Kappax: A software coding methodology

Carlos Fernandez
TSB - Health and Wellbeing Technologies group, UPV Spain
carferll@eln.upv.es

Abstract

Software coding methodologies are more and more needed to deal with the classical problems related with the software crisis problem.

In this paper, a software coding methodology based on the disconnected model and a tools set in order to take profit of that methodology is presented supported by the experience that TSB-ITACA has in this field.

1 Introduction

Currently, changes in data philosophies as disconnected models, library reutilization and persistent data layers propose new trends on software coding methodologies. Companies that have a big quantity of projects and people flowing continuously must have a solid software methodology to allow them to get used quickly to the source code of a previously started project and to continue the development the easiest and fastest way possible.

Traditionally, the Object Oriented Model was a very good idea to encapsulate entities in order to clarify coding, but, in addition it is needed to be clearer in the specification of those entities, facilitating the serialization, naming correctly classes and functions, defining better the separation between scenarios, etc. The coding methodologies was born to deal with this. Nevertheless, not only is important the way that a code is created for a project is important, but also the software tools available inside a companies workflow is critical to create quality software in less time. Therefore to waste little time to define a good project in order to create common tools or libraries, allows the developer to save a great amount of time to improve the software quality.

In addition to this, those software coding methodologies must be based on a robust tool to define persistent data layers and an easy object serialization, in order to allow the disconnected model idea.

This paper is divided into four sections. First of all the classical framework is presented ir order to have a state of the art current view. Before, the coding methodology is proposed. Section 4 present an internal project in order to provide tools to the developers to improve the coding process. Section 5 concludes the paper.

2 Classical Framework

From the software crisis age lots of software methodologies are appearing in the market[DS90][McC96]. Traditionally, software methodologies are more based in the process on software analysis and design phase than in explicitly in the coding phase. Nevertheless, those software methodologies, acquire soon reticent studies, that had revealed that the hard methodology practice, with they hard schedule exigences and they control techniques, have not results according to the complexity, time, and the money wasted[NA99][TBT00]. At the beginning of century XXI, those reticences was exploded in a manifest called ”Agile Manifesto”[BBvB$^+$01], this manifest are based in the bureaucracy elimination os the software methodology, in order to valuate (a) the people and his interaction over the processes and the tools, (b)the working software over the documentation, (c)the customer collaboration over the contractual negotiation and (d) the change response over the schedule. This manifesto unleashed a dogmatic war that even continuous. In any case, this view, more focused on the development, was make appear numerous coding methodologies in the both worlds [Mic97] [Sta06] [HD05]. The coding methodology standardization is crucial to create maintainable code, easy to understand and to work. In addition, the general tools creation in order to reuse code at different levels in a development group can help to the time to market in a software development strategy at medium place.

Besides, new trends on software architecting are in the market that can be profit with different coding methodologies, in example, disconnected models. Disconnected models allows to get the data and keep in mind without maintain the connection with the server open, instead of keep the connection alive during the data management. In order to profit this idea it is needed in a effective way some design and coding techniques must be applied[Amb04] [Amb00b] [Amb00a].

Figure 1. Microsoft and Mono:: Dataset structure

In the next section a software coding methodology, that can deal with disconnected models,in effective way in time coding and understanding coding.

3 Kappax Architecture

In this section, a new coding methodology is presented, this methodology is based on the expertise on application development that have the TSB-ITACA group. This methodology, called Kappax, is a coding methodology, that is thought to automatize the most mechanical processes, in order to promote the code reuse, make the code clearer, and waste the development time only in the complex cases. Kappax methodology is based on practice issues that propose light changes to the Oriented Object Programming in order to profit the new data paradigms and to solve practical Oriented Object Programming problems.

3.1 DataSets and Disconnected Model

One important model that are more and more present on the market is the Disconnected Model[Esp04]. The Disconnected Model is though to allow build in-memory objects and relate contents coming from different tables of different databases and even from different data sources. When inter-related tables are involved with the process of query and update, code strategies are important to preserve scalability and maintain high performance. Sometimes compound queries can be more effectively accomplished splitting queries; sometimes not.

Figure 2. Microsoft typed DataSet Structure

The DataSet [Cor] concept was proposed and implemented by Microsoft in his .Net Framework. Nevertheless, other platforms, like Java [Soh] and Mono::[Pro] are trying to use the same concept in order to make his developers a disconnected model easy use. DataSets are data and metadata cache that are allocated on memory in order to be used as interchange file between databases and applications. The metadata allocated into the DataSets allow to define a relational view of the data. Internally, those tools are structured into tables, columns and files, having relations between the tables and restrictions as Foreign constraints, Unique key ... etc. In sum, a DataSet is a little database in memory. In figure 1 a DataSet structure is shown.

Moreover, The DataSets has the capability to keep a version of the data. This allow the DataSet able to make transactions, accepting the changes or making rollbacks depending the situations. Datasets have not information about the data sources, there are third controllers that works as drivers in order to get data from several sources and fill the DataSet. In addition, DataSets are serializable usually using XML allowing an easy data transport. In this way, Microsoft has implemented with his DataSet an easy method to create and use DataSet called Typed DataSets. In the figure 2 a typed DataSet example is shown. A Typed DataSet is a DataSet which have their tables, columns, relation and restrictions predefined at design time. This kind of DataSet is defined by a XML Schema(XSD). Using the *XSD.exe* Microsoft tool with the Typed DataSet definition, it is possible to create a class to manage the DataSet easily on design time, improving the coding clarity and making easier the implementation.

3.2 Kappax Model

In the figure 3 a very basic Kappax model architectural view. In that figure, it is clearly defined four layers are well explained following:

Figure 3. Kappax basic Architecture

Helper layer

The helper layer though as a developing framework extension. The helper main mission is to automatize the project common functions not only in a single project but even also in the all development group projects. In this case, there are two main levels where the helper is needed: the project level helper and the development group level helper.

- *The development group kevel helper* is a set of libraries in several languages that are developed in a general framework in order not only to help the all project software implementation but also to mark the development group programming bases. There are lots of advantages of having a general framework to implement application in a group but mainly two: the code reuse and the easy programmer set-up in a new project.
- On the other hand,*the project level helper* is only though in the project general framework. In practice, the general framework must to be upgraded to be included into a particular project. In this case, instead of change the general framework, is more useful to create a new layer over the general framework in order to make more appropriated for this project. This is the project level helper.

The helper must implement functions not only to facilitate the coding, but also to make transparent to the programmer as basic issues as possible. In example, database access, security issues, encrypt/decrypt modules, user interface management, development automatization etc...

Example Model

In order to make clearer the explanation understanding, an example model is proposed. In the example, a simple problem is described, a veterinary clinic, there are animals, that

Figure 4. Example DataBase Model

Figure 5. Example Object model)

can be cats and dogs, and those animals can suffer illnesses. On the figure 4 the database model of the example is defined. It is important to emphasize in the example an inheritance between animal and cat and dog, and a relational table between animal and illness.

On the other hand, in the figure 5 the problem object model is presented as is usually solved on Object Oriented Programming(OOP). At this model, the inheritance continues, and the relational table is changed by an aggregation.

Data Access Logic(DAL) Layer

The Data Access Logic(DAL) is the layer occupied to separate the database model, from the object model. The main objective of this separation is to avoid, as possible, that a database or, in general, datasource change affects to other layers over the DAL layer. In fact, DAL layer is formed by an entity set, that propose views of the data that can be used as normal object by superior layers, but only the entities that are needed. In this way, any abstract class have location in DAL layer. In Our example, the DAL layer will be formed

Figure 6. DAL layer

by three classes: the class DALDog that represent the entity dog, the class DALCat that represent the entity cat, and the class DALIllness that represent the entity illness. The abstract class animal founded on the object model, have not representation in the DAL layer, because, in fact, there are no animals, only there are cats and dogs, and the animal information are included into the inheritance.

The DAL entities are better builded using DataSets, Typed DataSets if possible. The information (located initially in the data sources and posteriorly collected by a external data retriever, probably located on the helper), is stored in a DataSet in the DAL Entity. It is important to emphasize that the DAL entity is the only entity that know what are the data sources and what is the original data format and this information never must be known in superior layers. In addition, only in this layer is allowed access to data sources for read and write.

The data stores in the entity can be required by a superior layer, and in this case, the DAL entity transform the data, in a business format ins order to facilitate the programming in business layers. The view can be stored in a DataSet too.

In the figure6 a DALDog entity graphical representation is shown. In this case, the information collected from the database in two tables(Dog and Animal),is converted to only one, called Dog, that contains info from both tables.

DAL is an easily automatized layer, is important to create functions name guidelines in order to make clearer his understanding. Kappax has guidelines to name functions depending its actions:

- *'Recupera' Functions* are functions that are capable to get information from the data sources and store it into the local DataSet deleting previous information stored in it. Those functions begins by 'Recupera' and finalize with the data required and the arguments needed. In example *void RecuperaCatsbyName(string Name)*

- *'Agrega' Functions* are functions similar to 'Recupera' ones, with the difference that the previous information stores in the DataSet is maintained.

- *'Cachea' Functions* are other 'Recupera' function type. This function is though when a big data quantity must be called on demand. When a 'Cachea' is called, if the data searched is on memory, nothing is done, but if the data in not on memory, a request to data sources is made in order to get this information.

- *'Muestra' Functions* are functions that allow create views from the original DataSet. Those functions begins by 'Muestra' and can continue by the data requires and filters In example *DataSet Muestra()* and *DataSet MuestraCatsbyName(string Name)*

- *'Actualiza' Functions*. This kind of function transport the changes occurred in the in-memory DataSet to the respective data sources

- *'Anyade', 'Modifica' y 'Borra' Functions* are functions that inserts, modifies and delete the data, respectively, into the internal DataSet. This modification is only made on memory, the data modification will not be effective until the 'Actualiza' method was called.

Figure 7. Example Architecture

Business Logic(BL)Layer

In the Business Logic layer, the application core development is made. This layer are formed by objects that represent application scenarios, that access to the DAL entities in order to get the information needed to solve the complete application scenario. A Business Logic object can not call to others object of their same layer, always must call to inferior layers to get the needed information, making if needed several Business Logic Layers in the same application. This is an important thing, because, the call between same layer objects might provoke a very ineffective memory use and complex relations can make the code difficult to understand.

Figure 8. BL Layer Example

In the figure 7 a general architectural view of the example is shown. As can be seen on that figure two main scenarios are proposed, blCat and blDog. Those scenarios, take into account the actions to realize to Cats and Dogs respectively. The blCat class use DALCat and DALIlness DAL entities in order to define his whole scenario. On the other hand, blDog use DALDog and DALIlness DAL entities, to deal with its scenario. In this case, TIlness entity class is shared, this means that DALIlness objects will be created in both Business Logic classes.

Business Logic Classes can make use of the DataSets to create views over the whole class or parts in order to pass this information to superior classes. In figure 8 a Business Layer DataSet example example is presented. This DataSet, that correspond to blDog class, is a composition of data located on DALDog and DALCat objects. Is important to note that the two tables presented are represented into the database as four tables, reducing the complexity. With this, the superior layers are totally abstracted from the Data Access Logic, working with easy data sets.

One important thing that must be mentioned is the plurality problem. The DataSets are thought to store and manage set of entities in a efficient way, in practice, it is very different in an scenario to work with one entity or lots of same type entities. Usually scenarios that manage only one entity data, uses the whole data of the entity, nevertheless, scenarios that build list of entities only use the entity identification data. For this, is better to create plural classes for lists and singular classes to manage entity data. In this way, in our example, blCats and blDogs entities are needed in order to build Cats and Dogs lists.

User Interface Layer

The User Interface Layer use business layer data in order to create user interfaces that build views to the user, showing de data in a determinate format. In this layer can be several presentations of the same data in order to represent this data in different formats(resident application, Web Application ...) in this case all the different interface make use of the same libraries that contains the Business Logic classes

4 OntoDemiurgo Project

As it is mentioned before,it is very important to build a project in order to schedule the creation several helper tools that allow the development group deal in less time with development usual problems. In this section, a project, called OntoDemiurgo, created for the TSB development group needs is exposed.

The OntoDemiurgo Project is a Platform and information systems TSB Area internal project. This project is though in order to help the TSB developer at the coding stage building and sharing common libraries and automatic code generation tools that automatize at the maximum the code process.

4.1 BetComun Library

The Betcomun library is the TSB development group common helper library. This library is continuously updating with all group common code. This library contains lots of functions and algorithms created in .NET and Java, that helps the developer in some environments. Some of the most important functions are listed below:

- *GBBDD Library:* This library is though to facilitate the developer to deal with database access using disconnected model. This library uses DataSets to get and update data from the database implementing Update queries, logs and transactions in a transparent way to the developer.
- *Blowfish:* This library is an easy of use blowfish algorithm implementation, that allow encrypt/decrypt data.
- *GZIP:* This library is an easy of use GZIP algorithm implementation, that allow compress/decompress data.
- *Digital Sign:* This library is an easy of use Digital sign algorithm implementation, that allow sign digitally documents.
- *Process Manager:* This library is a process scheduler.

4.2 K Language

As it was presented before, build DAL classes are very mechanic. This issue makes DAL classes serious candidates to be automatically generated. In this way a subproject of the OntoDemiurgo project is the K language. K is a pseudo-declarative language that is able to generate code automatically, in fact, in the current version, can generate DAL classes in several programming languages, in this case, Java and .NET.

Figure 9. K Architecture

Nevertheless, K language can not be considered Automatic programming, because the language is not a requirements specification and is oriented to help the developer generating code, not substituting him.

4.2.1 K Architecture

In the figure 9 the architecture of the K Compiler id presented. In the figure, it is possible to see the flow that a '.k' file follow to be a compiled code. The '.k' file is compiled in the compilator and converted to intermediate language. Once it is converted, the translator implement the intermediate code to the final language. The K language can support several languages at the same time, even different versions of the same language.

4.2.2 K Code Example

A commented K code example is presented following:

```
string pathraiz = "D:";
string cadenaconexiondisenyo ="";
string cadenaconexion = "";
string proveedorrecuperacion = RecuperaTablaBBDD;
string proveedoractualizacion = ActualizaTablaBBDD;
```

This code correspond to the K file header. At this point is some global variables are specified: 'pathraiz' is the files root where the compiler output is created, 'cadenaconexiondisenyo' is the connection string to the database which the compiler use in order to retrieve the database structure, 'cadenaconexion' is the connection string that are used by the application on execution time, 'proveedorrecuperacion' and 'proveedoractualizacion' are the helpers that will deal whit the database connection on execution time.

```
 clasedal DALPaciente
{
    namespace K_EHaD.DAL;
    usa [K_EHaD.Esquemas.BBDD,K_EHaD.Esquemas.DAL];
    array tablasbbdd =["UHD_Pacientes" ,"UHD_Episodio"];
```

This is the DAL class definition and header, where are defined the 'namespace', the import packages('usa') and some global variables: 'tablasbbdd', which are the database tables located in the class DAL DataSet

```
 esquemaBBDD dsDALPaciente
{
    namespace K_EHaD.Esquemas.BBDD;
}
```

This define the DAL class DataSet Schema. This code uses the global variable 'tablasbbdd' to create the typed DataSet.

```
esquemaXSD dsPaciente
{
    namespace K_EHaD.Esquemas.DAL;
    tabla Paciente
    {
        PK (Episodio);
        ADD UHD_Episodio
            (- Paciente,(string)Codigo as Episodio);
        ADD UHD_Episodio->UHD_Pacientes
            ((string) Id as Identificador,
            (string)
            (Apellido1+" "+Apellido2+", "+Nombre)
             as Nombre);
    }
}
```

This code define the output DataSet Schema referred to the DAL class DataSet Schema. In this case only a table in creates into the DataSet called 'Paciente' with primary key 'Episodio' with the 'UHD_Episodio' table fields, except 'Paciente' and renaming 'Codigo' as 'Episodio', and the 'UHD_Pacientes' table information related to 'UHD_Episodio', renaming 'Id' as 'Identificador' and building 'Nombre' as a string with the fields 'Apellido1','Apellido2' and 'Nombre' separated by a space and a comma.

```
// metodo recupera
metodorecupera  RecuperaporID
{
    Rparam entrada =
    (decimal UHD_Pacientes.Id = id);
}
```

This code generate a ’Recupera’ method that takes as argument the patient ’Id’

```
metodorecupera  RecuperaporID
{
    Rparam entrada =
     (decimal[] UHD_Pacientes.Id= id);
}
```

This code generate a ’Recupera’ method that takes as argument an array of the patient ’Id’

```
metodorecupera  RecuperaporID
{
    Rparam entrada =
    (decimal UHD_Pacientes.Id =id ,
    decimal UHD_Episodio.Codigo = episodio);
}
```

This code generate a ’Recupera’ method that takes as argument the patient ’Id’ and the ’Episodio’.

```
//metodomuestra
metodomuestra MuestraPacientes
{
    string esquemasalida = dsPaciente;

}
```

This code generate a ’Muestra’ method that creates a ’dsPaciente’ DataSet

```
//metodoanyade
metodoanyade AnyadePaciente
{
    // por defecto este es el que se pone
    //array tablasbbdd =["Pacientes" ,"Episodio"];
}
```

This code generate a ’Anyade’ method that adds a patient to the DataSet

```
//metodomodifica
metodomodifica ModificaPaciente
{
    // por defecto este es el que se pone
    //array tablasbbdd =["Pacientes" ,"Episodio"];
}
```

Figure 10. K Execution(I)

This code generate a ’Modifica’ method that modifies a patient to the DataSet

```
//metodoborra
  metodoborra BorraPaciente
  {
      // por defecto este es el que se pone
      //array tablasbbdd =["Pacientes" ,"Episodio"];
  }
```

This code generate a ’Borra’ method that deletes a patient to the DataSet

```
//metodoactualiza
  metodoactualiza Actualizar
  {
      // por defecto este es el que se pone
      //array tablasbbdd =["Pacientes" ,"Episodio"];
              //+ definicion cadenaconexion
  }
}
```

This code generate a ’Actualiza’ method that execute on the database all the changes realized on the DataSet.

In figures 10 and 11 an compiler execution screenshot is shown.

5 Conclusions

In this paper a coding methodology called Kappax is presented, that are in use in the TSB lab, that can be used in order to profit the disconnected model proposed on[Esp04]. In addition a development framework, to take allow incorporate this methodology easily to the TSB group was exposed.

In the future, news tools will be incorporated to BET-Comun library, and the K Language will incorporate news patterns to create more automated code in order to increase more and more the capability of those tools to help the developer.

Figure 11. K Execution(II)

References

[Amb00a] Scott Ambler. The design of a robust persistence layer for relational databases. Technical report, Ronin International, 2000.

[Amb00b] Scott Ambler. Mapping objects to relational databases. Technical report, Ronin International, 2000.

[Amb04] Scott Ambler. *The Object Primer*. Cambridge University Press, 3 edition, 2004.

[BBvB+01] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunnimgham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Agile manifesto. Technical report, 2001.

[Cor] Microsoft Corp. Dataset class: http://msdn.microsoft.com/library /default.asp?url=/ library/ en-us/ cpref/ html/ frlrfsystemdatadatasetclasstopic.asp.

[DS90] Peter DeGrace and Leslie Hulet Stahl. *Wicked problems, righteous solutions*. Yourdon Press, 1990.

[Esp04] Leonardo Esposito. Ado.net best practices. *CoDe Magazine*, 2004.

[HD05] Vic Hartog and Dennis Doomen. Coding standard:csharp. Technical report, Phillips Medical System, 2005.

[McC96] Steve McConnell. *Rapid Development. Taming wild software schedules*. Microsoft Press, 1996.

[Mic97] Sun Microsistems. Java code conventions. Technical report, Sun Microsistems, 1997.

[NA99] Joe Nandhakumar and David Avison. The fiction of methodological development: a field study of information systems development. *Information Technology and People*, (12):176–191, 1999.

[Pro] Mono Project. Mono project documentation: http://www.go-mono.com/docs/.

[Soh] Sampsa Sohlman. Dataset java class: http://dataset.sohlman.com/.

[Sta06] Richard Stallman. Gnu coding standards. Technical report, GNU coding standards, 2006.

[TBT00] Duane Truex, Richard Baskerville, and Julie Travis. Amethodical systems development: The deferred meaning of systems development methods". *Accounting, Management and Information Technology*, (10):53–79, 2000.