Technology for Prototyping Expert Systems Based on Transformations (PESoT): a Method

Aleksandr Yurin[0000-0001-9089-5730]

Matrosov Institute for System Dynamics and Control Theory, Siberian Branch of Russian Academy of Sciences, Lermontov St. 134, Irkutsk, Russia
iskander@icc.ru

Abstract. The development of intelligent systems and their software components (modules) continues to be a complex and time-consuming task. One of the ways to improve the efficiency of this process is to use the principles of generative and visual programming, as well as the model transformations in the context of conceptualization, formalization, and automatic codification of knowledge. This work is devoted to the description of one of the elements of the information technology for Prototyping Experts Systems and these components based on Transformations (namely, PESoT), in particular, to the method. This method implements and redefines the basic principles of the model-driven approach: models and the development process in the form of a chain of sequential transformations of models with more abstraction to ones with less abstraction and obtaining interpreted program codes and specifications at the end of this chain. Domain conceptual models and specific spreadsheets are used as source (computational-independent) models. The method is designed for non-programming users and reduces the time for creating prototypes of intelligent modules and expert systems by automating the codification stage and using existing domain models.

Keywords: Information Technology, Prototyping, Experts System, Method, Model Transformation, Code Generation

1 Introduction

The creation of intelligent systems and their software components (modules) continues to be a difficult and time-consuming task. The solving of this task requires the development of programming theory, as well as the creation of special means.

Researches in this area have been conducted since the advent of intelligent systems and at this moment they can be divided into three main groups:

1. Approaches and software for ontological and cognitive modeling (Protégé, FreeMind, Xebece, TheBrain, XMind, IBM Rational Rose, StarUML, etc.), that oriented on non-programming users, and support modeling by the domain terms.

2. Classical programming tools in the form of editors, shells, and frameworks (Expert System Designer, Expert System Creator, ARITY Expert Development Package,

CxPERT, Exsys Developer, etc.), that oriented on programmers, and require an in-depth study of programming languages.

3. Integrated approaches and software platforms (AT-Technology, MOKA, CommonKADS, OSTIS, IACPaaS), that combine features of previous groups of researches.

So, we can define the following main disadvantages of existing solutions:

- narrow specialization from a viewpoint of a support of formats and programming languages for expert systems and knowledge bases;
- a low integration capability with visual modeling software;
- the complexity of the reusing previously developed conceptual (domain) models;
- high requirements for users in the field of programming.

In this connection, the involving non-programming users to the development process and automation of conceptualization, formalization, and codification are promising trends.

One of the ways to fulfill these conditions is to use the principles of generative and visual programming, as well as model transformations in the context of conceptualization, formalization and automatic codification.

These principles are implemented separately in various methods and tools. However, in most cases, existing solutions are focused on a narrow range of target platforms and have high qualification requirements for developers. The integrated use of the considered principles is implemented within a group of approaches based on model transformations. Currently, this group of approaches is known as Model-Driven Engineering (MDE) [1].

We used a standardized MDE that applies some standards in the field of software engineering, such as UML, XMI, and MOF, for the development of new information technology. Our technology, shortly PESoT [2], consists of:

- methods and tools for creating expert systems and knowledge bases (that use the logical rules formalism) based on model transformations;
- the original UML-based graphical notation for building models of logical rules – a Rule Visual Modeling Language (RVML) [3];
- the domain-specific declarative language for describing transformations – a Transformation Model Representation Language (TMRL) [4];

This work is devoted to the description of one of the elements of this technology, in particular, the method.

2 Background

2.1 Main Principles of Model-Driven Engineering

Model-Driven Engineering (MDE) or Model-Driven Development (MDD) is a software design approach that uses the information models as the major artifacts, which, in turn, can be used for obtaining other models and generating programming codes [1, 3, 5]. This approach enables programmers and non-programmers (depending on the implementation) to create software based on conceptual models.

The main MDD principles are the following:

- A model is an abstract description of a system (a process) by a formal language. As a rule, models are visualized with the aid of certain graphic notations and serialized (represented) in XML.
- A metamodel is a model of a formal language used to create models (a model of models).
- A four-layer metamodeling architecture is a concept that defines the different layers of abstraction (M0-M3), where the objects of reality are represented at the lowest level (M0), then a level of models (M1), a level of metamodels (M2) and a level of a meta-metamodel (M3).
- Model transformation is the automatic generation of a target model from a source model with the accordance of a set of transformation rules [6]. In this case, each transformation rule describes the correspondence between the elements of a source and a target metamodels.

There are many works devoted to the model transformations. At the same time, model transformations can be considered from different viewpoints [7]: by the type of results (Model-to-Model (M2M); Model-to-Text (M2T) and Text-to-Model (T2M)); by modeling languages used (endogenous, exogenous); by the abstraction level of models (vertical, horizontal); by the transformation direction (unidirectional, bidirectional). Currently, there are some ways to implement the model transformation: graph grammars (graph rewriting) (e.g., VIsual Automated model TRAnsformations (VIATRA2), Graph REwriting And Transformation (GReAT), etc.); visual design of transformation rules and category theory (e.g., Henshin); transformation standards (e.g., Query/View/Transformation); declarative and procedural programming languages; languages for transforming XML documents (e.g., eXtensible Stylesheet Language Transformations, etc.).

There are examples of successful use of MDE for the development of database applications (e.g., ECO, for Enterprise Core Objects), agent-oriented monitoring applications, decision support systems, embedded systems (software components) for the Internet, and rule-based expert systems [8].

Today, the main MDE implementations (initiatives) are the following: OMG Model Driven Architecture (MDA), Eclipse Modeling Framework (EMF), Model Integrated Computing (MIC), Microsoft Software Factories, JetBrains MPS.

MDA is the most standardized MDE that uses the following software standards: MOF, XMI, CWM, UML, and QVT.

So, we can formalize MDE [3, 9]:

$$MDE = \left\langle MOF, UML, CIM, PIM, PSM, PDM, F_{CIM\text{-to-}PIM}, F_{PIM-to-PSM}, F_{PSM-to-CODE} \right\rangle,$$

where *MOF (Meta Object Facility*) is an abstract language for describing models (a metamodel description language); *UML (Unified Modelling Language)* is a unified modeling language; *CIM (Computation Independent Model)* is a model that hides any details of the implementation and processes of software, and describes only the software and environment requirements; *PIM (Platform Independent Model)* is a model that hides details of the software implementation that depend on the platform, and contains elements that do not change when the software interacts with any platform; *PSM (Platform Specific Model)* is a model of software that taking into account im-

plementation details and processes, dependent on a specific platform; *PDM (Platform Description Model)* is a set of technical characteristics and descriptions of the technologies and interfaces that make up the platform; $F_{CIM-to-PIM} : CIM \to PIM$, $F_{PIM-to-PSM} : PIM \to PSM$, $F_{PSM-to-CODE} : PSM \to CODE$ are the rules for models transformations.

This formalization was used in the development of PESoT technology.

2.2 Main Elements of the PESoT Technology

To overcome the shortcomings described in the introduction, we developed the PESoT technology that includes methods and tools for prototyping rule-based expert systems and decision-making software components for intelligent systems.

The following tasks were solved when developing the PESoT technology:

- to analyze the modern approaches, methods, and tools for creating rule-based expert systems and knowledge-bases based on the transformations;
- to develop principles (elements of theory, algorithms, methods) for creating rule-based expert systems and knowledge-bases based on the transformations;
- to develop languages and software to support the proposed methods;
- to test the developed methods and software when solving practical and educational tasks.

The main results of the solved tasks, as well as the elements of PESoT technology, are the following:

- a method for creating rule-based expert systems and knowledge bases based on the sequential transformations;
- a method for automated creating domain models as computational-independent models based on transformation, both conceptual models (using XMI, XTM) [10] and spreadsheets of a specific structure (using CSV) [11];
- a method for automated creating software components-converters for conceptual models transformations [12];
- a UML-based graphical notation for designing rule-based models – a Rule Visual Modeling Language (RVML) [3];
- a domain-specific language for the description of transformations – a Transformation Model Representation Language (TMRL) [4];
- Personal Knowledge Base Designer (PKBD) – a tool for creating knowledge-bases and expert systems [13];
- Knowledge Base Development System (KBDS) - a tool for creating model transformation software components [14];
- The results of testing when developing software to solve problems in the field of:
  - reliability and safety of technical systems (a module for defining the causes of damage and destruction of elements of technical systems of the software complex "Expertise of ISI» [15]; knowledge bases for predicting degradation processes in petrochemistry (systems for identifying technical states of constructions) [16]; an intelligent failure analysis process scheduler [17]; a module for creating conceptual models based on the analysis of tables from reports on industrial safety expertise [18];

- detection of banned messages and clients who violate the rules for using the SMS informing service of the «SMS organizer» platform [11];
- developing a domain-specific knowledge base editor [19];
- educational tasks within the educational process on the basis of the Institute of data analysis and information technologies of the Irkutsk national research technical University (IrNITU) [3].

Next, we will consider the first result and element of the PESoT technology – the method of creating expert systems and knowledge bases based on transformations.

3 Method

The developed method is based on MDE principles and standards in the field of software engineering.

The main features of the method (its novelty) are:

- application of model transformations in the context of knowledge engineering and creation of a certain type of software: rule-based expert systems;
- qualitative redefinition of the main stages of the standardized MDE, in particular, Model Driven Architecture (MDA);
- development and use of original languages and software;
- support for certain languages of modeling and creating knowledge bases (as part of our testing).

3.1 Formalized Statement

The following formalization of the proposed method is proposed:

$$MDE^{ES} = \left\langle MOF, L^{ES}, CIM^{ES}, PIM^{ES}, PSM^{ES}, PDM^{ES}, F^{ES}_{CIM-to-PIM}, F^{ES}_{PIM-to-PSM}, F^{ES}_{PSM-to-CODE} \right\rangle$$

where L^{ES} is a set of languages and formalisms used for modeling; in our case $L^{ES} = \{UML, CM, ET, DT, CT, RVML\}$ where UML is a Unified Modelling Language; CM is a concept or mind maps formalism; ET is an event trees formalism; DT is a formalism for the representation of decision tables; CT is a formalism for the representation of caninicalised tables; $RVML$ is a Rule Visual Modeling Language;

CIM^{ES} is a computation-independent model for PESoT, in our case, it are domain models represented with the aid of L^{ES};

PIM^{ES} is a platform-independent model for PESoT, in our case this model represent logical rules in our notation RVML;

PSM^{ES} is a platform-specific model for PESoT, in our case this model takes into account the features of the programming language, we use RVML;

PDM^{ES} is a set platform description models for PESoT, in our case $PDM^{ES} = \{CLIPS, DROOLS, PHP, PKBD\}$.

$F^{ES}_{CIM-to-PIM}, F^{ES}_{PIM-to-PSM}, F^{ES}_{PSM-to-CODE}$ are the rules for model transformations.

3.2 Main Stages

The process of creating prototypes of knowledge bases and expert systems is represented by the sequence of stages described below (Fig. 1.).

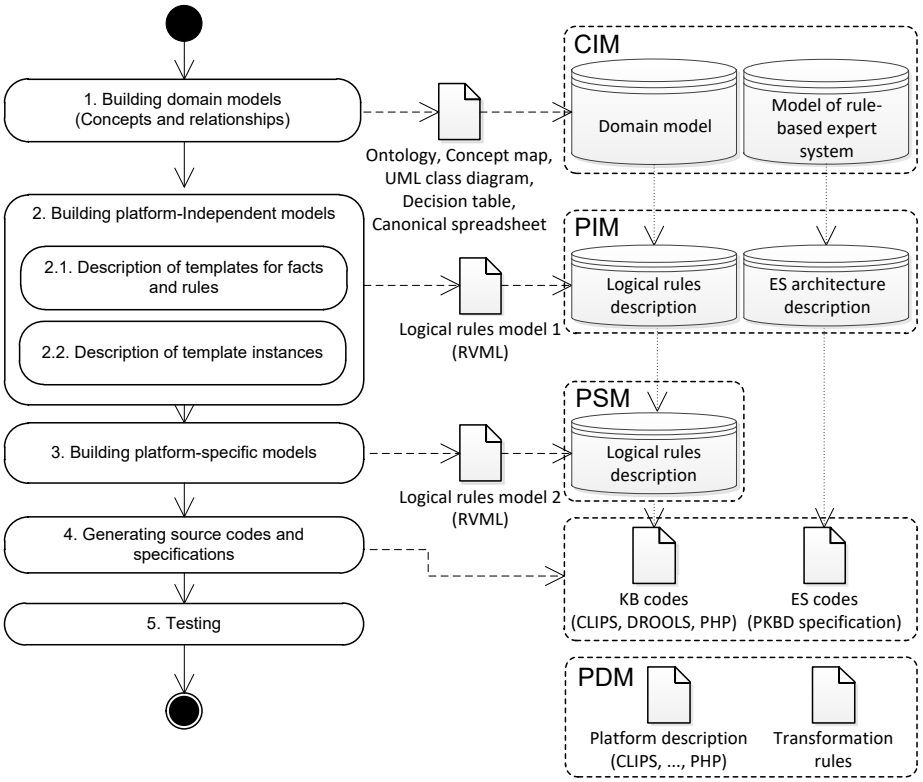

Fig. 1. Stages and models of PESoT.

Stage 1: Building domain models. The results of this stage are:

- a domain model;
- a model of a rule-based expert system.

The models acquired will be considered as a CIM and can be represented in the forms of an OWL-ontology, UML-models (in particular, as UML class diagrams), concept (mind) maps, decision, or canonical tables.

We do not concretize the ways of forming these models. On the one hand, relevant concepts emerge in a bottom-up fashion by analyzing the domain and the model specification. On the other hand, a top-down approach is followed through the analysis of relevant existing ontologies and data models. But, the efficiency of this stage can be improved by reusing the existing models created using various ontological and conceptual (cognitive) editors, CASE-tools or office suites (e.g., Protégé, OntoStudio,

CmapTools, FreeMind, TheBrain, Xmind, IBM Rational Rose Enterprise, StarUML, Microsoft Excel, etc.).

It is very important that the domain model of this stage be semantically significant, that is, it should describe any cause-and-effect relationships in a particular domain. So, in addition to the relationships of 'is-part-of' and 'is-a', the relationship 'depends-on' is introduced; this relationship describes of cause-and-effect relationships. In the case of UML class diagrams, the types of relationships are defined by the mechanism of stereotypes.

A model of a rule-based expert system is formed at the level of the requirements for the composition of its main modules. Since the type of created systems is defined earlier - this is a rule-based expert system, and then the corresponding template (which defines the main architectural elements and includes the relevant concepts) is used to form this model. The main architectural elements of the expert system are: «input form»; «output form» etc., which are derived from «graphic user interface form»; «inference engine», which is derived from «handler»; «knowledge-base». Next, we built-in this model in our tool.

Most of the software that supports the MDA approach (e.g., Bold for Delphi) does not realize this stage and only suggests to develop the software starting at the next stage. In this case, the domain model (even presented in the form of the ontology) is considered as a PIM that describes the main concepts and business logic (that is acceptable for databases). In the case of developing intelligent systems, this stage is necessary and corresponds to the stage of knowledge conceptualization and can be considered as one of the proposed modifications of the MDA/MDD.

Stage 2: Building platform-independent models. The results of this stage are:

- description of logical rules;
- a detailed description of an ES architecture.

The models of this stage resulted from the transformation of a CIM. In the process of a CIM transformation, the concepts are transformed into the fact templates and rule elements such as the conditions and actions, and the cause-and-effect relationships are transformed into logical rules. RVML is used for visualization and subsequent clarification of PIM elements.

For the implementation of this stage and subsequent stages, we used PKBD [13] with the built-in model of the expert system architecture.

Stage 3: Building platform-specific models. The number of these models is determined by the number of platforms, for which an ES is created. PSMs result from automatic transformations of PIMs by special tools with consequent modification by the end-user.

In our case, the end-user is to specify the RVML models of rules taking into account the features of a certain knowledge representation language (e.g., CLIPS) such as priorities of rules, 'by default' values of slots and a certainty factor.

Stage 4: Generating source codes and specifications. At this stage, the interpretation of RVML diagrams is performed automatically using PKBD. The results of this stage are as follows:

- a knowledge base code for a certain programming language (e.g., CLIPS);
- specifications of an expert system for the PKBD interpreter.

Source codes and specifications are created syntactically correct and they are independent of the semantic content of the models.

Stage 5: Testing. At this stage, the obtained program codes are tested in special software (in our PKBD interpreter).

Since the generation of source codes is completely automatic and syntactically correct, the end-user can test only the semantic correctness of the designed models by «running» them for different values of the initial facts. The PKBD interpreter uses the generated code and also synthesizes the elements of the user interface forms for access to the PSM model elements.

The testing criteria are the correctness of the logical inference and the correctness of its results.

It should be noted that the end-users (e.g., domain experts or analysts) are actively involved in designing a CIM and a PIM and partially a PSM. All of the model transformations and generation of program codes are implemented with specialized software that includes a PDM.

The described sequence of stages almost coincides with the MDA approach but the stage's content is redefined based on features of the rule-based expert systems (ES) engineering.

We shall further consider in detail the models presented and its metamodels.

3.3 Models and Metamodels

The description of the models and their transformations is important for the MDA approach. We represent the models in set-theoretic form.

The CIM can be presented in the form of the ontology and described as follows:

$$CIM^{ES} = \left\langle Ont^{D}, Ont^{RB\_ES} \right\rangle,$$

where Ont^{D} is the domain ontology (e.g., the reliability of technical systems); Ont^{RB_ES} is the ontology of rule-based ESs that includes the description of the main architectural elements, which are necessary for the implementation of the approach proposed.

The domain ontology Ont^{D} includes the concepts (i.e., the classes and instances of classes) and the relationships between them:

$$Ont^{D} = \left\langle BT\_D, CN, \Pr, Obj, R^{D} \right\rangle,$$

where BT_D is the list of basic data types; $BT\_D = \{\text{literal}, \text{object}, \text{collection}\}$; CN are the names of classes; Pr are the names of class properties; Obj are the concepts (constants, objects); lastly, R^{D} is the finite set of relationships between concepts. We have $R^{D} = \{R^{D}_{is-a}, R^{D}_{pr}, R^{D}_{c}\}$, where R^{D}_{isa} are the 'is-a' relationships between classes CN, R^{D}_{pr} are the relationships between classes and properties, and $pr\, R^{D}_{pr} \langle cn, cn\_bt \rangle$ where $pr \in \Pr$, $cn \in CN$, $cn\_bt \in CN \cup BT\_D$; thus, the property pr_i in the subject domain D describes class cn_j. Finally, R^{D}_{c} are the cause-and-effect relationships (the 'depends-on' relationships) between concepts.

A detailed description of this model presented in [3]. To provide correct CIM transformation we develop its metamodel with the use of MOF (Meta Object Facility) (Fig.2).

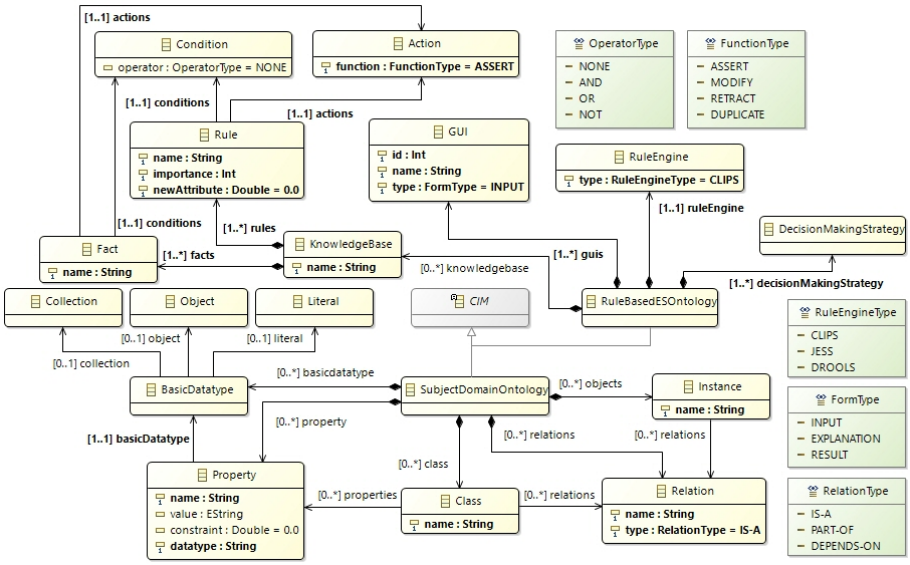

Fig. 2. A fragment of a CIM metamodel.

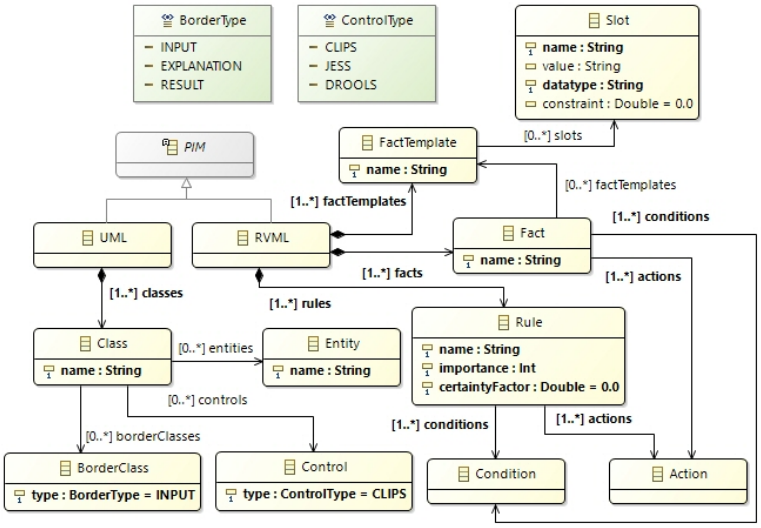

Fig. 3. A fragment of a PIM (PSM) metamodel.

The PIM is described with two models and can be represented as follows:

$$PIM^{ES} = \left\langle RVML^{RB\_KB}, UML^{RB\_ES} \right\rangle,$$

where $RVML^{RB\_KB}$ is a model of the knowledge base and UML^{RB_ES} is a model of the ES architecture. A detailed description of these models presented in [3]. The PIM metamodel corresponds to the PSM metamodel and can be represented with the aid of MOF (Fig.3).

3.4 Transformations

For our method, it is necessary to implement a sequence of exogenous horizontal transformations:

- the M2M-transformation for $T^{ES}_{CIM-to-PIM} : CIM \to PIM$;
- the M2M-transformation for $T^{ES}_{PIM-to-PSM} : PIM \to PSM$;
- the M2C-transformation for $T^{ES}_{PSM-to-Code} : PSM \to Code$.

This transformation can be represented in the form of a table (Table 1).

Table 1. A fragment of a mapping table of models' elements: CIM to PIM (PIM and PSM are equal), and PSM to Code).

CIM elements	**PIM (PSM) elements**	**Code elements**		
		CLIPS	**DROOLS**	**PHP**
Class (name, description)	Template (name, description)	deftemplate	declare	class
Property	Slot (description, value)	slot	<Property>	var
Property value	Slot value	default	"<value>"	"<value>"
Relationship	Rule (nodal element)	defrule	rule …when … then … end	if (…) {…}

The model transformation rules are implemented in the form of specifications on the imperative programming language Object Pascal for PKBD, and on TMRL and PHP for KBDS.

The implemented specifications meet the requirements of completeness, formality, and flexibility. These specifications contain all the necessary (within the approach proposed) information for solving the task, all objects of the model are well formalized, at the same time the specifications are compact enough and understandable (readable).

4 Implementation

The proposed method is implemented by the original software, namely, Personal Knowledge Base Designer and Knowledge Base Development System.

4.1 Personal Knowledge Base Designer (PKBD)

PKBD [13] is a tool for prototyping rule-based expert systems and knowledge bases. It is implemented as a desktop application designed for non-programmers.

PKBD supports RVML, and has a modular architecture that provides the ability to add modules (dynamic link libraries) that provide generation of source codes and integration with domain models designers. Currently, CLIPS, Drools, PHP, IBM Rational Rose, StarUML, XMind, CMapTools, and Microsoft Excel support DLLs are included.

Main functions of PKBD are:

- designing elements of rule bases (fact templates, facts, and rules) by non-programmers using a set of wizards and defined sources of conceptual models;
- checking the integrity of the developed knowledge bases (syntactic and semantic control);
- representing knowledge base elements using RVML;
- generating knowledge base codes in the CLIPS, Drools, and PHP formats;
- testing developed knowledge base codes (logical inference) using the integrated CLIPS rule engine;
- integrating with CASE-tools: IBM Rational Rose, StarUML, XMind, Protégé, and CMapTools, regarding import and transformation of conceptual models to highlight the main entities (concepts) and relationships for creating knowledge base drafts;
- integrating with TabbyXL [20] in terms of import and transformation of canonical spreadsheet tables to highlight the main entities (concepts) and relationships for creating knowledge base drafts;
- integrating with Microsoft Excel in terms of import and transformation of decision tables [11] to highlight the main entities (concepts) and relationships for creating knowledge base drafts.
- interacting with the KBDS service.

4.2 Knowledge Base Development System (KBDS)

KBDS [12, 14] is a tool for the development of software components (modules) that transform conceptual models presented in the XML format (the most common format for the exchange and storage of the different conceptual models) to source codes or other models. It is implemented as a web application designed for non-programmers.

KBDS supports TMRL, and has built-in modules that provide generation of source codes and integration with domain models designers. Currently, CLIPS, OWL DL, IBM Rational Rose, and CMapTools are included.

Main functions of KBDS are:

- the source code generation for CLIPS and OWL DL;
- the automated synthesis of an ontological and a rule-based model (the internal representation of knowledge in the KBDS) based on the analysis of input conceptual models;
- the use of RVML for the representation and modeling the logical rules;

- the visual representation and modeling knowledge in the form of the graph-based ontological model.

5 Testing and Discussion

The method was tested when solving educational and practical (real-world) tasks [3, 9, 15, 18, 22-24]. In particular, when solving educational tasks when creating prototypes of the simplest knowledge bases based on UML class diagrams and CMapTools concept maps, the method and software tools allowed in some cases to reduce the creation time by 60% [3].

In the case of real-world tasks, we used an indirect method to evaluate effectiveness of the PESoT method.

The indirect method of evaluation considered the main stages of expert systems engineering and the time of their implementation. At the same time, the conceptualization stage was decomposed into stages of problem identification, retrieving, and structuring knowledge, according to [24] (Table 2).

Table 2. Оценка времени разработки ЭС косвенным способов.

Stages of expert systems engineering	**Standard method [24] (weeks)**	**PESoT (weeks)**
Problem identification	1-2	0.75-1.5
Retrieving	4-12	3-8
Structuring	2-4	1.5-3
Formalizing	4-8	3-6
Programming	4-8	-
Testing	1-2	1-2
Total	16-36	9.25-20.5

Thus (Table 2), the development of expert systems by the standard method on average lasts from 4 to 9 months. When applying PESoT, the implementation stage is excluded from the development process, since automatic code generation is used, this reduces the expert systems development time by 1-2 months.

Also, when implementing the stages of retrieving, structuring, and formalizing knowledge, the knowledge engineer is excluded from the development process (or the participation time is reduced).

Based on these facts and the worst assumptions about the achievability of these effects, the execution time of these stages will be reduced by 25%, i.e. by 0.5-1.5 months. The final time reduction will be from 1.5 to 3.5 months.

It should also be noted that according to the concept of MDE, the main focus of application development is transferred from the actual programming stage to the stage

of creating the model. At the same time, by creating a model once, the developer gets the basic ability to generate source codes for other supported languages.

Therefore, an additional effect occurs when you need to transfer development to a new technology platform since you can use the old platform-independent model and develop a new only the platform-dependent one. When using the proposed method, the expert system development time will be only 1.5 to 2.5 months, since it will require repeated work only at the testing stage. Of course, this is only possible if there is software that provides code generation for the target language.

Besides, the PESoT method provides:

- reducing the risk of design errors: on a relatively simple and completely customer-based platform-independent model (as opposed to a traditional model cluttered with implementation details), it is easy to find and correct such errors;
- using cognitive graphics at the stage of knowledge retrieval;
- eliminating the programming errors due to automatic code generation.

6 Conclusions

In this work, we consider a method for creating rule-based expert systems and knowledge bases. This method is an element of PESoT technology; its features are the following: a redefined chain of model transformations, original means of implementation: language and software.

This method reduces the time for prototyping expert systems and knowledge bases by automating the codification stage and using existing domain models. Evaluation of the method is presented in [3].

The practical significance of the results is defined by their use in the educational process of IrNRTU, as well as when working with the "IrkutskNIIhimmash" joint-stock company and LLC "Centrasib".

7 Acknowledgments

The research was supported by the Program of the Fundamental Research of the Siberian Branch of the Russian Academy of Sciences, project no. IV.38.1.2 (reg. no. AAAA-A17-117032210079-1), project no. IV.38.1.3 (reg. no. AAAA-A17-117032210077-7). Results are achieved using the Centre of collective usage «Integrated information network of Irkutsk scientific educational complex».

References

1. Da Silva, A.R.: Model-driven engineering: A survey supported by the unified conceptual model. Computer Languages, Systems & Structures 43, 139–155 (2015). DOI: 10.1016/j.cl.2015.06.001

2. Dorodnykh, N.O., Yurin, A.Yu.: Technology for creating rule-based expert systems based on model transformations, Novosibirsk, SB RAS (2019). (in Russian) DOI: 10.15372/TECHNOLOGY2019DNO
3. Grishenko, M.A. Dorodnykh, N.O., Nikolaychuk, O.A., Yurin, A.Yu.: Designing rule-based expert systems with the aid of the model-driven development approach. Expert Systems 35(5), 1–23 (2018). DOI: 10.1111/exsy.12291
4. Dorodnykh, N.O., Yurin, A.Yu.: A domain-specific language for transformation models. CEUR Workshop Proceedings (ITAMS-2018) 2221, 70–75 (2018).
5. Cretu, L.G. Florin, D.: Model-Driven Engineering of Information Systems: Principles, Techniques, and Practice. Apple Academic Press (2014).
6. Kleppe, A., Warmer, J., Bast, W.: MDA Explained: The Model Driven Architecture: Practice and Promise, 1st ed., Addison-Wesley (2003).
7. Mens, T., Gorp, P.V.: A Taxonomy of Model Transformations. Electronic Notes in Theoretical Computer Science 152, 125-142 (2006).
8. Nofal, M.A., Fouad, K.M.: Developing web-based Semantic and fuzzy expert systems using proposed tool. International Journal of Computer Applications 112(7), 38-45 (2015).
9. Dorodnykh, N.O., Yurin, A.Y., Stolbov, A.B.: Ontology Driven Development of Rule-Based Expert Systems. Proceedings of the 3rd Russian-Pacific Conference on Computer Technology and Applications (RPC), 1-6 (2018). DOI: 10.1109/RPC.2018.8482174
10. Dorodnykh, N.O., Yurin, A.Yu.: Towards Ontology Engineering Based on Transformation of Conceptual Models and Spreadsheet Data: A Case Study. Advances in Intelligent Systems and Computing. Computational Methods in Systems and Software (CoMeSySo 2019) 1046, 233-247 (2019). DOI: 10.1007/978-3-030-30329-7_22
11. Dorodnykh, N.O., Yurin, A.Yu.: Development of Software Decision-Making Modules Based on a Model-Driven Approach. CEUR Workshop Proceedings. Russian Advances in Artificial Intelligence: selected contributions to the Russian Conference on Artificial intelligence (RCAI 2020) 2648, 265-279 (2020).
12. Bychkov, I.V., Dorodnykh, N.O., Yurin, A.Yu.: Approach for the development of software components for the creation of knowledge bases based on conceptual models. Computational technologies 21(4), 16-36 (2016). (in Russian)
13. Yurin, A.Yu., Dorodnykh, N.O.: Personal knowledge base designer: Software for expert systems prototyping. SoftwareX 11, 100411 (2020). DOI: 10.1016/j.softx.2020.100411
14. Dorodnykh, N.O.: Web-based software for automating development of knowledge bases on the basis of transformation of conceptual models. Open Semantic Technologies for Intelligent Systems 1, 145–150 (2017).
15. Berman, A.F., Nikolaichuk, O.A., Yurin, A.Yu., Kuznetsov, K.A.: Support of Decision-Making Based on a Production Approach in the Performance of an Industrial Safety Review. Chemical and Petroleum Engineering 50(1-2), 730–738 (2015). DOI: 10.1007/s10556-015-9970-x
16. Nikolaychuk, O.A., Yurin, A.Y.: Computer-aided identification of mechanical system's technical state with the aid of case-based reasoning. Expert Systems with Applications 34, 635-642 (2008). DOI: 10.1016/j.eswa.2006.10.001
17. Berman, A.F., Nikolaychuk, O.A., Yurin, A.Yu.: Intelligent planner for control of failures analysis of unique mechanical systems. Expert Systems with Applications 37, 7101–7107 (2010). DOI: 10.1016/j.eswa.2010.03.005
18. Dorodnykh, N.O., Yurin, A.Yu., Shigarov, A.O.: Conceptual Model Engineering for Industrial Safety Inspection Based on Spreadsheet Data Analysis // Communications in Computer and Information Science. Modelling and Development of Intelligent Systems (MDIS 2019) 1126, 51–65 (2020). DOI: 10.1007/978-3-030-39237-6_4

19. Yurin, A.Yu., Berman, A.F., Nikolaychuk, O.A., Dorodnykh, N.O., Grishenko, M.A.: The domain-specific editor for rule-based knowledge bases. Proceedings of the 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 1130-1135 (2018). DOI: 10.23919/MIPRO.2018.8400176
20. Shigarov, A., Khristyuk, V., Mikhailov, A.: TabbyXL: Software platform for rule-based spreadsheet data extraction and transformation. SoftwareX 10, 100270 (2019). DOI: 10.1016/j.softx.2019.100270
21. Berman, A.F., Dorodnykh, N.O., Nikolaychuk, O.A., Yurin, A.Y.: Knowledge bases engineering on the basis of fault trees analysis. CEUR Workshop Proceedings. Information Technologies: Algorithms, Models, Systems (ITAMS 2018) 2221, 25-31 (2018).
22. Yurin, A.Yu., Berman, A.F., Nikolaychuk, O.A., Dorodnykh, N.O.: Knowledge Base Engineering for Industrial Safety Expertise: A Model-Driven Development Approach. Studies in Systems, Decision and Control 199, 112-124 (2019). DOI: 10.1007/978-3-030-12072-6_11
23. Yurin, A.Yu., Dorodnykh, N.O., Nikolaychuk, O.A., Grishenko, M.A.: Prototyping Rule-Based Expert Systems with the Aid of Model Transformations. Journal of Computer Science 14 (5), 680-698 (2018). DOI: 10.3844/jcssp.2018.680.698
24. Jackson P.: Introduction To Expert Systems. 3rd edition. Addison Wesley (1999).