

Object Persistence

Persistence approaches in object oriented environment

Patrik Hildenborg phg01001@student.mdh.se

Muhammad Irfan Tahir min04002@student.mdh.se

18th May 2005

Abstract

The need for persistence has been around for as long as computers have been able to store data, in memory and on discs. Many different designs for persistence were developed and throughout the years changed to accommodate the ever higher demand from application developers. When the introduction to object oriented programming arrived, it quickly became a popular way to design applications. Unfortunately, the persistence designs available were not always compatible with the object oriented environment. The need for object persistence was quickly recognized and development for more object friendly persistence techniques took off. Today, there are many techniques that support object persistence and it is widely used in today's applications. A good object persistence framework will shield the programmer from persistence specific information. Instead the programmer can concentrate on the business domain of the application, improving the quality of the product.

In this paper, we will discuss how important persistence is in Object oriented applications. The paper will discuss different approaches of object persistence and which approach is better to adopt in object-oriented applications. Different object persistence approaches in .NET and Java will be discussed.

Contents

1	Introduction	3
2	About the design of a persistence layer	4
3	Object Persistence and Approaches in Object oriented applications	5
3.1	Object Persistence Approaches	7
3.1.1	Gateway-based object persistence (GOP)	7
3.1.2	Object-relational DBMSs (ORDBMSs)	8
3.2	Object-oriented DBMSs (OODBMSs)	8
3.3	Major characteristics and requirements of object-oriented applications and choice of each of the three approaches to object persistence	9
3.3.1	Data Modeling	9
3.3.2	Data access	11
3.3.3	Data sharing	13
4	Object persistence in Java	14
4.1	Serialization	14
4.1.1	Write object	15
4.1.2	Read object	15
4.1.3	Customizing streaming	16
4.1.4	Advantages / Disadvantages	16
4.2	Enterprise JavaBeans (EJB)	16
4.2.1	Container managed persistence	17
4.2.2	Bean managed persistence	17
4.2.3	Advantages / Disadvantages	17
4.3	Java Data Objects (JDO)	18
4.3.1	Advantages / Disadvantages	19
5	Object Persistence in .NET	19
5.1	Serialization	20
5.1.1	XmlSerializer	20
5.1.2	.NET Formatters	21
5.1.3	Advantages / Disadvantages	21
5.2	ObjectSpaces	21
5.2.1	Advantages / Disadvantages	22
6	Conclusion	22

1 Introduction

Often, you can split an application into two parts, one part containing how the application works, another part containing objects and information that the application works with, for example objects of type `Employee` in an employee registration application. The latter part is called the business domain[10]. And when you work with your application you would probably want to save the changes that you have made to the business domain. This means that the application must have means to save the information you have added and also restore it the next time you start the application. When we talk about persistence we mean just that, a way to store information between runs of a program (application).

You can achieve persistence in several ways. One for example is that you write all the essential information to a text file, and when you want to restore the information, you just read from the text file. To accomplish this you would have to write code for writing the information to a file and also for reading the data from a file. This might be a good solution for applications where the information to be saved is limited and is not going to change. However, once that information gets more complex it will be hard to represent it in a simple text file. Also if you later have to change the information to be stored you have to rewrite code for writing and reading the information to a file. That is time wasted for a programmer. One solution is to use relational databases, but as we shall see it is not always the best choice.

As you might have figured out from the name, object persistence is to store objects or object information. One way to do this is as stated earlier, to use text files. Though text files are flexible and straightforward to work with they are not object friendly[12]. When the information we wish to save to a text file becomes more complex than a simple parameter list, saving to text files will be a hard task. In object oriented programming for example, how would you represent different relationships like inheritance and references to other objects? You can also use relational databases to store your objects. The problem here is that relational databases are structured in a tabular configuration while object oriented instances are typically structured in a hierarchical manner. This difference in structure is called "impedance mismatch"[11]. Relational databases and object oriented instances are just too different for it to be an easy way to map from one structure to another. Because of this the programmer must, when using relational databases, spend a lot of time specifying just how objects should be represented in the database.

For easy use of object persistence we would want either an object oriented file format that is well integrated in the programming environment, or a well-specified interface which can bridge the gap between for example, relational databases structure and the object oriented structure, and overcome the "impedance mismatch". There are different techniques for solving these problems and we will have a look at some solutions later in the paper. Also worth mentioning is that different programming environments like .NET and Java sometimes use different techniques and we will look at what techniques they use. There is also different kinds of databases that support object oriented environments and thus eliminating "impedance mismatch".

2 About the design of a persistence layer

The design of the persistence layer in an application is an important task, and depending on what type of application it is different designs may be required. Some common approaches[22] to design our own object persistence layer will be presented below.

One approach is to include the persistence behavior in the business domain and include all the storage specific code in their class implementation. This design is fast, both in runtime and in time it takes to implement it. But it is seldom a good idea to mix business code with persistence code. For example, a simple change as in a change of name in a row or a column in a database would require the programmer to change all objects in the design. This design is best usable in small projects.

One design to try to remedy the first approach is to separate the business logic from the persistence logic. You create so called "data classes" that handles all the persistence in the application. While this approach gathers all persistence specific code in one place and separates it from the business domain, it still requires change in all the data classes when simple changes to the database is made. This approach is still fast both in runtime and in implementation. Still to dependent on the data-store to be able to use this design in larger software projects.

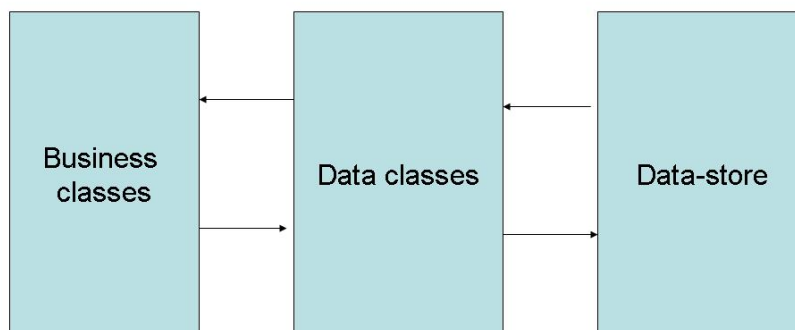


Figure 1: Second approach

The third approach is to create a more robust persistence layer between the application and its data-store. The design is intended so that simple changes to the data-store doesn't require any change in the object code. A way to do this is to use xml files to store all persistence specific information, and let the classes that handles the persistent behavior be the mechanism for working with the information that the xml file contains. So in the case

that the programmer have to change something in the data-store, only a change in the xml files are required. This way of designing a persistent layer provides you with a good tool to create large and more complex applications since the programmer doesn't have to bother himself with the persistence logic of the application. The downside is that it requires a great deal of time and effort to create a good robust persistence layer, and the application will also be slower because of the overhead that this design produces. In the next section different robust persistence approaches will be presented.

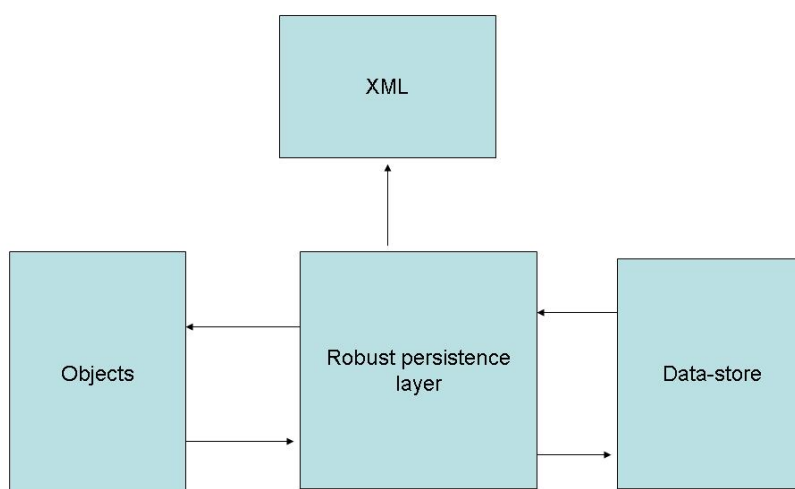


Figure 2: Third approach

3 Object Persistence and Approaches in Object oriented applications

[1-4] Object-oriented models have rapidly become the model of choice for programming most new computer applications. The most of applications have to deal with the persistent data, so the importance of persistence in data objects is increasing day by day. So, it is essential to add persistence to objects to make Object Oriented applications more useful than before.

There are three classes/approaches of solutions for implementing persistence in object oriented applications. First, the gateway-based object persistence approach, which involves adding object-oriented programming access to persistent data stored using traditional non-object-oriented data stores. Second, the object-relational database management system (DBMS) approach, which involves enhancing the extremely popular relational data model

by adding object-oriented modeling features, and third, the object-oriented DBMS approach (also called the persistent programming language approach), which involves adding persistence support to objects in an object-oriented programming language.

These above-mentioned three object persistence approaches are able to support certain classes of object-oriented applications, and each approach has been therefore affected by the requirements of the class of applications it supports.

In this section you will learn about,

- major characteristics and requirements of object-oriented applications
- how these characteristics and requirements may affect the method and choice of system in order to make the objects persistent in the application.
- user as well as programming interfaces which are provided by different tools for object oriented applications that create and manipulate persistence objects.
- choosing a particular mechanism in order to make objects persistent, including implementation requirements and limitations of above-mentioned approaches.
- how the object oriented applications(which are sharing same data) can interoperate to each other, and
- problems and solutions of how object oriented applications can live with non-object oriented applications that access the same data.

[3-4] In old ages, when there was no object orientation introduced, applications typically used relational database management system (DBMS) to store their persistent data and many of them still do so.

[3-4] Relational DBMS is typically developed to provide support for storing data used in traditional business applications such as banking transactions etc. The relational model is the basis of many commercial relational DBMS products which is now widely used for both retrieving and updating data. The relational model is simple and based on table format i.e. columns and rows for storing and viewing data. The data of basic types such as integer, string, and decimal, and other special types such as BLOB (binary large object) and CLOB (character large object) can be stored in a table. The use of standard query language in SQL makes it possible for applications to transparently access relational DBMS data from different vendors.

[4] If we compare the simple data model used by traditional business applications using a relational DBMS with object-oriented applications, we see the difference that object-oriented applications make extensive use of many new object-oriented features. e.g. user-extensible type system, encapsulation, inheritance, dynamic binding of methods, complex and composite objects and object identity.

[4] The data models supported by relational DBMS were offering limited services to handle the complex (object-oriented) business and non-business applications. So a lot of work has been done in designing and implementing systems to handle and manage the object's persistency.

3.1 Object Persistence Approaches

Three approaches are used today to handle the object persistence.

3.1.1 Gateway-based object persistence (GOP)

[4] This approach is a "middleware" approach in the sense that it is both application and data-independent. We can explain that 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. The most usage of this approach can be seen when programmers want to use existing non-object-oriented data stores but write applications using object-oriented programming models. 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. During the execution (at run 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.

Usage Information: [5] 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. The most important specification is CORBA (Common Object Request Broker Architecture), which defines the fundamental architecture for object interactions. The following specifications (which are adopted by OMG, Object Management Group) are directly related to object persistence. i.e. Persistent Object Service, Object Query Service, Object Relationships Service, Object Transaction Service, and Object Security Service.

Advantages:[4] It is a very good approach for

- 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, are best suited for using GOP systems.

Disadvantages/Drawbacks:[4] It is not good for

- 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.

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.

3.1.2 Object-relational DBMSs (ORDBMSs)

The ORDBMS approach is a bottom-up approach, being data (or database) centric. The relational model has become very important and successful in practice and the SQL is already a global standard. Object-relational DBMSs is used to add support for object-oriented data modeling by extending both the relational data model and the query language while keeping the already successful technology (especially the SQL) of a relational DBMS relatively intact.

Usage information 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 (or will be) built by extending existing relational DBMSs (e.g.: DB2, Informix, Oracle, and Sybase).

Advantages It is the best approach for

- 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.

3.2 Object-oriented DBMSs (OODBMSs)

The OODBMS approach is a top-down approach, being application (or programming language) centric. The basic principle to built OODBMS is to provide the best way to add persistence to objects to make them persistent that are used in an object-oriented programming language (OOPL) like C++ or Smalltalk. The OODBMSs are frequently referred to as persistent programming language systems because they have their roots in object-oriented programming languages. Object-oriented DBMSs are not only used to add persistence to object-oriented programming language but, historically, many object-oriented DBMSs were built to serve the market for computer-aided design/computer-aided manufacturing (CAD/CAM) applications for fast navigational access, versions, and long transactions.

Usage Information

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, Matisse). This approach is essentially a top-down approach, being application (or programming language) centric.

Advantages

It is the best approach for

- 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.
- the applications that do not have complex query and that are prepared to sacrifice some integrity and security for achieving good performance.

Disadvantages/Drawbacks

- 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.

3.3 Major characteristics and requirements of object-oriented applications and choice of each of the three approaches to object persistence

This section will discuss the major characteristics and requirements of object-oriented applications and how they affect the choice of each of the three approaches to object persistence. This section is further divided into three sub-topics discussed below:

3.3.1 Data Modeling

[4] In this section we will come up with various programming language features that are used in building object-oriented applications. There are number of 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

Table Data modeling [4]			
Feature	Gateway-Based Object Persistence (GOP)	Object-Relational Database Management System (ORDBMS)	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 <u>unencapsulated</u>)	Supported (but broken for queries)
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

Figure 3: Table 1: Data Modelling

issues arise in providing support for an object-oriented data model and the table above discuss those issues in detail.

For detailed information of above table, visit the website www.ibm.com.

3.3.2 Data access

[4][6] This section describes about how application objects can be created and stored. This section also describes about the support provided for navigational and ad hoc query types of access to persistent data and the interaction between client and server, particularly the method by which objects are communicated between client and server. At the end of this section, we will discuss some important application support items including schema evolution, integrity constraints etc. The table below gives the snapshot of all these details.

Table Data access [4]			
Feature	Gateway-Based Object Persistence (GOP)	Object-Relational Database Management System (ORDBMS)	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 (prefetching/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
Schema evolution	Limited support (complete support might be difficult to provide)	Supported	Supported
Integrity constraints and triggers	No support	Strongly supported	No support

Figure 4: Table 2: Data Access

The brief detail of above features is written below:-

Creating and accessing persistent data

[4] 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 nontransparent as these are done using a separate UPDATE statement. [5]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.

Navigation

[4] [8]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. 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.

Ad hoc query facility

[4][5]A system using GOP approach typically does not implement a new query language on the object representation. The query under GOP system works on the underlying data model that is not object-oriented and this does not work well with the application object model and create problems of impedance mismatching. [4]An ORDBMS supports queries in excellent way and handles well in most of the work in optimization and index management. In OODBMS, the support of query language is an extension of the object-oriented programming language. The OODBMS query languages do not support encapsulation and are allowed to access the structure of the data. This is unavoidable after the time when ad hoc queries required arbitrary computations on the data.

Object server versus page server

[4]In a 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 [5]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.

Schema evolution [4][5]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.

[4][5] 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. [4] ORDBMSs can provide strong support for schema evolution of table definitions. [4] In OODBMSs, the data model is complex so schema evolution in an OODBMS cannot be completely automated as in a relational DBMS.

Integrity constraints and triggers [4] 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.

3.3.3 Data sharing

In this section we describe the support 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 (see Table below).

Table Data sharing [4]			
Feature	Gateway-Based Object Persistence (GOP)	Object-Relational Database Management System (ORDBMS)	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 data store	Strongly supported	Limited support

ACID = atomicity, consistency, isolation, durability

Figure 5: Table 3: Data Sharing

ACID transactions

[4][7] 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).

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.

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.

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.

4 Object persistence in Java

As Java is an object oriented programming language, it is no surprise that a need for object persistence exists. Java has different ways to deal with object persistence depending if you are working against databases or not. Some of the object persistence techniques currently accessible in java will be presented below. There will only be an introduction to each technique, the important thing is that the reader know that these techniques exists. For a more detailed explanation the reader is encourage to read the references.

4.1 Serialization

Using serialization[12] in Java makes it easy to save object and object structures to files (and sockets). It works in such a way that you write your object to a stream, file stream or a socket stream. In Java this work can be done automatically which saves the programmer a lot of time. It can also be done in a more customizable way which can be specified by the programmer for more flexibility.

First of all, to make an object serializable in Java, you have to let the class which describes the object to add *implements Serializable* to its class definition.

```

class Employee implements Serializable {
    int empNr;
    Employee coworker;
    .
    .
}

```

No methods from the *Serializable* interface have to be implemented. Important to remember though, is that for a class to be serializable, all data fields defined in the class also have to be serializable. In the example above this is not a problem, since the data field *Employee coworker* is a reference to an instance of the same class, and by its definition the *Employee* class is serializable. All standard types like *int*, *string* and so on are already serializable by definition.

4.1.1 Write object

Now, to write an object to a stream, you have to use the class *ObjectOutputStream* which implements the interface *ObjectOutput*. *ObjectOutputStream* can not serialize an object by itself but have to be created on top of some other *OutputStream*, for example a file stream. Then all you have to do is to use the *writeObject()* method in *ObjectOutputStream* to write the object to the stream. Last we will flush and close the stream. Code for this is given below.

```

Employee emp;
FileOutputStream file = new FileOutputStream("filename.out");
ObjectOutput out = new ObjectOutputStream(file);
out.writeObject(emp);
out.flush();
out.close();

```

The *ObjectOutputStream* does not only write the specified object to the stream, but also all other objects that are referenced within the specified object. For example, in our *Employee* class, this means that the *Employee coworker* also will be written to the stream. This makes saving large data structures like a tree very easy, since every node in the tree also will be saved. Also, the *ObjectOutputStream* remembers the type of the object being written, so it can be read back from the stream correctly.

4.1.2 Read object

Now, to write an object to a stream without being able to read it back would be kind of pointless. And to read an object from a stream in Java you will have to use the *ObjectInputStream* class. As with *ObjectOutputStream*, *ObjectInputstream* must be created on top of another stream (file stream). Then we just use the *readObject()* method to read back our object.

```

FileInputStream file = new FileInputStream("filename.out");

```

```
ObjectInputStream in = new ObjectInputStream( file );
Employee e = (Employee) in.readObject();
```

For each object detected in the stream, a new one will be created in memory and will then get its values from the stream. This includes restoring references between objects in the stream. This means that our saved *Employee* object will still reference the *Employee coworker* object after it have been read back from the stream. We also have to typecast the object being returned from *readObject()* in order to use it as an *Employee* object. In this example we know what type of object that is to be returned. If you don't know you can ask the returned object about information about itself by calling the *getClass()* method.

4.1.3 Customizing streaming

You can customize the streaming of objects. You can for example write additional information to the stream. You can make some data fields transient, this means that those fields will not be written to the stream. You can also use the *ObjectInputValidation* interface to have error checking while reading objects from a stream.

4.1.4 Advantages / Disadvantages

Serialization in Java is very object friendly, you can easily save large structures of object data. Unfortunately it lacks the functionality of databases, for example to query the persistent objects.

4.2 Enterprise JavaBeans (EJB)

EJB is a technology for ease of creating and managing business objects on the server side of the application. When you work with EJB you will work with so called beans which are objects with special properties. There are two types of beans, entity beans and session beans. We will only be interested in the entity bean since entity beans are persistence capable, and session beans are not. EJB uses relational databases to handle the persistence functionality.

When you create an ordinary object in java it will only exist as long as the application is running. When you create an entity bean in EJB it will automatically be persistent. Each entity bean is encapsulated in a container which provides an interface for managing and manipulating the beans. This container works behind the scenes and stores, and retrieves information from and to the database. Typically, each kind of entity bean has its own table in the database, and each instance of that bean makes up a row in that table. Also, there are two types of object persistence in EJB, bean-managed persistence (BMP) and container-managed persistence (CMP).

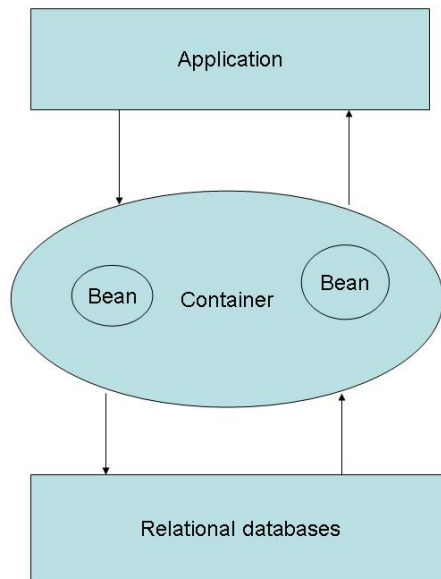


Figure 6: EJB Enviroment

4.2.1 Container managed persistence

When you use container-managed persistence you let the container handle all the communication to the database. You don't have to implement any database dependent code in your entity bean. Instead the information of how the bean interact with the database is written in the beans deployment descriptor[14]. Exactly what an deployment descriptor is and how it works is beyond the scope of this paper, lets just say for now that it is an xml file containing information of how to manage your entity bean, how to map the bean to the underlying data-store for example. Since you don't have any database specific calls in your entity bean, it will not be limited to any one type of storage mechanism. You can query the data-store by calling the functions defined in the bean.

4.2.2 Bean managed persistence

When you use bean managed persistence you move the responsibility to manage persistence from the container to the bean itself [15]. Bean managed persistence requires you to write your own database access calls for querying the database. Though this requires more work than using container-manage persistence, it will give you more control over how the entity bean access the database.

4.2.3 Advantages / Disadvantages

Entity beans presents you with a easy way to manage object persistence. When using container-managed persistence you don't require any sql knowledge and it also makes the

bean portable, all you have to do is to use different containers to be able to use different storage mechanism (different relational databases). And if you think that container-managed persistent is inflexible you can use bean-managed persistence to have more control over how the bean access the underlying data-store, but you will make the bean dependent on that data-store. The biggest downside is that a bean can't survive outside the container environment, this makes it only usable in the EJB environment.

4.3 Java Data Objects (JDO)

JDO is the newest addition to object persistence in java[16]. It is an API, independent of the underlying data store. You can use JDO with both relational databases, object databases and files. JDO works like a transparent interface between business logic and persistence logic and shields the user from data-store dependent behavior. For example, using a relational database as a data-store does not require you to know sql when using JDO. Instead JDO handles queries to the underlying data-store with the JDO query language (JDOQL) which is very java like, this will make it easy for programmers well educated in java to learn JDOQL. Since JDO acts like a transparent interface over different data-stores, JDO objects will be easily portable.

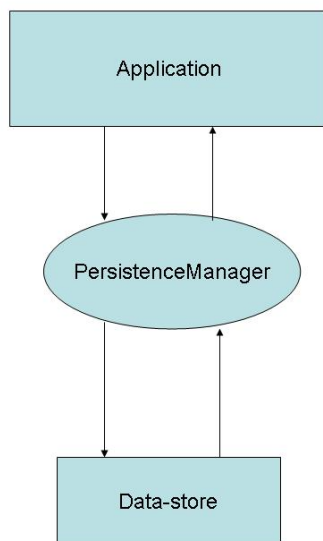


Figure 7: JDO Enviroment

When you create a persistent object in JDO you will create an ordinary object in java, the only requirement is that you include a no argument constructor. There is no persistent specific information in the class specification, no JDO specific information either for that matter. Instead, the persistence of an object in JDO is determined by an xml file (object

schema). It is this file that will contain the persistence specific information about the class and its persistent fields. You then have to link the xml file to the class, and you do this by making the class enhanced by running the JDO enhancer, which is a byte modifier. This approach, which separates JDO from the object, makes it possible to use objects outside the JDO environment. JDO have a *PersistenceMananger* object[17] that handle persistent instances. You will use the *PersistenceMananger* to create persistent instances (with JDO enhancer) and to handle the passing of information from and to the underlying data store. More information about JDO may be found by looking at the references.

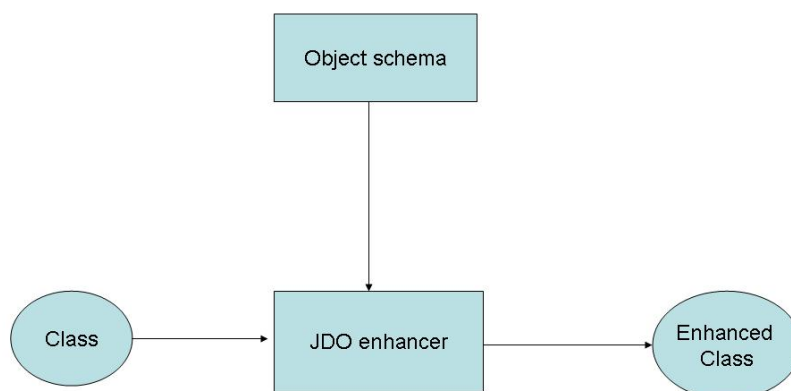


Figure 8: JDO Enhancer

4.3.1 Advantages / Disadvantages

As stated earlier, all JDO dependent code is separated from the object, which makes it possible to use that class outside the JDO environment. JDO supports different kinds of data-stores so portability is an easy task. Though the JDOQL provide a consistent language between different data-stores it is less flexible than for example ordinary sql code[18]. This may create a problem when you want to pass more complex queries to the database.

5 Object Persistence in .NET

.NET like Java has the need for object persistence. There are a lot of vendor specific frameworks out there that supports o/r mapping (object / relational) for .NET. Since .NET in its nature is very similar to Java it follows that many techniques in .NET is very

similar to those in Java. In fact, Serialization in Java and JDO both have equalities in .NET. Remember that only an introduction will be given, so the eager reader should follow the references for more information.

5.1 Serialization

As in Java, .NET provides you with a way to serialize your objects [19]. But in .NET there are two different approaches currently available. One is using the *XmlSerializer* class defined in the *System.Xml.Serialization* namespace. The other one is to use so called .Net formatters which is similar to the serialization process in Java. The *XmlSerializer* is the easier approach but is not as powerful as the .NET formatters.

5.1.1 XmlSerializer

When using the *XmlSerializer* the persistent fields of an object will be saved in an xml file. There are a few limitations and requirements when using *XmlSerilazer*. First, *XmlSerilazer* will only save public fields, this makes it impossible to recreate the state of an object that has private members. Second, *XmlSerializer* have limited support for core classes, like for example *HashTable*, which means that you cannot serialize these classes. To address this you can use the .NET formatters instead. Last, the class which you want to make serializable have to have a default constructor.

The thing you have to do to make a class persistent with *XmlSerializer* is to first create an *XmlSerializer* object and pass along what type of class you wish to make persistent in the constructor. Then, you create a stream, may it be a file stream or a stream to a buffer in memory, to which the object is to be saved. Then last but not least you will have to use the *Serialize()* method to save your object. The code below gives an example of how you can save an instance of class *Employee* to the file "file.xml".

```
Employee emp;  
XmlSerializer xs = new XmlSerializer(typeof(Employee));  
XmlTextWriter writer = new XmlTextWriter("file.xml", null);  
xs.Serialize(writer, emp);
```

To later retrieve the object from your xml file you just use the *Deserialize()* method.

```
Employee new_emp;  
new_emp = xs.Deserialize(writer);
```

Like serialization in java, *XmlSerializer* will save any other object referenced in the object that you serialize, which means that you will keep your data structure. There is also ways to customize *XmlSerializer*, you can for example change the format of the produced xml file or choose which fields in the class you don't wish to save.

5.1.2 .NET Formatters

There are two kinds of .NET Formatters currently available in .NET, one for binary streams and one for SOAP messages. .NET Formatters works very similar to serialization in java. To make a class serializable you have to mark the class with the *Serializable attribute*. And just like in java it is required that every referenced class in the object also is serializable. Also, like in Java, you can customize .NET Formatters, for example specify fields that you don't wish to be serialized and so on.

Because of the similarities with serialization in Java, only a short code example will be introduced to show how it might look like when you implement a *BinaryFormatter* in C#. The *BinaryFormatter* lies under the *System.Runtime.Serialization.Formatters.Binary* namespace.

```
[ Serializable ]  
public class Employee  
{ ... }
```

```
BinaryFormatter bf = new BinaryFormatter();  
FileStream fs = new FileStream("output.bin", FileMode.Create);  
bf.Serialize(fs, emp);  
fs.Close();
```

And later to deserialize the object you can write the following.

```
FileStream input = new FileStream("output.bin", FileMode.Open);  
Employee emp2 = (Employee) bf.Deserialize(input);  
input.Close();
```

5.1.3 Advantages / Disadvantages

XmlSerilazer provides you with an easy way to serialize objects, only a few lines of code and you have saved your object. But the *XmlSerialzer* is unfortunately restricted in its ways to perform serialization by not being able to serialize some classes. The binary formatter on the other hand, doesn't have that weakness, but is a little harder to implement. Also, no one of these methods provide the functionality of a database.

5.2 ObjectSpaces

You might say that ObjectSpaces is .NET equality to JDO in java. As with JDO, ObjectSpaces provide you with an interface independent of the underlying data store. It separates the persistent logic from the business logic, which results in that you only have to work with objects and not with complicated sql queries. Instead queries to the database are written in an object query language called OPath which use conventional C# operators (compare with JDOQL). ObjectSpaces provides support for xml data-store using XMLObjectSpace and sql server support with SqlObjectSpaces[20]. Currently, ObjectSpaces only

support relational databases, but support for object relational databases will probably be supported in the future.

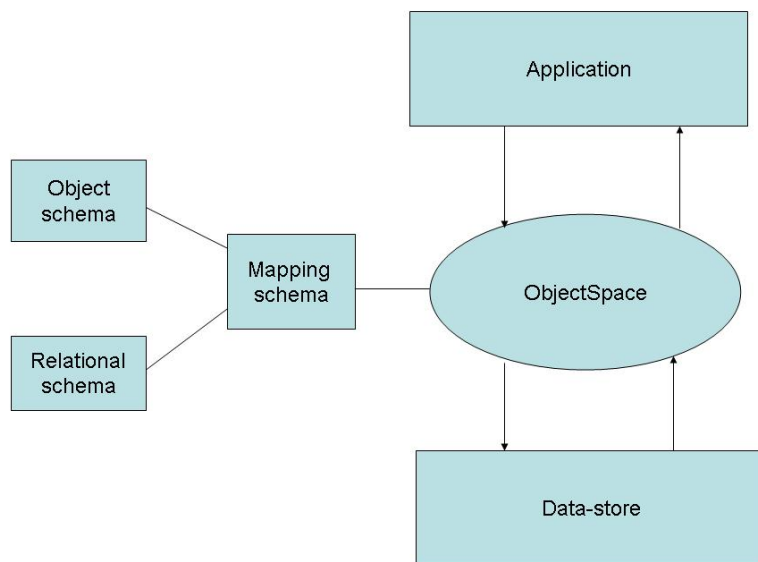


Figure 9: ObjectSpace Enviroment

Like in JDO, there is no ObjectSpace dependent code in the class itself, instead you define the behavior of the class in an object schema, which is an xml file[21]. You will also have to write an xml file describing the underlying data-store, the relational schema. For example if it is a relational data store, then this xml file will describe what tables exists and the fields in these tables. Last but not least you create a mapping schema that links these xml files together. As Java has *PersistenceFactory* objects, .NET has *ObjectSpace* objects. The functionality is the same, to handle the persistent objects and the communication to the underlying data-store.

5.2.1 Advantages / Disadvantages

Like all the other object-relational mapping frameworks out there ObjectSpaces let you think about the business logic instead of persistence logic, which will save you a lot of time. The setbacks is the same as with JDO that the query language supplied with ObjectSpaces isn't flexible enough which makes it hard to create complex sql queries.

6 Conclusion

Today's programming environments provide you with a wide variety of object persistence frameworks. This will greatly reduce development time and increase the quality of the application itself since the programmer doesn't need to bother himself with the persistence logic of the application. However, the frameworks available might not always accommodate your needs. Though there has been advancement in the field of object persistence there is still more to be done. This makes it also important to be able to design your own persistence layer that suits your specific needs. And depending on the size of your application, and the different environments it will work with, different design strategies are necessary.

[4-8]In the future, it is likely that we will see the continued presence of OODBMSs that address the needs of specialized markets, the continued prominence of ORDBMSs that address the needs of traditional commercial markets, and the growing importance and prevalence of the gateways-integrated with object query, object transaction and workflow, and object security (i.e., a full-function object middleware or multidatabase). It therefore becomes extremely important for an object-oriented application developer to choose the right type of system for storing objects. As evidenced by our discussion of the various issues, this task could be a fairly daunting one. We hope that our discussion provides valuable insight to developers to make it easier to choose the right persistent system for an object-oriented application.

References

- [1] C. Booch, *Object-Oriented Analysis*, Object-Oriented Analysis and Design with Applications, second edition, The Benjamin/Cummings Publishing Company, Redwood City, CA (1994)
- [2] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, NJ (1991).
- [3] J. Gosling and H. McGilton, "The Java Language Environment: A White Paper," Sun Microsystems (1994). http://www.javasoft.com/whitePaper/javawhitepaper_1.html
- [4] Kermin.c, *Object Persistence in Object-Oriented applications*, <http://www.research.ibm.com/journal/sj/361/sriniref.html>
- [5] *ObjectStore Software* www.persistence.com
- [6] Jörg Kienzle and Alexander Romanovsky, "A Framework Based on Design Patterns for Providing Persistence in Object-Oriented Programming Languages", submitted to JMLC 2000
- [7] Jörg Kienzle and Alexander Romanovsky, "Object Persistence: A Framework Based on Design Patterns", Poster A0, submitted to ECOOP 2000.
- [8] M. J. Carey, M. J. Franklin, M. Livny, and E. J. Shekita, "Data Caching Tradeoffs in Client-Server DBMS Architectures," Proceedings of the ACM SIGMOD Conference (1991).
- [9] A.Akhter, *Article: Research on Object Persistence*
- [10] Roos R., *Java Data Objects*, <http://www.solarmetric.com/images/JavaDataObjects-RobinRoos-1.0.pdf>
- [11] Scott W. Ambler, *Impedance Mismatch*, <http://www.agiledata.org/essays/impedanceMismatch.html>
- [12] Jim Coker, *Object Persistence and Distribution*, <http://java.sun.com/developer/technicalArticles/RMI/ObjectPersist/>
- [13] Ed Ort, *Ease of Development in Enterprise JavaBeans Technology*, <http://java.sun.com/developer/technicalArticles/ebeans/ejbease/#ejbcon>
- [14] Sun Microsystem, *What is an Entity Bean*, http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/EJBConcepts4.html#62950
- [15] , *Bean-Managed or Container-Managed Entity Beans*, http://help.sap.com/saphelp_nw04/helpdata/en/dc/05503e30a9d549e10000000a114084/content.htm

- [16] , *Java Data Objects (JDO)*, http://help.sap.com/saphelp_nw04/helpdata/en/91/7917fab0279f418e33762cbbfa6470/content.htm
- [17] Barry & Associates, *JDO PersistenceManager*, http://www.service-architecture.com/database/articles/jdo_persistence_manager.html
- [18] Lisa Oliver and Jeff Heaton, *Introduction to Java Data Objects (JDO)*, http://www.developer.com/db/article.php/10920_1580231_2
- [19] Dan Frumin, *Serialization in .NET*, <http://www.ondotnet.com/pub/a/dotnet/2004/01/26/serializationpt1.html>
- [20] Gopalan Suresh Raj, *Microsoft ObjectSpaces (OS)*, http://my.execpc.com/~gopalan/dotnet/object_spaces/object_spaces.html
- [21] Ryan Dawson, *ObjectSpaces*, <http://www.longhornblogs.com/rdawson/archive/2004/03/23/2822.aspx>
- [22] Scott W. Ambler, *The Design of a Robust Persistence Layer For Relational Databases*, <http://www.ambysoft.com/persistenceLayer.pdf>