Object Persistence Techniques - A Study of Approaches, Benefits, Limits and Challenges

Clarence J M Tauro
Deparment of Computer Science
Christ University, Bangalore, India

Ritesh Kumar Sahai
Department of Computer Science
Christ University, Bangalore, India

Sandhya Rani A.
Department of Computer Science
Christ University, Bangalore, India

ABSTRACT
Object-Oriented paradigm becomes pioneer and best choice while selecting language and writing software solution. In last few decades there was significant change observed in developing software solutions. Most of the application developers prefer the object oriented model to exploit its benefits. The major benefit we can obtain from Object-Orientation is of course object itself and the feature that enable us making an object persistent. Object-Persistence feature contributes a major role in designing data model. If the techniques used for Object-Persistence are designed correctly then, we can obtain major benefits in the areas of software productivity, maintainability and cost reduction. There are many ways of implementing Object-Persistence among which Gateway-based method, Object-Relational database method and Object-Oriented database method are the three major categories.

In this paper, we discuss about the characteristics of various Object-Persistence techniques, the relevant areas in which those techniques can be employed efficiently and how those techniques can be used effectively on the basis of application characteristics and requirements. We also discuss about the benefits and limitations of persistence techniques. Further, our discussion continues on various challenges that come along the way of Object-Persistence and possible solutions to handle those challenges.

Keywords
Object-Persistence; Gateway-Based Object-Persistence; Object-Oriented Database; Object-Relational Database; Data Model; Data Access; Data Sharing

1 INTRODUCTION
Object-Oriented development is bottom-up approach, which focus on data instead of procedures that manipulates the data. Object-Orientation is an efficient, powerful, reliable and well-known technique that mimics a network of real life processes, scalable applications, tasks, and business rules of a domain. An object present in a software application consumes some amount of memory space and exists for a specific time. When an object is being created by an application, the scope of object is limited to the application life cycle. When the application terminates, object's life also terminates. The reason is, object is being stored temporarily in main memory. To keep object alive, application need to store the object in persistence storage [1].

The concept of maintaining the state of an object is termed as Object-Persistence. 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 [3]. A persistent object continues to exist, even when the application that created or used the object has finished execution.

In an application, persistence can be implemented in many ways, and few of them [16] are listed below:

- Storing object in a simple text file
- Storing object in an indexed sequence access mechanism
- Storing object in relational database
- Storing in an Object-Oriented database

A very simple way is to use text file as storage. All required information can be stored on file by doing file write operation. This stored information can be retrieved when restore is required by performing file read operation. This kind of persistence mechanism might be good for small applications, where limited information saving is required and information does not change frequently. But, representing complex information in a text file is very complicated. Lot of development efforts will be wasted in maintaining this. However text files are very flexible, easy to implement, can be accessed by more than one program, but they are not object friendly [5]. Object-Oriented programs exhibit various kinds of relationships with other objects, for example inheritance and references with other objects. The challenge here is how we can represent and maintain the different kinds of relationships on text file while saving the object.

Object-Persistence can also be implemented by another mechanism in which relational databases are used for object storage. This approach is most appropriate for storing business data as it can be easily formatted into rows and columns [16]. This mechanism can be a good choice when application needs 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 hierarchal order and relational database structured in tabular format. These are two different structures and a difference in two structures is known as "impedance mismatch" problem [6]. Due to these limitations of databases, this mechanism is not a best solution.

Implementation mechanism for Object-Persistence can be broadly classified into three categories. First one is the gateway-based approach which adds Object-Oriented programming access to persistent data stored using traditional non Object-Oriented data stores. The second mechanism is the Object-Relational DBMS approach which enhanced the extremely popular relational data model by adding Object-Oriented modeling features and the final mechanism is the Object-Oriented DBMS approach which adds persistence support to objects in an Object-Oriented programming language. OODBMS approach is more appropriate for unformatted and/or scientific information.

2 PERSISTENCE OF OBJECT - FUNDAMENTAL APPROACHES

Object-Oriented languages mainly focus on objects, object state and their behavior. An object can be created, initialized, accessed and manipulated by certain methods defined in class. By default, the object created by a program is transient, and object state is maintained till the program life. To maintain the state of object beyond the program life, also called persistent object, four approaches were identified and used widely in applications, namely persistence by class, persistence by creation, persistence by marking and persistence by reachability.

2.1 Persistence by Class

The simple approach is to declare a class to be persistent. Based on Object-Oriented property all objects that belong to this class will become persistent. However this is not flexible, as it is often required to have single class with both transient and persistent objects [4]. Declaring a class to be persistent is referred and interpreted as "persistable" by many OODB systems; objects belonging to the persistable class can be made persistent.

2.2 Persistence by Creation

An object can be transient or persistent. To make an object persistent new syntax has been introduced; by extending the syntax that used for creating transient objects. The object is persistent or transient decided by its creation.

2.3 Persistence by Marking

A different approach compare to persistent by class and creation. In this approach we mark an object as persistent after object creation process. All objects are created as transient without any change in object creation, based on requirement; an identified object can be marked explicitly as persistent object. Marking should be done after object creation and before program terminates.

2.4 Persistence by Reachability

In this approach selected objects are declared as persistent objects, often referred as root. All other objects, which are referred from this persistent object (root), directly or indirectly also becomes persistent object. In other words all objects referenced or reachable from root persistent objects are persistent. Advantage of this approach is to make complete data structure persistent by merely declaring the root of this structure as persistent. However it is much expensive to follow the chains in detection for a database system.

In an execution environment several objects will be created and performing specified task. All objects in a system communicate to other objects. The system state is defined by considering all objects and the relationship among them. Object pointers are used to express the relationship among objects in an Object-Oriented environment. "PersisentPtr" was introduced to keep track of the relationship between persistent object even after the program termination [12].

3 OBJECT PERSISTENCE TECHNIQUES – BENEFITS AND LIMITATIONS

In recent development era, most of the application developers prefer Object-Oriented models for implementing advance applications and hence object oriented paradigm is globally adopted now. As most applications are developed using object oriented model, the usage of persistent data is also increasing day by day. Due to this reason, there is a great demand to focus on different Object-Persistence techniques in order to develop more efficient object oriented applications.

There are three major techniques used for implementing Object-Persistence namely Gateway-Based, Object-Relational DBMS and Object-Oriented DBMS techniques. Each of these techniques has their own advantages and limitations. So selection of a best Object-Persistence technique clearly depends on the type and nature of the application.

3.1 Gateway-Based Object-Persistence (GOP)

The Gateway-Based Object-Persistence (GOP) approach is featured by being both application and data-independent. This approach is mainly followed in applications in which traditional non-Object-Oriented data stores are used to store data for an object.

This 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 [7]. This category of Object-Persistence methodology provides a kind of runtime mapping or translation between the two models in a manner that is transparent to the programmer. However, the application developer has to give more focus in the process when non-trivial mappings between types in the models have to be stated explicitly.

This approach widely supports when programmers wants to use existing non-Object-Oriented data stores but write applications using Object-Oriented programming models[8]. 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 approach performs a mapping between both Object-Oriented schema and non-Object-Oriented data store schema [9]. During the execution time of application, 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 advantages and limitations of Gateway-Based Object-Persistence methodology.

The Object Management Group (OMG) develops the standard activities which are relevant to GOP. The most important specification OMG has adopted is CORBA (Common Object Request Broker Architecture). Other than CORBA, Persistent Object Service, Object Query Service, Object Relationships Service, Object Transaction Service, and Object Security Services are the specifications which are adopted by OMG which directly relates to Object-Persistence.

Other than accessing and updating data in legacy databases, a GOP application can also access other OODBMSs and even it can store complex data objects natively in them. But, there are some issues and challenges with this feature and efforts are being carried out by the experts in resolving those issues. Integration of Object-Persistence with object query, object transaction and workflow, and object security are some of the issues which are related to GOP approach.

Table 1. Gateway-Based Object Persistence

Benefits	Limitations
• GOP provides Object-Oriented access to legacy applications as well as non-Object-Oriented data.	• It maps Object-Oriented models blindly to non-Object-Oriented databases because it gives bad

Benefits	Limitations
• Supports and manage shared, distributed, heterogeneous, and language-neutral persistent business objects. • Helps to integrate enterprise information systems and provide a common framework for developing Object-Oriented software solutions. • Developing a common GOP application that legacy applications continue to work on data that are also accessed by newer applications. • Developing solutions 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.	performance and complex application logic. • Lacks in handling randomly and arbitrarily complex objects in a legacy database system.

There are many systems available which are evolved as the applications of GOP approach namely VisualAge C++ Data Access Builder, SMRC, ObjectStore Gateway, Persistence, UniSQL/M, Gemstone/Gateway and Subtleware/SQL.

3.1.1 Object-Relational DBMSs (ORDBMSs)

The Object-Relational DBMS persistence (ORDBMS) method is a bottom-up approach which is characterized by being data oriented. In most of the today's database applications, the relational model is very extensively used and SQL is already considered as a global standard. ORDBMSs mainly supports for object oriented data modeling by incorporating both the relational data model and the query language. At the same time it retains the already successful technology like SQL of a relational DBMS relatively intact.

This type of persistence methodology attempts to build on, or extend, the existing relational data model to work with objects. The RDBMS has a standard query language to expand, implemented by successful vendors and it has been extremely successful in business related applications [7]. RDBMS persistence depends on a persistence delegate, code that hides or abstracts the details of object and table while maintaining table concurrency [13].

By the extension of the SOL standard, the standards activities are carried out on this area. X3H2 which is the American committee responsible for the specification of the SQL standard has been continuously working on object extensions to SQL. In the new draft of the SQL standard named SQL3, these object extensions have been included. Currently, the SQL3 standard is an ongoing attempt to build standard extensions to the query language and relational model. Table 2 lists the advantages and limitations of Object-Relational DBMSs Persistence methodology.

There are two different categories of ORDBMS applications available. The first category of ORDBMS applications are those that have been implemented from scratch which include Illustra and UniSQL. The second categories of ORDBMS applications are those that are implemented by extending the existing relational DBMSs which include DB2, Informix, Oracle, and Sybase.

Table 2. Object-Relational DBMS

Benefits	Limitations
• Very effective in developing applications that requires extremely good query support, excellent security, integrity, concurrency and robustness, and high transaction rates. • It extends the applications and use of existing, legacy data stored in relational databases. • ORDBMS addresses the mismatch and performance issues while accessing relational data from an Object-Oriented programming language.	• Limited focus and only concentrate on data stored in relational databases or whatever in the future can be stored in extended relational databases.

3.2 Object-Oriented DBMSs (OODBMSs)

The OODBMSs are normally referred as persistent programming language systems as they have their core platform in Object-Oriented programming languages. Object-Oriented DBMSs methodology is a top-down approach which is characterized by being application or programming language centric. Table 3 describes the advantage and limitations of Object-Oriented DBMSs.

"Extended database" and "Persistent programming language" are the two available methods for creation of an Object-Oriented database [4]. The functionality of extended database method is to add the concepts of object orientation to existing Object-Oriented language. The functionality of persistent programming languages method is to inherit the features of existing Object-Oriented languages and to extend the features to deal with databases by adding persistence and collections technologies.

The OODBMS provides an effective methodology to add persistence to objects so that they can be used in an Object-Oriented programming language (OOPL) like C++ or Smalltalk.

The Object Database Management Group (ODMG) consortium which is formed of OODBMS vendors specifies the standard activities for OODBMSs. ODMG has specified the ODMG-93 standard which defines an Object Definition Language (ODL), an Object Query Language (OQL), C++ and Smalltalk language mappings to ODL and OQL.

Object-Oriented DBMSs support for persistent objects from more than one programming language, advanced transaction models, schema evolution, distribution of data, versions and dynamic generation of new types. Among these features, few of them have less impact on the object orientation but still, Object-Oriented DBMSs includes them in their systems and applications. Gemstone, Objectivity/DB, ObjectStore, Ontos, O2, Itasca and Matisse are some of the Object-Oriented DBMSs which are available.

Table 3. Object-Oriented DBMS

Benefits	Limitations
• OODBMS models allow excellent support for managing complex objects and encapsulation, real-time systems that need to handle large and complex applications would require an object oriented approach [14]. • Object-Oriented databases are very useful for applications that need excellent navigational performance. • Object-Oriented model allows storing application objects, e.g., presentation or view objects. • An object-oriented based database provides seamless persistence from a programming language point of view. • OODBMS provides extensive support for the data modeling features of one or more Object-Oriented programming languages, which avoids mismatch issues.	• Object-Oriented databases lacks in providing good query facility as provided by ORDBMSs. • Object-Oriented database is not much efficient, the transaction rates supported by it do not yet approach the high rates achieved by relational databases on standard transaction processing benchmarks.

4 APPLICATION CHARACTERISTICS AND REQUIREMENTS

This section describes about the various Object-Oriented applications requirements, their usage, behavior and characteristics. It also explains how these characteristics and requirements are implemented by different Object-Persistence methodologies. On a broader perspective, we have classified the characteristics as data modeling, data access and data sharing characteristics. The below section discusses all these characteristics in detail.

4.1 Data Modeling Characteristics

A data model is a group of conceptual tools for describing data, data semantics, data relationships and consistency constraints. The Object-Oriented data model extends the representation of entities by adding basic object oriented elements such as encapsulation, methods and object identity.

Table 4. Data Modeling Characteristics [8]

Feature	Object-Persistence Approaches		
	Gateway-Based Object-Persistence (GOP)	Object-Relational Database Management System (OR DBMS)	Object-Oriented Database Management System (OODBMS)
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
Object Identity (OID)	Support limited by underlying database	Starting to provide support through row identification	Supported
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)
Relationships	Can be supported using schema mapping and code generation	Strong support available including referential integrity	Supported using class libraries
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
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)

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 object orientation and collection types.

4.1.1 Complex Object

A Complex object is something that can be viewed as a single thing in the real world but it actually consists of many sub-

objects. These sub-objects are actually the attributes of the complex object.

There are two types of complex object. The first types of complex objects are unstructured, difficult to determine the structure and require a large amount of storage. Examples of this type of complex object include Binary Large Objects (BLOB). The second types of complex objects have a clear structure and the sub-objects are said to be in a part-of relationship. For example, region objects are complex application objects that are frequently used in GIS applications.

4.1.2 Composite Object

A composite object is a group of related objects that can be processed in a single transaction. In a composite object, fields that reference other objects can be used to manipulate the dependent objects at the same time as the original object. With this, transactional safety can be assured, since either the entire transaction succeeds or if it fails the system state remains same as if the transaction never started. Table 4 compares the three approaches under various data modeling parameters.

4.1.3 Object Identity

Object identity is normally implemented using a unique, system-generated Object ID called OID. The value of the OID is not visible to the external user, but is used by the system internally in order to identify each object uniquely and to create and manage inter-object references. Even though there is a change in some or all of the values of variables or definitions of methods over a period of time, an object retains its identity.

4.1.4 Encapsulation

An object hiding its attributes behind its operations is termed as encapsulation. Encapsulation is most often achieved through information hiding (not just data hiding), which is the process of hiding all the secrets of an object that do not contribute to its essential characteristics [2].

4.1.5 Relationships

The relationship between any two objects encompasses the assumptions that each makes about the other, including what operations can be performed and what behavior results [2].

4.1.6 Method Overriding, Overloading and Dynamic Dispatching

In GOP systems, methods are not stored in the database and hence they are executed only using the object representation of the data. Where as in the case of ORDBMSs, both methods and data can be stored within the database and hence they are able to dispatch methods on objects within the DBMS server. In OODBMS systems, normally methods are executed at the client environment and hence most systems do not store methods in the database.

4.1.7 Inheritance

Inheritance is one of the object oriented concept that when a class of object is defined, any subclass that is defined can inherit the definitions of one or more general classes. The objective of arranging objects in a hierarchy of classes is to share properties of the objects in a useful, economical and meaningful ways through inheritance [10].

4.2 Data Sharing Characteristics

This section describes about how the various Object-Persistence methods provide data sharing support for applications. It describes how sharing of data takes place between concurrent users, how crash recovery is performed, details about advanced transaction models like long transactions, versioning and nested transactions.

4.2.1 Crash Recovery

A GOP system does not provide a very strong support on crash recovery feature but, it is able to provide whatever support is available in the underlying data store. On the other hand, ORDBMSs are strong in this area since these systems extend relation DBMSs. OODBMS systems do provide recovery support but it is not much robust in this area.

Table 5. Data Sharing Characteristics [8]

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

4.2.2 ACID Transactions

ACID stands for atomicity, consistency, isolation, and durability. GOP System provides limited support for ACID 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 like tuple, page, and table locks. OODBMSs support the conventional type of ACID transactions and also they support various types of locking. The standard lock types are page locks and object locks which are also known as record locks in RDBMSs [8].

4.2.3 Security, Views, and Integrity

ORDBMSs guarantee that the entire application executes in its own address space and it exhibits an effective security mechanism by using the view mechanism. An OODBMS system allows clients to cache data for acceptable performance by using the page server concept.

4.2.4 Advanced Transaction Models

GOP and ORDBMS approaches does not support advanced transaction model very well whereas OODBMSs approach provides better support for advanced transaction model. Table 5 shows the comparisons among GOP, ORDBMS and OODBMS methods with respect to data sharing characteristics.

4.3 Data Access Characteristics

This section explains about different data access features that are exhibit from various Object-Persistence methodologies. It describes how application objects are created and stored, in what way it provides support for navigational and ad hoc query types of access to persistent data, how interaction takes place between client and server and the support characteristics for schema evolution and integrity constraints.

4.3.1 Schema Evolution

There are two different phases involved in the Schema evolution. The initial phase involves schema change and the second phase involves modification and development of the existing data to their new representation based on the modified schema.

A GOP system provides a very limited support for schema evolution. In contrast, an ORDBMS system can provide a strong support for schema evolution of table definitions. An OODBMS system can also provide support for schema evolution but since the data model is complex, the schema evolution in an OODBMS cannot be completely automated as in case of a relational DBMS. Table 6 shows data access comparisons among GOP, ORDBMS and OODBMS methodologies.

4.3.2 Persistent Data Creation and Access

Persistence feature can be added to an object by following any of the two major methods; one method is by overloading the new operator and the other method is by inheriting a common class whose definition and implementation part is provided by the database system.

Accessing of persistence data can be made virtually transparent to the application in all three Object-Persistence methodologies. However, data updating is not transparent in a GOP system and hence the underlying application has to inform the GOP system whenever there is a change in the object state. In an ORDBMS system, updates are performed using a separate UPDATE statement and hence they are non-transparent. The extent of transparency varies in the OODBMS systems; in some cases, updates can be made completely transparent where as in others, updates needs to be explicitly specified by the application.

4.3.3 Triggers and Integrity Constraints

A GOP system cannot provide support for triggers and integrity constraints. An ORDBMS system can provide excellent support for and integrity constraints. OODBMSs cannot virtually provide support for triggers and integrity constraints.

4.3.4 Navigation

In GOP systems, navigation feature is supported by mapping object accesses to the databases where the data is stored. In case of OODBMS systems, the applications require fast navigational access. For example, in an integrated circuit application, the verification and routing requires fast access to component objects since it is an extremely CPU-intensive operation. In order to provide extremely fast navigational access to data, OODBMS systems makes use of operating system support for page faulting.

Table 6. Data Access Characteristics [8]

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

4.3.5 Ad-hoc Query Facility

A GOP system normally does not implement a new query language on the object representation. In this system, the query works on the base data model which is non-Object-Oriented and fails to work well with the application object model; this creates impedance mismatch problem [8]. In contrast, an ORDBMS system supports queries in a most efficient manner and performs well with the optimization and index management. In an OODBMS system, the support of query language is an extension of the Object-Oriented programming language.

4.3.6 Object Server Vs Page Server

In a normal client-server environment, both client and server need to work on the assigned loads or the tasks. Hence, a database management system has to utilize the available resources of client and the server in an efficient way. Queries are issued to an object server in order to request for a single object or a group of objects. GOP and ORDBMSs systems can be considered as object servers, where as an OODBMS system can be both object and page server. ObjectStore and O2 are few examples of page server architectures.

5 EMERGING CHALLENGES

5.1 Close Language Binding

There is a layer between database and the application using the database for storage in the traditional relational systems. Data is transient when it is available in application's memory and persistent when it is stored in database. The application program often required to perform read and write operation on database elements. However this separation is not present on Object-Oriented database, application program manage persistent as well transient data in similar way. In OODBMS any kind of data can be persistent and same methods can be used to perform operations on persistent as well as transient data. Also, it offers the object to be retrieved identically regardless of location. There are various approaches to implement Object-Persistence (discussed above in section 2) and a best way to achieve this by using "persistence by reachability" [15]. The language interface for OODBMS offers mechanisms to define and open databases, commit or abort transactions, acquire locks, and accessing data within database, and it does not require any additional overheads and constructors. If an object has not been retrieved from the database yet, in such cases the program does not need to do any additional operations. The underlying database system will automatically recognize the situation and retrieve the object [15].

5.2 Unified Development Process

Close language binding offers scalability and design flexibility that simplifies the lives of system analysts and designers. The good point of Object-Oriented DBMS is that it supports object oriented language and applies the object-orientation concept in database as well. Object-Oriented DBMS follows semantically same concept like Object-Oriented modeling and design, Object-Oriented analysis, and Object-Oriented languages allowing a unified conceptual approach during the whole development cycle. Unified approach has big advantage that it simplifies development and eases the communication between users, analysts, and developers. However in Object-Relational DBMS, the relational database objects have to be mapped to tables. This required some amount of time to create the relational tables and views, and it is also complicated to keep up the model with changes in the physical implementation. All this leads complexity in applications as well as creates difficulty to maintain the database. Object-Oriented database do not required semantic transformations as the same underlying object model is used during the whole development process. This unified approach offers higher quality systems that are flexible, scalable and easier to maintain [15].

5.3 High-Level Functional Requirements

In persistent Object-Oriented systems, following issues require critical attention [17]:

• The uniform interfaces need to be independent from data store type (JDBC, LDAP and some other interfaces, such as Informix early-times proprietary Java Object Interface, were important to deal with).

• Data store connection management – establishment, closing, transactions; persistent object lifecycle – creation, read, update, and delete (CRUD) operations.

• Mass creation of objects, data stores usage of them or their attributes (e.g. search operations). In distributed object environments, it is important to balance the load of database operations.

5.4 Object-Relational Impedance Mismatch

The Object-Oriented paradigm is efficient, capable and well known in application development. Object-orientation is based on proven software engineering principles. However the relational paradigm is based on proven mathematical principles. Both are having different structure and because the underlying paradigms are different, the two technologies do not work together seamlessly. The impedance mismatch becomes clearly visible when we look at the preferred approach to access; by using the object paradigm you traverse objects via their relationships whereas with the relational paradigm we join the data rows of tables. This fundamental difference in the structure results in a non-ideal combination of object and relational technologies. Following are challenges of storing objects in relational database [16].

• Mapping Object-Oriented classes to relational tables
• Inheritance relationship and relational tables
• Mapping association in relational tables
• Multiplicity, association class and link table
• Mapping aggregation-composition and shared aggregation
• Shared aggregation and reference table

5.5 Transparency and Object-Fault

Persistent of object should be independent of how application program modified and manipulates that object. Persistence independence, also referred as transparency, requires that it is indistinguishable whether programming code is operating on persistent or transient data [10]. Persistence independence is achieved by combining the features of 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.

5.6 Serialization

Simple arrays and hashes can be represented and written on file using commas, tabs, spaces or user defined format. Complex and nested data structures such as arrays of arrays or arrays of hashes have to lower down or serialized before writing into file. Also data items can be of global type and may contain references to other data items or pointers to native "C" data structure.

5.6.1 Object Storage Problem

In persistence by reachability mechanism, persistence can be defined as a network of persistent objects whose root is serializable. In other words, any object that a persistent object can touch automatically becomes persistent and thus a member of the network. In Java execution environment, during storage time, all the objects are converted in the network to a byte stream for storage in a flat file. The restriction of serialization stem from the fact that an individual object cannot be modified and the entire network of objects must be accessed as a whole. There is overhead that each time a member of the object network is stored, the entire byte stream that contains all objects in the network has to be serialized and stored. The opposite procedure occurs each

time a persistent object is accessed. A multi-user enterprise application requires fundamental database capabilities, like transaction and queries, however object serialization lacking in this area. In general we can say serialization is inadequate for robust application development. However ODMG binding for Java provides a good alternative for persistent object development with Java [18].

5.7 Data Source Related Issues

JDBC provides standard for communication between Java based application and relational database. Several frameworks have been designed with the focus on independence from data store type, however still JDBC-compliant data stores captured the popularity and most of attention and efforts. JDBC itself is quite convenient and simple set of interfaces, the actual implementations tend to deviate from the standard by providing proprietary extensions in some cases or postponing certain important implementations like metadata until better times [17]. This resulted into tough job for balancing between proprietary vendor extensions usage and the complexity involved in encapsulating the differences into application abstractions. Metadata, mappings of SQL data types to JDBC data types, processing of BLOB/CLOB data and unique incremental Id's were some of the concerns in this area.

5.8 Versioning Issues

Sometimes with change in application requirement, class definitions for all objects change over time, and that requires attention to deal with versioning objects. The serialization mechanism may have read in out dated objects, whose structure has been modified and different from current version of the class it belongs to. The possible cases that may occur should to be outlined and well defined in serialization specifications. A major change to an object, or changes in its location in a hierarchy, requires the developers to manually convert the out-of-date objects, in some cases this can be handled automatically or nearly so [5].

5.9 Concurrency Control

In recent development, concurrency control becomes basic need for scalable applications, which allows multiple users to access the data storage at the same time. Effective and efficient concurrency control mechanism enables guaranteed data integrity, and consistent information received by users. In real time execution environment, different applications or users may want concurrent access to persistent data stores, such as file or database. Few systems ignore concurrency issue altogether; others offer different types of locking schemes.

5.10 Boundaries

Traditional and ordinary files are based on byte stream and generally they do not provide any limitation and restriction on boundaries. File users need to decide how to represent distinct data item on file and make them recognizable on disk so that they can be retrieved easily. Indexed Sequential Access Method (ISAM) and DBM systems are based on Record-Oriented structure and RDBMS provides record and column boundaries. It is always good to have well defined data structure, slotted in grid structure else "impedance mismatch" will occur. Latest solutions, such as OR-DBMS and OO-DBMS, attempt to make this "restriction" or "failure" a non-issue.

5.11 Security Issues

Security is a big concern when a serialized object traveling across the internet through sockets or network communication. This serialized object can be read by unintended parties, or may be tampered and modified while in transit. To prevent the data from being written when the object is serialized; sensitive data, such as socket file descriptors, or other handles to system resources, should be made private as well transient. Also when the object is retrieved back from a stream, only the originating class can assign a value to the private data field. A validation method can also be used to check the integrity of a group of objects when they are retrieved from a stream.

To avoid security problems, the efficient way is to encrypt the serialization stream, ensuring both privacy and integrity. To achieve this, customized readObject and writeObject methods can be implemented, or can be achieved by using the Externalizable. For a global application, ObjectInputStream and ObjectOutputStream class can be customized to encrypt the entire object stream [5].

5.12 Machine independence

During computation operation files are being created in different systems and on different type of machines. These files are being used by other systems with different type of machines. This requires continuous tracking of differences in size and byte order of integer, and floating-point representation.

6 CONCLUSION

In this paper we have discussed the different Object-Persistence techniques, advantages and limitations of each of these techniques. Our discussion continued further on the characteristics and requirements of applications; and how well these features are supported by major category of Object-Persistence techniques, such as GOP, ORDBMS and OODBMS. Later we discussed more about the object persistent challenges.

Gateway-based technique basically uses traditional non-Object-Oriented data stores, for example flat files, relational and hierarchical databases. Gateway-based is middleware approach and very good for providing a common framework for building Object-Oriented applications and integrating diversified enterprise information systems. It offers excellent support for managing shared, distributed, heterogeneous, and language neutral persistent business objects. GOP is having disadvantage that it blindly maps Object-Oriented models to non-Object-Oriented databases. It is most suitable for applications that have critical need to access legacy and heterogeneous data; while allowing legacy applications to continue to work on legacy data.

Object-Relational technique involves improving the relational data model by adding the Object-Oriented modeling features to it. Object-Relational technique is a bottom-up approach and very efficient for extending the usefulness of existing, legacy data stored in relational databases. It addresses the mismatch and performance issues while accessing relational data from an Object-Oriented programming language. Object-Relational techniques have the best robustness, concurrency, and crash recovery characteristics among all three Object-Persistence methods. The major drawback of Object-Relational technique is that it concentrates only on data stored in relational databases or whatever in the future can be stored in extended relational databases. Object-Relational is very effective technique in developing applications that requires extremely good query support, excellent security, integrity, concurrency and robustness, and high transaction rates.

Object-Oriented database adds persistence support to objects in an Object-Oriented programming language. Object-Oriented is a top-down approach and provides excellent support for storing application objects, for example, presentation or view objects. Object-Oriented databases are best for providing seamless persistence, from a programming language point of view. The good point about Object-Oriented databases is that it avoids the impedance mismatch by providing extensive support for the data modeling features. On the negative side of Object-Oriented databases, it lacks in providing good query facility as provided by Object-Relational systems. OODBMS is suitable for applications that need excellent navigational performance, that do not have complex query, and that are prepared to sacrifice some integrity and security for achieving good performance. Object-Oriented databases are being preferred in modern trend as it offers close language binding, unified development process that helps developer a lot in development process.

All persistence techniques, including GOP, ORDBMS, and OODBMS are having both advantages as well as limitations, and they have their own positive and negative impact on object data. Recent development trend shows that, all three techniques are having some importance and they play major roles for different application characteristics and requirements. Object-Oriented database satisfies the needs of specialized markets, so it is obvious to see the continued presence of it. On the other side Object-Relational database fulfill the needs of traditional commercial markets; Gateway-based technique combined with object query, object transaction and workflow, and object security, hence its importance and existence is keep increasing. Effective and efficient persistence technique selection is an important factor and it clearly depends upon the application characteristics and requirements. Based on requirements, an application developer can choose best suitable Object-Persistence technique for storing objects. Every Object-Persistence technique has some associated challenges in it; there has been a continuous effort carried out in addressing those challenges and already many solutions exists in the object storage field.

7 REFERENCES

[1] Clarence J M Tauro, N Ganesan, Ritesh Kumar Sahai and Sandhya Rani A., Comparative Study on Object Persistence Methods. International Journal of Computer Applications 42(7):17-, March 2012. Published by Foundation of Computer Science, New York, USA

[2] C. Booch, Object-Oriented Analysis and Design with Applications, second edition, The Benjamin/Cummings Publishing Company, Redwood City, CA (1994).

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

[4] Silberschatz−Korth−Sudarshan: Database System Concepts, Fourth Edition, 2001

[5] Jim Coker, Object Persistence and Distribution, http://java.sun.com/developer/technicalArticles/RMI/ObjectPersist/index.html, Feb 1997.

[6] Scott W. Ambler, Impedance Mismatch, http://www.agiledata.org /essays/ impedanceMismatch.html , 2005

[7] Raffi Khatchadourian, Object Databases: an Analytical Approach, http://www.cse.ohio-state.edu/~khatchad/reports/ khatchad-objdb.pdf, 2006

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

[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] 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

[14] 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

[15] Richard G. Gibson, Object Oriented Technologies: Opportunities and Challenges, Idea Group Publishing, 1999, Page: 47

[16] Bhuvan Unhelkar, Practical Object Oriented Design, Thomson Social Press, 2005

[17] Adomas Svirskas, Jurgita Sakalauskaite, An Approach for Solving Java Object Persistence Issues using RDBMS and other Data Sources.

[18] Douglas Barry, Solving the Java Object Storage Problem, 0018-9162/1998 IEEE