ISSN 0361-7688, Programming and Computer Software, 2015, Vol. 41, No. 4, pp. 208–214. © Pleiades Publishing, Ltd., 2015.
Original Russian Text © D.V. Koznov, E.V. Larchik, A.N. Terekhov, 2015, published in Programmirovanie, 2015, Vol. 41, No. 4.

# View to View Transformations in Domain Specific Modeling

**D. V. Koznov, E. V. Larchik, and A. N. Terekhov**
*St. Petersburg State University, Universitetskii pr. 28, St. Petersburg, 198504 Russia*
*E-mail: d.koznov@spbu.ru*

Received March 17, 2015

**Abstract**—Model-based software development tools should provide user-friendly navigation services to allow users browsing (show/hide and zoom in/zoom out) various model details in diagrams. This functionality is crucial in development of large-scale models. In this paper, we present an approach for formal definition of navigation services based on transformations. We introduce view-to-view transformations applying ATL, and present formal definition of navigation service. We also present a prototype implementing the approach. The prototype adds facilities of navigation servicers design to GMF and uses KIELLER toolset for layout of transformation results. A graphical editor for UML class diagrams with 12 navigation services, which have been specified using the approach, is presented.

**DOI:** 10.1134/S0361768815040039

## 1. INTRODUCTION

Model-Based Software Development (MBD) is an approach of software construction creating models that are more close to concepts of a certain domain than abstractions of algorithms, data structures, and programs [1].

Domain-Specific Modeling (DSM) is a branch of MBD, and offers tools and methods for modeling languages/tools development to meet requirements of particular domains. As a rule, domain means a software company or even one project. Thus, universality is sacrificed for accuracy of modeling abstractions, which, in turn, results in the increase in quality of target code generators, as well as other services for automatic processing of models [2]. The approach is quite popular now due to considerable progress in software toolsets for development of DSM solutions (the so-called DSM platforms [4]): MetaEdit+ [5], Microsoft Visual and Modeling SDK [6], EMP [7, 8], Microsoft Visio [9, 10], QReal [11, 12], etc. (see surveys in [3, 13, 14]). In fact, MBD is the next step after UML. It should be noted that the DSM approach is beginning to apply in other domains, particularly, in business informatics [15, 16].

Modern software toolsets for working with visual models should comprise user-friendly *navigation services*, which allow the user to browse (show/hide, zoom in/zoom out) various model details in diagrams. Actually, the purpose of such services is to provide the user with special “windows” through which she/he can observe the model; moreover, the user should be free to choose what she/he wants to see through these windows. For example, while working with the model of UML classes, the user can show or hide attributes and methods of the selected class, show the whole inheritance hierarchy for this class, show all classes of the model that are associated with the given class. This is especially important when working with large-scale models: for instance, the UML metamodel comprises about 250 classes and, being shown in one diagram, becomes completely unreadable.

In MBD, the transformation approach is being intensively developed. Transformation is conversion of one or several source models into a set of target models according to a certain set of rules [17]. There are model-to-model (m2m) and model-to-code (m2c) transformations [18], transformations of databases [19], etc. To define a transformation, special formal languages are used (for example, QVT [20] and ATL [21]). Surveys on this topic can be found in [22, 23].

However, transformations were not used for development of navigation services until now. In this case, as an area for the transformation, we can use the dynamic view of the model presented in a diagram; thus, the transformation converts one dynamic view to another. If a layout algorithm is applied to the result of such a transformation, we will actuality have a navigation service. Thus, specifications of such transformations, with other top-level models, can be as an input for the generator of a graphical editor for a domain-specific language.

The idea stated above was previously introduced in [24]. In this paper, we present formal definitions of the dynamic view, view to view transformation (v2v transformation), and navigation service. The v2v transformation is defined by means of the Atlas Transformation Language (ATL) [21]. To evaluate the proposed approach a prototype toolset is developed. The toolset is integrated with Eclipse GMF and uses KIELLER [25] for automatic layout of transformation results.

## 2. CONTEXT

### *2.1. Transformations in MBD*

Transformations are used in MBD for model generation, model synchronization, model refactoring; they also can be useful for supporting integrity constraints imposed on the model and for construction of various system views on request.

The transformations for generation of models are used when the source and target models describe a system on different levels of abstraction (for example, generation of executable models according to design models). To define these transformations, graph grammars and graph transformations are used [30, 31]. In [45], for this purpose, the EXPRESS declarative language is employed. This type of transformation is also used for model-based generation of the target code (see [18]), which is important for using models as programming languages [41].

The need for automatic synchronization of models arises when the users concurrently work with several interconnected models within one project. In [32], a framework is presented that implements (by means of transformations) a mechanism of incremental distribution of changes in one model to others.

Refactoring is a process of changing a certain artifact in order to make its structure more usable and understandable, while preserving its semantics [28]. Refactoring is generally used for programming, but also for designing other artifacts: documentations [34, 35], UML models, etc. In the latter case, refactoring can be implemented by means of transformations [27, 29].

Often, a number of integrity constraints are imposed on models being constructed, which allow one to construct more accurate specifications as compared to the use of the original modeling language. Object Constraint Language (OCL) is used for that, and there are many successful studies of application of OCL in various contexts [42, 43]. The transformation approach can also be used for development of model constraints [33].

In [48], transformations are used to construct various system views on request: the user is offered a set of views, each being a presentation of the model from a certain viewpoint. In this case, the transformation is a filter that shows in the diagram only elements required for the given view. This approach is most close to ours, however, it assumes that the views are static and the user cannot change them without changing the model.

Thus, all approaches described above do not consider transformations of dynamic views of the model.

### *2.2. Eclipse Modeling Project*

This project is rather an initiative, which combines various projects around MBD based on the Eclipse platform. The Eclipse Modeling Project (EMP) involves more than ten mature projects and several dozens of prototype (immature) projects. In our study, we use two EMP projects (GMF and ATL) and also the KIELER project, which is closely related to the GMF, even though it does not officially enter the EMP.

The Graphical Modeling Framework (GMF) is designed for specification of a set of metamodels to generate automatically a new graphical editor. It should be noted that the GMF can easily be extended, since it has open source code and transparent modular architecture. This makes the GMF attractive for use in experiments with new technologies of visual modeling.

The Kiel Integrated Environment for Layout Eclipse Rich client (KIELER) is a project started in 2004 at the Kiel University (Germany). The project is focused on constructing toolsets enabling application of layout algorithms to diagram presentation of the models [25]. The target platform of the project is Eclipse. The KIELER is a set of open-source Eclipse plugins, which are integrated with basic EMP projects, including EMF and GMF [36]. In most cases, when constructing the GMF editor, the KIELER can be used without modifications, although it allows changing and extending its algorithms.

### *2.3. ATL*

The QVT is a standard for creating transformations. It is developed by the Object Management Group (OMG) [20, 38]. In practice, however, the ATL is used more frequently; the ATL is a language and a toolkit developed by OBEO company in cooperation with the French Institute for Research in Computer Science and Automation (INRIA) [21]. The ATL includes both declarative (commonly used) and imperative (for specific cases) parts. The declarative part of the ATL is based on the concept of rule. The rules can call/inherit other rules. The transformation in the ATL consists of a set of rules; by default, each rule is applied to all elements of the source model. Using a special modifier *lazy*, one can allow the rule to be executed only when being called from another rule. It is important that the transformation defined in the ATL can be executed in two—standard and refining—modes. In the first case, elements of the target model are created only by executing a certain rule; in the second case, elements for which no rules were called are copied into the target model. The refining mode is preferable if the source and target metamodels coincide, and the transformation modifies only a small part of the source model [37]. We actively used these options in our study.

The ATL toolkit allows designing and executing transformations defined in the ATL [21]. The toolkit provides syntax highlighting and a built-in debugger. This product is intensively developing and widely used in practice, since it is oriented to the format of EMF metamodels and is a part of the EMP initiative.

## 3. THE APPROACH

### *3.1. V2V Transformations*

Suppose that we have a metamodel $\mathcal{M}$ (abstract syntax), which defines a certain modeling language, and let $\mathcal{N}$ be the notation (concrete syntax) corresponding to this model. Let $T = t_1, ..., t_n$ be a set of *static views* (types of diagrams) for this modeling language. We assume also that a *metamodel of dynamic views* $\mathcal{VM}$ is specified, which defines structure of dynamic views. Note that the metamodel $\mathcal{VM}$ does not depend on the diagram types and defines a typical graph of any diagram. The metamodel $\mathcal{VM}$ is important: it is for this metamodel that the v2v transformations are defined.

Thus, $\mathcal{VM}$ defines structure of *dynamic views*; we denote such a view by $v_t$, where $t \in T$ is a type of diagram for the given view. First of all, $\mathcal{VM}$ allows one to specify selection of elements from the model (entities and associations) $M$, which corresponds to the metamodel $\mathcal{M}$. Note that this selection may contain replicas, since the same entities and associations are often shown in one diagram several times to make information in the diagram more evident. Second, to depict model elements in the diagram, it is important not only that the element must be present in the diagram but also how it is presented (i.e., which subset of its properties is presented in the selection). For example, attributes and methods of a certain class can be either shown or hidden. It is important that this property is applied to each individual class (in the general case, each particular element of the diagram). We call such properties *display options*.

Thus, the dynamic view can be formally defined as

$$v_t : \{(m, DisplayOption), m \in M\}.$$

We denote the set of all possible dynamic views of the type $t$ by $V_t$, while the set of all possible dynamic views of the notation $\mathcal{N}$ is denoted by $V_{\mathcal{N}}$. A v2v transformation is defined as

$$v2v\_tr_{t_1,t_2}^{\mathcal{M},\mathcal{N},\mathcal{V},\mathcal{M}} : V_{t_1} \times \overline{Sub} \times Params \longrightarrow V_{t_2}. \quad (1)$$

Thus, such a transformation is defined for metamodel $\mathcal{M}$, notation $\mathcal{N}$, metamodel $\mathcal{VM}_1$, and a pair of static views $t_1$ and $t_2$ of the notation $\mathcal{N}$. The v2v transformation converts the dynamic view $v_{t_1}$ (first parameter) into the dynamic view $v_{t_2}$. Note that, in the majority of cases, $t_1 = t_2$; i.e., the types of the source and resultant views coincide. As the second parameter, we use the selection $\overline{sub}$ to which the transformation is applied. This selection is organized as follows:

$$\overline{sub} = \begin{cases} \overline{m}, & \text{if } v_{t_1} = \varnothing \\ \overline{v_{t_1}}, & \text{if } v_{t_1} \neq \varnothing. \end{cases}$$

Here, $\overline{m} \in \overline{M}$, where $\overline{M}$ is the set of all possible selections from the model $M$, and $\overline{v_{t_1}} \in \overline{V_{t_1}}$, where $\overline{V_{t_1}}$ is the set of all possible selections from $V_{t_1}$. If $v_{t_1} \neq \varnothing$, then the transformation is applied directly to the subset of the model (for example, to the model element selected in the browser) rather than to the diagram. It is not correct to combine both situations applying transformation to the model selection because sometimes one model element is presented on a diagram more than once. For example, a class could be shown in a UML class diagram twice, every occurrence can have different display options, and it is meaningful to apply different v2v transformations to each occurrence. A similar situation often occurs in diagrams of states and transitions when, for example, the same state is shown several times as a result of transitions and onece as a source of transitions; this proves to be useful so as not to draw long transition lines, which reduce legibility of the diagram. Note that these state occurrences may have different display options.

Values of the transformation parameters must be set by the user of the corresponding navigation service, which includes the transformation: they can be specified directly (for example, via a dialog window) or be transmitted indirectly (for example, information about what element is currently selected by the user).

### *3.2. Navigation Services*

Denote by $d_t$ the diagram that is constructed in the graphical editor $E_{\mathcal{M},\mathcal{V},\mathcal{M}}^{T}$ and corresponds to a diagram type $t \in T$. The set of all diagrams of type $t$ is denoted by $D_t$. Each diagram depicts a certain dynamic view; the diagram differs from the dynamic view in that it contains graphical and layout information: the size and coordinates of graphical elements, fonts, colors, etc. The set of all possible diagrams for the editor $E_{\mathcal{M},\mathcal{V},\mathcal{M}}^{T}$ is denoted by $D_E$.

Let us define a function *getview* : $D_E \longrightarrow V_E$ that yields the view depicted in the diagram. In this case, the following statement holds:

$$\forall d_t \in D_t \exists! v_t \in V_t : getview(d_t) = v_t.$$

Obviously, this function is irreversible.

We can now define the navigation service as

$$navigation\_srevice_{E,t_1,t_2} ::= v2v_{tr_{t_1,t_2}}^{\mathcal{M},\mathcal{N},\mathcal{V},\mathcal{M}}, \quad layolt\_algorithm_{E,t_1,t_2}, GUIspec. \tag{2}$$

It can be seen that the navigation service includes the corresponding v2v transformation; in addition, it contains the graphical user interface specification (GUISpec), which is the place in the user's editor from which the transformation is called: the context menu of the element, toolbar, main menu, etc., and, sometimes, a dialog box for entering the transformation parameters. Moreover, the navigation service includes the layout algorithm:

$$layolt\_algorithm_{E,t_1,t_2} : D_{t_1} \times V_{t_2} \times Params \longrightarrow D_{t_2}.$$

This algorithm has the following parameters: the original diagram to which the transformation is applied, a new view obtained as a result of transformation, and the proper parameters of the algorithm. The original diagram is required to provide the target diagram with the layout that maximally close to the original one: diagram segments which were not included in the transformation input should remain unchanged. In other words, intuitive expectations of the user concerning the partial transformation of the diagram (user mental map) are taken into account as much as possible [40].

### *3.3. Specification of v2v Transformation in the ATL*

Let us consider the process of writing v2v transformations for navigation services of a UML class editor constructed using the Eclipse GMF. As a view metamodel, we use the GMF Notation metamodel [39] (below, it is referred to as the notation). The diagram is the main element of the notation; it consists of a list of children, which comprises nodes corresponding to UML classes, and a list of edges, which comprises edges describing various associations between the classes. Each node of the diagram, in turn, contains a list of children for the nodes describing operations and attributes of the corresponding class. Each node of the diagram has a unique identifier (id). Nodes corresponding to the classes, attributes, and operations differ in their type (see the field "type" in the class "Node"). In the examples given below, we use the literal CLASS_NODE for the corresponding types.

As in the case of any other transformation written in the ATL, in the beginning of the v2v transformation, we specify source and target metamodels (here, both metamodels correspond to the notation). We also use the modifier *refining* to indicate that the transformation is executed in the mode where changes in the model are allowed.

```
module v2v_transformation_example;

create OUT : Notation refining IN :
Notationl;
```

This is followed by the description of transformation rules, which modify the view. The simplest example of a v2v transformation is to show all model elements in the diagram. It contains only one rule Node2Node defined for the diagram nodes:

```
rule Node2Node {
    from old_node : Notation!Node (old_node.type = CLASS_NODE)
    to new_node : Notation!Node (
        visible <- true,
    )
}
```

This rule contains a filter on the node type and, therefore, is called only for the nodes corresponding to the UML classes of the diagram. For each such a node, this rule assigns the value *true* to the property *visible*, thus making them visible in the diagram (the point is that a file with the GMF diagram contains references to all model elements, and these references have extra, so to speak, diagram attributes, in particular, the property *visible*).

Suppose that it is required to hide a particular element of the model in the diagram. To do this, we need a more complex v2v transformation, since a certain action must be applied only to a particular element rather than to all elements in the diagram. In this case, we need a transformation parameter whose value is the id of the element to be hidden. Because the ATL do not allow us to define and execute transformations with parameters, we extend the ATL as follows. In the code of the transformation, we use a special lowercase literal, which will be assigned a particular value before compiling and executing the transformation. Thus, to hide the element, the condition concerning the node id is added to the filter of the Node2Node rule, and the property *visible* is assigned the value *false* in the code of the rule:

**Table**

| No. | Name | Description | Volume | Complexity |
|---|---|---|---|---|
| 1 | Hide class | Hide the selected class in the diagram | 1 | E |
| 2 | Show all classes | Show all previously hidden classes in the diagram | 1 | E |
| 3 | Hide properties of a class | Hide attributes and methods of the selected class | 1 | H |
| 4 | Show properties of a class | Show properties of the selected class in the diagram | 1 | H |
| 5 | Hide all properties | Hide attributes and methods of all classes | 1 | M |
| 6 | Show all properties | Show all previously hidden properties of classes | 1 | M |
| 7 | Show successors tree | Show all subclasses | 2 | M |
| 8 | Hide successors tree | Hide all subclasses | 2 | M |
| 9 | Show predecessors | Show all superclasses | 2 | H |
| 10 | Hide predecessors | Hide all superclasses | 2 | H |
| 11 | Show associated classes | Show all classes associated with the selected class | 2 | M |
| 12 | Hide associated classes | Hide all classes associated with the selected class | 2 | M |

```
  rule Node2Node {
      from  old_node  :  Notation!Node  (old_node.type  =  CLASS_NODE  and
old_node.id - 'CuRRENT_NODE_ID')
      to new_node : Notation!Node (
        visible <- true,
      )
  }
```

## 4. CASE STUDY

To evaluate the proposed approach, we developed an Eclipse plugin, which make it possible to integrate the navigation services constructed via the v2v transformations written in the ATL into the graphical editors created with the help of the Eclipse GMF. The developer that uses this plugin must specify, in addition to the transformation code in ATL, the extra information required to complete specification of navigation service: the type of elements (in terms of the GMF domain model) to which the v2v transformation is applied and the name of the navigation service. This information is used to construct the context menu items for the corresponding elements of the editor. To specify this information, we use the GUIspec parameter of the navigation service (see (2)). When the user calls the navigation service, the code of its v2v transformation is loaded; then, the code is compiled and applied to a specialized copy of the current diagram. The result of the transformation is passed to the KIELER, which performs automatic layout of the diagram elements. The layout algorithm, as well as its settings, is selected by the user in the standard options of the KIELER.

Using the Eclipse GMF extended as shown above, we constructed a graphical editor for UML class diagrams, which implements the transformations listed in the table.

We estimated these transformations according to the following criteria: (i) volume—the number of matched rules and helpers used to implement the transformation; (ii) complexity—easy (E) complexity means that only very basic knowledge of the ATL is required, medium (M) complexity means that it is required to know standard ATL constructions and to understand the metamodel on which the transformation is defined, and high (H) complexity requires strong knowledge of the ATL, including good skills in working with collections; and (iii) performance, under which we mean the speed of transformation execution for large-scale diagrams (it was found to be 2–3 s for the diagram comprising 100 classes and 90 associations). The performance of the ATL itself is very high, and the most part of time is spent on a preparatory work.

Thus, the developer who creates model editor using GMF may implement a wide range of navigation services easily. This work becomes a part of model development in the GMF (design of a new editor) and does not involve elaboration of the program code: it is required to construct only a small set of rules written in the ATL (5–20 lines per a service). The ATL itself is a part of the Eclipse Modeling Project, which simplifies its learning and application for the users who work with the GMF. At the same time, the implementation of such navigation services in the editor's code from scratch would require deep integration with the Java code of the GMF and understanding the way the GMF engine works, as well as writing a great amount of code. In this case, each navigation service would include a large number of various procedures for bypassing the model and diagrams, which one would have to write and debug manually many times (this code can hardly be reused), since it involves specific actions for each situation, although having a lot of commonalities.

## 5. CONCLUSIONS

One of the main directions in further research is the improvement of facilities for specifying the navigation services and the integration of the process for constructing the above specifications with the development process of the GMF models. The phenomenon of partial layouts is also of interest: many currently-available layout algorithms do not possess sufficient robustness; i.e., small changes in the diagram, followed by application of these algorithms, result in complete redrawing of the graph and loss of the user mental map. Finally, it is required to design transformations for large-scale diagrams, first of all, behavior models (BPMN and statecharts), large database schemas, etc. It is also required to increase transformation performance by improving the implementation of the proposed approach.

## ACKNOWLEDGMENTS

This work was supported by the Russian Foundation for Basic Research, project no. 14-01-00407.

## REFERENCES

1. Broy, M., Seamless method- and model-based software and systems engineering, *The Future of Software Engineering*, Nanz, S., Ed., Berlin: Springer, 2010, pp. 33–47.
2. Kelly, S. and Tolvanen, J.-P., *Domain-Specific Modeling: Enabling Full Code Generation,* New York: Wiley, 2008.
3. Koznov, D. and Ol'khovich, L., Visual project languages, *Sistemnoe programmirovanie* (System Programming), Terekhov, A.N. and Bulychev, D.Yu., Eds., St. Petersburg: Izd. SPbGU, 2004, pp. 148–168.
4. Pavlinov, A., Koznov, D., Peregudov, A., Bugaichenko, D., Kazakova, A., Chernyatchik, R., Fesenko, T., and Ivanov, A., About development tools of domain-specific visual languages, *Sistemnoe programmirovanie* (System Programming), Terekhov, A.N. and Bulychev, D.Yu., Eds., St. Petersburg: Izd. SPbGU, 2006, no. 2, pp. 121–147.
5. MetaCase tools, MetaEdit+. http://www.metacase.com.
6. Microsoft Visual Studio Visualization and Modeling SDK. http://msdn.microsoft.com/en-us/library/vstudio/bb126259.aspx.
7. Eclipse Modeling Project. http://www.eclipse.org/modeling.
8. Sorokin, A. and Koznov, D., Review of the Eclipse Modeling Project, *Sistemnoe programmirovanie* (System Programming), Terekhov, A.N. and Bulychev, D.Yu., Eds., St. Petersburg: Izd. SPbGU, 2010, no. 5, pp. 6–31.
9. Microsoft Visio. http://office.microsoft.com/ru-ru/visio.
10. Koznov, D.V., Ivanov, A.N., Mishkis, A.I., and Zalevskii, Ya.I., Support of conceptual modeling when developing visual languages with the use of Microsoft DSL TOOLS, *Sistemnoe programmirovanie* (System Programming), Terekhov, A.N. and Bulychev, D.Yu., Eds., St. Petersburg: Izd. SPbGU, 2009, no. 4, pp. 104–126.
11. Terekhov, A.N., Bryksin, T.A., Litvinov, Yu.V., Smirnov, K.K., Nikandrov, G.A., Ivanov, V.Yu., and Takun, E.I., Architecture of the QReal visual modeling environment, *Sistemnoe programmirovanie*, 2009, vol. 4, pp. 172–197.
12. Osechkina, M.S., Bryksin, T.A., Litvinov, Yu.V., and Kirilenko, Ya.A., Support of mouse gestures in Meta-CASE systems, *Sistemnoe programmirovanie*, 2010, vol. 5, no. 1, pp. 52–75.
13. Pavlinov, A.A., Koznov, D.V., Peregudov, A.F., Bugaichenko, D.Yu., Kazakova, A.S., Chernyatchik, R.I., and Ivanov, A.N., About development tools of domain-specific visual languages, *Sistemnoe programmirovanie*, 2006, vol. 2, no. 1, pp. 116–141.
14. Polyakov, V.A. and Bryksin, T.A., Approaches for definition of diagram interpretation semantics in the framework of the DSM approach, *Sistemnoe programmirovanie,* 2012, vol. 7, pp. 187–216.
15. Grigoriev, L. and Kudryavtsev, D., ORG-Master: Combining classifications, matrices and diagrams in the Enterprise architecture modeling tool, in *Communications in Computer and Information Science*, 2013, pp. 250–257.
16. Kudryavtsev, D. and Gavrilova, T., Diagrammatic knowledge modeling for managers: Ontology-based approach, *Proc. Int. Conf. on Knowledge Engineering and Ontology Development (KEOD)*, 2011, pp. 386–389.
17. Sendall, S. and Kozaczynski, W., Model transformation: The heart and soul of model driven software development, *IEEE Software*, 2003, pp. 42–45.
18. Koch, N., Classification of model transformation techniques used in UML-based Web engineering, *Software, IET*, 2007, vol. 1, no. 3, pp. 98–111.
19. Halpin, T.A. and Proper, H.A., Database schema transformation and optimization, *Proc. 14th Int. Conf. Object-Oriented and Entity-Relationship Modeling (OOER)*, Gold Coast, 1995, pp. 191–203.
20. MOF 2.0 Query/Views/Transformations RFP (revised), OMG Document ad/2002-04-10, Object Management Group, 2002.
21. The Eclipse Foundation, Atlas Transformation Language (ATL). http://wiki.eclipse.org/ATL.
22. Czarnecki, K. and Helsen, S., Feature-based survey of model transformation approaches, *IBM Syst. J.*, 2006, vol. 45, no. 3, pp. 621–645.
23. Biehl, M., *Literature Study on Model Transformations*, Stockholm: Royal Institute of Technology, 2010.

24. Koznov, D.V., About specification of diagram transformations in graphical editors, *Vestn. S.-Peterb. Univ.,* 2011, vol. 10, no. 3, pp. 100–111.

25. The Kiel University, KIELER Eclipse project. http://www.informatik.uni-kiel.de/rtsys/kieler.

26. Kruchten, P., The 4+1 view model of architecture, *IEEE Software,* 1995, vol. 12, no. 6, pp. 42–50.

27. Whittle, J., Transformations and software languages: automating transformations in UML, *Proc. 5th Int. Conf. on the Unified Modeling Language,* 2002.

28. Fowler, M., *Refactoring: Improving the Design of Existing Code,* Addison Wesley, 1999.

29. Egyed, A., Compositional and relational reasoning during class abstraction, *Proc. 6th Int. Conf. on the Unified Modeling Language,* 2003.

30. Agrawai, A., Karsai, G., and Shi, F., A UML-based graph transformation approach for implementing domain-specific model transformations, *Int. J. Software Systems Modeling*, 2003.

31. Sendall, S., Perroin, G., Guelfi, N., and Biberstein, O., Supporting model-to-model transformations: The VMT approach, *Workshop on Model Driven Architecture: Foundations and Applications,* Holland, 2003.

32. Egyed, A., Instant and incremental transformation of models, *Proc. 19th IEEE Int. Conf. on Automated Software Engineering (ASE)*, pp. 362–365.

33. Dong, J. and Yang, S., QVT based model transformation for design pattern evolutions, *Proc. 10th Int. Conf. on Internet and Multimedia Systems and Applications (IMSA),* 2006, pp. 16–22.

34. Koznov, D.V. and Romanovsky, K.Yu., DocLine: A method for software product lines documentation development, *Program. Comput. Software*, 2008, vol. 34, no. 4, pp. 216–224.

35. Koznov, D.V. and Romanovsky, K.Yu., Automated refactoring of software product lines documentation, *Sistemnoe programmirovanie* (System Programming), Terekhov, A.N. and Bulychev, D.Yu., Eds., St. Petersburg: Izd. SPbGU, 2009, no. 4, pp. 127–149.

36. Fuhrmann, H. and von Hanxleden, R., Taming graphical modeling, *Proc. 13th Int. Conf. on Model Driven Engineering Languages and Systems: Part I*, Oslo, 2010, pp. 196–210.

37. Jouault, F. and Kurtev, I., Transforming models with ATL, *Proc. Model Transformations in Practice Workshop (MTIP) at MoDELS Conference,* Montego Bay, 2005.

38. Jouault, F. and Kurtev, I., On the architectural alignment of ATL and QVT, *Proc. ACM Symposium on Applied Computing (SAC)*, (New York, 2006) New York: ACM, 2006, pp. 1188–1195.

39. Shatalin, A. and Tikhomirov, A., Graphical modeling framework architecture overview, *Proc. Eclipse Modeling Symposium,* 2006.

40. Sun, Y., Gray, J., Langer, P., Wimmer, M., and White, J., A WYSIWYG approach for configuring model layout using model transformations, *Proc. 10th Workshop on Domain-Specific Modeling*, Reno, 2010.

41. Fowler, M., *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, Reading: Addison-Wesley, 2003.

42. Chae, H.S., Yeom, K., and Kim, T.Y., Specifying and validating structural constraints of analysis class models using OCL, *Information Software Technology*, 2008, vol. 50, no. 5, pp. 436–448.

43. Kim, T.Y., Kim, Y.K., and?Chae, H.S., Towards improving OCL-based descriptions of software metrics, *Proc. 33rd Annual IEEE International Computer Software and Applications Conference, (COMPSAC),* 2009, pp. 172–179.

44. Bull, R.I. and Favre, J.-M., Visualization in the context of model driven engineering, *Proc. MoDELS Workshop on Model Driven Development of Advanced User Interfaces*, Montego Bay, 2005.

45. Semenov, V.A., Alekseeva, E.V., Morozov, S.V., and Tarlapan, O.A., Composite approach for constructing visualization applications, *Tr. Inst. Sistemnogo Program. Ross. Akad. Nauk*, 2004, vol. 5, pp. 175–214.

*Translated by Yu. Kornienko*