NHibernateMapper - A Tool for Rapid Development of Data Access Layer for Interoperable GIS Solutions

Stevica S. Cvetković, Miloš D. Bogdanović and Leonid V. Stoimenov

***Abstract*—In this paper we presented architecture and implementation details of an object–to–relational-database mapping specification tool (NHibernateMapper) based on NHibernate ORM framework. The tool is able to automatically traverse provided database schema and domain class library to be matched and to suggest correspondences. Interactive GUI provides ability to accept or reject suggested correspondences as well as to manually define mappings in order to prune it into a final form. By analyzing database schema of spatially enabled DBMSs, this tool supports the development of Data Access Layer for GIS solutions. If provided with a domain class library generated on the basis of application ontology used by GIS solution, NHibernateMapper becomes a tool that enables rapid development of Data Access Layer for Interoperable GIS solutions.**

***Index Terms*—Schema mapping, Object relational mapping.**

I. Introduction

Most of the modern information systems use databases for storage of persistent information, mostly because of their reliability and standardized query language. On the other hand, object-oriented programming languages such as Java, C# and C++ are most commonly applied for the system programming. Conventional software development approaches for database access within object-oriented languages raises a problem known as “object-relational impedance mismatch”. It occurs because the object-oriented and relational database paradigms are based on different principles [3] and manifests itself in several specific differences: inheritance implementation, association’s implementation, data types, etc.

Object Relational Mapping (ORM) tools have been developed in an attempt to overcome mentioned problem. As described in the Agile methodology for software and databases development [3], ORM tools automatically create Data Access Layer based on database model. The most successful representatives of ORM tools are Hibernate [11], Oracle TopLink [12] and recently introduced Microsoft Entity Framework (EF) [1,2]. With these tools, application developer is encouraged to think in terms of data layer objects and their relationships. The ORM takes all the control of handling objects and relationships at runtime. It automatically tracks updates made to the objects and performs the necessary SQL insert, update, and delete statements at commit time.

Hibernate is widely used, comprehensive and stable open source ORM framework. Furthermore, it is available for both Java and .NET environments (NHibernate). Although there are Hibernate tools for automatic generation of object model and mapping files, based on existing database, there are only few commercial tools which are capable to map already created object model to existing database. This is of crucial importance when trying to map standardized domain knowledge (application programming classes) to legacy database systems. For this purpose NHibernateMapper tool is developed, which is able to generate NHibernate mapping file (*.hbm* file) based on the already available domain knowledge and existing database. These three components (application programming classes, database and mapping file) are sufficient premises for generation of complete Data Access Layer using NHibernate ORM mechanism [8].

The aim of this paper is to present a tool that can be used for rapid development of data access layer for interoperable GIS solutions. Its primary purpose is to generate objects–to–database mapping specifications according to NHibernate ORM framework. It is used to assist in semi-automatically determining correspondences between GIS solution domain class model and the underlying database. In particular, correspondences are determined between domain class member attributes and database columns. The tool contains the visual front-end which suggests mapping using “Levenshtein distance” and “Data type correspondence” heuristics. In order to define final mapping definitions, these suggestions must be pruned by user interaction. Defined mapping definitions are then used to generate XML-based mapping file.

The paper is organized as follows: Section 2 describes the purpose of created tool and gives brief overview of previous research in schema mapping tools. Section 3 presents overall software architecture and user interface of the tool. Detailed description of implementation including technology specific details is given in Section 4. We concluded and discussed about future work in Section 5.

S. S. Cvetković, M. D. Bogdanović and L. V. Stoimenov are with the University of Niš, Faculty of Electronic Engineering, Niš, Serbia (e-mail: {stevica.cvetkovic, milos.bogdanovic, leonid.stoimenov}@elfak.ni.ac.rs).

II. Motivation and Related Work

Our motivation in this work was to apply NHibernateMapper for development of Interoperable Geographic Information System (GIS) [4]. Specifically, its purpose is for support of rapid generation of Data Access Layer using existing ORM solutions. In order to achieve interoperability of developed GIS, domain ontologies could be used as a baseline model in the GIS construction process. Ontologies give us an opportunity to define a meaning of the information and are mainly used during communication of different GIS in order to solve the meaning of the available data. However, ontologies could be also treated as a starting model for the GIS development [13]. By representing ontology in the form of object-oriented classes, ontology concepts can be observed as domain objects that should be mapped to existing relational database. In order to provide object-to-database mapping, we developed NHibernateMapper tool. It is intended for mapping of object-oriented GIS ontology classes into existing databases.

Mapping data between different representations is a common problem which has been addressed from either an algorithmic or a visualization point of view. Much work has been done in this field, differing mainly in the degree of automation achieved. In [5] is given an overview of a variety of approaches to schema mapping and automatic mappings generation. Attempt to automatically determine a mapping between two DTDs is described in [6]. In [7] and [10] are presented approaches for specifying data mapping between two XML schemas using a combination of automated schema analysis agents and selective user interaction. There are also commercial tools like Altova MapForce [8] and Microsoft BizTalk Mapper [9] which could be applied for different schema mapping scenarios including Xml-to-Xml, Database–to-Xml and Database-to-Database mapping. However, they proved to be too generic and complex for learning of specific domain application.

III. NHibernateMapper Architecture

Starting point in the process of *NHibernateMapper* mapping file generation is existing relational database as well as specific domain knowledge in the form of application programming classes. This can be observed in Fig. 1 which represents the NHibernateMapper overall architecture. In order to generate mapping file, the tool utilizes publicly available schema definition of NHibernate mapping file (*nhibernate-mapping.xsd*). This file is the third input element

When using NHibernateMapper tool, the user is required to select an existing database as well as domain object-oriented classes in form of Dynamic Linked Library (DLL). NHibernateMapper will automatically extract database schema of previously selected database and class member attributes of selected DLL.

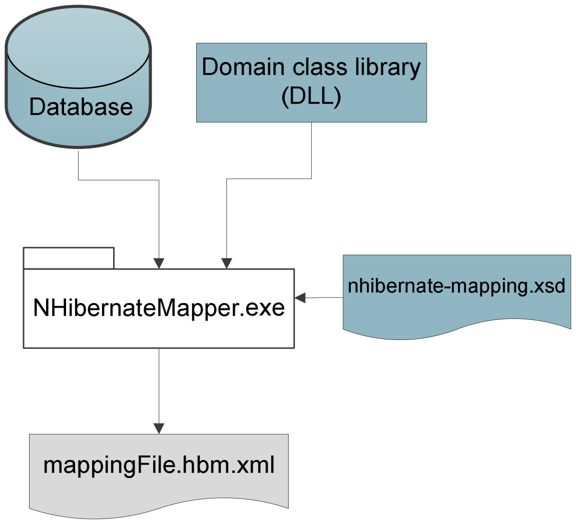
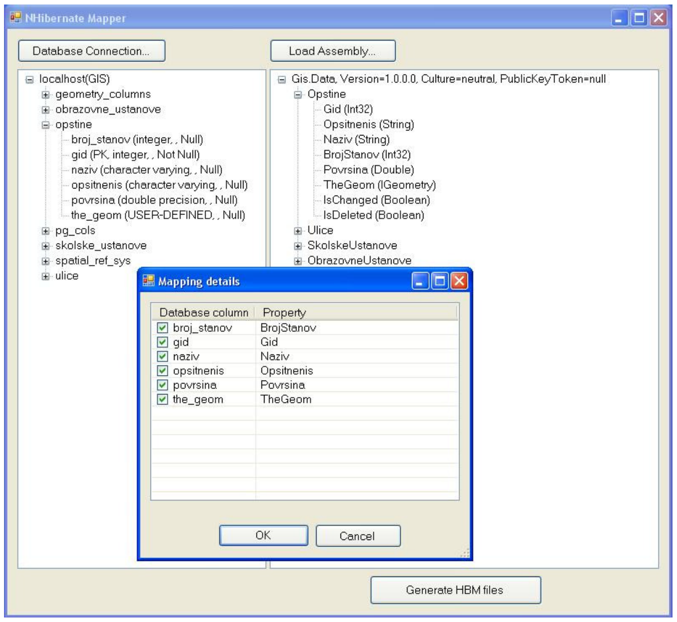


Fig. 1. NHibernateMapper Architecture.

User has to manually select the specific data table and an object-oriented class to be mapped. Then, the database schema and class attributes are automatically analyzed and suggestions of candidate mappings are generated and displayed to the user. Candidate mappings are generated for each column (column-to-attribute mapping), based on string matching algorithm as well as data type comparison. The user is able to accept or reject suggested correspondences. Also, the users can take advantage of the interactive GUI in which suggestions could be pruned into a final mapping. Beside accepting or rejecting proposed mapping, there is also possibility to redefine mappings manually. When the user is finally satisfied with mapping definition, it can be saved to a NHibernate XML mapping file. Illustration of NHibernateMapper GUI is given in Fig. 2.



```
localhost(GIS)
  geometry_columns
  obrazovne_ustanove
  opstine
    broj_stanov (integer, , Null)
    gid (PK, integer, , Not Null)
    naziv (character varying, , Null)
    opsitnenis (character varying, , Null)
    povrsina (double precision, , Null)
    the_geom (USER-DEFINED, , Null)
  pg_cols
  skolske_ustanove
  spatial_ref_sys
  ulice
```

```
Gis.Data, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
  Opstine
    Gid (Int32)
    Opsitnenis (String)
    Naziv (String)
    BrojStanov (Int32)
    Povrsina (Double)
    TheGeom (IGeometry)
    IsChanged (Boolean)
    IsDeleted (Boolean)
  Ulice
  SkolskeUstanove
  ObrazovneUstanove
```

Database column	Property
✓ broj_stanov	BrojStanov
✓ gid	Gid
✓ naziv	Naziv
✓ opsitnenis	Opsitnenis
✓ povrsina	Povrsina
✓ the_geom	TheGeom

Fig. 2. NHibernateMapper GUI.

IV. Implementation Details

NHibernateMapper has been implemented entirely in C# using Microsoft .NET Framework 4.0. As a result, *NHibernateMapper.dll* class library is created. All programming classes in the library could be divided into two major groups: classes for database schema retrieving (Fig. 3.) and classes for mapping file manipulation (Fig. 5).

Class *DbInfo* is a central class of the database schema retrieving part (Fig. 3). It represents object model of database schema. *DbInfo* class obtains and connects schema information including: tables (*DbTableInfo*), columns (*DbColumnInfo*) and constraints (*DbConstraintInfo*).

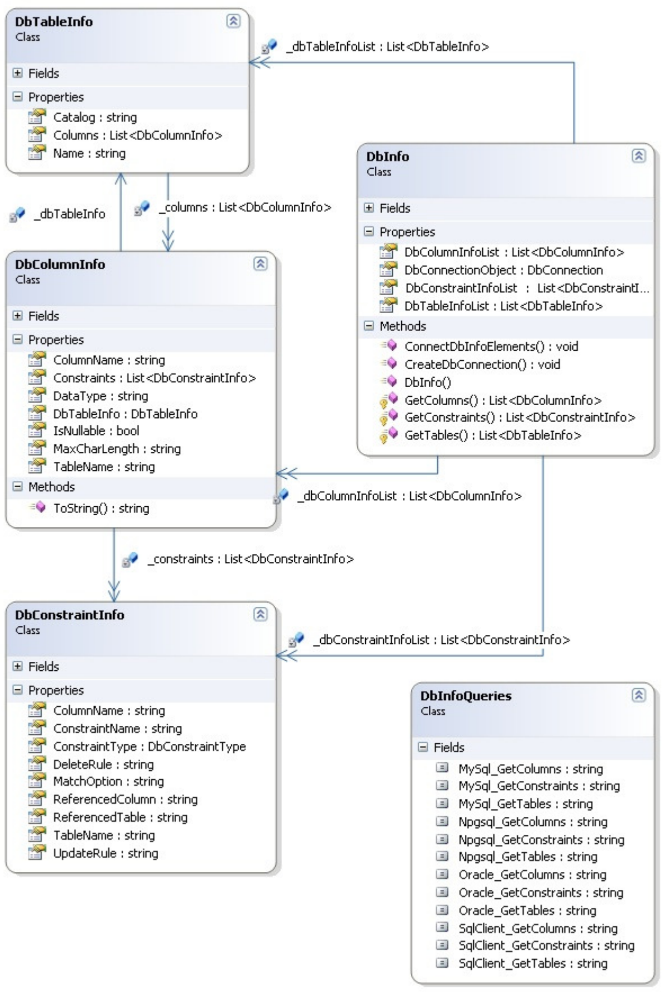

DbTableInfo
Class
Fields
Properties
Catalog : string
Columns : List<DbColumnInfo>
Name : string
_dbTableInfoList : List<DbTableInfo>
_dbTableInfo
_columns : List<DbColumnInfo>
DbInfo
Class
Fields
Properties
DbColumnInfoList : List<DbColumnInfo>
DbConnectionObject : DbConnection
DbConstraintInfoList : List<DbConstraintI...
DbTableInfoList : List<DbTableInfo>
Methods
ConnectDbInfoElements() : void
CreateDbConnection() : void
DbInfo()
GetColumns() : List<DbColumnInfo>
GetConstraints() : List<DbConstraintInfo>
GetTables() : List<DbTableInfo>
DbColumnInfo
Class
Fields
Properties
ColumnName : string
Constraints : List<DbConstraintInfo>
DataType : string
DbTableInfo : DbTableInfo
IsNullable : bool
MaxCharLength : string
TableName : string
Methods
ToString() : string
_dbColumnInfoList : List<DbColumnInfo>
_constraints : List<DbConstraintInfo>
DbConstraintInfo
Class
Fields
Properties
ColumnName : string
ConstraintName : string
ConstraintType : DbConstraintType
DeleteRule : string
MatchOption : string
ReferencedColumn : string
ReferencedTable : string
TableName : string
UpdateRule : string
_dbConstraintInfoList : List<DbConstraintInfo>
DbInfoQueries
Class
Fields
MySql_GetColumns : string
MySql_GetConstraints : string
MySql_GetTables : string
Npgsql_GetColumns : string
Npgsql_GetConstraints : string
Npgsql_GetTables : string
Oracle_GetColumns : string
Oracle_GetConstraints : string
Oracle_GetTables : string
SqlClient_GetColumns : string
SqlClient_GetConstraints : string
SqlClient_GetTables : string

Fig. 3. Diagram of classes for database schema retrieval.

Current implementation supports following DBMS-s: MS SQL Server, MySQL and PostgreSQL. However, configurable design offers possibility to introduce support for any other DBMS which has data provider for .NET Framework. Introduction of support for new DBMS could be done in three successive steps:

- Define three additional queries that retrieve database schema information. These queries (GetTables, GetColumns and GetConstraints) are part of a class DbInfoQueries.
- Copy data provider (in form of DLL) offered by DBMS producer. We have already used MySql.Data.dll for MySQL and *Npgsql.dll* for PostgreSQL.
- Change configuration file so it includes added data provider declaration.

Example of existing configuration file is given in Fig. 4.

```
<system.data>
   ...
      <DbProviderFactories>
        <add name="PostgreSQL Data Provider"
            invariant="Npgsql"
            description=".Net Framework Data Provider for
PostgreSQL"
            type="Npgsql.NpgsqlFactory, Npgsql, Version=2.0.6.0,
                Culture=neutral,
PublicKeyToken=5d8b90d52f46fda7">
        </add>
        <add name="MySQL Data Provider"
            invariant="MySql.Data.MySqlClient"
            description=".Net Framework Data Provider for MySQL"
   type=  "MySql.Data.MySqlClient.MySqlClientFactory,
MySql.Data, Version=6.1.3.0, Culture=neutral,
   PublicKeyToken=c5687fc88969c44d">
        </add>
        ...
      </DbProviderFactories>
   ...
</system.data>
```

Fig. 4. Example of database configuration file.

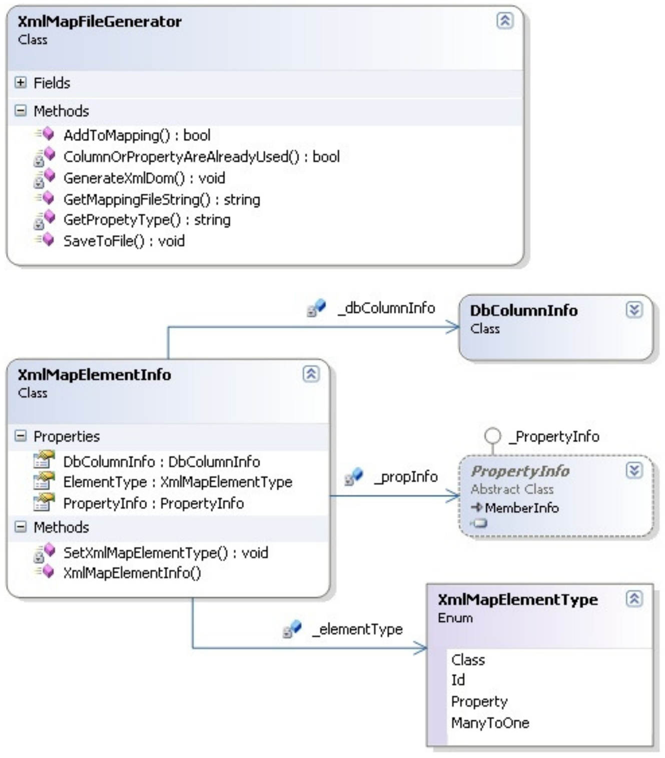


Fig. 5. Diagram of classes used for mapping files manipulation.

Classes for mapping file manipulation are responsible for populating of object model of XML mapping file as well as its storing into file system (Fig. 5). Class *XmlMapElementInfo* manages mapping of one database column with a corresponding class attribute. Central class of this part is *XmlMapFileGenerator* which connects all mapped elements into a whole and enables saving into XML mapping file. It also implements string matching algorithms including “Levenshtein distance” and “Data type correspondence”. Architecture is flexible enough to allow incorporation of new heuristic algorithms.

On a programming level, mapping file is represented by its object model generated from schema definition file. From the developer’s point of view, using schema definition enables validation of mapping file as well as its storing into XML file. Complete process of mapping file generation using previously described programming classes could be summarized in code snippet in Fig. 6.

```
public string GenerateXmlMappingFile()
{
     // Database connection and db schema extraction
    DbInfo dbInfo =DbInfoFactory.GetDbInfo(
    DbProviderInvariantNames.Npgsql, connString);

    //Extract table structure information
    List<DbTableInfo> tableInfoList = dbInfo.DbTableInfoList;

    // Mapping file initialization
    XmlMapFileGenerator xmlFileGen = new
XmlMapFileGenerator();

    // Itearation through tables and columns
    foreach (DbTableInfo tableInfo in tableInfoList){
    foreach (DbColumnInfo colInfo in tableInfo.Columns){
      //Generation of a mapping file element
      PropertyInfo propInfo =
typeof(TestMappingClass).GetProperty("Gid");
      XmlMapElementInfo xmlMapEl = new XmlMapElementInfo(
    colInfo, propInfo);

      //Insert element into mapping file
       xmlFileGen.AddToMapping(xmlMapEl);
    }
    }

    // Return mapping file as string
     return xmlFileGen.GetMappingFileString();
}
```

Fig. 6. Code snippet for mapping file generation process.

As already mentioned, the purpose of developed tool is its application in the field of Interoperable GIS solutions. Therefore we have tested NHibernateMapper with an existing GIS database implemented in PostgreSQL. Example of generated mapping file is presented in Fig. 7.

```
<xml version="1.0"?>
<hibernate-mapping
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="urn:nhibernate-mapping-2.2">
  ...
    <class name="Gis.Data.Ulice, Gis.Data" table="ulice">
      <id name="Gid" column="gid" type="System.Int32" />
      <property name="UliceName" type="System.String"
column="name" />
      <property name="TheGeom"
type="GeoAPI.Geometries.IGeometry" column="the_geom" />

   <property name="Tip" type="System.String" column="tip" />
    </class>

      <class name="Gis.Data.ObrazovneUstanove, Gis.Data"
table="obrazovne_ustanove">
      <id name="IdObrazovne" column="id_obrazovne"
type="System.String" />
      <property name="Adresa" type="System.String"
column="adresa" />
      <property name="Povrsina" type="System.Double"
column="povrsina" />
      <property name="TheGeom"
type="GeoAPI.Geometries.IGeometry" column="the_geom" />
    </class>
  ...

</hibernate-mapping>
```

Fig. 7. Example of generated NHibernate mapping file of test GIS database.

V. Conclusion

NHibernateMapper belongs to the group of schema mapping tools. It is specifically designed for semi-automatic object-to-database mapping specification based on NHibernate ORM. Developed tool allows mapping of information retrieved from existing database with domain model classes. As a starting point for the domain classes definition, ontologies could be applied. Ontologies offer uniform mechanism to define a meaning of the information and represent convenient way for domain model definitions. By representing ontology in the form of object-oriented classes, ontology concepts can be observed as domain objects that will be mapped to existing relational database. NHibernateMapper is primarily developed for application in the field of Geographic Information Systems. This tool could be coupled with an already existing tool that transforms ontology concepts into object-oriented classes [4]. In this way, NHibernateMapper should support generation of Data Access Layer using existing ORM tools. By its semi-automatic mapping generation and coupled with a tool described in [4], NHibernateMapper can significantly contribute to rapid development of Data Access Layer for Interoperable GIS solutions.

Advantage of NHibernateMapper is its configurable design that allows support for intuitive introduction of new DBMS. Currently NHibernateMapper supports only Hibernate mapping file specification. Improvement plan assumes extension to other popular ORM mapping file definitions, like Microsoft Entity Framework. There is also plan for integration of the tool into complete solution for WebGIS application source code generator [14].

REFERENCES

[1] A. Adya, J. Blakeley, S. Melnik and S. Muralidhar, "*Anatomy of the ADO.NET Entity Framework*". In Chan, C. Y. Ooi B. C. & Zhou, A. (eds), Proceedings of the ACM SIGMOD International Conference on Management of Data, June 11-14, Beijing, China, 2007, pp. 877-888.

[2] P. Castro, S. Melnik and A. Adya, "*ADO.NET entity framework: raising the level of abstraction in data programming,*" In Chan, C. Y. Ooi B. C. & Zhou, A. (eds), Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, 2007, pp. 1070-1072.

[3] S. Ambler, *Agile Database Techniques*, Wiley, 2003.

[4] A. Stanimirović, M. Bogdanović and Stoimenov, "*Data Access Layer Generation for Interoperable GIS Environments*," 12th AGILE International Conference on Geographic Information Science, 02-05 June, Hannover, Germany, ISSN 2073-8013, 2009

[5] E. Rahm, P. A. Bernstein, *A survey of approaches to automatic schema mapping*, The VLDB Journal, Springer Verlag, 10: 334-350, 2001

[6] H. Su, H. Kuno and E. A. Rudensteinern, "*Automating the Transformation of XML Documents*," Proc Workshop on Web Information and Data Management, 2001.

[7] L. L. Yan, R. J. Miller, L. M. Haas, and R. Fagin, "*Data-driven understanding and refinement of schema mappings*." In SIGMOD Conference, 2001.

[8] Altova Mapforce, http://www.altova.com/products_mapforce.html

[9] M. A. Goulde, *Microsoft's BizTalk Framework adds messaging to XML,* E-Business Strategies & Solutions, 1999, pp.10-14.

[10] S. Bossung, H. Stoeckle, J. Grundy, R. Amor and J. Hosking, "*Automated Data Mapping Specification via Schema Heuristics and User Interaction*," 19th International Conference on Automatic Software Engineering (ASE'04), 2004, pp. 208—217.

[11] C. Bauer, G. King, *Java Persistence with Hibernate*, Manning Publications, 2006.

[12] OracleTopLink, http://www.oracle.com/technology/products/ias/toplink

[13] F. T. Fonseca, M. J. Egenhofer, "*Ontology-Driven Geographic Information System*," 7th ACM Symposium on Advances in Geographic Information System, Kansas City, 1999.

[14] M. Bogdanović, A. Stanimirović and L. Stoimenov, "*Ginis MvcGen – a Prototype of WebGIS Application Source Code Generator*," ICEST 2010, Ohrid, Makedonija, 2010, pp. 307 – 310.