ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/261094908

Guidelines for Framework Development Process

Conference Paper · October 2011

DOI: 10.1109/CEE-SECR.2011.6188465

CITATIONS
6

READS
12,438

4 authors, including:

Stanojevic Vojislav
University of Belgrade
19 PUBLICATIONS **41** CITATIONS
SEE PROFILE

Siniša Vlajić
Faculty of organisational sciences - University of Belgrade
44 PUBLICATIONS **69** CITATIONS
SEE PROFILE

Milos Milic
University of Belgrade
9 PUBLICATIONS **14** CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

SilabMDD View project

All content following this page was uploaded by Stanojevic Vojislav on 24 June 2016.

The user has requested enhancement of the downloaded file.

GUIDELINES FOR FRAMEWORK DEVELOPMENT PROCESS

VOJISLAV STANOJEVIĆ, SINIŠA VLAJIĆ, MILIĆ MILOŠ
Software Engenieering Labaratory
Faculty of Organizational Sciences -, University of Belgrade,
Jove Ilića 154, Belgrade, 11000, Serbia
vojkans@fon.bg.ac.rs, vlajic@fon.bg.ac.rs,
mmilic@fon.bg.ac.rs,
http://silab.fon.rs

MARINA OGNJANOVIĆ
Saga D.O.O
Milentija Popovića 9, Belgrade 11000, Serbia
yellowskyday@gmail.com

Abstract in English— **Framework as a term is very frequently used in software engineering, especially in relation to object-oriented software design and implementation. In general, a framework could be defined as an application generator for one particular domain, or more to the point, it represents a skeleton of an application, that includes the complete code for the basic functions of a system, which can be conformed to the needs of one specific application. In this paper, the definitions and properties of frameworks development process are presented. The second part of the paper will present guidelines for design and implementation of framework for developing desktop applications based on relationship meta-data. The framework should, on the basis of an arbitrary problem domain represented by a meta model, generate application skeleton using three-tier architecture and then implement basic CRUD database operations for the defined problem domain. Finally, conclusion summarizes all done and notes guidelines for further improvements of the implemented framework.**

Keywords: Framework; code generation, desktop application; relational model; software patterns; c#

I. INTRODUCTION

Nowadays there is constant growing need for new business software solutions. This indicates that the only way to fulfill market needs is to develop a framework which will shorten software development time.

Term framework is used very often in software engineering, especially when one talks about design and implementation of complex object-oriented software. The most accepted definition for an object oriented framework is: „ a framework is a set of classes that embodies an abstract design for solutions to a family of problems" [1]. Another good definition is "a framework is a reusable design of all or part of a system that is represented by a set of abstract classes and the way their instances interact"[2].

Generally speaking, framework could be defined as a frame of application which is consisted of developed code for all basic functions of a system, which can be adjusted for the purpose of concrete application **[3]**.

One of main reasons for framework development is code reuse. Frameworks are developed to ease reuse of code. First object oriented framework, MVC for Smalltalk, was developed during 80's. Lots of papers and even more lines of code are written since, in order to build perfect framework, but still there is no unique methodology for framework development and documentation, primarily because of its wide domain.

This was motivation for a research taken in software engineering laboratory at Faculty of Organizational sciences. Main goal of the research was to develop a framework, ***called Oz framework***, which will be used to generate an application, based on relational model metadata. One of main obstacles was the fact that there was not a methodology for framework development of this particular domain. Therefore one of the side goals was to give a proposition of framework development methodology. This paper will give insight of the research results. It is important to address that domain of interests are frameworks for small business applications which are using relational database to store its data.

Both, Oz framework and generated applications are implemented using C# programming language. In general, Oz framework generates three tier application with incorporated CRUD (create, read, update, delete) operations on relational database for defined domain model. As implementation of

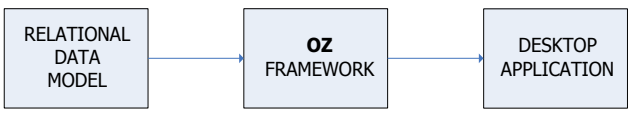


Fig 1. General idea of Oz framework

CRUD operations depends on data model, relational model will be input for this framework, and as written, output will be concrete three tier business application (figure 1).

Software development differs from development of framework. One of main differences is that in analysis phase of software development only specific domain is analyzed. On the other hand, framework domain analysis involve whole domain. Result of software development is concrete application, while result of framework development is a framework which can be used for developing several concrete applications. (Figure 2)
In this paper we will try to summarize guidelines from Oz framework development experience.

Paper is organized as follows: in first section definitions of framework development process, as seen in the eyes of prominent people for this particular area, will be presented. In this chapter authors will give their opinion on framework development process. After it ***guidelines*** for every of framework development phases will be given through example of Oz framework development. In next section framework instantiation is shown with focus on defined **hot spots**. Last section of this paper will summarize all done and at its very end further researches will be proposed.

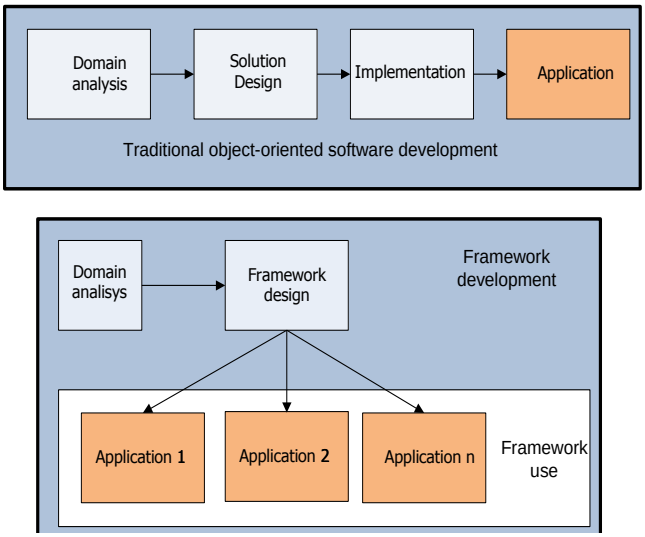


Fig 2 Traditional object-oriented software development VS framework development

FRAMEWORK DEVELOPMENT PROCESS

As said before, there are lots of papers that cover this topic and in next section we will give short overview of important ones.

Markiewicz and Lucena propose classic framework development which is consisted of domain analysis, design, and framework implementation **[9]**. Johnson consider this approach theoretically ideal yet very inefficient and not payable in practice. Therefore he suggests iterative way of framework development according to which while one group of programmers develops framework other one tries to develop application for specific domain using the framework **[10]**. Xavier Amatriain adds he believe in an application-driven approach as that presented by Johnson in which the framework development process is iterative and where the user feedback is necessary on a regular basis. He said that such a process model has proven useful in their case. He also stated that it is not just unnecessary to have a previous domain analysis model before starting the framework design but it is the framework design that ends up generating an appropriate domain meta-model. **[7]**. He doesn't exclude the fact that, at very beginning, some king of domain analysis is necessary in order to understand basic requirements. Taligent company states that it is best to develop small, focused frameworks, from existing solutions, iteratively, with complete documentation and backup **[7]**. Bosch dismembers two different activities in framework development: development of ***framework core*** and development of ***internal add-ins***. Core exists, unchanged, in every framework instance (every single use of framework) **[8]**. Core is consisted of **frozen spots**, in other words it is consisted of interfaces and classes which are already implemented in framework and as such are used in every framework instance. On the other hand, parts of code that are generated for every concrete application (specific domain model) are ***internal add-ins***. In case of Oz framework these classes are generated from relational model metadata. Parts of code which makes code flexible are called ***hot spots***. These hot spots in most cases are some abstract classes or methods that must be overridden **[8].** Final application, generated using framework, is consisted of **framework core**, **internal add-ins** and application specific code (**implemented hot spots**).

From the ***Oz framework*** definition it is obvious that the framework domain is very wide. Therefore, one of main tasks in Oz framework development of is to determine architecture of applications which the framework should generate. The architecture must be reusable and very flexible. Designing object-oriented software is hard, and designing *reusable* object-oriented software is even harder. Design should be specific to the problem at hand but also general enough to address future problems and requirements. You also want to avoid redesign, or at least minimize it " **[9]**. All authors have same opinion, which we share, that use of design patterns[1] is inevitable in framework development.
Patterns that are identified as significant in framework development are: **[10], [7]**

- Pattern of three examples
- white box
- component library
- Hot spots pattern

[1] In software engineering, a **design pattern** is a general reusable solution to a commonly occurring problem in software design. A design pattern is not a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations. [3]

- Pluggable objects
- grained objects
- black box
- Visual builder
- Programming language tool

In analysis phase of ***Oz framework*** development ***pattern of three examples*** is highly used and was very helpful. That pattern suggests that domain definition should be made using generalization of three concrete applications. That enables optimum in domain span and gives good grounds for abstractions. ***White box pattern*** suggests use of inheritance as simplest way of making and changing code in object-oriented software development. ***Component library pattern*** suggests that one should create component library consisted of components which are used in most applications (for example some utility classes). This can simplify use of framework in large sense. ***Hot spots pattern*** shows that code which will be changed for every application should be located in few classes. This will simplify framework use, and users (programmers) will impeccably know where they must make changes of code for specific application generated by framework. These patterns are used in development of Oz framework.

FRAMEWORK DEVELOPMENT PROCESS

In this section we will present the way that Oz framework was developed. As said before, Oz framework ought to generate a ***three tier application*** which will implement CRUD operations for specific domain.

After short research the Bosch's method is evaluated as most suitable for development of the Oz framework.

Primary objective was to determine architecture of desktop application which will be created using the framework. It was necessary to do some abstractions of existing application architectures in order to reveal which components will be the same in every created application, and as such will be part of ***framework core***. It was also necessary to determine components that will vary in every application created by the framework. These generated components will make the ***internal add-ins of framework***.

Existence of many finished applications was crucial in choosing Bosch method to be starting point for framework development process determination.

Oz framework development had these phases **[8]**:

- **Framework development** which has following sub-phases:
  - **Domain analysis** – in domain analysis, as described in ***pattern of three examples***, three applications were analyzed to determine general architecture of applications that will be created by Oz framework. Result, at the end of this phase of framework development, were classes which made ***framework core*** and classes that made ***internal add-ins***.
  - **Framework design** – was done according to Larman's method **[10]**. Besides framework architecture design, one of most important things during this phase was to determine all necessary transformations for generation of components that are part of ***internal add-ins***.
  - **Implementation** – framework was implemented using C# programming language.
  - **Framework testing** – this phase of software won't be considered in this paper.
  - **Framework documenting** - this is the phase that Bosch didn't isolate, but lots of other researchers point out that this is also very important phase.
- **Framework use**– it was conducted as case study, and overview will be shown
- **Evolution and maintenance of framework** – further evolution exceeds limits of this paper

A. Domain analysis

First question relevant in this phase of framework development is:

How to start domain analysis?

It the case of framework for creation of small business applications one should decide to do it only when there are more than two applications made in the domain of interest. It is also interesting to take a look at Xavier Amatriains interpretation of how Swiss Ericsson company model the framework development process. He stated that "a list of requirements on at least two applications should be provided together with a list of requirements on the framework. A list of future requirements should also be provided. The project team should include members with knowledge of each application and a member with knowledge on framework design. Information should be gathered from as many sources as possible. Requirements and use cases should be separated into framework-specific and application-specific and they should also be divided into functional and non-functional. High-level abstractions should be identified, preparing for the identification of the framework. But only abstractions that are in the framework domain should be introduced and afterwards they should be named the same in the static model". Finally existing frameworks should be studied trying to reuse as much design as possible. Therefore best answer to this question is to use pattern of three examples, which recommends that the domain analysis should be based on ***three specific desktop applications***. Following this guideline e.g. the pattern of three examples, three specific applications were analyzed to insight similarities and differences in order to do the abstractions.

Software systems that were analyzed were:

- Application "***Shoe store***" for storehouse management, developed for Mastaco company

- Application "***Admission test***" for organizing admission test, developed as seminar paper for course "Information system Design"
- Application "***Restaurant***", for ordering

Objective of this phase is, as said earlier, to define general architecture of business applications which the framework should generate. As main tool for determining which components will be part of the architecture, we will use patterns in their general sense.

Analysis of architecture components

In this section all parts of three tier architecture are analyzed (figure 3). These components were built using generalization of classes from the three specific applications. Perceived concrete classes which were exactly same in every application were added to ***component library***.

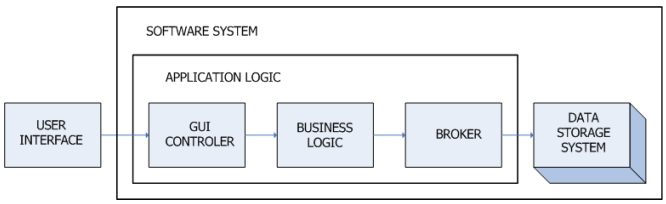


Fig 3 Three tier architecture – components

Goal of component analysis is to get interfaces, abstract or general concrete classes for all components in question.

Firstly, elements of **graphical user interface (GUI)** from all three applications were analyzed. All **GUI** controls and methods that are called in background were also analyzed, and abstractions are made in order to determine interfaces and general class which will be inherited by every concrete form. In general graphical forms are used to represent data from database to the end user and as such are highly correlated to database meta-data. This indicates that database relationship model already contains almost all information need to make ***code generator,*** which can generate classes that will make ***internal add-ins***. Only information that one needs to provide are those related to component which will display data for each field from database (text field, drop down menu, list box, date chooser…) although some generalizations can be made based on field data type and foreign key constrains. All forms have some similar behavior and that can be abstracted to a superclass. As Amatriain also said existing solutions should be examined and large frameworks should be divided into sub-frameworks. Therefore at a beginning one should focus on only one presentation technology because it is easier to built small focus frameworks a then to do abstractions.
A static model for each application should be developed then introducing abstractions common to several applications into the framework. Subsystems should have high cohesion and low coupling, and as said before, only way to achieve this task is to use ***software patterns***.

After that, ***GUI Controller*** was analyzed in order to get ***abstract GUI controller*** which will be inherited by GUI controllers for every form. As it has function to map values from components to domain objects and vice versa one can easily make code generator based similar to the GUI code generator.

In next step, classes that make ***business logic*** were analyzed in order to determine general ones. One of the most important parts of every application is ***business logic***. Requirements are very important for this part of software development. In early phase of software development requirements are made in form of Use-Case or user stories. On this basis one identifies system operations and writes contracts for it. In each contract one must define its signature, pre and post conditions. Every application have to enable CRUD operations and it is possible to do abstractions and make generic system operation for them. To do this in proper way one should use software patterns, especialy **template method pattern.** This is obviously a place where one should identifiy and make ***hot-spots*** of a framework.

Next task will be ***domain class*** analysis. All domain classes can be easily generated based on database meta-data. For this purpose one can use some of already existing solutions, for instance some IDE built-in entity class generators for some persistent mechanisms.

In final step of architecture analysis database broker should be analyzed. One of objectives is to define set of generic methods for database broker that are same (most common) for all concrete applications in question. In other words interface for database broker should be one of the results in this stage. Another result should be to determine implementations of database broker for concrete database management systems.

B. Framework design

The objective with this activity is to end up with a flexible framework. The activity is effort consuming since it is difficult to find the right abstractions and identify the stable and variable parts of the framework, which often results in a number of design iterations. To improve the extensibility and flexibility of the framework to meet the future instantiation needs, design patterns may be used. The resulting framework can be a white-box, black-box or visual builder framework depending on the domain, effort available and intended users of the framework.

One should base framework design on grounds of previous phase. One of the key contributions of previous phase is separation of classes that are part of **framework core** and those that make framework **internal add-ins**. Abstract classes and interfaces that makes framework core are usually organized as **component library.**

Next step in this stage of framework development is to determine whether is there a way to define a ***code generator*** for classes which will be part of ***internal add-ins*** in every framework instance, although this can be done manually in every framework instantiation. Thought this is in a way already done in previous stage of framework

development here one should determine and design all necessary transformations which will be used during code generation. There are lots of tools, like FreeMarker, Apache Jakarta Velocity etc., which can be helpful for code generator development. Code generator can be observed as a kind of application, therefore it can be created using any of existing software development methods. One of key differences from classic software development is inputs e.g. domain. As mentioned before ***database meta-data*** will be inputs for this application. In a raw framework there will be just a few addition data that user of a framework have to provide. One can call this as framework fine tuning.

One of conclusion that can be made in this point is that there is not a strict line between domain analysis and framework design phase. This is reason why some authors do not point out domain analysis as a phase of framework development.

Oz framework design phase

In the case of Oz framework development during the design phase of framework development both component library and code generators are designed on basis of domain analysis phase.

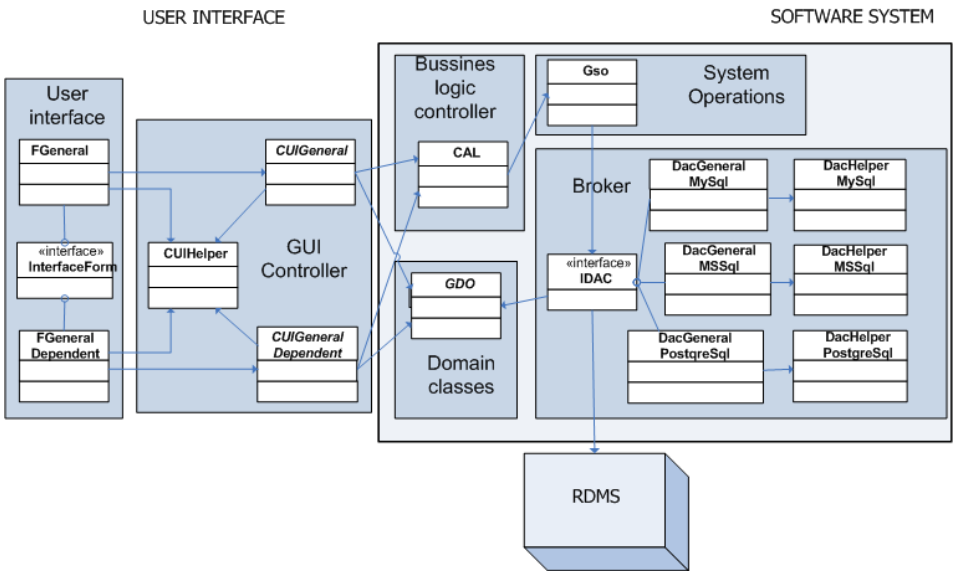


Fig 4. Framework core architecture

Framework core is consisted of (figure 4.):

- User interface
  - Forms
    - *FormInterface* - interface
    - *FGeneral*, *FGeneralDependant* – general class
  - CUI Controller
    - *CUIGeneral*, *CUIGeneralDependant*– Abstract class that has general methods for CUIControllers
    - *CUIHelper* – class that provides services for form and GUI controllers
- Application logic
  - Controller
    - *Cal* – concrete class
  - Business logic
    - System operations
      - *Gso* – general class with general behavior
    - Domain classes
      - *Gdo* – abstract domain class
  - Database broker
    - *IDAC* – database broker interface
  - *DacGeneralMySql*, *DacGeneralMSSql*, *DacGeneralPostgreSql* – concrete classes that implements IDAC interface according to particular database management system
  - *DacHelperMySql*, *DacHelperMSSql*, *DacHelperPostgreSql* – helper classes for database broker
- Helper classes
  - *Log* – class used for error logging
  - *Validation* – class used for validation

According to conducted analysis and design figure 5. shows relation between core and internal add-ins. Classes that are painted in orange are part of ***framework core***, and those that are painted with white color are part of ***internal add-ins***.

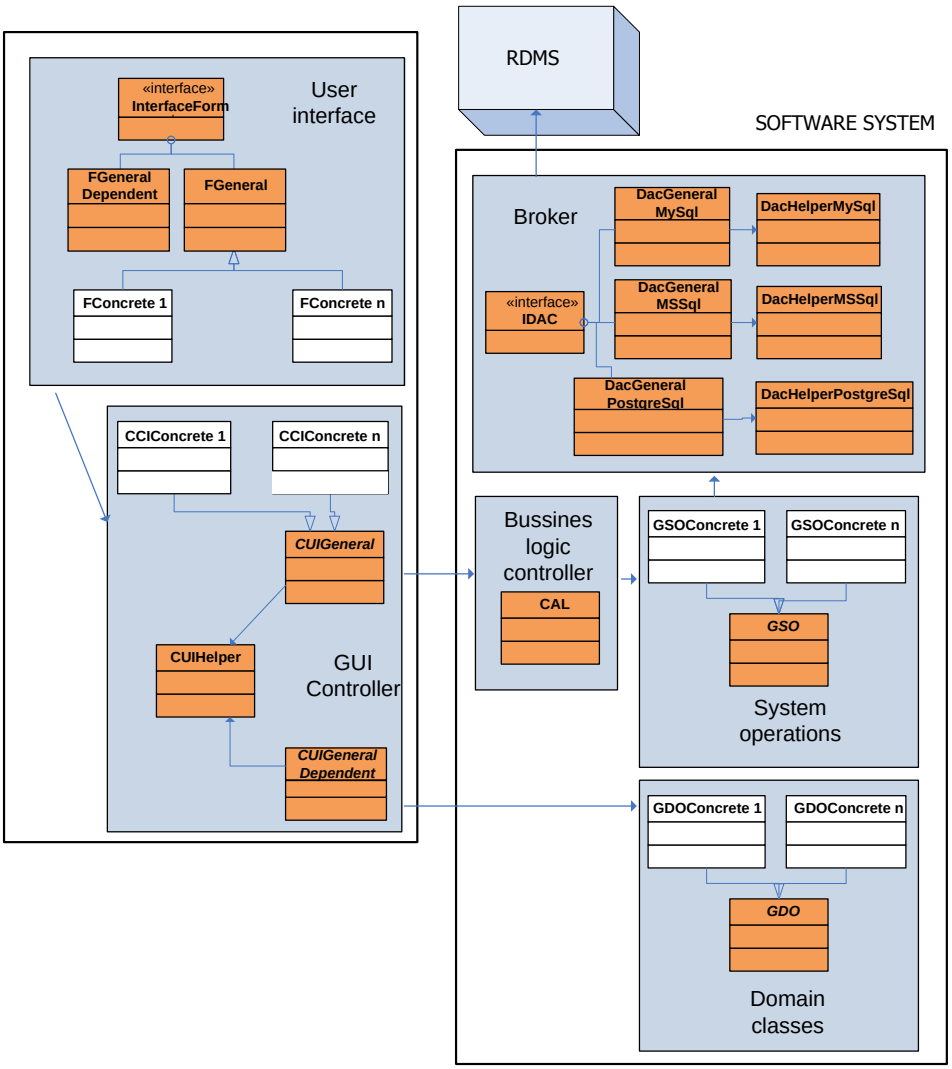


Fig 5. Relation between framework core and internal add-ins classes.

As written next task was to determine transformations needed to develop required internal add-ins classes. In this stage of Oz framework development class templates were designed and developed for code generator. There are lots of open source tools which can be used for this. We mentioned some in previous text.

Although someone will find unnecessary Larman's software development method was used for code generator development. Based on requirements determined during domain analysis and starting phase of framework design code generator was developed.

In order to give insight to code generator, design of system operation createForm will be shown. All other system operations are designed in same way.

Contract: ***CreateForm***

Operation: ***CreateForm*** (File Form)
Connection to UC: SK1
Pre-conditions: System is connected to database
Post-conditions: Graphic forms are created

Graphic form is created for every table from underlying database; only forms of dependent tables will be excluded. Form will have the same name as form designer file. Every form will inherit ***FGeneral*** form which is part of ***framework***

core. Only methods that are called on form opening and method for disabling components are overridden. Parameterized constructor is added which will set reference for proper CUI Controller. Figure 6. shows this system operation.

- Helper methods – purpose of these methods is to enable/disable controls on the form

```
public static string CreateHeader(string className, bool dependent)...

public static string CreateConstructor(string tableName, DataTable tblShema, bool dependent)...
public static string CreateEventMethods(string tableName, DataTable tblShema, ArrayList dependent
public static string CreateDataGridViewMethods(string tableName, DataTable tblShema)...
public static string CreateFieldManipulationMethods(string tableName, DataTable tblShema)...
// ...
```

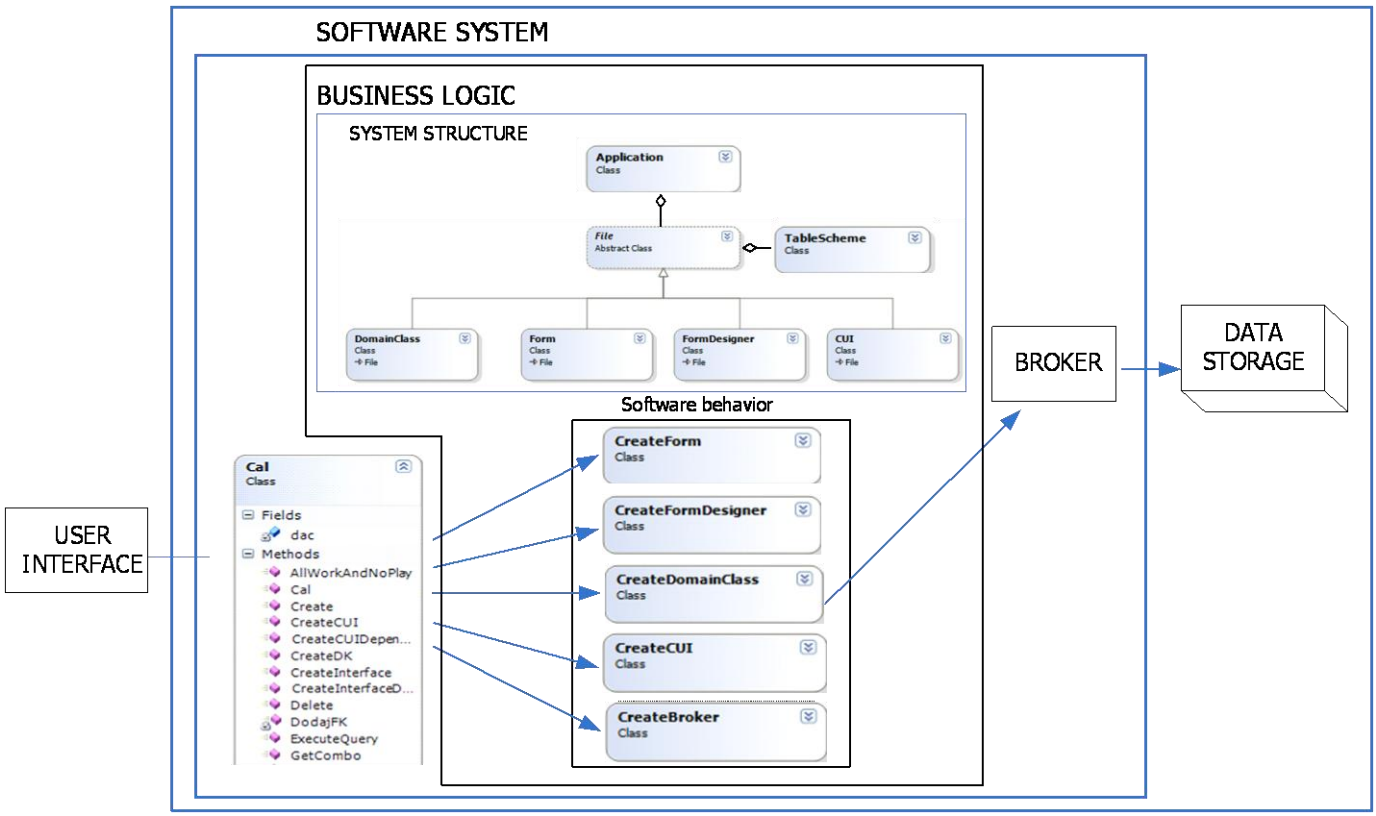
Fig 6. CreateForm system operation methods

Region of ***CreateForm*** class are as follows:

- Header
- Constructors – are called when form is created. They are in charge to initialize all form controls (components) and to create instance of particular GUI Controller

Oz framework code generator architecture is shown on Fig 7.

Fig 7. Source code of *CreateForm* system operation



C. *Framefork instatiation*

Framework development process is iterative and incremental. In every framework use user will see all good and bad sides of framwork and this will be starting point for next increment.

In most cases user will have to provide additional information during or after framework instantiation.

During the use e.g. test of generated application user should determine potential hot spots which can be made in next increment of framework development.
Next section will show the strength of designed hotspot.

D. *Hot spots*

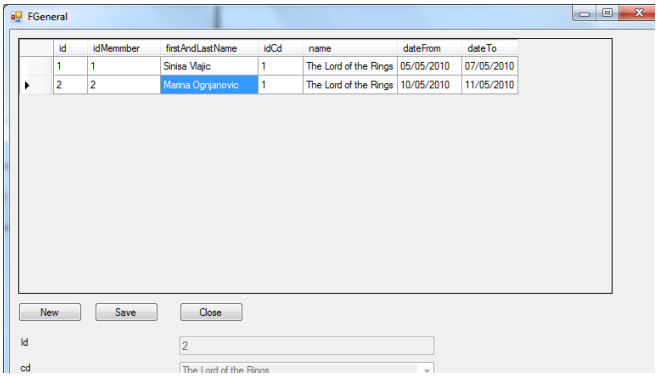
In order to change application functionalities one must override generic methods of system operation with concrete one, change generated ones (not recommended) or create completely new classes and integrate them in generated application. This should be done very carefully because every change can have impact on lot of things especially if user tries to change any generic method, therefore framework should enable code change only in specified (hot spots) methods of generated code.

Further text will show changes of two functionalities in "CD club" application generated by Oz framework.
It can be noticed that list of rents shown on form is not so intuitive, because end user of program won't be satisfied with fact that he/she can see only id number both for CD and a member. In order to change it programmer must override method ***GetSelectAll()*** in ***DK.Renting*** class.

Table 1. Necessary changes in GetSelectAll method

```
public override string GetSelectAll(){
          StringBuilder sb = new StringBuilder();
          sb.Append("select ren.id, ren.idMember,
memm.firstLastName, ren.idCd, CD.name,ren.dateFrom,
ren.dateTo from ");
           sb.Append(TABLE_NAME);
           sb.Append(" as ren inner join CD on
ren.idCd=CD.id inner join member as memm on
ren.idMember=memm.id");
           return sb.ToString();
       }
```

Fig 8. Look of "Update membership card" form after code modification



Now, shown list is clearer to end user (figure 8)

FINAL CONCLUSION

Objective of this paper was to show guidelines for development of framework which is used for generation of three tier desktop applications based on relational metadata. After brief introduction and presentation of various framework definitions, ***patterns*** that are widely used in framework development were identified. Most widely used patterns are **pattern of three examples, white box, component library, hot spots, pluggable objects…**
Proposed framework development process is based on Bosch's method for framework development, which is consisted of ***Domain analysis***, ***Framework design***, ***Implementation***, ***Framework testing*** phases **[8]**.

In domain analysis authors suggest that one should use ***pattern of three examples,*** especially for type of framework in question. Objective of this phase is to do abstractions in order to get classes that will be unchanged in every generated application and those which will differ depending on specific domain model. This should be done using ***software patterns***. On the example Oz framework authors has given guidelines how the component of three tier application should be analyzed and all necessary abstractions are done. As a result of analysis components that are part of ***framework core (application frame)*** (they are same in every generated application) and those that depends on relational metadata and as such are part of ***internal add-ins*** should be identified**.**

Framework design, or more concrete the code generator**,** is done using Larman's method of software development. Five system operations were determined:

- CreateApplicationFrame (Application app)
- CreateFormDesigner (File FormDesigner)
- CreateForm(File Form)
- CreateCUI(File CUI),
- CreateDC(File DomainClass)

Hot spots are also noted at the end of this paper, and authors gave guidelines how user can use these hot spots to change functionalities of generated applications.

REFERENCES

[1] Johnson, R. E. and Foote, J. (1988). Designing Reusable Classes. *Journal of Object Oriented Programming*, 1(2):22-35.

[2] M. Mattsson, J. Bosch, Characterizing Stability in Evolving Frameworks, I*n Proceedings of the 29th International Conference on Technology of Object-Oriented Languages and Systems*, TOOLS EUROPE '99, Nancy, France, pp. 118-130, June

[3] Dr Siniša Vlajić: *Projektovanje programa (skripta)*, FON, Beograd 2004

[4] Marcus Eduardo Markiewicz, Carlos J.P. Lucena: *Object Oriented Framework Development, "* Crossroads*"* Volume 7 , Issue 4 (June 2001), ISSN:1528-4972

[5] Don Roberts, Ralph Johnson: Evolving Frameworks, A Pattern Language for Developing Object-Oriented Frameworks, http://st-ww.cs.illinois.edu/users/droberts/evolve.html

[6] Xavier Amatriain: CLAM: A Framework for Audio and Music Application Development. IEEESoftware 24(1): 82-85 (2007)

[7] Taligent (1994): Building Object-Oriented Frameworks, A Taligent White Paper. https://lhcb-comp.web.cern.ch/lhcb-comp/Components/postscript/buildingoo.pdf

[8] M. Mattsson, J. Bosch, M. Fayad: Framework Integration: Problems, Causes, Solutions, Communications of the ACM ,Volume 42 , Issue 10 (October 1999), Pages: 80 – 87, 1999, ISSN:0001-0782

[9] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Design Patterns:Elements of Reusable Object-Oriented Software, Addison-Wesley Pub Co; 1st edition (January 15, 1995)

[10] Craig Larman: *Applying UML and Patterns*, Second edition, Prentice Hall, ,New Jersy, 1998

View publication stats