Scientific Visualization, 2024, volume 16, number 4, pages 82 - 101, DOI: 10.26583/sv.16.4.08

An Approach to Developing Data Visualization Tools Based on Domain Specific Modeling

A. D. Dzheiranian[1,A], I. D. Ermakov[2,B], K. A. Proskuryakov[3,A], L. N. Lyadova[4,A]

[A] National Research University – Higher School of Economics (HSE University), Perm, Russia
[B] Perm State University (PSU), Perm, Russia

[1] ORCID: 0009-0000-8916-2855, addzheyranyan@edu.hse.ru
[2] ORCID: 0000-0003-2897-7158, john.ermakov27@gmail.com
[3] ORCID: 0009-0001-1678-5653, k.proskuryakov22@gmail.com
[4] ORCID: 0000-0001-5643-747X, LNLyadova@gmail.com

Abstract

An approach to the development of data visualization tools is described that provides the ability to customize to the needs of users and the specifics of the domains in which they work, based on domain-specific modeling. The results of the analysis of data visualization tools and the possibility of customizing them to subject area based on the needs of users and the tasks they solve are briefly presented. It is shown that existing tools require programming skills from users to customize the data visualization format or to develop new visualization models. It is proposed to use domain-specific modeling tools (language toolkits) to create domain-specific languages (DSL) designed to develop new data visualization models that reflect the specifics of the tasks solved by users. The use of the developed tools does not require users to have professional knowledge of programming languages. The architecture of a knowledge-driven software system is described. The core of the system is a multifaceted ontology, which includes descriptions of languages and domains, as well as rules for generating new languages and transforming constructed models. Languages are designed to describe different classes of diagrams. The system includes tools for automating the creation of new DSLs by mapping the domain ontology onto the metamodel of the base language according to user-specified rules, which are also stored in the ontology. The classification of different types of diagrams forms the basis for creating an ontology of data visualization languages. An example of a basic language for creating diagrams is described. The ability to customize the DSL and transform visualization models developed with its help and generate code that implements the model is demonstrated. It is shown how interactive visualization can be created on the basis of the developed metamodels, extending the possibilities of visualizing large structured data.

Keywords: data visualization, interactive visualization, domain-specific modeling, domain-specific language, metamodeling, grammar, multifaceted ontology, model transformation.

1. Introduction

Visualization tools have become widespread in various industries, business functions and IT disciplines, both in the private and public sectors. They are actively used in such areas as energy, cartography, medicine, finance, sociology and many others. In this context, data visualization serves as a method of data analysis. The efficiency of experts and analysts is largely determined by the quality of data visualization and the results of their analysis.

Numerous studies have been devoted to the issues of data visualization quality.

Unreadable and confusing visualizations spread misinformation, mislead, and reduce the effectiveness of researchers, the quality of management decisions, etc. The authors of the paper [1] emphasize the scale of the problem by referring to resources that collect tens of thousands of users criticizing graphs and charts (reddit.com/r/dataisugly) made with obvious errors. Existing data visualization methods allow building mostly simple and overly generalized visualizations, which, in addition, often lack systematicity and universality. The lack of proper tools for assessing the quality of data visualization further exacerbates the problem. The article proposes VisQualdex, a systematic set of guidelines for static data visualization. The categorization is based on the theory of graphic grammars. Dozens of criteria are proposed to identify various errors (errors of different categories and scales). The proposed set of guidelines has been peer-reviewed and tested by experts in the fields of data visualization, data science, graphic design, information technology, and computer science. The implementation of the recommendations is available as a web server, developed as a single-page application in JavaScript using the principles of Vue.js and Material Design.

Researchers pay special attention to the issues of using visualization tools in education.

The authors of the paper [2] show that infographics are important for the presentation and transmission of complex information, in particular, they discuss the impact of visualization on the results of the educational process organized in interaction with infographics and multimedia. The results of the study showed that the use of infographics and visual training is an effective approach to organizing the educational process.

The purpose of the study, the results of which are presented in [3], is to study the theoretical and practical aspects of knowledge visualization in the educational process. The authors compare different models of knowledge representation, consider the principles of visualization and the structure of visualization competencies, including analytical, algorithmic, and other components. Using various methods of modeling, visualization and systematization of data, the authors conclude that visualization of knowledge requires taking into account the features of the educational process in the context of didactic principles, etc. The formation of visualization skills in the course of educational activities is a difficult task, depending on many factors. These skills are the basic components of building visualization competence. Solving the knowledge visualization problem requires the use of modern innovative technologies covering basic digital technologies, big data technologies, multidimensional data mining methods, etc.

The software tools that are used in the educational process and affect the formation of professional competencies among students are discussed in the paper [4]. The goal of the project is to develop a technology for creating educational VR content that should increase the efficiency of learning using many different software environments used in the process of forming the final virtual space.

The authors of the paper [5] describe an approach to understanding the idea of the graph models cognitive clarity. They present a conceptual scheme for structuring concepts related to cognitive clarity. According to this scheme, they distinguish the factors of the formation of cognitive clarity. Cognitive clarity is defined as a set of intrinsic characteristics of the visual image of a model, the effects of which are manifested by visual analysis of the model. The authors conclude that the factors of formation of cognitive clarity are of the greatest interest (due to their constructiveness). The article provides a detailed diagram of the approach to understanding the idea of cognitive clarity and discusses the individual components of this scheme. A generalized algorithm is proposed for preparing and conducting an experiment to solve a certain problem of visual analysis by an analyst with the participation of factors, the influence of which is hypothesized before the experiment. As a result, the effect of changing the level of cognitive clarity is assessed and the nature of dependence on given factors or its absence is revealed, which makes it possible to accept, reject or clarify the initial hypothesis.

The effectiveness of data visualization is discussed in articles [6, 7]. The perception of graphic information is associated with certain visual cognitive processes, based on the study of which, the authors give general recommendations for creating effective scientific visualizations.

The needs of end users (data analysts) include the need to create custom chart types for specific tasks and subject areas, since basic chart types with basic geometry can limit information transfer [1] and lead to inefficient visualization, which, in turn, can lead to errors in decision making [8].

Users need to create visualizations tailored to the specifics of the tasks and subject areas they solve. Thus, it is necessary to incorporate custom visualization specifications into data visualization tools. These specifications define how users can specify their requirements for creating visualizations [9].

Static visualization may not be effective for analyzing large data sets, so there is a need to quickly and easily create interactive visualizations [10]. These methods should be able to work with different types and data sources [9].

Researchers [8, 11] note a limited degree of flexibility in manipulating chart elements and a lack of focus on the user's real needs for existing data visualization tools. Often this setting requires the use of the programming language [10]. Most users don't have deep programming knowledge and skills. Therefore, there is a need to develop no-code or low-code platforms for creating customized visualization models.

The authors of the paper [12] discuss various techniques and tools for data visualization, offer their classification, formulate the problems faced by users, and new opportunities that should be provided by promising visualization tools. In particular, it is noted that one of the key tasks is to develop methods for automating data visualization, ensuring efficient and interactive visualization, regardless of the size and complexity of the data. The implementation of these methods facilitates research in areas with intensive use of data [13], allows continuous monitoring and analysis of this data, visualization of the analysis results.

Automatic visual analysis involves finding, cleaning, integrating, and visualizing data. Ontologies are widely used to solve these problems.

In the papers [13, 14] authors propose to use ontologies as part of architecture, as the core of an analytical knowledge-driven platform. In this case, a multifaceted ontology is used, which allows to implement semantic searching and indexing data, avoid duplication of data, to expand the functionality of information and analytical systems, to create domain specific languages (DSLs) and research models and scenarios with these DSLs, as well as automatically interpret data and data analysis results to provide them for different groups of users according to their familiar terminology.

The available data visualization toolkits can be divided into the following groups:

1) spreadsheets (e.g., Excel, Google Sheets),
2) analytical platforms (e.g., Microsoft Power BI, Tableau),
3) chart editors (e.g., Miro, ChartBlocks).

The standard tools of the first two groups are limited to basic chart types and visual effects customization options. The tools of the third group allow to create customized visualizations, edit the location of elements, but do not provide the ability to customize to user's domains.

The use of general-purpose programming languages (for example, Python libraries for data visualization: Matplotlib, Seaborn, Plotly, etc.) contributes to the creation of expressive visualizations for solving specific problems but requires programming skills from chart developers. Also, the created solution cannot be reused for other visualizations, and it is a "black box" where it is not clear how the visualization is configured [10].

Visualization reflecting the specifics of the subject area can greatly contribute to a better understanding of the presented data, the results of their analysis, formal models. The creation of such tools can be based on Domain Specific Modeling (DSM), which involves the creation and use of Domain Specific Languages (DSLs) to develop new data visualization models, customized variants of diagrams.

There are few publications on creating DSL for data visualization. Based on the overview, most DSLs focus on a small set of standard charts (pie charts, histograms, etc.) or visualization of specific data types (for example, geospatial, etc.). They differ in levels of abstraction, usage contexts, and implementation capabilities.

Article [15] describes the process of developing DSL for building and transforming data visualization methods. This DSL is built into the Haskell programming language. Authors provide several levels of abstraction: at the lowest level, the user can create an element consisting of a specific primitive form and a set of visual parameters. It is important to note that the basic constructions of the language are limited to histogram and pie chart. However, users can arrange their elements differently to create more complex renderings.

The variation model of visualization implemented using DSL built into the PureScript programming language is presented in paper [16]. This DSL allows you to create variational visualizations and their combinations, for example, overlapping alternative histograms. The article also discusses methods for representing variation and adding variation to visualizations through DSL. Developed tools provide creation, management, navigation and rendering of various visualizations.

The researchers in the paper [17] present a DSL focused on data geovisualizations. They use a compiler to facilitate automatic creation of visualizations and data preprocessing. Their system uses multi-core parallelism to speed up data preprocessing.

An approach that helps the user create domain-specific visualizations of models using CSP (Communicating Sequential Processes) is suggested in paper [18]. CSP is a formal language primarily used to describe parallel and distributed systems. The authors have successfully created various visualizations of CSP models demonstrating the capabilities of the proposed approach. However, these tools are not universal, the capabilities of the language limit its use.

These tools enable users to develop new types of charts. But there are no customization options for various subject areas, or these options are limited. In addition, professional programming skills are required from users, which also limits the use of the described tools.

A *language-oriented approach* [19] (in this case it is *DSM-based approach*) to the implementation of data visualization tools can become the main one in the development of a data visualization system with the ability to customize to the needs of users.

Creating new visualization options customized for specific domains and user tasks is divided into two stages:

1. Developing a *domain-specific language* (*DSL*) that reflects user needs.
2. Building *customized data visualization* based on the created DSL.

To create DSL, DSM platforms (language toolkits) are used, which automate the development of model editors and their transformation tools, code generators. The task of reducing the complexity of creating the languages themselves and automating their development is becoming urgent.

Thus, end users face two questions when solving data visualization tasks:

1. How to create or customize visualizations according to the user's needs?

2. How to reduce programming competency requirements when using visualization tools that allow customization?

To expand the capabilities of visualization tools, a language-oriented approach is proposed, the use of knowledge-driven language tools that allow automating the creation of domain specific languages based on the use of a multifaceted ontology [19, 20, 21, 22].

2. Generalized Structure of Knowledge-Driven Language Toolkits

The purpose of the presented project is to develop and approve an approach to creating tools for effective data visualization, considering the recommendations proposed in [6, 7], which provide the ability to create custom types of diagrams for specific tasks and domains, as well as the ability to create interactive visualizations [10].

The peculiarity of requirements lies in the need to configure visualization for the specifics of domains and tasks solved by users, or to develop new types of diagrams. This result can be achieved through combining different types of diagrams and interactivity, etc. These requirements should be considered when creating tools for developing visualization models (graphs, schemes, diagrams).

The basis for the implementation of data visualization tools proposed in this paper is a *knowledge-driven DSM platform* that provides the ability to create hierarchies of domain-specific languages [20, 21, 22] designed to create new types of models (new graphic notations) that meet the needs of users. Reducing the complexity of developing new DSLs is achieved using basic languages, their addition and combination in accordance with the rules determined by users. Visualization models built using the created DSL can be translated into program code that implements efficient user visualization of data, based on the specified transformation rules of the "Model–Text" type.

The main *functional requirements* for data visualization tools are shown in the format of the Use Case diagram in Figure 1. A *domain expert* must develop a domain ontology in which users (*visualization developers*) solve their problems. The *DSM expert* is responsible for developing metamodels of basic languages (in this case, languages for developing basic types of diagrams that can be used as the basis for creating new DSLs that implement user needs), as well as for developing rules for generating new languages that reflect the specifics of the domains in which users work – rules for mapping the ontology of the subject area to the metamodel of the selected base language. When developing a visualization, the user can use the existing DSL by configuring parameters or user can develop a new DSL based on existing language. Generating code in a programming language using the built visualization model allows you to create interactive visualizations controlled by events, combine different types of visualizations when working with complex data structures obtained from different sources. Transformation (code generation) rules are developed by a DSM expert.

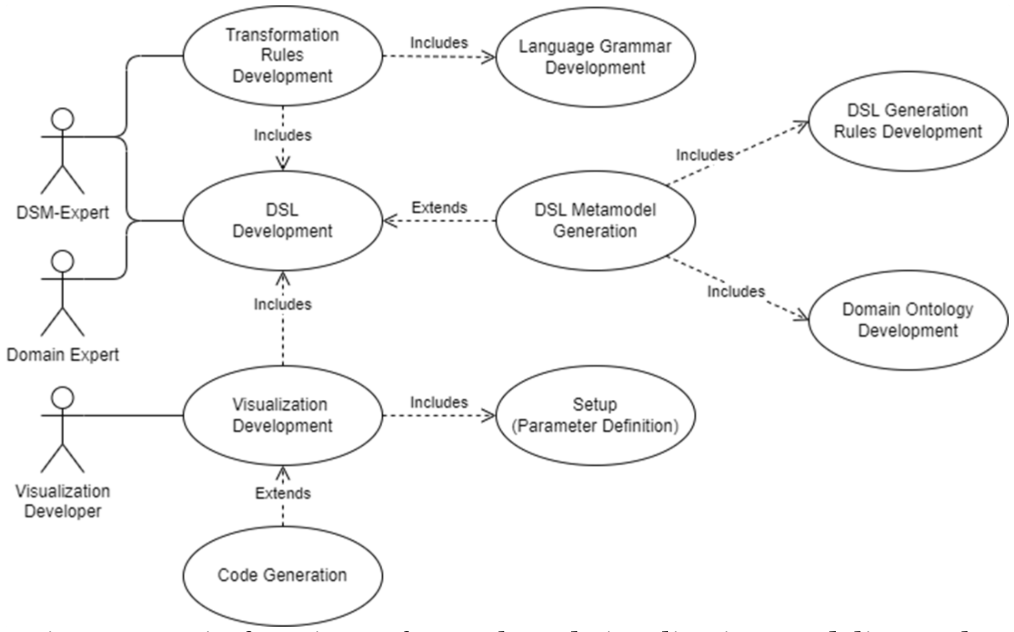
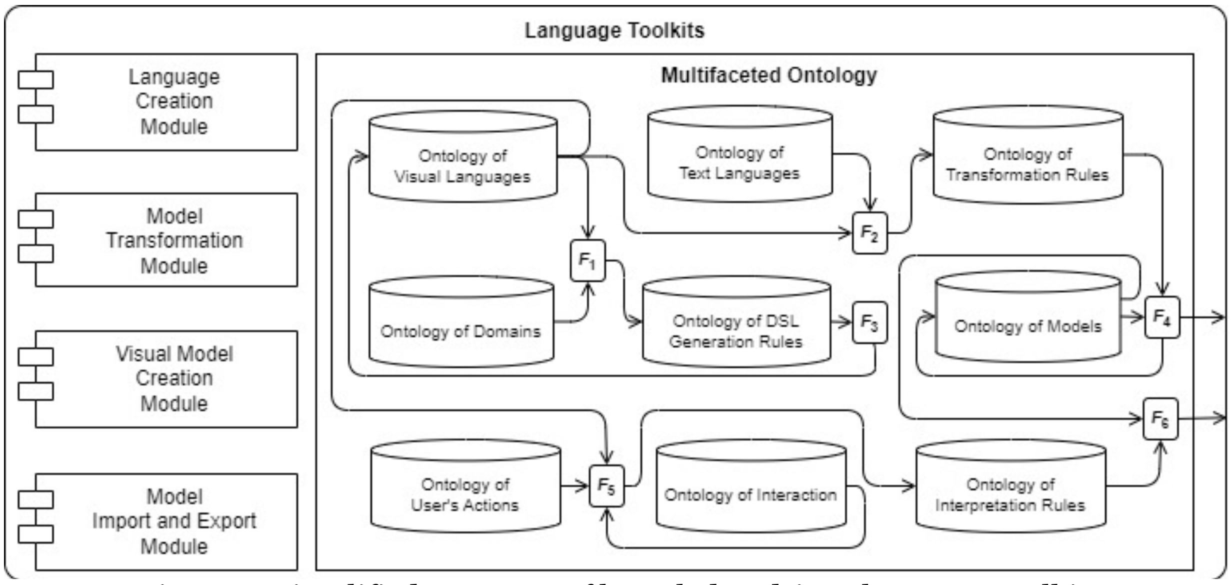

Transformation Rules Development
Includes
Language Grammar Development
DSL Generation Rules Development
Includes
Includes
DSM-Expert
DSL Development
Extends
DSL Metamodel Generation
Domain Expert
Includes
Includes
Domain Ontology Development
Visualization Development
Includes
Setup (Parameter Definition)
Visualization Developer
Extends
Code Generation

Figure 1: Main functions of DSM-based visualization modeling tools

The core of the language toolkits with which the described functions should be implemented is a multifaceted ontology [20, 21, 22]. It contains domain descriptions and includes an ontology of languages, where visual languages (languages for creating diagrams, etc.) are described using metamodels, and text languages grammars are represented as Wirth diagrams. Rules for generation visual DSL metamodels and code generation rules (rules of transformation "Model-Text" type) are included in the ontology.

The development of DSL metamodels, new DSL generation rules, and code generation does not require knowledge of programming languages because users work with editors and designers in a visual environment. To create an ontology of the subject area, it is proposed to use the Mask method [23]. Thus, the requirements for user qualifications and programming skills are reduced.

The generalized structure of the language toolkits used to develop a prototype of data visualization tools based on a language-oriented approach is shown in Figure 2.



Language Toolkits
Language Creation Module
Model Transformation Module
Visual Model Creation Module
Model Import and Export Module
Multifaceted Ontology
Ontology of Visual Languages
Ontology of Text Languages
Ontology of Transformation Rules
Ontology of Domains
Ontology of DSL Generation Rules
Ontology of Models
Ontology of User's Actions
Ontology of Interaction
Ontology of Interpretation Rules
F_1
F_2
F_3
F_4
F_5
F_6

Figure 2: Simplified structure of knowledge-driven language toolkits

Only components that need to be implemented to create custom DSM-based data visualization are shown in Figure 2. *Multifaceted ontology* is the core of the system. It includes:

– Ontology of *visual modeling languages*, including metamodels of basic languages intended for the development of visual models, as well as metamodels of languages (DSLs) created on their basis.

– Ontology of *text languages*, containing descriptions of text languages grammars.

– Ontology of *transformation rules* type of "Model-Text" (code generation rules) that contains twos, the first elements of which are elements of visual languages metamodels, and the second are non-terminal symbols of text languages constructions.

– Ontology of *domains* in which users solve their tasks for which DSLs are created.

– Ontology of *rules for generating metamodels* of new DSL including a description of the rules for mapping ontologies of domains onto visual languages metamodels selected as basic for generation of DSLs reflecting the specifics of domains.

– Ontology of *models* containing information on created models.

– Ontology of user's *actions* storing information on actions of users when they are working with models (for example, focusing the mouse cursor or clicking).

– Ontology of *interaction*, representing information on possible actions of users when working in interactive mode in the system, for example, building a new model, or scaling its part, etc.

– Ontology of *interpretation rules* storing triples, the first elements of which are elements of metamodels of visual languages, the second elements are user's actions, and the third are interactive actions that can be used in the development of interactive visualizations.

A multifaceted ontology also contains *functions* that implement basic algorithms providing automation of the DSL metamodels generation, model transformation (in particular, code generation – model transformation of "Model–Text" type), creation of interactive models, their interpretation:

– F_1 is a function that automates the development of new languages via mapping the domain ontology elements onto the elements of the selected base language metamodel, creating DSL generation rules. The result is generation (mapping) rules defined by the user using a special constructor.

– F_2 is a function that maps text language constructs to elements of visual language metamodels to create model transformation (code generation) rules.

– F_3 is a function that implements the generation of a new DSL metamodel according to the rules obtained using F_1. The metamodel of the new DSL is preserved in the ontology of visual languages. The new DSL is a subclass of the base language to which the rules for mapping the domain ontology to its metamodel are applied (rules are set by the user when executing F_1).

– F_4 is a function that implements model transformations in accordance with the specified rules (for text target languages, code generation is performed as transformation of the "Model–Text" type).

– F_5 is a function that takes elements of metamodels of visual languages, user scripts and interactive actions to create interpretation rules for interactive models (visualizations).

– F_6 is a function that applies predefined interpretation rules to visual models to provide users with the ability to interact with models, create interactive visualizations.

The multifaceted ontology also includes the ontology of data sources, the ontology of functional modules, which are also used for creating visualization models.

The formal basis for creating visual models is a hypergraph with poles (*HP*-graph) [22].

When developing visual languages, language toolkits build their descriptions in the form of graph grammars or metamodels. To create these descriptions, the corresponding metalanguages should be used (like formal grammars of textual languages, which can be set in the form of BPF (Backus-Naur form) or Wirth diagrams). The metalanguage should provide the ability to represent the *HP*-graph, all its elements. Assessment of metalanguages is given in Table 1. The most suitable metalanguage in terms of representing all the elements that can be used in the graphical model is the *GOPPRR* language. (*Graph*, *Object*, *Port*, *Property*, *Relationship*, *Role*) of the MetaEdit+ DSM platform, where *Graph* represents the whole model, *Object* is a model element represented by a vertex, *Port* represents a point through which a vertex-object is associated with other vertices, *Property* is a property of a vertex or relationship, *Relationship* is a relationship between objects, *Role* is the role of the apex in communication.

Table 1: Comparison of metalanguages used in language toolkits

Metalanguage	**Model**	**Object of Model**	**Relationship**	**Role**	**Pole**	**Property**
EMOF	–	Class	Property, Association	–	–	Property
Ecore	–	EClass	EReference	–	–	EAttribute
GOPPRR	Graph	Object	Relationship	Role	Port	Property
ArkM3	–	Class	Association, Composition	–	–	Property
WebGME MML	FCO	FCO	Pointer, Set, Connection, Containment, Inheritance, Mixin	Connection Role	–	Attribute

None of the languages allows you to describe in the model "generalized" connections in which many vertices can participate.

The developed graph model (*HP*-graph) has expressive capabilities that allow you to represent various types of visual models, schemes, and diagrams. To implement language toolkits, a language *HPGPR* is proposed [22], which is an extension of GOPPRR. The developed metalanguage (Figure 3) allows to describe metamodels of visual languages using all the elements of *HP*-graphs.

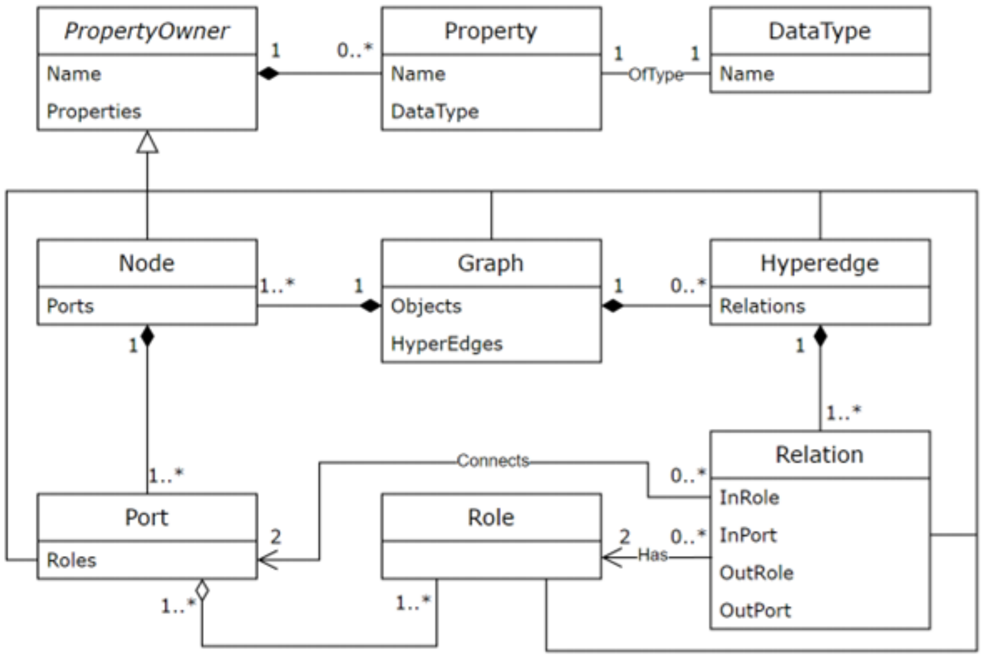


Figure 3: Description of metalanguage HPGPR

To implement the proposed approach, the following tasks must be solved:

1. Analyzing different types of diagrams and building an ontology of data visualization languages based on the classification of diagrams.

2. Developing metamodels of basic visual languages for different types of diagrams used for data visualization (it is the basis for creating new types of visualization).

3. Developing rules for generating new DSLs (new data visualization languages) that reflect the specifics of user's domains and generating new types of diagrams based on the developed rules.

4. Developing transformation (code generation) rules to implement a new type of data visualization.

The results need to be illustrated with examples of data visualization using designed facilities.

3. Developing an Ontology of Data Visualization Languages

The variety of visualization methods is quite large and continues to expand, which is confirmed by the constant emergence of new specialized types of diagrams (tree, chord, network, etc.).

The choice of the most suitable type of visualization is primarily determined by what task the user solves, what idea he should convey using the diagram [24, 25]. To determine which visualization methods are sufficient to implement most visualization ideas, it is necessary to dwell on a specific taxonomy, to classify data visualization methods.

It was decided to focus on the five-category structure proposed by Andy Kirk [25]:

1) comparison of categories;
2) evaluation of hierarchies and relations of part to whole;
3) presentation of changes over time;
4) determination of types of connections and relationships;
5) mapping geospatial data.

When developing the classification, the following characteristics were considered:

1) the degree of popularity of imaging methods;

2) the inclusion, in addition to standard visualization methods (linear graphs, histograms, pie charts, histograms, scatter diagrams, etc.) of methods for visualizing abstract data characterized by multidimensionality and the absence of explicit spatial bindings;

3) the ability to present any type of visualization in an interactive form.

As a result, a classification was developed that included sixteen visualization methods, distinguished depending on the purpose of creating visualization (Figure 4). This classification is the basis for the development of the ontology of data visualization languages, the creation of metamodels of these languages.

The process of building an ontology of data visualization languages includes the following steps:

1. A formal description of an abstract diagram including properties common to all chart types (title, legend, width, height, and so on).
2. Distinguishing types of diagrams into separate classes based on the developed classification of diagrams (Figure 4).
3. Add class-specific chart elements to the chart class descriptions.
4. Defining relationships between the classes.

Formalized model is created for each type of diagrams – a representation of the diagram in the ontology is built (metamodel of DSL is created that allows to describe this type of visualization).

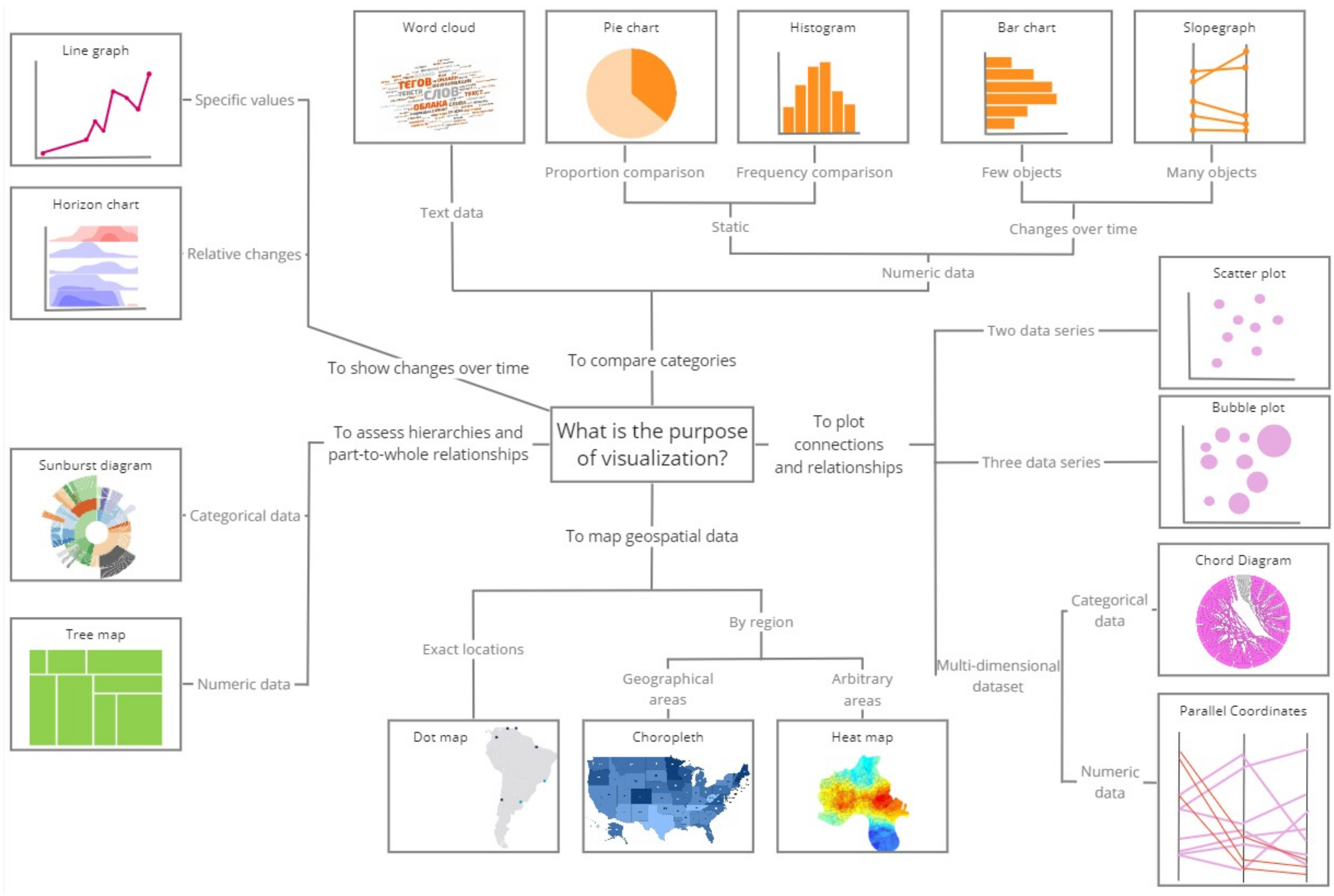


Figure 4: Classification of visualization methods by purpose

Figure 5 shows the abstract diagram class and its subclasses in the ontology. Each of the diagrams consists of specific elements with their own properties, which are also presented in the ontology (Figure 6).

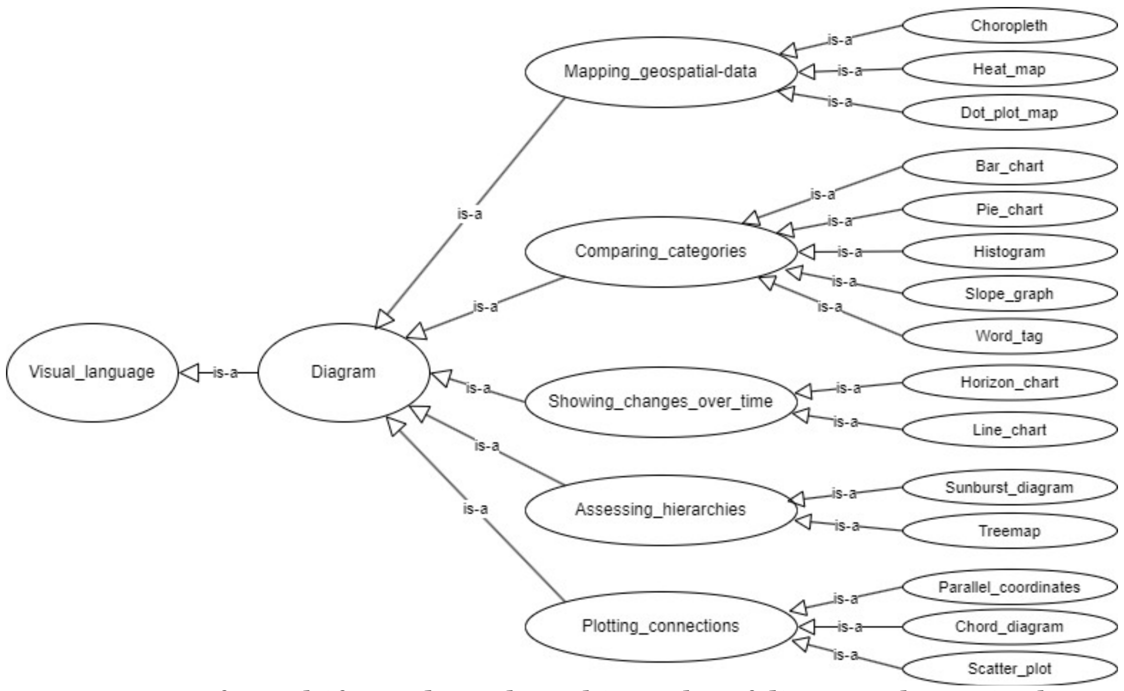


Figure 5: Fragment of a multifaceted ontology: hierarchy of diagram classes in the ontology

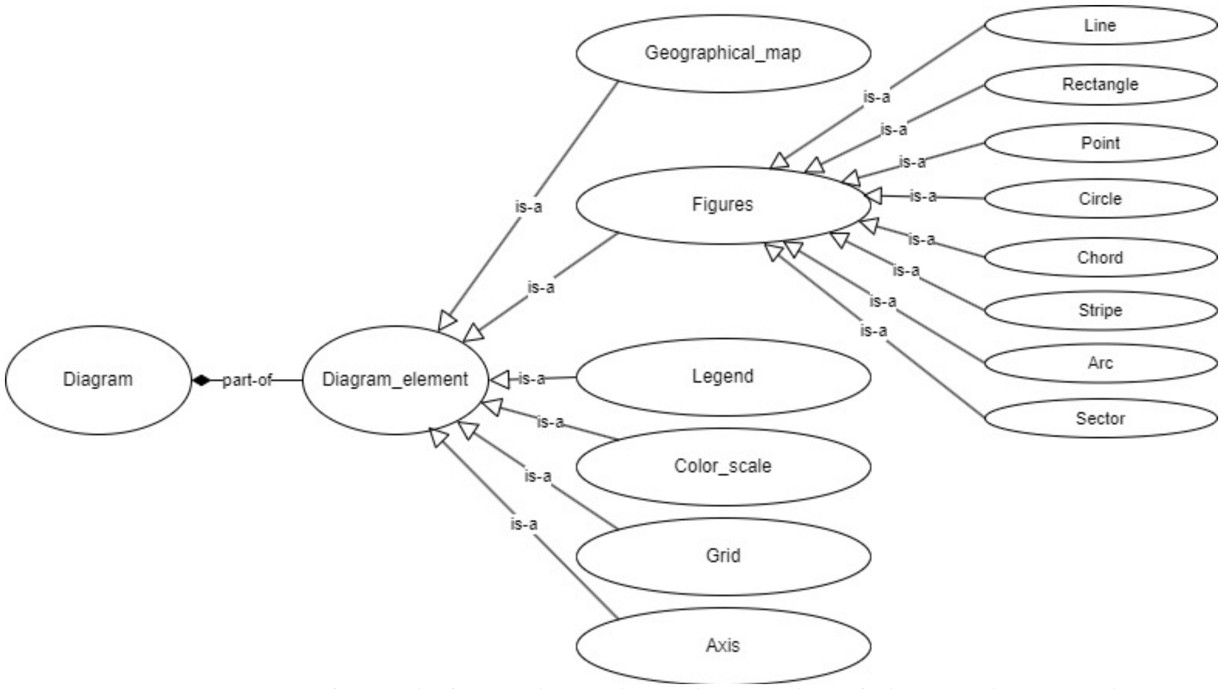


Figure 6: Fragment of a multifaceted ontology: hierarchy of chart's element classes in the ontology

The designed ontology is the basis for developing customizable data visualization through the creation of domain-specific languages and the generation of code implementing the created visualization model.

4. Customized Data Visualization with DSLs: Experimental Results

To approve the proposed approach (to implement of a research prototype visualization tools), it is necessary to solve following tasks:

1) to build an ontology of languages and to supplement it with metamodels of basic languages for creating diagrams of existing types (data visualization models in traditional ways);

2) to set rules for generating new DSLs to visualize data in customized style based on domain ontology and basic languages metamodels;
3) to define rules for generating code to implement customized data visualization;
4) to extend data visualization models for creation interactive visualization.

The results of experiments aimed at creating DSLs for describing diagrams for specific domains, customizing languages and implementing interactive data visualization using these languages are described below.

4.1. Developing DSL Metamodels Based on Domain Ontology

After defining the main classes and their unique elements in the ontology of languages, it is necessary to define hierarchical ("is-a" – "class-subclass" relation) and "part-whole" ("part-of") relationships between classes, as was shown above (Figures 5, 6). Relationships "is-a" type are automatically created between the parent class and the child class in the hierarchy (Figure 5).

Special relationships are created to show the relationship between a particular chart and its elements (Figure 6). All elements of diagrams have their own sets of attributes.

Each chart (diagram) type has its own base language. The adaptation of the base language to the needs of users is performed by matching the elements of the diagrams presented in the metamodel of the base language with the elements in the description of the ontology of the corresponding subject area. Figure 7 presents a language metamodel of DSL that is customized to show ratings.

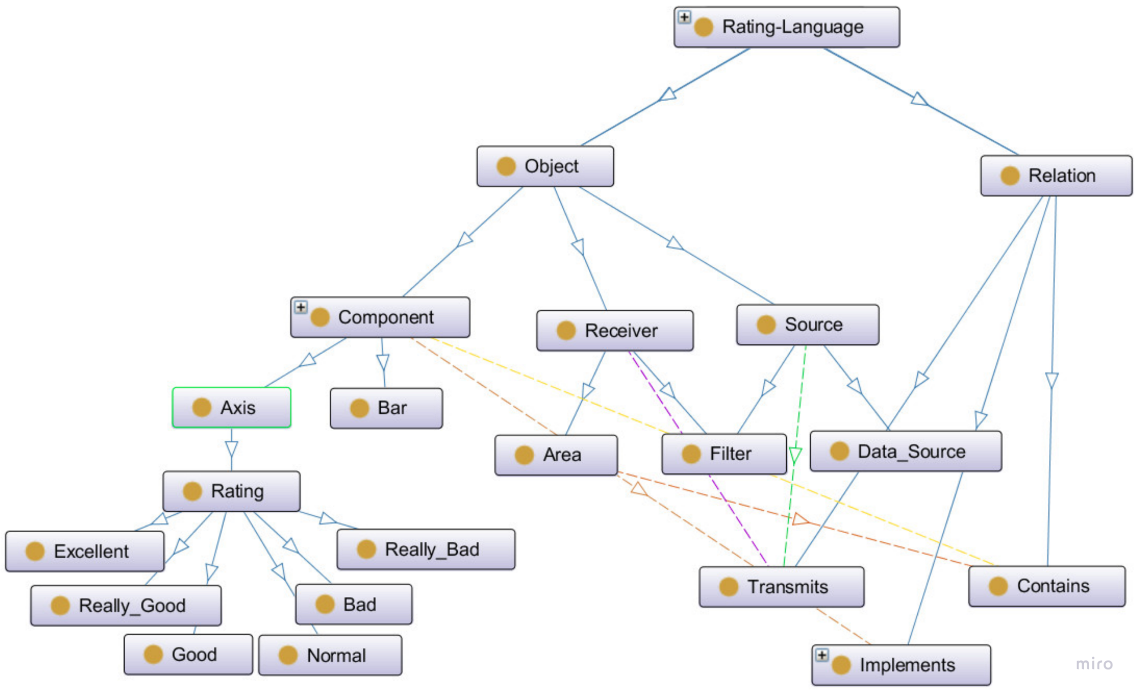


Figure 7: Metamodel of DSL "Rating-Language" in the ontology of visual languages

The designed DSL metamodel presents the basic components of the developed graphical notation, such as "Axis" (axes), "Bar" (column), as well as "Area" (area). Each component has properties (attributes), the description of which isn't shown so as not to clutter up the illustration. The class "Rating" is associated with the axis class, which is used to display the value area of valid ratings, in this case "Excellent", "Really_Good", "Good", "Normal", "Bad" and "Readlly_Bad". Classes such as "Source" and "Receiver" are used to represent the source and sink of data, respectively. Both classes are associated with another "Filter" class that allows filter data according user's requirements.

Using the described language, the user can build his own visualization model, develop a diagram in the visual constructor, determining the values of all attributes specified in the language metamodel. Figure 8 shows a diagram developed using the described DSL.

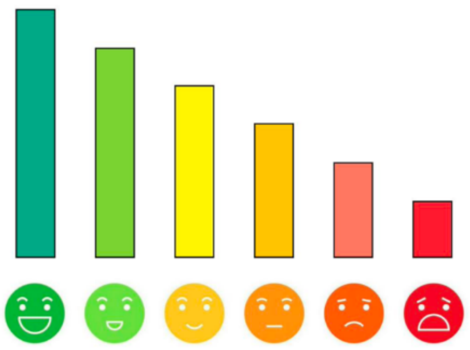


Figure 8: User's chart built with DSL "Rating-Language"

With developed DSL, visualization models are built that meet the needs of users, but the implementation of the developed data visualization model requires either code generation or model interpretation.

Consider another example. Researchers (zoologists) need to visualize data on the distribution of bee species across Europe. To do this, they can use the proposed approach. Users can take the created basic metamodel again, apply the generation algorithm for new domain and get a new language. It will differ from the base one in that it will have no "Axis" component, and the concept of "Figure" is redefined by the concept of "Pie Chart". So, they will be able to build the diagram shown in Figure 9.

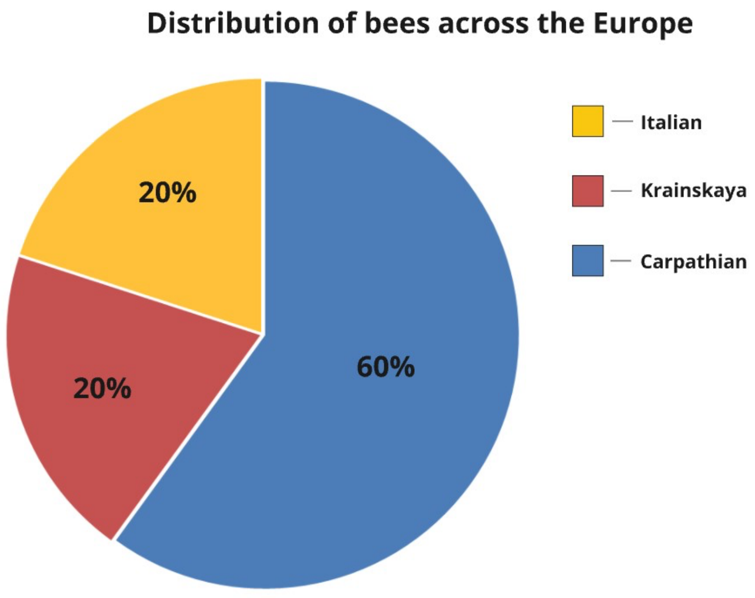


Figure 9: Diagram ("Pie Chart") showing distribution of bee species across Europe

Let's say that from the available data it is possible to extract information not only about the ratio of species, but also the distribution of a particular species by European countries. This information allows users to build new charts detailing the built one. In order not to open a new diagram for each type of bee separately and quickly switch between the necessary data, it is possible to use the "Event" object (event) in the developed model, which allows you to determine the reaction to events, user actions. This object allows users to create interactive visualizations of the data. In this case, for the selected element of the diagram (sector), on which the cursor is placed, it is proposed to build a new diagram using the data of the original data set that is “contained” in the selected fragment. After adding the required component and setting up an additional chart, extended charts should be built interactively to complement the original pie charts. The view of these diagrams is shown in Figure 10.

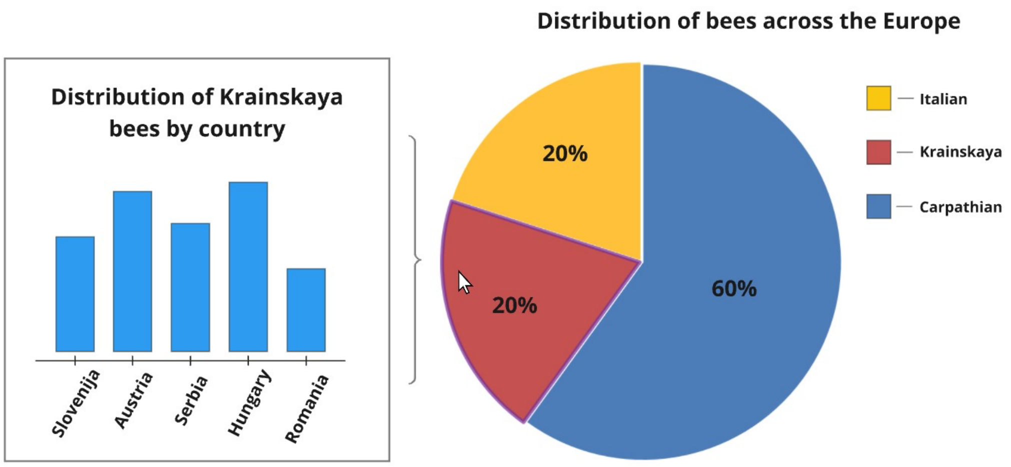


Figure 10: The extended diagram (“Pie Chart”) showing distribution of bee species by European countries

A metamodel of the language designed to create this type of visualization is shown in Figure 11. The diagram metamodel is built on the basis of the language described above (fragments of the ontology of languages are shown in Figures 5, 6) developed to create the different types of diagrams.

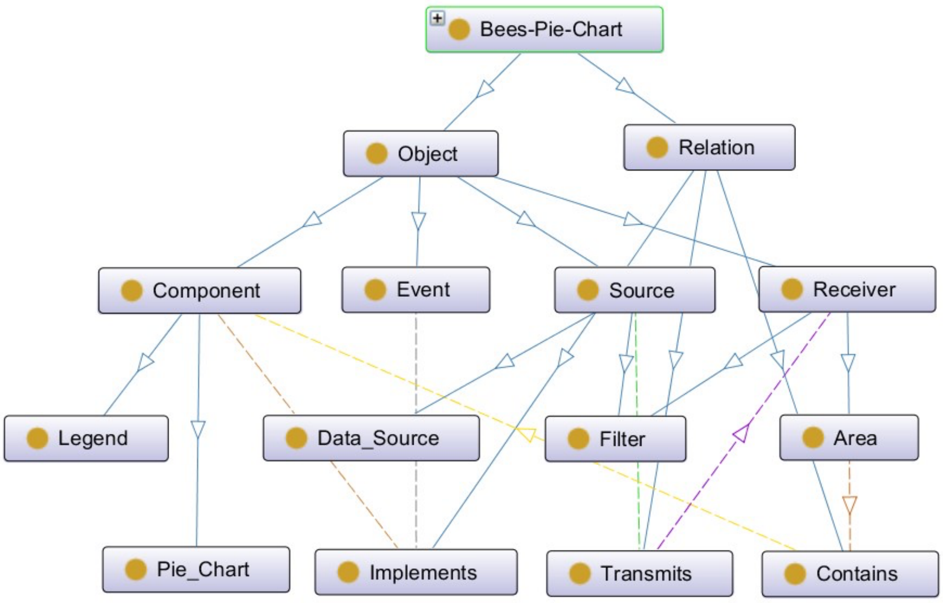


Figure 11: Metamodel of DSL designed to create an interactive visualization based on "Pie-chart"

By applying transformation rules, we can obtain program code that implements the developed interactive visualization model.

4.2. Code Generation for Customized Data Visualization

Most modern DSM platforms use a *template-based code generation approach*, which allows efficient reuse of templates (patterns) [26]. Templates are described not for a specific model, but for a metamodel [27] (in this case, for a visualization language metamodel). Each template consists of two parts – static and dynamic. The *static part* is the same for all models developed with using the language defined by the metamodel, and the *dynamic part* uses the information (parameters, values of attributes, etc.) extracted from the specific model.

In this paper, we propose to use the approach described in [23]. Visual environment (transformation rule constructor) is used for creating rules for transforming models. Each rule consists of the left side, which indicates the objects of the visual language metamodel, and the corresponding right side, which represents the constructions of text languages, which are templates of target code.

Python and *R* are the preferred programming languages for data visualization purposes. *Python* is widely used due to its many suitable libraries, including *Matplotlib*, *Seaborn*, and *Plotly*, as well as its simple syntax. Although *R* is inferior to *Python*, it also has a rich arsenal of visualization tools and remains a popular choice in academia. Thus, it makes sense to use *Python* or *R* code snippets containing library function calls to visualize data as text language constructs in templates.

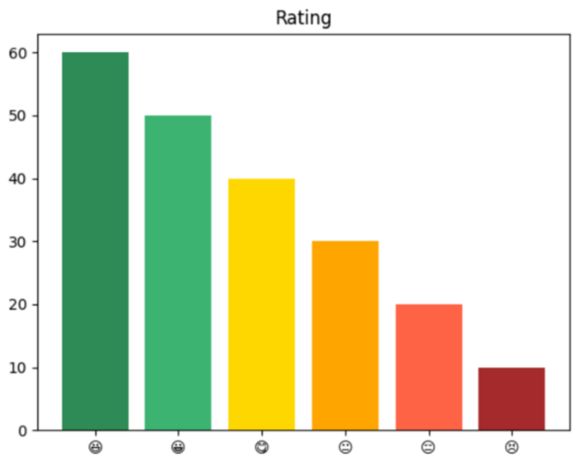
After creation of transformation rules, user can apply them to user's specific models to generate custom visualization code. The code generation algorithm is based on bypassing the internal representation of models in the form of a graph in the ontology.

Users can generate program code in the appropriate programming language for customized visualization according to the rules specified by the user who developed the model applying the developed templates. If transformation rules are designed for a base language, they will apply to all models developed using all DSLs built on the base language. Thus, the developed generation rules can be reused for the entire hierarchy of developed languages. This reduces the complexity of customized data visualization development, simplifies development of new visual models for users.

As an example, the function code for building a histogram in Matplotlib is shown in Figure 12 (*a*). This code is generated on the basis of the above model according to the transformation rules specified by the user in the transformation rule constructor and presented in the ontology. The result of execution of this code is presented in Figure 12 (*b*) as built customized diagram.

```python
import matplotlib.pyplot as plt

groups = ['😆', '😀', '😋', '😐', '😔', '😣']
values = [60, 50, 40, 30, 20, 10]
colors = ['seagreen', 'mediumseagreen', 'gold',
          'orange', 'tomato', 'brown']

plt.bar(groups, values, color=colors)

plt.title('Rating')

plt.show()
```

a



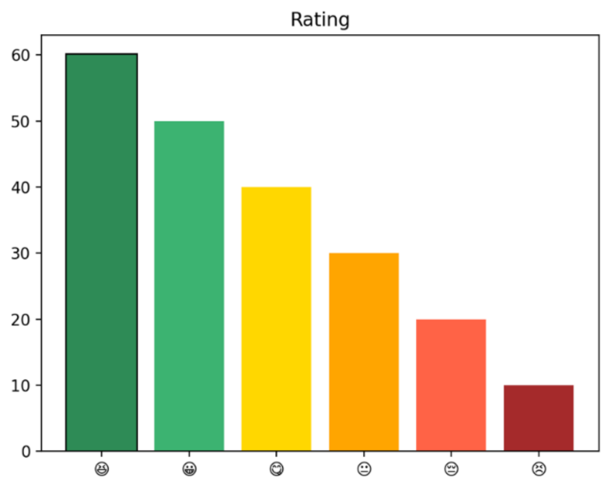
b

Figure 12: The code, generated according to transformation rules (*a*), and the result of this code execution (*b*)

If interpretation rules, described for a language developed by user, contain rules for user interaction with built visualizations, then the code generated at the “Model-Text” transformation will support the described interpretation rules (Figure 13). Thus, the visualization becomes interactive.

```
%matplotlib notebook
import matplotlib.pyplot as plt

groups = ['😆', '😀', '😋', '😐', '😔', '😣']
values = [60, 50, 40, 30, 20, 10]
colors = ['seagreen', 'mediumseagreen', 'gold',
          'orange', 'tomato', 'brown']

fig, ax = plt.subplots()
bars = ax.bar(groups, values)

def on_hover(event):
    for bar in bars:
        if bar.contains(event)[0]:
            bar.set_alpha(1.0)
            bar.set_edgecolor('black')
            bar.set_linewidth(2)
        else:
            bar.set_alpha(0.8)
            bar.set_edgecolor('none')
            bar.set_linewidth(1)
    fig.canvas.draw_idle()

fig.canvas.mpl_connect
        ('motion_notify_event', on_hover)

plt.bar(groups, values, color=colors)
plt.title('Rating')
plt.show()
```

a



b

Figure 13: Interactive visualization code generated by transformation rules with interpretation of events (user actions) (*a*), and the result of this code execution (*b*)

When interpretation rules are added by user, transformation rules are automatically extended with these capabilities without the need to re-describe the transformation rules. The code generated for the same transformation rule, considering the interpretation rule, according to which when the mouse hovers over a specific column, it should be highlighted, is shown in Figure 13 (*a*), and the result of executing the generated code is shown in Figure 13 (*b*). The listing presented in Figure 13 (*a*) shows that the generated code has really become more complicated and now includes processing the mouse pointing event on the histogram columns using the “on_hover” function. When mouse id hovered over a column, its transparency will decrease, and its border will be painted black.

The result of the code generation (applying transformation rules for the metamodel shown in Figure 11 is presented in Figure 14.

```
%matplotlib notebook
import matplotlib.pyplot as plt
labels = ['Carpathian', 'Krainskaya', 'Italian']
sizes = [60, 20, 20]
colors = ['#4682b4', '#e57373', '#fdd835']
explode = (0, 0, 0)

countries = ['Slovenia', 'Austria', 'Serbia', 'Hungary', 'Romania']
values = [10, 25, 20, 30, 15]
fig, ax_pie = plt.subplots(figsize=(16, 7))
wedges, texts, autotexts = ax_pie.pie(
    sizes, explode=explode, colors=colors, autopct='%1.0f%%', startangle=90,
    textprops={'fontsize': 14}, wedgeprops={'linewidth': 1, 'edgecolor': 'white'})
ax_pie.legend(wedges, labels, loc="center left", bbox_to_anchor=(0.0, 0.5), title="Bees")
ax_pie.set_title('Distribution of bees across Europe', fontsize=16)
ax_bar = None

def update_bar_chart():
    global ax_bar
    if ax_bar is None:
        ax_bar = fig.add_axes([0.7, 0.2, 0.25, 0.6])
    ax_bar.clear()
    ax_bar.bar(countries, values, color='skyblue', width=0.4)
    ax_bar.set_title('Distribution of Krainskaya bees by country', fontsize=12)
    ax_bar.set_xlabel('Countries')
    ax_bar.set_ylabel('Count')
    fig.canvas.draw_idle()
def hide_bar_chart():
    global ax_bar
    if ax_bar:
        ax_bar.remove()
        ax_bar = None
        fig.canvas.draw_idle()
def on_hover(event):
    found = False
    for i, wedge in enumerate(wedges):
        if wedge.contains(event)[0]:
            if labels[i] == 'Krainskaya':
                update_bar_chart()
            else:
                hide_bar_chart()
            wedge.set_alpha(1.0)
            found = True
        else:
            wedge.set_alpha(0.8)
    if not found:
        hide_bar_chart()
    fig.canvas.draw_idle()

fig.canvas.mpl_connect('motion_notify_event', on_hover)
ax_pie.axis('equal')
plt.show()
```

Figure 14: Code generated for interactive visualization with metamodel shown in Figure 11

5. Discussion of Results

The DSM-based approach to developing data visualization tools presented in the paper overcomes some of the limitations of existing visualization tools and provides users with the ability to customize data visualization models for different domains and tasks via creating special languages (DSL) without high requirements for user's programming skills.

The ability to reuse created languages, developed transformation rules reduces the complexity of creating visualizations. All developed DSL metamodels and language grammars, transformation rules are stored in the ontology and can be reused. Users can adjust the parameters of models defined with languages, or through creating new languages based on previously developed ones. Users can extend models with interpretation rules for different actions of users to develop interactive data visualizations.

Previously, language tools were used to describe graphic notations, to develop visual languages intended for the design of information systems: to describe data structures, algorithms, etc., to develop and generate code for simulation models.

The results presented in this paper show the promise of the proposed approach to the development of languages and models for various purposes.

As the results presented in this paper have shown, the possibilities of applying this approach can be wider: the language toolkit allows solving problems of effective data visualization and can be integrated into systems for various purposes, in particular, it can be useful in creating analytical systems, etc.

6. Conclusion

The scientific novelty of the approach to the development of language toolkits is in the use of ontology at all stages of the life cycle of the DSM tools, which provides the ability to flexibly tune to different domains and user's tasks, expanding the functionality of the system.

The task of approval the approach when creating data visualization tools is solved:

– Based on the analysis of data visualization methods and tools, metamodels of basic languages were designed for the development of standard types of diagrams. These metamodels are the basis for generating custom visualizations of data obtained from various sources, tuned to the specifics of the user's domains and tasks.

– To illustrate the possibilities of customizing visualization to the needs of users, DSL metamodels are developed on the basis of basic languages, which include non-standard elements of diagrams, as well as elements that provide interactive visualization. Transformation rules have been built for created DSLs (rules for generating code that creates custom visualizations with specified characteristics).

The practical applicability of this approach is shown by simple examples, but in the future, it is supposed to expand the capabilities of the developed software (research prototype), in particular, the possibilities of interactive visualization through the interpretation of the created models. The theoretical basis for developing this approach is Vega-Lite [28] – a high-level grammar that allows to quickly specify interactive data visualization. In addition, it is proposed to expand the possibilities of generating new DSLs, to implement the ability to integrate several languages as base, to combine their capabilities in the new DSL.

Visualization tools can be expanded by means of evaluating built visualizations for compliance with the criteria and recommendations proposed in [6, 7], which should help users avoid errors during development and create effective visualizations.

References:

1. J. Sawicki and M. Burdukiewicz, “VisQualdex: a Comprehensive Guide to Good Data Visualization,” *Scientific Visualization*, vol. 15, no 1, pp 127–149, 2023. DOI: 10.26583/sv.15.1.11.
2. S. M. Alyahya, “Evaluating Computer Interactions and Infographics Usability: Analyzing Individual’s Performance through Viewing Patterns,” *Scientific Visualization*, vol 15, no 5, pp 111–135, 2023. DOI: 10.26583/sv.15.5.10.
3. T. M. Shamsutdinova, “Visualization of Knowledge in the Educational Process,” *Scientific Visualization*, vol 15, no 1, pp 100–111, 2023. DOI: 10.26583/sv.15.1.09.
4. V. A. Nemtinov, A. A. Rodina, A. B. Borisenko, V. V. Morozov., Yu. V. Protasova, and K. V. Nemtinov, “Integrated Use of Various Software Environments for Increasing the Level of Visualization and Perception of Information,” Scientific Visualization, vol 15, no 2, pp 1–10, 2023. DOI: 10.26583/sv.15.2.01.
5. R. A. Isaev and A. G. Podvesovskii, “Cognitive Clarity of Graph Models: an Approach to Understanding the Idea and a Way to Identify Influencing Factors Based on Visual Analysis,” *Scientific Visualization*, vol 14, no 4, pp 38–51, 2022. DOI: 10.26583/sv.14.4.04.

6. S. R. Midway, “Principles of Effective Data Visualization,” *Patterns*, vol 1, no 9, article 100141, 2020. DOI: 10.1016/j.patter.2020.100141.

7. S. R. Midway, J. R. Brum, and M. Robertson, “Show and tell: approaches for effective figures,” *Limnology and Oceanography Letters* (*L&O Letters*), vol 8, no 2, pp 213–219, 2023. DOI: 10.1002/lol2.10288.

8. E. Oral, R. Chawla, M. Wijkstra, N. Mahyar, and E. Dimara, “From Information to Choice: A Critical Inquiry Into Visualization Tools for Decision Making,” *IEEE Transactions on Visualization and Computer Graphics*, vol 30, no 1, pp 359–369, 2024. DOI: 10.1109/TVCG.2023.3326593.

9. X. Qin, Y. Luo, N. Tang, and G. Li, “Making Data Visualization More Efficient and Effective: a Survey,” *The VLDB Journal*, vol 29, no 1, pp 93–117, 2019. DOI: 10.1007/s00778-019-00588-3.

10. R. Morgan, G. Grossmann, M. Schrefl, M. Stumptner, and T. Payne, “VizDSL: A Visual DSL for Interactive Information Visualization,” *Proc of the 30th Int Conf “Advanced Information Systems Engineering”* (*CAiSE* 2018), 2018, pp 440–455. DOI: 10.1007/978-3-319-91563-0_27.

11. M. T. Cepero García and L. G. Montané-Jiménez, “Visualization to Support Decision-Making in Cities: Advances, Technology, Challenges, and Opportunities,” *Proc of the 8th Int Conf in Software Engineering Research and Innovation* (*CONISOFT*), 2020, pp 198–207. DOI: 10.1109/CONISOFT50191.2020.00037.

12. H. M. Shakeel, S. Iram, H. Al-Aqrabi, T. Alsboui, and R. Hill, “A Comprehensive State-of-the-Art Survey on Data Visualization Tools: Research Developments, Challenges and Future Domain Specific Visualization Framework,” *IEEE Access*, vol 10, pp 96581–96601, 2022. DOI: 10.1109/ACCESS.2022.3205115.

13. V. S. Zayakin, L. N. Lyadova, V. V. Lanin, E. B. Zamyatina, and E. A. Rabchevskiy, “An Ontology-Driven Approach to the Analytical Platform Development for Data-Intensive Domains,” *Knowledge Discovery, Knowledge Engineering and Knowledge Management. IC3K* 2021. *Communications in Computer and Information Science*, vol 1718, pp 129–149, 2023. Springer, Cham. DOI: 10.1007/978-3-031-35924-8_8.

14. V. Zayakin, L. Lyadova, and E. Rabchevskiy, “Design Patterns for a Knowledge-Driven Analytical Platform,” *Proceedings of the Institute for System Programming of the RAS* (*Proceedings of ISP RAS*), vol 34, no 2, pp 43–56, 2022. DOI: 10.15514/ISPRAS-2022-34(2)-4.

15. K. Smeltzer, M. Erwig, and R. Metoyer, “A transformational Approach to Data Visualization,”Proc of the International Conference on Generative Programming: Concepts and Experiences (GPCE 2014). 2014. P. 53–62. DOI: 10.1145/2658761.2658769.

16. Smeltzer K., Erwig M. A Domain-Specific Language for Exploratory Data Visualization // *Proc of the 17th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences* (GPCE 2018), 2018, pp 1–13. DOI: 10.1145/3278122.3278138.

17. C. Ledur, D. Griebler, I. Manssour, and L. G. Fernandes, “A High-Level DSL for Geospatial Visualizations with Multi-core Parallelism Support,” *Proc of the IEEE 41st Annual Computer Software and Applications Conference* (*COMPSAC*). 2017, pp 298–304. DOI: 10.1109/COMPSAC.2017.18.

18. L. Ladenberger, I. Dobrikov, and M. Leuschel, “An Approach for Creating Domain Specific Visualisations of CSP Models,” *Software Engineering and Formal Methods. SEFM* 2014. *Lecture Notes in Computer Science*(), vol 8938, pp 20–35. Springer, Cham. DOI: 10.1007/978-3-319-15201-1_2.

19. A. D. Dzheiranian, I. D. Ermakov, K. A. Proskuryakov, and L. N. Lyadova, “Designing Data Visualization System Based on Language-Oriented Approach,”*Proceedings of the Institute for System Programming of the RAS* (*Proceedings of ISP RAS*), vol 36, no 2, pp 127–140, 2024. DOI: 10.15514/ISPRAS-2024-36(2)-10.

20. L. Lyadova, A. Sukhov, and M. Nureev, “An Ontology-Based Approach to the Domain Specific Languages Design,” *Proc of the 15th IEEE Int Conf on Application of Information*

and Communication Technologies (*AICT*2021), 2021, 6 pp. DOI: 10.1109/AICT52784.2021.9620493.

21. G. Kulagin, I. Ermakov, and L. Lyadova, "Ontology-Based Development of Domain-Specific Languages via Customizing Base Language," *Proc of the 16th IEEE Int Conf on Application of Information and Communication Technologies* (*AICT*2022), 2022, 6 pp. DOI: 10.1109/AICT55583.2022.10013619.

22. L. Lyadova, I. Ermakov, V. Lanin, and K. Proskuryakov, "Approach to the Development of Ontology-Driven Language Toolkits Based on Metamodeling," *Proc of the IEEE 17th International Conference on Application of Information and Communication Technologies* (*AICT*2023), 2023, 6 pp. DOI: 10.1109/AICT59525.2023.10313152.

23. N. Matta, J. L. Ermine, G. Aubertin, and J. Y. Trivin, "Knowledge Capitalization with a Knowledge Engineering Approach: The Mask Method," *Knowledge Management and Organizational Memories*, 2002, pp 17–28. Springer, Boston, MA. DOI: 10.1007/978-1-4615-0947-9_2.

24. G. Zelazny, *The Say It with Charts Complete Toolkit*. New York: McGraw-Hill Professional, 2006. 312 pp.

25. A. Kirk, *Data Visualization: A Successful Design Process*. Birmingham: Packt Publishing Ltd, 2012. 189 pp.

26. N. Kahani, M. Bagherzadeh, and J. Cordy, "Survey and Classification of Model Transformation Tools," *Software & Systems Modeling*, vol 18, pp 2361–2397, 2019. DOI: 10.1007/s10270-018-0665-6.

27. J. Ding, J. Lu, G. Wang, J. Ma, D. Kiritsis, and Y. Yan, "Code Generation Approach Supporting Complex System Modeling Based on Graph Pattern Matching," *IFAC-PapersOnLine*, vol 55, no 10, pp 3004–3009, 2022. DOI: 10.1016/j.ifacol.2022.10.189.

28. A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer, "Vega-Lite: A Grammar of Interactive Graphics," *IEEE Transactions on Visualization and Computer Graphics*, vol 23, no 1, pp 341–350, 2016. DOI: 10.1109/TVCG.2016.2599030.