DOI 10.34185/1562-9945-1-144-2023-11
УДК 614.2+574/578+004.38

O.A. Lytvynov, M.O. Lytvynov

ON APPLICATION OF FRAME-BASED APPROACH FOR INFORMATION SYSTEMS DEVELOPMENT

Annotation. The work is devoted to specific of frame-based approach application for information systems development. Three-layered generator provided to make the process of multi-layered information systems development more effective and robust. The results of its application in the context of building more flexible and robust, testable and maintainable software are also discussed.
Key words: frame-based, software development, code-generator.

Importance and relevance of the research topic. Decade-to-decade and even year-to-year the complexity of software keeps growing. Today's business requires more than just an information system that responds the immediate needs. It requires more flexible and agile, maintainable, and testable, robust and scalable IT infrastructure able to meet the challenges of the business. How to design, realize and maintain such systems? What principles, architecture and patterns should be used by the developers to tackle that overwhelming complexity? These questions are stated before developers, managers, scientists.

Multi-layered and service-oriented [4] software architectures, domain-driven design approach with its modifications [1-3], SOLID principles [5], TDD [6] and BDD [7] approaches can be seen as an answer to the question mentioned above. But the implementing of such approaches results in dozens of huge projects with hundreds and even thousands of classes and sophisticated testing infrastructure necessary to support continuous refactoring of the system. It complicates the process of development and maintenance, increases the level of mental load on personal.

To resolve the problem of software complexity, some authors propose to bet on the quality of the development process rather than on individuals and technologies. According to [8] the quality of the product depends on the quality of the process. The process can be defined as a set of defined, structured activities that will accomplish a specific organizational goal. The improvement of the process is based on applying of specific activities directed to identify, standardize and implement best practices,

project's lessons learned in the organization. From this point of view, the standards become the assets of the organization, which build a foundation for strategy to achieve the robustness of the process. Making the solutions standardized leads the organization to the condition when an incoming requirement (functional, system etc.) causes the retrieval of a set of typical tasks and solutions from, for example, a kind of ontology (e. g. terminological knowledge base). One of the issues to implement that approach is the dependency of the solution on a variety of details linked with the programming languages, technologies and frameworks which are used to realize the system components. These dependencies block the port of the solution to another platform, languages and technologies and make the solution difficult to understand, even when the code is well structured and self-documenting.

One way to overcome such problems is to represent the solution as a set of models going down from more abstract business-oriented models towards more detailed and platform-specific ones. It is the way of model-driven architecture (MDA) provided by OMG (Object Management Group) consortium. The key ideas of the approach are as follows: the description of the system by the models on different levels of abstraction connected to phases of a software development cycle; the producing of the low-level models from the high-level ones through a set of transformations leading by the defined transformation rules. According to MDA there are four levels of system description: computation independent model (CIM), also known as domain model, used to define business processes of the organization for which the information system (IS) will be developed; platform independent model (PIM) defines the system architecture and specification can be implemented in various platforms; platform specific model (PSM) offers detailed technical specification of the system and can be transformed to implementation model (code) according to transformation rules automatically or semi-automatically; implementation model (IM) – a set of artifacts represented using programming languages define the information needed to create IS. The main problems of MDA appeared in practice are as follows: the lack of CIM formal description; accordingly, inability to build effective CIM-PIM transformation; behavior description and the completeness of the generated code. Creation of CIM level is not unified but it is assumed that this level is represented by the model of business processes. For example, in [9] authors suggest using of DFD (data-flow diagrams) as the language to describe CIM model. Evaluation summary of the methods described in [10].

Taking in mind that the process of information system development is, in fact, the process of transformation of customer-level requirements and the quality attrib-

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

utes in a set of interacting software components structured and defined in accordance with the architecture, principles and technologies, operating in the operating environment, we face two main questions:

How the requirements and solution specific should be defined in order to be effectively transformed (regarding the previous experience) into a set of artifacts, helping developers to realize software components as quickly as possible.

What the origin is and how to build the transformation mechanism.

To answer the first question, we should define the minimal description able to represent all necessary information about the domain to build an effective transformation. Next, we should define what are the artifacts could help us to build the project rapidly. And last, how we can effectively reuse the experience acquired in previous projects and how to build the transformation engine (e.g., the unit responsible for transformation of the model into the set of artifacts).

Task definition. The results of previous research were published in [11-13]. The provided minimal description is based on domain description using high-level language and then can be transformed into a set of artifacts. For this purpose, we propose to use an extended frame-based language described in detail in [12]. This language can describe domain in a full, platform independent manner, considering all necessary information that could be effectively used by the transformation mechanism for system construction and maintenance. The expressiveness of the language is based on the descriptive power of facets used by both frame and slot structures. The model is used as a foundation for: component generation, allowing to speed up the development process; run-time model interpretation, allowing to exclude the necessity to build and test the classes.

We can think of the frame as a set of projections, some of which represent more abstract information that can be easily understood by the experts, some form more detailed view needed for generation of more complete and valuable assets. The frame described in that way as of the complete synthesis-oriented structure able to incorporate all necessary knowledge needed to produce the solution.

In general, the set of artifacts which could be acquired may include not only the set of fully/partially generated components (classes and interfaces, database scheme etc.), and tests, but also guidelines and recommendations, ordered set of prioritized and estimated tasks, probable risks, sketches of iteration plans and other process-oriented information [13].

How to construct the transformation engine, which is responsible for artifacts acquiring, how to build it rather flexible to be quickly adapted to new architectures and technologies. These are the questions the presented work is devoted to.

Main part. The experience of previous projects cannot be independent of the platform and technologies they use. Besides, as mentioned above, they relate to the selected architecture and technologies. Lessons learned bring a set of effective practices that can be generalized and standardized in the form of templates which can be used as a foundation by the transformation engine, a part of the system which is responsible for frame-based model interpretation. The process of training the engine comprises of creating of the transformation templates, creating or adaptation of the engine's logic to transform the model into artifacts using the templates, improving the descriptive power of the language which is used to describe the system (as a rule it considers adding new facets).

Let us look at the specific features of the frame-based language used to describe the system.

Frame is a data structure that is usually used to represent a single object or a class of related objects, or a general concept. Compared to the object-oriented approach, it is also the same as a class. There can be relations between frames as well as between classes in OOP: inheritance, aggregations, associations.

Slot is a part of a frame that describes an attribute or property of an object represented by the frame. It can also describe the relationship between its own frame and another frame. In addition to storing attribute values, slots can also have restrictions on allowed values. In comparison to the object-oriented approach, slots can be thought of as fields (structural slots), properties, and methods (functional slots).

Facet is a part of slot (for example, part of property description). It allows slots to define multiple attributes in addition to their value, such as data type, limits on allowed values. It is the main point where frame-based approach differs from the object-oriented approach. In modern languages, there exists a mechanism of attributes-annotations which is used to specify additional information about a property or a method of a class. But attributes-annotations are not related to classical OOP, they are related to metaprogramming, implementation of reflection mechanisms. But in case of frame-based approach, it is an organic and powerful mechanism used to represent the knowledge about entity or event in a compact form.

Simply put, a frame describes knowledge about a real-world object or event-scenario, a slot is a frame component, and a slot edge is a slot component. It is im-

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

portant to note that as the slot notion is not limited by the idea of attributes and methods representation, so a facet is not limited.

Thus, we can assume that due to the introduction of faces and the absence of restrictions on slots and facets, it is possible to obtain full and coherent description of the features of the system component becomes possible, and, accordingly, provide generation of software components with high-level of completeness.

The current variant of the frame-based language metamodel is shown in Fig. 1.

As we can see the language supports multiple inheritance, slots are divided into property slots, function slots (behavioral slots) and attribute slots (used to provide additional information, for example, the name package, assembly) without any restrictions on facets. The main difference from the classical frame-based model is the introduction of an additional unrestricted set of facets to the frame description (in the classic version, this task could be solved by the introduction of additional slot or slots).

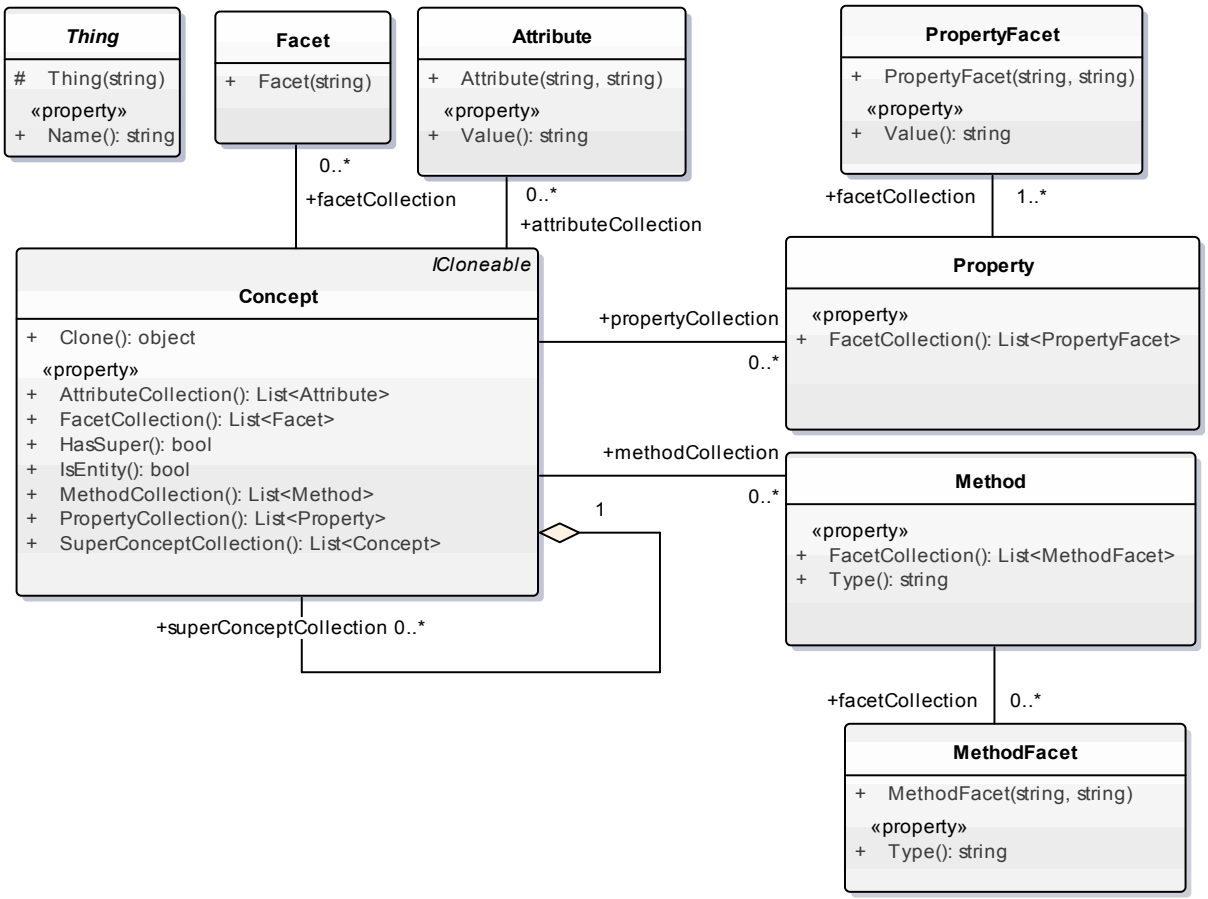

Figure 1 - Main concepts of XCore frame-based language

The Grammar in Antlr4 notation used to represent XCore script-language used is shown in Listing 1.

Listing 1

```
compileUnit : concept + EOF     ;
concept
: ID  (':' superconcept*)? ('facets' ':' facet*)? '{' prop-
erty* attribute* method*'}';
superconcept  : ID | (',' ID);
facet   : ID | (',' ID);
property : ID ':' ID (',' parameters)* ';';
parameters: ID '=' value;
attribute: '(' ID ')' ATTRIBUTEVALUE ';';
value: number | ID;
number: MINUS? DIGIT (DIGIT)*;
method: ID ':' ID (',' method_parameters)* ';';
method_parameters: ID ID;

ID : (IDSYMBOL | DIGIT) (IDSYMBOL)*;
ATTRIBUTEVALUE: STRING | (STRING'.'STRING)*;
IDSYMBOL: LETTER | UNDERSCORE | MINUS | SquareBrackets   ;
LETTER  : ('a'..'z') | ('A'..'Z');
STRING : [a-zA-Z]+;
MINUS: '-';
SquareBrackets: '[' | ']';
DIGIT: ('0' .. '9');
UNDERSCORE: '_';
WS: [ \r \n \t ] + -> channel (HIDDEN);
```

An example of entities description using the script-language is shown in Listing 2.

Listing 2

```
Identifiable facets: interface
{
  Id:int;
  (ns) Common.Domain.Contract;
  (asm) Common.Domain.Contract;
}


ThingBase : Identifiable facets: abstract
{
```


ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

```
    Id:int, nrmin = 1, pk = 1 ;

    (ns) Common.Domain;
    (asm) Common.Domain;
  }

  RegisterableEntityBase : ThingBase facets: abstract
  {
    IsActive:bool, nrmin = 1, defval = 1, needsProc = 1, in=1;
    CreateDateTime:DateTime, nrmin = 1, defval = now;
    FinishDateTime:DateTime, needsProc = 1;
    ModifyTime:DateTime, nrmin = 1, defval = now;

    (ns) Common.Domain;
    (asm) Common.Domain;
  }

  AddressableEntityBase : RegisterableEntityBase facets: ab-
stract
  {
    Address:string, nrmin = 1, needsProc = 1, validate=1,
              needsProc = 1, in=1;
    Phone:string, nrmin = 1, validate=1, needsProc = 1, in=1;
    Email:string, nrmin = 1, validate=1, needsProc = 1, in=1;

    (ns) Common.Domain;
    (asm) Common.Domain;
  }


  Delivery: RegisterableEntityBase facets: entity-dto
  {
    ProviderId:int, nrmin = 1,needsGetCollection=1, check=1,
                needsProc = 1;
    Description:string, validate=1, in=1, needsProc = 1,
                needsGet = 1;
    DeliverStatusId: int, nrmin = 1, defval = 1, needsProc = 1,
                check=1, in=1;
    (ns) LMS.Core.Domain;
    (asm) LMS.Core.Domain;
  }
```

To transform the description written in the XCore language into metalanguage objects we can use Antlr library (Fig. 1).

After that, the obtained model can be mapped to several specific representations which, in accordance with the MDA approach, can be associated with the PIM (platform independent model) level of system description. An example of a DSL that describes the application area is shown in Fig. 2. An important feature is that the class to which the Concept of the frame representation was mapped has a reference to the frame of which it was mapped.

The way of the transformation of the script into software components is shown in Fig. 3.

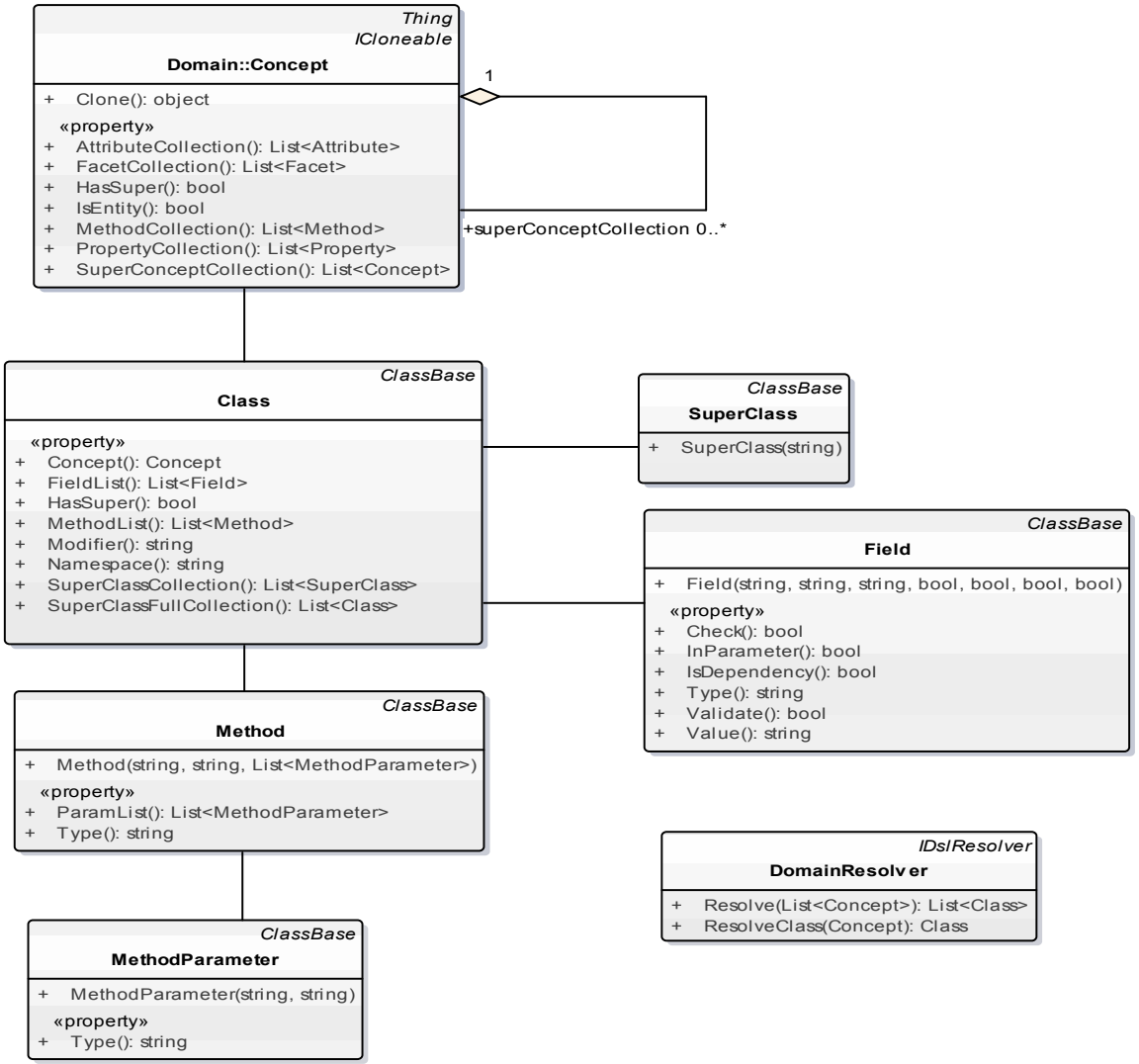

Figure 2 - The structure of DSL for application domain

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

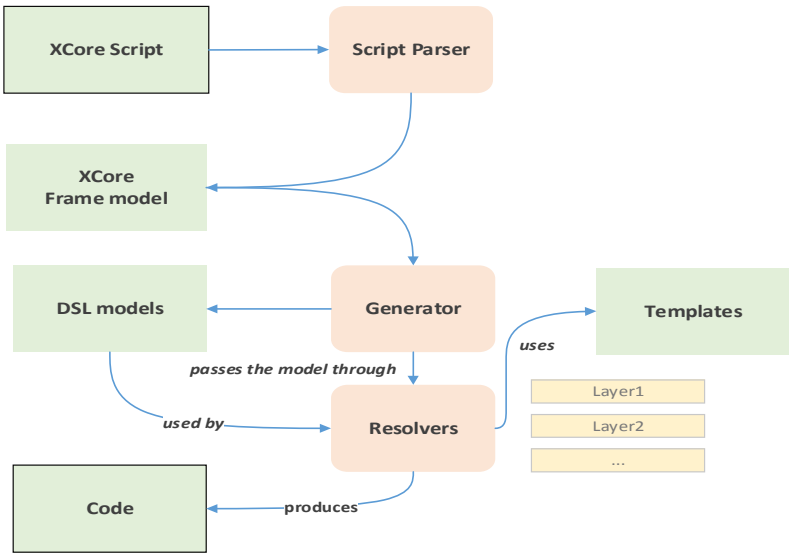

Figure 3 - The way of the transformation of the script into software components

As we can see, DSL models can be then transformed into the code using a set of resolvers which are responsible for DSL model interpretation. The result of interpretation can be associated with the PSM (platform specific model) level of system description according to the MDA approach. Since the components of the application are different from each other, separate Resolvers modules are used according to the parts. For example, C# and MySQL code use CSharpResolver and MySqlResolver respectively. Some of the resolvers are shown in Fig. 4.

The transformation of the model into code is based on templates. For example, C# generators have a template for individual class attributes and a template for the class as a whole. SQL generators have templates for defining tables and procedures, etc.

An example of the template responsible for the C# class representation is shown in Listing 3.

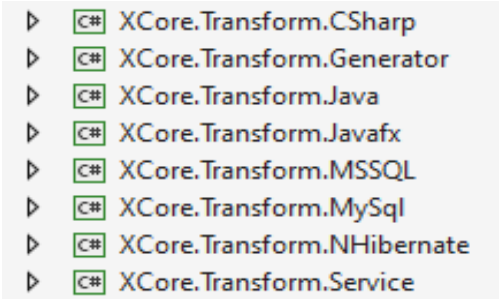

Figure 4 - The structure of generator unit

Listing 3

```
using Common.Domain;

namespace [namespace]
{ [using]
  /// <summary>
        /// The [ClassName]
        /// </summary>
        public[modifier] class [ClassName] [: Heirs]
        {
        #region Fields

        [Fields]
        #endregion

        public [ClassName](){}

        #region Methods

        [Methods]
        #endregion
        }
}
```

Experiment. In order to estimate the effectiveness of the provided approach, the following experiment was set up.

Firstly, the typical project should be selected, and architecture and technologies should be defined, governing by the typical system requirements. We made that in accordance with the approach described in [14]. Briefly, we assume that the development should be guided by SOLID principles and flexible service-oriented architecture. After the architecture was defined typical functions was selected and implemented without the help of the frame-based approach and code generation (but taking in mind its further application, because the structure of the solution and coding style should be fit to effective application of the generator).

After the realization of typical functions and refactoring aimed to achieve the desired level of standardization, we started to train the generator to acquire the results we get earlier but without manual coding. At this point it should be evaluated which templates, or even resolvers (generator components responsible for converting the model into code) should be created or modified.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

After the generator's training - which consists in creating the necessary changes, usually at the level of resolvers and templates, and obtaining positive results, we should try to build new system components for new functions similar to those that have been already implemented. If there is a need of the generated code modification, it is necessary to evaluate the time and effort spent on this modification. And after that, add the modifications to the templates to increase the quality of the generated code.

After the implementation of all functions, we should evaluate the effectiveness of the provided approach comparing the potential time and efforts expenses which would be required for solving the problem in the conventional way (without code generation) to the expenses with the use of the provided generator, considering the expenses spent on the generator's tuning etc.

Features of the test project. The project should be connected with the real application area, but it should be rather simple to avoid the complications connected with sophisticated business rules. Based on these principles, we have selected a typical coursework task that first year Master's Degree students are expected to realize for the discipline "Robustness of Information Technologies".

The set of functional requirements is shown in Fig. 5.

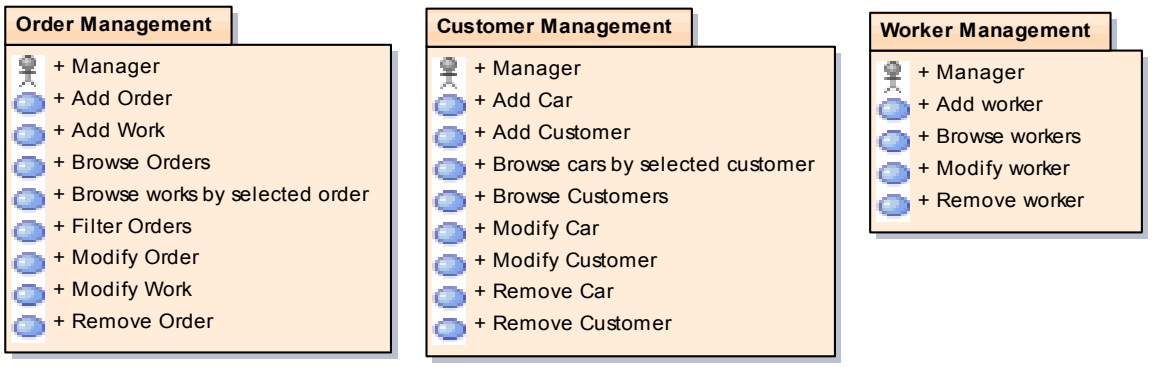

Figure 5 - Test project use cases

Analysis of the obtained results. Thus, in the case of implementation of the testing project with a new architecture (meaning the number of layers, and typical patterns and components used within these layers) and a new technology stack, the acquired results can be described as follows.

1. To achieve the maximum effectiveness we need to thoroughly analyze the project in order to discover the subset of the typical functions which would be considered as the foundation of the whole set of functions to be implemented. In the case of the testing project these are the functions related to orders and works (see

the interval from the start to “remove orders and works” use-cases at the “Function-time” chart shown in Fig. 6).

2. During the implementation of the functions, it is necessary to train the generator. In our opinion, an important point here is the synergy of the architecture and generator. That means that the generator relies on the prepared infrastructure consisting of basic and auxiliary classes that significantly simplify the construction of new code and, accordingly, its generation, because the generation relies on standardization of typical tasks’ solutions. In our case, the peak time spent on training the generator fell in the middle of the implementation of the work order functions (Fig. 7). Tuning of the generator was carried out mainly due to the addition of new transformation templates, and the construction and improvement of model transformers in the code. Thus, platform independent layers of the system (frame representation and domain-specific languages) were not modified.

3. During new functions implementation with the use of the generator (see Fig. 6 starting with “browse customers and cars”), the generator’s tuning is mostly connected with the modification of the existing templates and transformers. As we can see in Fig. 6, the effect achieved after the implementation of the first functions using the generator.

4. During the modification of already existing components, the use of the generator also proved to be effective (unfortunately, we cannot provide an adequate comparison of the time and effort required in the case of modification with and without the use of the generator). Updating the system is always associated with a greater effort on the part of the developer than with development, because it is necessary to focus more attention on the exclusion of damage to ready-made functions, which can be partially implemented thanks to the creation of tests (for the development of which, by the way, developers also need to spend a time and effort). Exclusion of syntactic and semantic errors, which usually occur during such modifications, with the help of a generator significantly reduces not only the time, but also the effort (mental stress) spent on solving problems.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

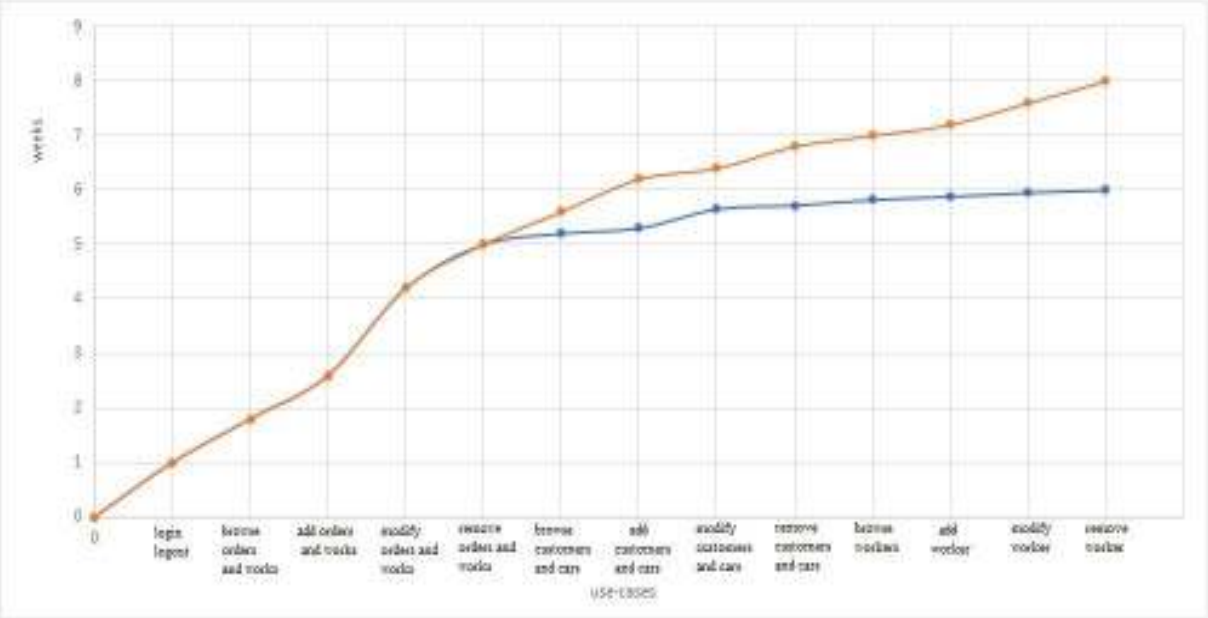

Figure 6 - “Function-time” chart

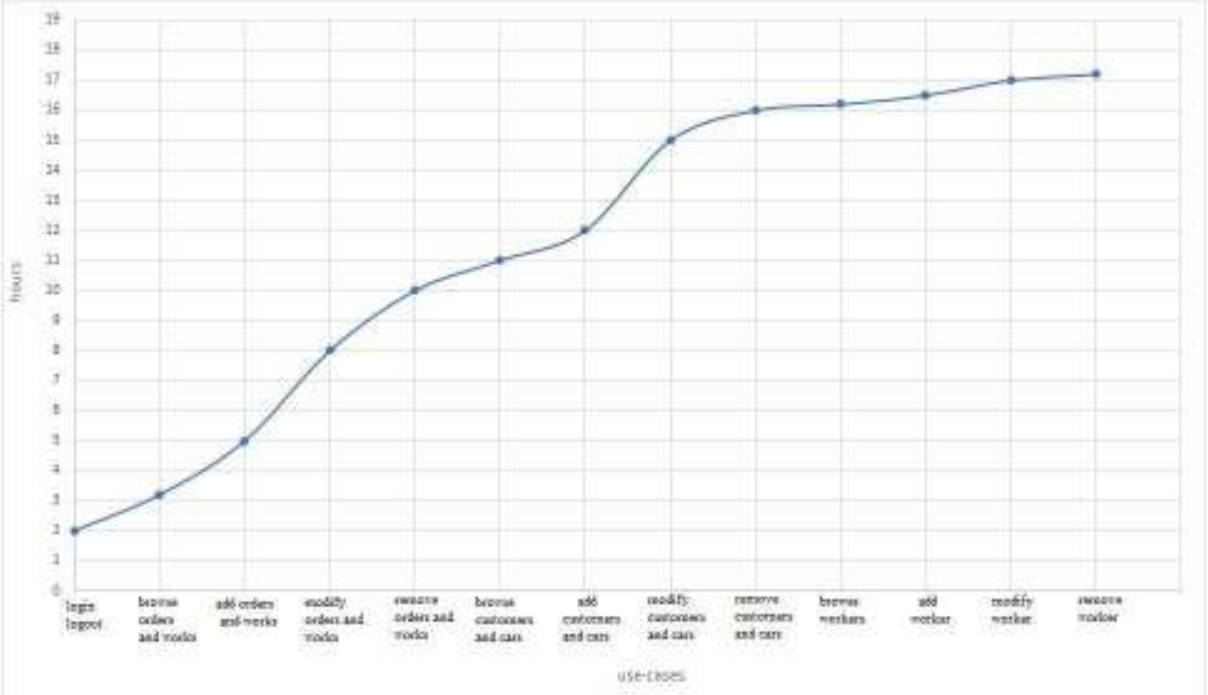

Figure 7 - “Generator tuning - time” chart

As for the estimating specific, it should be noted that the hypothetical time required to realize the project without the generator was calculated based on the minimum time required to develop functions without the use of the generator. Here we did not consider the time and effort spent on switching the focus of the developer's attention from one context to another (context means the connection with the specific of technologies used and features of application domain), errors which

appear in result of creating new code by copying existing code, time required for debugging, time required for formatting, commenting classes and functions, etc. We considered only the time required to create the necessary components (creating new files, writing code).

Summary. Thus, the use of the frame-based approach and the generator had a significant effect in the process of developing the information system. Due to the fact that the generator is built on a frame model, it makes it possible to fully describe and generate the necessary components and to obtain a more complete code.

In addition to this, the effectiveness of the generator has been proven not only during the system development, but also during its maintenance. As mentioned above, the generator is configured for a specific architecture, and in this regard, the architecture can help or hinder the use of the generator. Due to the fact that the system was built using a flexible architecture, the use of the generator was expedient.

REFERENCES

1. Eric Evans 2003 - Domain-Driven Design - Tackling Complexity in the Heart of Software. Addison-Wesley Professional, 2003. 560 p.

2. Vaughn Vernon. Implementing Domain-Driven Design. Addison-Wesley Professional; 1st edition. 2013. 656 p.

3. Robert C. Martin. Clean Architecture. A Craftsman’s Guide to Software Structure and Design. 2018. Pearson, 1st edition. 432 p.

4. Thomas Erl. Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall; 2016.

5. Martin C. Robert Agile Principles, Patterns, and Practices in C#. / C. Robert Martin, Micah Martin // Prentice Hall. – 2006. – 768 p.

6. Beck K. Test-Driven Development By Example. / K. Beck – Addison-Wesley. – 2002. — 240 p.

7. North Dan. Introducing BDD. [Електронний ресурс]. Режим доступа: http://dannorth.net/introducing-bdd/ 2006 — Загл. с экрана.

8. Mary Beth Chrissis. CMMI® for Development Guidelines for Process Integration and Product Improvement, Addison-Wesley Professional; 3 edition (March 20, 2011). – 688 p.

9. M. Kardoš, and M. Drozdová, “Analytical method of CIM to PIM transformation in model driven architecture (MDA)”, JIOS, vol. 34, no. 1, (2010), pp. 89-99

10. A.Kriouile, T. Gadi, Y. Balouki, CIM to PIM Transformation: A criteria Based Evaluation, International Journal Computer Technology & Applications, vol. 4, no. 4, pp. 616-625, 2013.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

11. Литвинов О.А., Грузін Д.Л., Гурєєв П.П. Особливості автоматизації розробки функцій в багатошарових інформаційних системах. // Системні технології. Регіональний міжвузівський збірник наукових праць. – Випуск 1(102). – Дніпро, 2016. – С.- 36-41
12. Litvinov A.A. On a frame-based language used for software modelling. // System technologies. – N.1(108). – Dnipro, 2017. – 55-63 p.
13. Litvinov A., S. Trotsenko, D. Ponomarenko, A. Sazonova. On improving the process of multi-layered information system development. II Всеукраїнська науково-практична конференція «ПЕРСПЕКТИВНІ НАПРЯМКИ СУЧАСНОЇ ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ ТА КОМП'ЮТЕРНИХ СИСТЕМ» MEICS-2017.
14. Litvinov A. A., Gerasimov V. V., Kovalchuk D. S., Krokhin V. V. On basic principles of minimum valuable information system development and preparation of professional software developers. System technologies. – N.1(120). – Dnipro, 2019. – 107-114 p.

Received 31.01.2023.
Accepted 08.02.2023.

Використання фреймового підходу для розробки інформаційних систем

Робота присвячена специфіці застосування фреймового підходу для розробки інформаційних систем. Даний підхід базується на описі прикладної області за допомогою скриптів на фреймовій мові, особливості якої детально описані в [12]. Важливою рисою такого опису є застосування концепції граней. Так, фрейм описує знання про об'єкт реального світу або сценарій події, слот є складовою частиною фрейму, а грань слота є компонентом слоту. При цьому у фреймовому підході як на слоти, так і на грані немає обмежень, які існують у класичному об'єктно-орієнтованому підході. Можна сказати, що завдяки саме введенню граней і відсутності обмежень на слоти стає можливим отримати повний і зв'язний опис усіх особливостей та рис об'єкту прикладної області і, відповідно, забезпечити генерацію програмних компонентів з високим рівнем повноти коду. Результати попередніх досліджень опубліковано в роботах [11-13]. Дана робота присвячена питанням впровадження та оцінці використання підходу для побудови інформаційних систем.

В роботі запропонована модель системи, що складається з трьох шарів: загальної моделі, у ролі якої вступає описання домену на фреймової мові; декількох домено-специфічних моделей в які трансформується фреймова модель; платформо-специфічні перетворювачі, що відповідають за перетворення домено-специфічних моделей у відповідні програмні компоненти. При цьому для трансформації домено-специфічних моделей у відповідні програмні компоненти використ-

товуються шаблони та програмні модулі, задачею яких є відображення моделі у код згідно шаблонам. Шаблони і компоненти, що відповідають за перетворення шаблонів у код, формуються на базі досвіду практичної розробки та залежать від архітектури та технологій. Можна казати, що процес навчання генератора програмних компонентів у даній системі складається в побудові та модифікації шаблонів та логіки трансформації.

У роботі розглядаються результати проведеного експерименту побудови навчального проекту (рівень магістерської курсової роботи) з застосуванням запропонованого підходу. Експеримент складався з двох фаз: вибір та реалізація типових функцій з навчанням генератору; реалізація решти функцій з застосуванням генератору.

В ході проведення експерименту було отримано наступні результати. Ефективність генератору залежить від структури проекту, стандартизації компонентів, принципів побудови (таких як SOLID). Тобто повинна існувати синергія між архітектурою проекту та генератором. З цього приводу дуже критичними є перші кроки, що складаються у виборі та реалізації типових функцій на яких буде навчатися генератор. Якщо функції для навчання вибрані вірно і далі не буде значних змін в архітектурі – час потрібний для налагодження генератору після його навчання буде мінімальний, реалізація нових функції та адаптація вже існуючої системи під зміну вимог буде проводитися дуже швидко. У нашому випадку при налаштуванні генератору на нову архітектуру за найгрубішими оцінками випередження розробки на два тижні, при 17 годинах часу витрачених на навчання та налаштування генератору.

Литвинов Олександр Анатолійович - кандидат технічних наук, доцент кафедри електронних обчислювальних машин Дніпровського національного університету ім. О. Гончара.

Литвинов Михайло Олександрович – студент-магістр кафедри електронних обчислювальних машин Дніпровського національного університету ім. О. Гончара.

Lytvynov Oleksandr Anatoliyovych - candidate of technical Sciences, associate professor of the department of electronic computing machines of the Oles Honchar Dnipro National University.

Lytvynov Mykhailo Oleksandrovych – second year Master's Degree student of the department of electronic computing machines of the Oles Honchar Dnipro National University.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)