Comparative Study on Object Persistence Methods

Clarence J M Tauro
Christ University
Bangalore, India

N Ganesan
Director (MCA)
RICM, Bangalore

Ritesh Kumar Sahai
Christ University
Bangalore, India

Sandhya Rani A
Christ University
Bangalore, India

ABSTRACT

In last few decades there was rapid shift observed while writing software solutions. Object oriented model is now globally adapted and preferred by most of the application developers. Object persistence plays a key role in designing data model, business objects working with other business objects. Object persistence could be very efficient if designed correctly. A typical design leads extra overheads in terms of cost, resource utilization, and time. Considering the importance of object persistence, it is very essential to concentrate more on this area. Gateway-based, Object-relational database and Object-oriented Database are the three major groups of solutions available to implement object persistence.

In this paper, we discuss about the features available in Object Persistence methodologies, how and where we should use them efficiently based on the application requirements. Our discussion continues further on positive and negative sides of object persistence methodologies by considering limitations and different application requirements.

General Terms

Object Oriented Programming, Object Persistent, Persistent Programming Language.

Keywords

Object persistence; gateway based object persistence; object-oriented database; object-relational database, data model, data access, data sharing.

1. INTRODUCTION

When an object is being created by an application, the scope of object is limited to the application life cycle. With the end of application, object life line also ends. The reason is, object is being stored temporarily in main memory. To keep object alive, application need to store the object in persistence storage.

Object persistence refers to the concept of saving the state of an object so that it can be restored and used at a later time. An object that does not persist basically dies when it goes out of scope [2]. Maintaining the state of an object is called persistence. There are many ways to implement persistence in applications. A simple example is to use text file as storage. All required information can be stored on file by using file write operation. This information can be retrieved when restore is required by performing file read operation. This solution is not efficient and might be good for small applications, where limited information saving is required and information is not changing frequently. Representing information on simple text file becomes very complicated, where information to be stored is very complex in nature. Lot of development efforts will be wasted to maintain this. However text files are very flexible, easy to implement, can be accessed by more than one program, but they are not object friendly [4]. Object oriented programs offer variety of relationship with other objects, for example inheritance and references with other objects. The challenge here is how we can represent and maintain the different kind of relationships on text file.

Another solution is to use relational database as storage, but due to limitations of databases, it is not a best solution. This can be a good choice when application need database related functionalities like, transaction with rollback, record locking and indexing. Databases are generally expensive; managing them is even more difficult. Object oriented instances are basically structured in hierarchical and relational database structured in tabular format. These are two different structures and a difference in two approaches is known as "impedance mismatch" problem [5].

Solution towards achieving persistence in object-oriented applications categories in three classes: the gateway-based method adds object-oriented programming access to persistent data stored using traditional non object-oriented data stores, the object-relational DBMS method enhanced the extremely popular relational data model by including object-oriented modeling features, and the object-oriented DBMS method adds persistence support to objects in an object-oriented programming language [3].

2. PERSISTENCE OF OBJECT

Object-oriented programming languages are based on objects. Object type, object identity, and object creation is defined by object oriented system. The objects being created by object oriented application is transient and ends when program terminates [8].

Persistence independence, also called transparency, requires that it is indistinguishable whether code is operating on persistent or transient data [10]. Transparency is achieved by combining reachability-based identification with an object-faulting mechanism. The notion of object fault [11] is similar to the notion of page fault in the context of demand-paging virtual memory.

There are four mechanisms namely persistence by class, persistence by creation, persistence by marking and persistence by reference to achieve object persistence.

2.1 Persistence by Class

In this mechanism, we declare class to be persistent. All objects of the class are then persistent objects. Simple, but not much flexible, it is often useful to have both transient and persistent objects in a single class [8]. In many OODB systems, declaring a class to be persistent is interpreted as ``persistable''; objects in the class potentially can be made persistent.

2.2 Persistence by Creation

This category for object persistent introduces new syntax to create persistent objects, by extending the syntax for creating transient objects.

2.3 Persistence by Marking

In this object persistent mechanism we mark an object persistent after it is created (and before the program terminates). All objects are created as transient objects, but, if an object is to persist beyond the execution of program, it must be marked explicitly as persistent before program terminates.

2.4 Persistence by Reference

In this approach we explicitly declared the persistent object (sometimes referred as root object also). Objects can be one or more based on the requirements. Objects who referred the root persistent object either directly or indirectly also becomes the persistent objects. It is easy to make the entire data structure persistent by just declaring the root of the structure as persistent, but it becomes expensive when we try to trace the chains in detection for a database system.

An object is basically defined by its state and behavior. The system state cannot be defined completely without considering the relationship between other objects. In object oriented environment reference pointers are used to express the relationship between objects. To keep the relations between persistent objects after a program termination alive the special pointer class PersistentPtr is introduced [12].

3. OBJECT PERSISTAECE APPROACHES

Object-oriented models are globally adopted and preferred by most application developers for writing advance applications. Majority of today’s applications have to deal with the persistent data, therefore object persistence is becoming a critical need and its use in applications is increasing day by day. So, it is essential to focus more on object persistence in order to make Object Oriented applications more efficient and useful [3].

Persistence deals with more than just the lifetime of data. In object-oriented databases, not only does the state of an object persist, but its class must also transcend any individual program, so that every program interprets this saved state in the same way [1]. This clearly makes it challenging to maintain the integrity of a database as it grows, particularly if we must change the class of an object.

Three classes, Gateway-Based, Object-Relational DBMS and Object-Oriented DBMS object persistence approaches enable us to achieve persistence in certain classes of object-oriented applications, and each approach has been therefore affected by the requirements of the class of applications it supports.

3.1 Gateway-Based Object Persistence (GOP)

The gateway-based object persistence (GOP) approach is a middleware solution that attempts to bridge the gap between the object-oriented paradigm data model and the non-object-oriented data model used to store the objects [6]. These systems provide a sort of runtime mapping or translation between the two models in a fashion that is transparent to the programmer. The only attention required by the application developer in process; when non-trivial mappings between types in the models need to be explicitly stated.

This approach is featured by being both application and data-independent. It is used to support an object-oriented programming model for applications which are using traditional non-object-oriented data stores to store data for an object.

Table 1. Gateway-Based Object Persistence

Advantages	***Limitations***
• Integrating enterprise information systems and providing a common framework for building object-oriented applications. • Managing shared, distributed, heterogeneous, and language-neutral persistent business objects. • Building a GOP application that legacy applications continue to work on data that are also being accessed by the new application. • Providing object-oriented access to legacy non-object-oriented data. • Building applications that have an overwhelming need to access legacy data and heterogeneous data access, while allowing legacy applications to continue to work on the legacy data.	• Randomly/arbitrarily complex objects in a legacy database system. • Blindly mapping object-oriented models to non-object-oriented databases because it gives bad performance and complex application logic.

This approach commonly supports when programmers intend to use existing non-object-oriented data stores but write applications using object-oriented programming models[3]. The data store schema that is used to store the persistent state of the objects in the data store is different from the objects having a different model (object-oriented) for an application. So, the system which is adopting GOP method performs a mapping between both object-oriented schema and non-object-oriented data store schema [9]. While application is executing, the GOP system translates objects from the representation used in the data store to the representation used in the application and vice versa. Table 1 explains the capacity and limitations of Gateway-Based Object Persistence.

The standards activity relevant to GOP is being developed by the Object Management Group (OMG). Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable and reusable enterprise applications in distributed, heterogeneous environments [7]. The most important specification OMG has adopted is CORBA (Common Object Request Broker Architecture). Aside from CORBA, the following adopted specifications are directly related to object persistence: Persistent Object Service, Object Query Service, Object Relationships Service, Object Transaction Service, and Object Security Service.

Gateway-Based Object Persistence Applications: There are several systems that are using GOP e.g. VisualAge C++ Data Access Builder, SMRC, ObjectStore Gateway, Persistence, UniSQL/M, Gemstone/Gateway and Subtleware/SQL.

GOP applications can access other OODBMSs and can store complex objects natively in them while continuing to access and update data in legacy databases but this feature is still facing some problems and challenges and experts are working on them. Some problems are related to integration of object persistence with object query, object transaction and workflow, and object security. The OMG group is continuously specifying standards in this area for greater use of objects.

3.2 Object-Relational DBMSs (ORDBMSs)

The object-relational DBMS persistence (ORDBMS) solution is a bottom-up solution that attempts to build on, or extend, the existing relational data model to work with objects. The premise here is that the RDBMS has been extremely successful in business applications, implemented by successful vendors, and already has a standard query language to expand [6]. RDBMS persistence depends on a persistence delegate, code that hides or abstracts the details of object and table while maintaining table concurrency [18].

This method is a bottom-up approach which is featured by being data (or database) oriented. In today’s database applications, the relational model is very much successful in practice and already SQL is accepted as an universal standard. ORDBMSs is used to add support for object oriented data modeling by extending both the relational data model and the query language along with retaining the already successful technology like SQL of a relational DBMS relatively intact.

Table 2. Object-Relational DBMSs

Advantages	Limitations
• Extending the usefulness of existing, legacy data stored in relational databases. • Addressing the mismatch and performance issues while accessing relational data from an object-oriented programming language. • Applications that need extremely good query support, excellent security, integrity, concurrency and robustness, and high transaction rates.	• Focuses only on data stored in relational databases or whatever in the future can be stored in extended relational databases.

The standards activity on this area is based on an extension of the SOL standard. X3H2 (the American committee responsible for the specification of the SQL standard) has been working on object extensions to SQL. These extensions have become part of the new draft of the SQL standard named SQL3. The SQL3 standard is an ongoing attempt to standardize extensions to the relational model and query language.

ORDBMS Applications: There are two classes of object-relational DBMSs in the market; those that have been built from scratch (e.g., Illustra, UniSQL), and those that are built by extending existing relational DBMSs (e.g.: DB2, Informix, Oracle, and Sybase). Following table (Table 2) listed the advantages and limitations of Object-Relational DBMSs.

3.3 Object-Oriented DBMSs (OODBMSs)

This approach is a top-down approach which is featured by being application or programming language centric. The main usage of OODBMS is to provide an effective method to add persistence to objects so that they can be used in an object-oriented programming language (OOPL) like C++ or Smalltalk.

There are two approaches to creating an object-oriented database, “Extended database” and “Persistent programming language” [8]. Extended database add the concepts of object orientation to existing object-oriented language, and Persistent programming languages extend existing object-oriented languages to deal with databases by adding concepts such as persistence and collections.

The OODBMSs are normally referred to as persistent programming language systems since they have their base platform in object-oriented programming languages.

Table 3. Object-Oriented DBMSs

Advantages	Limitations
• Storing application objects, e.g., presentation or view objects. • Providing seamless persistence from a programming language point of view. • Avoiding mismatch issues by providing extensive support for the data modeling features of one or more object-oriented programming languages. • The applications that need excellent navigational performance. • Object-oriented database models allow better support for managing complex objects and encapsulation, real-time systems that need to handle large and complex applications would require an object oriented approach [20].	• OODBMSs do not provide as good a query facility as ORDBMSs. • The transaction rates supported by the OODBMSs do not yet approach the high rates achieved by relational DBMSs on standard transaction processing benchmarks.

The standards activities for OODBMSs have been specified by the Object Database Management Group (ODMG). ODMG is a consortium that consists mainly of OODBMS vendors. ODMG has specified the ODMG-93 standard. ODMG-93 defines an Object Definition Language (ODL), an Object Query Language (OQL), C++ and Smalltalk language mappings to ODL and OQL.

OODBMS Applications: Object-oriented DBMSs support for persistent objects from more than one programming language, distribution of data, advanced transaction models, versions, schema evolution, and dynamic generation of new types. Even though many of these features have little to do with object orientation, object-oriented DBMSs emphasize them in their systems and applications. There are several object-oriented DBMSs in the market (e.g., Gemstone, Objectivity/DB, ObjectStore, Ontos, O2, Itasca and Matisse). Table 3 describes the advantage and limitations of Object-Oriented DBMSs.

4. GATEWAY-BASED, OBJECT-RELATIONAL DBMS AND OBJECT-ORIENTED DBMS VS OBJECT ORIENTED APPLICATION CHARACTERISTICS AND REQUIREMENTS

This section will discuss about the different object-oriented applications requirement, their need, behavior and characteristics. The focus will be on how and which approach will be useful and why?

4.1 Data Modeling

A data model is a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints. The object-oriented data model extends the representation of entities by adding notions of encapsulation, methods (functions), and object identity. The object-relational data model combines features of the object-oriented data model and the relational data model.

The object-relational data model extends the relational data model by providing a richer type system including collection types, and object orientation. Object orientation provides inheritance with subtypes and sub-tables, as well as object (tuple) references.

4.1.1 Object Identity

Object identity is a major concern for object persistence. In a programming environment an object can be created, assigned or copied, and can be deleted or accessed by program. Generally virtual address memory of a transient object is considered as object identifier.

4.1.2 Complex Object

An object can have any number of attributes, arguments, and/or elements. A complex object mechanism allows an object to contain attributes that can themselves be objects.

4.1.3 Composite Object

Object-oriented applications utilize a composite object as a group of objects that are part of a parent object that is typically a collection. Composite objects are individual objects that are related and form part of a group.

4.1.4 Relationships

Instead of "building in" a few fixed relationships, systems must be capable of supporting user-defined relationships, for two reasons: first, these few proposed relationships are not sufficient for all applications; second, their required semantics vary from one installation to another, from one application to another, or even from one use of the relationship to another [16].

4.1.5 Encapsulation

Encapsulation is a technique for minimizing interdependencies between separately-written and separately-compiled modules by defining a strict external interface: objects are accessible only through their external operations [13].

4.1.6 Inheritance

The objective of organizing objects in a hierarchy of classes is to share properties of the objects in useful, economical and meaningful ways through inheritance [10].

Table 4. Data Modeling [3]

Feature	**Object Persistence Approaches**		
	Gateway-Based Object Persistence (GOP)	***Object-Relational Database Management System (OR DBMS)***	***Object-Oriented Database Management System (OODBMS)***
Object Identity (OID)	Support limited by underlying database	Starting to provide support through row identification	Supported
Complex Objects (objects containing non-first-normal form data)	Can be supported using schema mapping	Supported by extensions to the relational data model	Supported
Composite Objects (grouping of objects for copying, deleting, etc.)	Can be supported using schema mapping(however, there can be limitations)	Starting to provide support through a combination of triggers, abstract data types, and collection types	Supported using class libraries
Relationships	Can be supported using schema mapping and code generation	Strong support available including referential integrity	Supported using class libraries
Encapsulation	Supported at application but not at database	To be supported using abstract data types (row objects will remain un-encapsulated)	Supported (but broken for queries)

Feature	Object Persistence Approaches		
	Gateway-Based Object Persistence (GOP)	***Object-Relational Database Management System (OR DBMS)***	***Object-Oriented Database Management System (OODBMS)***
Inheritance	Can be supported using schema mapping (however, there can be technical limitations)	To be supported (separate inheritance hierarchies for tables and abstract data types)	Supported as in an object oriented programming language (OOPL)
Method overriding, overloading, and dynamic dispatching	Supported as in an OOPL	Supported (method dispatching is based on the generic function model not the classical object model)	Supported as in an OOPL

4.1.7 Method Overriding, Overloading and Dynamic Dispatching

Virtual operations represent a powerful mechanism for implementing software reuse, late binding and polymorphism. A virtual operation is the specification of a routine that can be redefined in descendant classes [14].

Collection types include nested relations, sets, multisets, and arrays, and the object-relational model permits attributes of a table to be collections. Table 4 compares the three approaches under various data modeling parameters.

There are various modeling features given by existing object-oriented programming languages like C++ and Smalltalk. The applications that are written in these programming languages use a number of object-oriented modeling features like encapsulation, inheritance, and dynamic binding. There are several complex issues arise in providing support for an object-oriented data model and the table below discuss those issues in detail.

4.2 Data Access

In this section, we discuss about how application objects can be created and stored, how support is provided for navigational and ad hoc query types of access to persistent data and the interaction between client and server, specifically the method by which objects are communicated between client and server. Also we discuss some important application support items including schema evolution, integrity constraints etc. Table 5 provides data access comparisons among GOP, ORDBMS and OODBMS approaches.

The brief detail of above features is written below:-

4.2.1 Creating and accessing persistent data

The best way to support persistence is to do it in a way that it is possible to create persistent and transient objects of the same type in an application. There are two main methods of adding persistence to objects of an instance, one is by overloading the new operator and other is by requiring that every class having persistent instances inherit from a common class and definition and implementation of this common class is provided by the database system. The reading of persistent data in all three approaches can be made virtually transparent to the application. However, updating data in a GOP system is typically not transparent and an application will need to inform the system explicitly of objects that have been changed. Updating data in GOP can be done by having some (little) encapsulation. For example, update of relationships, but changing an atomic field like an integer is impossible to encapsulate.

In an ORDBMS, updates are non-transparent as these are done using a separate UPDATE statement. The OODBMSs vary in their degree of transparency, ranging from ObjectStore where updates can be made completely transparent, to other systems such as Versant where an object has to be explicitly marked "dirty" by an application.

4.2.2 Navigation

OODBMS development was driven by the applications that needed fast navigational access (e.g., verification and routing an integrated circuit might be an extremely CPU-intensive operation that requires fast access to component objects). OODBMSs (e.g.,ObjectStore) provide extremely fast navigational access to data by making use of operating system support for page faulting. In GOP system, navigation can be supported by mapping object accesses to the databases that store the data. Naive algorithms for navigation using a relational database could cause very poor performance because of generating one SQL query for every object access.

Table 5. Data Access [3]

Feature	**Object Persistence Approaches**		
	Gateway-Based Object Persistence (GOP)	**Object-Relational Database Management System (OR DBMS)**	**Object-Oriented Database Management System (OODBMS)**
Creating and accessing persistent data	Supported (might not be entirely transparent to the application)	Supported (not transparent since application always has to take explicit action)	Supported (degree of transparency depends on individual product)
Navigation	Can be supported by transparently mapping object accesses to underlying database operations (pre-fetching/caching needed for good performance)	Currently supported by joins (to be supported efficiently using row identification)	Supported efficiently by most products
Ad hoc query facility	Supported using data store specific query	Excellent support (impedance mismatch	Supported but with limitations

Feature	Object Persistence Approaches		
	Gateway-Based Object Persistence (GOP)	**Object-Relational Database Management System (OR DBMS)**	**Object-Oriented Database Management System (OODBMS)**
	language (not integrated well with object representation)	remains an issue)	
Object server vs. page server	Object server	Object server	Can be page server or object server
Schema evolution	Limited support (complete support might be difficult to provide)	Supported	Supported
Integrity constraints and triggers	No support	Strongly supported	No support

GOP systems handle this performance problem by maintaining a large cache of application objects in main memory, and by providing facilities for fetching objects before they are needed.

4.2.3 Ad hoc query facility
A system which uses GOP method normally does not implement a new query language on the representation of the object. The query under GOP method works on the base data model which is not object-oriented and this does not proceed well with the object model of the application and hence it creates the problems of impedance mismatching. In the areas related to optimization and index management, an ORDBMS supports queries in an efficient manner. In OODBMS, the support of query language is an extension of the object-oriented programming language [3]. Encapsulation is not supported in OODBMS query languages but they are allowed to access the structure of the data. This can not be avoided after the time when ad-hoc queries needed arbitrary computations on the data.

4.2.4 Object server Vs page server
In client/server architecture, the workload and tasks are divided both for client and server. So the database management systems need to make use of the resources available at the client and the server in efficient way. An object server can either receive requests for a single object (which is using for instance, an object identifier) or a set of objects using a query. ORDBMSs and GOP systems can be considered as object servers, but OODBMSs can be both object and page servers. Examples of page server architectures include ObjectStore and O2.

4.2.5 Schema evolution
Two separate parts are involved in Schema evolution. The first involves changing the schema, and the second involves changing and developing existing Data (that is in the form of the old schema) to their new representation based on the modified schema.

In a GOP system, schema evolution support might be extremely limited. However, schema without change in the underlying data might be easy to achieve and we can call it mapping evolution. ORDBMSs can provide strong support for schema evolution of table definitions. In OODBMSs, the data model is complex so schema evolution in an OODBMS cannot be completely automated as in a relational DBMS.

4.2.6 Integrity constraints and triggers
There is no GOP system available in these days that provide support for integrity constraints and triggers. ORDBMSs provide excellent support for integrity constraints and triggers. OODBMSs provide virtually no support for integrity constraints and triggers.

4.3 Data Sharing
In this, we discuss about how support is provided for applications by the various DBMSs for sharing data between concurrent users, crash recovery, advanced transaction models (long transactions, versioning, nested transactions), and distributed access to data.

4.3.1 ACID transactions
OODBMSs support the conventional type of short transactions termed ACID transactions. OODBMSs do support various types of locking. The standard lock types are page locks and object locks (also known as record locks in RDBMSs). GOP System provide limited support for ACID (atomicity, consistency, isolation, and durability) transactions since the object cache maintained at the application is loosely coupled to the DBMS. ORDBMSs support all the traditional lock types available in relational DBMS (tuple, page, and table locks).

Table 6. Data Sharing [3]

Feature	Object Persistence Approaches		
	Gateway-Based Object Persistence (GOP)	**Object-Relational Database Management System (OR DBMS)**	**Object-Oriented Database Management System (OODBMS)**
ACID transactions	Support limited by the underlying data store (cache management might cause complications)	Supported	Supported
Crash recovery	Recovery handled by the backend data store (cache is not recovered)	Strongly supported	Supported (degree of support varies with individual product)
Advanced transaction model	No support	No support	Supported in some products
Security, views, and integrity	Support determined by the underlying	Strongly supported	Limited support

Feature	Object Persistence Approaches		
	Gateway-Based Object Persistence (GOP)	**Object-Relational Database Management System (OR DBMS)**	**Object-Oriented Database Manageme nt System (OODBMS)**
	data store		

4.3.2 Crash recovery
GOP systems provide whatever support is available in the underlying data store. ORDBMSs are strong in this area because of relation DBMS extension. OODBMSs provide recovery support and this support is not robust as it is in commercial relational DBMSs which provide more advanced features such as media recovery.

4.3.3 Advanced transaction models
OODBMSs provide better support for advanced transaction model that is not supported very well by existing relational DBMSs and GOP or ORDBMSs.

4.3.4 Security, views, and integrity
ORDBMSs support robust security mechanisms using the view mechanism, and by ensuring that the entire application executes in its own address space. In contrast, OODBMSs by using the page server concept, allow clients to cache data for acceptable performance.

Data sharing characteristics compared among GOP, ORDBMS and OODBMS in Table 6. ACID transactions, crash recovery, advanced transaction models and security, views and integrity are the parameters used for comparison.PERSISTENT SYSTEMS: USING C++, JAVA, .NET

The Object Database Management Group (ODMG) standards define classes and other constructs for creating and accessing persistent objects from C++ and from Java [8].

4.4 Persistent C++ Systems
C++ is a powerful language and very much preferred for system programming. C++ language is based on object oriented concepts, so its object oriented features provides extended support for persistent even without changing the language itself [8]. Inheritance is a great feature of OOPs that help us extending the persistent feature in sub-classes. For example we can declare a class called PersistentObject with number of attributes and corresponding methods to support persistence; any other classes that should be persistent can be made a subclass of this class, and thereby inherit the support for persistence.

In C++ class libraries are very much used for writing the components, and same can be used for providing the support for object persistent. There are both positive and negative aspects of class libraries if using for persistent support. Class libraries require minimal changes to C++ and relatively easy to implement. However, it comes with some drawbacks also. The programmer needs a deep analysis and much more time is required to write a program that handles persistent objects. The complexity for the programmer is to specify integrity constraints on the schema or to provide support for declarative querying. Some persistent C++ implementations support extensions to the C++ syntax to make these tasks easier. EC++ objects use C++ transparently to distribution and persistence [15].

ODMG has been working on standardizing language extensions to C++ and Smalltalk to support persistence and on defining the class libraries to support persistence. The OMDG standard provides all functionality via class libraries, without any extension to language.

4.5 Persistent Java Systems
Java is the most preferred language for writing databases based applications. Due to rapid growth in usage, Java was also improved with market trends. There was requirement and demand to additional support for persistent encourages many programmers and organizations to add frameworks and define standards. Compare to C++, Java is differentiating in use of persistent by reachability.

We should make class persistence capable if the object of this class is reachable from persistent root. This can be achieved by running a post processor on the class code generated by compiling the Java program. Manually making a class persistent capable is possible. By inserting the appropriate declaration we can make class persistence capable, however it is a complex process.

The serialization and RMI features of Java, make basic object persistence a possibility without excessive effort required from the programmer [17]. However some knowledge of data movement and the underlying storage mechanism is still required.

Enterprise Java Beans (EJB) is an interface layer implemented on top of a JDBC/RDBMS which provides a consistent method of presenting persistent data application components that can be shared across many simultaneous remote and local client connections [18]. Frameworks such as JDO and Hibernate allow a Java programmer to program completely in the object-oriented paradigm while persisting data in a relational database [19]. The framework handles the mapping of objects to relational tables.

The ODMG model for object persistence in Java programs differs from the model for persistence support in C++ programs. The biggest difference is the use of persistence by reachability in Java [8]. Objects are not explicitly created in a database. Instead, names are given to objects in the database that serve as roots for persistence. These objects, and any objects reachable from these objects, are persistent.

The ODMG standards for Java define collection type such as DSet, DBag, and DList that extend the standard Java collection types [8]. Java already defines an iterator type to iterate over collections.

4.6 Persistent .NET Systems
Microsoft .NET is similar to Java in nature. .NET provides services to components at runtime via interception. At component creation, .NET creates an interceptor that wraps the component's interface and contains the .NET "property" logic to provide services at runtime [21]. There are many customized frameworks are available which supports object-relational mapping for .NET. Microsoft .NET provides two methods to serialize the objects. First method is using the XmlSerializer class defined in the System.Xml.Serialization namespace [9]. The second method is to use .Net formatters which are similar to the serialization process in Java. The XmlSerializer is the easier method but is not as efficient as the .NET formatters.

5. CONCLUSION

In this paper we have discussed the various features used in object-oriented applications; how well these features are supported in the GOP, ORDBMS and OODBMS object persistence methods; also the advantages and limitations of each of these methods.

The GOP method is simplest approach and referred as middleware approach. GOP method stores persistent objects using relational databases, hierarchical databases, or flat files. GOP is having many benefits but comes with drawbacks as well. This approach is good for integrating diversified enterprise information systems and providing a common framework for building object-oriented applications. GOP is recommended for managing shared, distributed, heterogeneous, and language neutral persistent business objects. Looking into the negative aspects, there are some disadvantages to blindly mapping object-oriented models to non-object-oriented databases. Applications that have an extreme need to access legacy data and heterogeneous data, while allowing legacy applications to continue to work on the legacy data are best suitable for using GOP method.

The ORDBMS is featured with handling complex data type, powerful query languages and high protection. ORDBMS method is considered as a bottom-up method, which combined the features of both relational and object oriented model. ORDBMS enhances the relational data model by applying object-oriented modeling features to it. This method is best suitable for extending the usefulness of existing, legacy data stored in relational databases. It has the good base for fixing the issues of impedance mismatch and performance penalty. Also, when all three object persistence methods are compared, ORDBMS method has the best robustness, concurrency, and crash recovery features. But, on the negative side, this approach concentrates only on data stored in relational databases or whatever data that can be stored in future extended relational databases. The best suitable applications to follow this approach are the one which strive for extremely good query support, high quality security, integrity, concurrency, robustness and high transaction rates.

The OODBMS method is considered as a top-down method, which involves adding persistence support to objects in an object-oriented programming language. It is an efficient method for saving application objects, e.g., view objects or presentation objects. This method is considered to be one of the best future methods for providing an efficient persistence mechanism, from a programming language perspective. Impedance mismatch is avoided by OODBMSs approach by providing wider support for the data modeling characteristics. But on the negative side, OODBMSs do not provide as good a query feature as ORDBMSs. In addition to this, the transaction rates which are supported by the OODBMSs have not yet reach the high rates supported by relational DBMSs. Applications which does not have complex query, that require an efficient navigational performance and that are ready to compromise with the security and integrity for attaining good performance are perfectly matched for using OODBMSs.

Even though every object persistence method has its own positive and negative impact on object data, in the coming days, it is more obvious that we will see the continued presence of OODBMSs that satisfy the needs of specialized markets, the continued existence of ORDBMSs that satisfy the needs of traditional commercial markets, and the increasing importance and existence of the Gateways combined with object query, object transaction and workflow, and object security. So selection of a best persistence method is an important factor for an object-oriented application developer for storing objects depending on the type of object-oriented application.

6. REFERENCES

[1] C. Booch, Object-Oriented Analysis and Design with Applications,

second edition, The Benjamin/Cummings Publishing Company, Redwood City, CA (1994).

[2] Matt Weisfeld, The Object-Oriented Thought Process, Third Edition 3ed.Sep.2008

[3] V. Srinivasan and D. T Chang, "Object persistence in object-oriented applications, " IBM Systems Journal, vol. 36, pp. 66–87, 1997

[4] Jim Coker, Object Persistence and Distribution, http://java.sun.com/developer/technicalArticles/RMI/ObjectPersist/

[5] Scott W. Ambler, Impedance Mismatch, http://www.agiledata.org/essays/impedanceMismatch.html

[6] Raffi Khatchadourian, Object Databases: an Analytical Approach, http://www.cse.ohio-state.edu/~khatchad/reports/khatchad-objdb.pdf

[7] Common Object Request Broker Architecture (CORBA) Specification, Version 3.2, http://www.omg.org/spec/CORBA/3.2/Interfaces/PDF

[8] Silberschatz−Korth−Sudarshan: Database System Concepts, Fourth Edition

[9] Patrik Hildenborg, Muhammad Irfan Tahir, Object Persistence: Persistence approaches in object oriented environment, http://www.idt.mdh.se/kurser/cd5130/msl/2005lp4/downloads/reports/object_persistence.pdf

[10] Ashrafuzzaman, M.; Kusalik, A.J., An implementation architecture for orthogonally persistent deductive object-oriented database systems, Database Engineering and Applications, 1999. IDEAS '99. International Symposium Proceedings

[11] S. J. White and D. J. DeWitt. A performance study of alternative object faulting and pointer swizzling strategies. In L.-Y. Yuan, editor, International Conference on Very Large Databases, number 18, pages 419–431, Vancouver, Canada, August 23-27, 1992.

[12] Vogelsang, H.; Brinkschulte, U.; Stormanolakis, M.;,Archiving system states by persistent objects, Engineering of Computer-Based Systems,1996.

[13] Systems,1996. Proceedings., IEEE Symposium and Workshop on Engineering of Computer-Based SystemsA. Snyder, “Encapsulation and Inheritance in Object-Oriented Programming Languages,” OOPSLA ’86 Proceedings, ACM Sigplan Notices, Vol 21 N. 11, pp. 38-45, 1986.

[14] B. Strousrtup, The C++ Programming Language, Addison-Wesley Series in Computer Science, 1986.

[15] Sequeira, M.; Marques, J.A.;, Can C++ be used for programming distributed and persistent objects?, Object Orientation in Operating Systems, 1991.

[16] Heiler, S.; Dayal, U.; Orenstein, J.; Radke-Sproull, S.;, An Object-Oriented Approach to Data Management: Why Design Databases Need It, 24th Conference on Design Automation, 1987.

[17] Tom Lunney and Aidan McCaughey, Proceedings of the 2nd international conference on "Principles and practice of programming in Java" June 2003, Publisher: Computer Science Press, Inc.

[18] Richard T. Baldwin,"Views, Objects, and Persistence for Accessing a High Volume Global Data Set", Proceedings 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies, 2003 , Page(s): 77 - 81

[19] James H. Paterson and John Haddow, "Approaches to object persistence in java projects" Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education ACM New York, NY, USA ©2004 , Volume 36

[20] Juhnyoung Lee, Sang H. Son, Myung-Joon Lee, Issues in Developing Object-Oriented Database Systems for Real-Time Applications, Proceedings of the IEEE Workshop on Real-Time Applications, 1994. On page(s): 136 - 140

[21] Roger Barga, David Lomet, Stelios Paparizos, Haifeng Yu, Sirish Chandrasekaran, Persistent Applications via Automatic Recovery, Proceedings of the Seventh International Database Engineering and Applications Symposium (IDEAS'03)