Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 62 (2015) 178 – 185

Procedia Computer Science

The 2015 International Conference on Soft Computing and Software Engineering (SCSE 2015)

Integration of Object Oriented Host Program with Network DBMS

Dr. Shivanand M. Handigund[a]*, Siddappa G. Makanur[b], Dr. M. Sreenivasa Rao[c]

aDept. of Computer Science & Engineering, Bangalore Institute of Technology, BANGALORE, India, smhandigund@gmail.com
bDept. of Information Science & Engineering, STJ Institute of Technology, RANEBENNUR, India, sgmakanur@hotmail.com
cSchool of Information Technology, JN Technological University, HYDERABAD, India, srmeda@gmail.com

Abstract

Several mapping techniques are in use for the storage of objects in Network Database Management System (NDBMS). Though there is a generation gap between the evolution of NDBMS and Object Oriented Technology (OOT), both are either analogous or complementary to each other. Therefore the mapping technique to map class diagram onto Bachman diagram has been evolved. Host program accessing the database and accessing the independent data file may differ in the number and use of attributes and classes. Hence along with the mapping techniques, the implementation subsets of structural and behavioral aspect are to be considered. Moreover, the persistent closure (connected dependent objects) is to be maintained during the storage and retrieval of the objects. Thus, the mere mapping technique is not sufficient for the storage and retrieval of objects as the host program has to establish the relevancy of the database with respect to its authorized subset. In this paper, we have made an attempt to develop a guidelines to assist the programmer to determine the closure of every mapped class and accordingly to design the persistent constructor and loader for the storage and retrieval of the objects respectively. The persistent closure is determined by the type of interrelationship (degree of cohesion) between objects (classes) is identified. For illustrative portion of the class diagram depicting the business process and its implementation in the NDBMS, a table containing the implemented classes with their connected classes is prepared, and then the guidelines are proposed to integrate object oriented host program to access the database through NDBMS.

Peer-review under responsibility of organizing committee of The 2015 International Conference on Soft Computing and Software Engineering (SCSE 2015)

Keywords: Transient object; persistent object; object to database mapping; host program; Network DBMS; CODASYL; Record Type; Set Type; User Work Area; Activation Depth; Persistent Closure; OID.

* Corresponding author.
*E-mail address:*smhandigund@gmail.com

Peer-review under responsibility of organizing committee of The 2015 International Conference on Soft Computing and Software Engineering (SCSE 2015)
doi:10.1016/j.procs.2015.08.435

1. Introduction

The goal of this paper is represented in the form of vision, mission and objectives as follows. *Vision*: To design an efficient methodology for the ossification of Object Oriented host program to NDBMS. *Mission*: To develop a methodology that establishes the transformation between the program content (authorized part of the database) with the business process database in the NDBMS.
Objectives:

- To develop the guidelines set to identify the dependent objects (closure).
- To develop the guidelines set to design persistent constructor and persistent loader to keep the object in the valid state.
- To establish the mapping between the programmers subset of database with the business process database.

The class to NDBMS mapping method[18] stops at producing the schema definition and does not address the issue of host program integration and persistent closure that is to be maintained during the retrieval of objects from the NDBMS along with the underlying mapping used. At present these activity is being done by the programmer without any guidelines.

2. Related work

Today's information systems are designed and developed using an object oriented techniques and have used UML as a de facto standard for design diagrams. However for the storage of objects the Database Management System (DBMS) product used is not an Object Oriented Database Management System (OODBMS) in most of the cases[18].

Amongst the DBMSs the NDBMS is the most efficient[7] one and Database Key (unique in the database space) oriented and not a data oriented. The NDBMS built on CODASYL (Conference On Data System Languages) standards has the architecture where the database schema DDL (Data Definition Language) is independent of host language and the subschema DDL is for the specific programming language[8,14,15]. The Object Data Management Group (ODMG) has recommended[19] the OOBMS's features such as Object Interchange Format, Object Identity (OID), Language integration and database entry points are to be supported in the object oriented data model, these features are supported in NDBMS by User Work Area(UWA) Templates, Database Key, Language specific subschema, system owned set type respectively. Thus mappings with NDBMS are more suitable for the storage of objects. The NDBMS is developed much earlier but NDBMS has analogous[10] as well as complementing features to OOT features. Accordingly mapping techniques and methodology was developed to transfer Class Diagram to Bachman Diagram (pictorial representation of network database schema) and database schema definition[18] which are used to store the objects using NDBMS.

In a certain attempt made to integrate object present in the host program (transient) and persistent objects present in the database together with pointer swizzeling techniques[16]. The swizzeling technique converts OID to POID and vice versa. This technique is very complex and how closure property is protected is not clear. In the middleware like Hibernate only mapping is supported and no considerations for the closure property[3] during the storage/retrieval. In product such as Enterprises NDBMS on ClearPath OS where the host program can be written using Java[4], the subschema is designed without considering the object orientation. There are no guidelines or methodology to enforce persistent closure during read/write operations available. Thus there is a need of methodology or guidelines to transform persistent object to transient object and vice versa, that helps the object oriented program integration with NDBMS.

3. Proposed application program integration

3.1. Introduction

In case of object oriented host program, object contains reference attributes connecting or pointing to the address (similar to OID) of other objects using which different relationships are implemented. Hence just equating the attributes of object with the data items of the record object received from the database leaves the reference variable

that may exist in the object undefined. When such connected objects are required to be present in the program memory and to stay connected (closure) to put the objects in the valid state. There is a need of specialized loader that can take care of placing the object along with its must connected objects in the host program's memory. Thus the objects are to be transformed from persistent state to transient state and similarly transient object is to be made persistent as per the mapping techniques used using persistent constructor. In this paper an attempt has been made to provide guidelines or methodology to design such constructors and loaders for each persistent class.

In this paper all the classes are assumed to be the persistent capable classes whose objects are both transient and persistent and we will drop the word persistent whenever there is no confusion. The class diagram of the information system is expressed using UML standards during the design phase. The classes are designed based on the entity concept and normalized[1]. This class diagram is represented graphically using UML standards, which is more abstract, independent of programming language and lack of implementation guidance. During the implementation the object oriented features are implemented using the features (or constructs) of the programming language chosen to implement. During implementation, this class model expressed in UML is transformed into implementation class model specific to that application program (with business process) and accordingly coded. Following example (shown in Fig.1), illustrates how classes in the UML class diagram are transformed into implementation classes in the object oriented host program (ignoring the methods and the visibility) for a College information system.

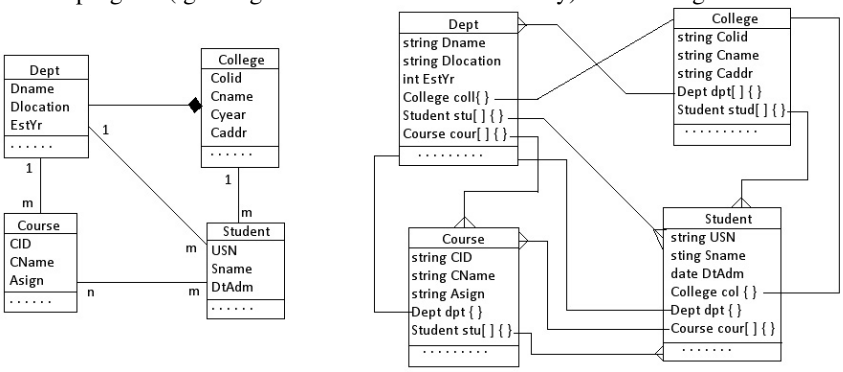

Fig.1 Class Diagram and its implementation in the host program

3.2. Implementation of an association

There is no direct construct is available to code an association like an inheritance. The programmer has to make suitable constructs to realize association and has to take care of the constraints[5,17]. Normally associations are implemented by embedding the reference attribute of the participating class type (called as pseudo attribute) in a class[12]. We will use the notation { } to represent the reference or pointer or OID of the connected object[6]. The example in the Fig.1 shows UML class diagram and its implementation class diagram used in the typical object oriented host program.

Aggregation is a specialized form of an association that "has-a" or "whole-part" kind of a relationship between aggregate class and aggregated class in which all its objects have their own lifecycle. The program implements aggregation similar to the association through pseudo attributes which are referencing to part objects, i.e. the aggregate class contains its part class represented by reference attribute of that type[17]. Composition is a stronger form of aggregation with additional constraints such as part belongs to at most one assembly and life time of container object and content objects remains the same[16]. The constructor of composition object has to enforce these constraints by making the pseudo attribute of part type[17]. Thus association, aggregation and compositions are implemented as bidirectional association in the object oriented program.

3.3. Persistent object store (using network database)

An information system comprises of several subsystems and each sub system is implemented as an application program. Normally information system has its own persistent objects requirements and expressed as the persistent

class diagram and it mapped database schema. The individual application program (host program) has its own database subschema. Thus an information system is built using two different domains, one the computational model of a programming language, other the database model to store, retrieve and share large amount of data[16]. The host program has to use appropriate mapping techniques to store the objects created in the program into the database or object store. The programmer has to store/retrieve objects explicitly from the object store using suitable data manipulation commands and protecting persistence closure.

3.4. Mapping techniques used

In NDBMS, all the associations are binary with cardinality ratio 1:m and represented by the set type (not a mathematical set and also called as named set type) that connects two different record types that are associated. We use set type in place of named set type whenever there is no confusion. Thus an association is implemented directly as set type that implements one: many association in which record type corresponding to the class on cardinality one is represented by the owner and record type corresponding to the class on cardinality many is represented as member of the set type. In the input class diagram the association is represented by a connecting line & normally name is not assigned. The name of the set type will be generated by the concatenation of first three letters of the owner name and last three letters of member name. Thus at the semantic level 1: m association is mapped to set type directly with its participating owner & member along with other constraints[18]. These mapping provides the host program the way how to store the object in the database and to read/write the record objects from the database. The following table (Table 1) summarizes the mapping techniques used in this paper.

Table 1 Mapping Summary

OOT Feature	Mapped to NDBMS	Details	In the host program
Class	Record Type	Class attributes are directly mapped to data item of record type with same name and data type	Class
Association 1:m	Set Type	Record type corresponding to the class at cardinality one is the owner of the class and record type corresponding to class at cardinality m in the mapped set type.	Reference attribute
Association 1:1	Set Type	1:1 is represented as 1:m	Reference attribute
Association m:n	Two Set Types 1:m and 1: n with the help of *dummy* record type	In one set type the record type at cardinality m is the owner and dummy record type is the member. In other set type record type at cardinality n side is the owner and same dummy record type is the member	Reference attribute
Composition	Set type	The record type at whole is the owner and record type at the part is the member.	Reference attribute

Using the above mapping techniques, the class diagram shown in the Fig.1 is transformed into the Bachman Diagram (NDBMS schema) [18] as shown the following Fig. 2.

An application program or host program is written using specific Object Oriented Programming Language (OOPL) that uses its own constructs to implement OO features. Inheritance is supported directly in the OOPLs but no constructs to implement association, aggregation and composition are available. These features are implemented using reference attributes that refers the related objects[5,6,9,17]. Accordingly these mappings should be used to read, write, update and delete operations with the closure property maintained. Thus the host program has to consider the mapping techniques to store object along with the it implemented classes and their closure. The diagrams (Fig.2 and Fig.3) show the relationship between the mapped NDBMS schema and the object model implemented in the host program in NDBMS environment. The application program interacts with database through User Work Area (UWA), which is the bridge between the transient objects and their underlying record types & set types that are used in NDBMS on the basis of mapping techniques used.

Hence specialized constructors to transform the object from its transient form to the persistent form using the underlying mapping techniques are used. Conversely persistent loader has to read a persistent object from the database and transform it to its transient form based on the underlying mapping technique and programming technique.

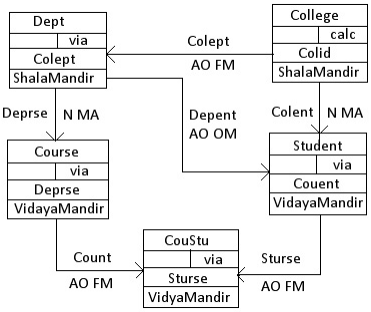

Fig.2. Mapped Bachman Diagram

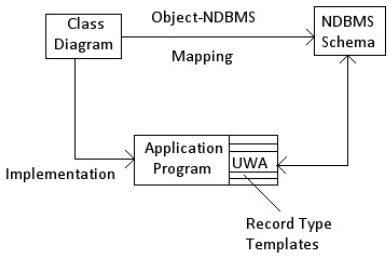

Fig.3. Application program interacts with NDBMS through UWA.

3.5. Persistent closure

The Persistence Closure principle states that any storage and retrieval mechanism must handle the objects and all its direct and indirect dependents together[11]. This object and its dependent objects (such as full participation, aggregation or composition) make up the closure. The object may have dependent objects, these dependent objects may have their dependent objects (indirect) transitively. This forms tree of dependent objects with main object as the root. This size of the tree may become so large that it may not fit into the program memory. Thus tree size of the dependent objects that are to be loaded from the database to the program is restricted with the help of the activation (levels of transitivity) depth, i.e. how many dependent objects are to be loaded along with root object[2] into the program. Thus every class has a closure property which defines the dependent record objects which are to be handled along with during the storage and retrieval.

Activation depth denotes the length of the reference chain from an object to another. Objects beyond the activation depth are not loaded into the memory treating them as inactive[13]. This activation depth is fixed based on the persistence closure required by the application requirement. To help the programmer to determine the closure for each class, the closure table is to be designed based on the degree of participation or cohesion amongst the classes and their program implementation. Following sections discusses the issue of determining the persistence closure and how to access these objects.

3.6. Determination of persistent closure

The closure of class is mainly application dependent or based on the business rules. However in case of non availability of such details, the structural property of the class diagram such as cohesion between the classes can be used. In the following section how this property is used to determine the closure of a class is discussed.

For Association: The closure property for the association is determined by the degree of participation of the connected objects such as full participation or partial participation. Partial participation exists between loosely coupled (low cohesion) objects and full participation (high cohesion) exists between tightly coupled objects. Thus cohesion is to be considered while determining the objects in the closure in addition to the application semantics. In case of total participation the object is to be present along with its connected object and should be stay connected to keep the object to be in a valid state. The aggregation is treated as association with partial participation of the part objects.

For Composition: The composite (whole) object consists of other (part) objects. The part object does not exists on its own but always attached to its whole. Thus presence of part objects are depend on the presence of the whole and conversely whole object will not be a complete one without its part objects.

The activation depth is application dependent based on the cohesion amongst the objects. This cohesion can be identified from class diagram to identify the dependent objects which will be the direct dependents. From this dependent object using their cohesion (semantic) property indirect objects can be identified. Thus the activation depth or transitivity moves deeper and deeper which is to be restricted based on the semantics of the application.

The following Table 2 helps the programmer (as a tool) to determine dependent objects and how to reach them. The entries of Table 2 are based on the implemented classes and their mapping used for the storage of objects using NDBMS which is based on the example shown in the Fig.1 and Fig. 2 respectively.

From the below Table 2 for each class the closure can be determined. For example the class Dept has three reference attributes, each represents the relationship it has with other classes. These reference attributes are mapped to corresponding set types. The connected objects are easily accessed by traversing its set occurrence. Now which of these relationships hold the required objects that are to be handled along based on the business rule or application semantics and the class diagram, accordingly entries are made in the Table 2. Here in a college environment the Dept is always attached to the College and its closure property is Y, i.e. if Dept does not connected to a College object then the state of the Dept object is invalid. Thus if the connected record object is part of closure that record is be accessed through the corresponding set type as the owner or the member as the case may be. Here Dept object should be stay connected to its College object, if this is not present in the program memory it is to be loaded to the host program memory. Transitively from the connected College object its dependent (indirect) objects are to be made part closure at the second level if the activation depth of reference attribute Colept of Dept is two. Normally activation depth is one (immediate dependents are enough) in most of the cases.

Table 2. Implementation Mapping.

Sl (1)	Class Name (2)	Mapped to Record Type (3)	Type of Ref Attribute (4)	Closure (5)	Mapped to Set Type (6)	Refers to Owner/ Member of set type (7)	Transitivity Depth (8)	Remarks (9)
1	Dept	Dept	College	Y	Colept	Owner	1	Attached to College
			Student	N	Depent	Member		Optional
			Course	N	Deprse	Member		Optional
2	College	College	Dept	Y	Colept	Member	1	Composed of Depts
			Student	N	Colent	Member		Optional
3	Course	Course	Student	N	Couent	Owner of Dummy* in Sturse		Optional
			Dept	Y	Deprse	Owner	1	Attached to Dept
4	Student	Student	College	Y	Colent	Owner	1	Attached to College
			Dept	Y	Depent	Owner	1	Attached to Dept
			Course	Y	Sturse	Owner of Dummy* in Couent	1	Attached to Course

* Represented by CouStu and always a member and need not have any data.

This determination of closure can be generalized as follows. For each class C in the program (column 2) which has one or more reference attributes R_i referring other classes C_i(column 4). For each such reference attributes closure property (column 5) connects the other object as dependent and connected as the owner or a member (column 7) through its mapped set type S_i (column 6). Then record type of C_i as an owner/member of the set type will be part of the closure. Populate the attributes values of object of such connected class through S_i in their templates of UWA. Now the templates in UWA contain the attributes of connected objects. Following are the guidelines to identify such closure.

Input : Table 2. Output : Closure (list of dependents) for each class.

1. For each class C the Closure = C ; // self (in col 2) or root object
2. for each reference attribute type R_i (in col 4) in Class C // i =1..n # ref attributes in C
3. If R_i.closure = 'Y' then (in col 5) //connects to dependent object
4. Closure = Closure U C_i // add related object
5. If (transitivity of R_i (in col 8) > 1) then repeat steps 2 to 4 for C_i in place of C for transitivity of R_i-1 times
6. End if
7. End if
8. Next for // next ref attribute R_i
9. Print the contents of Closure of C // now Closure contains the connected classes of C.
10. Next for //next class C

11. End

While accessing the object of a class all its connected objects of other classes should be handled along. In the following section we will discuss how to implement read/write operation with the connected objects.

3.7. Persistent constructor

The application program will interact with the database according to the attached subschema through the UWA. This UWA is created during the compilation process and acts as a database gateway. UWA contains the templates (place holder) (Fig. 3) for each record type in the attached subschema and corresponds to the program variable. While storing, the program has to copy the attribute values(dropping the reference types) of the object to the corresponding data items of templates in UWA then issue Data Manipulation Language (DML) command such as STORE after locating the its position in the database using FIND command.

The object structure in the program and the object structure in the database are bridged via mapping techniques adopted. While storing and retrieving the objects, the objects are transformed from transient form to persistent form and vice versa according to adopted mapping.

The state of the object in the program (transient object) is to be transformed into its record type occurrence or record object (persistent) into its template in the UWA then stored in the appropriate set type occurrences or location in the database. Thus, there is a need of specialized constructor that has to take care of populating object states along with states of its related (closure) objects into the corresponding templates in the UWA. These related templates are used to FIND the location where the record object should be stored as a part of the network database. Thus a persistent constructor for a class C has to do the following tasks and tasks are designed by refereeing Table 2, which is prepared for the given implementation. Following are the guidelines to design persistent constructor used to store the objects.

1. Identify the mapped record type K_c for the class C //root
2. Populate the attributes values of C into the data items of template for K_c in the UWA.
3. For each reference attributes R_i in C repeat steps up to step 5
4. If the closure of the reference type R_i is 'Y' then the owner/member class C_c of the mapped set type will be in the closure. Then populate the attributes values of Cc into the data items in the template of the mapped record type R_c.
5. If the transitivity depth of R_i >1 then the next level (indirectly) connected objects will also be part of closure. i.e. closure of C_c is also to be added to the closure of C. the dependent object to class C_c are also be loaded into their templates in UWA using the similar procedure (steps 3 to 4) using C_c in place of C.
6. Now object and its connected objects contents are stored in their templates in the UWA (complete closure).
7. These record templates values are used to identify proper set occurrences (using FIND) where the object(root) is to be stored as the record object in the database ensuring closure property, then issue the STORE command to make record object as a part of database. If such connected record objects are not at all present in the database. First store such record objects similarly using the above procedure then store this object (root).

Thus the transient object is converted to its persistent form and made part of persistent object collection or database.

3.8 Persistent loader

While reading (loading) the record from the database, the record object is read into its template of UWA, this template connected to the variable in the procedure oriented host program and equated. But in case of object oriented host program, the state of the stored record object is read into its template of UWA. From the data items available in the template, its transient object is to be created using the procedure similar to parameterized constructor in the host program environment.

Thus the state of persistent object (record object) from the database has to be transformed into its transient form and made part of transient objects of the host program protecting the closure property. To do this task a routine such as persistent loader is required for each class to read objects.

Following are the tasks that are to be performed by the Persistent Loader refereeing the Table 2.

1. Read the required record object R of class C from the database into its template in the UWA using suitable DML commands. //root object
2. From the implementation mapping table (Table 2) identify the connected record objects through the reference attributes R_i of type C_i and their mapped set type S_i.
3. For each set type S_i the connected record object may be the member/owner
4. If closure property of R_i = 'Y' then
5. Read contents of connected record object into template R_i of C_i
6. End if
7. End for // next R_i
8. Create the transient (root) object of C along with its connected objects of class C_i s using the procedure similar to the parameterized class constructor receiving the parameter (data items) from the templates of C and C_i in the UWA. Inside this constructor of C, equate data items to their corresponding attributes, next for each reference attribute R_i whose referring class is in the closure of C, create the object of C_i (recursively if transitivity is >1) and equate its reference (i.e. OID) to the reference attribute R_i of C. If specific R_i is not in the closure then set it to null.

Thus object is made part of transient object collection in the host program.

4. Conclusion

In this paper we have made an attempt to identify the dependent objects (closure) that are determined with help of the implementation mapping table. The guidelines are framed to transform transient objects to persistent objects (persistent constructor) and vice-versa (persistent loader). Thus we have attempted to keep the object in the valid state protecting the persistent closure. Persistent constructor/loader in case of implementation of inheritance/multiple inheritances in the host program along with their mappings are to be addressed.

5. References

1. Ajeet C and Dr. Shivanand M. H, "An Automated Methodology to Design a Clustered Class", International Journal of Advanced Research in Computer Science, Vol.2 , No. 3, May-June 2011.
2. Alan. Dearle, G. N. C. Kirby, and R. Morrison. "Orthogonal persistence Revisited". In Proceedings of the Second international conference on Object databases, ICOODB'09, pages 1–22, Berlin, Heidelberg, 2010.Springer
3. Christian Bauer and Gavin King, "Java Persistence with Hibernate". Second Edition of Hibernate in Action. 1-932394-88-5, Manning Publications Co.. Manning Publications Co. http://www.manning.com/bauer2.
4. ClearPath Enterprise Servers, Enterprise Network Database Server for ClearPath OS 2200 Subschema Data Definition Language. Administration, Operations, and Programming Guide Level 20R1, November 2006, Part no 7831 0752–005. By UNISYS Corporation, USA.
5. D Akehurst, G Howells, K MCDonal-Maier, "Implementing Associations: UML 2.0 to Java 5", Software Systems Model, Springer-Verlag, DOI 10.1007/s10270-006-0020-1, Apr 2006.
6. David Johnson, "Association Implementation, the Key to Efficient Persistence", Journal of Object Technology, Vol.4, March-April 2005.
7. Donald K. Burleson, "Inside the Database Object Model", CRC Press, NW, USA, 1998.
8. Elmasri and Navathe, " Database Systems" 2nd Edn, Pearson Education Publisher, 1994.
9. Gonzalao G, Carlos R C, Juan L, "Mapping UML associations into Java code", Journal of Object Technology, Vol. 2, No.5, Oct 2003.
10. Malkolm Atkinson et al, "The Object Oriented Database System Manifesto", First International Conference on Deductive and Object-Oriented Databases, Kyoto, Japan, 1989.
11. Meyer Bertrand, "Object Oriented Software Construction", Second Edition, Interactive Software Engineering Inc, USA, 1997.
12. Michael Blaha, James R Rumbaugh, " Object Oriented Modeling and Design with UML" Second Edn, Pearson Education Inc.
13. Michael Grossniklaus, Moira Norrie, Lecture Notes, " Object-Oriented Databases db4o: Part 1", ETH Zürichhttp://www.globis.ethz.ch/education/oodb/slides/03-db4o-part-1.pdf
14. Prime Computers Inc, "The DBMS Administrator's Guide", PDR3276, Massachusetts, USA, 1979.
15. Robert W Tailor and Randall L F,"CODASYL Database Management System" Computing Surveys, Vol 8, No1, March 1976, ACM.
16. Thomas Connolly and Carolyn Begg, "Database Systems", IV Ed Peasrson. 2005.
17. Yann G. G et al, "Bridging the gap between modeling and programming language", 2002, http://www.ptidej.net /publications/documents/Research+report+AAC+July02.doc.pdf.
18. S. M. Handigund et al, "An Ameliorated Methodology to Establish Compatibility between Structural Parts of Object Oriented Technology and Network DBMS", The International Journal of Soft Computing and Software Engineering [JSCSE], Vol. 3, No. 3, 2013.
19. Rick G. G. Cattell(Ed) et. al. Object Database Standard (ODMG 3.0), Morgan Kaufmann Publishers, 1999.