National Research University Higher School of Economics

as a manuscript

Ulitin Boris

MODELS, METHODS AND SOFTWARE TOOLS FOR TRANSFORMATION OF THE DOMAIN-SPECIFIC LANGUAGES FOR INTERFACES OF GENERAL-PURPOSE SOFTWARE SYSTEMS

PhD Dissertation Summary

for the purpose of obtaining academic degree
Doctor of Philosophy in Computer Science

Nizhny Novgorod – 2021

The PhD dissertation was prepared at National Research University Higher School of Economics

Academic Supervisor:	Eduard Babkin, candidate of technical sciences, PhD, assistant professor, Nizhny Novgorod branch of National Research University Higher School of Economics

Content

DISSERTATION TOPIC

The relevance of the given topic. The focus of the work is the study of models and methods for the implementation of the evolution of external domain-specific languages (DSL) for modeling the structure of interfaces in order to increase the efficiency of support processes for the life cycle of general-purpose software systems. In this case, a structure means a set of objects involved in the interface and the relationships between them. Algorithmic DSLs used to describe solutions to a problem within the subject area are beyond the scope of this research. It is important to note that the DSL-based approaches used in this work are applicable for modeling both software interfaces between various components of a complex software system and human-machine interfaces [14] (i.e. objects and relationships displayed on the screen and used by the user). However, in this study we demonstrate the application and development of these approaches only to enhance the effectiveness of the development of human-machine interfaces as part of a general-purpose software systems.

From the point of view of computer science, DSL is a computer language (including programming or modeling) with limited expressive capabilities, focused on a certain subject area [17]. From a more general point of view (including the linguistic aspects of DSL as a language as a whole), DSL is an artificially created language that semantically and syntactically corresponds to a certain subject area [2].

Most often, the DSL is developed as part of some more general software system (the main system in relation to the DSL). In this sense, DSL is associated with a general-purpose language used to create the main system and can be internal or external in relation to it [17].

Internal DSLs are written in the language of the main application and are incorporated into this language [17]. From this point of view, internal DSLs represent a specific way of using the general-purpose language used to create the main application. As a consequence, for the development of internal DSL,

approaches typical for general-purpose languages are applicable, which analysis is beyond the scope of this study.

Unlike internal DSLs, external DSLs are written separately from the main system. After creation of such languages are embedded in the main application as a compiler or interpreter [17]. The main advantage of external DSLs is that they can best reflect domain concepts and customer requirements [6].

In this work, we consider only the external DSL used for general-purpose modeling software systems interfaces. This is effective and reasonable from the point of view that the human-machine interface of software systems has limited functionality (directly related to the functionality of the system in general), like the DSL, and in this sense can be considered as a specific type of DSL [26]. As a result, the introduction of elements of the development and modification of DSL in the development and modification of software systems can be effective and provide their greater flexibility in relation to the requirements of various categories of users [14].

Interest in the DSL appears both among researchers and among practitioners [26, 31]. First of all, this is due to the fact that DSLs provide a convenient, understandable and fairly simple from the user's point of view mechanism for managing the subject area for which they were created [17].

The work of many domestic scientists is devoted to the study of the problems of developing DSL, in particular I.S. Anureev (И.С. Ануреев) [1], B.N. Gaifullin (Б.Н. Гайфуллин) and V.E. Tumanov (В.Е. Туманов) [2], A.O. Sukhov (А.О. Сухов) [6], V.G. Fedorenkov (В.Г. Федоренков) and P.V. Balakshin (П.В. Балакшин) [7], etc. This topic is also reflected in the works of such foreign scientists as Pablo Gómez-Abajo, Esther Guerra, Juan de Lara [18], Aleksandar Popovic, Ivan Lukovic, Vladimir Dimitrieski, Verislav Djuki [36], Walter Cazzola, Edoardo Vacch, etc.

In addition, a large number of works are devoted to research in related areas, in particular: the definition and dynamics of semantics are studied in works A.P. Ershov (А.П. Ершов) [4], L.A. Kalinichenko (Л.А. Калиниченко) [5, 25],

Guizzardi [19, 20], Stoy [39], Strachey [33], etc., definition of syntactic aspects of DSL – in works by Evans [16], Laird [27], etc.

In most modern studies [17, 26, 34] the process of developing a DSL includes the following stages: making a decision (on the need to create a DSL, in other words, an analysis of its applicability), analysis of the subject area (for which the DSL is created), the design of the DSL (including the definition of the entire structure of the DSL and the selection of the most suitable type of DSL), the implementation of the DSL and the deployment of the DSL.

Also sometimes considered as a separate stage of the support of the DSL (including the possibility of the evolution of the DSL) [13]. Evolution can occur both under the influence of changes in the subject area itself ([13, 14]), and under the influence of internal factors, such as changes in the behavior of users of the system [27], their heterogeneity [35].

However, as the analysis of publications shows, the works do not highlight the mechanism of tracking changes in the subject area and their translation to the DSL model. Moreover, it is believed that by the time the DSL was developed, the domain model itself had already been created manually and transferred to the DSL model (as [18, 23]). It is fair to say that a number of researchers, including Peter Bell [10], Josh G.M. Mengerink, Alexander Serebrenik, Mark van den Brand [29], Ramon R.H.Schiffelers [30], Jonathan Sprinkle, Gabor Karsai [38], investigate the evolution of graphical domain models. But they do not consider the subsequent transfer of the changes made to the models and the rules that provide them to the DSL model [38]. On the contrary, these authors try to keep the DSL structure unchanged, which does not correspond to the assumption that the DSL model is identical to the domain model, which means that any change in the domain model should result in an equivalent change in the DSL model.

The issue of DSL evolution is even more relevant in the case of DSL for modeling human-machine interfaces of general-purpose software systems, since the life cycle of software systems presupposes the presence of a software system support stage, which implies its development (modification) for new user requirements [27].

To emphasize the importance of this requirement, the work uses the concept of a dynamic context, in the conditions of which the stage of supporting a software system takes place. The dynamic context in the work is understood as a set of changing models of the subject area and / or a set of user competencies. The presence of a dynamic context makes it necessary to adapt the interfaces of the software system to the new requirements of users, as well as in accordance with changes in the models of the subject area for which the software system was created [14].

As a consequence, changes in the system must be reflected in the DSL, which also exists in a dynamic context. However, in the life cycle of the DSL, the support (maintenance) stage is optional, and the existing DSL development tools do not support the functionality for organizing the full evolution of the DSL [26]. As a consequence, there may be problems of efficiency in using a software system based on DSL [13].

In this regard, the urgent task of proposing approaches, methods and tools for supporting the evolution of DSL for modeling human-machine interfaces of general-purpose software systems.

The goal of the study is to improve the efficiency of life cycle support processes for general-purpose software systems by developing new models and methods for the evolution of DSLs for interface modeling.

To assess the effectiveness, the criteria are used in accordance with GOST R ISO/IEC 25010-2015 (ГОСТ Р ИСО/МЭК 25010-2015) [3], including the time of modification of the DSL, the number of manually entered lines of code for modifying interfaces, etc.

To achieve the goal, it is necessary to implement the following ***research objectives***:

- analyze existing approaches to the development and transformation (evolution) of DSL for modeling human-machine interfaces of general-purpose software systems;
- to develop a formal model underlying the proposed approach to

transformation and evolution of DSL for modeling human-machine interfaces of general-purpose software systems;

- develop algorithms for transforming models (both the subject area and the DSL for modeling human-machine interfaces of general-purpose software systems);
- to conduct approbation and testing of the obtained results (models, algorithms and software prototypes) on DSL for modeling human-machine interfaces of general-purpose software systems in various subject areas with a dynamic context.

The object of the research is the process of supporting the life cycle of general-purpose software systems in terms of the evolution of human-machine interface models as an example of an external DSL in dynamic contexts.

The subject of the research is the model-oriented transformation processes of the external DSL for modeling the human-machine interfaces of general-purpose software systems in a dynamic context in the case of changes in the domain models or the competencies of DSL users.

KEY RESULTS

Research methods. In the dissertation work, the following formal mathematical methods are used: theory of functions, theory of graphs [37] and graph grammars [8], elements of predicate logic [12]. UML diagrams [19] and conceptual (semantic) models in the form of codified ontologies and object-relational models [8] are used as tools of modeling the domain.

In the practical part of the work compilation methods, declarative means for transformations on graphs [11], object-oriented and event-oriented programming techniques are used.

Scientific novelty:

- based on the analysis of existing approaches and methods for the development of DSL, a generalized model-oriented structure of DSL is presented, which is a unified representation of all levels of the DSL structure;

- various types (models) of the DSL evolution are formalized for modeling the human-machine interface for general-purpose software systems, which allows one to determine the coordinated evolution of the DSM and all levels of the DSL in accordance with the proposed projection approach;
- built and implemented a set of cross-model transformations (based on graph transformations) for organizing the horizontal and vertical evolution of the external DSL for modeling interfaces of general-purpose software systems;
- on the basis of the selected set of cross-model transformations, a new method (called the projection approach) is proposed for the development of an external DSL for modeling human-machine interfaces of general-purpose software systems. The proposed projection approach makes it possible to automate the process of developing DSL for modeling interfaces of general-purpose software systems and use the representation of the domain in the form of a domain semantic model (DSM) in the design and implementation of DSL, as well as organize the automated evolution of DSL;
- in accordance with the proposed method, algorithmization and software implementation of the procedure for transforming a human-machine interface was performed as an example of an external DSL for modeling human-machine interfaces of general-purpose software systems for two subject areas.

As a result of the analysis of the performed software implementation of the transformation procedure of the human-machine interface as an example of an external DSL, reusable results were revealed. On this basis, it is concluded that the proposed approach and models of the DSL evolution are applicable to the implementation of the DSL evolution in various subject areas and are the basis for the development of universal software tools to support the complete DSL life cycle (including the DSL evolution stage).

Key aspects to be defended:

- formalized model of DSL evolution based on cross-model, graph transformations for modeling interfaces of general-purpose software systems in dynamic contexts;

- a new method (called the projection approach) to the development of an external DSL for modeling interfaces of general-purpose software systems, which makes it possible to automate the process of developing DSLs for modeling interfaces of general-purpose software systems and use the results of domain analysis in the design, implementation and modification of DSLs;
- reusable software implementation of the human-machine interface transformation procedure as an example of an external DSL, used as part of two prototypes of software environments for different subject areas.

Personal contribution to the aspects to be defended includes the development and implementation of models, methods for the evolution of DSL for modeling interfaces within the proposed projection approach in order to improve the efficiency of support processes for the life cycle of general-purpose software systems. In addition, the proposed approach was tested in the process of implementing software environments based on DSL for various subject areas.

As a result of approbation, the scientific articles were published. The applicant has a Certificate of state registration of a computer program №2018616788 «Postgraduate admission office» (v.1.0).

Practical value of the results.

As a result of the study, a new engineering method (projection approach) was developed for the development of an external DSL for modeling interfaces of general-purpose software systems, which makes it possible to increase the efficiency and convenience of support processes for the evolution of DSLs for modeling human-machine interfaces of general-purpose software systems. In support of the proposed approach, algorithmization and software implementation of the procedure for transforming the human-machine interface was performed as an example of an external DSL.

Prototypes of software environments were implemented using the proposed projection approach based on DSL (with the possibility of evolution in dynamic contexts) for the subject areas "Software system of the University Admissions

Committee" and "Software system for allocating resources of the railway station" (Java language and Eclipse plugins were used for development, the total number of lines of code of both prototypes – 22483). The developed prototypes make it possible not only to execute scenarios on the DSL, but also to modify the structure of all DSL levels in real time without the need to re-create the DSL and the software information system as a whole.

The experience of operating the developed software prototypes has shown that their use minimizes the number of errors and conflicts during the modification (evolution) of the DSL and the software environment as a whole, since all changes to the DSL are made in an automated manner, without the need for manual modification. As a result, the total time spent on modifying the DSL and the software environment as a whole decreases.

The proposed approach to supporting the evolution of DSL for modeling interfaces of general-purpose software systems is applicable to the implementation of general-purpose software environments of various subject areas with a dynamic context of use, in particular, in areas with a high degree of adaptability, for example, in Smart systems and cyber-physical systems [26].

PUBLICATIONS AND APPROBATION OF RESEARCH

First-tier publications

1. Ulitin, B. Ontology-based reconfigurable DSL for planning technical services / B. Ulitin, E. Babkin // IFAC-PapersOnLine. – 2019. – v. 52, no. 13. – pp. 1138-1144. – main co-author.

Second-tier publications

2. Ulitin, B. Adapting Domain-Specific Interfaces Using Invariants Mechanisms / B. Ulitin, T. Babkina // Advanced Information Systems Engineering Workshops. CAiSE 2021. Lecture Notes in Business Information Processing, vol 423. – 2021. – pp. 81-92. – main co-author.
3. Ulitin, B. Knowledge life cycle management as a key aspect of digitalization / E. Babkin, T. Poletaeva, B. Ulitin // Communications in Computer and

Information Science. 11th International Joint Conference on Knowledge Discovery, Knowledge Engineering, and Knowledge Management, IC3K 2019, Vienna, Austria, September 17-19, 2019, Revised Selected Papers Issue 1297: Knowledge Discovery, Knowledge Engineering and Knowledge Management. – 2020. – pp. 429-452.

4. Ulitin, B. Automated formal verification of model transformations using the invariants mechanism / B. Ulitin, E. Babkin, T. Babkina, A. Vizgunov // Perspectives in Business Informatics Research. 18th International Conference, BIR 2019, Katowice, Poland, September 23-25, 2019 Proceedings. Lecture Notes in Business Information Processing. – 2019. – i. 365. – pp. 59-73. – main co-author.
5. Ulitin, B. Digitalization: A meeting point of knowledge management and enterprise engineering / E. Babkin, T. Poletaeva, B. Ulitin // IC3K 2019 - Proceedings of the 11th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management. – 2019. – v. 3. – pp. 22-36.
6. Ulitin, B. A Projection-Based Approach for Development of Domain-Specific Languages / B. Ulitin, E. Babkin, T. Babkina // Perspectives in Business Informatics Research. 17th International Conference, BIR 2018, Stockholm, Sweden, September 24-26, 2018 Proceedings. Lecture Notes in Business Information Processing. – 2018. – i. 330. – pp. 219-234. – main co-author.
7. Ulitin, B. Multi-agent simulation modeling of online Internet discussions / E. Babkin, T. Babkina, B. Ulitin // Business Informatics. – 2018. – № 2 (44) – pp. 17-29. (Улитин, Б.И. Моделирование динамики онлайн-дискуссий в сети Интернет с использованием многоагентных систем / Э.А. Бабкин, Т.С. Бабкина, Б.И. Улитин // Бизнес-информатика. – 2018. – № 2 (44) – с. 17-29.)
8. Ulitin, B. Ontology and DSL co-evolution using graph transformations methods / B. Ulitin, E. Babkin // Perspectives in Business Informatics Research. 16th International Conference, BIR 2017, Copenhagen, Denmark,

August 28-30, 2017 Proceedings. Lecture Notes in Business Information Processing. – 2017. – 2017. – i. 295. – pp. 233-247. – main co-author.

9. Ulitin, B. Combination of DSL and DCSP for decision support in dynamic contexts / B. Ulitin, E. Babkin, T. Babkina // Perspectives in Business Informatics Research. 15th International Conference, BIR 2016, Prague, Czech Republic, September 15-16, 2016 Proceedings. Lecture Notes in Business Information Processing. – 2016. – i. 261. – pp. 159-173. – main co-author.

Other publications

10. Ulitin, B. Providing Models of DSL Evolution Using Model-to-Model Transformations and Invariants Mechanisms / B. Ulitin, E. Babkin // Digital Transformation and New Challenges, Lecture Notes in Information Systems and Organisation. – 2020. – v. 40. – pp. 37-48. – main co-author.
11. Ulitin, B. An Object-Oriented Model for Smart Devices in Internet of Things / B. Ulitin, E. Babkin // Proceedings of the 22st Conference of Open Innovations Association FRUCT, Jyvaskyla, Finland. – 2018. – pp. 263-271. – main co-author.
12. Ulitin, B. Ontology-based DSL development using graph transformations methods / B. Ulitin, E. Babkin, T. Babkina // Journal of Systems Integration. – 2018. – v.9 (2). – pp. 37-51. – main co-author.

Certificates of state registration of a computer program and databases:

13. Certificate of state registration of a computer program №2018616788 «Postgraduate admission office» (v.1.0).

Reports at conferences and seminars

The results of the dissertation research were presented at the following international conferences and seminars:

1. 15th International Conference on Perspectives in Business Informatics Research, Prague, Czech Republic, September 15-16, 2016. Report: «Combination of DSL and DCSP for decision support in dynamic contexts»;

2. 16th International Conference on Perspectives in Business Informatics Research, Copenhagen, Denmark, August 28-30, 2017. Report: «Ontology and DSL co-evolution using graph transformations methods»;
3. 17th International Conference on Perspectives in Business Informatics Research, Stockholm, Sweden, September 24-26, 2018. Report: «A Projection-Based Approach for Development of Domain-Specific Languages»;
4. 22st Conference of Open Innovations Association FRUCT, Jyvaskyla, Finland, May 15-18, 2018. Report: «An Object-Oriented Model for Smart Devices in Internet of Things»;
5. 18th International Conference on Perspectives in Business Informatics Research, Katowice, Poland, September 23-25, 2019. Report: «Automated formal verification of model transformations using the invariants mechanism»;
6. 11th International Joint Conference on Knowledge Discovery, Knowledge Engineering, and Knowledge Management, Vienna, Austria, September 17-19, 2019. Report: «Digitalization: A meeting point of knowledge management and enterprise engineering»;
7. 15th International Workshop on Enterprise & Organizational Modeling and Simulation (EOMAS в рамках CAiSE 2019), Rome, Italy, June 3-4, 2019. Report: «Providing Models of DSL Evolution Using Model-to-Model Transformations and Invariants Mechanisms»;
8. 9th IFAC/IFIP/IFORS/IISE/INFORMS Conference Manufacturing Modelling, Management and Control (MIM), Berlin, Germany, August 28-30, 2019. Report: «Ontology-based reconfigurable DSL for planning technical services»;
9. 1st International Workshop on Model-driven Organizational and Business Agility (MOBA-2021) в рамках 33rd International Conference on Advanced Information Systems Engineering, June 28-29, 2021. Report: «Adapting Domain-Specific Interfaces Using Invariants Mechanisms».

CONTENTS

The dissertation work consists of an introduction, 4 chapters, a conclusion, a list of references, including 116 sources (including 17 domestic), a glossary and 7 appendices. The work is presented on 162 sheets of typewritten text, contains 45 figures, 4 tables.

The introduction presents the relevance of the work, the purpose and objectives of the dissertation research, scientific novelty, theoretical and practical significance of the research, and also provides a summary of the work.

Chapter 1 is devoted to the analysis of the existing classical methodology for the design and implementation of DSL for modeling human-machine interfaces of general-purpose software systems in the context of the life cycle of general-purpose software systems. For this, the chapter provides a definition of DSL and its place in general-purpose software systems. The description of the main stages of the classical life cycle of DSL and software systems is presented. The disadvantages of the existing methods of DSL life cycle management for modeling human-machine interfaces of general-purpose software systems are highlighted.

In particular, the life cycle of DSL and software systems do not fully correspond to each other, since in the case of software systems, the support and maintenance stage with the possibility of modification is mandatory, while in the case of DSL this life cycle stage is optional and is not supported by most existing DSL development platforms.

In addition, there are duplication of processes during the developing DSL according to the classical approach. For example, at the stage of analyzing the subject area, a semantic model of the subject area is formed, which in the future is not formally used when defining the semantics of the DSL, carried out manually. As a result, when making changes to the semantic model of the domain, it is also necessary to manually correct the semantic model of the DSL, and subsequently the syntax of the DSL.

These limitations of the classical approach make it impossible to automate the main stages of the DSL life cycle, and also do not allow organizing its evolution, since any change at all levels of the DSL structure is carried out manually and leads to the creation of a new DSL that is not consistent with the previous DSL. This disadvantage becomes critical in the case of DSL for general-purpose software systems at the stage of their maintenance and modification in accordance with user requirements.

To eliminate the shortcomings of the classical approach, it is proposed to use an approach based on a model-oriented representation of all levels of the DSL structure with the organization of the DSL development process through cross-model transformations between various models: a semantic domain model, a semantic DSL model, a DSL metamodel.

Chapter 2 is devoted to the analysis of existing methods and formalisms used to describe the structure of the DSL for modeling human-machine interfaces of general-purpose software systems. In particular, the formalizations of such artifacts as: the domain semantic model (DSM), the constituent structures of the DSL are considered: semantics, abstract (metamodel) and specific syntax.

In particular, it is highlighted that the structure of the DSL includes semantics, abstract and concrete syntax (Fig. 1). In this case, the semantics of the DSL is a projection of the DSM and can be obtained by cross-model transformations from the formalized representation of the DSM (Fig. 2) in the following form:

$$DSM = (\mathcal{H}_C, \mathcal{H}_R, O, R, A, M, D) \quad (1)$$

where

- $\mathcal{H}_C$ and $\mathcal{H}_R$ are sets of classes and relations schemas. Each schema is constituted by a set of attributes, the type of each attribute is a class. In both $\mathcal{H}_C$ and $\mathcal{H}_R$ are defined partial orders allowing the representation of concepts and relation taxonomies;
- O and R are sets of class and relation instances also called objects and tuples;

- A is a set of axioms represented by special rules expressing constraints about the represented knowledge;
- M is a set of reasoning modules that are logic programs constituted by a set of (disjunctive) rules that allows to reason about the represented and stored knowledge, so new knowledge not explicitly declared can be inferred;
- D is a set of descriptors (i.e. production rules in a two-dimensional object-oriented attribute grammar) enabling the recognition of class (concept) instances contained in O, so their annotation, extraction and storing is possible.

Considering that any model is a combination (E, R) of a set of entities and relationships between them, the DSL metamodel can be formalized in a model-oriented form in the following form.

- ***A set of entities*** of the meta-model $Set = \{set_i\},\ i \in \mathbb{N},\ i < \infty$, where every entity $set_i = \{SName_i, SICount_i, Attr_i, Opp_i, SRest_i\}$ is characterized by its name ($SName_i$, which is unique within the current model), available amount of exemplars of this entity ($SICount_i \in \mathbb{N},\ SICount_i \geq 0$), a set of attributes ($Attr_i = \{attr_{j_i}\},\ j_i \in \mathbb{N},\ j_i < \infty$), a set of operation on exemplars of this entity ($Opp_i = \{opp_{j_i}\},\ j_i \in \mathbb{N},\ j_i < \infty$) and a set of restrictions ($SRest_i = \{srest_{j_i}\},\ j_i \in \mathbb{N},\ j_i < \infty$).
- ***A set of relations*** between the entities $Rel = \{rel_i\},\ i \in \mathbb{N},\ i < \infty$, where every relation $rel_i = \{RName_i, RType_i, RMult_i, RRest_i\}$ is identified by its name ($RName_i$, which is unique within the current model), type ($RType_i \in \mathbb{N},\ RType_i \geq 0$), defining the nature of the relation, the multiplicity ($RMult_i \in \mathbb{N},\ RMult_i \geq 0$), which defines, how many exemplars of entities, participating in current relation, can be used, and a set of restrictions ($RRest_i = \{rrest_{j_i}\},\ j_i \in \mathbb{N},\ j_i < \infty$).

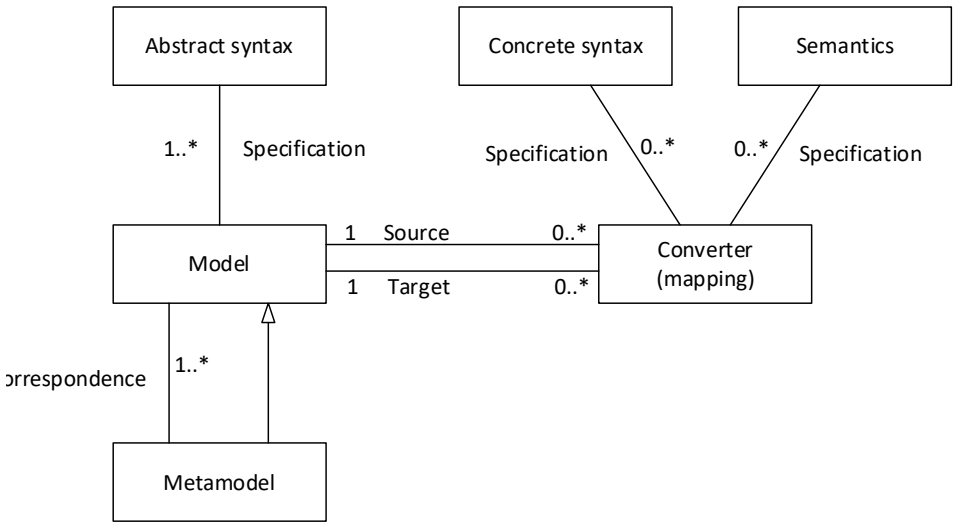

Fig. 1. Determination of the DSL structure in a model-oriented form [13]

Based on these ideas, the structure of the DSL metamodel can be defined as $(E, R, Rest, Opp)$, where the first two elements E и Rel epresent the object level, and the rest ones $Rest = \bigcup_{i=1}^{|E|} SRest_i \bigcup_{i=1}^{|E|} RRest_i$, Opp represent the functional aspects of working with DSL. Interpreting the set E as a set of domain entities, R as a set of relations between them and $Opp, Rest$ as a set of operations on entities and restrictions on them (respectively), it can be argued that the DSM structure (presented in Section 2.1) and the structure of the DSL metamodel correspond to each other.

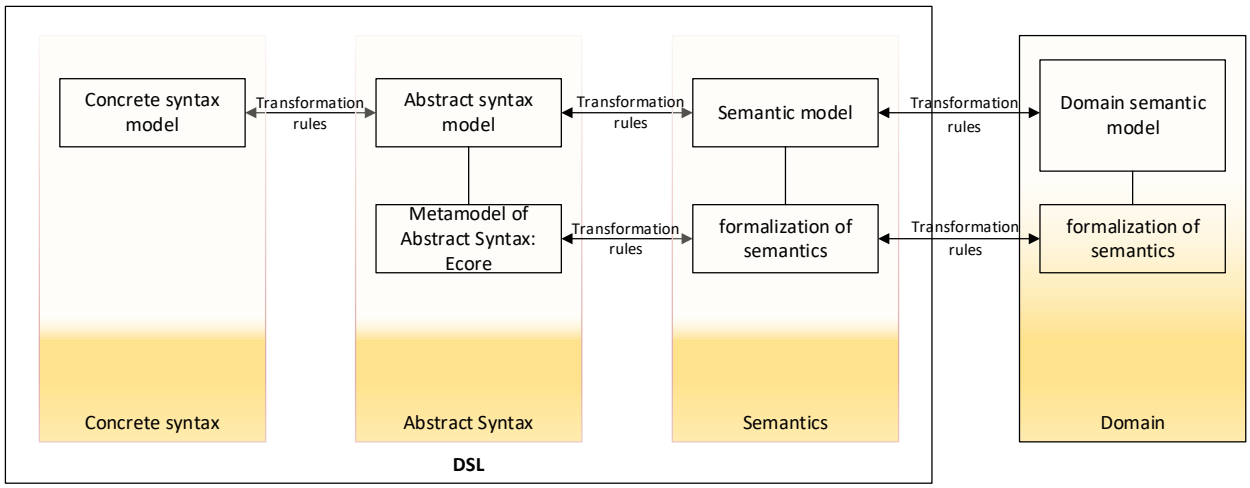

Fig. 2. Definition of the DSL structure using cross-model (M2M) transformation mechanisms

The syntactic level of DSL can be formalized as a triple $(O_{syntax}, R_{syntax}, Rule_{syntax})$, where $O_{syntax} \subseteq E$ and $R_{syntax} \subseteq R$ are subsets of sets of objects and relations between them of the DSL metamodel, and $Rule_{syntax}$ is a set of rules describing the mappings between the metamodel and a concrete DSL syntax.

This definition of a specific DSL syntax based on its metamodel does not depend on the type of specific DSL syntax (for example, textual or visual).

Under these conditions, we can say about a complete model-oriented representation of the structure of the DSL syntax. The structure allows not only to describe both levels of DSL syntax in a structured and unified manner, but also to optimize the process of development and further development of DSL by introducing several DSL syntactic dialects on one immutable metamodel. Moreover, control over the evolution of the DSL can be provided in a similar way both at the meta-level and at the level of a specific syntax, without the need to recreate the entire structure of the DSL every time changes are required. This is important because DSL can have several specific syntaxes, defined under a single metamodel.

For transitions between different models (DSM, models of different levels of DSL structure), cross-model (M2M) transformations are used. In the process of transformation preserves the essential characteristics (properties) of the models and their objects (for example, object property belonging to a particular class).

In contrast to the existing options for describing the structure of the LNP (in particular, textual [6, 21] and visual [18] DSLs), reflecting either the abstract syntax of the DSL, or the specific syntax of the DSL, without reflecting the semantic aspects, the presented generalized structure allows you to describe any DSL, regardless of their type, and eliminates the shortcomings of the classical approach to the development of DSL, described in Chapter 1. In particular, it provides the possibility of transferring the results of the stage of analysis of the subject area to the stage of direct development of the DSL by applying cross-model transformations

to the DSM. In the future, similar transformations can be used to construct all levels of the DSL syntax. This makes it possible to organize coordinated changes in the DSM and DSL structure and directly use the DSL representation in the form of DSM in the process of DSL development for modeling human-machine interfaces of general-purpose software systems, automating the processes of determining the DSL structure, thereby eliminating the need to re-create DSL in case of modification DSM in the process of evolution of the subject area.

Chapter 3 contains a detailed description of the proposed new projection approach to the development of DSLs for modeling human-machine interfaces of general-purpose software systems, based on the model-oriented representation of the DSL structure formulated in the previous chapter.

In this case, the semantics of the DSL is completely based on the semantic model of the domain and can be described using the corresponding DSM. From this point of view, the semantics of the DSL can be described as a triple $Sem = (O, R, A)$, where O – set of domain objects, R – relations between them, A – set of restrictions of the subject area (both on objects and on the relations between them).

The metamodel (abstract syntax) of the DSL can be represented as a set $MM = (E, Rel, Rest, Opp)$, where the first two elements E and Rel represent the object level, and the rest ones $Rest = \bigcup_{i=1}^{|E|} SRest_i \bigcup_{i=1}^{|E|} RRest_i$, Opp п represent the functional aspects of working with DSL. Unlike other existing ways of representing the DSL metamodel (for example, in accordance with the ideas by J. Luoma from [28]), in this case, at the metamodel level, we get a complete object-oriented representation, with the ability to transform it to the level of a specific syntax.

The concrete syntax can be formalized as $CS = (O_{syntax}, R_{syntax})$, where $O_{syntax} \subseteq E$ and $R_{syntax} \subseteq R$ are subsets of sets of objects and relations between them, the DSL metamodel.

As a consequence, transitions can be made between the DSM and the models describing the semantic level of the DSL, the level of abstract syntax and the level

of concrete syntax using the cross-model transformations described in Section 2.4. Thus expressing $Rule_{sem}$, $Rule_{MM}$, $Rule_{syntax}$ accordingly, the rules for the transition from the DSL to the DSL semantics model, from the DSL semantics model to the DSL abstract syntax (metamodel) model and from the DSL metamodel to the concrete syntax model, we can assert that the DSL structure in a model-oriented form is a set: $(Sem, Rule_{sem}, MM, Rule_{MM}, CS, Rule_{syntax})$.

In this case, the result of applying transformation rules to the DSM to obtain a semantic DSL model is called a projection. The DSL metamodel obtained from each individual projection by means of the corresponding transformation rules constitutes the syntax of the language. Finally, models of a specific syntax obtained from the metamodel by applying the appropriate transformation rules are different dialects of DSL, since they are based on a single metamodel and, as a consequence, contain different representations for the same objects and operations on them.

Based on this, the following new hierarchy of the DSL development process is proposed in accordance with the projection approach (Fig. 3).

In our case, this hierarchy is divided into four levels in accordance with the stages of development of the DSL. Each lower level is based on top level model artifacts [40].

Unified meta-metamodel (kernel meta-meta-model, KM3 [22]) defines common foundations for all metamodels and lower-level models.

The structure of the semantic hierarchy determines the appropriate process for creating external DSLs for modeling human-machine interfaces of general-purpose software systems. It begins with the definition of DSM containing all the key objects of the target domain and the relationship between them. When the DSM is created, we can build a semantic model of the DSL using the semantic projection operation. Any semantic projection performs a certain M2M-transformation of the DSM into some of its fragments. Thus, the semantic projection completely determines the semantic model of a particular dialect of DSL. In this case, the semantic model

becomes an object-time structure, since it should be adapted in accordance with changes in the DSM over time, thereby defining a new filling of the DSL object.

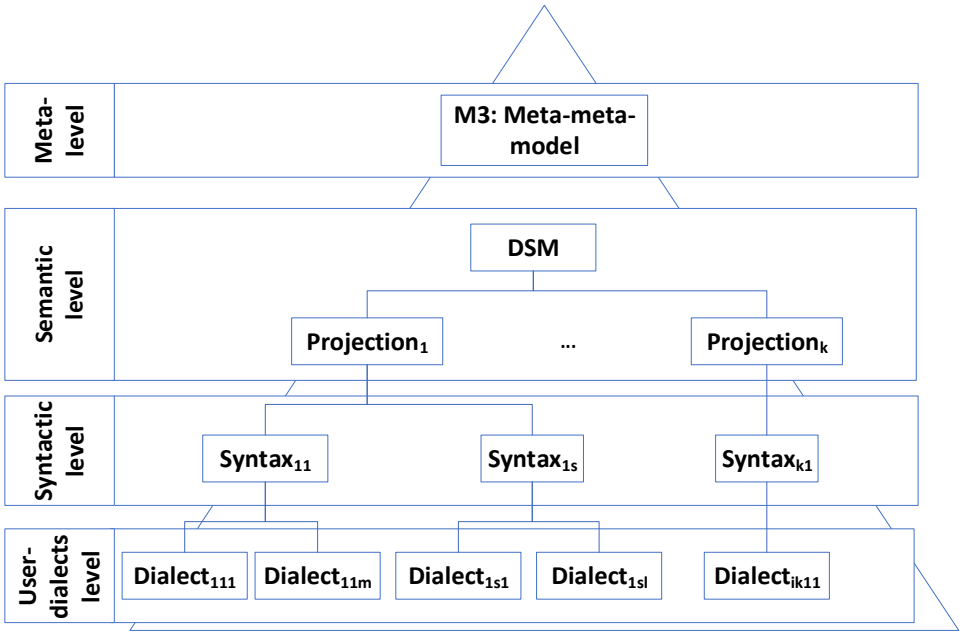

Fig. 3. Semantic hierarchy of the DSL development process in the projection approach

After the result of the semantic projection is successfully obtained (thereby, the semantic model of the DSL is determined), the syntactic level of the DSL can be obtained by cross-model (M2M) transformation of the result of the corresponding semantic projection. The syntactic models of DSL obtained in this case are independent of each other and are determined by end users in accordance with the adaptation of the semantic projection to their own tasks.

Finally, the syntax models created are used by the end users of the DSL, who define a set of DSL dialects within one particular syntax model.

Thus, in accordance with the proposed approach, the DSL is created taking into account the knowledge of the subject area and the requirements of users. The first fact makes it possible to organize coordinated changes in the DSM and the semantic model of the DSL, and the second guarantees the ability to make changes to the syntax of the DSL without the need to redefine the semantic level of the DSL.

The proposed approach differs significantly from the classical approach to constructing a DSL (Fig. 4), in which the transition between different levels of the DSL structure is implemented manually, and the results of the analysis of the subject area, recorded in the form of an DSM, are used only informally. In this case, changes at various levels of the DSL structure cannot be implemented independently of each other, since changes in the DSL syntax require the initial implementation of the corresponding changes at the level of the DSL semantics and its metamodel. As a consequence, each change in the target subject area leads to the need to manually redefine all levels of the DSL structure. A similar process is repeated when changes to the DSL are caused by end users. As a result, at the output we get a lot of incompatible DSL dialects that cannot be compared with each other due to differences at all levels of the DSL structure.

In comparison with traditional approaches, the proposed projection approach to the development of DSL for modeling human-machine interfaces of general-purpose software systems (Fig. 4, below) is organized in strict accordance with the target subject area and life cycle of software systems.

In this case, the results of the stages of the life cycle are not only recorded as artifacts, but are also used in the implementation of subsequent stages of the life cycle. So, for example, the results of the stage of analyzing the subject area are recorded in the form of DSM (represented, for example, in the form of an ontology), then this model is transformed with the help of cross-model transformations into a DSL semantic model, which in this case may use DSM partly (its individual components). Thus, it is possible to build several semantic models for different DSLs on one semantic model of the domain. Then, using cross-model transformations on the basis of the semantic model of the DSL, a metamodel (abstract syntax) of the DSL is built, on the basis of which the specific syntax of the language is further defined.

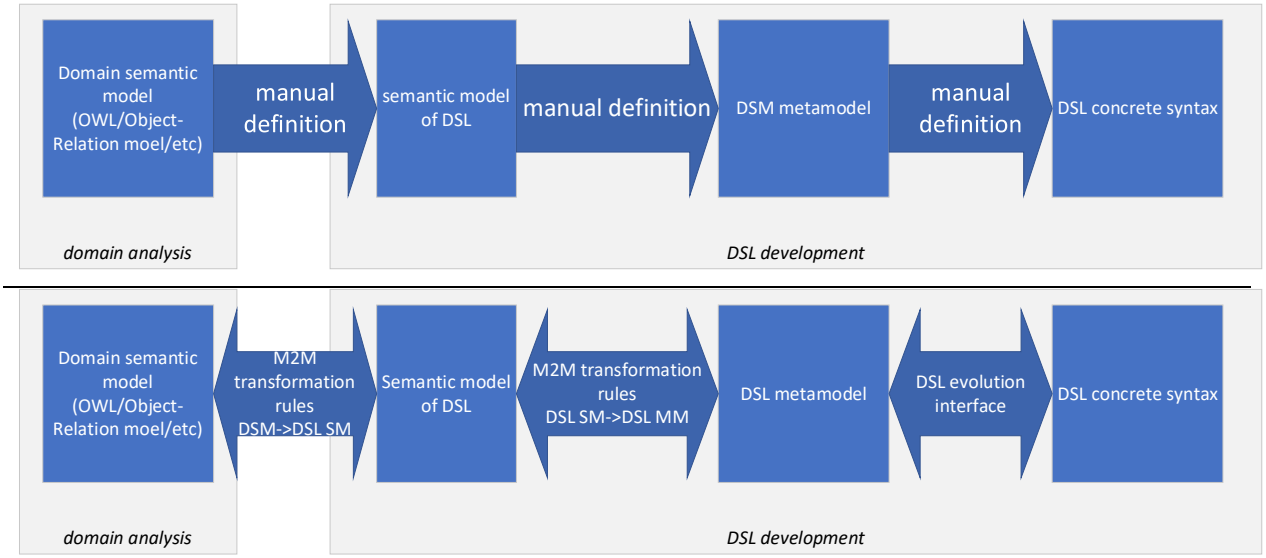

Fig. 4. Differences between the traditional (above) and projection (below) approaches to the development of DSL

At the same time, since cross-model transformations are bidirectional, changes in one of the models can be automatically transferred to other models, thereby maintaining the consistency of all models with each other. The only stage at which user participation is required is the stage of determining the concrete syntax of the language. However, even this stage is partially automated, since the user uses the objects of the DSL metamodel, defining only a set of commands for working with them through the human-machine interface of the DSL evolution built into the system.

As a result, we can define several DSL syntactic dialects within one specific DSL semantic model, which will be consistent and between which users can make mutual transitions without redefining the DSL semantic models.

In the proposed projection approach, the transition between the various components of the DSL structure occurs through the application of a set of rules for cross-model transformations based on the formalizations of the horizontal and vertical DSL evolution formulated in the chapter.

Vertical evolution means the change in level of conceptualization (perspective) of the model. In this case of evolution new entities are added into the model (with(out) changes in the set of relations). It means, that vertical evolution

can be formalized by the following manner: (E^1, R^1) is a result of vertical evolution of the model (E, R) if $|E| \neq |E^1|$ and $|R| \neq |R^1|$. In terms of DSLs it means, that the object level will be changed: some objects will be introduced, some will be removed from the language – alongside with the corresponding changes on the functional level.

Horizontal evolution means preserving the level of conceptualization, but changing the sets of attributes for some entities, or changing the set of relations. It means, that vertical evolution can be formalized as follows: (E^1, R^1) is a result of horizontal evolution of the model (E, R) if $|E| = |E^1|$ and $\exists\ e_i \in E, e_i^1 \in E^1$: $Attr_i \cap Attr_i^1 = Attr_i \wedge |Attr_i| \neq |Attr_i^1|$ and $\exists\ r_i \in R,\ r_j \in R^1$: $r_i \notin R^1 \vee\ r_j \notin R$. In terms of DSL it means, that the object level will be changed only in terms of objects attributes with corresponding changes on the functional level (so-called constructors of objects) or in terms of connections between them (in this case changes on the functional level depend on the nature of changed connections).

The chapter also shows that to implement any kind of evolution, the following set of rules is sufficient:

- creating (adding) an entity;
- deleting an entity;
- creating (adding) relationships between entities;
- deleting a relationship between entities;
- creating (adding) an attribute;
- deleting an attribute.

It is important to note that this set of rules (transformations) is universal and can be applied to implement any kind of evolution of any model. Consequently, it is applicable to the implementation of the coordinated evolution of the DSM and models of all DSL levels.

Then, it can be argued that evolution operations can be formalized as follows: $F_v = f(Rule_{CE}, Rule_{DE}, Rule_{CR}, Rule_{DR})$ and $F_h = g(Rule_{CA}, Rule_{DA}, Rule_{CR}, Rule_{DR})$, where f and g – some functions (sequences of

transformations) formalizing cross-model transformations, $Rule_{CE}$ and $Rule_{DE}$ – rules (cross-model transformations) for adding and removing model entities, respectively, $Rule_{CR}$ and $Rule_{DR}$ – rules (cross-model transformations) for adding and removing relationships between model entities, respectively, $Rule_{CA}$ and $Rule_{DA}$ – rules (cross-model transformations) for adding and removing attributes of model entities, respectively.

All the above rules$Rule_{CE}, Rule_{DE}, Rule_{CR}, Rule_{DR}, Rule_{CA}, Rule_{DA}$ do not depend on what entities / relations / attributes are used in them. These rules have a double nature, since they apply to a set of some model objects and also require an additional parameter specifying which of the objects in the set should be transformed.

Taking into account, that F_v and F_h are a superposition of the above rules of cross-model transformations, it can also be argued that in this case the evolution procedures are the result of the sequential application of various rules of cross-model transformations and, therefore, when they are applied, the essential characteristics of the models indicated above are also preserved.

Thus, a set of cross-model transformations of models is obtained, which does not depend on the form in which the DSM and models of all levels of the DSL are presented. As a consequence, this set of rules is reusable and can be adapted to organize the evolution of any type of DSL.

Based on this set of rules for cross-model transformations, the chapter presents the details of algorithms and software implementation of the procedure for the evolution of the human-machine interface of general-purpose software systems. In this case, the algorithm of the DSL evolution procedures can be defined as follows:

1. The main objects of the meta-level of the initial (source) model (classes, etc.) are determined.
2. The main objects of the meta-level of the target model are determined.

3. A correspondence is established between the objects of the meta-level of the source and target models by defining the ATL transformation rules between them.
4. The specified transformation rules are applied by means of the transformation software environment (in our case, the Eclipse Modeling Project [15] is used).

The developed software algorithms are used to implement software systems in two subject areas, described in the next chapter.

Chapter 4 contains a description and analysis of the software implementation of the procedure for the evolution of the human-machine interface of general-purpose software systems as an example of an external POL in two subject areas: "Software system of the University Admissions Committee" and "Software system for allocating resources of the railway station". The definition of the rules of cross-model transformations in the ATL language [9] for the implementation of the horizontal and vertical evolution of the LEP for modeling the human-machine interfaces of general-purpose software systems is presented. Tools (modules) have been developed to support the evolution of DSLs for modeling human-machine interfaces of general-purpose software systems.

In particular, it is shown that each of the implemented software environments contains a separate module for organizing any types of DSL evolution, both visual (in the case of the Admissions Committee) and text (in the case of a railway station). At the same time, changes to the DSL are made through the interface and does not require manual changes to the DSL structure (Fig. 5, Fig. 6). The changes made are applied in real time, eliminating the need for manual editing of the DSL and the software system in general.

In the case of a railway station, two variants of DSL modification scenarios are presented: for vertical and horizontal evolution. In the case of horizontal evolution, the user defines a new command in the DSL. Subsequently, to implement vertical evolution, a new entity is introduced into the DSL, which is also used to the created team in real time.

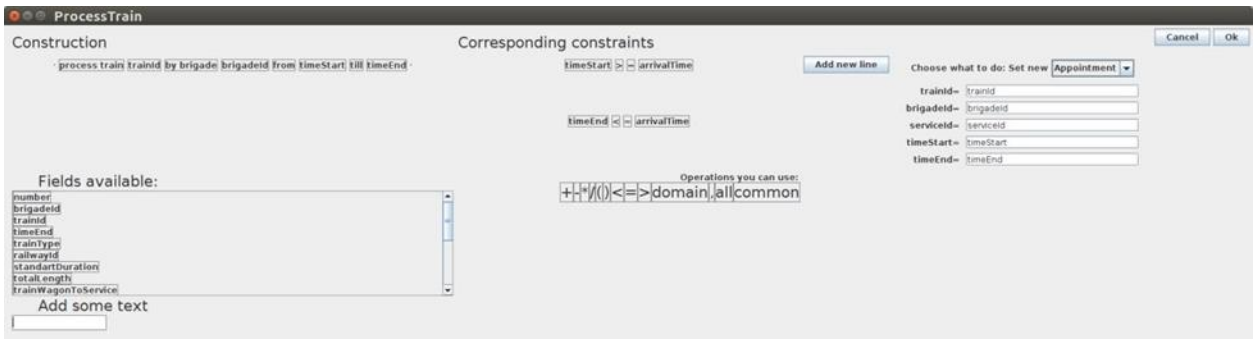

Fig. 5. DSL command creation (editing) interface

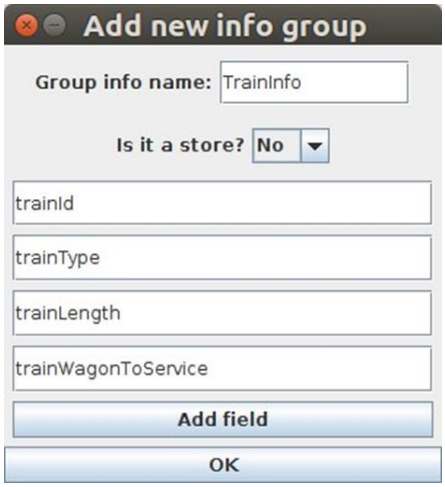

Fig. 6. Interface for adding a new entity to the DSL metamodel

In this case, when creating a new entity without the considered approach, the user would have to change: the grammar of the DSL metamodel (describing both the entity as a whole and all its attributes together with domains), fragments of the parser for recognizing the corresponding commands associated with the created entity, fragments of constraints associated with the created entity – for the solver. Finally, there are syntactic constructions at the level of a concrete DSL syntax. In total, these changes (in the above example) would amount to about 2000 lines of source code in all components of the system. In the proposed system, thanks to the implemented system of rules for the implementation of the coordinated evolution of the DSM and all levels of the DSL, these changes are made automatically and the user does not need to change the source code of various components of the system.

Moreover, the user may generally not have deep knowledge in the field of grammars of the language and programming, since all changes are formulated in a

form that is understandable to him (consistent with the subject area) through the system interface and are applied (translated) to all levels of the DSL automatically.

As a consequence, significantly reduced the probability of error when making changes to the application source code, which is also an advantage of the proposed system and approach in general.

The chapter also provides the results of evaluating the characteristics of existing and proposed software systems in accordance with GOST R ISO / IEC 25010-2015 (ГОСТ Р ИСО/МЭК 25010-2015) according to the following criteria: functionality, reliability, usability, efficiency, portability, maintainability and portability.

In terms of functionality, the following compliance with the quality criterion can be distinguished:

- *Suitability* – demonstrated on various cases implemented using both software environments; the approbation of the presented solutions conducted;
- *Correctness* - the results obtained in the course of the work of software environments were checked and evaluated by subject matter experts. In addition, the verification of the correct operation of the software environments was checked on archived data, after which the results obtained were evaluated with an archived reference value. As a result of such experiments in the case of the Admissions Committee (the comparison was made for 183 indicators) – the results coincided in 100% of cases. In the case of a railway station, the distribution of trains (the comparison took place during the reproduction of 54 experiments, in each case, on average, 60 trains arriving at the station were considered) – a deviation from the reference value was found in 3 experiments. However, this deviation is caused by the fact that the proposed system was unable to find a solution during the time allotted for the search for a solution (this time coincides with the time during which the solution was found by the existing OSA (АРМ «ОСА») system). However, in general, the solution found by the program (found after the allotted time expired) coincided with the reference one.

- *Ability to interact* – in this case, we consider only the ability to interact with the user, which the built software environments fully provide, since the interaction is carried out through graphical user interfaces.
- *Consistency – the built software environments were developed in full compliance with the current Acceptance Rules (in the case of the Admissions Committee) and the Standards and Guidelines for the transportation and management of freight complexes of railway stations (in the case of a railway station), which allows to approve the consistency of the built solutions.* In addition, the implemented evolution modules in both software environments will also allow in the future to ensure the change of software environments in accordance with the changes in these documents, while maintaining compliance with the requirements for consistency.
- *Security* – issues of ensuring the security of the built software environments are not included in this study, therefore in this case we can only assert the security of the built software environments, provided by standard means: separation of data and access interfaces to them, protection of the data used at the server level with the database and etc. Thus, the security of the built software environments is satisfactory.

As a result, from the point of view of functionality, the built software solutions meet most of the presented criteria: suitability, correctness, interoperability, consistency. As a result, we can conclude that the methods used in the development of the presented software environments allow building software solutions for which it is important to ensure a high degree of consistency with existing standards and domain constraints that ensure interaction with the end user. However, they are not applicable when it comes to the development of the so-called. critical systems for which the safety indicator is critical.

From the point of view of reliability, the following compliance with the quality criterion can be distinguished:

- *Stability* – both of the considered software environments provide for the possibility of checking user scripts, both at the stage of their input (checking

the used constructs) and at the stage of execution (finding conflicts). However, the module for organizing the evolution of the DSL of both environments requires manual control by the user (in particular, for the correspondence of the subject area in the case of adding new entities). In addition, in the case of a railway station, the system has restrictions on the number of users simultaneously executing scripts - no more than 15 users. In the case when the number of concurrent users exceeded this limit, the system experienced freezes, and some of the scripts did not have time to be processed. Thus, it is possible to assert a satisfactory level of compliance of software environments with this quality criterion.

- *Resilience to error* – both built software environments have built-in mechanisms for checking user scripts, both at the stage of their input (checking the used constructs) and at the stage of execution (finding conflicts). As a consequence, it can be argued that the constructed systems are resistant to errors.

Thus, the constructed systems provide sufficient compliance with the reliability criteria, being robust to errors. However, this is not enough for using the presented approaches in the case of high-load and high-performance systems, for which it is critical to organize the simultaneous access of a large number of users (more than 50) and to process a large number of requests in real time.

Similar conclusions can be drawn by analyzing the compliance of the presented software environments with performance criteria:

- *The nature of change over time and the nature of resource change* – from the point of view of the fact that any modification of the structures of the language and its metamodel leads to the complication of processing these structures, the proposed methods provide only a satisfactory compliance with this quality criterion. However, as shown above, this becomes critical only when more than 15 users work with the system and some of the scripts remain unprocessed in the allotted time. In the case when the number of users is less, even after modifying the language constructions and adding new entities, the

processing time of operations changed by 1.3ms on average (in the case of a railway station).

From the point of view of practicality, the developed software environments are of high quality, since they meet the following criteria:

- *Comprehensibility and Learnability* – the developed software environments are completely based on the terminology and semantic model of the domain, as a result, are understandable to end users. It is important to note that in the case of using both software environments, it was not necessary to carry out separate training activities, which confirms the high degree of compliance with the criteria for comprehensibility and learnability.
- *Ease of use* – scenarios in the case of a DSL railway station are formulated using commands used in the subject area, thereby ensuring the ease of creating these scenarios. In the case of the Admissions Committee, all interaction takes place through graphical interfaces, which also greatly simplifies the ability to generate various reports on the admission campaign and enter information on applicants.

From the point of view of maintainability, the presented software environments provide the possibility of modification in real time and meet the following criteria:

- *Variability and Testability* – both environments contain separate modules for organizing the evolution of the DSL, which provide the ability to modify its designs through the interface in real time, without the need for manual changes. So, in the case of modifying the main structure of the Admissions Committee – Applicant module (adding an additional attribute), the total number of lines of code added was 479. Without the proposed approach, all these lines would have to be entered manually; in the case of the proposed system, all of them are generated automatically without the intervention of the end user. The changes made are immediately available to the end user, so they can be immediately checked for correctness, which indicates the ease of testing the changes made.

Thus, the presented approaches can be used for the development of systems (and DSL for such systems), for which it is important to provide support for the possibility of simple and quick changes, without the need for manual changes at the source code level.

Finally, from the point of view of mobility and portability, the constructed systems satisfy the following criteria:

- *Adaptability and Compliance* – similar to the compliance of mutability, the built software environments provide a high degree of adaptability of the built solutions in real time, as well as ensure compliance with various Provisions and constraints of the subject area. In this regard, it is also important to note the cross-platform nature (and, as a consequence, portability) of the built solutions, since the Java language was used in their development.

Thus, the built software environments provide a high degree of adaptability. As a result, the approaches used in their development can be used to build other systems, for which it is necessary to ensure the ease of making changes (adaptability) and to maintain compliance with the existing restrictions (and provisions) of the subject area.

Summing up the presented analysis of the quality of the constructed software environments, it can be argued that the approaches, methods and algorithms used in the development can also be applied in the case of constructing general-purpose adaptive systems. However, their use is not recommended for highly loaded, high-performance, critical systems, where it is important to ensure a high level of security and system performance in the case of a large number of simultaneous users and requests.

The conclusion of the work contains a list of the main results of the study, an assessment of the level of achievement of the set goal, as well as proposals for the further development and practical application of the results obtained in various subject areas.

CONCLUSION

Thus, within the framework of this work, the issues of the development and evolution of domain-specific interface modeling languages for general-purpose software systems in dynamic contexts were considered. The main goal of the study is to improve the efficiency of lifecycle support processes for general-purpose software systems by developing new models and methods for the evolution of LSP for interface modeling. The projection approach proposed in this work allows you to change the structure of all DSL levels in real time without the need to re-create the application as a whole. This is the main difference between our own solution and existing approaches [10, 13, 14, 24], the main idea of which is to refuse to implement a full-fledged evolution of the DSL in favor of re-creating the structure of the language if it is necessary to modify it.

The scientific novelty of the study is as follows:

- based on the analysis of existing approaches and methods to the development of DSL, a generalized model-oriented structure of DSL is presented (Section 2.3), which is a unified representation of all levels of the DSL structure;
- various types (models) of DSL evolution have been formalized for modeling the human-machine interface for general-purpose software systems (Section 3.2), which makes it possible to determine the coordinated evolution of the DSM and all DSL levels in accordance with the proposed projection approach;
- proposed and implemented a set of cross-model transformations (based on graph transformations - presented in Section 3.3) for organizing the horizontal and vertical evolution of the external DSL for modeling interfaces of general-purpose software systems;
- on the basis of the selected system of cross-model transformations, a new method (called the projection approach) is proposed for the development of an external DSL (Section 3.1), which makes it possible to automate the process of developing a POL for modeling interfaces of general-purpose

software systems and use the results of domain analysis in the design and implementation of DSL, as well as organize the automated evolution of DSL;

- in accordance with the proposed method, algorithmization (Sections 3.2 and 3.3) and software implementation (Sections 4.2 and 4.3) of the procedure for transforming the human-machine interface were performed as an example of an external DSL for two subject areas.

As part of the practical part of the work, implemented prototypes of information environments for two subject areas: "Software system of the University Admissions Committee" and "Software system for allocating resources of the railway station". The constructed prototypes are based on the proposed projection approach, providing the user not only with functionality for solving problems of the subject area, but also for organizing the coordinated evolution of all levels of DSL and DSM.

Unlike existing domain-specific solutions for various domains [18, 26, 34], the presented software prototypes contain separate modules for organizing the evolution of the DSL without the need to make manual changes to various levels of the DSL structure. At the same time, the created DSL dialects remain compatible with the previously defined ones, since changes are made not only at the level of the specific DSL syntax, but are transferred, using the rules of cross-model transformations proposed by the authors, to the levels of the abstract syntax of DSL and DSM. This ensures complete consistency of evolution at all levels of DSL and DSM.

To achieve these results, such formal mathematical methods were used as elements of function theory (Section 3.3) – to formalize the relationship between minimum structural unit (MSU) and minimum structural component (MSC) and implement cross-model transformations, elements of graph theory (Section 2.4, Sections 3.2 and 3.3) - to formalize transformations between models DSM and DSL.

The projection approach to the development of DSL proposed in this work differs significantly from the classical approach [35, 37], in which the DSM is

considered only as a basis for the development of DSLs and is not used in any way as an artifact in the process of DSL implementation. In the case of the proposed approach, the development of the DSL is carried out sequentially, starting from the definition of the DSM, followed by transformation through cross-model transformations into a semantic model (metamodel) of the DSL and transformation to the level of a specific syntax, followed by the definition of individual commands in the case of a text DSL. Based on the ideas of cross-model transformations and the object representation of the models used, the approach guarantees the consistency of all levels of the DSL with the DSM, since the formal transformation mechanisms are based on graph transformations while preserving the essential characteristics (properties) of the models.

The efficiency and flexibility of the proposed approach is shown an example of use during the development of two different general. It is important to note that the approach is universal and is applied to both visual and textual types of DSL for software systems, which is demonstrated in the situations under consideration.

The proposed approach, models and methods can be used to develop adaptive general-purpose systems. However, their use is not recommended for highly loaded, high-performance, critical systems, where it is important to ensure a high level of security and system performance in the case of a large number of simultaneous users and requests.

As a result of the implementation of software for two different subject areas, reusable data structures and algorithms for cross-model transformations are distinguished to support the evolution of DSL. Thus, the basis for the creation of general-purpose tools is formed.

In the future, a possible extension of the approach can be the implementation of a full-fledged universal environment for the development of DSL (by the example of MetaEdit+ [32]), which allows not only to modify the set of created DSL

constructions, but also to support the organization of the evolution of any DSL created by the tools of the environment

LITERATURE

1. Anureev, I.S. Predmetno-orientirovannye sistemy perekhodov: ob"ektnaya model' i yazyk // Sistemnaya informatika. – 2013. – №. 1. – pp. 1-34. (Ануреев, И.С. Предметно-ориентированные системы переходов: объектная модель и язык // Системная информатика. – 2013. – №. 1. – с. 1-34.)
2. Gajfullin, B.N., Tumanov, V.E. Predmetno-orientirovannye sistemy nauchnoj osvedomlennosti v nauke i obrazovanii // Sovremennye informacionnye tekhnologii i IT-obrazovanie. – 2012. – №. 8. – pp. 741-750. (Гайфуллин, Б.Н., Туманов, В.Е. Предметно-ориентированные системы научной осведомленности в науке и образовании // Современные информационные технологии и ИТ-образование. – 2012. – №. 8. – с. 741-750.)
3. OST R ISO/MEK 25010-2015: SISTEMNAYA I PROGRAMMNAYA INZHENERIYA Trebovaniya i ocenka kachestva sistem i programmnogo obespecheniya (SQuaRE). Modeli kachestva sistem i programmnyh produktov // Moskva: Standartinform. – 2015. (ГОСТ Р ИСО/МЭК 25010-2015: СИСТЕМНАЯ И ПРОГРАММНАЯ ИНЖЕНЕРИЯ Требования и оценка качества систем и программного обеспечения (SQuaRE). Модели качества систем и программных продуктов // Москва: Стандартинформ. – 2015.)
4. Ershov, A.P. Denotacionnyj podhod k opisaniyu transformacionnoj semantiki: Slajdy doklada na seminare proekta CIP v Myunhenskom tekhnicheskom universitete. – 1982. (Ершов, А.П. Денотационный подход к описанию трансформационной семантики: Слайды доклада на семинаре проекта CIP в Мюнхенском техническом университете. – 1982.)

5. Konstruirovanie kanonicheskih informacionnyh modelej dlya integrirovannyh informacionnyh sistem / V. N. Zaharov, L. A. Kalinichenko, I. A. Sokolov, S. A. Stupnikov // Informatika i ee primeneniya. — 2007. — T. 1, № 2. — pp. 15–38. (Конструирование канонических информационных моделей для интегрированных информационных систем / В. Н. Захаров, Л. А. Калиниченко, И. А. Соколов, С. А. Ступников // Информатика и ее применения. — 2007. — Т. 1, № 2. — с. 15–38.)
6. Suhov, A.O. Klassifikaciya predmetno-orientirovannyh yazykov i yazykovyh instrumentariev // Matematika programmnyh sistem. – 2012. – pp. 74-83. (Сухов, А.О. Классификация предметно-ориентированных языков и языковых инструментариев // Математика программных систем. – 2012. – с. 74-83.)
7. Fedorenkov, V.G., Balakshin, P.V. Osobennosti primeneniya predmetno-orientirovannyh yazykov dlya testirovaniya veb-prilozhenij // Programmnye produkty i sistemy. – 2019. – №4. – pp. 601-606. (Федоренков, В.Г., Балакшин, П.В. Особенности применения предметно-ориентированных языков для тестирования веб-приложений // Программные продукты и системы. – 2019. – №4. – с. 601-606.)
8. Akehurst, D., Kent, S. A relational approach to defining transformations in a metamodel // J.-M. Jézéquel, H. Hussmann, and S. Cook (eds.), Proc. Fifth International Conference on the Unified Modeling Language – The Language and its Applications, LNCS. – 2002. – vol. 2460. – pp. 243–258.
9. ATL Transformation Language [Электронный ресурс] – URL: http://www.eclipse.org/atl/.
10. Bell, P. Automated Transformation of Statements within Evolving Domain Specific Languages // Computer Science and Information System Reports. – 2007. – pp. 172–177.

11. Bergmann, G. et al. Change-driven model transformations / G. Bergmann, I. Ráth, G. Varró, D. Varró // Software & Systems Modeling. – 2021. – v. 11(3). – pp. 431-461.
12. Cabot, J. et al. Verification and validation of declarative model-to-model transformations through invariants / J. Cabot, R. Clarisó, E. Guerra, J. de Lara // J Syst Softw. – 2010. – v. 83(2). – pp. 283–302.
13. Cleenewerck, T. Component-Based DSL Development // Software Language Engineering. – 2003. – pp. 245-264.
14. Cleenewerck, T. Evolution and Reuse of Language Specifications for DSLs (ERLS) / T. Cleenewerck, K. Czarnecki, J. Striegnitz, M. Volter // Object-Oriented Technology. ECOOP 2004 Workshop Reader. – 2004. – pp. 187-201.
15. Eclipse Graphical Modeling Project (GMP) [Электронный ресурс] – URL: http://www.eclipse.org/modeling/gmp/.
16. Evans, E. Domain-Driven Design: Tackling Complexity in the Heart of Software. – Addison-Wesley, 2013. – 560 p. – ISBN 978-0-321-12521-7.
17. Fowler, M. Domain specific languages / M. Fowler, R. Parsons. – Addison Wesley, 2010. – 413 p. – ISBN: 978-0-321-71294-3.
18. Gómez-Abajo, P. A domain-specific language for model mutation and its application to the automated generation of exercises / Pablo Gómez-Abajo, Esther Guerra, Juan de Lara // Computer Languages, Systems & Structures. – 2016. – v. 49. – pp. 152-173.
19. Guizzardi, G. Ontological foundations for structural conceptual models. – Enschede, The Netherlands, 2005. – 418 p. – ISBN 90-75176-81-3.
20. Guizzardi, G., Halpin, T. Ontological Foundations for Conceptual Modeling // Applied Ontology. – 2008. – v. 3. – pp. 91-110.
21. Haav, H.-M. et al. Ontology-Based Integration of Software Artefacts for DSL Development / H.-M. Haav, A. Ojamaa, P. Grigorenko, V. Kotkas // On the Move to Meaningful Internet Systems: OTM 2015 Workshops. Lecture Notes in Computer Science. – 2015. – v. 9416. – pp. 309-318.

22. Hausmann, J.H., Heckel, R., Sauer, S. Extended model relations with graphical consistency conditions // UML 2002 Workshop on Consistency Problems in UML-based Software Development. – 2002. – pp. 61–74.
23. Kelly, S., Tolvanen J.-P. Domain-specific modeling: enabling full code generation. – Hoboken, New Jersey, USA: Wiley-IEEE ComputerSociety Press, 2008. – 444 p. – ISBN 978-0-470-03666-2.
24. Kessentini, W., Sahraoui, H., Wimmer, M. Automated metamodel/model co-evolution: A search-based approach // Information and Software Technology. – 2019. – pp. 49-67.
25. Kogalovsky M. R., Kalinichenko L. A. Conceptual and ontological modeling in information systems // Programming. – 2009. – v. 35 (5). – pp. 241-256.
26. Kosar, T., Bohra, B., Mernik, M.: Domain-Specific Languages: A Systematic Mapping Study // Information and Software Technology. – 2016. – pp.77-90.
27. Laird, P., Barrett, S. Towards Dynamic Evolution of Domain Specific Languages // Software Language Engineering. – 2010. – pp. 144-153.
28. Luoma, J., Kelly, S., Tolvanen, J.-P. Defining domain-specific modeling languages: Collected experiences // Proceedings of the 4th OOPSLA Workshop on Domain-Specific Modeling (DSM'04). – 2004.
29. Mengerink, J.G.M. et al. Udapt Edapt Extensions for Industrial Application / J.G.M. Mengerink, A. Serebrenik, M. van den Brand, R.R.H. Schiffelers // ITSLE 2016 Industry Track for Software Language Engineering October 31, 2016, Amsterdam, Netherlands. – 2016. – pp. 21-22.
30. Mengerink, J.G.M. et al. A Complete Operator Library for DSL Evolution Specification / J.G.M. Mengerink, A. Serebrenik, R.R.H. Schiffelers, M.G.J. van den Brand // MDSE 32nd International Conference on Software Maintenance and Evolution, 2016. – 2016. – pp. 144-154.
31. Mernik, M., Heering, J., Sloane, A. When and how to develop domain-specific languages // ACM computing surveys (CSUR). – 2005. – v. 37, no. 4. – pp. 316-344.
32. MetaCase+ [Электронный ресурс] – URL: http://www.metacase.com/

33. Milne, C., Strachey, R. A Theory of Programming Language Semantics (2 Vol). – Chapman & Hall, 1976. – 868 p. – ISBN 978-0-412142-60-4.
34. Parr, T. (2012). Language Implementation Patterns: Create Your Own Domain-Specific and General Programming Languages. – Pragmatic Bookshelf, 2012. – 389 p. – ISBN 978-1-934356-45-6.
35. Pereira, M.J.V. et al. Ontological approach for DSL development / M.J.V. Pereira, J. Fonseca, P.R. Henriques // Computer Languages, Systems & Structures. – 2016. – v. 45. – pp. 35-52.
36. Popovic, A.A. et al. DSL for modeling application-specific functionalities of business applications / A. Popovic, I. Lukovic, V. Dimitrieski, V. Djuki // Computer Languages, Systems & Structures. – 2015. – pp. 69-95.
37. Schürr, A. Graph-Transformation-Driven Correct-by-Construction Development of Communication System Topology Adaptation Algorithms // I. Schaefer, D. Karagiannis, A. Vogelsang, D. Méndez, C. Seidl (Eds.): Modellierung, LNI. – 2018. – pp. 15–29.
38. Sprinkle, J. A domain-specific visual language for domain model evolution // Journal of Visual Languages & Computing. – 2004. – pp. 291–307.
39. Stoy, J.E. Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory. – MIT Press, 1985. – 450 p. – ISBN: 978-0-262-69076-8.
40. Ulitin, B., Babkin, E., Babkina, T. A Projection-Based Approach for Development of Domain-Specific Languages // Lecture Notes in Business Information Processing Issue 330: Perspectives in Business Informatics Research. – 2018. – pp. 219-234.