Building Geo-Scientific Applications on Top of *GeoToolKit*: a Case Study of Data Integration[1]

Oleg Balovnev, Martin Breunig, Armin B. Cremers, Marcus Pant
Institute of Computer Science III, University of Bonn

Abstract.
Today's geo-information systems are historically grown products which are hardly extensible to meet the requirements imposed by 3D/4D-modeling. The next generation GISs should benefit from modern software engineering technologies. A component-based design encourages a fast „assembly" of applications from high-level software building blocks. Following this approach a complex general-purpose geo-information system can be substituted by a family of specialized subsystems which due to the common design basis are open for mutual data exchange. We introduce GeoToolKit - a component software intended for the development of 3D/4D geo-scientific applications. We also present our experience in building different types of geo-scientific applications on top of GeoToolKit. We show that common data types inherited by diverse applications from the GeoToolKit spatial class hierarchy create an excellent basis for the database-level integration of heterogeneous geo-scientific data.

Introduction

Today's geo-information systems are complex „historically grown" software packages. Application programming within such systems can be extremely complicated. Internal data structures and functions are often completely hidden from the user. As a result they are hardly extensible to meet the requirements imposed, for instance, by 3D/4D-modeling. The next generation GISs should benefit from modern software engineering technologies. Among the most promising technologies is a component-based software design. Software building blocks with correlated interfaces encourage a fast „assembly" of special-purpose applications for particular domains. A necessary basis for the integration of such components can be provided by an object-oriented programming environment. Application-specific components can be customized and reused for the development of applications in related domains. Following this approach a general purpose geo-information system can be substituted by a family of specialized subsystems which due to the common design basis are open for the inter-communication and mutual data exchange.

GeoToolKit

Within the collaborative research center SFB350 at the University of Bonn we have developed a component software called *GeoToolKit* [1] which is intended to facilitate the design and implementation of 3D/4D geo-applications. The idea was to provide for an application developer a range of geo-oriented software building blocks involving DBMS-based spatial data maintenance, special support for efficient spatial retrieval, communication, visualization and graphical interfaces which the user could assemble in a ready-to-use application. Therefore *GeoToolKit* is not a GIS-in-a-box package - it is rather a library of C++ classes that allows the incorporation of spatial functionality within an application under development. It is primarily oriented on software engineers with the C++ experience involved in the development of special-purpose geo-applications which can be hardly modeled within standard GISs.

GeoToolKit evolved on the basis of the experience we gained within the collaborative research center while developing diverse geo-scientific applications. GeoToolKit's architecture and functionality was initially inspired by requirements of *GeoStore*[2] - an information system for the database support of 3D geological modeling. At the same time we tried to make a toolkit general enough to be used in the development of a possibly wide range of applications. However, it was obvious for us that a toolkit should be more than just a set of empty interface specifications, i.e. it should do something. *GeoToolKit* always provides at least one complete implementation for all object types and functions specified in the interface.

Persistent Spatial Data Management

GeoToolKit addresses foremost 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 *GeoToolKit's* data model for persistent spatial objects in an OMT-like notation [3].

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

[1] This work is funded by the German Research Foundation (DFG) within the collaborative research center SFB350 „Interactions between and Modeling of Continental Geosystems" at the University.of Bonn

provides a representation for 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.

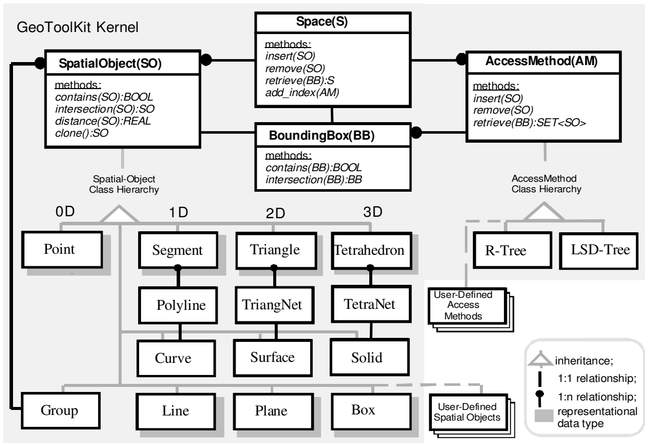

Fig. 1 GeoToolKit's spatial object data model

Currently GeoToolKit offers classes for the representation of and manipulation with 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 *GeoToolKit* conventions can be included in the class library for the 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. Sometimes, however, it may be beneficial to create a completely new type. Then, instead of re-using *GeoToolKit*'s types, an application developer creates his own class as a specialization of the most general *Spatial-Object* class. In this case he is responsible to provide the complete geometric functionality, which is defined in the abstract root class.

Geo-scientific applications are characterized by extremely complex non-regular shapes, which are usually decomposed into a set of adjacent spatial primitives. Correspondingly, *GeoToolKit* offers classes *Polyline*, *Triangle-Net* 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. As mentioned above, the initial choice of the geometric functionality was inspired by the requirements of interactive geological modeling, which, fortunately, turned out to be general enough for a lot of other geo-scientific applications. Currently *GeoToolKit* provides the following operations: intersection of spatial objects, partitioning of spaces and spatial objects by a plane, clipping a part of a space by means of bounding box, 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 specified either exactly 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.

Visualization

An adequate visualization is extremely important for geo-scientific applications. Diverse external stand-alone graphical viewers can be utilized for the visualization of geo-scientific data. As input they use text files in a special format (e.g. VRML). Therefore a really interactive communication with an external program (including database management systems) is hardly possible. Apart from this, a typical viewer supports only a very limited set of operations which can be applied to the visualized objects. Operations are either very general (in the case of general-purpose viewers) or very special (in the case of specialized ones). In both cases a viewer cannot provide the functionality the user needs for his particular tasks.

To enable the integration of arbitrary operations and really interactive communication with spatial databases we included in *GeoToolKit* a special support for the development of 2D/3D viewer tightly coupled with the database component. *GeoToolKit* provides 2D-/3D-*Viewer* classes which realize a basic functionality required for the successful navigation in space (e.g. moving/rotating of the viewpoint for 3D viewer), which enables a more convenient and effective selection of spatial objects from the database. The user can always customize his own viewer as a specialization of one of the embedded viewer classes by extending it with special-purpose operations. Though the functionality of 2D- and 3D-viewers significantly differs, there are some basic design principles which are common for both viewers.

All visual objects including user-defined classes are defined as specializations of the root *VisualObject* class. A visual object contains a reference to the corresponding database object and, if necessary, a representation of this object in the internal format of a concrete visualization tool. Usually it contains some auxiliary data members needed for the advanced management (e.g. a layer specification in the 2D-viewer). The *VisualObject* class hierarchy completely reproduces the persistent SpatialObject class hierarchy, thus providing a distinct separation of the database and visualization components.

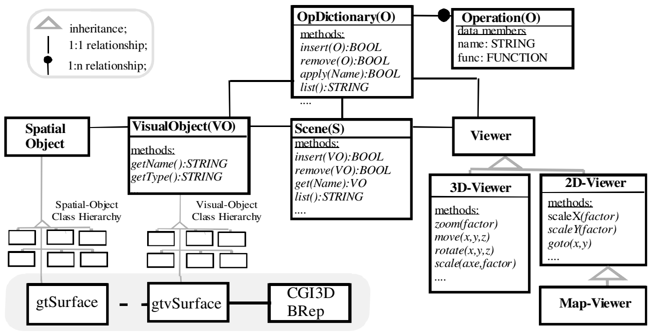

Fig. 2. Visual Object class hierarchy.

The interface specified in the root class serves primarily for the organization of a flexible interactive communication with the selected object. Among others it specifies a function that returns a textual representation of the object with different level of details. This representation is utilized, for example, in the data browser. In addition to the visualization of database objects *GeoToolKit* offers service objects such as 2D/3D bounding boxes or line segments which can be interactively created and manipulated by the user. These objects serve usually for the graphical specification of spatial queries.

Each visual object class can maintain its own dictionary of operations which can be applied to its instances. The default list of operations is pre-defined in the root class. It includes general operations as, for example, *hide* an object or *load* it in the data browser. All visual objects automatically inherit the default list of operations. The developer can easily customize the operation list using methods provided with the basic visual object class.

Visual objects are maintained in a special collection class called *Scene*. We can bind to a scene a list of operations which will be applied to all its elements. A default list includes such operations as retrieval of an object according to its coordinate, name, type, priority, etc. Due to the set-oriented nature of the functions associated to a scene it is a usual way to integrate interactive queries within a graphical viewer. Query formulation and presentation of results in spatial databases is much more complicated than in „textual“ databases. Though traditional methods of query formulation are also reasonable in spatial databases a graphical form is often much more transparent and compact, i.e. convenient for the end-user than alphanumeric expressions and masks. The same is true for the presentation of the results with the only difference that it is rather required than desired. At the same time spatial data contain a lot of thematic information which is alphanumeric by its nature. Masks and query expressions are still the most convenient form for querying such types of values. Since spatial and alphanumeric data coexist in a single object very often a query contains both spatial and thematic predicates. Such mixed queries should combine both graphical and non-graphical querying methods which complement one another.

By default a viewer provides a general purpose interactive query manager which is activated through the universal function binding mechanism described above. A user can always substitute the default query manager with his own one, which can realize more convenient application-specific masks instead of error-prone string-based queries. Query results can be either transferred from the temporal buffer into the scene or used for the iterative step-by-step refinement of the retrieval conditions, when every subsequent query is applied to the results of previous one. The query manager activation callback can be attached to a visual service object (e.g. bounding box), which is used for the graphical specification of selection criteria. In this case the query manager is automatically activated with the default selection predicate inherited from the bounding box, thus realizing a mixed graphical-textual data retrieval strategy.

Binding of a logical scene to the concrete window implementation is performed in the *Viewer* class. A viewer window consists of two areas: a drawing area and a control panel, which serves for the button- and menu based interaction with the viewer and the presentation of textual information Fig. 3 presents a variety of viewers implemented in *GeoStore* on the basis of *GeoToolKit*'s viewer classes. There are a 2D-map viewer, a 3D-viewer, a specialized 2D-viewer for geological sections. The user can navigate in the drawing area with the mouse and can specify service objects (e.g. a clipping box). A control panel consists of the menu bar, an optional control buttons bar and a message area. The menu bar contains several pull-down menus which provide methods for the activation of pre-defined and customized functions. Customized functions associated with the scene are selected in a scrolled list (operation selection menu). The mostly often used functions (e.g. load a database object per name) can be included in menus explicitly.

The visualization components are integrated into the OSF/Motif® environment. For the implementation of drawing area we used *OpenGL*® and more high-level object-oriented graphical libraries *CGI*® and *Cgi3D* [4] developed at the University of Bonn.

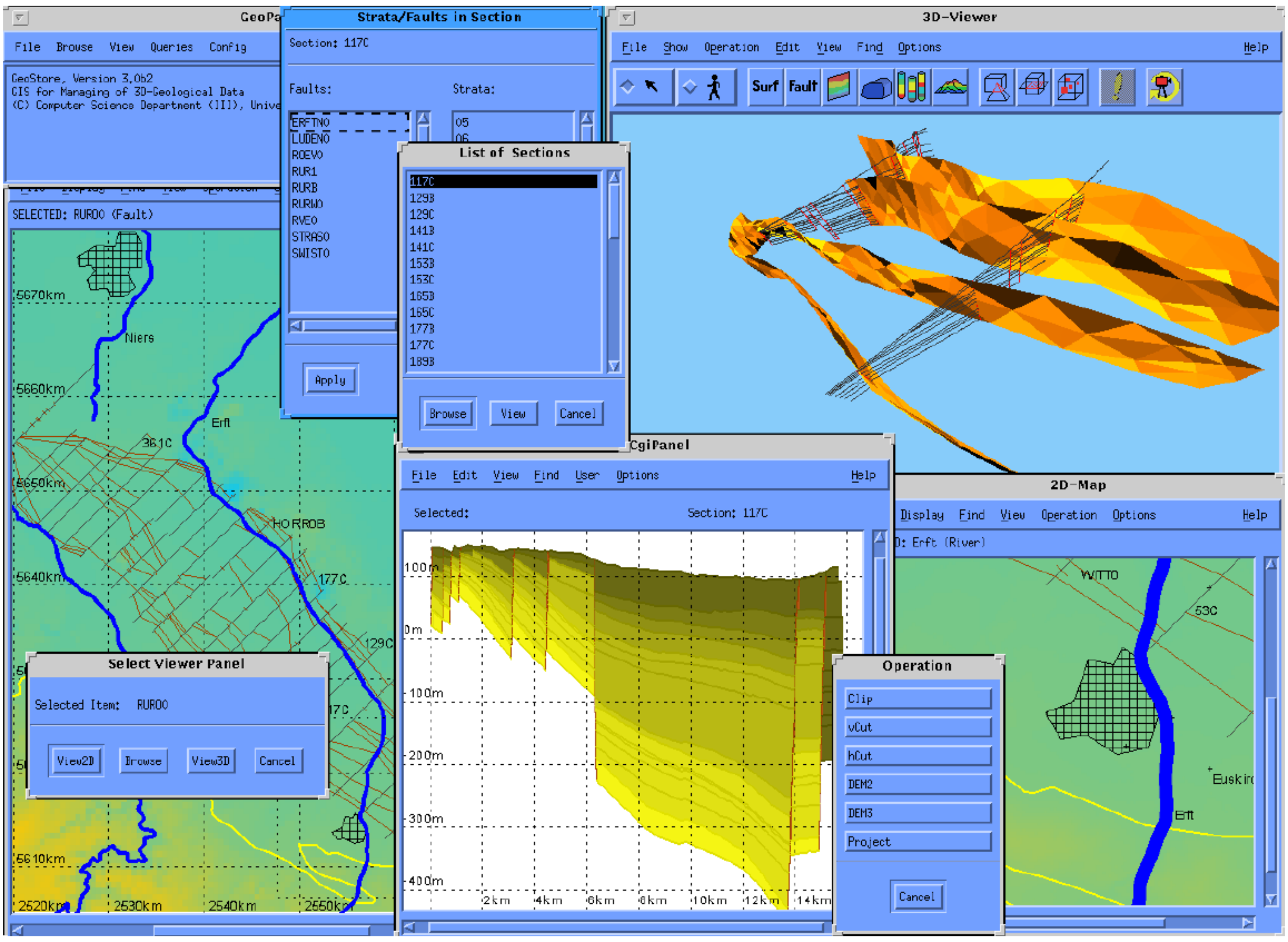

Fig. 4 *GeoStore's Windows: 2D-map viewer; 2D-viewer for geological cross-section(CgiPanel); 3D-viewer; operation selection menu.*

Building Applications on top of *GeoToolKit*

We present below our experience gained during the development of three different applications each illustrating diverse aspects of *GeoToolKit's* usage. The first and the most advanced one, *GeoStore*, was initially developed without *GeoToolKit*. Moreover, it was just our *GeoStore* experience, that inspired the architecture and functionality of *GeoToolKit*. That's why before launching the development of new applications, we decided to re-design *GeoStore* on the *GeoToolKit* basis in order to evaluate explicitly the benefits this approach would bring.

The second application deals with the maintenance of spatio-temporal data. Since such objects were not supported in *GeoToolKit* this application illustrates how *GeoToolKit* can be extended in order to meet new challenges.

The last application illustrates how to integrate in *GeoStore* data which are stored in *GeoToolKit* non-compliant databases.

Reconstructing *GeoStore*

Modeling geological structures and history always starts with a careful geometric analysis of the present assembly of geological surfaces, bodies and property distributions, being the key to reveal the nature and interaction of earlier processes. A process of computer-aided development of a consistent geometric model is known in geology *as interactive 3D/4D modeling* [5]. The model is further tested by backward restoration and balancing. The final model is used for the specification of boundary conditions for 3D transport models, or for the production and consistent updates of geological maps.

The starting point for the interactive geological 3D/4D-modeling is the digitalization of geological *sections* gained from open-cast workings. A section is a geological abstraction obtained as a result of an intersection of a vertical plane with geological strata and faults. Geological strata are sheet-like structures in earth with different mineral composition, texture and/or grain size. A fracture in earth materials along which the opposite sides have been displaced is known as a fault. A stratum is modeled as a list of layered bodies which are usually specified by their bounding surfaces. A fault is modeled as a simple surface. Within a section strata and faults are represented as point sets grouped in startigraphic and fault lines. Each point in a section contains 3D-coordinates is complemented with geologically-specific data such as stratigraphy. The second step in the interactive 3D-modeling is the generation of *triangulated surfaces* spread between sections. The final step, which is a part of our current work, is the transition from stratigraphical bounding surfaces to *volumes*.

The intention of *GeoStore* was to supply geo-scientists with a tool which would provide a consistent storage and efficient access for the data involved in all stages of the interactive 3D-modeling using a modern database technology. We began development of *GeoStore's* development with the elaboration of an object data model. There are three basic classes: *Section*, *Stratum* and *Fault* with extensions used as entry point to the database. A stratum in a section (*StratInSection*) contains an ordered list of stratigraphical lines. A stratigraphical line (*StratLine*) is a specialization of a geometric line, extended for the efficient computations with the additional topological information, namely a list of stratigraphical lines above the given one (*Hanging*) and the bounding faults (*Left Fault* and *Right Fault*).

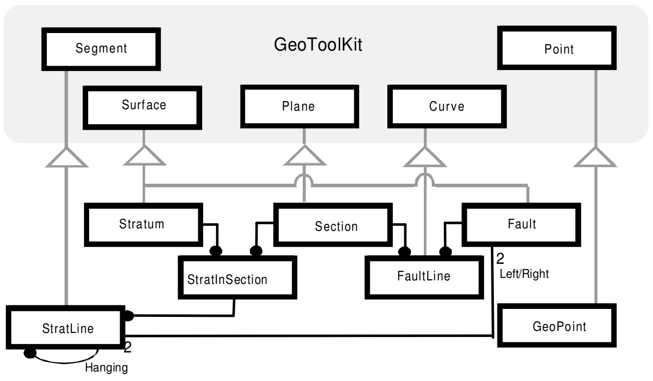

Fig. 4. *GeoStore's* class hierarchy.

Currently *GeoStore* supports two basic types of spatial queries:

- *Queries on strata and faults within a section*. The user, for example, may select all strata with a certain hanging stratum in a given depth interval located between specified faults. This type of queries uses primarily the topological relationships explicitly stored in the database.
- *Queries on triangulated stratigraphic and fault surfaces*. This type of queries deals primarily with geometric 3D-operations like horizontal or vertical slices through stratigraphic and fault surfaces, an intersection between surfaces or clipping of a part of a surface within a bounding box.

While redesigning *GeoStore* on top of *GeoToolKit* the main design principle we followed was to inherit from the geometric kernel as much functionality as possible (Fig.4). Following this approach, a user-defined class automatically inherits the whole geometric functionality of its parent. For every geological entity we found its geometric "pattern", which served as its ancestor in the class hierarchy. For example, a geological fault is represented now as a specialization of the geometric entity *Solid*, which was extended with geology-specific features (e.g. stratigraphy) and relationships (e.g. the *Hanging* stratum). By using the inheritance we achieved that every geological entity could be treated as a geometric object. *GeoToolKit*'s geometric kernel was able to operate directly on the collections of geological objects without intermediate transformations. In the case of the incorporation we would have to create a temporary space and to copy there the geometric ingredients of geological objects - the task which may be too expensive in the database context.

A solid may have two alternative representations: a bounding surface approximated in turn by a triangle network and a tetrahedron network. The first one is widely used in computer graphics. However, we chose the alternative representation of solids because it has a serious advantage for geo-scientific applications: any internal point not only automatically inherits values characteristic to the whole solid but also preserves them during series of diverse geometric manipulations and transformations.

However, to find a geometric pattern for such a sophisticated object as a geological section was not easy. There were conflicts between its representation as a rectangle or as a combination of curves. Therefore we implemented the *Section* class as a composite object containing both components. Since *Section* inherits directly from the abstract *SpatialObject* class, it provides both the representation and the implementation for all functions specified in the *SpatialObject* class. Practically the re-implementation resulted in delegating the functionality to elementary geometric components.

GeoStore is characterized by its tight coupling between database and visualization. Due to this the user is able to switch at any time between the object browser and one of the internal 2D/3D-graphical viewers, which are implemented as specialization of the *GeoToolKit* embedded viewers. In addition to a default interpretation the user can assign to a service object his own interpretation. For example, on the 2D-map a line segment can be used to specify a bounded vertical plane, that proved to be a very convenient method to specify vertical slices of underlying geological substances known in geology as cross-sections.

Topological relationships (e.g. neighborhood) used to provide very useful "guidelines" not available earlier in the "pure" geometric world. In order to benefit from this additional information we simply re-implemented virtual functions. An overridden function either substitutes the geometric function completely or performs a kind of pre-selection for the geometric function calling it with the restricted subset of spatial objects.

A qualitative analysis of the *GeoToolKit*-based reconstruction with the development of the initial *Oracle-based* version followed by the native *ObjectStore* version demonstrated the superiority of the methodology proposed. First of all, taking into account the complex structure of data with multiple interrelationships between objects the object-oriented data-model turned out to be more suitable for the development of *GeoStore* because it avoided the "impedance mismatch" between data as stored in the database and the representation required by the geometric algorithms for the efficient computing in main memory. In comparison with the native *ObjectStore* implementation *GeoStore* classes contain now only geology-specific data members and methods. A considerable part of geometric relationships between geological objects was directly inherited from the *GeoToolKit* classes.

Spatio-Temporal Data Browser

The ability to follow a topological evolution of geological entities is of special interest for geoscientists. Geological entities are characterized by relative large and irregular time intervals between time states available. On the contrary, for the smooth animation small regular time intervals are required. Therefore missed time states need to be interpolated. An interpolation between primitive simplex objects (simplexes) is straightforward. An interpolation of complexes can be reduced to the simplex-to-simplex interpolation only if complexes have the same cardinality. However, an object may change with time its size and/or shape in such a way that it will need more simplexes for the adequate representation than before. For example, in the result of deformations a flat platform (for the representation of which two triangles are quite enough) may transform into a spherical surface which will need a much larger number of triangles for the qualitative representation.

GeoDeform [6] is a geo-scientific application for calculating geological deformations which have been developed at the Geological Institute of the Bonn University. For the visualization of spatial objects changing in time *GeoDeform* uses a model proposed in the graphical library GRAPE [7]. According to this model each time state contains two representations of the same object with different number of simplexes (a discretization factor). The first representation (post-discretization) corresponds to the approximation of the current state of the object with the discretization required by its current size and shape. The second one (pre-discretization) corresponds to the approximation of the current state of the object but with the discretization used in the previous state. Due to this extension an interpolation can be always performed between representations with the same discretization factor: the post-discretization of the previous state and pre-discretization of the current state of the object.

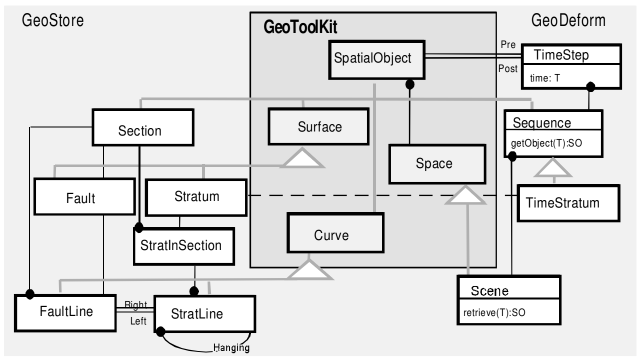

Fig. 5. Spatio-temporal extensions for *GeoToolKit.*

To provide an appropriate maintenance of a large number of spatio-temporal objects *GeoDeform* was extended with a database component developed on the *GeoToolKit* basis. The task was to integrate into *GeoToolKit*'s pure spatial classes a concept of time so that a spatial functionality already available could be re-utilized and a maximal level of compatibility with GRAPE was provided. To represent different time states of the same spatial object we introduced a class *TimeStep* (Fig. 5), which contains a time tag and two references (pre and post) to spatial objects. If the pre- and post-discretization factors are equal, pre and post links simply refer to the same spatial object. The model proposed is general enough since an arbitrary spatial object can be chosen as a representation of a time state. In the case of *GeoDeform* there are geological strata and faults modeled through G*eoStore*'s *Stratum* and *Fault* classes which in turn are defined as a specialization of *GeoToolKit*'s *Surface* class.

A sequence of *TimeStep* instances characterizing different states of the same spatial object are gathered into a spatio-temporal object (class *TimeSequence*). Being a specialization of the abstract class *SpatialObject,* a sequence can be treated in the same way as any other spatial object, i.e. it can be inserted into a space as well as participate in all geometric operations. The spatial functionality is delegated by default to the spatial object referred to in the latest *TimeStep* instance.

A selection for the interpolation differs from a common selection with a specified key. If there is no object in the time sequence that hits exactly the time stamp t specified in the retrieve function, instead of NULL it should return a pair of neighbor time steps with time tags t1 and t2, so that t1 < t < t2. The same is valid for the time interval. If interval's margins do not exactly hit the time step instances in the sequence, the resulting set includes all time steps fitting the interval completely extended with the nearest ancestor of the time step with the lowest time tag and the nearest successor of the time step with the highest time tag. Any *TimeSequence* instance can be inserted in and spatially retrieved from the *GeoToolKit*'s space as any other spatial object. However, to perform a temporal retrieval we introduced a special container class (*Scene*) which is capable of both spatial and temporal retrieval.

Though the data model presented was developed for a particular application it turned out to satisfy in main the requirements of geo-scientists and seems to be general enough to be used in various geo-applications. After the test stage we are going to include it in *GeoToolKit* as a standard component.

Well Data Management

The main objective of the geo-information system *WellStore* was to provide an integrated storage of all kinds of data related to drilling wells. In the past raw-data tables and diagrams were maintained in the form of flat ASCII files or even as paper-sources. Multiple data-formats with non-observable number of mutations made analysis and computations on well-data a complex and error-prone task. Since data were often dispersed between different sources it was hardly possible to inquire the whole data set and to get a comprehensive view on the problem. As a further drawback existing software systems perform just one or two steps in data processing taking no care of how the results can be used by other applications.

The necessity to provide an efficient maintenance of large amounts of well data and support a reliable shared access to data from multiple applications from various geo-scientific areas made us turn to standard database management systems. A typical well data element consist of a mandatory header incorporating diverse accounting data and the well location. Optionally various measurements may be coupled with the well: electrical logs, samples, stratigraphical interpretations. In general well data are characterized by the lack of direct interrelations between separate well entities. However, data referred to within a single well may have very sophisticated hierarchical structures which need non-standard data types for their representation (Fig.6). In addition to the traditional selection criteria (e.g. well id) wells are often retrieved from the database according to diverse spatial criteria involving neighborhood or occurrence in a certain region. Taking into account non-standard data types which were hardly to represent within relational DBMSs we decided to design *WellStore* on the object-oriented basis on top the *ObjectStore* database management system.

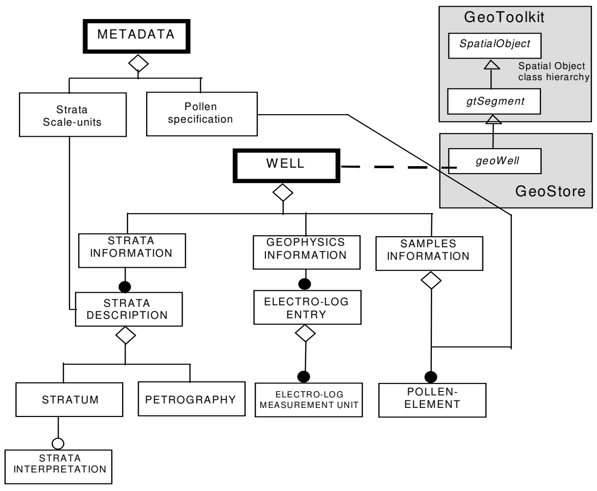

Fig. 6. The *WellStore* object model

WellStore provides an interactive browsing of well data stored in the database in order to select a well for further processing by one of application-specific routines preliminary linked to *WellStore*. If necessary, the results of the processing are stored back in the database (e.g., in form of various stratigraphical interpretations). Currently *WellStore* provides the following selection criteria (and their arbitrary combinations): id, location (region), depth, strata, existence of stratatigraphical, geophysical or samples data. The user can additionally specify whether he needs stratigraphical interpretations or just the original stratigraphical data.

One of our goals was to provide a system with extensible functionality. New implementations of geo-scientific algorithm can easily be linked to the system. As an example of such module integration we implemented a palynologic algorithm, developed at Bonn University, which provides a statistical analysis of pollen data to allow a review of the climatic situation in former ages. Results of the analysis are indications of precipitation, average temperature and humidity. Another example of such „algorithm plug-in“ approach is implemented in *WellStore*'s geophysics component. It permits to perform „cross-plots“ (Fig. 7) of the e-log data. This diagram illustrates relationships between a well's geophysical units of measurement stored in its e-log information. The integration of an algorithm for the frequency of existence of a concrete species of pollen data inside the strata is currently in progress.

An interpretation editor embedded in *WellStore* permits to define, to edit and to store user's own strata interpretations. The editor displays the lithological profile and the original stratigraphical data of the current well. The lithological profile can be manipulated by changing the boundaries and descriptions of a single stratum. The integrity-check control mechanism guarantees semantic correctness of the changes. A well's electro-log diagram can serve as an additional guideline during the interpretation. Starting from this diagram the geo-scientist can define and store individual strata-boundaries and strata-descriptions based on the original stratigraphical data.

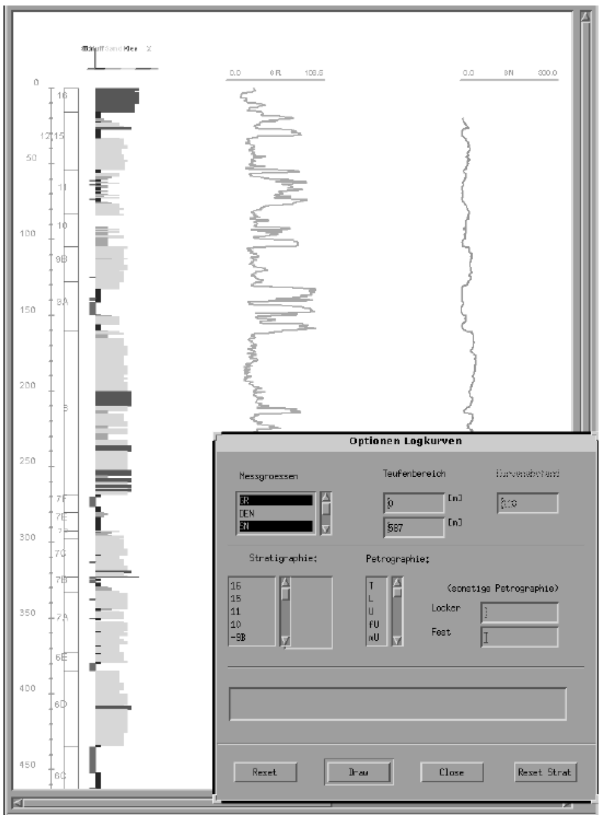

Fig. 7 ***WellStore's*** E-log diagram

Such facilities as 3D-spatial retrieval, spatial manipulations, 3D-visualization, which are extremely important for the interactive 3D-modeling generally do not play a significant role in a typical well management system. To avoid redundancy they were not taken in the account while designing basic data structures. However, wells may additionally provide a lot of useful information which can be used to check models generated in the process of interactive 3D-modeling. From the other side, a lot of well processing applications would benefit considerably when the user could additionally get an integrated 3D-model of the geological region surrounding the well. The idea to make well data available for *GeoStore* and vice versa, seemed to be very promising.

As mentioned above, *GeoStore* benefits considerably from the fact that its data types are built on top of the *GeoToolKit* class library. All application-specific data such as stratigraphic surfaces and faults are retrieved, processed and visualized primarily by *GeoToolKit*'s runtime environment (geometric kernel). The data types stored in a well database are not compliant with *GeoToolKit*'s classes. To make well database entities „known" to the GeoToolKit kernel we followed a wrapper approach.

First of all we found in the *GeoToolKit* class library a geometric pattern for a well. In our case a simple 3D-line segment turned out to be the most relevant geometric representation (in general, a more complex geometric representation may be required, e.g. a 3D-curve). Consequently, we specify a new class *geoWell* as a specialization *GeoToolKit*'s class *gtSegment*. To avoid an unnecessary redundancy (which may cause serious consistency problems by updates) the *geoWell* class contains only a reference to the corresponding well instance in the well database. Every well contains all data needed for the correct reconstruction of its geometric shape. The problem is that these data are often stored in the way not suitable for the direct usage by the *gtSegment* methods. Sometimes they are even dispersed in different data segments. To capture this we had to redefine only the data access methods. For the efficiency purposes the mostly often accessed data members can be copied in the *geoWell* object.

We can insert instances of the newly defined *geoWell* class in *GeoStore*'s space. Being included in space the wells can be automatically retrieved, manipulated and visualized by GeoToolKit's geometric kernel as any other GeoStore native class. Thus we automatically benefit from the functionality already implemented in *GeoToolKit.* For example, we automatically can get a projection of wells on the vertical plane thus contributing in the construction of consistent geological cross-sections.

When a well database is attached to *GeoStore*, a temporal snapshot of the current state of the well database is captured. At the end of the *GeoStore* session this temporal sub-space and its elements are removed from the *GeoStore* working space.

Conclusions

The partial re-design of *GeoStore* as well as the implementation of several new applications on top of *GeoToolKit* proved the advantages of the *GeoToolKit*-based development. The *GeoStore* classes contain now only geology-specific data members and methods. A significant part of geometric relationships between geological objects can be inherited from *GeoToolKit* classes. This results in considerable reduction of code, that not only accelerates the development process but also increases the reliability of applications since the re-use of already approved methods preserves the developer from inevitable errors during the new development. The developers could focus on the application semantics instead of such "creative" tasks as optimal assembling of spatial objects from multiple tables or the implementation of routine geometric algorithms. Since application specific classes no longer contain geometry-related components, the classes become significantly shorter and, consequently, more understandable for external users. From the other side, our experience demonstrates, that the *GeoToolKit* core can be easily extended to satisfy special requirements. The *GeoToolKit* functionality (spatial retrieval, indexing, etc.) is available in a full volume for extended objects as well.

Developing all applications on the common *GeoToolKit* basis we make a significant contribution in the non-redundant and consistent maintenance of shared data. Data designed and stored within a particular application become available for other *GeoToolKit*-compliant applications as well. Moreover, already existing geodata stores can be connected to *GeoToolKit*-compliant data „circuits".

Apart from this, the object-modeling technique turned out to be an appropriate environment for the communication between geo- and computer scientists. Geologists not-experienced in software design quickly adopted this technique and successfully applied it, utilizing the pre-defined set of *GeoToolKit* entities as building blocks. The data model designed in such way is then practically one-to-one mapped on the *GeoToolKit* class library.

Acknowledgments

This work was done in the close cooperation with the Geological Institute at the University of Bonn. We thank Prof. A. Siehl, R.Alms, T.Jentzsch, M.Klett, T.Utescher who provided the geological backgrounds for the work. Special thanks to Norbert Klein, Dirk Moebius, Wolfgang Mueller and Michael Werner who made a significant contribution in the implementation of software.

References

[1] O. Balovnev, M. Breunig, A.B. Cremers. From GeoStore to GeoToolKit: The second step. In: Proceedings of the 5th International Symposium on Spatial Databases, LNCS, Vol. 1262, Springer, 1997, pp. 223-237.

[2] Th. Bode, M. Breunig, A.B. Cremers. First Experiences with GeoStore, an Information System for Geologically Defined Geometries. In Proceedings IGIS'94, LNCS, Vol. 884, Springer, 1994, pp. 35-44.

[3] J. Rumbaugh et al. Object-Oriented Modeling and Design. Prentice Hall, New Jersey, 500p.

[4] D. Fellner. Extensible Image Synthesis. In. P. Wisskirchen (Ed). Object-Oriented and Mixed Programming Paradigms, Springer, 1996, 7-21.

[5] R. Alms, K. Klesper, A. Siehl. Three-Dimensional Modeling of Geological Features with Examples from the Cenozoic Lower Rhine Basin. In: Foester, A, Merriam, D (Eds.) Geological Modelling and Mapping, 113ß133, Plenum Press, New York, 1996

[6] R. 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, XXII General Assembly of Geophysical Society, Vienna, Austria, 21-25 April 1997, In: Physics and Chemistry of the Earth (in print).

[7] K. Polthier, M. Rumpf, A concept for Time-Dependent Processes. In: Goebel et al. (Eds) Visualization in Scientific Computing. 137-153, Springer, Vienna.