Review of db4o from db4objects

Pascal Hauser
University of Applied Sciences Rapperswil, Switzerland
CH-8640 Rapperswil
pascal@hausers.info

Abstract

Db4o is a product from db4objects which is reviewed in this article. Db4objects claims that db4o can compete with traditional RDBMSs. Db4o cannot replace RDBMs, but it can be a good alternative in several fields of application. Before deciding if db4o should be used in a given scenario, one should look at a deeper level; what does db4o provide, where it can be used and where it can't. This article recapitulates what object databases are in general, and then covers some features of database systems with referring to db4o. The article explains what db4o is, what its features are and where it can be used.

1 Object Databases

Object databases have several benefits over other database systems when data is processed using an object oriented environment. This is the case, because "… an object-oriented database system is a database system which directly supports an object-oriented data model." [Kim1990]. Object database systems were introduced in the early 1990, to support the new model of object oriented programming better. But the predicted displacement of relational database technology has remained unfilled as they have had technical deficiencies. Today, the most used database systems are still relational database management systems (RDBMS).

By using an object database, an object – from object-oriented programming languages like Java, C# etc. – can be persisted easily. Of course – like on any other DBMS – there have to be methods for storing, updating, retrieving, querying and deleting objects. The advantage of object databases is that the persistence is transparent to the programmer, which is not the case when using relational DBMS. Historically seen, data counted to the most important values of a company. Relational DBMSs were available before object oriented methodologies and data was stored therefore in these DBMS first. As these Relational DBMSs did a great job – and still do! –, object databases had a difficult start for getting a significant market share in the DBMS market. But as Relational DBMSs were favored by people that are liable for data, the application developers using object oriented languages have a methodology mismatch. A lot of time is spent for writing mappings between the object schema and the database schema inside the Relational DBMS.

Other approaches have been tried, like Object-Relational DBMS (Oracle [Ora2005] for example) or Object-Relational Mapper software. An object relational DBMS is basically a Relational DBMS, but has a built-in functionality to have a mapping between the relational data model and the object model – were both models are inside the database. These objects can be used easier by the object oriented language (OO) developer. But this does not solve the problem, rather it moves it. Instead of the OO programmer, now the DBMS administrator could take care of the mapping between objects and relational data. But this approach is not a native support for OO. Still, a mapping between the OO objects and the OO part of the DBMS has to be done. This means SQL – or something alike – has to be executed. Even it is now simpler as the object structures are similarly, it is still a impedance mismatch. Using an Object Relation Mapper (like Hibernate [Hib08] for example) does not solve the problem either. It softens it, as the OO developer has less work to do, as instead of writing database access code he only has to write schema mappings.

The Object Management Group (OMG) is an industry consortium which develops enterprise integration standards [OMG]. As the name implies, most standards have something to do with objects. This is also based on history as OMG was founded in 1989, when several Object Oriented platforms were available but not interoperability. The OMG developed some well-known standards like the Unified Modeling Language (UML) [UML]. The Object Data Management Group is a subgroup of OMG. It was created to develop standards related to object data management. The first standard – ODMG 1.0 – was released in 1993. It contains a language for creating an object model inside an ODBMS and an object query language to query for objects in an ODBMS – like SQL is for RDBMS. The last release was ODMG 3.0 after which the ODMG was abandoned. In 2006, the OMG has decided [ODMGNG] to develop the "4th generation" standard for object databases. For this reason, the Object Database Technology Working Group (ODBT WG) was set up. The ODBT WG has not released a standard yet and there are no signs that something alike will happen in the near future. But they do a great job in centralize the discussion on object data persistence, where a lot of experts attend. As a side note, maybe there will not be any next object query language as thoughts currently tend to use something development language integrated instead of a stand-alone language. We will look at this further below, in the chapter about data querying.

Therefore the preferred way for an OO developer is still to have an object database. But everyone thinking about using an object database compares it with existing non-object DBMS – rightly, as the effective data is the valuable good that may not be jeopardized. As the company db4objects indirectly claims that its product db4o can compete as an object database with other DBMS [db4oBC05], this article analyses db4o and figures out for which of the statements holds true.

2 Comparison of Object Databases against other Types of DBMS

The previous part – Object Databases – already covered the differences for a developer using an object oriented programming language. This part shows what is different at a lower level, the data storage level. On a Relational DBMS, the main part consisted of defining a meta-data model. This consists of tables which are structured containers that hold the data. RDBMS were extended in early development to support securing the consistency of data. For this, one is able to define foreign key references from one table to another, check constraints etc. To work with the data, the SQL standard [ISO9075] was introduced. Most RDBMS were extended and support features like triggers, stored procedures and indexes, to only name a few. They are used for having data consistency – also meaning program logic – or better performance. Indexes are solely for performance. Triggers on the other side ensure data consistency or execute additional tasks when data changes. But this raises the question how much program logic should reside inside a DBMS. Stored Procedures can be used for both. Sometimes they contain some SQL that executes faster if executed through a stored procedure, on other uses, they contain massive amount of application logic and heavily work with the data inside the DBMS. For some DBMS – like Oracle – application logic was even preferably kept inside the DBMS. Another aspect – for all DBMS – is security. Operator of a DBMS wants to control access to the data.

Looking at Object-Relational DBMS, as stated before, are extended RDBMS. Normally, these are complete RDBMS which simply have some extension. One can introduce a new data structure – a meta-data model – and create a mapping between the new structure and the existing relational structure. Also, SQL remains intact, but is extended to operate object oriented alike.

Coming to Object DBMSs, things are working different. As the DBMS is – by intention – far less stand-alone, needs are different. By using OO languages, a developer normally intends to keep the whole application logic inside application and only use the DBMS for 'simple' persistence storage of data. Therefore Stored Procedures, Triggers etc. are not as needed as in

RDBMS. One might ask why there should be a management system at all. But there is surely a need, as database systems can contain a lot of data, and having a lot of data is about storage management. If data should be consistent, available to multiple applications, manageable independent of a program – also meaning direct access to all of the data inside the object database –, replicating data transparent to the application and having another level of access control to the store, the object database has still to be managed and therefore leads to the assumption that even for object databases there should still be an object database management system.

3 Architecture of db4o

Unlike most other DBMS, db4o is not built as a server system but as a library. This allows for some important features of db4o, like a very small memory footprint [db4oOSODB] – which makes, by the way, db4o very useable on mobile devices. On the other side, db4o is not an object database management system but an object database. This does not allow db4o to compete with DBMS on traditional fields of DBMSs. But it is usable on fields where 'heavy' DBMS are not usable, and even in some scenarios where (R)DBMS were used traditionally, db4o is an alternative. Details are explained later in the fields of application section. As db4o can't be directly compared with another DBMS, we further describe db4o with only referencing to DBMSs on specific topics, where appropriate.

3.1. Tiers

Db4o consists of one or two tiers. Its primary intention is to be used as a one tier application. This is due to the fact that it is a library which is embedded into another application, meaning the amount of tiers is given by the application as db4o does not introduces a new tier. But the db4o architecture is layered into a server and a client part. The server part can also be run in its own process space, where it can listen for network connections for communication. This gives a typical Client/Server scenario, unless the application introduces itself as an intermediate tier – for example if the application logic, including accessing db4o, is not the client itself but exposes itself through a web service. When db4o is run as separate server, there has to be – still – an application which initializes and starts the db4o server. But this is simple. Also simple are the functions, the server does. It takes and executes queries and returns the objects. It is only a network layer between; the rest remains the same as using a process embedded server. Additionally there exists an out-of-band signaling [db4oOOBS] to – for example – stop the server. But everything else can be signaled as this is not implemented by the db4o server part but by the surrounding application that starts the server.

3.2. Caching

If an object is retrieved from the data store, an object of the target class is instantiated and filled with the data from the store. This process is also known as activation. An activated object may have references to other objects. Either the referenced objects have also to be loaded, or instead of referencing another object directly, an intermediate object is used, which can return the target object. This is done by not containing a direct reference to the object but supporting lazy loading the object from the data store. Both approaches are supported by db4o. This, as db4o implemented some custom collections which support lazy loading. But if two objects of any type reference the same – a third object –, where these two objects can also be queried by two different queries – but in the same transaction – the object database has to return the identical third object as a reference on the two objects. Accessing the third object through the first and changing a value should also let the second object to see the changed value. To support this, the object database has to know, which objects were already activated and keep references to them. This is caching, as db4o needs to have a reference to every object, so it can use an already activated object, when needed. To allow this, each object needs to be uniquely identifiable.

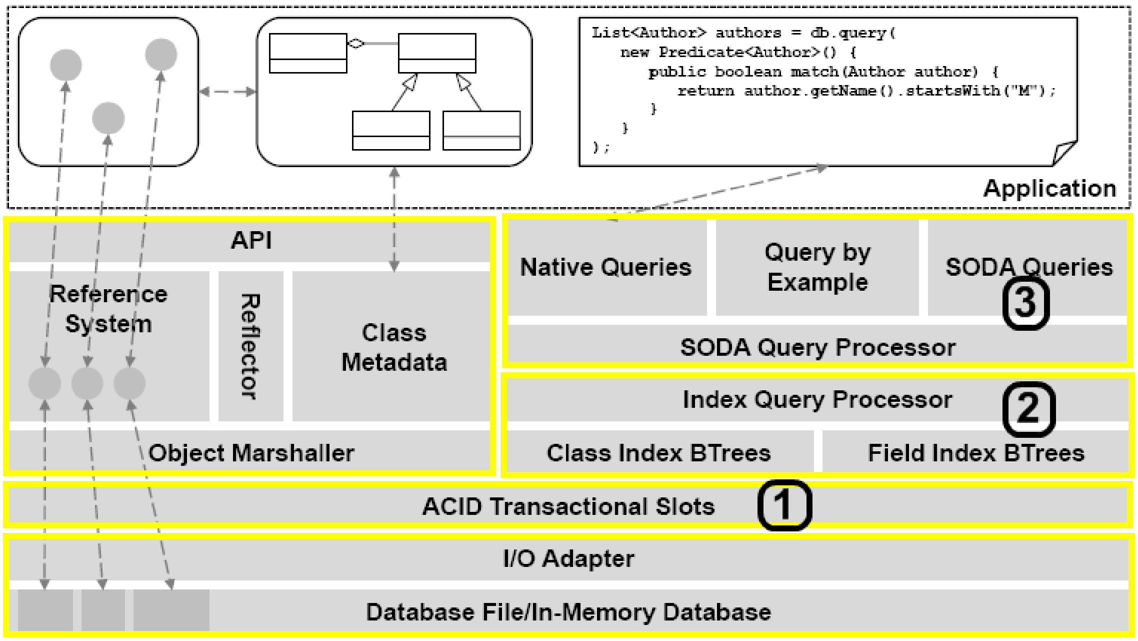
Db4o has implemented its own caching algorithm. Cache entries are based on a hash code. The cache itself is organized as an efficient tree structure – as well are the indexes, by the way, that are built as fast B-Trees. Analyzing the details of the rather complex cache implementation is out of scope of this document. A description of how the cache is working was, unfortunately, not found on the db4o website.

3.3. Security

As db4o is only an object database and not an object database management system, it is not standalone which means that everything goes through an application. Therefore the application can fully control authentication and authorization as needed. On the level of db4o – in embedded mode – there are no integrated security mechanisms. There is no user management or something else. This allows db4o to be very flexible, from using no authentication at all to all possible authentication mechanisms, without having an underlying db mechanism which has to be mapped. But, of course, even for simple applications, the developer is asked. Every needed authentication or authorization has to be implemented by the application developer. The data itself is stored in a file. Access to the file should be controlled by using the mechanisms of the underlying platform. This holds true for every database system.

In Client/Server mode of db4o, instead of running only in-process, the db4o server part listens for network connections. By using this approach, one can specify different user/password credentials which are allowed to access the server. The client part has then to specify username and password to be able to connect to the server. This is the only implemented security mechanism and therefore only authentication is implemented and not authorization – although there exists a minimal authorization capability, as one can switch files where objects can be stored in different files and credentials can be granted based on a file level, this seems to be a very limited approach and is therefore not considered.

4 Typical Features



```
List<Author> authors = db.query(
   new Predicate<Author>() {
      public boolean match(Author author) {
         return author.getName().startsWith("M");
      }
   }
);
```

After we looked at the general architecture of db4o, we now look a bit in more detail at some components of db4o. The Figure above shows an overview of the db4o architecture.

4.1. ACID

One of the most central concepts is the ACID (Atomicity, Consistency, Isolation and Durability) model. Without it, database reliability would not be possible. Db4o supports the ACID model [db4oACID]. This is done by fore steps. First, all changes are written as a change set to the database – the real objects are not yet affected. If any error occurs before the next step, the change set is simply discarded. The data remains consistent. As a next step, the database is set in commit-mode. After this change, the changes can no longer be discarded, meaning all changes will be written to the objects. In the third step, the changes are written to the objects and logged. In case of an error, on recovery the remaining changes are applied. At last, the database is set back to normal-mode. Now, new commits can take place. Therefore, A and D are fulfilled as everything is changed or nothing, and changes are durable as saved to the storage. Consistency should be checked by db4o, which is also true for isolation. The later statements were taken from the db4o documentation and not verified by the author.

Db4o supports the four levels of isolation (Read-Uncommitted, Read-Committed, Repeatable-Read, Serializable), but with some unexpected behavior. At the default level, read-committed, changes from other transactions will affect the running transaction – meaning changes will be visible. But this is only true for objects not yet cached. This can lead to a inconsistent view of the data. Therefore one should try to imagine what can where happen, and take a look at the refresh mechanism provided by db4o.

4.2. Concurrency

Most RDBMS supports concurrency by using some built-in features. Most scenarios can be supported by using transactions and transaction isolation levels. By using transaction isolation levels, the DBMS can lock rows and block other transactions. To get the desired behavior, an application can choose between four isolation levels – Read Uncommitted, Read Commited, Repeatable Read and Serializable. Along with isolation levels, an application has normally also the possibility to use optimistic locking [Kun1981]. By using optimistic locking, one assumes that no change is made since retrieving data and storing updated data back to the data store. But as changes can happen, before actually saving, checks are made, to ensure that no changes have taken place. This can be done, for example, by having a version attribute on each data object, which is incremented on each change. Before saving, the retrieved version is compared to the current version on the data. If the versions are equal, no change took place in the mean time and the changed data can be persisted. To allow this, the DBMS has to support the serializable isolation level or another kind of locking to make sure that an application can retrieve the current version of the data and – if it matches with the earlier retrieved version – save the updated data with being sure that between the retrieval of the version in the store and the save, no other save takes place. The versioning can either be self-made or by using support features some DBMS supports. For example SQL Server [MSSQL2008] provides timestamp and rowversion as special data types for this scenario – as a side note, these data types are also used for replication to detect changes in data on synchronizing the data, as on replication a simple number increment is not sufficient as if two systems replication have each update the data once they both have the same incremented number both different data.

Db4o supports optimistic and pessimistic locking only indirectly. The db4o documentation [db4oConCur] specifies a possible solution; using the locking mechanism of the programming language. But this is mainly a workaround. The solution suggests, before using an object, one can acquire a semaphore and releasing it after using the object. With this solution we simply have one semaphore for each object in the database. To use the same semaphore at all parallel executions, db4o provides a semaphore store. To get the right semaphore, they are simply named by the id of an object, which db4o provides – but the naming is left to the developer.

Along with an ID, each db4o object also has a version which can be used to do optimistic concurrency.

4.3. Indexing

Indexing is a central concept for fast searching over stored data. Without indexes, searching for data with a specific value in an attribute forces the data base to read all data – of the type being searched – and check if the attribute of each data object matches the value being searched. This is – of course – true for RDBMS as well as for object databases. An index is a data structure which stores the data of an attribute in an ordered fashion with a reference to the real data object. Now when the indexed attribute is searched, instead of going through all data – the index is used. As the index is ordered, the matching data objects can be identified much faster as lookup over the index can be done in an optimized way – for example the index can be a data structure of a tree type, which leads to – approximately, without proof – only N(log(n)) comparisons. Of course, indexes are not free, meaning in this case, instead of computing and I/O power, more storage space is used.

Db4o also supports indexes [db4oIndexes]. These can be specified in a configuration file. This might sound uncommon, but this is due the fact that db4o is not a management system. Looking at RDBMS, some performance optimization is normally done in advance, during development. Most RDBMSs also support some kind of profiling. These tools are used when performance problems exist. Through profiling one can detect where the problem is located and then take counter measures – for example creating a new index, on the fly. On db4o, one has to change the configuration file and restart the application. This has an impact for the start-up time of the application, as indexes are built, when the data base is opened the next time. But for the fields where db4o is in used, this is normally not a drawback.

Beside that db4o does not support some extended performance mechanisms like full-text indexing, the indexing capabilities of db4o are also limited to the above described case. Using other DBMSs, there are possibilities to optimize the indexes. For example on huge databases, one can specify, where the index should be stored. Also, normally, fill factors can be specified, which influence the behavior when an index actualized when new data arrives. This is very useful if one know the data change behavior – meaning how frequent data is changed or inserted and in which amounts. Most RDBMS supports the concept of a view, which aggregates data from multiple tables. Some RDBMS also support building indexes of views. This allows for fast lookup of data from a user point of view, which can go over several types of data. Getting the same functionality in db4o would mean using the most appropriate index on a root object and then navigating through the object graph, or loading objects by its kind each with an index and create an object type to type map to get the desired data. This is – of course – much slower.

4.4. Data Querying

Data that was stored inside a database would be of no use, if there were not a way to read the data out of the database. This would also be possible without a query language. Like a file that could be read until the information of interest is found. Of course, this would be much too slow for most scenarios. Performance is one reason for a query language. On RDBMS, the most widely used standard is SQL [ISO9075] which stands for standard query language. Although, SQL has been heavily extended in the meanwhile, its original intention was to query for data. Querying for data – with SQL – is done in two dimensions. Selecting which data – which data objects, called tupels on RDBMS – and what of the data – which parts of a data object, called attributes on RDBMS – should be retrieved.

As stated above, most RDBMS support SQL for data querying. Beside of querying – with SQL –, it’s also possible to modify the data schema – through the use of the data modification language (DDL) part of the SQL specification. Most RDBMS implementers extend their implementation SQL with vendor specific operators. With these extensions, everything is possible. Mostly, these extensions are used for controlling the server or using some extended features – for example Oracle Spatial [OraSpa].

For object databases, it would not make much sense to use SQL as a primary query language. As data of objects are also structured, implementing an SQL parser would, however, be possible. For the developer of an application, the preferred way would be to directly query by using the object structure. By using SQL to query data stored in a RDBMS, this is not possible as the data structure is different to the object structure. But if an object-relational mapper like Hibernate [Hib08] is used, querying by using the object structure is possible again. As a side note, on .NET [NET] it is possible to query for objects by using LINQ [LINQ], an integrated – integrated into the programming language – query language. It is also possible to query against an RDBMS, but there has to be a mapping to the database data schema. In contrast to SQL, the ODMG developed an object query language (OQL). As the ODMG – as stated above – was abandoned in 2001, the OQL is not of relevance nowadays. For the next generation, the experts in the circle of the ODBT WG are thinking – instead of an OQL – more of using something development language integrated like LINQ.

Db4o supports multiple possibilities to query for data objects. The easiest for the developer, but also the most limited, is Query-By-Example [db4oQBE]. On this method, one creates a new object, called a template, and fills the attributes on which should be searched. Db4o returns all objects from the data store which have the attributes set to the same value as the template object. The normal – in db4o probably mostly used – way is to use native queries [db4oNQ]. On this method a developer can start a search and specify a callback method, which is – logically – executed for every object. The callback method can do whatever it likes and at the end return true of false, whether the object matches or not. This is an unlimited way and therefore leaves no wishes open to the developer. But, as one can imagine, this may not optimally from a performance point of view. In fact, db4o tries to understand what the method does and use an internal optimized way to retrieve the objects. But if that is not possible – due to complexity or dynamic behavior – db4o simply executes the method for every object, which is, of course, slow. Also a native query may be not optimal for db4o to process. But this is the same with SQL interpreters. For SQL, there exists also multiple ways to execute complex queries which have different run length behavior. Another way is to use SODA [db4oSODA]. SODA is a low-level API which allows direct access to the nodes of a query graph. This is the query method with the best performance. But one has to use string identifiers instead of objects, which is no longer type-safe and neither compile-time checked. Recently, a forth method came into place – LINQ support. Of course, this method is only available to the .NET users of db4o and not to the users of the Java alternative. This method, as stated above, allows querying by using the object structure. But there are limitations by the set of instructions supported by LINQ. Therefore this alternative is not as flexible as the native query approach. But it performs normally well.

Summarizing, db4o offers multiple methods for querying for object. One should be able to find suitable method for a given scenario. But – in contrary to SQL – all methods require an application, which needs programming. There is no general query language like SQL which can be used for data analyzing or by other applications. This does not allow db4o to be very flexible in a heterogeneous environment.

4.5. Replication

Replication – in the context of database systems – is a term meaning the transfer of data and metadata from one database to another, including the synchronization to maintain consistency. Replication is used for different reasons among it data availability – multiple servers so that if one fails, system as a whole is still running – and performance – by having multiple servers for serving clients, cope with small bandwidth lines by distributing data to servers at different locations to serve clients locally – or data distribution – like synchronizing part of the data to mobile devices.

Db4o comes with a replication system component called ‘db4o Replication System (dRS)’. It supports the replication between two db4o systems. Replication between a db4o and another database system is also possible through the use of Hibernate, but this is limited to the Java version of db4o. Therefore, one has to keep in mind, that if using the .Net version, dRS cannot be used to synchronize to relational DBMSs.

Benefits of the dRS are its simplicity in use. Replication can be done by only few lines of code. But the replication cannot be done administratively, there needs to be someone that programs an application that does the replication. This is contrary to most RDBMS, where administrators manage servers and replication between them.

5 Fields of application for Object Databases in general and for db4o in specific

Looking at what db4o supports, it’s definitively not a replacement for a RDBMS, where they are typically used. For example, having a high-availability solution or keeping databases in a data center is not possible with db4o. But with features like the OO approach, the small memory footprint, zero administration etc, db4o it is a product one should certainly considerate to use on scenarios where a RDBMS may not be an ideal solution. This is, for example, the case for the industry solutions db4o states [db4oSOL]. These are applications that need to store data, but are more focused on the functionality of the application then on having full stand-alone flexibility with the data. This does not mean that data is not important, rather that data will be used in a predefined way and nor afterwards for dynamic data analysis etc. It is, as an example, ideal for mobile, disconnected or embedded devices. This is the case for mobile handheld applications or for industrial device solutions.

6 Analysis of – according to db4objects – unique features of object databases that db4o supports

6.1. Handling of Changes in Object Versions

As systems evolve, so do data structures of a system. Using an application with an RDBMS leads to having two structures, the object structure and the data structure. Both structures need to be synchronized. When the objects change, the database has normally also to be changed and a new application version cannot work with an old DB structure version and vice versa. On object databases this is different. One has the objects as the only structure and the database system has to take care of the reminding. This leads to other methods, as the data is – looking at an object database and not on an object database management system – no longer stand-alone. One case – for giving an example – is what happens if an attribute is being renamed. The data should definitively not be lost. But the system does not know how the structure changed. On RDBMS, one can use SQL to create the new schema and transform data as needed. On an object database, it depends of the implementation. Db4o checks on each class if it implements an on activate method. This process is described in [db4oCOSP]. If that is the case, the method is executed for every object instantiated from the store. In this method, on can check the actual object and run update logic. As removed attributes are not directly available in the object, db4o supports retrieval of this information through other – alternative to accessing data through the object – access methods. Alternatively, one can still introduce an object type with only one instance in the store to hold the db version. Using this object, one can determine the current version and write an external – to the application – update application which checks the version and executes step by step update applications, where each has the next version object model in it. Through this method, one has the complete object model available to write update ‘scripts’.

6.2. Agile Techniques for Object Databases

Every developer who had to develop an application that uses an RDBMS and had to maintain the data structures in the RDBMS knows, how the impedance mismatch feels and how time consuming it is. Also when developing in a team, the local database is quickly out of date and the application can maybe no longer be tested. This is, because a standalone heavy RDBMS does not fit with agile methodologies. Here, object databases fit – of course – much better into the scope. Here we will state some examples borrowed from the db4objects documentation [db4oATODB]. Refactoring is such a case. Renaming an attribute of a class is a simple task. One can imagine what this would case by using an RDBMS. Using the object database, when using refactoring, all code is updated and almost nothing needs to be done. The object database contains no structure which needs to be updated. On adding new test data during development, db4o updates its structure itself. This causes no collision with other developers. Only if the application is already used in production environments, an update method has to implemented, so that no data is lost. Another benefit is that a developer only has to have the source code. No instance of a DBMS and no DBMS management tools. Everything needed is in the source code repository, which – of course – makes things easier.

7 Conclusion

Db4o is a product that realizes the OO approach in a straight way. With knowing what it is intended for, where can it be used and with which difficulties one has to deal, developing applications – using db4o – is easy and really can cut-down development cost and time to market, as promised by db4objects. Db4o does not support many non-basic features, but what is supported is implemented in the right way, heading to be an optimal solution for where it is intended for.

8 References

[Kim1990]	Kim 1990, Object-Oriented Databases: Definition and Research Directions.
[OMG]	Object Management Group, 11.2008 http://www.omg.org
[UML]	Unified Modeling Language, 11.2008 http://www.omg.org/spec/UML/
[ODMG]	Object Data Management Group, 11.2008 http://www.odbms.org/odmg.html
[ODMGNG]	Next Generation Object Standard, 11.2008 http://www.odbms.org/odmg_ng.html
[Ora05]	Oracle Database Application Developer's Guide - Object-Relational Features, 2005 http://download.oracle.com/docs/cd/B19306_01/appdev.102/b14260/toc.htm
[Hib08]	Hibernate, 11.2008: http://www.hibernate.org/
[db4oBC05]	db4objects 2005, The Business Case for the Open Source Object Database http://www.db4o.com/about/company/backgrounder/db4objects%20Business%20Backgrounder%20November%202005.pdf
[ISO9075]	ISO Standard 9075, Information technology - Database languages – SQL
[db4oOSODB]	db4objects 11.2008, db4o Open Source Object Database http://www.db4o.com/about/productinformation/db4o%20Product%20Information%20V7.0.pdf
[db4oACID]	db4objects 11.2008, ACID Properties for db4o http://developer.db4o.com/Resources/view.aspx/Reference/Basic_Concepts/ACID_Model/ACID_Properties_For_Db4o

[Kun1981]	Kung and Robinson 1981, On optimistic methods for concurrency control
[MSSQL2008]	Microsoft 11.2008, Microsoft SQL Server 2008 http://www.microsoft.com/sqlserver/2008/en/us/default.aspx
[db4oConCur]	db4objects 11.2008, Concurrency Control http://developer.db4o.com/Resources/view.aspx/Reference/Client-Server/Concurrency_Control
[db4oIndexes]	db4objects 11.2008, Indexing http://developer.db4o.com/Resources/view.aspx/Reference/Tuning/Indexing
[OraSpa]	Oracle 11.2008, Oracle Spatial & Oracle Location http://www.oracle.com/technology/products/spatial/index.html
[LINQ]	Microsoft 11.2008, The LINQ Project http://msdn.microsoft.com/en-us/netframework/aa904594.aspx
[.NET]	Microsoft 11.2008, Microsoft .NET Framework http://www.microsoft.com/net/
[LINQ2SQL]	Microsoft 11.2008, LINQ to SQL: .NET Language-Integrated Query for Relational Data http://msdn.microsoft.com/en-us/library/bb425822.aspx
[db4oQBE]	db4objects 11.2008, Query By Example http://developer.db4o.com/Resources/view.aspx/Reference/Object_Lifecycle/Querying/Query_By_Example
[db4oNQ]	db4objects 11.2008, Native Queries http://developer.db4o.com/Resources/view.aspx/Reference/Object_Lifecycle/Querying/Native_Queries
[db4oSODA]	db4objects 11.2008, SODA Query http://developer.db4o.com/Resources/view.aspx/Reference/Object_Lifecycle/Querying/SODA_Query
[db4oCOSP]	db4objects 11.2008, Complex Object Structures, Persistence, and db4o http://www.db4o.com/about/productinformation/whitepapers/db4o%20Whitepaper%20-%20Complex%20Object%20Structures.pdf
[db4oOOBS]	db4objects 11.2008, Out-of-band Signalling http://developer.db4o.com/Resources/view.aspx/Reference/Client-Server/Networked/Out-of-band_Signalling
[db4oSOL]	db4objects 11.2008, Industry Solutions :: Introduction http://www.db4o.com/about/solutions/
[Opd1979]	W.F. Opdyke 1979, Refactoring Object-Oriented Frameworks
[db4oATODB]	db4objects 11.2008, Agile Techniques for Object Databases http://www.db4o.com/about/productinformation/whitepapers/db4o%20Whitepaper%20-%20Agile%20Techniques%20for%20Object%20Databases.pdf