Usage of persistence framework technologies for 3D geodata servers

Jörg Haist, Raphael Schnuck, Thorsten Reitz
Fraunhofer Institute for Computer Graphics
Fraunhoferstr. 5, 64283 Darmstadt, Germany
joerg.haist, raphael.schnuck, thorsten.reitz@igd.fraunhofer.de

SUMMARY

In this paper the authors present the works and research of the persistence framework integration into 3D geodata servers for city and landscape models. These frameworks are not used yet by geodata servers since the complex data types and proprietary SQL extensions used to query spatial data makes this relatively difficult. Thus, the authors had to find solutions for an usage with typical spatial objects. The Cityserver3D system, which uses an oracle spatial database as default data source, was extended by the Hibernate persistence framework. Hibernate offers functions for data access and management. For the integration, several extensions had also to be added to the persistence framework. These are mainly dealing with the object-relational characteristics and the query mechanisms of a spatial database. So, the developer can focus on the business logic and does not have to care about the data access. During the adoption of hibernate several topics arose which will explicitly described in the paper. Also, an outlook is given at the end of the paper.

KEYWORDS: *Geodata server, 3D GIS, Persistence framework, hibernate, 3D city models*

INTRODUCTION

In this paper, the authors present the works and research of integrating a common persistence framework into 3D geodata servers for city and landscape models. Since these frameworks are not used yet by geodata servers the authors had to find solutions for an usage with typical spatial objects. The design and implementation of these solutions are the focus of this paper.

Originally, the CityServer3D used direct JDBC access, using the drivers provided by the database manufacturers and wrapping them with a connection manager class. However, within the ongoing development of the CityServer3D, sophisticated database connection mechanisms had to be considered since the existing solution was not appropriate to the growing requirements. Also, the existing solution was too time-consuming regarding the maintenance and extensions.

As illustrated in Figure 1, the CityServer3D is designed as a 3-tier architecture (Reitz2005) in which the server components are using a database access layer to manage persistent data. GeoBase21, which is the successor of GeoBase20 (Schilling2005), is the main database scheme for the CityServer3D - it was modelled together with the metamodel (the internal server runtime model) and has a high coverage with the metamodel. Apart from geometries the database also stores the thematic and spatial classification of the data sets. Furthermore, attribute data sets and metadata can be held. The database scheme is built up around the table `Feature3d` which represents an object with a spatial and a semantic characteristic. `Feature3d` is modelled as a self-referencing table which allows modelling of complex buildings consisting of building parts, windows, doors, and other structural elements. Here, the database gives wide possibilities to manage data in several detail levels and also in different semantic aspects so that besides geometric characteristics, semantic ones can also be stored in the database. `Feature3d` follows the concepts behind the simple feature specifications of the OGC.

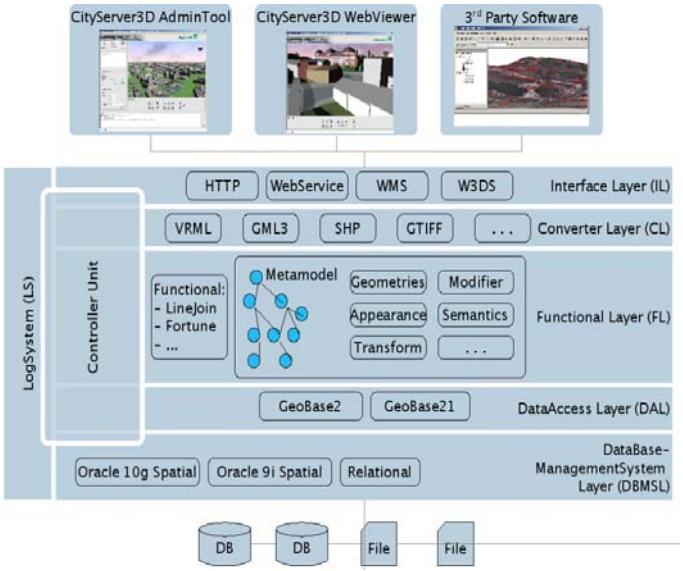

Figure 1: Layers and modules in the architecture of CityServer3D

The server component plays a central role within the CityServer3D, allowing access to data stored in various databases via different interfaces. The architecture permits not only to query the data stored in its own databases but also additional data sources. These can be web services or files loaded using import components. In order to realize this, the CityServer3D uses a metamodel to hold objects for modification, transformation and translation. Complementing these capabilities, the server has components for authentication and recording of transactions, so that it can be used as a technology for *e*Commerce.

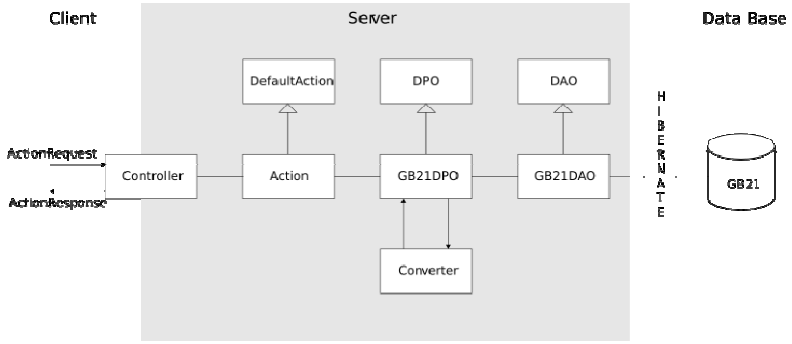

Figure 2: Sequence of an inquiry to a data base

Via the server's interfaces, clients can connect to it and use offered services. OGC-based web services like W3DS (Haist2005) or WMS can be used by any client which supports these interfaces whereas proprietary interfaces are used by the AdministrationTool or the WebViewer of the CityServer3D system. The actual working process of an inquiry to the Geobase2.1 data source can be seen in figure 2. Depending on the capabilities of the client, it can either send a standard `HttpRequest` (see the HttpServlet API) or it can directly use an interface-neutral `ActionRequest` object. This Request is parsed into an `ActionRequest` by a Facade, usually a Servlet, and then dispatches to the controller unit. The controller authenticates the request based on the credentials that came with it and

subsequently passes the `ActionRequest` on to the suitable Action.

An Action represents the smallest unit of work that a user can trigger from the outside and is in this case a specialization of the class `DefaultAction`, which implements fundamental functionalities for all actions. Within the Action, access to data is managed by retrieving a so-called DPO[6]. This is a variation of the classical J2EE pattern of using Data Access Objects (DAO) to manage data access. As it is the case with DAOs, a DPO is an interface that contains a series of access methods for loading or manipulating a specific data type; thus, there are sub-interfaces for the wide variety of required data types. In this case, since data from the GB21 data source is to be retrieved, the associated GB21DPO implementation of the interface is called.

The responsibility of this DPO is to provide (hence the name) data in the form of metamodel classes. For this, it accesses two additional components: Converters and classical DAOs. The GB21DAOs implement an DAO interface, similar to the DPO interface, but their implementations return objects with classes native to each specific format. From the DAOs the access to the data base with the aid of Hibernate functionalities happens.

PERSISTENCE FRAMEWORKS

Nearly all applications require persistent storage with which they are able to save the state of a program or created objects and restore them later, even if the process in which the objects were created was shut down meanwhile. Regarding spatial data servers and their persistence management, relational databases (RDBMS) are considered the standard storage system. However, if the server software is based on the object-oriented programming paradigm, the process of storing the objects in the RDBMS requires several changes to the data, some of which are quite obvious, while others are rather subtle. These differences have been analysed by Ambler in his publication called “the Object-Relational Impedance Mismatch”(Ambler2003) and in (Elverkemper2005).

One of the issues described by Ambler is the “Cultural Impedance Mismatch”. Summarized, this issue is that developers usually come from either OOP or the RDB world and try to extend their paradigm onto the other, leading to either suboptimal DB schemata that are not properly normalized and don't harness the power that relational DBs could offer or to hard-to-work-with object models.

The same mismatch often also applies when using Object-Relational Databases (ORDBMS). While these have often been hailed as being the “final” solution to storing objects, in practice several issues came up. One of these is that ORDBMS break some of the rules of conventional databases, e.g. on maintaining key integrity and on normalization and thus sacrifice some of the advantages a RDBMS could offer. Another issue is that usually, not actual objects in the programming language of choice are produced, but rather data that is more like a STRUCT (in the meaning of C Structs which are “mini-objects” without any behaviour), described in the database vendor's proprietary format. The translation of these STRUCTS to – in our example – Java objects proved also to be effected by the OR impedance mismatch.

Persistence frameworks are built to isolate the DB layer and the business process layer, resolving this mismatch and its related problems for the greater part. This includes managing the persistent objects for concurrent access. Persistence frameworks become very popular in the area of web-based applications, but can of course also be used in other application areas. These applications are most commonly written in Java (J2EE) or C# (.NET), with Ruby also becoming popular.

From an application developer's point of view, the main advantage of persistence frameworks is the possibility they give to the software engineers to concentrate on the business logic instead of on handling persistence functionality themselves, which is often an error-prone and tedious task. The frameworks provide solutions from managing the database connections (e.g. connection pooling) to abstracting complex data queries. This complete isolation of the database layer from the application

6 DPO: Data Provider Object

layer means that the whole application will be more serviceable and single layers can be replaced easily. Ideally, the aspect of managing the storage, retrieval and concurrent access to data is completely removed from business code and is being done in a transparent way, meaning an application developer does not have to extend framework specific classes or implement interfaces.

HIBERNATE FOR 3D GEODATA SERVERS

With respect to geospatial applications, the biggest lack in persistence frameworks is that they do not support geospatial database characteristics like the spatial data types built into Oracle Spatial or PostGIS.

Thus, the current implementation of a persistence framework has to be extended, since geodatabases use non-standard mechanisms to manage spatial data. In the case of 3D city and landscape models, these extensions are e.g.:

- Specific data types to manage geometries like polygons, multilines and others
- Query language extensions to retrieve objects with specific spatial properties or relationships
- Functions to manipulate geometries, e.g. aggregation or transformation
- Management of spatial reference systems

So, for an usage of a persistence framework, the new data types and the extensions to the query language have to be added to the persistence framework. For this, solutions were found that do not mean changing the original code of the persistence framework.

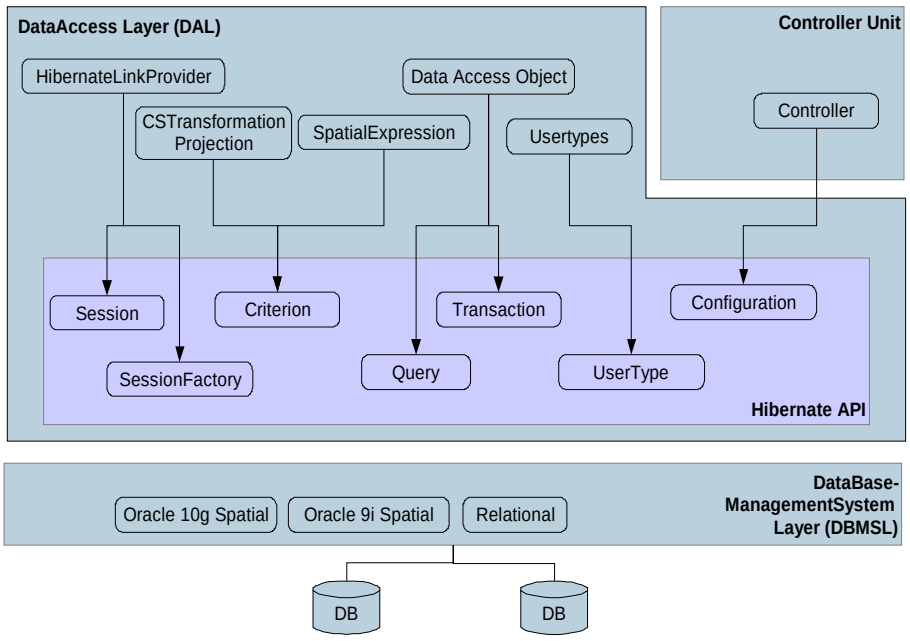

Figure 3: Usage of the Hibernate API in CityServer3D

An overview of the interface mechanisms used to connect the CityServer3D system, which uses an oracle spatial database as default data source, to the Hibernate persistence framework (Bauer&King2005) are shown in Figure 3.

One of the extension mechanisms Hibernate offers are so-called User Types. Using these, column types that are not natively supported by Hibernate like SDO_GEOMETRY can be transformed into Java objects and back. This first point is especially important since GIS extensions for commonly used databases like POSTGIS or Oracle Spatial make heavy use of so-called "User Defined Types" (UDT), which are essentially complete objects stored in a single column of a relation. One such UDT can for example consist of a collection of points, additional data like the type of the geometry stored (Polygon, Point Cloud...), a key of a coordinate reference system and information on how these points are connected (their topology).

Contrary to a JDBC-based solution and embedded SQL-code which would return ResultSets, the framework allows to retrieve "plain old java objects", that is, completely assembled and useable objects, from the data access layer. Two mechanisms can be used and combined. The first of these is the Criteria API, which allows queries by example objects or by constraints passed as objects. The second mechanism is in the case of Hibernate the so-called HQL, a query language with object-oriented elements. Both have in common that the translation to the runtime data objects is accomplished automatically.

Like mentioned before, one of the goals of using an persistence framework within our 3D-GIS was to make it easier for developers to transfer objects from or to the relational database. When using standard SQL queries over JDBC, very long and often complicated queries were rather common due to the complex proprietary extensions to SQL that come with Oracle Spatial and other GIS extensions. On the one hand, this takes a lot of embedded SQL-code within Java code and, on the other side, the mapping to runtime objects has to be manually done within the data access layer. To simplify this and to reduce redundant code, it was decided to use Hibernate's Criteria API. To give an impression how these object-oriented queries look like, consider the following example:

```
List pizzerias = sess.createCriteria(Building.class)
   .add(Restrictions.like("usage", "%Pizzeria%") )
   .add(new SpatialExpression
              ("mbb", queryPolygon, querySRS, queryMode)
        )
   .addOrder( Order.asc("name") )
   .setMaxResults(50)
   .list();
```

What happens here is that for a certain object type, in this case, `Building`, a series of constraints is added to the criteria query. Each of these constraints is given the name of the attribute it should operate on, as well as additional parameters required. In this example, all buildings whose usage attribute contains "Pizzeria" and which have the spatial relationship defined within queryMode to the queryPolygon will be returned. Also, the number of results is limited to 50 and the result is to be ordered by the attribute Name. This `SpatialExpression` used in the example is one possible type of encapsulation of the aforementioned complex spatial queries.

The second possibility of using the criteria API is the usage of Query-By-Example (QBE). In this query mode, an object of the type to be retrieved is constructed with all the parameters that the result objects should also have. This object is then used by Hibernate to automatically create conditional clauses as appropriate. For this process of translating Criteria Queries (as well as queries in the Hibernate Query Language (HQL)), Hibernate makes use of Dialect objects. Each of these Dialect objects describes the SQL variant specific to a single database. Consequently, Dialects like MySQLDialect or Oracle9iDialect exist. These can also be extended, in our case to cover the additional functions brought by Oracle Spatial. However, some functions cannot be used in this way, most specifically those functions that are applied within the selection part of an SQL statement. These are, quite important, as they enable the developer to use spatial reference system (SRS) transformations and other complex functionality offered by the database.

To bridge this gap, an new `SpatialCriteria` interface was derived from the existing Criteria

Interface and implemented by extending Hibernate's `CriteriaImpl`. This implementation can now be created directly (as opposed to Hibernate's standard way of using `session.createCriteria(someclass.class)`) by using a constructor that passes a reference to the session. It will automatically add functions like Oracle Spatial's `cs_transform` to those columns in the select clause that are of a geometry type.

This works well for simple queries, where essentially one type of object without a complex attached graph of additional objects needs to be retrieved. Hibernate has the capability of reconstructing entire object graphs from the database, and employs a technique called lazy initialization to do this. This means that actual queries for sub-objects are only executed when the Java application actually tries to access these sub-objects and brings with it performance and memory gains. When creating these queries, Hibernate does unfortunately not allow the use of projections beforehand to also affect these. A renouncement of lazy initialization would be possible by configuring Hibernate to not use it, but wasn't acceptable for the authors for these reasons.

Luckily, Hibernate also allows access to the resulting automatic generated SQL statements. They can be affected in two ways (King2005):

- Using custom SQL
- Using the filters API

Custom SQL can be used to replace the create, load, update and delete statements that Hibernate would create. These can be overwritten with the aid of the mapping tags `<sql-insert>`, `<sql-query>`, `<sql-update>` and `<sql-delete>` within the class mapping files:

```
<sql-insert>
   INSERT INTO Layer (NAME, ID) VALUES ( UPPER(?), ? )
</sql-insert>
```

Inside the custom SQL statements the integration of data base specific functions is possible, as in this case UPPER. The problem with this approach for the authors was that no possibility exists to merge dynamic parameters, like those needed for a conversion between different SRS. Since the interoperability of the CityServer3D has to be ensured, a firm integration of a certain spatial reference system was not applicable. Thus, the possibility of custom SQL is not practicable in the case of the CityServer3D.

The second possibility of manipulating of SQL are filters. These offers the possibility of defining global filters, which can be enabled or disabled for different sessions. The filters to define a restriction in the same way as if adding a constraint to the WHERE clause of a SQL statement.

```
<filter-def name="myFilter">
   <filter-param name="myFilterParam" type="string"/>
</filter-def>
```

The defined filter can be added to a class or a collection within the mapping file.

```
<class name="myClass" ...>
   <filter name="myFilter" condition=":myFilterParam =
                                      MY_FILTERED_COLUMN"/>
</class>
```

By default, filters are disabled within a session. They have to be enabled explicitly for a session.

```
session.enableFilter("myFilter").
   setParameter("myFilterParam","some-value");
```

Since the filters make available only an additional "where" attribute, employment of these manipulation possibility in the CityServer3D has not yet been used since we already handle spatial

queries using the criteria API. For others who are looking for a quick way to add spatial filtering to hibernate, this is a method worth investigating.

As can be seen, no perfect solution for the SRS transformations within complex object graphs exists yet without diving into the Hibernate source code and modifying it. Since this is however not wished, we currently retrieve objects that need to be transformed separately from the rest of the object graph. In this way, we can use Hibernate projects directly on the geometry columns. The following listing gives an example how a Grid object is retrieved and transformed.

```
Criteria c = this.getSession().createCriteria(Grid.class); c.setProjection(
Projections.projectionList()
   .add( new CSTransformationProjection("LL_POINT",
area.getSpatialReferenceSystem().getSrid()))                    .add(
new CSTransformationProjection("MBB",
area.getSpatialReferenceSystem().getSrid())));    c.add(se);
List result = c.list();
```

Together, the extensions described widely allow the usage of hibernate for geodata databases. When using a “pure SQL” database schema, some of these aren't even necessary, reducing the effort required. After this customization process, Hibernate's own abstraction of SQL, called HQL, as well as the Criteria API are used to simplify spatial queries of spatial objects while preserving access to the powerful functions some of the databases can offer.

EVALUATION

The simple use of the spatial conversion functionality provided by Oracle Spatial argues for the employment of a persistence framework, were that access complete. This would make it possible for GIS developers, without requiring additional implementations, to have conversions between two spatial reference systems handled by the persistence framework. Helping here would be a feature within Hibernate to influence the way the SELECT portion of the statement is created. It should be possible to base this on the type of the column/attribute to be persisted; if that last hurdle is taken, support can be regarded as being complete.

From experience it can be said however that the older hand written versions of the SQL statements were not faster than those created by Hibernate. When caching comes into play, Hibernate even starts to gain a hefty performance plus. By this performance testing, it was also found out that particularly within the Hibernate API only a fraction of the total time is used up (max. 5 %, with the rule being far below 1% for typical GIS processes). So Hibernate indeed makes available a quasi optimal solution from the performance view.

Our use of Hibernate indicates as an advantage that the productivity of the developers is increased, since they can concentrate to a large extent on the development of the business logic. For the productive use, as with each framework, a certain training time has to be spent on Hibernate.

CONCLUSIONS

Persistence frameworks like Hibernate offer the possibility to abstract the database away from an application, allowing the development of systems that are interoperable on the level of their data sources. The developments described within this paper were one step to also be able to do this abstraction step for the spatial extensions offered for MySQL, Oracle or PostgreSQL.

Besides the technological advantages like the clean detachment of the database access layer and the extended query possibilities some lessons learnt regarding the daily implementation work can be drawn. Due to the better separation of database and runtime objects, extensions in the database or on the Java code were better manageable than before the persistence framework integration. Also, the amount of side effects of changes in another layer that occurred during the tests of the extensions declined. The separation is also benefiting the work in a team of specialists: Each of the developers

can specialize in one aspect of the application, creating less conflicts and saving the time that would usually be required to assemble the knowledge for other areas of the application (see also Brooks1995).

Regarding the implementation some drawbacks occurred. The implementation of the User Types for the Oracle Spatial DB was more complicated than expected. The initial effort to integrate the persistence framework while preserving the functionality available before required about two person months, and to reach full SRS support another two person months were spent. By using Hibernate 2.1 the possibilities to integrate specific functions like `sdo.cs_transform` were limited and only feasible by workarounds. However, these have for the better part been addressed with Hibernate 3, so that there are no major barriers left for the use of a persistence framework in GIS applications. Most of the time spent was a one-time investment, though, and a large portion of it has already been recovered by better code reuse and the fact that most developers didn't need to learn the specifics of spatial databases. What can so far be confirmed is Hibernate's claim of high-speed persistence (King2005) – performance bottlenecks were usually located on the database level or on the application level, when complex processing had to be done.

Summarized, the obstacles that rose up when integrating a common OR mapping/persistence framework into a geodata server can be overcome, to reap benefits on various levels from performance though the use of caches, managing concurrency and developer usability through concern separation.

REFERENCES

Ambler, Scott, 2003, The Object-Relational Impedance Mismatch. In: Agile Database Techniques, Wiley Publishing

Bauer, Christian and King, Gavin, 2005, Hibernate in Action. Manning Publications

Brooks, Frederick, 1995, The Mythical Man-Month, Anniversary Edition, Addison-Wesley, pp. 31ff.

Elverkemper, Timo , 2005, A project management web application based on javaserver faces and hibernate. Diploma Thesis.

Haist, Jörg; Coors, Volker, 2005, The W3DS-Interface of Cityserver3D In: Kolbe, Gröger (Ed.); European Spatial Data Research (EuroSDR) u.a.: Next Generation 3D City Models. Workshop Papers: Participant's Edition. 2005, pp. 63-67

King, Gavin, 2005, Performance Q & A, online resource, http://www.hibernate.org/15.html, day of last access: 14.12.2005

King, Gavin, 2006, HIBERNATE - Relational Persistence for Idiomatic Java, online resource, http://www.hibernate.org/hib_docs/v3/reference/en/html/, day of last access: 08.03.2005

Reitz, Thorsten; Haist, Jörg (Supervisor), 2005, Architecture of an interoperable 3d GIS in special considerance of visualization applications. Furtwangen, Fachhochsch., Master Thesis

Schilling, Arne and Jasnoch, Uwe, 2005, Datenbank-basierte Visualisierung von Hamburg, in: Coors, Volker and Zipf, Alexander, 3D-Geoinformationssysteme. Heidelberg: Wichmann, pp. 251-264