

## **AN APPROACH TO MODELING AND EVOLUTION OF DATABASE MODEL THROUGH THE ENTITY FRAMEWORK CODE FIRST**

Malcon Miranda Mikami, Keyla Menecrys Sandrino, Maria Salete Marcon Gomes Vaz  
*Universidade Estadual de Ponta Grossa*  
*E-mails: malconmikami@gmail.com; keilasandrino6@gmail.com, salete@uepg.br*

**Abstract:** Applications that make use of modern database are in constant evolution, both in relation to changes in data, database schemas and their business rules. It is a challenge to manage these changes and ensure that everything evolves consistently. This paper presents an approach to the domain -driven development (DDD) and use of an object-relational mapping (ORM) tool that have the ability to evolve database schemas.

**Keywords:** domain-driven development; object-relational mapping; schema evolution

### **1. INTRODUCTION**

The modeling is to create models to explain features and behaviors of a system to represent a simplification of reality. Models created during step of modeling helps the designer to better visualize the system, allowing you to specify the structure or behavior of the system, and provide a guide for their development and document the decisions taken. BOOCH et al. (2000)

ELMASRI and NAVATHE (2005) stated that the conceptual modeling for database is an abstraction of a certain reality transcribed into a concept model supported by graphical models that include details of the database project on an independent level platform.

According to QIU (et al. 2013), unlike traditional applications, the development of database applications is more complex. For example, consider a system that uses a table to store the user authentication information and personal data. If the system requirements change and the system needs to store user authentication information and personal data separately, the original table split into two new tables, for example: USUARIO\_LOGIN and USUARIO\_DETALHES. The data and application code synchronized to be consistent with the new schemes.

The Model-Driven Architecture (MDA) and Domain-driven design (DDD) EVANS and FLOWER (2003), are developing approaches directed by models, which use them in various levels of abstraction starting from a conceptual model. The use of such standards contributes to creating cross-platform software with greater interoperability and easy to maintain, as the models created can be changed, have new features added and be recombined. MELLOR (2005).

According ÖQVIST (2011), the domain-oriented development should not confused with the initial project development in waterfall model, where the great design before any code written. DDD, on the contrary, based on refactoring and iterative methods, since it is a practical impossibility to produce a perfect domain model from the beginning. The model must evolve along with the software and understanding of staff about domain.

To deal with the changes of information required by the business and the necessary evolution of systems, many authors propose approach as Soft Domain-Driven Design (SDDD) SALAHAT (2009), which combines the use of Soft Systems Methodology (SSM), Unified Modeling Language (UML) and the DDD approach using ubiquitous language to facilitate communication between developers and domain experts.

For some time, research in the database area started to worry about support for unconventional applications, which are applications that work with types of non-traditional

data, such as, for example, spatial data, temporal and spatio-temporal Laender et al. (2005). Currently there are several models for geographic database modeling such as UML - GeoFrame LISBON and IOCHPE (1999) and the OMT-G, among others, as well as various tools for conceptual modeling geographic database using the MDA as ArgoCASEGEO + TerraLib GAZOLA (et al 2006) and L-OMT Design FROZZA and SCHALY (2010). Although these tools use the MDA they are linked to specific models and modeling languages and do not use a generic meta- model and cannot be extended to other geographical models, as well as incompatible with the relational model..

## **2. SYSTEMS FOR PRECISION AGRICULTURE**

According WILLERS (et al. 2009) the use of new technology for the management, control of agriculture and agricultural production activities, intends to optimize resources, increase production, quality and profits. Thus, the use of information technology is an important option especially in Precision Agriculture (PA), as the use and application of new knowledge in rural areas help the producer to identify strategies to increase efficiency in the management of crops, maximizing the profitability of crops and becoming the most competitive and sustainable agribusiness Medeiros et al. cited ALONÇO (2005)

There are several solutions on the market to meet the technological demands of use in PA, but not always fully meet the stages of the cycle and / or are proprietary solutions that do not allow modification, or are just software vendors of equipment / hardware that provide their products to market.

It is unlikely that a single proprietary system or open completely meets all areas and processes involved in the PA because of its complexity and breadth. The development of a well-structured skilled architecture, using components "pluggable" and open communication standards in service delivery can be the most appropriate solution for information system construction for use in the PA, however, the difficulty developers to produce a system with easy to understand code, easy to maintain and produce software with agility in the face of change, prevents the production of a product that meets the client's needs and has flexibility.

It is rare to find any application that requires not store any data for future research or to support decision making, whether operational or strategic. The integration of the produced code and the stored data model, through an object-relational framework allows a lower development time, where the framework provides the basic data access capabilities, while developers can focus on application logic. Applications released from fixed code dependencies on particular mechanism or data storage scheme, since support a conceptual model that is independent of the physical model or storage.

## **3. ENTITY FRAMEWORK**

The process of mapping of relational structures in objects compatible with a language or platform known as Object Relational Mapping (ORM). The ORM is a development technique used to reduce the impedance of the programming oriented to objects using relational databases. Classes represent the database tables and records of each table represented as instances of the corresponding classes.

The Entity Framework (EF) is a major tool of persistence present in the.NET platform. Important to highlight the existence of numerous options based on this type technology for

many different programming languages: EJB and Hibernate for Java, Castle ActiveRecord and NHibernate for .NET, CodeIgniter and Zend Framework for PHP.

According to LERMA (2010), the central benefit of EF is that it frees you from worrying about the structure of your database. All your data access and storage are done against a conceptual data model that reflects your own business objects.

With this technique, the programmer does not have to worry about the commands in Structured Query Language (SQL); it will use a simple programming interface that does all the work of persistence. In addition, it is not necessary a direct correspondence between the data tables and program classes. The relationship between the tables where originate the data and the object that makes it available is set by the programmer, isolating the program code of the changes to the organization of the data in database tables.

EF offers solutions to minimize the problem of impedance, developer abstracting many details of the databases, and providing a number of features that greatly increase your productivity. The Entity Framework architecture details in (Figure 1).

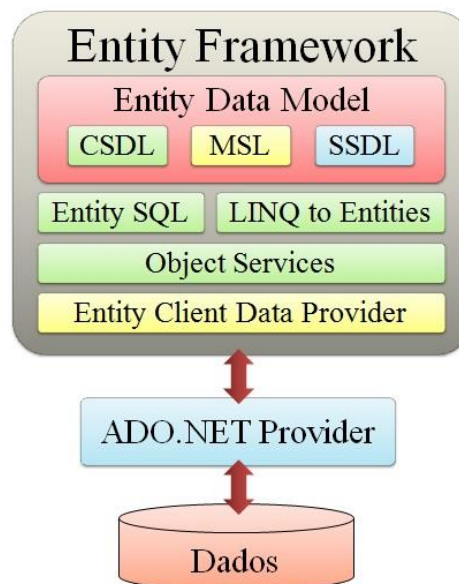


Figure 1 – Entity Framework Architecture

One of the great advantages of the EF was templates allow the creation of databases by way of a graphical tool, eliminating the need for developers to encode one or more classes representing the structures of a specific base.

Although this technique has advantages, such as generating a diagram with all classes that perform object-relational mapping, a new alternative development made available Entity Framework Code First. Here, the first class encoded to represent structures belonging to a database used by the application built.

#### 4. DATA MODELING

The Entity Framework Code First EF-CF second ANDRADE (2013), is commonly used when you want to have greater control for level of source code data model generated because the classes are written using the POCO approach - Plain Old CLR Object (Old and Simple

Object CLR) and then is that the database is generated from these classes, thus presenting full independence with the mapping file.

Developers who follow the principles of DDD where classes encoded first to generate the required database to persist data often use this approach.

To illustrate the use of Entity Framework Code First, we will represent part of the field of precision agriculture. As mentioned previously, within the standard Code First representation tables and views structures made by means of classes that have no direct dependence on the CS.

To describe entities and their attributes, we must make use of the ubiquitous language; so, the concept of the word should be clear to everyone involved in the project. The need for standardized terminology often neglected in the literature; however, it is a matter of critical importance for a number of reasons seconds SZYKMAN (et al).

The first reason is to reduce ambiguity in terms of modeling. Ambiguities can occur when multiple terms used to mean the same thing, when the same term used with various meanings. The distillation of a wide variety of taxonomies in accordance concise eliminates this problem, but this significantly reduces its occurrence.

A second issue related to the uniqueness, not at the level of individual terms interchangeably, but the concept level. This makes the processing of information that been shown to be more difficult, whether it is a human being trying to interpret information modeled by someone else, or algorithms developed design automation or thinking about a project.

The third reason for the development of a standardized terminology is that it increases the uniformity of information within the model, providing a greater degree of consistency within and across design facilitate indexing, searching and retrieving information from them.

In (Figure 2) it is possible to observe examples of classes used in conjunction with the EF-CF. In this particular case, the glebe - “Gleba” and farm – “Fazenda” class have properties whose names and types must generate the database settings.

```
public class Fazenda
{
    public string ID { get; set; }
    public string NomeFazenda { get; set; }
}

public class Gleba
{
    public string ID { get; set; }
    public string NomeGleba { get; set; }
    public int IDFazenda { get; set; }
    public virtual Fazenda Fazenda { get; set; }
}
```

**Figure 2 – Modeling Glebe and Farm Entities**

For the EF understand that these objects are part of the database, we must perform the mapping of entities (Figure 3) and after adding them to the application context (Figure 4)

```

public class MapeadorDeFazenda : MapeadorDeEntidade<Fazenda>
{
    protected override void MapearNomesDeTabelaEColunas()
    {
        this.Mapeador.HasKey( key => key.ID );
        this.Mapeador.Property( x => x.ID ).HasDatabaseGeneratedOption( DatabaseGeneratedOption.Identity );
        this.Mapeador.ToTable( "Fazenda", this.SchemaDeBanco );
    }
}

public class MapeadorDeGleba : MapeadorDeEntidade<Gleba>
{
    protected override void MapearRelacionamentosEChaves()
    {
        this.Mapeador.HasRequired(g => g.Fazenda).WithOptional(f => f.ID);
    }

    protected override void MapearNomesDeTabelaEColunas()
    {
        this.Mapeador.HasKey(key => key.ID);
        this.Mapeador.Property(x => x.ID).HasDatabaseGeneratedOption(DatabaseGeneratedOption.Identity);
        this.Mapeador.ToTable("Gleba", this.SchemaDeBanco);
    }
}

```

Figure 3 – Completing the mapping of entities

```

public class ContextoDaAplicacao : ContextoBase
{
    public ContextoDaAplicacao( FabricaConexao fabricaDeConexao, IFabricaDeDadosParaAuditoria
        : base( fabricaDeConexao, fabricaDeDadosParaAuditoria )
    {
    }

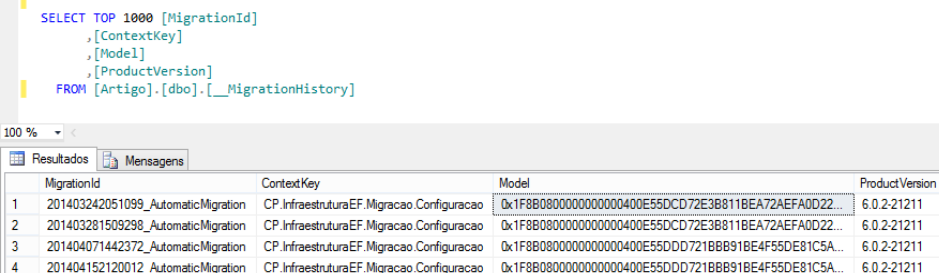
    public override void ExecutarQuandoCriarModeloDoContexto()
    {
        this.AdicionarMapeamentoDeEntidade(new MapeadorDeGleba().Mapeador );
        this.AdicionarMapeamentoDeEntidade(new MapeadorDeFazenda().Mapeador );
    }
}

```

Figure 4 – Adding classes to the context

During the development of a project, changes may occur in the planning, the code and the bank. Changes in the code are simple to get around, because we have tools for code version control, allowing back, joining versions and control the lives of the code. However, the control versions of relational database (RDBMS) is a bit more complex.

With the Entity Framework Code First can enable a property called Migrations (Migration) that with her, we can have versions of the database, go back versions and keep track. The Migrations oversees the POCO classes and creates update methods and downgrade to the required code (Figure 5) to apply the changes.



```

SELECT TOP 1000 [MigrationId]
, [ContextKey]
, [Model]
, [ProductVersion]
FROM [Artigo].[dbo].[__MigrationHistory]

```

MigrationId	ContextKey	Model	ProductVersion	
1	201403242051099_AutomaticMigration	CP.Infraestrutura.EF.Migracao.Configuracao	0x1F8B080000000000400E55DCD72E3B811BEA72AEFA0DD22...	6.0.2-21211
2	201403281509298_AutomaticMigration	CP.Infraestrutura.EF.Migracao.Configuracao	0x1F8B080000000000400E55DCD72E3B811BEA72AEFA0DD22...	6.0.2-21211
3	201404071442372_AutomaticMigration	CP.Infraestrutura.EF.Migracao.Configuracao	0x1F8B080000000000400E55DD721B8B91BE4F55DE81C5A...	6.0.2-21211
4	201404152120012_AutomaticMigration	CP.Infraestrutura.EF.Migracao.Configuracao	0x1F8B080000000000400E55DD721B8B91BE4F55DE81C5A...	6.0.2-21211

Figure 5 – Table \_\_Migrations with the upgrade scripts

For the article in question, we left the Migrations enabled from the start of the project, so the Migrations maintained the database always synchronized with the entities as we can see in the Figure 6.

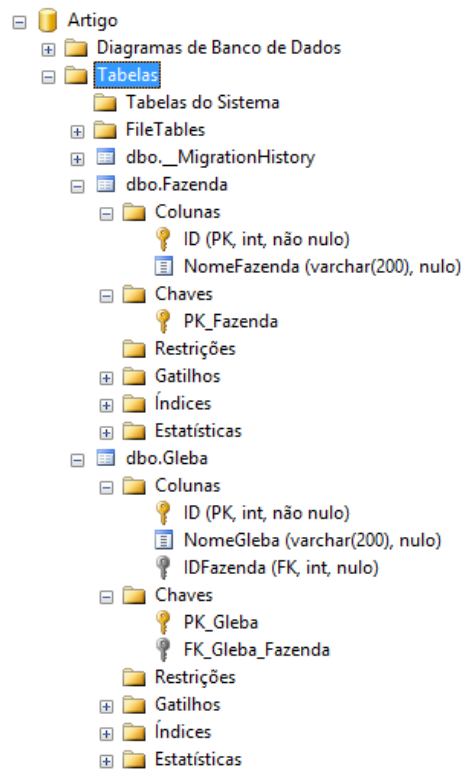


Figure 6 – Database structure

## 5. SPATIAL DATA

According BELUSSI, geographic information systems are more complex and structured than those treated in traditional information systems and this required an evolution of database technology available and the database design approach for working with data space; Moreover, the traditional conceptual modeling approaches have been extended to deal with the complexity and richness of spatial databases to provide:

- Basic spatial data in terms of data structures (e.g., point, line and polygon) and operations (e.g., boundary, perimeter, area, etc.) to describe and manipulate the location and extent of geometric objects included in a reference space 2 or 3 dimensions;
- Spatial relationships (for example, topological relationships such as adjacency and containment) between geometric objects needed for the spatial data query; and
- Spatial integrity constraints between objects space needed to maintain the integrity, and therefore the quality of a spatial database.

The EF can handle geographic data using types DbGeography and DbGeometry types. The type of data Geometry represents data in a Euclidean coordinate system (plan). The type of Geography data represents ellipsoidal data as a geographic coordinate system and stores data such as latitude and longitude coordinates.

To illustrate the use, will add two new properties to the farm class location with type DbGeography data, which will guard the position of office or central position of the farm and area – “area”, the DbGeometry type, which will guard the polygon representing the farm.

For Gleba class, add the property Area, the DbGeometry type, which will guard the polygon representing the plot. (Figure 7)

```

public class Fazenda
{
    public string ID { get; set; }
    public string NomeFazenda { get; set; }
    public DbGeography Localizacao { get; set; }
    public DbGeometry Area { get; set; }
}
public class Gleba
{
    public string ID { get; set; }
    public string NomeGleba { get; set; }
    public int IDFazenda { get; set; }
    public DbGeometry Area { get; set; }
    public virtual Fazenda Fazenda { get; set; }
}

```

Figure 7 – Property DbGeography e DbGeometry

The use of complex spatial data is perceived by EF, and this remained synchronized model represented by data entities in the database (Figure 8).

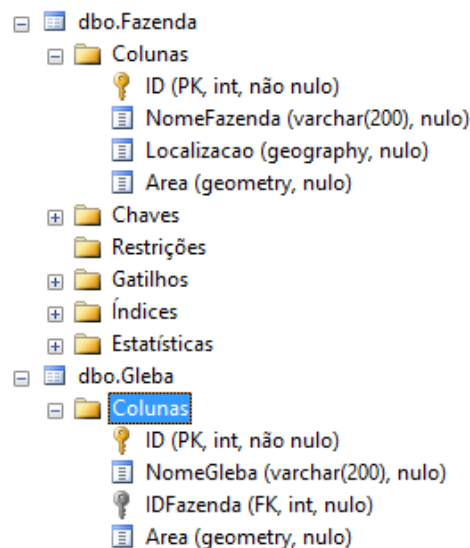


Figure 8 – Database with spatial data

Working with the spatial properties continues to be done in simple and integrated spatial functions of the database. In the Figure 9, we will send a query to the database manager, which should list in a certain amount of farms (n) ordered by the size of their areas in descending order.

```

var query = ContextoDaAplicacao.Fazenda
    .Select(Fazenda => new
    {
        Area = Area.Geometry.Area
    })
    .OrderByDescending(row => row.Area)
    .Take(N);

```

Figure 9 – Seeking in spatial data

## 6. CONCLUSIONS

The application of the field-oriented development within an iterative software development process promises to reduce the inherent complexity of software construction. There are two essential aspects for development. The first is about modeling to capture and distill real-world knowledge in a field of abstraction. The second, the architectural design of the software. It is encapsulating a business model in the global architectural context as well as the logic of structuring the business in the design level.

The adoption of the domain driven design practices depends on the availability of suitable tools, in this case the Entity Framework, which not only allowed the realization of domain modeling, but also address practical issues in implementing the application as persistence and transaction management.

Using the Entity Framework Code First was possible to minimize the problem of impedance between the logical models and physical model application data through constant evolution models. The tool can automatically generate the physical data model in the ANSI SQL standard 92/99/03, compatible with database management systems and geospatial data since their managers have such settings.

## REFERENCES

- ALONÇO, A. S. **Desenvolvimento de um veículo aéreo não tripulado (VANT) para utilização em atividades inerentes à agricultura de precisão**. Congresso brasileiro de engenharia agrícola, 35. Canoas. 2005.
- ANDRADE, W.; **Entity Framework Code Firsts – Primeiros Passos**, Publicado em: Development, 14 out 2013, Disponível em: <<http://sloblog.io/+dev/nIvR5FdWsr0/>> entity-framework-code-first-primeiros-passos> Access at: Dez/2013.
- BELUSSI, A.; Mauro Negri, Giuseppe Pelagatti. **Modelling Spatial Whole–Part relationships using an ISO-TC211 conformant approach**
- BOOCH, G., RUMBAUGH, J., and JACOBSON, I. **UML: Guia do Usuário**. 2000. Editora Campus.
- ELMASRI, R.; e NAVATHE, S. **Sistemas de banco de dados**. 2005. Pearson Addison Wesley.
- EVANS, E.; FOWLER, M. **Domain-Driven Design: Tackling Complexity in the Heart of Software**. Prentice Hall. 2003
- GAZOLA, A., SAMPAIO, G. B., Filho, J. L. Argocasegeo + terralib: **Bancos de dados geográficos para pequenas aplicações GIS**. Workshop de Computação e Aplicações. 2006. Anais do XXVI Congresso da SBC, volume 1.
- LAENDER, A.; DAVIS, C.; BRAUNER, D.; CÂMARA, G., QUEIROZ, G., BORGES, K.; FERREIRA, K.; LIGIANE, V. L., CARVALHO, M. **Bancos de Dados Geográficos**. MundoGEO. 2005.
- LERMAN, J. **Programming Entity Framework**, Second Edition. O’Reilly. 2010
- LISBOA, F.; IOCHPE, C. **Specifying analysis patterns for geographic databases on the basis of a conceptual framework**. 1999. Proceedings of the 7<sup>a</sup> ACM GIS.
- MELLOR, S. J., SCOTT, K., UHL, A., and WEISE, D. **MDA Destilada: Princípios de Arquitetura Orientada por Modelos**. Ciência Moderna. 2005



ÖQVIST, J. **Becoming More Agile With Domain-Driven Design**. Jesper – Oqvist. 2011

QIU, D.; LI, B.; SU, Z. **An Empirical Analysis of the Co-evolution of Schema and Code in Database Applications**. 9th Joint Meeting on Foundations of Software Engineering. Pages 125-135. 2013

SALAHAT, M.; WADE, S. UL-HAQ, I. **Application of a Systemic Soft Domain-Driven Design Framework**. World Academy of Science, Engineering and Technology, 57. 2009

SCHALY, K. W.; FROZZA, A. (2010). **Uma ferramenta para gerar bancos de dados geográficos a partir de diagramas OMT-G**. XI Escola Regional de Banco de Dados.

SZYKMAN, S.; SRIRAM, R. D., BOCHENEK, C.; RACZ, J. W.; SENFAUTE, J. **Design Repositories: Next-Generation Engineering Design Databases**. National Institute of Standards and Technology.

WILLERS, J. L.; JALLAS, E.; MCKINION, J. M.; SEAL, M. R.; TURNER, S. **Advanced in Modeling Agricultural System**. Springer, 257f. 2009