Object Oriented Database a Design Pattern

Talha Aziz[1] and Naeem Janjua[2]
SZABIST Karachi

ABSTRACT:*Object Oriented Database Management Systems (OODBMS) differ from Relation Database Management System (RDBMS) as data is stored as objects instead of rows and columns of a table. Inherited objects are stored with parents object's attributed and methods which has awful effects on memory and database as hierarchy grows deep and objects become heavy. For avoiding the heavy objects a pattern was discovered recently named CHNFP (Class Hierarchy Normal Form Pattern). It follows the concept of CHNF as well as it validates the CK Metrics[1]. In this paper, we propose a way to make it even more effective.*

1. INTRODUCTION

Object oriented database, lack a standard for describing the inheritance hierarchy of the classes schema [1]. With object oriented techniques comes features such as polymorphism, encapsulation, inheritance, and modularity. The CHNF (class hierarchy of normal form), is a methodology for implementing the normalization of classes' schema inheritance hierarchy. It is a good approach but the difficulty occurs for the design when object oriented databases become complex. Because when an object with deep hierarchy tree loads into the memory, it becomes heavy due to the depth in inheritance. The Object loads all its super class attributes, properties and methods upto root class. When this complex object goes into execution, it causes data redundancy and results in memory wastage. It would be effective if only required data is loaded into memory at the time of loading of object instead of loading duplicated data. Therefore it is better to split the object into smaller objects and load the object on the grounds of requirement. CHNFP pattern was discovered recently which offers a methodology to maintain inherited objects effectively in OODMS. Building upon it, we propose a pattern to store inheritance relationship between parent and child in such a way as to minimize data duplication and thus making the CHNFP pattern even more efficient and effective. Since the primary introduction of design patterns in, they have been widely adopted by software industry. Many software developers routinely apply design patterns in their software systems to reuse expert design experience and record design decisions. Several approaches on design pattern discovery have been proposed in the literature. Discovering the design patterns applied in a software system may help on not only the understanding of the system but also its maintenance and evolution [2]. Section II presents the related work. Section III details.

2. RELATED WORK

The Class Hierarchy Normal Form (CHNF) normalizes the class's schema. The object oriented models, inheritance semantics mainly expresses the class hierarchy. It is important to ensure and maintain an appropriate class hierarchy structure to fully express the inheritance semantics. An improper class hierarchy structure will function initially but, when a class's schema evolution occurs, an unreasonable structure that may cause loss of information, confusions in semantics and storage anomalies in implementation will appear [3].

The CHNF uses the concept of Boolean algebra for defining the schema of class hierarchy and deal with redundant class schema definitions so that a better class schema with no such redundancy can be obtained [3].
The CHNF proposes the paradigm of normalize class schema and the other hand we have another pattern for mapping the objects with relational data base, the concept of ORM (Object Relational Model) is a bridge between Object Oriented and Relational Model. ORM is a programming technique for converting data between incompatible type systems in relational databases and object-oriented programming languages. This creates, in effect, a virtual object database which can be used from within the programming language. [4].

ORM also have the concepts of Table per Class Hierarchy, Table per Concrete Class, and Table per Class, Relationships, and Factories [5].

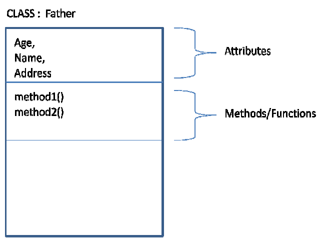
ORM defines the class schema which bases on relational databases, and the relational databases follow the normalization. EF (Entity Framework) is an object relational mapping (ORM) framework for the .NET Framework. This framework is an ORM offering from Microsoft for the .NET Framework. While Microsoft provided objects to manage the Object relational impedance mismatch [6]
.
It is difficult and probably incorrect to classify EF as an ORM because it is far more advanced than that. EF strives to accomplish something more sophisticated than your standard ORM tool by embracing the conceptual model as something concrete. EF accomplishes this by providing a framework for creating an abstract model on top of your relational model, hence overcoming the impedance mismatch. The nucleus of EF is its layer of abstraction that is divided into conceptual, mapping, and logical layers making up the EDM. In addition, EF utilizes two APIs, object services and the entity client, for working with the EDM and also uses two data manipulation constructs, Entity SQL (ESQL) and LINQ to Entities [5].
Other less known proposals in which objectoriented concepts are derived from structured development can also be mentioned. Some of these methods were merely extensions of structured development techniques. Masiero and Germano and Hull et al. put together object-oriented design with JSD, and the by-product of the design is

implemented in Ada. Bailin and Bulman combined objectoriented development with Structured System Analysis and the Entity-Relationship Model in an object-oriented requirements specification model. Lastly, Alabiso and Ward combined object-oriented development with Structured Analysis, Structured Design and the Entity-Relationship Model [7].

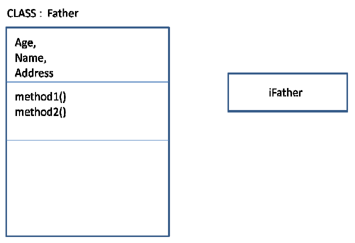
3. CHNFP

The CHNFP pattern discovered recently defines three basic rules:
Rule one: Every Class has its interface.
Rule two: Every Sub-Class will implement its parent interface.
Rule three: Every Sub-Class has no null constructor and has constructor of at least one parameter of its parent id.

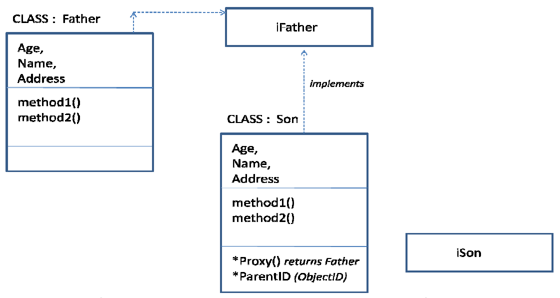
For example, we have a class Father with some attributes and methods.



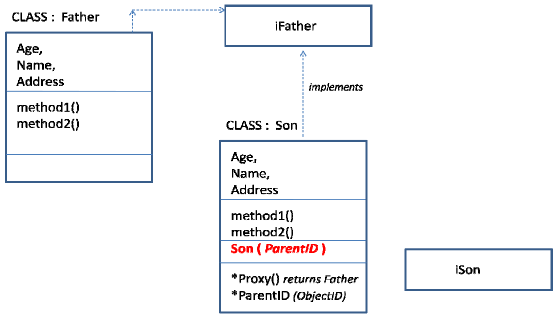
According to Rule One of the pattern, it should have an interface iFather.



According to Rule Two of the pattern, every sub class should implement its parent's interface.



And the final rule makes it compulsory for the child class to implement at least one non null constructor which binds it to the parent's object.



So, in a way, a *virtual proxy* has been implemented rather than actual inheritance. When object of class Son needs to use Father class' methods/properties, it can query the Father's object from the OODBMS by using proxy method with something similar to as follows:

```
Son objSon
      =    new    Son(<<    ParentID    /
getFatherObjectID >>);
objSon.Name; -> returns son's name
objSon.Proxy.Name;   ->   returns   son's
father name.
```

The advantage of this pattern is that the Parent Class (Father)'s data is stored in a single object which is referenced by all child (Son) objects. Whereas, in storing an object of inherited class stored data physically in OODBMS for attributes from all parent classes. This means that only one instance of the parent's object is stored in the database instead of it being stored with each child object reducing the size of the database considerably. Also, the parent object (it's methods or attributes) are invoked only when needed and thus making querying faster.

4. THE PATTERN

We have discovered an enhancement to the CHNFP Pattern. Introducing a utility class which maintains the relationship between parent and child facilitates in eliminating the need to coerce the programmer to implement a non null constructor pointing to the parent's object.

A. The Utility Class

The utility interface, say *Util.Inheritance*, contains the *inheritsFrom(object).* Any class that needs to inherit a parent class, implements this Util interface. When an object is to be associated with a super / parent class, the *inheritsFrom()* method is called and object id of the parent is passed. Another method *getParent()* will return the parent's object of the child if associated.

B. XML Repository

The relationship is maintained in an xml repository of the following structure.

```
<Util>
<Inheritance>
<parent objectid='xxx'>
<child objected='xxxx'/>
</parent>
</Inheritance>
</Util>
```

In this way, the logical data definition file maintaining the relationship information between objects will be created. Whenever a child needs to call any of the methods or attributes of its parent, *getParent()* method of the *Util.Inheritance* will be invoked.

e.g.

Say we have class *Son* with some methods and attributes.

```
Class Son
{
    attr1,
    attr2,
    attr3,
    method1(),
    method2()
};
```

And another class called Father

```
Class Father
{
    F_attr1,
    F_attr2,
    F_attr3,
    F_method1(),
    F_method2()
};
```

If this class is to be inherited from the class *Father*, the programmer will make the relationship using the *inheritFrom()* function of the *Util.Inheritance* interface.

```
Son s = new Son();
Father f = new Father();

s.inheritsFrom(f);
```

This would insert an entry into the XML repository as follows:

```
<Util>
<Inheritance>
<parent objectid='fatherobjectid'>
<child objected='childobjectid'/>
</parent>
</Inheritance>
</Util>
```

Now, whenever there is a need to access parent's methods or attributes, it can be done with something similar to the follows:

```
s.getParent().F_method1();
```

5. CONCLUSION

The discovered pattern is more efficient and effective than the CHNFP because it is keeping the relationships in a XML based logical file which is widely supported by technologies. The use of objects is fast and the size of the OODBMS is also reduced because only one instance of the parent's object is stored and is queries only when needed and if invoked explicitly by the programmer.
The same XML structure can be used for future work on enhancements of this pattern. For example, the same XML hierarchy can be used to define constrains (or primary key) on the data (object's attribute), a feature currently missing in OODBMS. An XML attribute can be added to the XML structure to mention a primary key and with appropriate methods of another class e.g. *Util.Constraints* can be designed to enable such a feature.

```
<Util>
<Inheritance>
<parent objectid='xxx'>
<child objected='xxxx' primary_key='attr1'/>
</parent>
</Inheritance>
</Util>
```

Once a feature of defining constraints is enabled for OODBMS as discussed above, further work to define indexes on attributes can be undertaken and doors for faster querying of objects can be opened.

6. REFERENCES

[1] A Pattern for the Effective Use of Object Oriented Databases by Ubaid, M.; Atique, N.; Begum, S.; [IEEE 2009, http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5267187]

[2] Instantiating and detecting design patterns: putting bits and pieces together Albin-Amiot, H.; Cointe, P.; Gueheneuc, Y.-G.; Jussien, N. Automated Software Engineering, 2001. (ASE 2001). Proceedings. 16th Annual International Conference onVolume , Issue , 26-29 Nov. 2001 Page(s):166-173—Digital Object Identifier

[3] A Boolean Algebra Approach for Class Hierarchy Normalization Advanced Database Research and Development Series; Vol. 6, Proceedings of the Fifth International Conference on Database Systems for Advanced Applications (DASFAA) Pages: 203 – 312, Year of Publication: 1997, ISBN:981-02-3107-5

[4] http://en.wikipedia.org/wiki/Object-relational_mapping Page visiting date: 15 February 2009 at 5:10 PM (PST).

[5] Pro LINQ Object Relational Mapping with C# 2008- Copyright © 2008 by Vijay P. Mehta; ISBN-13 (electronic): 978-1-4202-0597-5 —Email: info@apress.com, Available: http://www.apress.com.

[6] http://en.wikipedia.org/wiki/ADO.NET_Entity_Framework Page visiting date: 16 February 2009 at 1:35 AM (PST).

[7] A Brief History of the Object-Oriented Approach- Luiz Fernando Capretz University of Western Ontario- Department of Electrical & Computer Engineering - London, ON, CANADA, N6G 1H1 Available: lcapretz@acm.org

[8] Three example references from 1985 that use the term: T. Atwood, "An Object-Oriented DBMS for Design Support Applications," Proceedings of the IEEE COMPINT 85, pp. 299-307, September 1985; N. Derrett, W. Kent, and P. Lyngbaek, "Some Aspects of Operations in an Object-Oriented Database," Database Engineering, vol. 8, no. 4, IEEE Computer Society, December 1985; D. Maier, A. Otis, and A. Purdy, "Object-Oriented Database Development at Servio Logic," Database Engineering, vol. 18, no.4, December 1985.

[9] Kim, Won. Introduction to Object-Oriented Databases. The MIT Press, 1990. ISBN 0-262-11124-1

[10] Bancilhon, Francois; Delobel,Claude; and Kanellakis, Paris. Building an Object-Oriented Database System: The Story of O2. Morgan Kaufmann Publishers, 1992. ISBN 1-55860-169-4.

[11] Radding, Alan. (1995). So what the Hell is ODBMS? Computerworld, 29(45), 121.

[12] Burleson, Donald. (1994). OODBMSs gaining MIS ground but RDBMSs still own the road. Software Magazine, 14(11), 63
[13] A new data abstraction layer required for OODBMS—Eun-Sun Cho Sang-Yong Han Hyoung-Joo Kim—Dept. of Comput. Sci., Seoul Nat. Uni.; Publication Date: 25-27 Aug 1997 ;On page(s): 144-148—ISBN: 0-8186-8114-4 ;Digital Object Identifier: 10.1109/IDEAS.1997.625670—Current Version Published: 2002-08-06

[14] Object-Oriented Database System Concepts and Architectures Elisa Bertino University of Genoa Lorenzo Martino—ISBN 0-201-62439-7 Pages – 14 to 16
[15]Patterns of Enterprise Application Architecture By Martin Fowler, David Rice, Matthew Foemmel, Edward Hieatt, Robert Mee,Randy Stafford ; Publisher: Addison Wesley Pub Date: November 05, 2002 ; ISBN: 0-321-12742-0 ; Pages: 560
[16]The Database Behind the Brains db4o | The Open Source Object Database | Java and .NET By Rick Grehan | 2nd, Updated Edition | March 2006

[17] Cost-driven vertical class partitioning for methods in object oriented databases Chi-Wai Fung1, Kamalakar Karlapalem2, Qing Li3 Hong Kong, China; Edited by M.H. Scholl. Received: March 29, 1999 / Accepted: March 11, 2002 Published online: April 3, 2003 –_c Springer-Verlag 2002 Available E-mail: itqli@cityu.edu.hk

[18] The Definitive Guide to db4o Copyright © 2006 by Jim Paterson, Stefan Edlich, Henrik Hörning, Reidar Hörning ISBN-13: 978-1-59059-656-2 , ISBN-10: 1-59059-656-0 Email: info@apress.com, Available: http://www.apress.com.
[19] Design Patterns as Language Constructs; Jan Bosch—University of Karlskrona/Ronneby , Department of Computer Science and Business Administration ,S-372 25 Ronneby, Sweden—E-mail: Jan.Bosch@ide.hkr.se Available:http://www.pt.hk-r.se/~bosch

[20]http://en.wikipedia.org/wiki/Table_of_mathematical_symbols—Page visiting date: 16 March 2009.