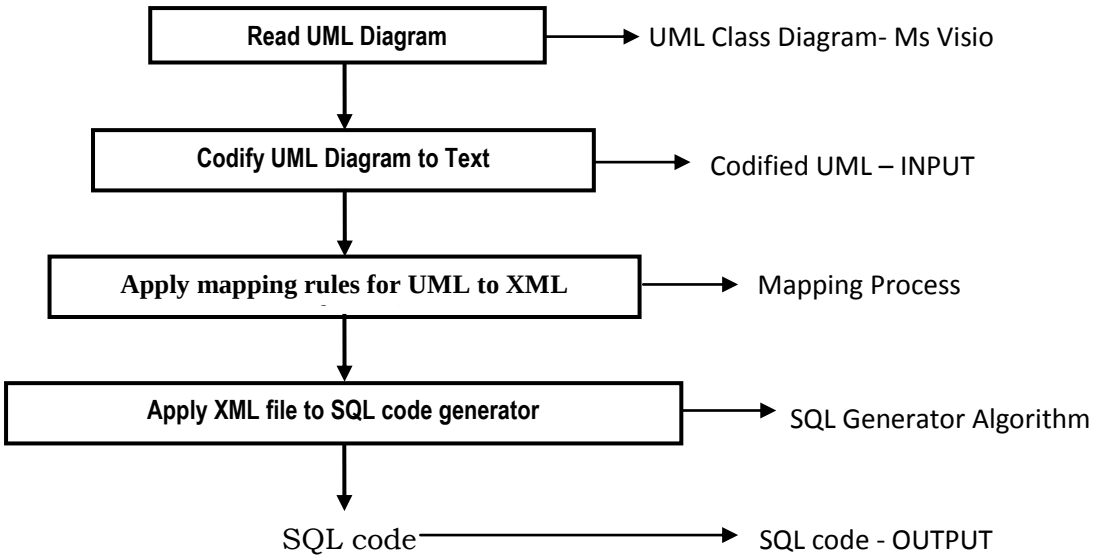

Figure 3.2: Sequences of our Approach

(b) Codified UML - INPUT

This is the second stage where the UML class diagram is changed to text form by knowing the different entities, their attributes and their relationship with each other that make up the class diagram.

(c) Mapping Process

Here the UML class is mapped to an XML element (<xs:element>) and a matching complex type declaration (<xs:complexType>) in XML schema. The XML complexType will be able to carry attributes, method, composition aggregation and generation.

(d) SQL Generator Algorithm

This is the algorithm that generates the SQL code equivalent to the XML mapped from the given UML codified diagram. The output from here is the SQL code.

Algorithm for generation XML Document

Here, an algorithm that assembles XML framework as regards stored XSD was designed.

The system draws out the features and techniques of each class in the algorithm then, go through all the classes to know the kinds of associations that exist between that class and other classes in the schema.

The general algorithm for our approach is:

Algorithm ConvertUMLClass (Main Module)
ReadUML()
ValidateClassDiagram()
GenerateXML()
ValidateXMLDocument()
END.
Algorithm GenerateSQLfile
ReadXML()
GenerateSQL
END

(a) Algorithm ConvertUMLClass(Transformation Module)

C=first Class
While C!=Null
C=C.Next
Create New Element E
Create New Attribute Name
E.Name=C_N
NbA=Number of attributes of the Class C
For i=1 to NbA do
Create New Attribute Attr(i).Name
Attr(i).Name=$C_A(i).A_n$
Create New Element A
A(i).type= $C_A(i).A_t$
A(i).visibility= $C_A(i).A_v$
A(i).defaultvalue= $C_A(i).A_d$
End for

NbM=Number of methods of the Class C
For i=1 to i=NbM do
Create New Attribute Method(i).Name
Method(i).Name=$C_M(i).M_n$
Create New Element M
M(i).type= $C_M(i).M_t$
M(i).visibility= $C_M(i).M_v$
End For

Create New Element Composition
Composition.type= $C_R(i).R_t$
Composition.cardinality= $C_R(i).R_c$
Composition.ClassRelation= $C_R(i).R_r$
Elseif Rel(i) is Aggregation then
Aggregation.type= $C_R(i).R_t$
Aggregation.cardinality= $C_R(i).R_c$
Aggregation.ClassRelation= $C_R(i).R_r$
Elseif Rel(i) is Generalization then
Generalization.ClassRelation= $C_R(i).R_r$
End If
End For
End While

(b) Algorithm GenerateSQLfile(Generator)

Input: Path XML file
Output: SQL-Query sql
Begin
Let P = a1n1a2n2...amnm
FromClause="From"
WhereClause="Where"
For i=1 to m do / Construct From Clause */*
FromClause += "$σ (ni) as Ti"
End For

```
For i=2 to m do /* Construct Where Clause */
If (ai = '//') then
WhereClause += "Ti-1.(ni−1.ID) ≤Ti.(ni.ID)
AND Ti-1.(ni−1.endID) ≥ Ti.(ni.endID)"
Else /* ai is '/' */
WhereClause += "Ti−1.(ni−1.ID) = Ti.(ni.parentID)"
End If
End For
sql="Select Tm.(nm.ID)" + FromClause + WhereClause
Return sql
End
```

Analysis of Algorithm

Our algorithm was analyzed based on time complexity during the conversion from UML classes to SQL code. Supposedly we have only one UML class, the time complexity for it to be converted to SQL code:

$$=time(attr + types) + time(rel + cardi) + time(methods) + time(keys)$$

This signifies that it touches each element in the input which means that its time complexity is linear and can be denoted as n

Total time taken for one UML class = n µs (Micro seconds) - - - - - - (1)

This means that if we have more than one UML classes, the time it takes our algorithm to complete its code conversion is the summation of individual times for each UML class of the input.

$$If\ f(n) = f_1(n) + f_2(n) + \ \ldots + f_m(n)\ and\ f(n) \leq f_{i+1}(n)\ \forall\ i = 1, 2, \ldots m,$$

Then

$$O\big(f(n)\big) = O\big(\max\big(f_1(n), f_2(n), \ldots, f_m(n)\big)\big).$$

$$Where\ f_i(n) = individual\ element\ time\ and\ f(n) = \ total\ time$$

Therefore, our algorithm time complexity is:

$O\big(f(n)\big) = O\big(\max\big(f_1(n), f_2(n), \ldots, f_m(n)\big)\big)\ - - - - - - -$ (2)

= O(n)

4. Implementation and Results

4.1. Minimum Hardware Requirements

The minimum hardware requirements of the system are as follows:

Intel Pentium processor not less than speed of 266HZ – 566HZ , 512MB RAM, 14" super video graphic adapter monitor (SVGA), 160GB of hard disk, A mouse or mouse sensitive used on laptops, CD ROM or USB port, Power or Voltage surge, Uninterrupted power supply (UPS).

4.2. Minimum Software Requirements

The following are the minimum software requirements of the system;

XP or later versions, Jdk6.0 Tool Kit, Microsoft Visio, Visual studio dot net.

4.3. Results

We tested our system with a sample UML class diagram of Fig. 4.1.

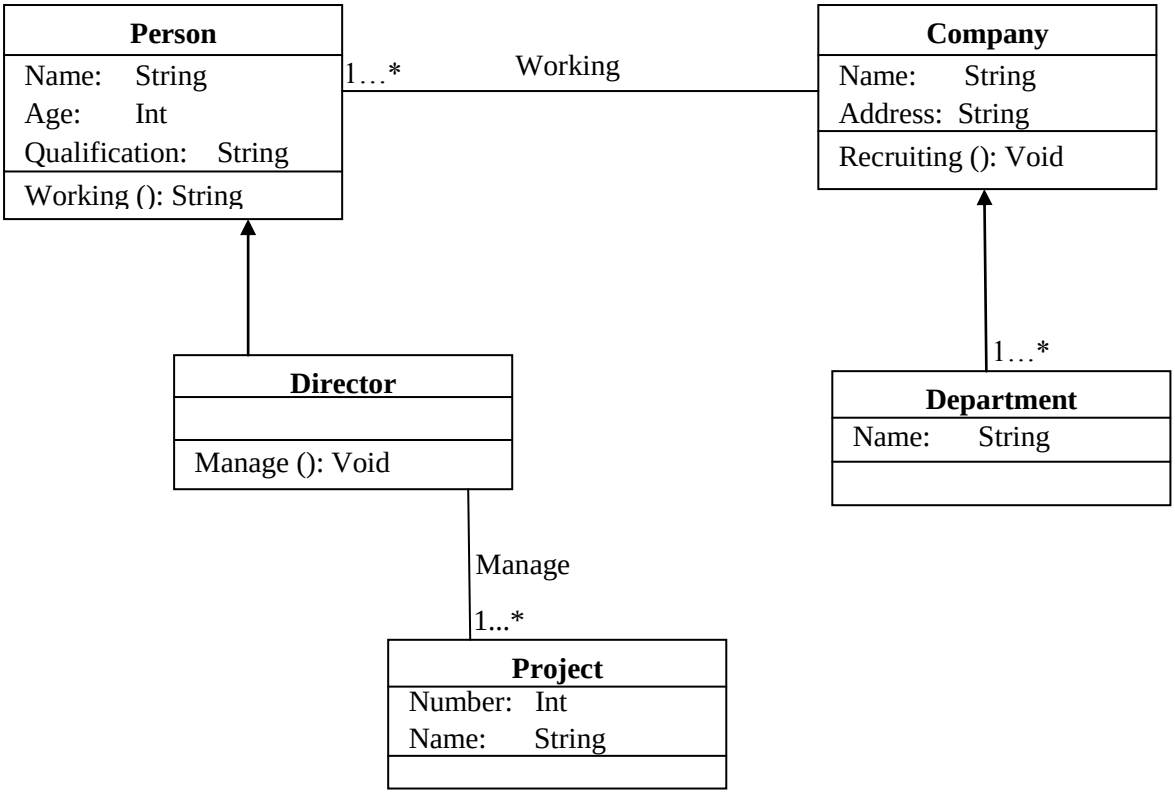

Figure 4.1: Sample UML class Diagram

The input to our software is the codified UML class diagram which is

```
Person;3;Matricule:String:Public:;Name:String:Public:;Age:Int:Protected:;1;Working:String:public;1;1..*:Company;0;0;0;
Company;2;Name:String:Public:;Adress:String:Public:;1;Recruiting:Void:Protected;1;1..1:Person;0;0;0;
Department;1;Name:String:Public:;0;0;1;1..*:Company;0;0;
Director;0;1;Manage:Void:Private;1;1..1:Project;0;0;1;Person;
Project:2:Number:Int:Public::Name:String:Protected::0:1:1..*: Director:0:0:0:
```

Comparative Analysis of our result with Existing Algorithm

To get a feel for the time complexity of our algorithm during the conversion processes, we adopted experimental analysis. We considered our BIG (O) analysis of time complexity which is of the linear order "n" and the existing algorithm time complexity which its order is Linearithmic "n log n" (Breaking up a large problem into smaller problems, solving them independently, and combining the solutions). Table 4.1 shows the comparative analysis of the time complexity of our algorithm to convert UML classes to SQL code and the existing algorithm to convert UML classes to XML code on different time intervals in micro seconds.

Table 4.1: Complexity analyses of two algorithms for UML class conversion

No of classes (Input Size)	Our algorithm (s)	Algorithm mapping UML models to XML schema (s)
5	25	35
10	57	65
15	75	93
20	123	146
25	138	183
30	150	203
35	165	271
40	200	293
45	225	305
50	393	437

Figure 4.2 shows the graphical representation of the data in table 4.1.

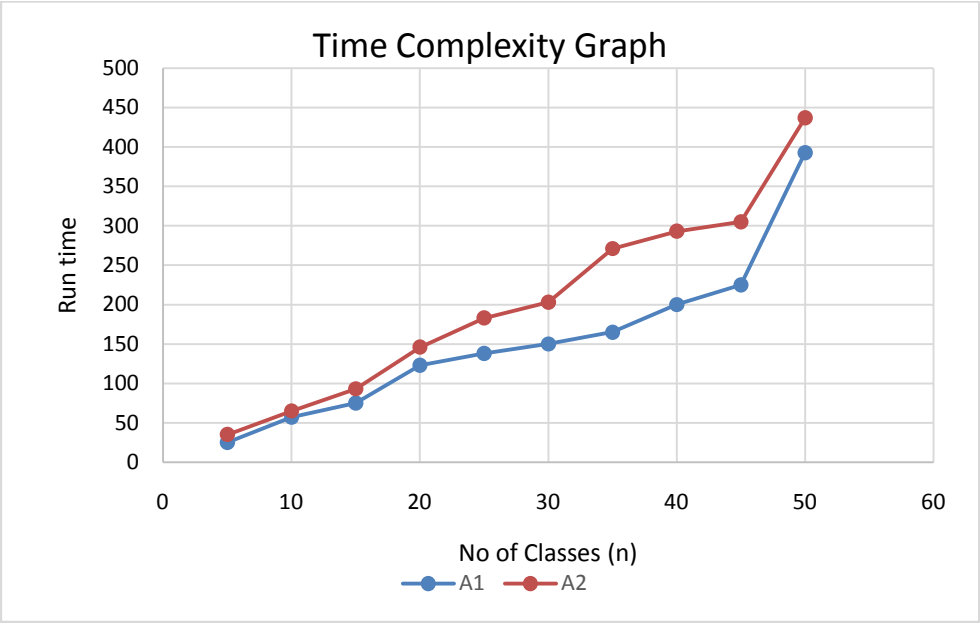

Figure 4.2: Graph displaying the execution time of the two algorithms

The graph shows clearly that our algorithm performs better in time complexity during execution with additional advantage of direct conversion to SQL code compared to what is currently obtainable to takes time in execution and also stops its conversion at XSD.

5. Conclusion

This research has introduced a straightforward clarification for mapping UML object-oriented model into SQL code through XML document using XSD technique. The scheme and the procedure of mapping application were discussed in this research. We have also presented a new approach that produces automatic code from UML class diagram with XML Schema transformation. We have converted UML diagram to text UML and this UML text is converted to XML schema through a tool. The tool also generated the SQL code equivalence of the XML file. We have implemented this approach through a prototype. Then this generated SQL code can be exported to any database management system for database development. This automatic code generation can reduce efforts and development time by reducing coding efforts.

References

[1]. Odell, J., & Burkhart, R. (1998). *Beyond objects: Unleashing the power of adaptive agents*. Tutorial presented at OOPSLA, Vancouver, B.C.

[2]. Wooldridge, M., & Jennings, N. R. (2000). Agent-oriented software engineering. In J Bradshaw (Ed.), *Handbook of agent technology*. AAAI/MIT Press.

[3]. Bernon, Carole Cossentino, Massimo Pavon (2005). An Overview of current trends in European Agent oriented software engineering research, Informatica 29(4).

[4]. Carlson, D. (2001) Modeling XML Applications with UML: Practical e-Business Applications, Addison-Wesley.

[5]. Mok, A., et al (2010) Converting Relational Databases into Object relational Database.

[6]. Booch, G. et al (2013) The Unified Modeling Language User Guide. Addison-Wesley, Reading, MA

[7]. Kurtev, I et al (2003) UML to XML-Architectural Modification.

[8]. Park, S. M (2010). The effects of personnel reform systems on Georgia state employees' attitudes: An empirical analysis from a principal-agent theoretical perspective. Public management review, 12(3), 403-437.