



RESEARCH REPORT

TORPEDO: A Testbed of Object-Relational Products for Enterprise Distributed Objects

Uncovering Database Access Optimization Strategies

<http://www.middlewareRESEARCH.com>

**The Middleware Company
Research Team**

RESEARCH LEAD Bruce Martin, Ph.D.
August 23, 2004

research@middleware-company.com

1 DISCLOSURES

1.1 Code of Conduct

The Middleware Company offers the world's leading knowledge network for middleware professionals. The Middleware Company operates communities, sells consulting and conducts research. As a research organization, The Middleware Company is dedicated to producing independent intelligence about techniques, technologies, products and practices in the middleware industry. Our goal is to provide practical information to aid technical decision making.

- Our research is credible. We publish only what we believe and can stand behind.
- Our research is honest. To the greatest extent allowable by law we publish the parameters, methodology and artifacts of a research endeavor. Where the research adheres to a specification, we publish that specification. Where the research produces source code, we publish the code for inspection. Where it produces quantitative results, we fully explain how they were produced and calculated.
- Our research is community-based. Where possible, we engage the community and relevant experts for participation, feedback, and validation.

If the research is sponsored, we give the sponsor the opportunity to prevent publication if they deem that publishing the results would harm them. This policy allows us to preserve our research integrity, and simultaneously creates incentives for organizations to sponsor creative experiments as opposed to scenarios they can "win."

This Code of Conduct applies to all research conducted and authored by The Middleware Company, and is reproduced in all our research reports. It does not apply to research products conducted by other organizations that we may publish or mention because we consider them of interest to the community.

1.2 Disclosures

The Middleware Company is an independently operating but wholly owned subsidiary of Veritas Software Corporation (www.veritas.com, NASDAQ: VRTSE).

It is possible and likely that Veritas may have business relationships with vendors mentioned in this report or with the vendors of products used in the compilation of this report.

2 TABLE OF CONTENTS

1	DISCLOSURES	2
1.1	Code of Conduct	2
1.2	Disclosures	2
2	TABLE OF CONTENTS	3
3	INTRODUCTION	4
3.1	The Promises of O-R Mapping Software	4
3.2	TORPEDO	5
4	TORPEDO DEFINED.....	5
4.1	Single Server, Multi-tiered Application Model	6
4.2	Clustered Multi-tiered Application Model	7
4.3	Database Access Optimization Strategies	8
4.3.1	Read only objects	8
4.3.2	Caching within a transaction	8
4.3.3	Caching across transactions.....	8
4.3.4	Partial load of object data	8
4.3.5	Bulk loads of multiple objects	8
4.3.6	Lazy loads of multiple objects	8
4.3.7	Bulk loads of related objects	8
4.3.8	Lazy loads of related objects	8
4.3.9	Batch update operations	9
4.4	The Auction Application	9
4.4.1	Object model.....	11
4.4.2	Auction Service Operations	11
4.4.2.1	listAuction	11
4.4.2.2	listAuctionTwiceWithTransaction	12
4.4.2.3	listAuctionTwiceWithoutTransaction	12
4.4.2.4	listPartialAuction	13
4.4.2.5	findAllAuctions	13
4.4.2.6	findHighBids	13
4.4.2.7	placeBid	14
4.4.2.8	Place2bids	14
4.4.3	Running TORPEDO Tests.....	14
4.4.4	Correctness.....	15
5	THE TORPEDO REFERENCE IMPLEMENTATION.....	15
5.1.1	Transactions	16
5.1.2	Database.....	16
6	COMPLIANT TORPEDO IMPLEMENTATIONS.....	17
7	REPORTING TORPEDO RESULTS.....	18
8	RELATED WORK	20
8.1	Object Relational Database Benchmarks	20
8.2	Object Oriented Database Benchmarks	20
8.3	RUBiS	20
9	ACKNOWLEDGMENTS.....	20
10	REFERENCES	20

Abstract

A popular architecture for enterprise applications is one of a stateless object-based server accessing persistent data through Object-Relational mapping software. The reported benefits of using Object-Relational mapping software are increased developer productivity, greater database portability and improved runtime performance over hand-written SQL due to caching. In spite of these supposed benefits, many software architects are suspicious of the "black box" nature of O-R mapping software. Discerning how O-R mapping software actually accesses a database is difficult.

The Testbed of Object Relational Products for Enterprise Distributed Objects (TORPEDO) is designed to reveal the sophistication of O-R mapping software in accessing databases in single server and clustered environments. TORPEDO defines a set of realistic application level operations that detect a significant set of database access optimizations. TORPEDO supports two standard Java APIs for O-R mapping, namely, Container Managed Persistence (CMP 2.x) and Java Data Objects (JDO). TORPEDO also supports Oracle's Toplink and Hibernate. There are dozens of commercial and open-source O-R mapping products supporting these APIs. Results from running TORPEDO on different O-R mapping systems are comparable.

The Middleware Company has made a reference implementation of TORPEDO available. Other TORPEDO implementations are possible. This document contains the functional and behavioral specification that all TORPEDO implementations must conform to.

3 INTRODUCTION

Object-oriented application design methodologies have become mainstream. Application designers use design and development tools that support UML-style decomposition of complex problems. One of the basic UML activities is the identification of objects and the relationships between them. Given this successful approach to modeling complex problems, the application developer then develops software that implements the identified objects and relationships.

Often, the identified objects are *persistent*. As such, they need to be represented in a database management system. If the database management system is an object-oriented database [AABDMZ89] [NP91] or an object-relational database [Stone97] [Stone99], then the representation of the programming language object is straightforward. The object concepts in the object-oriented programming language have direct counterparts in the database.

But almost all enterprises use relational databases. Relational tables cannot represent persistent objects directly. Instead, *object-relational mapping software* gives the object-oriented application the appearance of accessing persistent objects. Rather than formulating SQL relational queries to retrieve and update database values, the application software retrieves collections of objects using an object-oriented query language and accesses the resulting persistent objects via their methods.

3.1 The Promises of O-R Mapping Software

O-R mapping software has the potential to improve the productivity of application developers since they are relieved from producing and maintaining database code. Furthermore, it implements a layer of abstraction between the application and the database, greatly easing the task of porting an application from one database management system to another.

O-R mapping software also has the potential to improve the runtime performance of enterprise applications due to the caching it provides. The O-R mapping software can be viewed as an object-oriented cache of the relational database system. The O-R mapping software populates and manages the cache. If engineered properly, such a cache provides an opportunity for significant performance improvement for accessing data over SQL queries.

In spite of these benefits, software architects are often leery of using O-R mapping software. They often prefer to design and implement their own persistence layer in order to have control over exactly how and when the database is accessed. They are suspicious of the “black box” nature of O-R mapping software. Without considerable effort, it is hard to discern how O-R mapping software actually accesses the database. Furthermore, it is hard to compare products.

3.2 TORPEDO

The Testbed of Object Relational Products for Enterprise Distributed Objects (TORPEDO) is designed to reveal the sophistication of O-R mapping software in accessing relational databases and maintaining its cache. TORPEDO defines a standard set of realistic application level operations that were carefully selected to detect a significant set of database access optimizations. Furthermore, results from TORPEDO on different O-R mapping systems are comparable.

By its nature, O-R mapping software is integrated with an object-oriented programming language. The Java programming language has become extremely popular for enterprise applications – exactly those applications that make heavy use of relational databases. As such, the reference implementation of TORPEDO is targeted to use Java. Other implementations of TORPEDO for other object-oriented programming languages could be created. Results from those implementations would be comparable with results from the reference implementation. In section 6 we define the rules for developing alternative TORPEDO implementations.

There are *two* standard APIs in Java for O-R Mapping [CMP] [JDO]. The TORPEDO reference implementation supports both APIs. In particular, TORPEDO supports Container Managed Persistence (CMP 2.0) and Java Data Objects (JDO). CMP maps between Enterprise Java Entity Beans and relational databases and JDO maps between Java objects and relational databases.

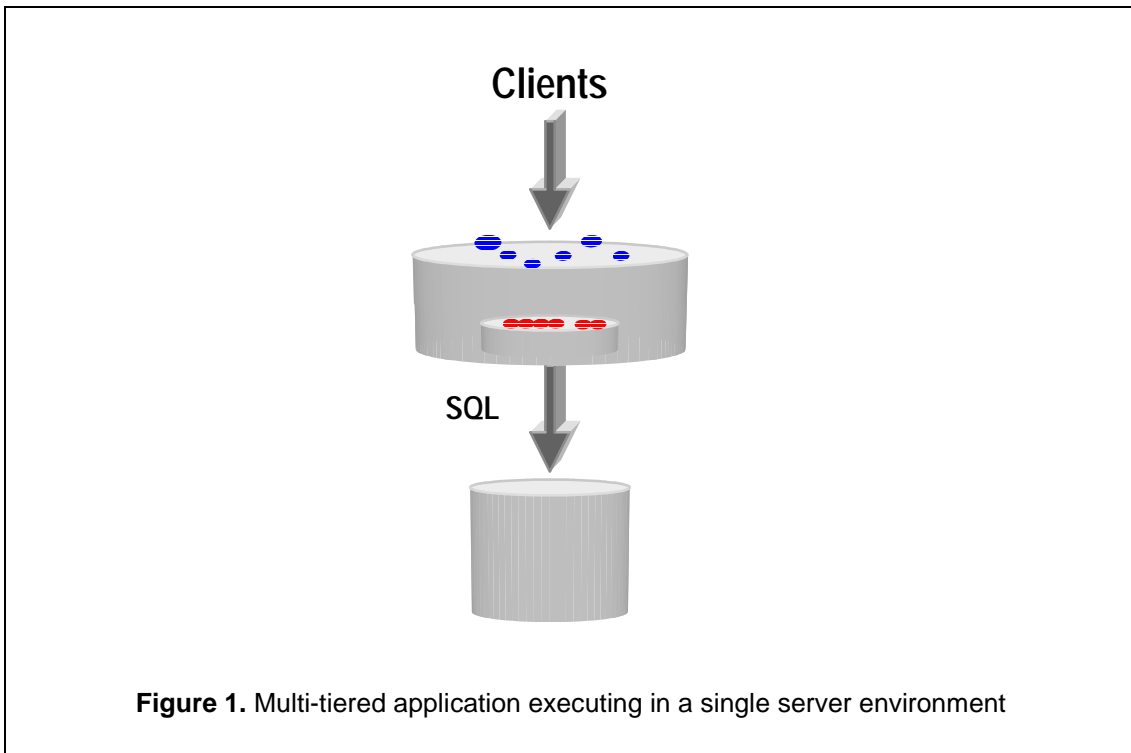
There are dozens of commercial and open-source O-R Mapping products for Java. These products typically support one or both of these standards. In addition, some of the products have their own native APIs for Java O-R mapping. In addition to supporting the two standard Java APIs for O-R mapping, the TORPEDO reference implementation currently supports the Oracle’s Toplink product and the open-source Hibernate technology [Hib].

This document is a functional specification for TORPEDO. In the next section we describe the single server and clustered application models that TORPEDO targets. We explain the nine optimizations that TORPEDO detects through seven application level operations. Section 5 describes the freely available TORPEDO reference implementation in detail. Section 6 defines a set of rules that alternate implementations of TORPEDO must adhere to in order to produce comparable results. Section 7 defines how TORPEDO results are reported. Finally, section 8 contrasts TORPEDO to performance benchmarks.

4 TORPEDO DEFINED

TORPEDO uncovers database access strategies of O-R mapping software by executing a *vertical slice* of a carefully crafted, multi-tiered on-line auction application. This simple application has an application architecture that is commonly found in enterprise applications. The TORPEDO application is deployed to both a single application server and to an application

server cluster. The single server deployment tests the sophistication of the O-R mapping software's cache. The clustered server deployment tests the ability of the O-R mapping software to operate in a scalable and fault tolerant environment. First we describe the single server, multi-tiered application model and then we describe the clustered version.



4.1 Single Server, Multi-tiered Application Model

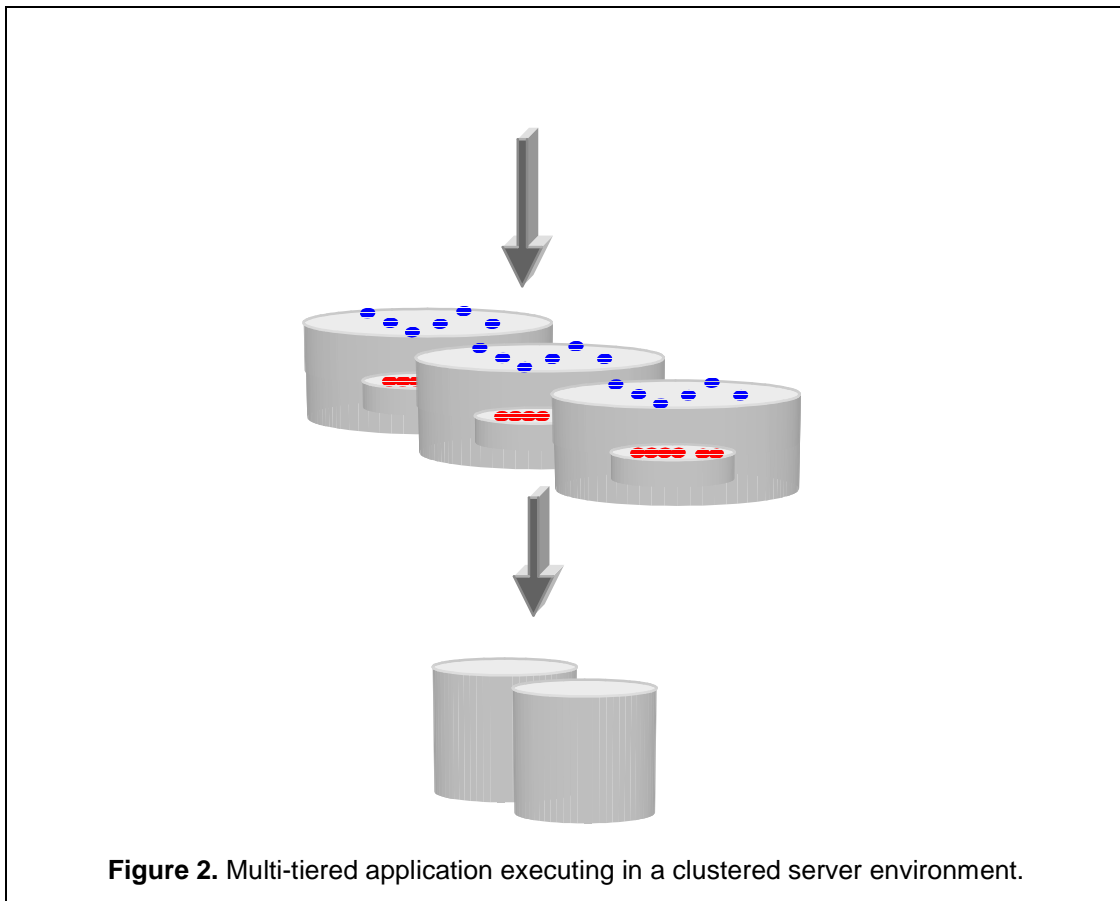
Figure 1 illustrates the common multi-tiered application model that TORPEDO targets. Distributed client code issues requests to a distributed object for some service. The distributed object executes in an application server. The service then communicates locally with persistent objects implemented by the O-R mapping software. Those objects represent cached database values. The O-R mapping software manages this cache, communicating with the database to read and write data values. All communication with the database is achieved using SQL.

TORPEDO tests database access from an application server, as opposed to testing direct two-tier database clients. While some O-R mapping software products do support two-tier clients, the benefits of caching are best understood in a long-lived shared server environment rather than in a single short-lived client. Furthermore, EJB's CMP standard is defined for accessing persistent data from an application server. We wanted to include support for CMP in TORPEDO and we wanted those results to be comparable to results from other O-R mapping software.

4.2 Clustered Multi-tiered Application Model

Figure 2 illustrates the multi-tiered application model executing in a clustered environment. TORPEDO tests the ability of the O-R mapping software to operate in such an environment. Distributed client code issues requests to a distributed object for some service. The fact that these are multiple servers that can provide the service is transparent to the client application code.

The request is directed to some server in the application server cluster according to some load balancing policy. This balances the load, allowing the service to scale. Furthermore, should a server fail, another server is available to process the request. The service then communicates locally with persistent objects implemented by the O-R mapping software. Those objects represent cached database values. The O-R mapping software manages this cache, communicating with the database to read and write data values. Since multiple servers



implement the service, the same data value may be cached in multiple servers. TORPEDO detects the ability of the O-R mapping software to correctly handle this.

As illustrated in figure 2, the database itself may be clustered, allowing access to it to scale and tolerate failures. However, this is not of interest in defining TORPEDO; the O-R mapping software is unaware of the database configuration since it is the database client.

4.3 Database Access Optimization Strategies

By executing a series of application level operations, TORPEDO detects various database access optimizations. We describe the optimizations here and then detail the TORPEDO application level operations in section 4.4.

4.3.1 Read only objects

Many applications just read objects without updating them. Intelligent O-R mapping software can make use of the fact and optimize access to objects that are read-only. For example, the data can be easily cached in multiple places and live in the cache for a long period of time. Since the object state is never updated, the management of the cache is simple.

4.3.2 Caching within a transaction

Some O-R mapping software caches object data within the boundaries of a transaction. Subsequent accesses to object data are against the cache and not the database. Similarly, all database updates are performed at transaction commit time instead of every time the object is modified.

4.3.3 Caching across transactions

While caching data within the scope of a transaction provides some benefit, caching data across transactions provides the real win. The data are not reloaded on subsequent transactions. The database is only accessed in a write-through fashion, when data is modified. This is a challenging optimization for O-R mapping software to accomplish, especially when supporting access to the data in a clustered environment or by other programs not using the O-R mapping software.

4.3.4 Partial load of object data

When O-R mapping software materializes an object, sometimes it is better to load some of the object's data rather than all of it. Intelligent O-R mapping software will only load the required object data and not all of it. This can save on memory resources used by the application as well as network traffic.

4.3.5 Bulk loads of multiple objects

When a query results in a collection of objects, sometimes it is better to load all of the resulting objects in a single database hit, rather than loading them one-by-one. A further refinement of this is to load just some of the resulting objects in batches.

4.3.6 Lazy loads of multiple objects

Sometimes it is better not to load all of the resulting objects of a query. Rather, the objects are loaded on an as needed basis.

4.3.7 Bulk loads of related objects

When an object-oriented application navigates from one related object to another, sometimes it makes sense to load all of the related objects in a single database hit. A further refinement of this is to load some of the related objects in batches.

4.3.8 Lazy loads of related objects

Sometimes it is better not to load all of the related objects. Rather, the related objects are loaded on an as needed basis.

4.3.9 Batch update operations

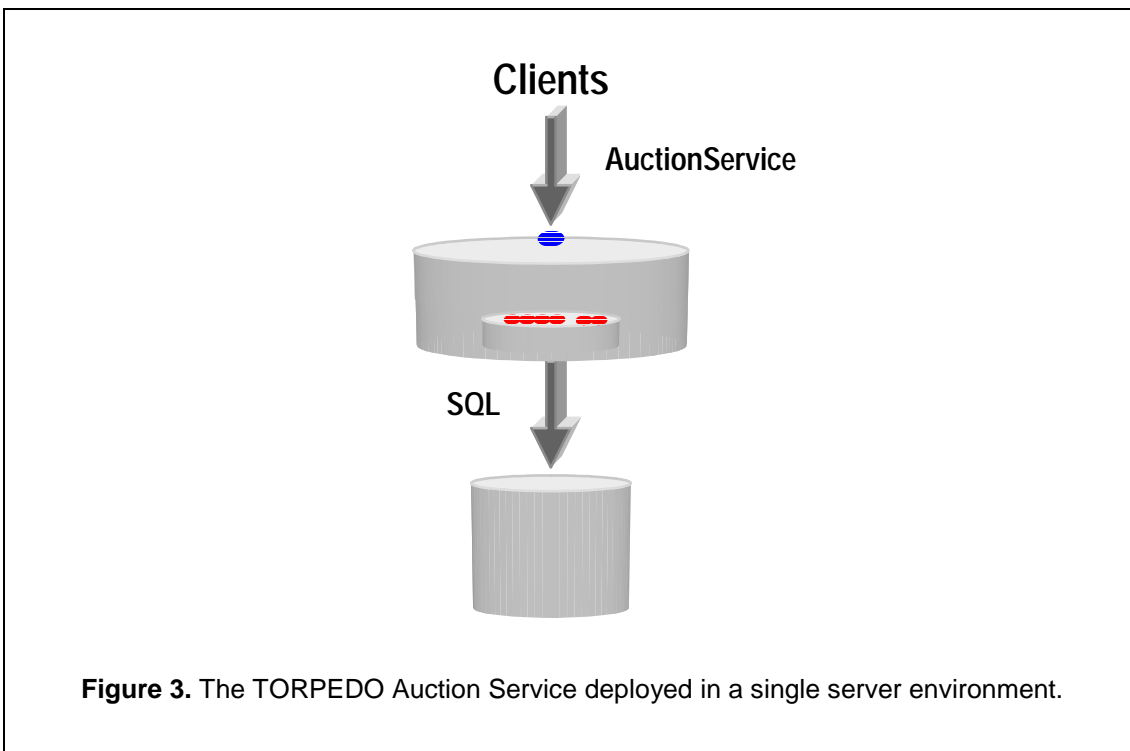
Rather than continually updating / inserting into a database during the course of a transaction, O-R mapping software could choose to batch all of the updates / inserts and execute them as a single database operation.

4.4 The Auction Application

TORPEDO uncovers database access strategies of O-R mapping software by executing a carefully crafted *vertical slice* of a multi-tiered on-line auction application. The auction application has been used for a couple of years to teach software architecture in the Middleware Company's Architecture course. [Mart] The on-line, interactive nature of a popular Internet auction places strenuous scalability and fault tolerance requirements on the application.

The auction application supports operations to create and authenticate users, open, list, find and close auctions and place bids. The TORPEDO vertical slice of the auction application is a minimal subset that is sufficient to uncover the database access optimizations described in section 4.3.

The TORPEDO Auction Service is illustrated in figure 3. Clients request auction services by using the *AuctionService* interface. In the reference implementation of TORPEDO, this is a Java interface for a stateless session Enterprise Java Bean. The interface is given in figure 4.



```

package com.middleware_company.torpedo.auction;
import javax.ejb.*;
import java.util.*;
import java.rmi.*;

public interface AuctionService extends javax.ejb.EJBObject {

    public AuctionInfo listAuction(String auctionID)
        throws AuctionServiceException, RemoteException;

    public AuctionInfo listAuctionTwiceWithTransaction(
        String auctionID)
        throws AuctionServiceException, RemoteException;

    public AuctionInfo listAuctionTwiceWithoutTransaction(
        String auctionID)
        throws AuctionServiceException, RemoteException;

    public AuctionInfo listPartialAuction(String auctionID)
        throws AuctionServiceException, RemoteException;

    public String placeBid(
        String auctionID,
        String bidID,
        String userID,
        Float amount,
        Float maxAmount)
        throws AuctionServiceException, RemoteException;

    public String place2bids(
        String auctionID1,
        String auctionID2,
        String bidID1,
        String bidID2,
        String userID,
        Float amount1,
        Float amount2,
        Float maxAmount1,
        Float maxAmount2)
        throws AuctionServiceException, RemoteException;

    public CollectionWithSQL findAllAuctions()
        throws AuctionServiceException, RemoteException;

    public CollectionWithSQL findHighBids(String auctionID)
        throws AuctionServiceException, RemoteException;
}

```

Figure 4. The AuctionService interface

As described in more detail in section 6, other communication protocols and middle-tier technologies, such as CORBA and SOAP/WebServices, are possible. The requirement is to support the same operational semantics as the reference implementation of the Auction Service.

4.4.1 Object model

The Auction Service is implemented in terms of an underlying *persistent* object model. Figure 5 illustrates the object model. This object model is what the O-R mapping software provides. It differs from the Auction Service interface given in figure 4. The implementation of the Auction Service interface operates on the persistent object model of figure 5.

The object model consists of Auction, Bid, User and Item objects. There are four different relationships between the objects. An auction has a one-to-one relationship with an item. An auction has a one-to-many relationship with a bid. A user has a one-to-many seller relationship with an auction. A user has a one-to-many buyer relationship with a bid.

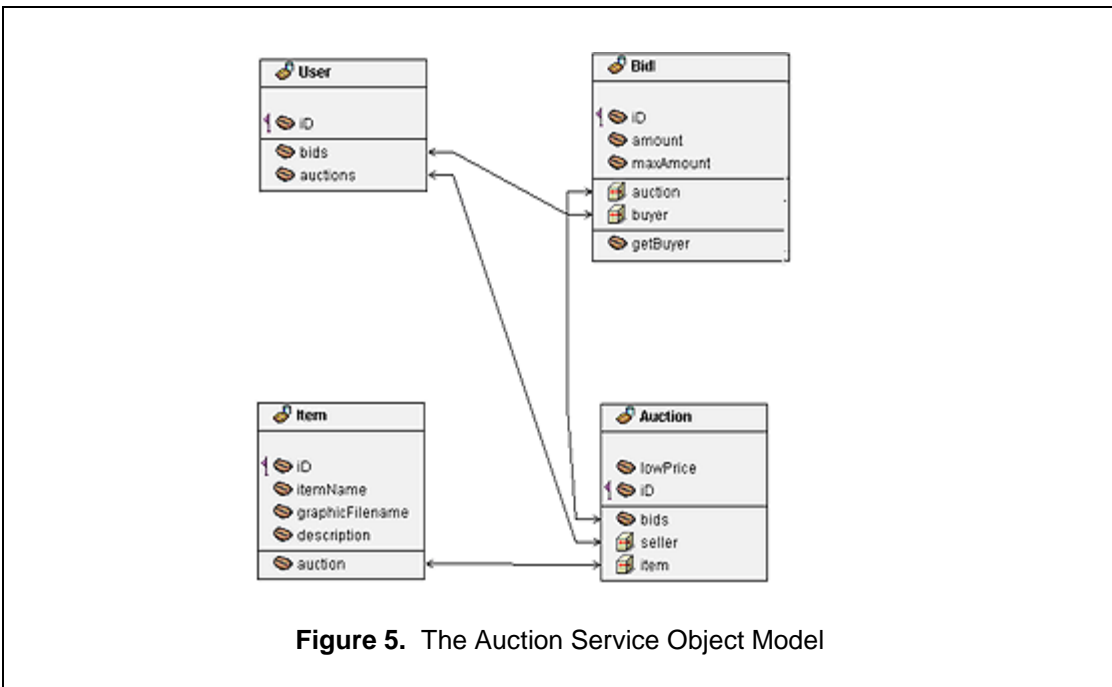


Figure 5. The Auction Service Object Model

4.4.2 Auction Service Operations

The following operations on the Auction Service test the O-R mapping software's sophistication in accessing a database. Each operation is explained and then related to the optimizations specified in section 4.3

4.4.2.1 listAuction

The listAuction operation returns information about the auction identified by the auction identifier.

```
AuctionInfo listAuction(String auctionID)
```

The AuctionInfo object includes the following data.

```
String description;  
Float lowPrice;  
Collection bids;  
String sellerName;  
String itemName;  
String itemGraphicFilename;  
String itemID;  
String auctionID;  
String sql;
```

AuctionInfo includes a collection of bid objects, as given by BidInfo. The BidInfo object includes:

```
Float bidAmount;  
Float maxBidAmount;  
String buyer;
```

The *sql* property in the AuctionInfo object contains the SQL statements that were executed by the O-R mapping software to obtain information about the auction.

The AuctionInfo object contains information from an auction object, as well as bid, item and user objects. The listAuction operation reveals how sophisticated the O-R mapping software is at retrieving data across related objects. In particular, it reveals if the O-R mapping software can perform bulk loads of related objects (4.3.7). All of this information could be obtained in a single SQL join statement.

4.4.2.2 listAuctionTwiceWithTransaction

The listAuctionTwiceWithTransaction returns information about the auction identified by the auction identifier. It returns the same information returned by the listAuction operation. The operation executes the listAuction operation two times, in succession.

```
AuctionInfo listAuctionTwiceWithTransaction(String auctionID)
```

The listAuctionTwiceWithTransaction operation reveals how sophisticated the O-R mapping software is at caching data *within* (4.3.2) transaction boundaries.

4.4.2.3 listAuctionTwiceWithoutTransaction

The listAuctionTwiceWithoutTransaction returns information about the auction identified by the auction identifier. It returns the same information returned by the listAuction operation. The operation executes the listAuction operation two times, in succession.

```
AuctionInfo listAuctionTwiceWithoutTransaction(String auctionID)
```

The listAuctionTwiceWithoutTransaction operation reveals how sophisticated the O-R mapping software is at caching data *across* (4.3.3) transaction boundaries. The listAuctionTwiceWithoutTransaction operation is run with the transaction attribute to NEVER. Each invoked listAuction operation is run as a separate transaction. Since it is invoked twice, this detects if the O-R mapping software is capable of caching data across transaction boundaries.

If the O-R mapping software supports intelligent management of read-only objects (4.3.1), the listAuctionTwiceWithoutTransaction can test it. The first access to the auction, bid, user and

item information would populate the cache and the second access would just access the cache because the objects are declared to be read-only.

4.4.2.4 listPartialAuction

The listPartialAuction returns some, but not all, information about the auction identified by the auction identifier.

```
AuctionInfo listPartialAuction(String auctionID)
```

In particular, the listPartialAuction returns an AuctionInfo object with the following string attributes set to "not needed"

```
itemGraphicFilename  
sellerName  
itemID  
description
```

Furthermore, the bids field of the AuctionInfo object is an empty collection.

The listPartialAuction operation tests the O-R mapping software's ability to partially load an object's data (4.3.4). The listPartialAuction also tests the O-R mapping software's ability to load related objects in a lazy fashion (4.3.8). Information from the related user and bid objects is not needed.

4.4.2.5 findAllAuctions

The findAllAuctions operation returns a collection of AuctionInfo objects, representing information about all of the auctions.

```
CollectionWithSQL findAllAuctions()
```

The type of the returned object is CollectionWithSQL. CollectionWithSQL has the following two properties:

```
Collection theCollection  
String theSQL
```

The first is the Collection of AuctionInfo objects and the second is the statements that were executed by the O-R mapping software to obtain information about all of the auctions.

The findAllAuctions operation tests the O-R mapping software's ability to perform a bulk load of multiple objects (4.3.5) and lazy load. When a lot of data is returned, this operation also tests the ability of the O-R mapping software to present the results in batches.

4.4.2.6 findHighBids

The findHighBids operation returns a collection of the highest bids for the auction identified by the auction identifier.

```
CollectionWithSQL findHighBids(String auctionID)
```

The findHighBids operation tests the O-R mapping software's ability to perform a bulk load of multiple objects (4.3.5). It also tests the O-R mapping software's query language support for aggregate functions. Lacking support, finding the high bids must be performed at the application rather than the database.

4.4.2.7 placeBid

The placeBid operation creates a new bid for an auction identified by the auction identifier. The bidID identifies the newly created bid. It is the client's responsibility to provide a unique bidID value for the new bid.

```
String placeBid(  
    String auctionID,  
    String bidID,  
    String userID,  
    Float amount,  
    Float maxAmount)
```

The placeBid operation tests the correctness of the various list and find operations in the presence of concurrent additions of bid information. All operations must return correct responses in the presence of concurrent placeBid operations. See section 4.4.4 for correctness criteria.

Note that the use of listAuctionTwice (4.4.2.3) to test for read-only objects is not required to operate with concurrent placeBid operations since with that test the auction information is declared to be read-only.

4.4.2.8 Place2bids

The place2bids operation places two bids on the auctions identified by auctionID1 and auctionID2.

```
String place2bids(  
    String auctionID1,  
    String auctionID2,  
    String bidID1,  
    String bidID2,  
    String userID,  
    Float amount1,  
    Float amount2,  
    Float maxAmount1,  
    Float maxAmount2)
```

The place2bids operation tests the O-R mapping software's ability to batch updates / inserts (4.3.9) into the database. The place2bids operation runs as a single atomic transaction.

4.4.3 Running TORPEDO Tests

TORPEDO defines a series of clients that are run to obtain the results of the operations, including the SQL that was generated by the O-R mapping software. There is one client for each operation listed in section 4.4.2. There is also a client that concurrently places bids and executes the list and find operations.

The TORPEDO clients are intended to be run with an empty O-R mapping cache and an initial database. This makes the tests deterministic and repeatable. The simplest way of achieving this is to start up the application server prior to running each test. The initial database should be restored after running the placeBid and place2Bid tests. The remaining operations do not modify the database.

4.4.4 Correctness

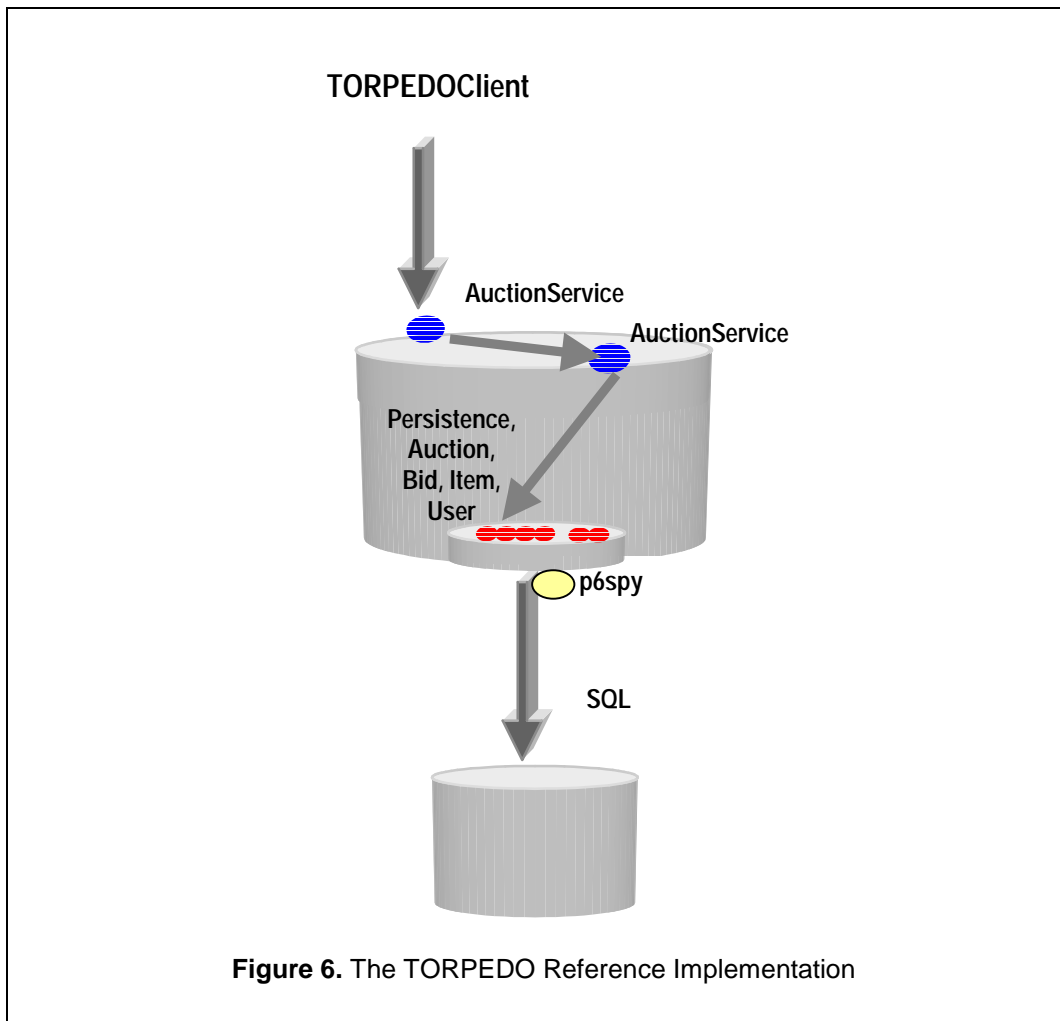
Many database management systems and some application servers have support for applications that can tolerate inconsistent data resulting from non-serialized transactions. This compromises transaction isolation for increased concurrency. In many systems this is manifested as a so-called *isolation level* property. However, the correctness criterion for the Auction application is strict serializability. That is, the transactions against the database need to appear as if they were executed in some serial order. Each of the TORPEDO operations is executed as an individual transaction.

Results returned from each operation must reflect serialized transactions. Any results that do not are considered incorrect and invalid.

5 THE TORPEDO REFERENCE IMPLEMENTATION

The Middleware Company has made available a reference implementation of TORPEDO. The implementation is a J2EE application that can be configured to use EJB Container Managed Persistence 2.0 (CMP), Java Data Objects (JDO) or Hibernate.

Figure 6 illustrates the reference implementation executing in a single server environment. The reference application can also be deployed to a clustered server environment.



The client accesses the auction service via the AuctionService interface given in figure 4. The AuctionService interface is implemented by two stateless session EJB components, TORPEDOAuctionServiceBean and AuctionServiceBean. Client requests are initially directed to TORPEDOAuctionServiceBean. It initializes the SQL capture facility and then delegates to the AuctionServiceBean.

An open-source JDBC driver called *p6spy* [p6spy] wraps any existing JDBC driver and logs the SQL sent to the database. By using the p6spy driver, the TORPEDOAuctionServiceBean captures the SQL generated for each test. Upon return from the corresponding calls to the AuctionServiceBean, the TORPEDOAuctionServiceBean adds the captured SQL to the operation result.

The AuctionServiceBean makes calls on the persistent object model. The persistent object model is abstracted via Java interfaces that are independent of the O-R mapping software. In particular, the AuctionServiceBean accesses the persistent object model through a Persistence interface, which serves as a factory for creating and finding objects implementing the Auction, Bid, Item and User interfaces. These abstract interfaces for the object model have allowed the CMP, JDO and Hibernate versions to be integrated into the TORPEDO framework. Other Java O-R mapping software could be integrated by supporting those five interfaces.

5.1.1 Transactions

The reference implementation uses J2EE's container managed transactions. All of the transaction attributes are set on the TORPEDOAuctionServiceBean methods as "NEVER". The transaction attributes are set on the AuctionServiceBean as "REQUIRED". This results in the EJB container starting a transaction for each AuctionServiceBean and allows the TORPEDOAuctionServiceBean to capture the commit statements on the database.

The listAuctionTwiceWithTransaction operation (4.4.2.2) is run with the transaction attribute set to "REQUIRED". This makes the entire operation a transaction. The listAuctionTwiceWithoutTransaction operation (4.4.2.3) is run with the transaction set to "NEVER". This means that the two invocations of the listAuction operation are individual transactions. This setting tests if the O-R mapping software can correctly cache object data across transaction boundaries.

5.1.2 Database

The TORPEDO reference implementation uses a Hypersonic SQL database [Hyp]. An initial database, populated with auction data, is included with the reference implementation. Other database systems could be used as long as a JDBC driver is available.

The CMP, JDO and Hibernate implementations of the persistent object model are all mapped to the same database schema. Figure 7 gives the schema.

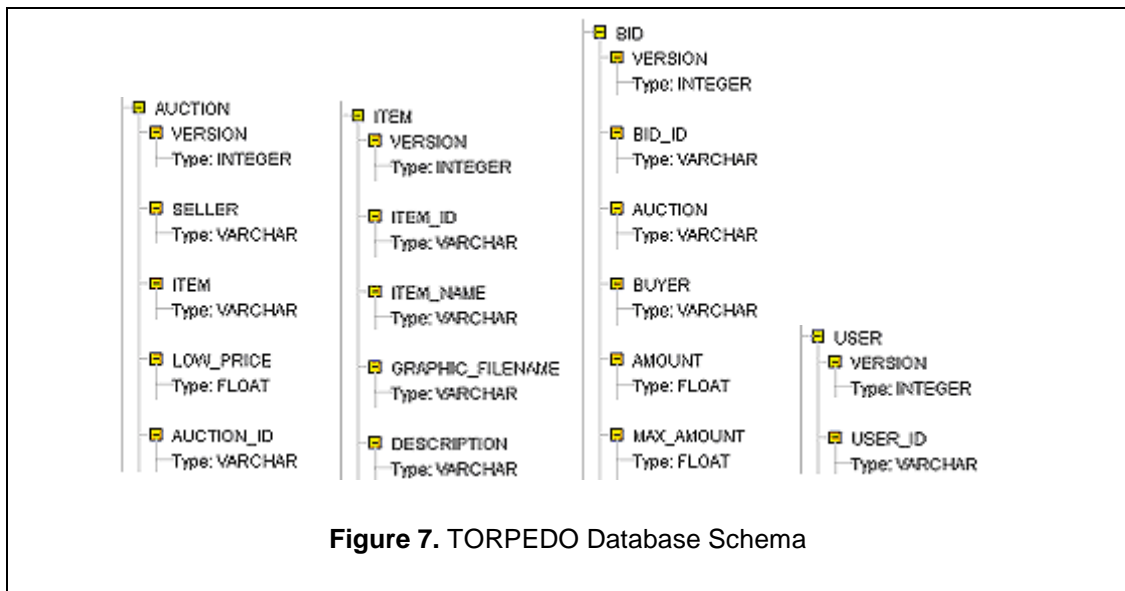


Figure 7. TORPEDO Database Schema

6 COMPLIANT TORPEDO IMPLEMENTATIONS

The goal of TORPEDO is to be able to compare O-R mapping software's access of a relational database from a standard multi-tier application. Results from the TORPEDO reference implementation are comparable. In order to test a particular O-R mapping software product supporting a standard API, the reference implementation is configured and tuned for that product. The tuning is done to produce to best correct result possible for that product.

Other TORPEDO implementations, not based directly on the reference implementation, are also possible. For example, TORPEDO implementations for other object-oriented languages, such as C#, C++ and Smalltalk, are possible. Similarly, TORPEDO implementations based on other distributed computing platforms are possible. But in order to ensure that the results are comparable with results from the TORPEDO reference implementation, certain implementation rules must be followed.

Alternate TORPEDO implementations must be created in the spirit of the reference implementation. The goals of implementing an alternate TORPEDO implementation should be to test O-R mapping software that is incompatible with the reference implementation and to produce comparable results with those produced from the reference implementation. Its architecture should be similar to the reference implementation's architecture. While this is a necessarily vague requirement, explicit requirements are:

1. The TORPEDO implementation must be designed as a client communicating with a middle-tier server. The middle-tier server need not have a J2EE interface. Other protocols and middleware frameworks, such as CORBA or SOAP, can be used. However, the semantics of the service interface must be the same.
2. The TORPEDO implementation must return the application result and the executed SQL to the client.
3. The TORPEDO implementation cannot do caching itself. The O-R mapping software performs caching, if any.
4. The database schema can be changed in the following ways: The names of tables and columns can be renamed. Columns can be added or removed from the tables.

5. Implementing the middle-tier in the database itself is not acceptable. In particular, stored procedures in the database are not allowed.
6. Each TORPEDO operation must be executed in a transaction. The transaction boundaries described in section 5.1.1 must be respected.
7. If the underlying middleware technologies support clustering, a clusterable version must be provided.

7 REPORTING TORPEDO RESULTS

When TORPEDO is run against a particular O-R mapping software product, extensive results are produced. The following serves as a *guide* for reporting those results.

Results may be *untuned* or *tuned*. Untuned results are results from running TORPEDO on a particular O-R mapping software product without providing any application specific information to the O-R mapping software. All of the defaults are used. Tuned results, on the other hand, are results that are produced after configuring the O-R mapping software with configuration values that make TORPEDO run more efficiently. Many O-R mapping products provide XML configuration files and specific APIs to support tuning.

There are three goals in reporting TORPEDO results. The first goal is to provide the actual results, namely the application results and the generated SQL statements. The second goal is to provide sufficient configuration information about all of the technologies being used in order for an independent party to be able to recreate the results. The third goal is to provide analysis of the results. In particular, the analysis should state if the resulting SQL does indeed reflect the optimizations associated with each operation. The following table guides the reporting of results.

Contact Information	Name, affiliation, email address, telephone, etc. of the individual conducting the TORPEDO test should be provided.
Configuration	
O-R Mapping System	The name and version of the O-R mapping software being used.
O-R Configuration	A description of the configuration. For example, a description of cache settings that were used.
O-R Configuration Files	This should be a reference to the actual configuration files that were used.
TORPEDO Implementation	If the reference implementation was used, provide the version used. If a different implementation was used, state so.
Application Server	The name and version of the application server being used.
Database System	The name and version of the database system being used.
Operating System	The name and version of the operating system being used.
Hardware Platform	The hardware platform being used.
Configuration Notes	Any additional configuration notes.
TORPEDO Operations	For each operation, the returned application result and the generated SQL should be provided. Additional operation specific configuration information and analysis should be included.
listAuction	
listAuctionTwiceWithTransaction	
listAuctionTwiceWithoutTransaction	
listPartialAuction	
placeBid	
place2Bids	
findAllAuctions	
findHighBids	
Clustered TORPEDO	If the O-R mapping software being tested supports execution in an application server cluster, the results should be reported for <i>each</i> operation. Otherwise, it should be stated that application server clusters are not supported.
Additional Notes	

Multiple versions of this table may be produced for a given product with different configurations. TORPEDO should first be run on an O-R mapping product without performing any tuning and then run it again after tuning the O-R mapping product.

8 RELATED WORK

To our knowledge there are no similar frameworks to test the sophistication of O-R mapping software. There are several performance benchmarks for object-oriented databases, object-relational databases and enterprise applications that could be used in doing performance studies of O-R mapping software.

8.1 Object Relational Database Benchmarks

BUCKY is a query-oriented benchmark from the University of Wisconsin for object relational databases. As described by the designers of the benchmark, "BUCKY tests many of the key features offered by object-relational systems, including row types and inheritance, references and path expressions, sets of atomic values and of references, methods and late binding, and user-defined abstract data types and their methods."

BUCKY has been primarily used to understand and compare object-relational database systems with each other and with relational database systems.

8.2 Object Oriented Database Benchmarks

Several benchmarks have been designed for object-oriented databases, such as OO1, HyperModel and OO7. These benchmarks test operations that are typical on design objects, that is CAD, CAM and CASE type applications and data. They could be useful for characterizing the performance of O-R mapping software. There is purportedly a JDO version of OO7 currently under development.

8.3 RUBiS

The RUBiS benchmark was developed at Rice University as a means of comparing application design patterns and application server performance. A variety of application architectures were implemented for the benchmark; each uses a unique combination of Java technologies, including JDBC, servlets, session beans, and entity beans.

A JDO implementation of RUBiS has been developed, allowing JDO's interface, application design patterns, and performance to be compared with these other J2EE technologies.

9 ACKNOWLEDGMENTS

We'd like to thank The Middleware Company for supporting this research. We'd also like to thank Amol Daterao for implementing the Hibernate support in TORPEDO.

10 REFERENCES

[AABDMZ89] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik. "The Object-Oriented Database System Manifesto." In *Proceedings of the First International Conference on Deductive and Object-Oriented Databases*, pages 223-40, Kyoto, Japan, December 1989.

[Catt95] Cattell, R.G. *The Object Database Standard: ODMG-93 (Release 1.2)*. Morgan Kaufman Publishers. San Francisco, 1995.

[CMP] *Enterprise JavaBeans™ Specification 2.0 Final Release 2*. Sun Microsystems.
<http://java.sun.com/products/ejb/>

[Hib] Hibernate Open Source Software. <http://www.hibernate.org/>

[Hyp] Hypersonic SQL database <http://hsqldb.sourceforge.net/>

[JDO] JSR-000012 Java™ Data Objects (JDO) Specification, Sun Microsystems. Craig Russell, Specification Lead. <http://jcp.org/aboutJava/communityprocess/final/jsr012/index2.html>

[Mart] Bruce Martin. Course Notes from The Middleware Company's EJB For Architects. 2002-2003.

[NP91] Nahouraii, E. & Petry, F. *Object-Oriented Databases*. IEEE, 1991.

[p6spy] P6Spy Open Source Software. <http://www.p6spy.com/>

[Stone97] Stonebraker, M. [*Architectural Options for Object-Relational DBMSs*](#). Informix Software, CA. Feb, 1997.

[Stone99] Stonebraker, M., Brown, P., and Moore, D. *Object-Relational DBMSs: Tracking the Next Great Wave 2e*. Morgan-Kaufman Publishers. San Francisco, 1999.