Bilgi Yönetimi Dergisi

Cilt: 6 Sayı: 2 Yıl: 2023

https://dergipark.org.tr/tr/pub/by

Peer-Reviewed Articles

Research Article

Article Info

Date submitted: 23.04.2023

Date accepted: 11.07.2023

Date published: 31.12.2023

Makale Bilgisi

Gönderildiği tarih: 23.04.2023

Kabul tarihi: 11.07.2023

Yayınlanma tarihi: 31.12.2023

Keywords

Object-Oriented Model, Information Management, Information System

Anahtar Sözcükler

Nesne Yönelimli Model, Bilgi Yönetimi, Bilgi Sistemi

DOI Numarası

10.33721/by.1270095

ORCID

0000-0003-0844-8160

CC BY

Managing Business Data with An Object-Oriented Approach

Nesne Yönelimli Bir Yaklaşımla İş Verilerinin Yönetimi

Cem Ufuk BAYTAR

İstanbul Topkapı Üniversitesi, İİSBF, Yönetim Bilişim Sistemleri Bölümü Öğretim Üyesi, *ufukbaytar@topkapi.edu.tr*

Abstract

Nowadays, managing data is so vital for companies in every sector to compete with competitors. Databases are the critical part of information systems to process raw data. Some of them are open source code and some of them are commercial ones. In this study, the main question is that how business data is managed based on the concept of persistence without a need to connect to a database management system to make a contribution for the problem of impedance mismatch. To find the answer of this question, a persistent object-oriented model has been proposed to establish an infrastructure for especially small companies to manage business data. When designing this model, the source of inspiration has been the concepts of persistence and delegation. Delegation contributes to diminish the effects of code scattering and code tangling problems and to increase modularity. It also plays an important role in the model in order to build an interface between users and the system. Serialization methodology has been applied to save data represented by persistent objects. C++ programming language was used for implementation of the model. The reliability of the proposed model has been proved based on Chidamber and Kemerer's metric set to measure object-oriented programming. Consequently, the first version of the model has been implemented without needing any database management system. It has also provided valuable functionalities, i.e., saving or loading data, listing data, describing data, inserting data based on object-oriented concepts. In the future, the researchers of the same field can make contributions for developing this model by implementing new features to make it more powerful technically.

Öz

Günümüzde, verileri yönetmek her sektördeki şirketlerin rakipleriyle rekabet edebilmesi için çok önemlidir. Veritabanları ham verilerin işlenmesi açısından bilgi sistemlerinin kritik bir parçasıdır. Bu veritabanlarının bazıları açık kaynak kodlu, bazıları ise ticari kodlardır. Bu çalışmada temel soru, iş verisinin bir veritabanı yönetim sistemine bağlantı gereksinimi olmadan nesnelerin kalıcılığı kavramına dayandırılarak empedans uyumsuzluğu problemine katkı sağlamak için nasıl yönetilebileceğidir. Bu sorunun cevabını bulabilmek amacıyla, özellikle küçük şirketlerin iş verilerini yönetecekleri bir altyapı oluşturmak üzere kalıcılaştırılmış nesne yönelimli bir model önerilmiştir. Bu modeli tasarlarken ilham kaynağı delegasyon ve nesnelerin kalıcılığı kavramları olmuştur. Delegasyon, nesne yönelimli kodlamada kod saçılması ve kod karıştırma sorunlarının etkilerinin azaltılmasına ve modülerliğin artmasına katkıda bulunur. Ayrıca, delegasyon kullanıcılar ve sistem arasında bir ara yüz oluşturulmasında model için önemli bir rol oynamaktadır. Serileştirme metodolojisi, kalıcı nesneler tarafından temsil edilen verileri kaydetmek için uygulanmıştır. Modelin uygulamasında C++ programlama dili kullanılmıştır. Önerilen modelin güvenilirliği, Chidamber ve Kemerer'in nesne yönelimli programlamayı ölçmek için belirlediği metrikler kümesine dayanarak kanıtlanmıştır. Sonuç olarak, modelin ilk hali herhangi bir veritabanı yönetim sistemine ihtiyaç duymadan uygulanmıştır.

**The research and publication process of this article was carried out in accordance with "Research and Publication Ethics".*

Model, verileri saklama veya yükleme, verileri listeleme, verileri tanımlama, veri ekleme gibi nesne yönelimli kavramlara dayalı değerli işlevler de sağlamıştır. Gelecekte, bu alandaki araştırmacılar modeli teknik olarak daha güçlü hale getirecek yeni özellikleri uygulayarak modelin gelişmesine katkı sağlayabilirler.

1. Introduction

Relational databases need the relational data model to save the data. This model is suitable for storing structured data. MySQL, PostgreSQL, and Oracle are some examples of a relational databases. NoSQL databases have not referential integrity constraint among data objects. They can operate in a distributed architecture (Lajam and Mohammed, 2022). They are non-relational databases because their data models, i.e., key-value, document, column family, and graph data model, are different from the relational data model (Moniruzzaman and Hossain, 2013). An object-oriented database (OODB) is a structure that is able to manage data represented by objects. In addition, it is based on an object-oriented model (OOM). In literature, there are studies about implementing of OOMs or OODBs. Some of the related works are summarized in Table 1.

Table 1

Related Works

Work	Subject / Related to	Concepts/Tools
Bergesio et al., 2017	OOM for orchestrating smart devices	Inheritance, polymorphism
Zuo et al., 2019	Developing open source data center package	Modelica
Ma et al., 2015	How to store OWL ontologies in object-oriented databases	Object-oriented database
Liu et al., 2015	OOM to navigate blind people in outdoor space	Objective orientation idea
Coruhlu and Yıldız, 2017	Modelling of object-oriented land division	Geographical database
Schubert et al, 2022	Using a graph database for integration of business objects from heterogenous Business Information Systems	Graph database
Truica et al., 2021	Document-Oriented Database Management Systems	XML, JSON
Candel et al., 2022	Metamodel for NoSQL and relational databases	U-Schema, NoSQL

There is an important problem between relational data model and object-oriented applications. It is called impedance mismatch or object-relational impedance mismatch. It occurs when trying to access a relational database from an object-oriented application since there is a gap between the object-oriented model of an application and the relational model in a database management system (Lajam and Mohammed, 2022). A persistent object store (POS) is useful to prevent the problem of impedance mismatch (Cortes et al., 2019).

In this study, a persistent object-oriented model was introduced to manage business data in a consistent and efficient way. For this purpose; i) the proposed OOM has been presented based on persistence and delegation concepts and serialization methodology, ii) the model has been implemented with C++ programming language and iii) the model does not require connecting to a database management system iv) the model contributes to the problem of impedance mismatch.

The contributions of this study to the literature are i) focusing on concepts of data persistence and persistent object store, ii) how to use the concept delegation to create the effect of inheritance, iii) how to implement an interface (like interface concept in Java) by using the delegation concept instead of

abstract functions, iv) pointing out how to reduce effects of code scattering and code tangling problems in OOP and v) applying OOP metrics (CK Metrics) to verify that the OOM is reliable.

The 2nd part of the study is about the concepts that is necessary in the development of the model, implementation of the proposed model has been explained in Chapter 3. Chapter 4 includes findings and the validation of the model. Finally, results obtained from the study are given in Chapter 5.

2. Summary of Literature

This part of the study includes information about data persistence, persistent object store, object-oriented programming concepts, for example, composition, delegation, vector and serialization and studies about persistent data models.

A database design includes conceptual, logical, and physical models to make sure that the structure to store data will be built in a suitable way and will meet the requirements of a database system. A conceptual design converts necessary requirements into a conceptual database schema. The logical design represents the data model of a database, i.e., data types, the format of storing data (tables, documents, nodes etc.). Physical design plays an vital role to match the data with the storage environment in an efficient way. In addition, it includes some peculiarities, i.e., specific data types, storage forms, partitioning, and clustering capabilities etc.(Zdepski et al., 2018). The key differences between relational database and object-oriented databases are summarized in Table 2 (databasetown, n.d.).

Table 2

Key Differences Between Relational Database and Object-Oriented Databases

Criteria	Relational Database	Object Oriented Database
Definition	Data is stored in tables which consist of rows and columns.	Data is stored in objects. Objects contain data.
Amount of data	It can handle large amounts of data.	It can handle larger and complex data.
Type of data	Relational database has single type of data.	It can handle different types of data.
How data is stored	Data is stored in the form of tables (having rows and columns).	Data is stored in the form of objects.
Data Manipulation Language	DML is as powerful as relational algebra. Such as SQL, QUEL and QBE.	DML is incorporated into object-oriented programming languages, such as C++, C#.
Learning	Learning relational database is a bit complex.	Object oriented databases are easier to learn as compared to relational database.
Structure	It does not provide a persistent storage structure because all relations are implemented as separate files.	It provides persistent storage for objects (having complex structure) as it uses indexing technique to find the pages that store the object.
Constraints	Relational model has key constraints, domain constraints, referential integrity and entity integrity constraints.	To check the integrity constraints is a basic problem in object-oriented database.
Cost	The maintenance cost of relational database may be lower than the cost of expertise required development and integration of object oriented database.	In some cases hardware and software cost of object oriented databases is lower cost than relational databases.

Data persistence refers to the ability of data to remain accessible and retrievable even after the completion of a software application or in the event of runtime crashes. Essentially, it involves securely storing and reliably loading data when required. In recent times, in-memory data persistence has emerged as the cutting-edge approach. The concept of in-memory data persistence revolves around storing data in either memory or external databases/storages to ensure its persistence and availability (Chen et al., 2019).

Persistent object store is a data storage system based on objects. Such a system saves and loads the data that is persistent in the form of objects (Brown and Morrison, 1992). POSs are useful to prevent the problem of impedance mismatch. It is a problem that occurs when an object oriented application tries to retrieve the relevant data in other types of databases, such as relational database management systems (RDBMSs). There are three types of Users do not need to connect a database and to create a query in order to get persistent data by using POSs (Atkinson et al., 1983; Chen et al., 2014).

There are some studies that have discussed about implementing persistent data model. Kozynchenko (2006) developed a persistent object-oriented model by using C++ as shown in Figure 1. He suggested an approach to build object-oriented models that provide persistency that is necessary for database systems. The model is based on the parent and child classes, objects linked by pointers, inheritance hierarchy and files structure provided by C++ programming language. Also, Chen et al. (2019) established a model for in-memory data persistence by using javascript and Intel's Persistent Memory Development Kit (PMDK), which is a C++ development kit to make the implementations of persistent memory, as shown in Figure 2 and Figure 3. Cortes et al. (2019) presented CAPre (Code-Analysis based Prefetching for Persistent Object Stores) that is a novel prefetching system for Persistent Object Stores based on static code analysis of object-oriented applications as seen in Figure 4.

Figure 1

Generalized Scheme of the C++ Persistent Object-Oriented Model

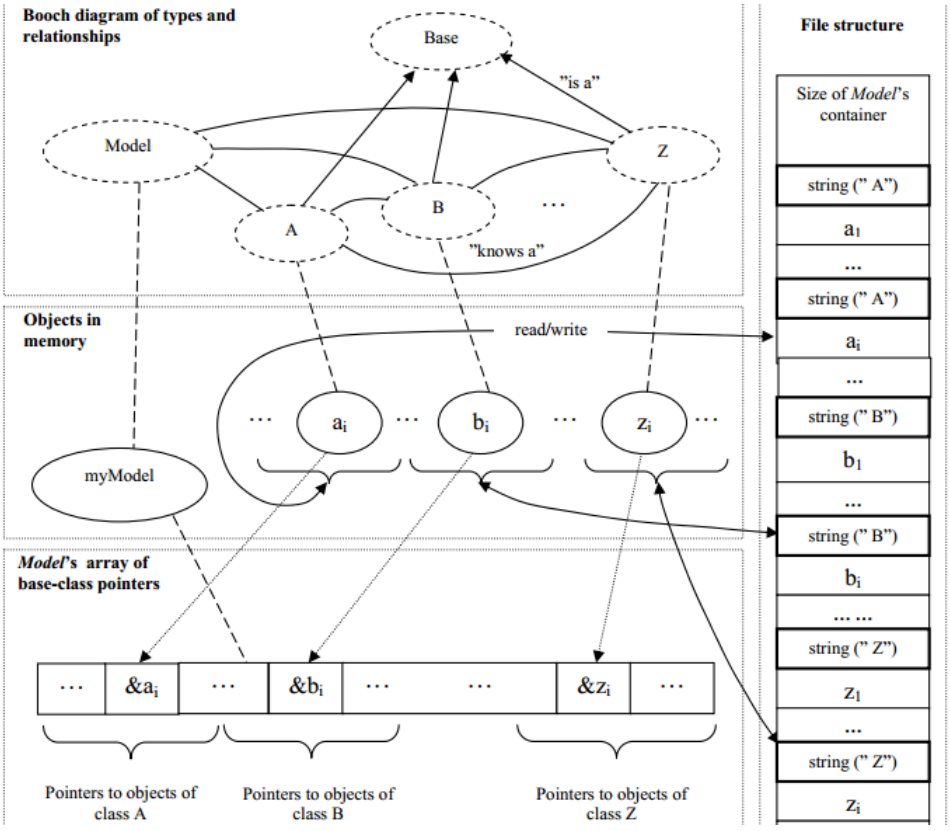


Figure 2

An Overview of Jdap

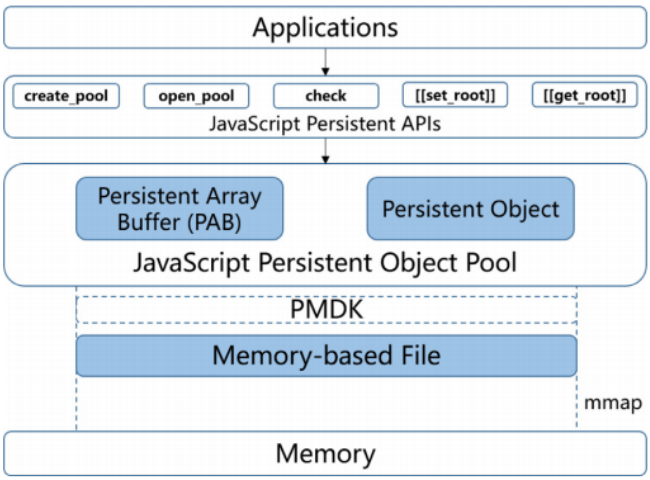


Figure 3

Javascript Persistent Object Pool

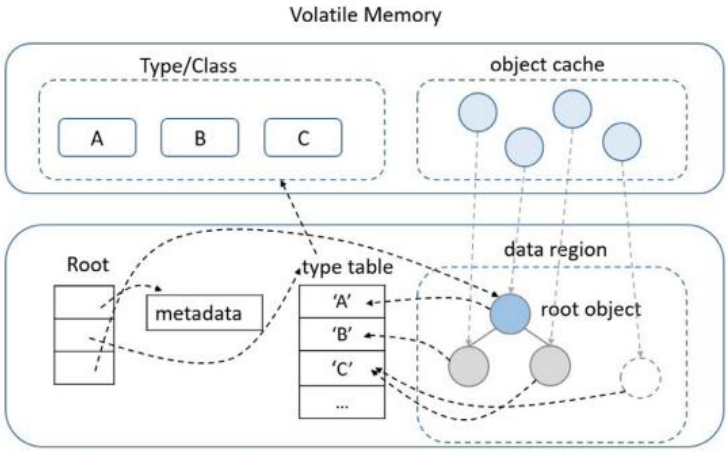
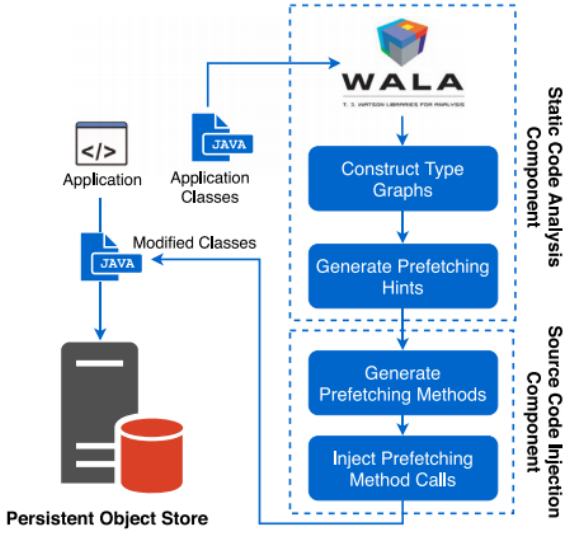


Figure 4

Overview of the Proposed Prefetching System



Composition refers to the relationship between objects where one object (referred to as the whole or parent) possesses another object (known as the part or child). In simpler terms, the parent object retains ownership of the child object, and the child object remains in existence as long as the parent object exists.

Figure 5

Composition Diagram in UML

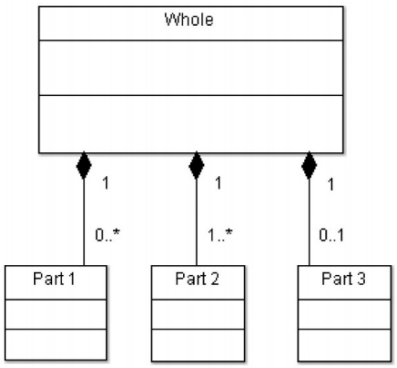


Figure 6

The Syntax of Composition

```
class A
{
  // body of a class
};

class B
{
  A objA;
  public:
  B(arg-list) : objA(arg-list1);
};
```

Figure 5 depicts a compositon diagram as an example (Lorenzo, 2020). This diagram explains that whole object has different parts. The syntax of compositon is given in Figure 6 (Geeksforgeeks, 2022).

Figure 7

Implementation of Delegation

```
class First
{
public:
        void print() { cout << "The Delegate"; }
};
class Second
{
First ob;
public:
        void print() { ob.print(); }
};
int main()
{
        Second ob1;
        ob1.print();
        return 0;
}
```

Delegation is a feature of C++ programming language that can be used instead of inheritance. It is also useful technique for object-oriented programmers. Delegation has the effect of interitance. In C++, it can change the behaviour of an object dynamically by changing an object's delegatee. A sample code of delegation is shown in Figure 7 (Johnson and Zweig, 1991; Geeksforgee, 2022).

C++ Standard Template Library (STL) includes different types of containers, for example, list, vector, map, etc. A vector is an array that has a dynamic chracteristic. It contains values (elements) of the same type. When a person adds an element to a vector, it can adjust its own size automatically. Every element follows other element in order (Geeksforgee, 2022; Pataki et al., 2011).

A data structure or an instance of a class can be saved in the memory of a computer system if they are converted to an appropriate format. Serialization makes such a conversion possible. Re-creation of the

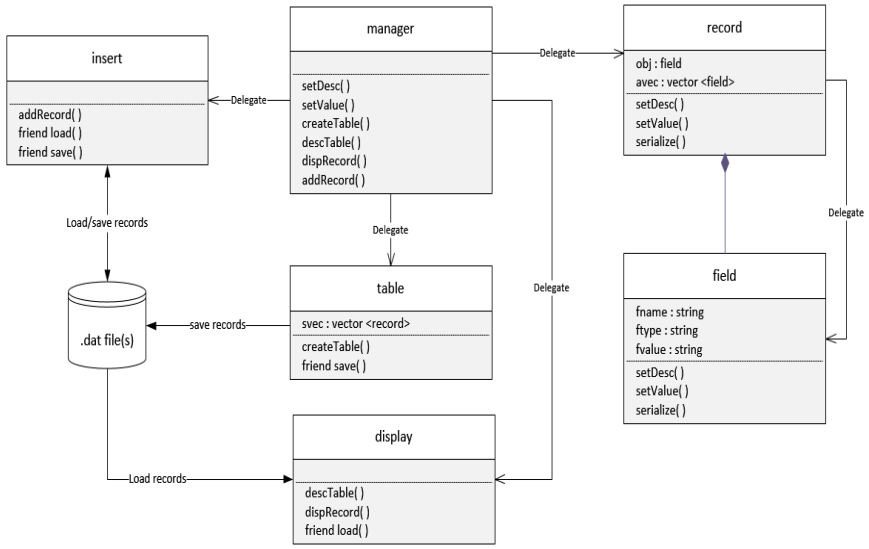
same object stored in the memory is called deserialization. Binary serialization stores an object in sequences of bits (Tauro et al., 2012). Object serialization causes storage amount of data to reduce (Carrera et al., 2018).

3. Material and Method

This chapter explains what the model is and gives information about implementation of object-oriented model. In this study, a persistent object-oriented model has been proposed in Figure 8. The main purpose of this model is to introduce an approach in order to build the base of an object-oriented tool that will help users to manage business data for especially small companies in every industry without needing any database management system by contributing to the problem of impedance mismatch.

Figure 8

The Proposed Persistent OOM



The model has class manager that has an interface role between a user and a system. The code sample is given in Figure 9. The manager class delegates tasks to other relevant classes. Delegation contributes to increase the level of modularity in the OOM and to decrease the effects of code scattering and code tangling issues in OOP. The record class has “has-a” relationship (composition) with the class field in order to define the structure of a record as an object. The class record delegates tasks (setDesc() and setValue()) to the class field. Vectors have been used as a container to keep the field objects together to build a record or to keep the record objects together to create a table.

Figure 9

Code Sample of Class Manager

```
class manager
{
public:
record setDesc(record  o)
{
        o.setDesc();
        return o;
}

record setValue(record  o)
{
        o.setValue();
        return o;
}

//some codes
};
```

In this study, C++ programming language has been used. C++ is one of languages that consumes less memory, expends least energy and is faster than other 27 programming languages in the research. In addition, C++ is one of pioneers that represents object-oriented paradigm (Pereira et al., 2017). C++ needs an external library to make a serialization/deserialization process. Boost library has been used in this study as depicted in Figure 10.

Figure 10

Boost Library

```
class record
{
public:
    vector <field> avec;
    // some codes

private:
    friend class boost::serialization::access;

    template <class Archive>
    void serialize(Archive& ar, const unsigned version)
    {
        ar& avec;
    }

};
```

The serialize function is designated as private within the respective classes (record and field). For storing an object in an archive (save function), the << or & operator is utilized, while the >> or & operator is employed to retrieve an object from an archive (load function), as illustrated in Figure 11 and Figure 12 When invoking one of these operators, the serialization function is called by the system. Instances of class records are stored in the relevant .dat file. Every .dat file has the role of a table that includes record objects as shown in Figure 8. Save () and Load () are friend functions. It means that they are not member functions of other classes. In other words, they are common functions used by other objects so that such a behaviour in the model prevents issues of scattering and tangling in object oriented coding.

Figure 11

Save Function

```
void save(string name, vector<record> svec, record o)
{

//some codes

{

    boost::archive::text_oarchive archive(outfile);
    archive << svec;

}

}
```

Figure 12

Load Function

```
vector<record> load(string name)
{

//some codes

{
    boost::archive::text_iarchive archive(infile);
    archive >> svec;

}

return svec;
}
```

4. Findings

This chapter includes the findings after implementing the model and provides the validation of the model. The proposed model has been realized some of SQL commands. Table 3 shows comparison of class functions in the OOM and MySQL equivalents.

Table 3

Comparison of Class Functions And MySQL Equivalents

Class Name	Functions In Classes	MySQL equivalent
record table	setDesc ( ) + createTable ( )	create table table_name (column1 datatype, column2 datatype, column3 datatype);
display	descTable( )	describe table_name
record insert	setValue ( ) + addRecord ( )	insert into table_name (column1, column2, column3) values (value1, value2, value3);
display	dispRecord ( )	select * from table_name

As stated in Figure 13 and Figure 14, setDesc() member function of the class record defines every necessary field belonged to a record, after that, createTable function of the class table adds this record object to the relevant dat file that behaves like a table. As a result, co-operation of these 2 member functions realizes the same functionality as “Create Table” command of MySQL.

Figure 13

setdesc() Function of Class Record

```
class record
{
// some codes

void setDesc()
{

    int n;
    cout << "enter the number of field :";
    cin >> n;
    for (int s = 0; s < n; s++)
    {
        obj.setDesc();
        avec.push_back(obj);
    }
}

 // some codes
}
```

Figure 14

createtable() Function of Class Table

```
class table
{

//some codes
void createTable(record  o)
{
    string name;
    cout << endl << "Enter table name:";
    cin >> name;

    save(name, svec, o);

}
};
```

In Figure 15, the object mng, which plays the role of interface, calls setDesc() and createTable() functions to describe and to create a table.

Figure 15

Describing and Creating a Table in the Main Function

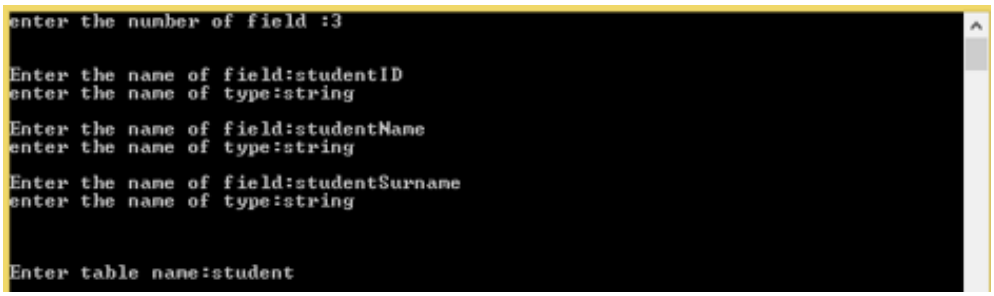
```
295  int main()
296  {
297
298      record o;
299      vector<record > svec;
300      manager mng;
301      table t;
302      display d;
303      insert ins;
304
305      /* Describing and Creating a table
306      ********** It is like "Create Table" command in SQL**********************
307      */
308      o = mng.setDesc(o);
309      mng.createTable(o, t);
310
311      return 0;
312
313  }
314
```

In the runtime, users can decide a function call by using the class manager. In addition, users can determine the number of fields that will be defined. Vectors can adjust their size automatically. Save

function also invokes the serialize function (dynamic binding) in the relevant class. All of such features make the structure of the model dynamic as shown in Figure 16.

Figure 16

Implementation of Describing and Creating a Table



Reliability is an important measurement to show the validity and quality of a software system. In other words, it is probability of that the program will perform necessary functions in a correct way (Johny, 2013).

Chidamber and Kemerer's metric set (CK Metrics) is well-known for measuring OOP. It has been used for the model proposed because it is suitable for object-oriented coding whose process is finished (Katic et al., 2013). CK Metrics set evaluates object-oriented design instead of software implementation (Ponnala and Reddy, 2019). It includes six metrics as shown in Table 4 (Basili et al., 1995; Bakar et al., 2014; Chidamber and Kemerer, 1994).

Table 4

Definitions of CK Metrics

Metrics	Definition
Weighted Methods per Class (WMC)	The number of methods defined in each class. If a class has more member functions, it will be more complex. This causes more errors to be happened
Depth of Inheritance Tree of a class (DIT)	The number of ancestors of a class. Well-designed OO systems have not a large inheritance tree.
Number of Children of a Class (NOC)	The number of direct descendants for each class. If a class has more children, it will be difficult to manage it
Coupling Between Object classes (CBO)	The number of classes which their members are used by a given class. Weakly coupling means less probability of occurring faults.
Response For a Class (RFC)	The number of functions directly called by member functions of a class. Larger RFC means higher complex, more fault-prone classes.
Lack of Cohesion on Methods (LCOM)	(The number of function pairs not using common instance variables) – (The number of function pairs using common instance variables). A class with low cohesion (high LCOM) among its methods suggests an inappropriate design.

Threshold values for CK Metrics, which are used to predict software reliability, are established by researchers and presented in Table 5.

Table 5

Threshold Values of CK Metrics

Related Works	WMC	DIT	NOC	CBO	RFC	LCOM
Calp & Arıcı, 2011	Low	Low	Low	Low	Low	Low
Goel & Bhatia, 2012	2	2	2	1	5	1
Zhou & Leung, 2006	0-15	0-6	0-6	0-8	0-35	0-1
Mago & Kaur, 2012	0-11	0-4	1-3	0-3	0-12	0
Edith & Chandra, 2010	0-15	0-6	0-6	0-8	0-35	0-1

Reliability is inversely proportional to CK metrics as follows:

- Reliability α 1/WMC
- Reliability α 1/RFC
- Reliability α 1/DIT
- Reliability α 1/LCOM
- Reliability α 1/CBO

Reliability of software based on CK Metrics is calculated according to rules as follows (Johny, 2013; Misra and Roy, 2015; Yılmaz and Tarhan, 2019):

- Weighted values for CK Metrics:
  - If lower threshold limit ≤ value of metric ≤ mean of threshold range, weighted value is 1.
  - If mean of threshold range ≤ value of metric ≤ upper threshold limit, weighted value is 2.
  - If value of metric is outside of threshold range, weighted value is 7.
- In terms of NOC metric:
  - (log(upper threshold limit of NOC))² is used for R-max (Maximum reliability value)
  - (log(lower threshold limit of NOC))² is used for R-min (Minimum reliability value)
  - If NOC value is outside of threshold range, it will be omitted.
- Reliability value (R) is calculated as follows:

$$R = k*(1/(wt(WMC)+wt(DIT)+wt(RFC)+wt(LOCM)+wt(CBO)) + (\log(wt(NOC)))^2 \quad (1)$$

where wt(WMC) is weighted value of WMC etc., k is 1.

$$R\text{-}max = k*(1/(1+1+1+1+1)) + (\log(\text{upper threshold of NOC}))^2 \quad (2)$$

where weighted value of every metric is maximum.

$$R\text{-}min = k*(1/(2+2+2+2+2)) + (\log(\text{lower threshold of NOC}))^2 \quad (3)$$

where weighted value of every metric is minimum.

R should be between R-max and R-min. In other words, R-min < R < R-max.

None of OO metrics can not individually explain the quality of object-oriented design. To evaluate the OOM, threshold limits (Mago and Kaur, 2012) in Table 4 have been preferred because the model on this work has provided the integrated evaluation of CK Metrics based on fuzzy logic.

The average values of metrics (Calp and Arıcı, 2011) belonged to classes in the proposed model are shown in Table 6.

Table 6

Values of CK Metrics in the Model Proposed

Class Name	WMC	DIT	NOC	CBO	RFC	LCOM
manager	6	0	0	4	6	0
insert	3	0	0	1	0	0
display	3	0	0	1	0	0
table	2	0	0	1	0	0
record	3	0	0	1	2	0
field	3	0	0	0	0	0
Average of metrics	3,33	0	0	1,33	1,33	0

In Table 7, it is stated that R-Max is 0,428, R-Min is 0,1. In addition, Reliability value of the model is 0,2 showing that the model is reliable because it is between R-Min and R-Max.

Table 7

Reliability Values of the Model Proposed

	WMC	DIT	NOC	CBO	RFC	LCOM
Average of metrics	3,33	0	0	1,33	1,33	0
R-Max	$1*(1/(1+1+1+1+1)) + (\log(3))^2 = 0,428$					
R-Min	$1*(1/(2+2+2+2+2)) + (\log(1))^2 = 0,1$					
R	$1*(1/(1+1+1+1+1)) = 0,2$					
	NOC is omitted					

Another point is that the metrics of the model are fit to metric values shown in Table 4. This also proves that the proposed model is reliable.

5. Conclusion

In this study, an OOM, based on persistence and delegation concepts, has been implemented. It has shown that users can manage business data with the persistent objects without a database management system to prevent the impedance mismatch problem. To manage business data, this implementation has also provided the realization of some SQL commands, i.e., select, insert, create table, describe table that is available in MySQL database management system. In addition, data represented by persistent objects has been saved by using serialization methodology. The implementation points out that the OOM has a dynamic structure. Interface has also been established by using the delegation concept in order to increase the modularity level. The reliability of the model has been proved based on CK Metrics. On the other hand, this model will need new features that make it technically more powerful in the future. Data conversion, data integrity checks, updating data, searching data can be given as examples of new features for researchers in the relevant field.

Compliance with Ethical Standards

Conflict of Interest: The author declare that there is no conflict of interest.

Ethics Committee Permission: Ethics comittee approval is not required for this study.

Authors Contribution Rate Statement: The author declares that he has contributed fully to the article.

Financial Support: No

References

Atkinson, M.P., Bailey, P.J., Chisholm, K.J., Cockshott, P.W. & Morrison, R. (1983). An approach to persistent programming. *The Computer Journal*, 26(4), 360-365, doi: http://dx.doi.org/10.1093/comjnl/26.4.360.

Bakar, A.D., Sultan, A., Zulzalil H. & Din, J. (2014). Predicting Maintainability of Object oriented Software Using Metric Threshold. *Information Technology Journal*, 13(8), 1540-1547, 2014.

Basili, V.R., Briand, L. & Melo, W.L. (1995). A Validation of Object-Oriented Design Metrics as Quality Indicators. Technical Report, Dep. of Computer Science, Univ. of Maryland, College Park, MD, USA. https://www.cs.umd.edu/~basili/publications/technical/T102.pdf

Bergesio, L., Bernardos, A.M. & Casar, J.R. (2017). An Object-Oriented Model for Object Orchestration in Smart Environments. *Procedia Computer Science*, 109C, 440-447.

Brown, A.L. & Morrison R. (1992). A generic persistent object store. *Software Eng. Journal*, 7(2), 161-168, doi: http://dx.doi.org/10.1049/sej.1992.0017.

Calp, M.H. & Arıcı, N. (2011). Nesne Yönelimli Tasarım Metrikleri ve Kalite Özellikleriyle İlişkisi. *Politeknik Dergisi*, 14(1), 9-14.

Candel, C.J.F., Ruiz, D.S. & García-Molina, J.J. (2022). A unified metamodel for NoSQL and relational databases. Information Systems, 104, 101898, 1-26, doi: https://doi.org/10.1016/j.is.2021.101898

Carrera, D., Rosales, J.& Gustavo, A. (2018). Optimizing Binary Serialization with an Independent Data Definition Format. *International Journal of Computer Applications*, 180, 15-18.

Chen, T.H., Shang, W., Jiang, Z.M., Hassan, A.E., Nasser, M. & Flora, P. (2014). Detecting performance anti-patterns for applications developed using object-relational mapping. Paper presented at the 36th International Conference on Software Engineering, Hyderabad, 2014, pp. 1001-1012. http://dx.doi.org/10.1145/2568225.2568259.

Chen, Y., You, L., Xu, H., Zhang, Q., Li, T., Li, C. & Huang, L. (2019). JDap: Supporting in-memory data persistence in javascript using Intel's PMDK. *Journal of Systems Architecture*, 101(2019), 101662, 1-12. doi: https://doi.org/10.1016/j.sysarc.2019.101662

Chidamber, S.R. & Kemerer, C.F. (1994). A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engineering*, 20(6), 476-493.

Cortes, T., Queralt, A. & Touma, R. (2019). CAPre: Code-Analysis based Prefetching for Persistent Object Stores. *Future Generation Computer Systems*, 111(2020), 491-506, doi: https://doi.org/10.1016/j.future.2019.10.023

Coruhlu, Y.E. & Yıldız, O. (2017). Geographical database for object-oriented land division modelling in Turkey. *Land Use Policy*, 68, 212-221.

Databasetown (n.d.). Relational Database vs Object-Oriented Database (Key Differences). Retrieved from https://databasetown.com/relational-database-vs-object-oriented-database-key-differences/

Edith, L.P. & Chandra, E (2010). Class Break Point Determination Using CK Metrics Thresholds. *Global Journal of Computer Science and Technology*, 10(14), 83-87.

Geeksforgee (2022, Febraury 11). Composition. Retrieved from https://www.geeksforgeeks.org/object-composition-delegation-in-c-with-example

Geeksforgeeks (2022, January 18). Object Delegation in C++. Retrieved from https://www.geeksforgeeks.org/object-delegation-in-cpp

Geeksforgeeks (2022, May 13). Vector in C++. Retrieved from https://www.geeksforgeeks.org/vector-in-cpp-stl

Goel, B.M. & Bhatia, P.K. (2012). Analysis of Reusability of Object-Oriented System using CK Metrics. *ACM SIGSOFT Software Engineering Notes*, 38(4), 1-5.

Johnson, R. & Zweig, J.M. (1991). Delegation in C++. *Journal of Object-Oriented Programming*, 4, 31-34.

Johny, A.P. (2013). Predicting Reliability of Software Using Thresholds of CK Metrics. *Int. J. Advanced Networking and Applications*, 4(6), 1778-1785.

Katic, M., Boticki, I.. & Fertalj, K. (2013). Impact of Aspect Oriented Programming on the Quality of Novices' Programs: A Comparative Study. *Journal of Information and Organizational Sciences*, 37(1), 45-61.

Kozynchenko, A. (2006). Constructing persistent object-oriented models with standard C++. *Journal of Object Technology*, 5(1), 69-81.

Lajam, O. & Mohammed, S. (2022). Revisiting Polyglot Persistence: From Principles to Practice. *International Journal of Advanced Computer Science and Applications*, 13(5), 872-882.

Liu, D., Chena, M., Lin, H., Zhang, H. & Yue, S. (2015). An object-oriented data model built for blind navigation in outdoor space. *Applied Geography*, 60, 84-94.

Lorenzo, T. (2020). Object-oriented event-graph modeling formalism to simulate manufacturing systems in the Industry 4.0 era. *Simulation Modelling Practice and Theory*, 99, 1-33.

Ma, Z.M., Zhang F. & Li, W. (2015). Storing OWL ontologies in object-oriented databases. *Knowledge-Based Systems*, 76, 240-255.

Mago, J. & Kaur, P. (2012). Analysis of quality of the design of the object oriented software using fuzzy logic. International Journal of Computer Applications, 21–25.

Misra, S. & Roy, B. (2015). Assessment of Object Oriented Metrics for Software Reliability. *International Journal of Engineering Research & Technology*, 4(1), 432-435.

Moniruzzaman, A. & Hossain, S. A. (2013). Nosql database: New era of databases for big data analytics-classification, characteristics and comparison. *International Journal of Database Theory and Application*, 6(4), 1-14.

Pataki, N., Szűgyi, Z. & Dévai, G. (2011). Measuring the Overhead of C++ Standard Template Library Safe Variants. *Electronic Notes in Theoretical Computer Science – ENTCS*, 264(5), 71-83.

Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernandes, J.P. & Saraiva, J. (2017). Energy Efficiency across Programming Languages: How Do Energy, Time, and Memory Relate? *ACM SIGPLAN International Conference on Software Language Engineering*, Vancouver, 2017, pp. 256-267.

Ponnala, R. & Reddy, C.R.K. (2019). Object Oriented Dynamic Metrics in Software Development: A Literature Review. *International Journal of Applied Engineering Research*, 14(22), 4161-4172.

Schubert, P., Blankenberg, C. & Gebel-Sauer, B. (2022). Using a graph database for the ontology-based information integration of business objects from heterogenous Business Information Systems. *Procedia Computer Science*, 196, 314–323.

Tauro, C. N., Mishra, G.S. & Bhagwat, A. (2012). A Study of Techniques of Implementing Binary Serialization in C++, Java and .NET. *International Journal of Computer Applications*, 45, 25-29.

Truica, C.O., Apostol, E.S., Darmont, J. & Pedersen, T.B. (2021). The Forgotten Document-Oriented Database Management Systems: An Overview and Benchmark of Native XML DODBMSes in Comparison with JSON DODBMSes. *Big Data Research*, 25, 1-14.

Yılmaz, N. & Tarhan, A. (2019). A two-dimensional method for evaluating maintainability and reliability of open source software. *Journal of the Faculty of Engineering and Architecture of Gazi University*, 3(4), 1807-1829.

Zdepski, C., Bini, T. & Matos, S. (2018). An Approach for Modeling Polyglot Persistence. 20th International Conference on Enterprise Information Systems, Maderia, 2018, pp. 120-126.

Zhou, Y. & Leung, H. (2006). Empirical analysis of object-oriented design metrics for predicting high and low severity faults. *IEEE Trans. Softw. Eng*., 32(10), 771-789.

Zuo, W., Fu, Y., Wetter, M., VanGilder, J.W. & Yang, P. (2019). Equation-based object-oriented modeling and simulation of data center cooling systems. *Energy&Buildings*, 198, 503-519.