

# eXtensible Database Adapter - a framework for CORBA/ODBMS integration

Serge Shumilov, Armin B. Cremers  
Department of Computer Science III  
University of Bonn  
Bonn, Germany  
e-mail: [shumilov@cs.uni-bonn.de](mailto:shumilov@cs.uni-bonn.de)

## Abstract

This paper describes the first hand experience gained in implementing remote CORBA access for object-oriented database ObjectStore. Integration of CORBA and Object-Oriented Database Management Systems (ODBMS) permits the encapsulation of the powerful database facilities within the heterogeneous CORBA environment. As opposed to current OMG ideas for handling persistent objects, this approach pays more attention to support distribution for existing databases. The main idea is to reduce costs of the wrapper's development using an adaptable to a proprietary database basis – an eXtensible Database Adapter (XDA). The XDA's prototype was evaluated in the development of CORBA wrapper for real object-oriented database<sup>1</sup>.

## 1. Introduction

Most today's real-world applications are often both decentralized and highly data-intensive. Any used technology has to provide an integrated strategy for solving both distribution and data management problems. One possible solution can be the integration of the database system with a standard distributed platform like *Common Object Request Broker Architecture (CORBA)* [10]. In contrast to database systems, CORBA provides a flexible transparent distribution model with a larger-scale set of services. Integration of both technologies permits the encapsulation of the powerful database facilities within the heterogeneous CORBA environment.

The paper is organized as follows: the second chapter reviews the existing CORBA/ODBMS integration tech-

---

<sup>1</sup> This work is funded by the German Research Foundation (DFG) within the collaborative research center SFB 350 at the University of Bonn and the joint project "Interoperable geo-scientific information systems".

---

Proceedings of the 2<sup>nd</sup> International Workshop on  
Computer Science and Information Technologies  
CSIT'2000  
Ufa, Russia, 2000

niques for providing distributed support for existing databases, Orbix [4] as the ORB implementation and ObjectStore [8] as the ODBMS. Chapter 3 provides detailed discussion on the proposed techniques implemented in the XDA prototype. A short case study demonstrating the way the XDA can be employed in practice is presented in chapter 4. The development of a CORBA wrapper on top of the XDA for an object-oriented 3D-4D spatial database *GeoStore* [3] is discussed. Chapter 5 discusses some possible future developments and outlines the future work. The last chapter summarizes the contributions of this article.

## 2. General approaches to CORBA/ODBMS integration

CORBA does not support object persistence directly, but allows an integration with different database systems. Usually it is a construction of a middle-tier layer between the CORBA clients and the database that wraps the original database interface and provides an equivalent CORBA compatible interface. Traditionally the layer's implementation is named a *wrapper*. The actual CORBA specification [10] defines two standard ways that could help to reduce costs of the wrapper's development: *Persistent Object Service* and *Object Database Adapter*.

### 2.1 Persistent Object Service

The *Persistent Object Service (POS)* is one of the standard Object Services specified in the OMG document [11]. Unfortunately, the standard is very general and complex. Some research groups that tried to implement it have discovered many ambiguous areas in the specification [12, 16]. Two examples are: the vague semantics of the operations, the weak specification of how POS interact with the database and other Object Services. Consequently the POS does not yet have any complete implementation and therefore we will not consider it as a sound approach. The second version of this service - *Persistent State Service* is now in the development process.

### 2.2 Object Database Adapter approach

A service like POS typically does not require extensions to the ORB and its standard components: it is just

treated like a regular application. Conversely an *Object Database Adapter (ODA)* approach provides a tight integration between the Object Request Broker (ORB) and the database system. The idea of the special object adapter for persistent objects at first was pursued in the ODMG standard [9] and later implemented in some projects [13].

The ODA is an object adapter for objects stored in a database. It does not completely replace the standard object adapter, but represents its extension that depends on a particular database system. ODA is responsible for the managing of the persistent state of implementation objects and is aimed at the simplification of their programming. It should also deal with other typical database features, e.g. locks and transactions, since persistent objects are generally shared.

### 3. First experience with commercially available adapters.

In the first wrapper's prototype we used two commercially available tools - *Orbix/ObjectStore Adapter (OOSA)* [5] and *Orbix Database Adapter Framework (ODAF)* [6]. The first one is a tuned to ObjectStore implementation of ODA for ObjectStore database system and the second one is a framework for the development of similar adapters for a general ODMG-compatible ODBMS. The presence of these tools was one of the reasons for choosing Orbix [4] as an implementation platform. The OOSA helped us to considerably speed up the wrapper's development in the initial stages. However, after several experiments with our prototype, we found that OOSA does not meet all requirements for providing distributed support for existing databases [3]. Now we will summarize the most critical of them:

1. *Soft, non-intrusive integration*: Since database schema evolution may be extremely time-consuming, ideally the CORBA-conformance should be achievable without any modifications in existed databases and applications. A more important that remotely accessible through CORBA databases should be also available for local database applications.
2. *Persistence of object references*: The reference should enable the adapter to find the stored objects and retrieve them from the database. The difference between CORBA references to transient and persistent objects is that the latter should be valid during substantially longer time. It could be stored or exchanged by the clients.
3. *Management of databases*: CORBA clients should have a possibility to work with persistent objects in the same way that they did with usual CORBA objects. More precisely, every access to persistent objects should be done within an open transaction and adapter should care about it. For advanced clients, the adapter should also provide a possibility to use the native database functionality, e.g. to manage

with database objects, databases, segments and transactions.

## 4. Proposed approach - eXtensible Database Adapter

Using our own experience with the OOSA/ODAF as an example, we developed an alternative adapter an eXtensible Database Adapter (XDA). In our study we tried to implement the most critical issues of CORBA/ODBMS integration that we missed in commercial adapters. The main idea of our approach is to make database objects accessible to CORBA clients through corresponding transient mediators and provide necessary for their development and management tools.

### 4.1 Mediators as intermediate communication components

A primary strength of object-oriented databases lies in their ability to model complex objects and inter-relationships among them. In the particular case of a ODBMS that adds database features to the C++ like ObjectStore, database entities are instances of usual C++ classes. To make them available to the CORBA clients, a wrapper should wrap the interface of every database class by an equivalent CORBA compatible class. This class implements all methods of original database classes by the equivalents and is responsible for the correct data mapping between CORBA C++ and ObjectStore C++ data models. The instances of these classes act as usual CORBA objects as well as database clients. In other words they *mediate* between CORBA client and database objects converting data parameters and delegating all function calls in the both directions. Because of this basic function we will name such objects *mediators* [17].

The transience of mediators not only saves database space, but also provides a complete separation between the original database and communication components. This separation allows working with the database not only through the new CORBA interface, but also through the original C++ interface. Moreover, when employing this method any already existing data store with a compatible schema can always be used through the CORBA-interface. Referential integrity of the data shared by local and CORBA-conformant applications is guaranteed by the use of the native ObjectStore transaction control mechanism.

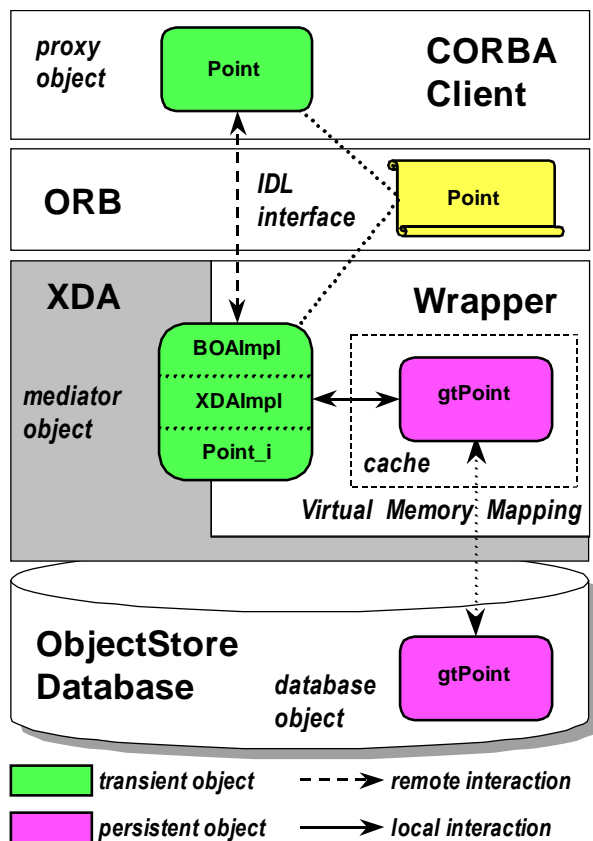


Figure 1 XDA binding approach.

## 4.2 Implementation of mediators in XDA

For each original ObjectStore C++ class, a server programmer should create the corresponding CORBA C++ mediator class with an equivalent IDL interface. Original data structures from the ObjectStore C++ data model should be represented by appropriate structures in the CORBA IDL model (Figure 1). Because of differences in object's representations in both models automatic mediator's code generation from the specifications of corresponding database classes is possible only in particular cases [1]. However, to make implementation of mediators easier XDA provides a special template class XDAImpl. The template should be inherited by every mediator and tuned to the corresponding database class.

The main idea of the template is to keep the specifications of mediators classes free from the XDA-related behavior and concentrate it in the template. This makes a clear separation between the functionality of the mediators that is related to the XDA and the functionality that is related to the mediation. Additionally the template implements the operations that are common to all ObjectStore objects and thus helps the programmer to speed up the whole development process.

## 4.3 Management of mediators

The price of transient mediators is the responsibility of XDA to control the lifetime of mediators – their crea-

tion, deletion and the binding to the corresponding database objects.

### Activation

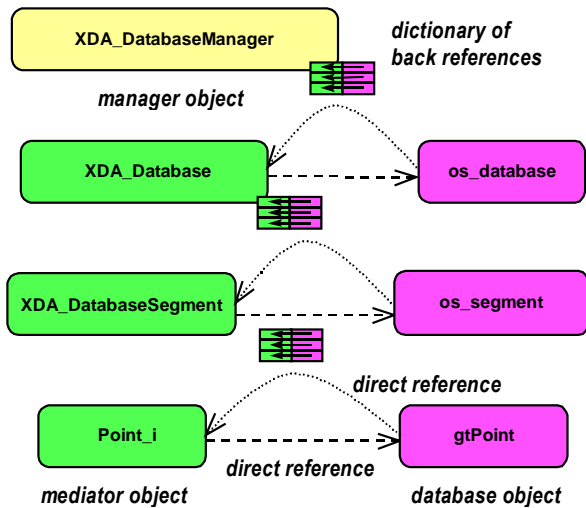
Some methods of database classes return as a result a reference to a new or an existing database object. However, CORBA compatible methods of the mediators must return a reference to the corresponding mediator object. Therefore XDA controls the binding between mediators and database objects providing a service that allows to find a mediator for a particular database object, if it already exists, or create a new mediator if it was not found. Here adapter will care about right correspondence of mediators to their database objects, e.g. to avoid the creation of multiple transient mediators for the same persistent database object.

A straightforward solution would be to use the global ORB table of available objects. The ORB will find the corresponding mediator object or invoke a loader that will make the new one. This simple strategy has two drawbacks. Since the bind() always increases the object's reference counter, it is not simple to determine the correct counter's value in the case of multiple clients. The second drawback is low search speed.

Another more efficient method was used in XDA. The back-references to the mediators are stored in the special transient dictionaries of references. Using a reference to a database object as the key it is possible to find a reference to the corresponding mediator. The uniqueness of persistent pointers and the use of hash tables make the retrieval from the dictionary very efficient. Moreover, XDA reduces the search organizing dictionaries in the hierarchy of small dictionaries according to object's locations (Figure 2). During the search, the manager finds the database and the segment to which the database object belongs and goes through the hierarchical organization of the dictionaries to the target mediator. If a programmer knows to which ObjectStore database or segment the resulting database object belongs, he can accelerate the search procedure directly looking in the corresponding dictionary (of the database or the segment).

### Deactivation

Mediators that are no longer needed should be temporarily removed from the server's address space to avoid memory overloading. An Evictor pattern proposed by IONA [7] provides a common framework for implementing different deactivation (eviction) strategies (least recently used, least frequently used, etc.).



**Figure 2 Hierarchical organization of transient dictionaries of back references.**

The simplest possible solution to accomplish deactivation of mediators is to include an additional method in the IDL interface through which CORBA clients notify the server that a particular object is no longer needed. The XDA prototype provides *evict()* functions that allow a CORBA client to explicitly deactivate a concrete mediator, mediators for all objects located in the segment or all objects in the database. Here the dictionaries of the back-references (Figure 2) are reused as tables of the active mediators.

This method gives the most effective results both in performance and in the optimal memory consumption, but makes CORBA clients explicitly aware of the object existence. The manual detection and reclamation of garbage increase code complexity and places a heavy burden on the developers of large programs. For these reasons the XDA provides also automated garbage collection.

However, it is not easy to determine when a mediator object is no longer needed. For example, to deactivate mediators which are idle for more than a specified time-out period or when memory runs short. Instead of them XDA uses another deactivation strategy based on integration with ObjectStore cache manager. The strategy assumes that all mediator objects that do not belong to the persistent objects mapped in the cache are recognized as possible candidates for deactivation. Real deactivation is made only if the amount of free memory becomes critical.

### Activation of previously deactivated objects or Persistent Interoperable Object References

CORBA clients work with remote server objects through *Interoperable Object References (IORs)* that identify remote objects in the CORBA environment. But if a mediator was previously deactivated, then the XDA's loader is asked by the ORB runtime system to retrieve the missed mediator. For unique identification of mediators and corresponding database objects is sufficient to know the type name of mediator and corre-

sponding local database pointers, since one mediator object can be implemented with help of several database objects. Therefore XDA modifies IOR references saving additional information in the marker.

When the XDA's loader is asked by the Orbix runtime system to retrieve the missed mediator, usually it retrieves the necessary information from IOR and creates the mediator. However, object activation need not always lead to the creation of a new mediator object. Actually, mediators have no state except the reference to the corresponding database object. Therefore XDA tries to use the same mediator for all database objects of the same type, if possible. Only the pointer stored on mediator persistent pointer and registration information in the ORB runtime system must be changed.

Evaluation of both methods shows that the second saves the server's memory when the server must manage with many small mediator objects. For example, it is very useful if a client navigates through a large collection. On the other hand this method lose in performance in general case, because activation of a new object assumes implicit deactivation of another object. When a client has two objects of the same type and by turn accesses them, almost every request causes an invocation of the loader. It might be possible to find an optimum compromise by defining a default amount for every type of mediators. This is currently one of our recent research topics.

The used format of the IOR is quite powerful and additionally allows the XDA, besides providing persistence to CORBA objects, to provide persistence to the corresponding object references. This means that a client that has an object reference can use it at any time without warning, even if the mediator has been deactivated or the server system has been restarted. With the persistence of object references, it makes perfect sense for the client to store an object reference for later use or send it to other client. For example, references to the persistent CORBA objects implemented by a server X can be stored by a server Y (a client of server X) and vice-versa, thereby enabling the construction of ORB-connected multi-databases.

## 4.4 Transaction management

Before the first request for an object is delivered, XDA automatically start a corresponding transaction, if necessary, and activate the corresponding mediator.

### Common techniques

There are two styles of transaction management generally present in commercial ODAs: *per-operation* and *phased*. In the *per-operation* style every operation invocation will be proceed within a separated transaction. When a request for method invocation arrives at the server, the adapter automatically begins a new transaction and commits (or aborts) it immediately after the results are returned to the client. In the *phased* style, multiple operations calls can be performed within a single transaction. In this case CORBA clients should

explicitly set the boundaries by invoking corresponding operations in the target object's interface.

The cardinal difference between both styles is that the transaction management performs *implicitly* for the client in the first case and *explicitly* in the second. The main advantage of the *per-operation* style is that it allows CORBA clients to work with persistent objects in the same way that they did with usual CORBA objects. They need know nothing about transactions. In addition, it does not lock the database objects for a long time as in the *phased* style that makes them quickly available for other concurrent clients. The disadvantage of this style is a restriction that any transaction may span no more than one operation call. For some applications it may be inefficient to start and to end transactions so frequently. In this situation the explicit transaction control method - *phased* is preferable. The measurements made in one client environment in the server side showed that in the *phased* style the operation invocation is approximately 2 (read-only) - 5 (update) times faster than in *per-operation* style. Since the both, implicit and explicit methods have their advantages and disadvantages the client should be able to employ them according to the situation.

### Implicit transaction management in XDA

For "pure" CORBA clients that do not want to care about transactions the XDA provides an automatic transaction control mechanism that combines the best characteristics of the both methods. By default, transaction boundaries are implicitly controlled by the transaction manager (Figure 3). The manager performs a "conservative" method for transaction management. Basic rules of this method are:

- if possible, to prevent the start of a new transaction
- to keep an open transaction so long time as possible
- to prefer "read\_only" to "update" transactions

Before a program can access persistent data, it must start a transaction. Mediators mark the beginning and end of transaction by using function calls that are provided by the XDA's transaction manager. `begin()` returns a pointer to an `XDA_Transaction` object that represents the *virtual transaction* of the current session. This separation between real transactions allows to dynamically define transaction boundaries, depending on run-time conditions.

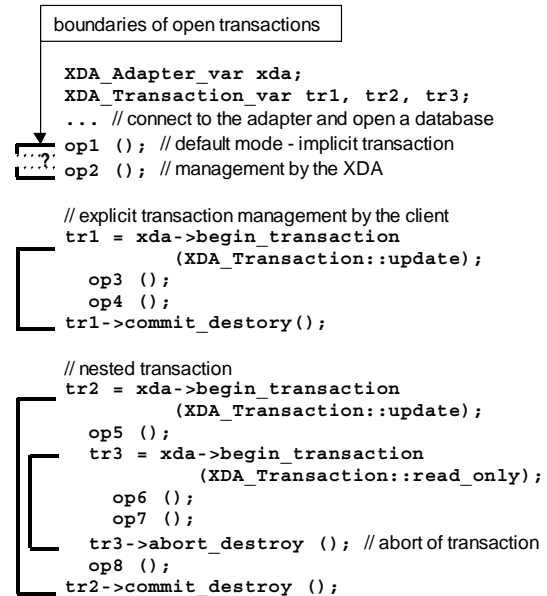


Figure 3 Example of the client code that uses different transaction control methods.

Transaction manager guarantees that access to persistent data will always take place within a transaction, but that does not mean that it will be started. For example, the transient objects do not need any transaction. The manager always tries to prevent the start of new transaction, and if it is possible to use an opened transaction. If the type of transaction is different, it will be closed and a new one will be started.

XDA's transaction manager works in the cooperation with ObjectStore's transaction manager. Since the latter works only for different UNIX processes, XDA requires that every CORBA client has its own server. This correspondence is controlled automatically by ORB daemon, registering the server in the *per-client-pid* mode. In the contrast to other techniques [2] this requirement reduces scalability of the server, but has better performance and reliability [15].

### Explicit transaction management in XDA

In the explicit transaction management responsibilities are split between the XDA transaction manager and the client. Usually after the client's explicit start of a new transaction the XDA switches off the automatic transaction control delegating this responsibility to the client. However, the client is able to switch it on later. For example, it might be useful in the situation where all nested operation calls should be done within separate sub-transactions.

If a client wants to manipulate the boundaries of transactions, it can use the corresponding methods from the `XDA_Adapter` and `XDA_Transaction` IDL interfaces. They allow the client to begin, commit or abort a transaction explicitly when it is necessary (Figure 3). The XDA also allows the client to nest transactions and support all transactional modes that ObjectStore provides: `read_only`, `update` and `abort_only`.

## 4.5 XDA's Architecture

Only two standard Object Adapters have been officially adopted by the OMG. The first was *Basic Object Adapter (BOA)*, which was specified in the first CORBA specification. Since the BOA specification is too general, each CORBA vendor filled in the details in proprietary way, making it hard to port server code from one CORBA product to another. For example, the BOA lacks clearly defined object activation mechanisms, which resulted in providing proprietary solutions. The release 2.3 of CORBA standard address that issues by replacement of the BOA by the *Portable Object Adapter (POA)* [14], which eliminates many limitations of the under-specified and uncompleted BOA.

The main reason behind POA being superior to BOA is that it allows programmers to build a consistent service for servant's management that will be portable between different ORB products. But none from both adapters is designed to deal with such typical for databases things as segments and transactions. When we have to deal with persistent objects, we should always care about two very important aspects of their persistence. It is how the objects will be saved in the database (segments) and how they will be accessed (transactions).

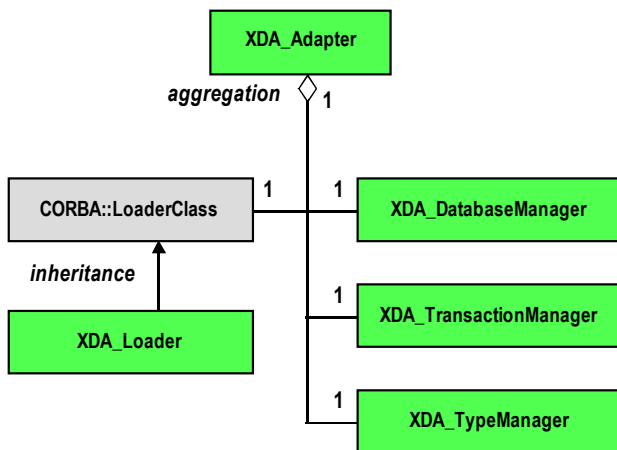


Figure 4 XDA class diagram.

We took the POA architecture as a base and tried to realize an adapter that will provide more services for manipulation with persistent objects. All XDA's functions are separated between independent components - *managers* (Figure 4). The core of the adapter consists of the four replaceable components: a *loader*, which retrieves persistent objects and creates new mediators when necessary; and three managers: a *database manager*, a *type manager* and a *transaction manager*.

The *database manager* is an instance of the `XDA_DatabaseManager` class. The manager keeps track of all mediators and controls the segmentation of persistent objects in the database.

The *type manager* is an instance of the `XDA_TypeManager` class. It administers meta-

information about the whole set of all available mediator classes and its relations to persistent counterparts.

The *transaction manager* is an instance of the `XDA_TransactionManager` class. It controls the start and the end of transactions.

The `XDA_Adapter` class represents the public facade for all managers, controls them and allows their replacement. This simplifies the use of the XDA. All that should be done is only to create an instance of the `XDA_Adapter` class.

## 4.6 Databases management

Additionally XDA provides for a CORBA client an IDL interface for the native ObjectStore functionality allowing the client to control the management of persistent objects. The interface includes methods for manipulation with usual ObjectStore objects such as databases, segments, objects and transactions. For example, using an IDL interface for ObjectStore database class `os_database` a client can create a root for every database object and then find the object in the database by that root (Figure 5). It is part of the original ObjectStore example that was rewritten for CORBA. In comments you can see related original code.

```

...
XDA_Adapter_var xda;
...
// connect to the server
// create a database
XDA_Database_var db1 = xda->create_database
("test.db", 0664, 1);

// open a transaction
XDA_Transaction_var tr1 = xda->begin_transaction
(XDA_Transaction::update);

// create an employee
Employee_var an_employee = ...;

// create a part
Part_var a_part = Part::narrow
(db1->create_object("part"));
a_part->part_id (111);
a_part->responsible_engineer (an_employee);

// create a root to the object
db1->create_root("part_root", a_part, 1);
...
// find object
a_part = Part::narrow
(db1->find_root ("part_root"));
...

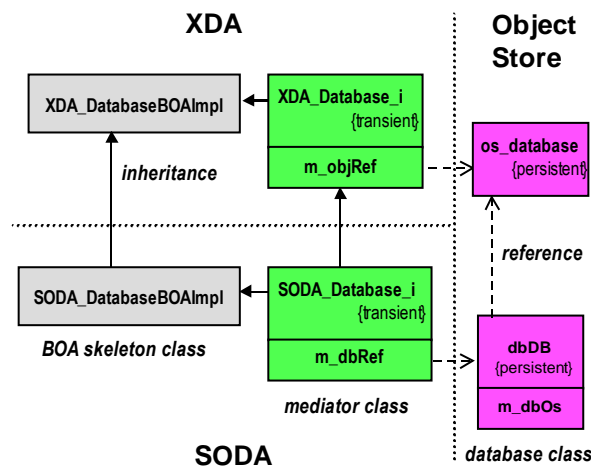
```

Figure 5 Example of the CORBA client code.

## 5. Evaluation of XDA

The XDA prototype is an almost complete adapter for the ObjectStore database system, providing wrapper developers with the full power of ObjectStore to define, manipulate, and share important application data. A big advantage of the XDA is the shorter time for the wrapper's development, since a programmer can partially reuse and tune the XDA functionality. The adaptability of XDA was evaluated by development of a CORBA wrapper prototype for the real object-oriented object-oriented spatial database *GeoStore*. The wrapper was

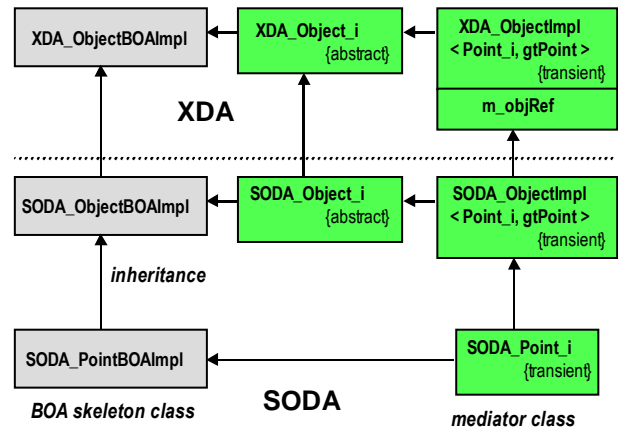
developed on top of XDA and called as *Spatial Object Database Adapter (SODA)* [15].



**Figure 6** XDA extension for a new mediator class for ObjectStore database.

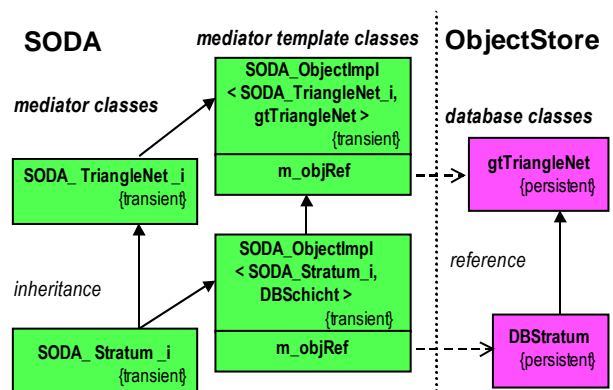
Through corresponding mediators XDA makes original ObjectStore classes (e.g. *os\_database*, *os\_transaction*) accessible for a CORBA client. However, sometimes a database can have its own classes that encapsulate the original ObjectStore classes and extend their functionality. In this situation XDA allows a server programmer to redefine some parts of his native functionality. For example, *GeoStore* encapsulates an *os\_database* class by a user-defined class to store an extended meta-information about a particular database. In the SODA prototype we defined our own class - *SODA\_Database\_i* that inherit and extend the functionality from the default XDA class *XDA\_Database\_i* (Figure 6).

To duplicate a proprietary scheme of objects management in the database XDA allows to extend functionality of *XDA\_Object* IDL interface. In SODA it was done defining a new interface *SODA\_Object* that inherit functionality of *XDA\_Object* class and replace them. New added functions need to be implemented in the corresponding classes (e.g. *SODA\_Object\_i* and *SODA\_ObjectImpl <...>* in Figure 7). Using mediator template class *XDA\_ObjectImpl*. the server programmer keeps implementations of the mediator classes (e.g. *SODA\_Point\_i*) free from the XDA dependent details.



**Figure 7** XDA extension for a new class for database entities.

One somewhat complex situation can arise if one database class inherits from one or more other database classes. For example, *DBStratum* inherits from *gtTriangleNet* and some other classes (Figure 8). Because of the multiple inheritance a reference to an instance of *DBStratum* class will differ from the same reference cast to the base *gtTriangleNet* class. Mediator template classes will solve this situation using different references. The inheritance relation between database classes will be transferred to the inheritance relation between corresponding mediator classes. So, in our example *SODA\_Stratum\_i* inherits *SODA\_TriangleNet\_i*. Mediator for *DBStratum* will have two instances of mediator template classes that will work with the same database object through different references (*m\_objRef*).



**Figure 8** Multiple inheritance between database objects.

## 6. Conclusions

In order to integrate Object-Oriented Database Management Systems (ODBMS) in CORBA environment, the mediating tier residing between both systems, beside standard tasks, must also perform some additional sophisticated functions. Since database schema evolution may be extremely time-consuming, ideally the CORBA-conformance should be achievable without any modifications in existed databases and applications. The most critical issues and requirements to the mediating tier were outlined and a new approach to non-destructive integration based on usage of transient mediators is presented. Several techniques aimed to improve the adaptability of object adapters and minimize the costs of the CORBA/ODBMS integration have been described. Those include the extensible component-based adapter architecture and flexible mediator templates that can be used for development of complex user-defined mediators. The "conservative" method for transaction management and conceptually new technique of mediator substitution based on the type information improved performance of the integrated system.

The presented techniques have been incorporated in proprietary adapter prototype - an eXtensible Database Adapter (XDA). Its extensibility and flexibility were tested by development of a CORBA wrapper for the object-oriented spatial database *GeoStore*. Construction of the wrapper shows that the use of XDA has substantially reduced the wrapper's development process and improves performance of CORBA/ODBMS system. The transience of mediators not only saved database space, but also allowed us to make the integration without any changes in the original database schema and without disturbing of existing local applications. The separation between the original database and CORBA components provided by XDA allows concurrently work with the database through both - CORBA and native database communication interfaces.

Evaluation of the XDA prototype shows that management of mediators and transaction are the most important adapter's tasks and have great influence on scalability and performance of the integrated system. They will remain one of the most important areas where more and more refined techniques will be applied [2]. Future developments of CORBA/ODBMS integration process will be certainly influenced by the modern rapidly evolving technologies such as XML and Java.

### Acknowledgment

This work would not have been possible without the excellent cooperation within the geo-scientific group of A.B. Cremers (University of Bonn) - special thanks to Oleg Balovnev and Andreas Bergmann for helpful comments on the XDA architecture.

## References

1. V. Amirbekyan. Integration of Object Databases with CORBA Environment. Ph.D. Thesis, Stanislaw Staszic University of Mining and Metallurgy, Cracow, Poland, 1999.
2. V. Amirbekyan, K. Zieliński. The Role of Transaction Management in CORBA/ODB Integrated Systems' Performance. In Proc. of the ACM Symposium on Applied Computing (SAC'2000), Como, Italy, March 2000.
3. O. Balovnev, A. Bergmann, M. Breunig, A.B. Cremers, S. Shumilov. A CORBA-based approach to data and systems integration for 3D geoscientific applications. In Proc. of the 8th International Symposium on Spatial Data Handling (SDH'98), Vancouver, Canada, July 1998, pp. 396-407.
4. IONA Technologies. Orbix Programmer's Guide. Version 2.3, IONA Technologies Ltd., Dublin, Ireland, October 1997.
5. IONA Technologies. Orbix+ObjectStore Adapter Programming Guide. Dublin, Ireland, April 1997.
6. IONA Technologies. Orbix Database Adapter Framework. Release 1.0. Dublin, Ireland, June 1997.
7. IONA Technologies. The IONA Orbix Cookbook: The Evictor, Dublin, Ireland, 1997.
8. Object Design. ObjectStore C++ API User Guide Release 5.0. Object Design Inc., 1997.
9. Object Database Management Group. The Object Database Standard: ODMG-93. R. Cattell (ed.), Morgan Kaufmann, 1994.
10. Object Management Group. CORBA 2.3/IIOP Specification. OMG formal document, 1999.
11. Object Management Group. CORBA services: Common Object Services Specification. OMG formal document, 1999.
12. J. Kleindienst, F. Plasil, P. Tuma. What We Are Missing in the Persistent Object Service Specification. In Proc. of the Workshop on Large Persistent and Distributed Systems (OOPSLA'96), 1996.
13. F. Reverbel, A. Maccabe. Making CORBA Objects Persistent: the Object Database Adapter Approach. In Proc. of the 3rd USENIX Conference on Object-Oriented Technologies and Systems (COOTS'97), Portland, Oregon, pp. 55-65, June 1997.
14. D. Schmidt, S. Vinoski. Object Interconnections. C++ Report, SIGS Publications, Columns 11-14, 1997-1998.
15. S. Shumilov, M. Breunig. Integration of 3D Geoscientific Visualisation Tools. In Proc. of the 6th EC-GI & GIS Workshop. The Spatial Information Society - Shaping the Future Lyon, France, June 2000.
16. P. Tuma: Persistence in CORBA, Ph.D. thesis, Charles University, Prague, 1997
17. G. Wiederhold. Value-added Mediation in Large-Scale Information Systems. in Robert Meersman and Leo Mark(ed): Database Application Semantics, Chapman and Hall, 1997, pages 34-56.