

Integration of GOCAD with an object-oriented geo-database system

M. Breunig¹, A.B. Cremers¹, R. Seidemann², S. Shumilov¹, A. Siehl²

1 Dept. of Computer Science III, University of Bonn, Germany

2 Institute for Geology, University of Bonn, Germany

e-mail: martin@cs.uni-bonn.de

1 Introduction

The wide-spread use of *GOCAD* [Mal92a, GC99] in the geosciences as an application for 3D-modeling makes it a significant tool for an open 3D-GIS environment. Therefore *GOCAD* should provide an open interface for an object-oriented data and methods exchange. Unfortunately, hitherto *GOCAD*'s 3D-objects are not managed by modern database technology. Therefore it is proposed to develop a coupling between *GOCAD* and *GeoToolKit* [BBC97, GT99] via object-oriented CORBA technology. *GeoToolKit* is an object-oriented geo-database kernel system for the support of 3D geological applications.

2 Advantages of using a DBMS to support 3D geological modeling

The necessity of an interconnection between geoscientific modeling software and databases systems is recognized and accepted since many years. Nevertheless, the integration of geoscientific modeling applications and databases, allowing a direct access from the modeling software onto the database system and vice versa, is not a standard within geoscientific applications.

Geoscientific data are heterogeneous and thus not easy to handle. To survey data in a specific region the most different methods are used, resulting in a wide variety of different types and formats of data. Moreover, these investigations can last over decades, entailing in a different quality of data, related to the date of surveying. A third aspect is the spatial distribution, which is rather irregular.

The storage and management of all these data in separate files is a common method, though it is hazardous. Usually, not only one geoscientific application, but a variety of applications are used for modeling, validation or visualization of a specific area. Therefore each of these applications interacts with each other by the exchange of data. At this, small scripts have to be written to convert a data set from one application to another. The representation of data into different formats leads to a redundant storage of data, which is (a) memory intensive and (b - and more important) might lead to working on the 'wrong' data, because the access to the most recent data/results depends on the prudence of the editor/s. At this, an additional aspect has to be mentioned: in most cases more than one editor is implementing a project. This might result in difficulties combining the results of each editor within a global result.

A database management system (DBMS) has several advantages which are relevant for the support of geological modeling. The *multi-user control* of a DBMS enables the simultaneous database access on the same data during the same time by different people. However, the long duration of geo-transactions would require long locks for those database objects that have been requested by the user query. That is why a *checkIn/checkOut mechanism* has to be provided for version control. One of the most attractive features of a DBMS for geologists, however, is the automatic *integrity control*. Spatial and temporal data can be tested against their geological plausibility. The integrity checks can be executed during the reading of the data into the database or during the execution of database queries. Of course, the loss of data during a system crash is also avoided by today's DBMS technology. Last, but not least, *security mechanisms* in different granularities guarantee data security.

The advantages of a DBMS show why it is a challenge to combine *GOCAD*'s advanced three-dimensional visualization and modeling facilities [Mal92b] with the functionality of a geo-database. In the following we present *GeoToolKit* which has been developed for the database support of geological data in the Lower Rhine Basin, Germany [ABB+98, BBC+98].

3 GeoToolKit

GeoToolKit foremost serves the efficient storage and retrieval of 3D-spatial objects within a database. To achieve this, *GeoToolKit* is tightly coupled with the object-oriented DBMS *ObjectStore*. Fig. 1 presents the data model of *GeoToolKit* for persistent spatial objects in an OMT-like notation [Rum91]. An abstract *SpatialObject* class - the root of the spatial object class hierarchy - specifies an interface, which is to be inherited by all concrete spatial objects. A concrete class provides a representation of the object as well as an implementation of the geometric functions. Thus *GeoToolKit* guarantees that any spatial object has at least the functionality of the most general class. Usually, concrete classes have additional functions which are peculiar to this geometric entity.

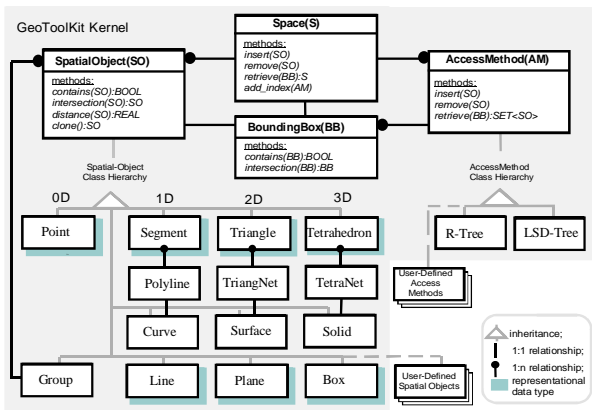


Fig. 1 Spatial object data model of *GeoToolkit*

Currently, *GeoToolkit* offers classes for the representation and manipulation of simple (point, segment, triangle, tetrahedron) and complex (curve, surface, solid) 3D spatial objects. The *GeoToolkit* class hierarchy is complete: any spatial object can be modeled either directly by one of the built-in spatial classes or as a composition of these classes within a group. A *Group* is a heterogeneous collection of spatial objects, which are further treated as a single object.

Any newly defined data type customized according to the conventions of *GeoToolkit* can be included in the class library for further re-utilization. A new type can inherit a representation and a functionality from one of the embedded classes redefining only functions which get, for example, a more efficient implementation. However, sometimes it may be beneficial to create a completely new type, instead of re-using *GeoToolkit's* types. In this case the class is created as a specialization of the most general *Spatial-Object* class and must provide the complete geometric functionality, which is defined in the abstract root class.

In most cases 3D geoscientific models built up by surfaces and bodies are characterized by extremely complex non-regular shapes, which are usually decomposed into a set of adjacent spatial primitives. Correspondingly, *GeoToolkit* offers classes *Polyline*, *TriangleNet* and *TetraNet* as default representations for the abstract classes *Curve*, *Surface* and *Solid* respectively.

GeoToolkit distinguishes between geometric predicates returning *true* or *false* (*equal*, *intersect*, *contains*) and geometric operations (*intersection*, *division*) returning a new spatial object. Geometric operations are algebraically closed: the result is a new spatial object, which can be stored in the database or used as an argument in other geometric operations. Currently, *GeoToolkit* provides the following operations: (1) intersection of spatial objects, (2) partitioning of spaces and spatial objects by a plane, (3) clipping a part of a space by means of bounding box, and (4) equality and containment checks.

Space is a special container class capable of efficient retrieval of its elements according to their location in space, which is exactly specified either as a point or as a bounding box. A space serves both as a container for spatial objects and as a program interface to the spatial query manager which is invoked by *retrieve* member functions. To provide an efficient retrieval a space utilizes spatial indexes. If the user is not satisfied with the embedded spatial indexes he can customize his own indexing method as a specialization of the abstract *AccessMethod* class.

We can summarize the advanced features of *GeoToolkit* as follows:

1. management of 0D-3D spatial data types with the object-oriented DBMS ObjectStore
2. efficient spatial access methods for large sets of 3D-objects
3. 3D DB queries supported by a 3D-visualisation component
4. input/output functions for *GOCAD*-objects
5. topology-based spatial representations (simplicial complexes)

The question to be answered in the next paragraph is how *GOCAD* can use these features and how the coupling can be technically realized.

4 Integrating GOCAD with GeoToolkit: CORBA based distributed architecture

A prototype implementation uses CORBA for transparent network access between different computation platforms such as Sun Solaris, SGI IRIX and Windows NT. Covered by a wrapper *GeoToolkit/GeoStore* could be regarded as a distributed 3D/4D geo-database [BBC+97]. The wrapper enables the easy access to the DBMS from other remote CORBA clients. Using CORBA clients may concurrently access the database, but they need CORBA compatibility. (Fig. 2).

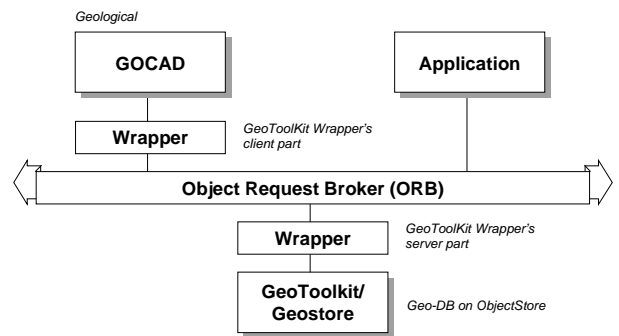


Fig. 2 General schema of 3D-GIS architecture.

The main advantages of a coupling between *GOCAD* and *GeoToolkit* concern:

1. The geological *GOCAD* objects can be stored in and retrieved from the object-oriented DBMS
2. The geological *GOCAD* objects stored in the DBMS can be retrieved at any time concurrently by other applications from the object-oriented DBMS, if they are compatible to CORBA
3. The spatial access methods of *GeoToolkit* can be used from *GOCAD*
4. The results of database queries can be visualized and processed in *GOCAD*

5 CORBA/OODBMS integration

An approach of CORBA/OODBMS integration was made regarding Orbix [IT97pg] as an ORB implementation and ObjectStore [OD97] as an OODBMS. The approach was evaluated and proved by the development of an ODA prototype and a

specialized Spatial ODA (SODA). For a more detailed description see [Shu99].

5.1 Existing approaches

CORBA does not directly support the persistence of objects, but it provides the integration facilities with different database systems. The last CORBA 2.0/IIOP specification [OMG98c] defines two standard ways: *Persistent Object Service* and *Object Database Adapter*.

5.1.1 Persistent Object Service

The POS is specified in a very general way [OMG98s] and complex to use. Some gaps within the specification were found by some research groups that tried to implement it [KPT96]. Consequently, there is no complete implementation of POS and due to this fact this approach will not be taken into account.

5.1.2 Object Database Adapter

The ODA is an object adapter for objects stored in a database. It does not completely replace the Basic Object Adapter (BOA), but represents its extension that depends from a particular database system. The ODA is responsible for the managing of the persistent state of implementation objects and aimed to simplify their programming. The implementation objects are stored in the database and are dynamically loaded into the memory of the server when a request arrives. Unlike the BOA, the ODA does not assume that every implementation is in the memory and should be prepared to deal with other typical database features, e.g. locks and transactions, since persistent objects are generally shared.

The most important aspects of the ODA functionality are:

Method of database objects binding - This is how database objects will be made available from the CORBA environment.

Object reference representation - The difference between CORBA references to transient and persistent objects is that the latter contain an information identifying a persistent object and its state in the database. It is a key through which the state can be found.

Object activation - A normal CORBA object should be activated by the object adapter before the first request for that object is delivered. BOA always makes a new instance and in standard CORBA object activation and instantiation are synonymous. Different from this, persistent objects are instantiated once and can be subsequently loaded (*activated*) from the database.

Object deactivation - The CORBA objects that were previously loaded from the database and are not longer needed should be removed from the server's address space to avoid memory overloading.

Transactions management - All persistent objects are accessible only within an open transaction. When a persistent object's method invocation arrives to the server the ODA automatically starts a new transaction and commits (or aborts) it immediately after the results are returned to the client.

Databases management - The ODA provides an IDL interface for the native database functionality. A CORBA client is able to perform the elementary database functionality, e.g. objects creation/deletion, management of transactions boundaries.

One of the reasons to choose Orbix as an implementation platform was the existence of the ODA implementation for ObjectStore database system - *Orbix/ObjectStore Adapter (OOSA)*

[IT97oosa]. The OOSA helped to considerably reduce the wrapper development costs in the initial stages. However, after several experiments with the prototype, the OOSA turned out to have serious limitations and drawbacks:

1. It stores OOSA-specific objects in the database resulting in the growth of the databases;
2. It cannot provide referential integrity for concurrent remote/local access;
3. It does not implement an IDL interface for data manipulations facilities of ObjectStore;
4. It does not allow to open a database only for *read* access.

5.1.3 Wrapper approaches

Apart from the standard approaches there are many non-standard approaches. The most of them do not assume the usage of some special software and require a programmer intervention. Usually it is a construction of the middle-tier layer between CORBA clients and a database that will wrap an original database interface and provide an equivalent CORBA compatible interface. We will name the layer as *wrapper* and approaches based on this principle as *wrapper approaches*.

A primary strength of object-oriented databases is their ability to model complex objects and inter-relationships among them. In the common case of an OODBMS that adds database features to the C++ like ObjectStore, database entities are instances of usual C++ classes. In addition to the state they have also the methods. To make them available to the clients as usual CORBA objects, a wrapper should wrap an interface of every database class by an equivalent CORBA compatible class. This class implements all their methods by their equivalents from the database class 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 skeletons and database objects converting data parameters and delegating all function calls in the both directions [Wie97]. Because of this basic function we will name such CORBA objects *mediators* and correspondingly such an approach a *mediator approach*.

5.1.4 Resume

A big advantage of the ODA approach over the wrapper approaches is a short time of the server development, since a server programmer can reuse the ODA functionality. In contrast, the wrapper technique requires more programmer intervention. Sometimes, however, it can be beneficial to use a "roll-your-own" approach, particularly where integration of existing applications with CORBA is required. For example, in opposite to the OOSA, a wrapper can make their mediator objects transient and do not waste the database. The whole database size becomes three times smaller as with the OOSA.

The wrapper approach turned out to be more complex, but also more flexible that gives the programmer a number of options to achieve a better overall performance. The worse OOSA performance was attributed to hidden expenses, such as the assigning of a persistent object pointer to an auxiliary CORBA TIE object, which is dynamically instantiated whenever a persistent object is loaded into the server memory.

5.2 Proposed approach

The Object Database Adapter (ODA) approach has a good basic idea of the additional software module that can be reused. From the other hand an ODA is not as flexible as a wrapper in cases where an integration with existing databases is required. Therefore, in our approach we took the ODA approach as the base and combined it with the mediators approach.

Since a database schema evolution may be extremely time-consuming, ideally the CORBA-compliance should be achievable without changing where and how the data are stored. Therefore the database objects should be made accessible to CORBA clients through corresponding transient mediators. The data members and the layouts of the persistent database objects remain private, only the interfaces (a set of methods) are made public at the expense of the implementation of equivalent IDL interfaces for the complete database schema. Using mediators, developers can encapsulate both data and data manipulation methods to give applications a runtime dynamism and self-contained intelligence that is difficult to achieve using other methods.

The transience of mediators has several advantages that have their own price. Every CORBA object consists from a database object and the corresponding mediator. Since the mediators are transient, they do not waste an original database and allow the ODA to support all kinds of transactions provided by the OODBMS. The price of this improvement is a responsibility to control the life circle of mediators – their creation and deletion. Care should be taken to avoid the creation of multiple mediators tied with the same database object.

In the proposed ODA architecture the mediators are controlled by specialized *managers* - pluggable ODA modules. The core of the adapter consists of the following five objects: one *wrapper object*, which is the public facade of the four replaceable components – a *loader*, who retrieves persistent objects and creates new mediators when necessary and three managers, a *database manager*, a *type manager* and a *transaction manager*.

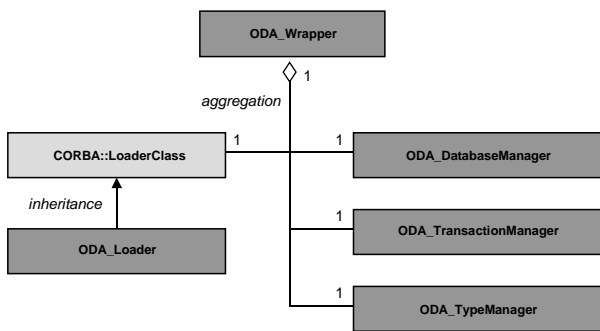


Fig. 3 ODA class diagram.

The *database manager* is an instance of the ODA_DatabaseManager class. The manager keeps track of all mediator for ObjectStore databases – instances of the ODA_Database class.

The *type manager* is an instance of the ODA_TypeManager class. It administers the whole set of all available mediator classes.

The *transaction manager* is an instance of the ODA_TransactionManager class. It controls the start and the end of ObjectStore transactions. In the automatic mode a client does not control transaction boundaries and the manager independently decides when and what kind of transaction should be started. It is

also responsible to control the mediators for all ObjectStore transactions – instances of the ODA_Transaction class.

The wrapper is an instance of the ODA_Wrapper class and always holds a reference to one transaction manager object, a reference to one database manager object and a reference to one type manager object. The server programmer can replace them by registering an instance of his/her own class. The code sample in Fig. 7 shows how to define a new database manager.

5.3 Method of database objects binding

Original database objects are made accessible in the CORBA environment with the help of the corresponding *transient* CORBA objects – mediators, that also act as local database clients (Fig. 4). For each original ObjectStore C++ a server programmer should create the corresponding CORBA C++ mediator class with an equivalent IDL interface. In this interface the original data structures from the ObjectStore C++ data model should be represented by appropriate structures in the CORBA IDL model. Each mediator object represents only one database object.

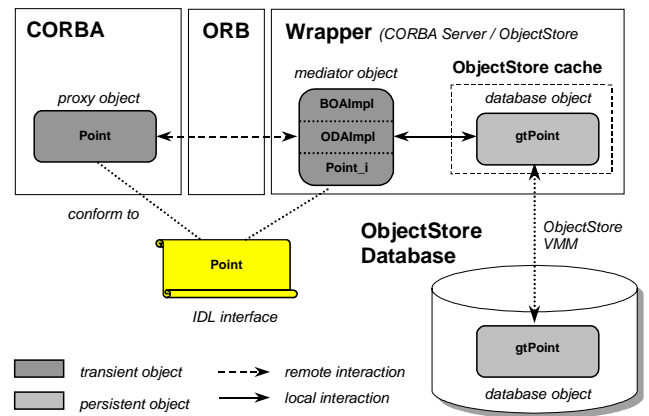


Fig. 4 ODA binding approach.

Orbix offers two ways to implement IDL interfaces, one using the C++ inheritance and known as the BOAImpl approach, the other using a delegation and known as the TIE approach [IT97pg]. The first approach was preferred to the second, because it is more simple for the ODA to control one mediator object as two objects: mediator plus its TIE. The binding between mediators and ODA core is organized with the help of a technique that is similar to the BOAImpl approach. The ODA defines a special template class ODA_ObjectImpl that implements an ODA specific functionality. 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 ODA-related behavior and concentrate it in the template. This makes the clear separation between a functionality of mediators that related to the ODA and a functionality that related to the mediation. Additionally the template implements the operations from the ODA_Object IDL interface that are common for all ObjectStore objects thus helps the programmer to speed up the whole development process. The complete schema of relations between these classes is shown in Fig. 5.

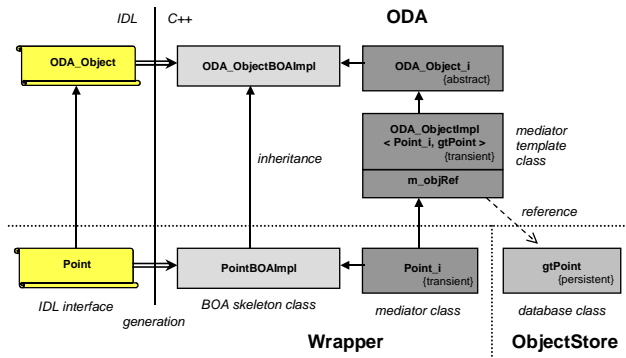


Fig. 5 Implementation of the user-defined mediator.

In contrast to the OOSA, mediator objects are made transient and do not waste the database space. The client request for the creation of a new persistent object results in the creation of only two objects: a transient mediator object and a persistent database object. The transience of mediators not only save the database space, but provides a complete separation between the original database and communication components. This separation allows to work with the database not only through the new CORBA interface, but also through an original C++ interface. Moreover, following this method any already existing data store with a compatible schema at any time can be used through CORBA-interface. Referential integrity of the data shared by local and CORBA-conform applications are guaranteed by the use of the native ObjectStore transaction control mechanism.

Some methods of database classes as the result value return a reference to a new or an existed database object. However, CORBA compatible methods of mediators should return the reference to the corresponding mediator object. The ODA helps in this situation. It controls the creation/deletion of mediators and provides an effective mediator searching service. This service is provided by the database manager. The back references to the mediators are stored in the hierarchical structure of transient dictionaries of references (Fig. 6). The database manager dictionary keeps track of the all mediators for instances of `os_database` class, they in their part have the dictionaries that store the references to the all mediators for ObjectStore segments (`os_segment`) that belong to the database. Every segment mediator has a dictionary that keep references to the all mediators for user defined objects from the related ObjectStore segment.

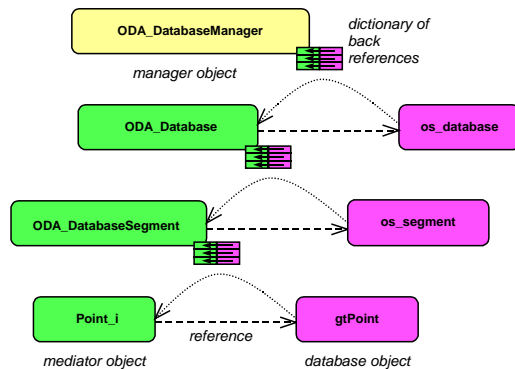


Fig. 6 Hierarchical organization of transient dictionaries of back references.

In such a dictionary in which keys are stored as references to database objects it is straight forward to find a reference to the corresponding mediator. Since the pointers to database objects are unique, the retrieval from the dictionary is very efficient. The database manager provides universal functions that return the reference to the mediator for a particular database object. If the mediator does not exist, than it will be automatically created. During the search the database manager recognizes the database and the segment, the database object belongs to, and goes through the hierarchical organization of the dictionaries to the target mediator. But usually it is known to which ObjectStore-database or -segment the resulted database object belongs. Therefore it is possible to speedup the search procedure directly calling the corresponding mediator that is responsible for the database or the segment.

5.4 ODA IDL interface

Usually in the ODA and wrapper approaches the client does not have such features. The ODA allows the client to control the management of persistent objects providing an IDL interface for the native ObjectStore functionality. It includes the methods for manipulation with the usual ObjectStore objects such as databases, segments, objects and transactions. For example, persistent objects in the ObjectStore database are identified by named root objects. Using the `ODA_Database` interface (Fig. 7) a client can create a root for every database object and than find the object by the root.

```
interface ODA_Database {
    ...
    // database maintenance operations
    void close ();
    void destroy ();
    void evict ();
    ...
    // segments maintenance operations
    ODA_DatabaseSegment create_segment ();
    ...
    // roots maintenance operations
    void create_root (in string name,
                    in ODA_Object objPtr, ...);
    ODA_Object find_root (in string type);
    ...
    // objects maintenance operations
    ODA_Object create_object (in string type);
    ODA_Object create_param_object (in string type,
                                    in any param);
    ...
};
```

Fig. 7 Example of ODA IDL interface.

5.5 Spatial ODA (SODA) as an ODA extension

The ODA prototype is almost a complete adapter for the ObjectStore database system. Since ObjectStore uses the principle of the persistent C++, for every new database a server programmer should implement only corresponding mediator classes. The ODA speedups the development process providing mediators for the original ObjectStore classes. However, sometimes they are encapsulated by some user-defined classes. For example, *GeoStore* [BBC+97] – a specialized 3D-4D spatial database encapsulates `os_database` class by the `dbDB` class to store an extended meta-information about a particular database. Since the ODA architecture consists of plug-in components, the server programmer can easily extend it. The programmer can write his/her own classes (e.g. `SODA_Database_i`) that inherit and extend the functionality from the default ODA classes (e.g. `ODA_Database_i`) (Fig. 8). This example is taken from the *Spatial Object Database Adapter* (SODA) prototype that was developed for *GeoStore* on top of the ODA.

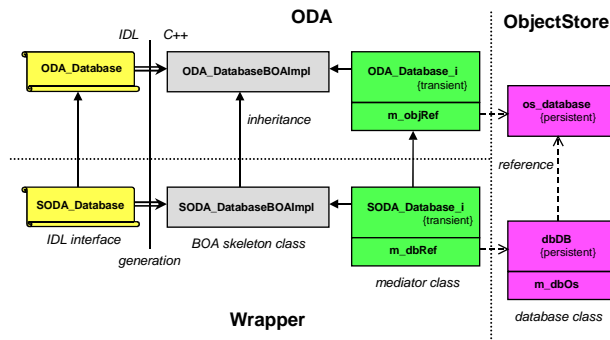


Fig. 8 Example of ODA extension for the database mediator class.

5.6 Open problems: Passing Objects by Value

It would be certainly good to be able to automatically generate the mediator's code from the specifications of corresponding database classes, but the difference between both data models makes this task not trivial. For example, the struct in IDL and C++ are not similar constructs. The first contains only data, cannot have operations and cannot inherit from another struct. So the only object oriented type in IDL is the interface. But it is also not an equivalent for the classes as they are in C++. CORBA won't return local objects from the server, it will return proxy objects that allow you to operate on remote objects that live on the server. Therefore a CORBA object can never be passed by value as a usual C++ object.

The usual approach for passing an object by value is to flatten its state into a struct or their sequence and pass that through an IDL operation call [Swa97]. The IDL struct is a poor substitute for object by value, because structs cannot have operations or inheritance. Often the client needs also corresponding methods for equivalence checking, comparison, conversion etc. For complex data structures often it is much more difficult to decide what kind of the representation to take, e.g. the sequence of references or the sequence of values.

Another possibility to pass objects by value is to use an ORB specific approach "opaque types" provided by Orbix. This is a non-standard extension to the IDL syntax and requires a special IDL compiler. A feature of this approach has minimal changes to the IDL syntax. An opaque type name is used as a place holder in the IDL compiler generated source files. Since the IDL does not describe the type at all, the details of the by value transfer are defined in the implementation code. The application programmer is required to write the appropriate marshalling code to insert and to extract the type from the CORBA::Request object. In C++ this is usually implemented as a class with three stream operator functions. Furthermore someone has to maintain compatible implementations of the opaque type across all platforms and languages.

Significant limitation that does not allow to use this approach for our needs is the mode of the passing. Opaque types are always passed by the value and it isn't possible to express a relationship between an interface and an opaque type. Alternatively, the state information could be declared within the IDL supporting the code automatically generated by the IDL compiler. This is the approach used in the joint submission to the RFP [BII+98].

6 Evaluation at an Example from Lower Saxony

6.1 Geological Objectives

The geoscientific goal of this research is the development and implementation of a method, which allows the computer aided construction of geological maps. At this, the geological map is generated by the cut of a 3D stratigraphical model with a digital elevation model (DEM) [Rap89][Sie93]. The verification of the resulting map gives geoscientists improved means to check the capability of the 3D model and to alter it accordingly.

Moreover, the 3D model might serve as input for subsequent investigations, e.g. numerical simulations on kinematic or dynamic models, by which an additional control of the generated 3D model is possible. In the study presented here, besides *GOCAD*, an additional, geophysical application (*IGMAS*, [GL88]) is used to provide further constraints for an iterative revision of the 3D model and thus a succeeding modification of the geological map.

For this reason, the 3D stratigraphic model, which is built up from spatially distributed and heterogeneous data, is the basis for all subsequent steps. As mentioned, the 3D model is not static or fixed, once it is generated, but it is subject to a continual change. These changes might become necessary because (a) additional data are incorporated or (b) other methods to verify the 3D model are applied.

For this reason, the integration of *GOCAD* as a geoscientific modeling tool with the OODB *GeoToolKit/GeoStore*, serves the strong needs for a consistent handling of both, data and 3D models (and parts of it) during the geoscientific modeling process.

The advantages which arise by the use of the OODB are manifold, concerning not only the management of data and models but the management of multi-user access as well. At this, the most important aspects are (a) a central management of all data, (b) a simplified incorporation of new data, resulting in an implicit actualisation of the data, instantly visible to all participating editors and (c) update support of queries of data and models.

However, to allow a proper use of the OODB, a strong cooperation between computer- and geo-scientists is necessary. They have to build up a data model, which matches the needs of the geoscientists occurring during the construction of the 3D model (Fig.1) and has to be general enough to be extensible to allow the integration of additional (a) data types and (b) applications.

From the user's viewpoint the integration should match two needs. On the one hand the usage of the integrated system has to be as simple as possible and on the other hand methods, which allow retrieval operations on the OODB, have to be enabled and specified inside of *GOCAD*. At this, the most important DB queries are those on the existence of spatial objects like points, lines, surfaces or bodies concerning their spatial or temporal definition and their direct visualization. In case of spatial queries an area (bounding box) is specified wherein all objects are given to the geoscientific application (*GOCAD*). In the latter case, we have to deal with different definitions of time. First, there is the geological time, which the data include as information and which is used for modeling stratigraphic boundary surfaces, e.g.. Other terms of time can be defined as that time the data were obtained, that time the data were incorporated within the DB or that time a 3D structural model or a part of it is generated. There may be many others different terms of time not mentioned here.

Term of time	Type of time	Possible query
geological time	Interpreted	Boundary surface of base upper jurassic
date of incorporation	real	Data incorporated after 01.01.2000
date of generation	real	Data/objects generated before 04.12.1998
date of obtainment	real	Well information younger then 13.10.1965

Tab. 1 Different terms of time.

As expected, all time depended queries occur while generating a 3D model and are mostly combined with spatial queries. Moreover there are not only queries on a specific area or term of time, but on the geometry (e.g. retrieve all closed surfaces or points) or the topology (e.g. retrieve all stratigraphic surfaces which are totally surrounded by faults) of objects. The geometrical and topological queries are combined with spatial and/or time dependent queries as well. In general, most queries on the OODB during the generation and validation of the 3D model represent combination of at least 2 types of queries.

6.2 Modeling the Study Area

Within the scope of the research a 3D stratigraphical model of a region located W of the city of Hannover, Germany (Fig.9) was generated. In doing so, the integrated system was iteratively developed and stepwise evaluated.

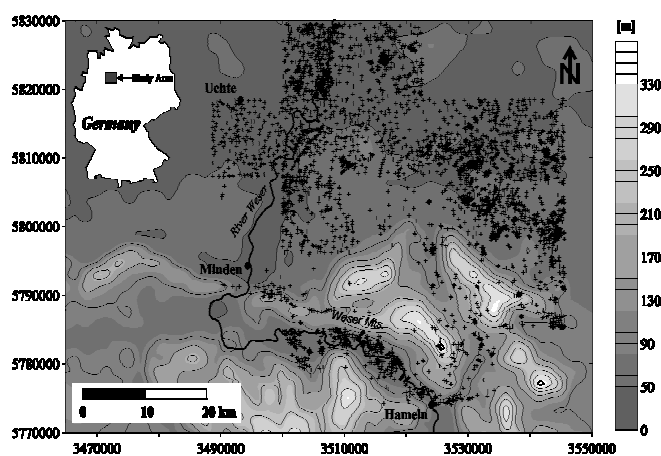


Fig. 9 Location and Morphology of the study area.

A large amount of mostly different types of data are available throughout the area. Focussing on geological modeling, information of more than 5000 wells in digital form (DASCH-format [PVV91]), 14 digitized contour maps and 4 cross sections [Koc96] as well as a Digital Elevation Model (DEM) were available and used.

An iterative geophysical validation of the 3D model, which was mentioned above, was performed on gravity and susceptibility data. However, the description of the iterative validation of the 3D model is not subject of this paper and can be found at [BCG+99] and [SG99].

7 Summary and outlook

The coupling between *GeoToolKit*, an object-oriented 3D-geo-database kernel system with the 3D-modeling tool *GOCAD* has been proposed.

The presented mediator-based Object Database Adapter (ODA) architecture provides a natural and effective way to providing distribution support for existing object-oriented databases. It allows to make existing databases CORBA-compliant without changes in the original database schema and disturbing of existing applications. The approach was proofed by the ODA prototype development, ODA for short. The ODA adaptability was demonstrated in the example of the Spatial ODA (SODA) – a specialized ODA implementation for GeoStore – a 3D-4D spatial database. The reuse of the ODA in SODA drastically reduced the amount of the server code. CORBA developers can easily extend or replace the supplied ODA functionality and save time by not re-inventing the wheel.

In our future work we also intend to enable other applications to access on *GeoToolKit's* data. We will also couple *IGMAS*, a geophysical 3D-modeling tool developed at the Freie Universität Berlin, with *GeoToolKit* to provide the object exchange between *GOCAD* and *IGMAS* with an open geo-database.

8 Acknowledgements

We like to thank the Deutsche Forschungsgemeinschaft (DFG) for the financial support within the joint project on *Interoperable Geo Information Systems* (IOGIS) and our partners BEB Erdgas und Erdöl GmbH, Mobil Oil AG, Preussag, Winthershall AG and NLfB for providing data.

9 References

- [ABB+98] A. Alms, O. Balovnev, M. Breunig, A.B. Cremers, T. Jentzsch, A. Siehl. Space-time modelling of the Lower Rhine Basin supported by an object-oriented database. In: *Physics and Chemistry of the earth*, Vol. 23, No. 3, p. 251-260, 1998.
- [BBC97] O. Balovnev, M. Breunig, A.B. Cremers. From GeoStore to GeoToolKit: The second step. In: *Proceedings of the 5th Intern. Symposium on Spatial Databases*, Berlin, LNCS No. 1262, Springer, Berlin, p. 223-237, 1997.
- [BBC+97] O. Balovnev, M. Breunig, A.B. Cremers, S. Shumilov. *GeoToolKit: Opening the Access to Object-Oriented Geodata Stores*. International Conference on Interoperating Geographic Information Systems (InterOp'97), Santa Barbara, CA, USA, December 1997. <http://www.informatik.uni-bonn.de/~shumilov/publications/papers/interop97/abstract.html>
- [BBC+98] O. Balovnev, M. Breunig, A.B. Cremers, M. Pant. Building geo-scientific applications on top of GeoToolKit: A case study of data integration. In: *Proceedings of the 10th Intern. Conference on Scientific and Statistical Database Management*, IEEE Computer Society, Los Alamitos, CA, Capri, 1-3 July, p. 260-269, 1998.

- [BII+98] BEA Sytems, International Business Machines Corporation, Iona Technologies, Netscape Communications Corporation, Novell, Visigenic Software. *Objects by Value Joint Revised Submission*. orbos/98-01-18, 1998.
- [BCG+99] M. Breunig, A.B. Cremers, H.-J. Götze, S. Schmidt, R. Seidemann, S. Shumilov, A. Siehl. First Steps Towards an Interoperable GIS - An Example From Southern Lower Saxony. In: Physics and Chemistry of the Earth, Vol. 24, No. 3, S. 179-190, 1999.
- [GT99] GeoToolKit Technical Documentation, 1999. <http://www.cs.uni-bonn.de/III/projekte/d4/gtk/index.htm>
- [GC99] GOCAD Technical Documentation, 1999. <http://www.ensg.u-nancy.fr/GOCAD>
- [GL88] H.-J. Götze, B. Lahmeyer, B. Application of Three-Dimensional Interactive Modeling in Gravity and Magnetics, In: Geophysics, Vol. 53, No. 8, p. 1096-1108, 1988.
- [IT97oosa] IONA Technologies Ltd. *Orbix+ObjectStore Adapter programming guide*. April 1997.
- [IT97pg] IONA Technologies Ltd. *Orbix Programmer's Guide*. Version 2.3, October 1997.
- [KPT96] J. Kleindienst, F. Plasil, P. Tuma. *What We Are Missing in the Persistent Object Service Specification*. OOPSLA'96 Workshop on Large Persistent and Distributed Systems, 1996. <http://nenya.ms.mff.cuni.cz/thegroup/rep/work0908.ps.Z>
- [Koc96] F. Kockel, et al.. *Geotektonischer Atlas von Nordwest-Deutschland*, BGR, Hannover, 1996.
- [Mal92a] J.L. Mallet. GOCAD: a computer aided design program for geological applications. In: A.K. Turner (ed.), *Three-Dimensional Modeling with Geoscientific Information Systems*, NATO ASI 354, Kluwer Academic Publishers, Dordrecht, p. 123-142, 1992.
- [Mal92b] J.L. Mallet. Discrete smooth interpolation in geometric modelling, *Computer Aided Design*, 24, 4, p. 178-192, 1992.
- [OD97] Object Design Inc. *ObjectStore C++ API User Guide*. Release 5.0, Object Design Inc., 1997.
- [OMG98c] Object Management Group. *CORBA 2.0/IIOP Specification*. OMG formal document 98-07-01. <http://www.omg.org/corba/c2indx.htm>
- [OMG98s] Object Management Group. *CORBA services: Common Object Services Specification*. OMG formal document 98-12-09. <http://www.omg.org/corba/csindx.htm>
- [PVV91] H. Preuss, R. Vinken, H.-H. Voss. *Symbolschlüssel Geologie.- Symbole für die Dokumentation und automatische Datenverarbeitung geologischer Feld- und Aufschlußdaten.- 328 p., NLF/BGR, 3rd ed, Hannover, 1991.*
- [Rap89] J.F. Raper. The 3D geoscientific mapping and modelling system: a conceptual design. In: Raper, J.F. (Ed.) *Three Dimensional applications in geographical information systems*, pp11-20. London: Taylor and Francis, 1989.
- [Rum91] Rumbaugh et al. *Object-Oriented Modeling and Design*. Prentice Hall, New Jersey, 500p., 1991.
- [Sie93] A. Siehl. Interaktive geometrische Modellierung geologischer Flächen und Körper. In: *Die Geowissenschaften*, No.10-11, p. 342-346, 1993
- [SG99] S. Schmidt, H.-J. Götze. Integration of Data Constraints and Potential Field Modelling - An Example From Southern Lower Saxony. In: Physics and Chemistry of the Earth, Vol. 24, No. 3, p. 191-196, 1999.
- [Shu99] S. Shumilov. *Mediator-based Object Database Adapter Architecture*. Proposed to: International Symposium on Distributed Objects and Applications. Edinburg, Scotland, September 1999. <http://www.informatik.uni-bonn.de/~shumilov/>
- [Swa97] M. Swainston-Rainford. *CORBA Objects by Value*. WWW page, 1997. <http://www.mjs-r.com/RESOURCE/idldesign/objectsb yvalue.html>
- [Wie97] G. Wiederhold. *Value-added Mediation in Large-Scale Information Systems*. In: Robert Meersman and Leo Mark (eds.): *Database Application Semantics*, Chapman and Hall, p 34-56, 1997. <http://www-db.stanford.edu/pub/gio/1995/DS6CH.ps>