Model Driven Architecture Based on Linked Data

Evgeny Cherkashin[1,2,5,6], Alexey Kopaygorodsky[3,6], Ljubica Kazi[4], Alexey Shigarov[1,2], Vyacheslav Paramonov[2,5] and Boris Shevchenko[6]

[1]Irkutsk Scientific Center of SB RAS, 134 Lermontov Street, Russia, 664033
[2]Matrosov Institute for System Dynamics and Control Theory of SB RAS, 134 Lermontov Street, Russia, 664033
[3]Melentiev Energy Systems Institute of SB RAS, 130 Lermontov Street, Russia, 664033
[4]University of Novi Sad, Technical faculty "Mihajlo Pupin", Đure Đakovića bb, Zrenjanin, Serbia, 23000
[5]Irkutsk State University, 20 Gagarina Avenue, Russia, 664002
[6]National Research Irkutsk State Technical University, 83 Lermontov Street, Russia, 664074
E-mail:eugeneai@icc.ru,digger@istu.edu,ljubica.kazi@gmail.com

***Abstract*—We consider tools for developing information systems with use of Model Driven Architecture (MDA) and Linked Open Data technologies (LOD). The original idea of LOD is to allow the software designers to develop program systems integrated by means of common ontologies and web protocols. MDA Platform Independent Model (PIM) is expressed as set of UML diagrams (Class Diagram, State Diagram, *etc.*). PIM forms a LOD graph and its namespace. All the PIM entities (MOF structures) are defined as ontology resources, *i.e.* with URI references to LOD terms. This allows us to translate PIM UML model to a set of triples and store them in an ontology warehouse for further transformation into a Platform Specific Model (PSM). The ClioPatria ontology server and the SWI Prolog language are used as tools of PIM and PSM storage, querying and processing.**

The tools will allow us to mediate the MDA static means of code generation and configuration at development stage with the techniques of flexible data structure processing at run time, thus, producing even more productive information system development and maintenance techniques. This research corresponds to nowadays direction of Semantic Web Software Engineering [1].

***Keywords*—model driven architecture; linked open data; logic programming; knowledge–based systems**

I. Introduction

Model Driven Architecture (MDA) is a software development methodology based on transformations of models of the software under development. The input models are to be more abstract than generated output ones, as well as main aim of MDA is technological support of special cases of software development techniques.

Linked Open Data (LOD) [2] technology has been suggested by W3C consortium to represent the semantic information in the published web content in a way that provides not only the possibility of its processing with software agents (Semantic Web), but also to link all available information into a single semantic graph using relations and global universal identifiers (URIs) of resources. The descriptive capabilities of semantic web technologies, HTML5 document publishing tools, and LOD technologies form an infrastructural basis for, *e.g.*, authoring and publishing documents [3]. The LOD provide a logical markup for the information presented in a document, informative basis for the different variants of visual representation and interpretation, logical connections with other documents, export information into other documents, procedural processing, *etc.* An important advantage of LOD usage in information environments is weakening of the requirements to the information warehouses: the document itself is a formalized data warehouse [4]. In some extent, this allows reallocation of the time spent on designing the database structure for storage of partially formalized documents to the process of solving a domain problem: the user (developer) markups the document text data with semantic meaning.

From the software designer point of view, LOD technologies allow development of program systems represent common and frequently used structures and data via common ontologies and web protocols [4]. An application of semantic models at run time [5] allowed software designers to partially cope with problem of requirements and data structure evolution in lifespan of information systems. A part of system configuration (static or dynamic) is build on domain ontology model interpretation, thus modifying information system behavior, resulting in a wider flexibility of system functioning. LOD data can be used for description of various contexts and aspects (see, Aspect-Oriented Programming) interface elements, their behavior, data security, transactions, and so on.

The present research deals with developing software instrumental tools for general design of information systems using both Model Driven Architecture (MDA) and LOD [6]. MDA Platform Independent Model (PIM) is expressed as a set of UML diagrams (Class Diagram, State Diagram, *etc.*). PIM forms a graph and a namespace of LOD space. All the PIM elements are defined as ontology resources, *i.e.* with URI references to LOD terms. This allows us to translate PIM UML model to set of triples and store them in an ontology warehouse for further transformation into a Platform Specific Model (PSM). The stage of PIM design is very important as it in principle provides an evolution of the information system implementation within the development of the programming technologies in the future. The ClioPatria ontology server and the SWI Prolog language [7] are used as tools of PIM and PSM storage, querying and processing.

We distinct four popular approaches of implementing PIM to PSM transformation, taking into account the runtime and other implementation platform features. The first approach is

model-to-model transformation with a model transformation language, such as QVT-X, ATL, *etc.* In this case PIM and resulting PSM are represented as XMI files. Model transformation language rules recognizes structures in PIM and construct structures of PSM, having account various Boolean conditions calculated with helper rules. The second variant of transformation is realized with Domain Specific Language (DSL) infrastructure [8]. The transformation procedure is presented as translation of one DSL instance, *i.e.* PIM, to an instance in another language.

The third approach is a library-based one, it allows deigning the resulting PSM and the source code easy enough. Transformation functions from a library are applied to predicates or complex structures as subtasks. Usage of interpreted or ahead-of-time-compilation languages as Python, Perl, PHP, ASP.NET, JavaScript, HTML as a target language for PSMs decreases the development time and make it less complex. Thus, application of the libraries is the direct interaction of PIM as a superposition of applied function. The fourth approach is the multistage transformation of the source PIM [9] on the base of logical inference. The transformation is declared as a logical theory and a scenario of subgoals to be achieved.

The simplest way of implementation is to use Prolog or a functional language, and we use Logtalk, a macro package of the Prolog logical language. The main advantage of the language usage is knowledge structuring thanks to the object-orientation. Objects are used for representation of the source PIM model and the resulting PIM structures as well as source generators. Moreover, the transformation objects is easily integrated with the libraries processing LOD and LOD warehouses.

The aim of this research is to construct tools for developing information systems with joint of MDA and LOD technologies in an indivisible cooperation.

II. Transformation software architecture

The architecture of the system is presented in Fig. 1. The PIM is designed with an UML editor. We use Modelio [10] as it contains means for formal definition of structured stereotypes, and it is open-sourced. The model of an information system is converted in a RDF graph encapsulated in a Logtalk object. The set of such objects are passed to the Transformation subsystem that is represented as a hierarchy of modules. The subsystem queries server storage of ontologies and library rules stored on the server via Pengines interface module. Library rules represent query helpers, which allow server-side preprocessing of the queries in its Inference machine.

The server is being implemented on the base of ClioPatria ontology storage, which in turn is realized with SWI-Prolog programming system with some C language subroutines. The ontology database supports durable storage and transactional querying as well as text search facility. ClioPatria SPARQL queries execution subsystem operates in a federated regime out of the box. The usage of Prolog based infrastructure for triple processing significantly reduces the structural and semantic

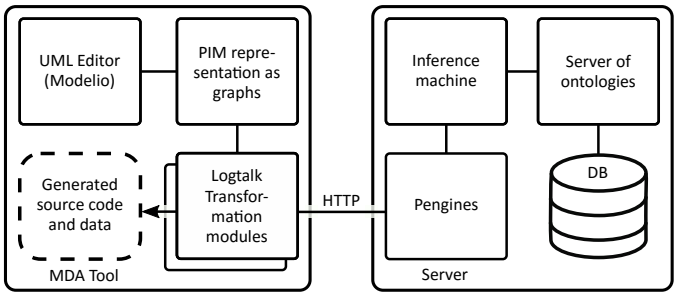

Fig. 1. A general architecture of the PIM transformation system

complexity of whole system. It supports several formats of compact storage of ontological data. The Pengines subsystem organizes a secure access to the triples from clients, it is implemented as a simple protocol on top of HTTP. There are a number of implementation libraries for the protocols for popular programming languages, including client-side Prolog, JavaScript, Java, Python.

Ontologies are loaded into ClioPatria via its web interface, or by means of startup script with a library predicates. ClioPatria has a simple interface for ontology browsing, transaction management, triple search by its literal values, and triple graph visualization for subset of triples. The system has tools for SPARQL query testing and debugging.

III. Implementation

The transformation is implemented as a Logtalk library integrated with a RDF storage. All the data processing is under control of SWI-Prolog. In our research we suppose that the input of the process is PIM, a set of logically linked UML-diagrams, ontologies and data stored in the warehouse of ontologies as well as other databases. These data sources are assembled in a PIM with a data description scenario represented as process of PSM generation out of the sources.

The output of the transformation is a set of configured interlinked Logtalk objects representing PSM. The source codes of the subsystems and the initial states of databases are generated from these object complexes.

At this stage of the research we implemented a processor of input XMI-2 XML files, which contain the UML-model of an information system, instances of Logtalk classes. The XMI files exported by Modelio UML designer consist of a model of the system as a set of top packages, and two profile packages. One profile package describes local definitions, and another represents the system and code generation definitions. The import processor converts the DOM tree of input XMI into corresponding graphs of triples. Each triple represents one relationship between two structural elements of an UML diagram. The instances have public interface used to query their graphs for the structural element compositions.

Platform model (PM) in a broad sense is the transformation procedure implemented as Logtalk components (instances) organized in an hierarchy of transformation modules (Fig. 2) in the image and likeness of our earlier work [9]. Each

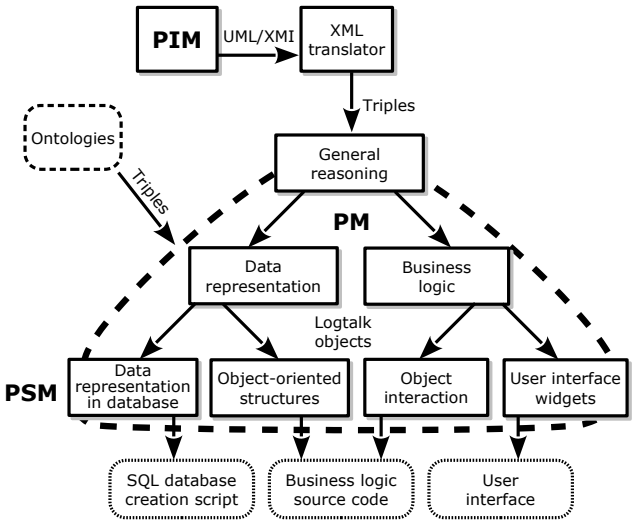

Fig. 2. A hierarchy of transformation modules representing PM

component (module) queries the sources and constructs two classes of data: own state of private data and structures of external objects. Own states, in particular, contain mappings of types between the source and target attributes, the implementation of designer decision on the general way of structure transformation. Own states, in fact, represent the module configuration of its transformation state.

External objects are structures constructing PSM, *i.e.*, the set or a category of configured objects reflecting PIM structures and target sources and data. Transformation modules generate the objects as a results of general decisions. Their configurations are also inferred from the properties of structural elements of PIM and the profile configurations. The leaf nodes of the hierarchy is the scenario of PSM generation. The usage of pure logical approach instead of the mix of two languages allowed us to reduce significantly the size and diversity of the PM description.

As soon as the scenario is fulfilled the PSM is converted into source codes and data. The conversion in [9] was made by means partial application of text templates. In this case we do not use templates and generate the source code with simple procedures. SWI-Prolong module index contains a simple template engine module simple-template intended for generation of static HTML pages. The Prolog system also contain a dictionary-like structures and predicates for dictionary data manipulation, which can be used to represent data to be filled in the templates. So open-source SWI-Prolog infrastructure is well equipped for developing MDA instruments.

A. Inference of the class structure

Target class hierarchies in nowadays information system is diversified. In web-applications, a PIM class or structure is mapped to a table or a SQL-query, an input form, an JavaScript object at user's browser, it can be a part of data transfer protocol like a JSON or XML object. Some frameworks allow

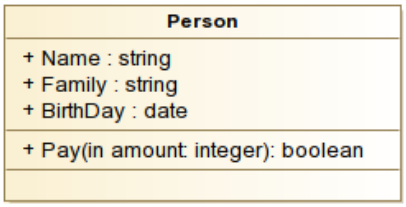

Fig. 3. A very simple Modelio PIM

one to define structures and their relations and map them to rational databases, *e.g.*, Django [8] or Entity framework. Most of the CASE-subsystems in UML composition tools do not support export of Class Diagram into ORM (Object-Relational Mapping) structures.

The main difference of MDA with respect to CASE technologies is the possibility of adaptation of PM to the special way of software development used in a particular software development group. To put it on another way, MDA implements the standard transformations, and there is tools for modification of the standard procedures to implement own special techniques. For example, if a programmer want to define a storage class for structures, the corresponding modification of standard transformation must be carried out. Similarly a whole UML class hierarchy can be immersed into an ORM by means of a programmer-defined stereotype and its implementation. Even an ORM itself may be implemented by means of special transformation.

B. Inference of the properties

One of the interesting application of the Semantic WEB technologies, namely Linked Open Data ones, is the inference of the properties of the generated structural elements. The properties of PSM elements, *e.g.*, instance attributes, are derived from PIM UML structure elements, *i.e.*, their standard UML definitions, analysis of relations being involved in, features of combinations of associated stereotypes and tag values, as well as knowledge and data obtained from local ontology warehouse and SPARQL queries to other servers (cached for a better performance), for example, to DBPedia.org.

Consider the user interface template generation for a web or desktop application. The model of the interface has a description of the structure and a specification of the properties of the constituent widgets. If we refer from PIM with tag values `rdf:object` and `rdf:type` of a special stereotype «`DBPedia`» to a corresponding DBPedia resource we can use the DBPedia subgraph for constructing `title` and `placeholder` attributes of an input field. In principle, we can figure out a label for the user locale.

Some UML-structures (OCL) and relations may be interpreted constructively as methods of objects: events, handlers, subscribers, database triggers and selectors (lookups widgets).

C. An example of simple transformation

Consider a transformation of a very simple example of PIM presented in Fig. 3. In order to import PIM we need to create holding instance `sample` and initialize it.

```
:- object(sample,instantiates(packageclass)).
:- initialization((::load_file('sample.xmi'),
                   ::register_prefixes,::process)).
:- end_object.
```

At the import stage, the source XMI converted into a graph. We show its small part in the turtle format.

```
@base <File://.../samples.xmi#> .
<#__k9i9BN_EeijsfqEI5>
  a uml:Class ;
  <#ownedAttribute> <#RN_EeijsfqEI5> ,
      <#hN_EeijsfqEI5>, <#9xN_EeijsfqEI5> ;
  <#ownedOperation> <#BN_EeijsfqEI5> ;
  rdfs:label "Person" .
<#__k9i9RN_EeijsfqEI5>
  a uml:Property ;
  <#type> <http://...#String> ;
  <#visibility> "public" ;
  rdfs:label "Name" .
<#BN_EeijsfqEI5>
  a uml:Operation ;
  <#ownedParameter> <#hN_EeijsfqEI5> ,
                    <#hN_EeijsfqEI5> ;
  <#visibility> "public" ;
  rdfs:label "Pay" .
```

Construction of the target class is realized by querying the graph with a parametric object `query(_Graph)`.

```
% ?- direct(sample,lp,cp)::tr(class,person,ID).
:- object(direct(_Package, _Local, _Code)).
tr(class, Class, ClassID):-
  ::package(Package), % get PIM
  query(Package)::class(Name, ClassID),
  ::new(Class, [instantiates(class)]),
  ::new(Attributes,[instantiates(params)]),
  ::new(Methods, [instantiates(methodlist)]),
  Class::name(Name), % get class name.
  forall( % process all attributes
      ::tr(attribute, Attribute, ClassID, _),
      Attributes::append(Attribute)),
  forall( % process all methods.
      ::tr(method, Method, ClassID,_),
      Methods::append(Method)),
  Class::attributes(Attributes), % construct
  Class::methods(Methods). % construct
:- end_object.
:- object(query(_Graph)). % querying object
class(Name, ID):- ::xmi(XMI), % the parameter
  XMI::rdf(ID, rdf:type, uml:'Class'),
  XMI::rdf(ID, rdfs:label, literal(Name)).
:- end_object.
```

Target structures constructed by blocks of elements and other blocks like in Python `llvmlite` library. New `items` are appended or prepended in a block. Rendering the sources is realized with methods of construction classes.

```
:- object(class, specializes(code_block)).
renderitem(Item, Result):- % . for each item.
 ^^renderitem(Item, Result). % call parent.
render(Result):-
 ^^render(Name),
 % .....
 root::indent, % Python block tabulation
 (  ::item(attributes(Attributes))->
    Attributes::render(DefAttrs),
    root::iswritef(ConstructorDef,
      'def __init__(self, %w):',[DefAttrs]),
    root::indent,
    Attributes::items(InstAttrs),
    findall(S, (lists::member(Attr, InstAttrs),
                Attr::item(name(AttrName)),
                root::iswritef(S, "self.%w=%w",
                          [AttrName, AttrName])),
            AttrAssigns),
    root::unindent,
    AttrList=[ConstructorDef|AttrAssigns];
    root::iswritef(ConstructorDef,
     'def __init__(self):', []),
    root::indent,
    root::iswritef(Pass,'pass', []),
    root::unindent,
    AttrList=[ConstructorDef, Pass]),
  % .....
  lists::append(AttrList, Methods, StringList),
  root::unindent,
  Result=[Signature | StringList].
:- end_object.
```

As a result we obtain the following Python code that ignores the types of the entities.

```
# ?- person::render_to(write).
class Person:
    def __init__(self, Name, Family, BirthDay):
        self.Name=Name
        self.Family=Family
        self.BirthDay=BirthDay
    def Pay(self, Amount):
        pass
```

IV. Applications

The described MDA tools are used and will be used in a number of interesting projects. The first project was to synthesize a Python interface driver of a cash register for a ticket system in a private cinema.

In the only Russian commercial planetary, a problem of connecting new cash register to a *ad hoc* ticket and billing system arose. The resources were limited by two weeks for main development and one programmer. There was an old version of a program for the previous cash machine, a documentation to the new one, as well as test utility implementing the protocol obtained from vendor driver pack. The description of the protocol in the manual was not precise: the sense of some fields were not understandable, the description of the automatons modeling the behavior in communications was too abstract, the documentation described two versions of the protocol, programmer has no previous experience with control of cash machines.

The decision of application of the MDA was made after the first investigation of the situation. The protocol commands and the automatons were represented as UML Class and State diagrams. After the application of the transformation adapted modules an interface component was generated. The component encoded API call of a command into the bytes of protocol string and decoded the answer byte string from the cash machine into the answer object. The interaction automaton of the computer and the cash device converted to a state machine with special states denoting exceptions, *i.e.* undocumented and

faulty behavior. Tree levels of the interaction has been implemented: low-level protocol for message exchange, middle-level automaton cash machine state control and computer synchronization, and high-level algorithm implementing cash machine usage in fiscal operations.

The low- and middle-level interaction were tested on dumped data of the test tool. The tests showed the protocol documentation mistakes and version variant. It also helped in PM and PIM refinement. The first two levels of interaction started functioning in two weeks. Another reserved two weeks was devoted to refinement of the fiscal operation, ticked printing design and deployment.

The second active project is a technological support of the development of New Generation Sequencing computation infrastructure for investigation of lake Baikal microbiome. The investigation is a composition of elementary bioinfomatic operations implemented by external software like Mothur and R. The generative tools are used to describe the operations as elements of a dataflow graph [11]. Each operation communicates other ones with connections of ports. The internal structures of its ports and parameters are represented with classes of UML Class Diagram. The PM adopted to generate set of stub Java classes reflecting Mothur operations of Rapidminer Studio. LOD is used to supply user descriptions in user interface for the operations.

A. Future projects

The LOD technologies will be used in development of an extension module for Electronic Medical Record (EMR) subsystem in Irkutsk Regional Oncological Dispancery. The aim of development is to incorporate into the diagnosis texts a markup that can be used in expert decision-making procedures, *e.g.*, for quality control.

A patient must be specially prepared for surgery: a set of tests must be taken, the tests must correspond to criteria, the patient must be properly hospitalized, special preparatory procedures must be carried out, *etc.* All results are filled in check lists. If patient's check list satisfies the conditions the patient is allowed to be surgically operated. After the surgery a set of new manipulations are to be carried out.

The stated development is the very place for our MDA tools and LOD technologies application. The EMR text field of the test results and diagnoses is used to store semiformalized representation of the input data for check lists. The EMRs are divided into classes (types), and we can associate the type and its text LOD markup template. The set of the attributes for each type is defined as classes of UML Class Diagram. Check lists and set of after-surgery operation are represented as classes as well. The input, check list, and new operations are joined in a special relation forming templates describing one quality control and planing module.

Thus, the usage of MDA in this task will allow incremental improvement of the quality control by means of visual construction of the templates. Using LOD allows us to develop the main EMR, and the quality control and planning subsystems absolutely independently, effectively hiding complex behavior of EMR texts from main middle-ware and database structures. It simplifies also the structure of the main database.

Another interesting project is a development of a recommender system for the Irkutsk region real estate market. The aim is to implement user means for assisting in decision making and choice refinement. The real estate objects (flats, houses, offices) have various attributes, which used to describe the objects. Each type of the objects are to be represented as classes in PIM, as well as the features of the user.

The attributes, features, and their relation constructed on the base of a domain ontology are the basis of logical processing of the database data and measuring the user interest to the object. In this case the transformation modules convert the input XMI-data into a subgraph of A-box of Irkutsk region real estate domain ontology and into interface classes describing entities of BrightstarDB object-RDF mapping database. The LOD is added as resource references to the external ontologies and individual resources. Thus, in this case the MDA tools are used as ontology modeling software.

V. Related technologies and standards

The most widely used technology of model transformation is ATL (ATLAS Transformation Language) [12] and its predecessor QVT, a OMG standard [13]. The language and its engine supports conversion from one XMI model to another one in the same format. The language structures describe recognition of properties of compositions of the source model and direct construction of new structures in the target model accounting the property values.

The ATL is supported with visual tools for rule construction. The tools are integrated in Eclipse IDE. Similar to ATL, Transformation Model Representation Language (TMRL) is developed in [14] but oriented on knowledge visual representation with following conversion them in various production rule languages (CLIPS, OWL). The visual presentation of transforming rules are used also in [15], where a general approach is adopted for the purpose. In a recent paper [16], ATL is used to transform Computational Independent Model (CIM) represented as BPMN diagram into set of UML diagrams, defining PIM. In [17], a converter of UML package structures represented in XMI was developed as a XSLT transformation.

OMG proposed ODM standard [18] for knowledge modeling. A Visual Ontology Modeler [19] is being developed for building and verifying OWL ontologies for ODM 1.0, as well as in the reverse direction, ontologies are represented as UML Class Diagrams [20].

In our approach we use the structurally simplest language, Prolog, and its infrastructure, which is capable to load various model sources, including XMI, semantic graphs, *etc.* with corresponding import module, as well as processing them *em masse*, organizing multi-stage modular transformations.

An XML based specifications are used in representing metadata about web services, WSDL [22], and the WS-BPEL language [21] for describing orchestration of web services within execution of an instance of a business process. The idea of the language is to describe complex systems down to

Web Services units, their internal and external structure and behavior, as well as services interaction in the script instance. Our project concentrates on programming in the small aspect of information system development, *i.e.* dealing with modeling and implementation software components composing the services.

VI. Conclusion

We considered an implementation and applications of indivisible usage of Model Driven Architecture and Linked Open Data, a Semantic Web, technologies in developing software on the base of model transformation. The transformation is realized as a hierarchy of objects represented in Logtalk. Results of analysis of the source model and synthesis of the target one are obtained with logical inference.

Usage of logical programming language simplifies the transformation procedure programming, the programmer is allowed to express semantics of recognized structures as properties of objects, manipulate object properties further. It also allows us to integrate the transformations with other Prolog libraries and services without usage of special language structures.

The approach is being tested in software development processes for synthesis of business-objects of information systems carcasses, interface modules for a protocol, as an ontology development tool, *etc.* Further development of the software is aimed at extending transformation modules, creating libraries of platform descriptions for open-source frameworks, develop techniques of change propagation, and developing open-source UML editors to support UML 2.5 profile specification [23] and XMI export.

Acknowledgment

The results are obtained with the partial support of the following projects:

- Irkutsk scientific center of SB RAS No 4.1.2;
- The Council for grants of the President of Russian Federation, state support of leading scientific schools of the Russian Federation (NSH-8081.2016.9);
- Russian Foundation for Basic Research No 17-07-01341.

The results obtained with the use of the network infrastructure of Telecommunication center of collective use "Integrated information-computational network of Irkutsk scientific-educational complex" (`http://net.icc.ru`). The authors are grateful to the community of Linked Open Vocabularies (`http://lov.okfn.org/dataset/lov/`) resource for assistance in the search for domain ontologies and Github.com for hosting sources at `https://github.com/isu-enterprise/icc.xmitransform`.

References

[1] Semantic WEB Software Engineering. URL:`http://www.webist.org/Documents/Previous_Invited_Speakers/2012/WEBIST2012_Pan.pdf`. Book: `https://www.iospress.nl/book/semantic-web-enabled-software-engineering/`.

[2] Ch. Bizer, T. Heath, T. Berners-Lee. "Linked Data – The Story So Far," Semantic Web and Information Systems. 2009. Vol. 5 (3). pp. 1–22.

[3] S. Capadisli, S., Guy, A., Verborgh, R., Lange, C., Auer, S., Berners-Lee, T. "Decentralised Authoring, Annotations and Notifications for a Read-Write Web with dokieli," In: Cabot J., De Virgilio R., Torlone R. (eds) Web Engineering. ICWE 2017. Lecture Notes in Computer Science, vol 10360. Springer, Cham. Preprint URL:`http://csarven.ca/dokieli-rww`, DOI:`10.1007/978-3-319-60131-1_33`.

[4] E. Cherkashin, I. Orlova. "Instrumental tools for construction of the digital archives of the documents based on Linked Data," Modern technologies, System analysis, Modeling. 4(56) 2017 pp. 100-107 (in Russian) `DOI: 10.26731/1813-9108.2017.4(56).100-107`, URL:`http://stsam.irgups.ru/sites/default/files/articles_pdf_files/100-107.pdf`

[5] A. Kopaygorodsky. "Use of ontologies in semantic information systems," Ontology of Design, 4(14), 2014, pp. 78–89 (in Russian) URL:`http://agora.guru.ru/scientific_journal/files/Ontology_Of_Designing_4_2014_opt1.pdf`

[6] D. Frankel. Model Driven Architecture: Applying MDA to Enterprise Computing. Wiley; 1 edition, 2003, 352 p.

[7] J. Wielemaker, W. Beek, M. Hildebrand, J. Ossenbruggen. "ClioPatria: A SWI-Prolog Infrastructure for the Semantic Web," Semantic Web, vol. 7, no. 5, 2016, pp. 529-541. DOI:`10.3233/SW-150191`

[8] D. Annenkov, E. Cherkashin, "Generation Technique for Django MVC Web Framework Using the Stratego Transformation Language," Proc. of 36th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), May 20–24, 2013, Opatija, Croatia, 2013, pp. 1084–1087.

[9] E. Cherkashin, A. Larionov et al., "Logical programming and data mining as engine for MDA model transformation implementation," 36th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), May 20–24, 2013, Opatija, Croatia, pp. 1029–1036.

[10] Modelio Open Source – UML and BPMN free modeling tool. URL:`https://www.modelio.org/`.

[11] Johnston W.M., Hanna J.R.P., Millar R.J. "Advances in Dataflow Programming Languages," ACM Computing Surveys. – 2004. – Vol. 36. – pp. 1–34.

[12] Jouault, F., Allilaire, F., Bezivin, J., Kurtev, I.: "Atl: A model transformation tool." Sci. Comput. Program. 72(1–2), pp. 31–39 (2008)

[13] The MOF Query/View/Transformation Specification Version 1.1. URL:`http://www.omg.org/spec/QVT/1.1`

[14] A. Berman, M. Grishchenko, N. Dorodnykh, O. Nikolaychuk, A. Yurin, "A model-driven approach and a tool to support creation of rule-based expert systems for industrial safety expertise," Proc. of the 12 th International Forum on Knowledge Asset Dynamics (IFKAD-2017) – Russia, St. Petersburg : Graduate School of 16 Management of St. Petersburg University. 2017. P. 2034–2050.

[15] A. Belghiat, M. Bourahla. "UML Class Diagrams to OWL Ontologies: A Graph Transformation based Approach," International Journal of Computer Applications. 41. pp. 41–46. 10.5120/5525-7566.

[16] Y. Rhazali, Y. Hadi, A. Mouloudi. "Model Transformation with ATL into MDA from CIM to PIM Structured through MVC," Procedia Computer Science 83 (2016) 10961101. URL:`https://doi.org/10.1016/j.procs.2016.04.229`

[17] UMLtoOWL: Converter from UML to OWL. URL: `http://www.sfu.ca/~dgasevic/projects/UMLtoOWL/`.

[18] ODM UML profile for OWL. URL:`http://www.omg.org/spec/ODM/1.0/PDF/`.

[19] OMG Ontology Domain Modeling example. URL:`https://thematix.com/tools/vom/`.

[20] OWL UML Visualizer. URL:`http://owlgred.lumii.lv/`.

[21] Business Process Execution Language – Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Business_Process_Execution_Language`

[22] Web Services Description Language – Wikipedia, the free encyclopedia. URL:`https://en.wikipedia.org/wiki/Web_Services_Description_Language`

[23] Unified Modeling Language, ver. 2.5 standard description. URL:`http://www.omg.org/spec/UML/2.5/PDF` 2004. – Vol. 36. – P. 1–34.