Baltic J. Modern Computing, Vol. 12 (2024), No. 1, 1-14
https://doi.org/10.22364/bjmc.2024.12.1.01

An Overview of Practical Applications of Model Transformation Language L0

Sergejs RIKAČOVS, Lelde LĀCE, Edgars RENCIS,
Agris ŠOSTAKS, Kārlis ČERĀNS

Institute of Mathematics and Computer Science, University of Latvia
Raina blvd. 29, Riga, LV-1459, Latvia

sergejs.rikacovs@lumii.lv, lelde.lace@lumii.lv,
edgars.rencis@lumii.lv, agris.šostaks@lumii.lv,
karlis.cerans@lumii.lv

ORCID 0009-0003-8989-0942, ORCID 0000-0001-7650-2355, ORCID 0000-0002-1606-4944,
ORCID 0009-0003-5987-1644, ORCID 0000-0002-0154-5294

Abstract. This paper highlights real-world applications of the low-level model transformation language L0. Specifically, it explores how L0 has been used in the following areas: implementing higher-level model transformation languages, developing the GrTP tool building platform, specifying tools within the GrTP platform and student teaching. The paper emphasizes the practical utility of L0, showcasing its usability in diverse applications.

Keywords: Model transformation language, L0, MOLA, GrTP

1. Introduction

Model transformation languages are at the core of model-driven architecture (MDA (Kleppe et al., 2003)) applications. Typically, for an end user, a high-level model transformation language (as, e.g., QVT (Web, a), ATL (Web, b; Jouault and Kurtev, 2006.), EPSILON (Kolovos et al., 2008), MOLA (Kalnins et al., 2004a; Kalnins at al., 2004b) is most convenient to describe various model management tasks. Such a language usually provides means for model fragment pattern description, identification, and transformation.

Still, there are situations when the need for lower-level model transformation languages arises. Such situations can appear e.g., in implementation of high-level languages, or in the settings, where the high-level languages are not readily available and are difficult to be built (for instance, in supporting custom model-based environments, as e.g., custom visual modeling tool building platforms (Barzdins et al., 2007). Some of lower-level model transformation language examples include ATL byte code (Jouault

and Allilaire, 2006; Web, c), ATC (Estévez et al., 2006), EOL (Kolovos et al., 2006) and L0 (Rikacovs, 2008), that is analyzed further in this paper.

The base model transformation language L0 (Barzdins et al., 2008; Rikacovs, 2008) has been developed to provide the primitive means for low-level access to model repositories, with the intention both to implement higher-level model transformation languages, and to use it directly in model transformation applications.

In this paper we demonstrate the actual applications of the L0 language both in the higher-level language implementation and direct language application areas, so proving the viability of the introduced language concept.

Our case study highlights several key areas where L0 has seen successful applications. From implementing higher-level languages by bootstrapping and using it as a main transformation language for the development of the GrTP platform to using it in student teaching process, L0 has showcased its usability and versatility.

By sharing experiential insights gained from these applications, our paper enhances the understanding of ways model transformation languages can be applied in practical scenarios.

This paper is structured as follows. Section 2 provides a brief overview of the key features of the L0 language. Each section from 3 to 6 is focusing on a specific aspect of the practical application of the L0 model transformation language. Section 3 explores how L0 has been utilized in implementing higher level model transformation languages through bootstrapping (Barzdins et al., 2008; Šostaks, 2010; Rencis, 2008). Section 4 details L0's role as the primary transformation language in the development of the GrTP platform (Bārzdiņš et al., 2007). Section 5 discusses the application of L0 in specifying tools developed on the basis of the GrTP platform. Section 6 describes the incorporation of L0 in the student teaching process within the Computer Science Master's Program at the University of Latvia. Section 7 concludes the paper, summarizing the key findings and implications derived from the practical applications discussed in the preceding sections.

2. L0 language

The L0 language is a textual low-level imperative procedural model transformation language. It contains a minimal but sufficient set of commands for processing models. Control flow is organized by using low-level control flow control commands. The access to model class instances is organized using the concept of a typed reference - at one particular moment a reference points to exactly one object (or to the special **null** value). In the program, these typed references are introduced using the **pointer** command. Association instances are accessed using two references that point to the source and target objects of the corresponding link and the name of the association role. Manipulation of attribute values typically occurs by first reading the value of the attribute into a variable of elementary type (introduced with the **var** command), then processing the value of that variable and writing it back to the object's attribute.

All action commands available in L0 can be grouped as follows:

- commands for creating and destroying objects (**addObj**, **deleteObj**)
- commands for creating/deleting links (association instances) (**addLink**, **deleteLink**)
- command for reading / setting attribute values (**setAttr**)
- commands for instance traversing (**first**, **first from**, **next**)
- low-level control flow control commands
    - labels (introduced with **label** command)
    - commands for unconditional transfer of control (**goto**, **return**, **call**)
    - commands for conditional transfer of control (**type**, **var**, **attr**, **link**, **noLink**, **pointer**)
- commands for assigning variable/pointer values (**setPointer**, **setPointerF**, **setVar**)

A more detailed description of L0 language can be find in (Rikacovs, 2008).

3. Implementation of higher-level model transformation languages

High level model transformation languages are at the core of model-driven architecture (MDA) applications (Kleppe et al., 2003). A typical construct found in a high-level model transformation language is pattern. Still, the model transformation languages are implemented on top of metamodel based data stores, as EMF (Web, f), MDR (Web, g), JR (Opmanis and Cerans, 2010). These data stores natively provide low level API, for manipulating metamodels and their instances. An important problem in implementation of high-level model transformation languages is to find a way, how to map the patterns found in a high level model transformation language into low level operations found in an interface of a typical metamodel based data store. Practice shows that it is a difficult problem (Kalnins et al., 2004c; Kalnins et al., 2006; Šostaks, 2010). One of the reasons for that is the big semantic gap between high level pattern and low-level operations, found in API of a metamodel based data store.

We demonstrate, how the language L0 has been used as a basis for an implementation of a high-level graphical model transformation language MOLA (Kalnins et al., 2004a; Kalnins at al., 2004b). Since L0 contains only the basic means for model description and manipulation, the means necessary for writing MOLA implementation have been gradually added to L0, obtaining a sequence of model transformation languages L1, L2, L3 (Barzdins et al., 2008; Sostaks and Kalnins, 2008; Rencis, 2008; Šostaks, 2010), where each following language contains more and more advanced features and compiler from L3 to L2 and L2 to L1 and L1 to L0 is built in L0 (bootstrapping process).

The language family L0, L1, L2 and L3 for MOLA implementation has the following properties:

- The first language in the family is the base language – language L0 (Rikacovs, 2008), which constructs are rather close to constructs found in the API of a typical metamodel based data store.
- Every language L_i is obtained by supplementing language L_{i-1} with new constructs.
- Language L3 already contains rather advanced facilities and its expressivity is rather close to the expressivity of a typical model transformation language.
- For every language from this family, except language L0, that is the base language, a compiler written in L0 is built: from L3 to L2, from L2 to L1, from L1 to L0.

Transformation language L1 is obtained by adding pattern definition facilities to L0 language. The pattern in language L1 can contain constraints, restricts allowed attribute values, presence of links of a specific types between objects and so on (Rencis, 2008) . To be able to use pattern definition constructs in L1 we extend the syntax of **first** and **next** commands:

```
first <pointerName1> : <className>
    [ from <pointerName2> by <roleName> ]
[ suchthat
    begin
        <L1 Commands>
    end
];
```

```
next <pointerName>
[ suchthat
    begin
        <L1Commands>
    end
];
```

Fig. 1. Extended versions of first and next command

Suchthat block is kind of a new Boolean expression – *begin-end* expression. This expression evaluates to true, iff program execution reaches **end** command, otherwise its value is false.

Transformation language L2 is obtained by adding *foreach* loop facilities to L1 language:

```
foreach <pointerName1> : <className>
[ from <pointerName2> by <roleName> ]
[ suchthat
    begin
        <L2Commands>
    end
]
```

```
do
begin
    < L2Commands>
end;
```

Fig. 2. Foreach loop in L1

L3 language is obtained by adding „*if-then-else*" construct to L2:

```
28  if
29  begin
30      <L3Commands>
31  end
32  then
33  begin
34      <L3Commands>
35  end
```

```
36  [ else
37      begin
38          <L3Commands>
39      end
40  ];
```

Fig. 3. If–then–else statement in L2

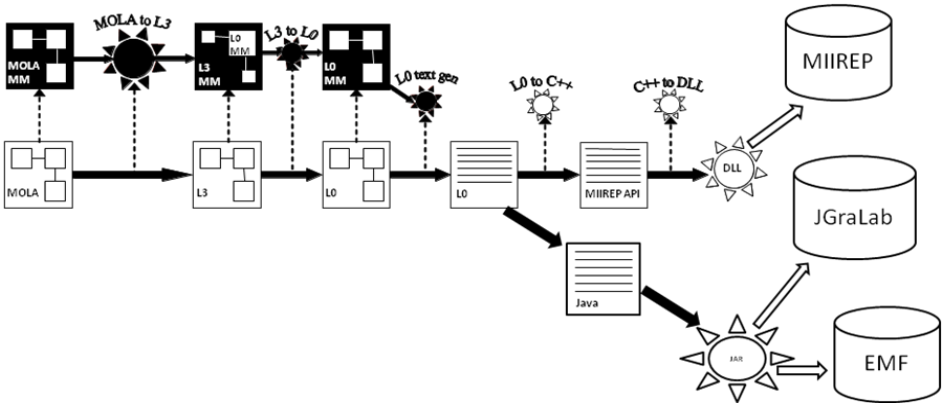

Fig. 4. MOLA- L0 compilation schema (Šostaks, 2010)

Finally, when we have language L3, we can build MOLA compiler to L3. Thanks to the fact that expressivity of MOLA and L3 are rather close it is much easier to build MOLA compiler to L3, than MOLA compiler to L0. Full compilation schema is shown in figure 4. This idea (MOLA compiler to L3) has been practically approbated in the context of A.Šostak`s thesis (Šostaks, 2010). Obtained results confirmed advantages of bootstrapping approach for implementation of model transformation languages – compiler development was possible with moderate allocation of resources and developed compiler proved to generate quite efficient (from performance point of view) code.

With this new approach to MOLA implementation based on bootstrapping process, it was possible to achieve a significant speed improvement (in some examples up to 65x order, if compared to an earlier implementation, based on SQL queries (Šostaks, 2010)). For test models of size N <=10000, which is a typical model size in the MDSD (model driven software development) context, the transformation execution time was less than

one second. Considering that the examples used in the performance tests were selected as typical for MDSD transformations, these tests confirmed that the implementation of MOLA through languages of the Lx family (that is based on language L0) has been sufficiently efficient for use in typical MDSD tasks.

4. Development of GrTP platform

Another important L0 use case is a graphical tool building platform GrTP (Bārzdiņš et al., 2007), operating in conformance with MDA principles. L0 has been used as the main transformation language in the implementation of the platform.

The main idea of the GrTP platform is strict separation of domain model processing and user interface component processing. The only permissible way to define correspondences between domain elements and user interface elements is through model transformations. An architecture of this platform can be seen in Fig.5.

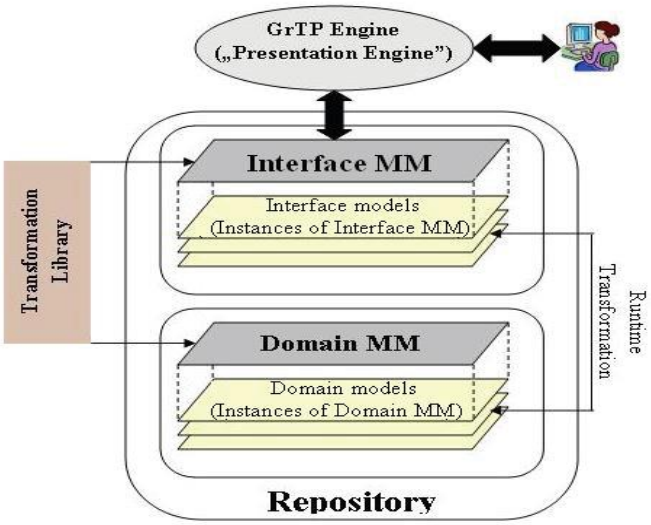

Fig. 5. The structure of GrTP (Bārzdiņš et al., 2007)

Graphical tool building platform GrTP is based on the following ideas:

- There are two special kinds of metamodels:
  - User interface metamodel. Every instance of interface metamodel consists of graphical view that represents respective instance of domain metamodel. Also interface model contains instances of classes

that represent symbol palette, its elements, and toolbar and user actions, performed on a specific user interface element.
    - Domain metamodel. In case if we are building a class diagram editor instances of domain metamodel will represent a specific UML class diagram that conforms to UML class diagram metamodel.
- A library of presentation engines is created. A presentation engine is a software component that can visualize instances of a specific metamodel and handle user actions that are specified in a metamodel.
- Transformations, defining an internal logic (how a specific tool handles user actions) of a specific tool are created. These transformations define correspondences between domain model instances and user interface model instances. In process of handling almost every user action a transformation analyzing and processing this action is called.
- GrTP platform is based on a highly efficient (from performance point of view) metamodel based in-memory data store. This data store contains interface metamodel (instances of this metamodel are interface models) and domain metamodel (instances of this metamodel are domain models).

In the context of the platform implementation the transformation performance is very important. In fact, the transformations should be executed in a very limited amount of time before the user has noticed a delay in tool operation. Another requirement that must be met by the language used to write transformations in the context of the above-mentioned platforms is that the language in which transformations are written must be easy to use - such that it is "relatively convenient" for the programmer (toolmaker) to write transformations in this language.

L0 was used as a main transformation language in the development of the GrTP platform, it fulfilled all requirements for such a language. One can note that it was not some higher-level model transformation language like MOLA used for this task, as it has turned out that the pattern recognition facilities were not that much necessary.

The usage of L0 in GrTP platform building confirms that L0 is usable in real world applications. The total size of the source code of transformations developed in this project exceeds 20`000 lines of L0.

5. Specific tools developed on the basis of GrTP platform

Several practical tools have been developed by using GrTP platform. These tools are described in the following subsections.

In the context of the GrTP platform development in general, L0 language was used to develop model transformations specifying the internal logic of the GrTP platform (it is to implement the core GrTP platform functionality). In the context of the development of specific tools on the basis of the GrTP platform, L0 was used to specify how a tool reacts to user events and how it maintains domain and presentation models in a synchronized state.

5.1. GradeTwo

GradeTwo (Web, d) is a graphical UML modelling tool build on the basis of the metamodel-based model transformation-driven graphical tool building platform GrTP. GrTP incorporates two main features of good software – it is very easy to use (new domain specific tools can be generated in hours not days) while it still provides a very big expressiveness (almost every feature you can imagine you can also integrate in the tool being created). GradeTwo tool is used in several courses, including System Modeling and Knowledge Engineering of the master`s program at the Faculty of Computer Science of the University of Latvia. The GradeTwo tool stands well among the other UML editors even in 2020ies by offering convenient means for visual diagram arrangement.

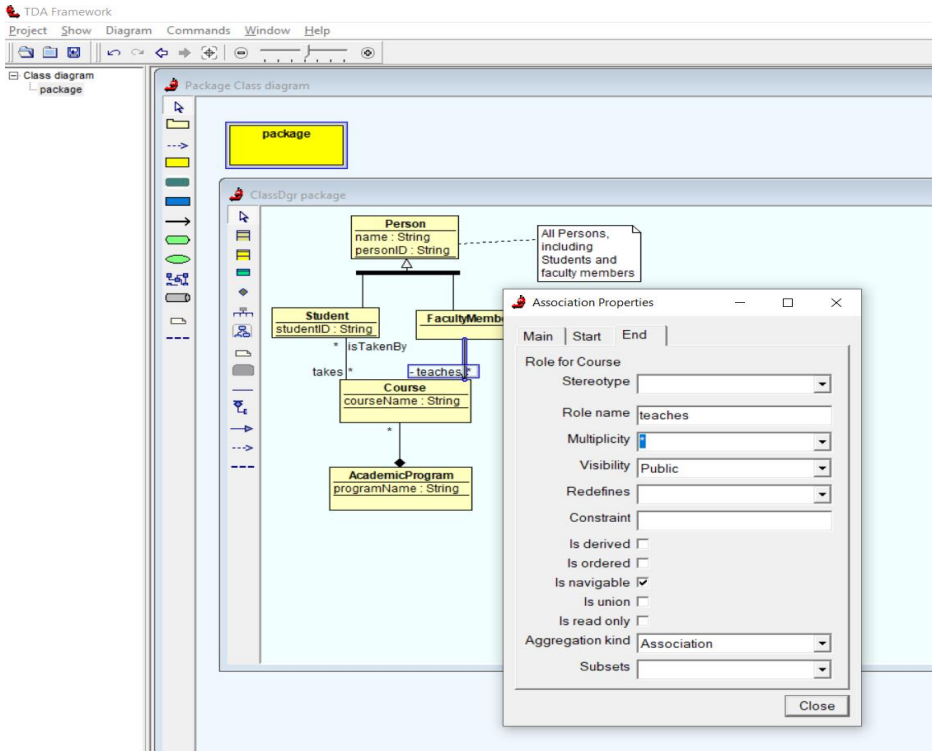

Fig. 6. GradeTwo tool

5.2. Viziquer

Viziquer (Zviedris and Barzdins, 2011; Barzdins et al., 2009b) is a graphical tool that allows one to connect to a SPARQL (Web, e) endpoint and construct visual queries (which are then translated to SPARQL) to get data from this endpoint. Viziquer allows

one to perform visualization and analysis of the data schema of the Sparql endpoint. This tool is designed to make working with RDF data easier. It should be noted that Viziquer (which was built using the L0 language) was one of the first tools for visual construction of SPARQL queries. These concepts have been further developed, e.g., in (Cerans et al., 2018).

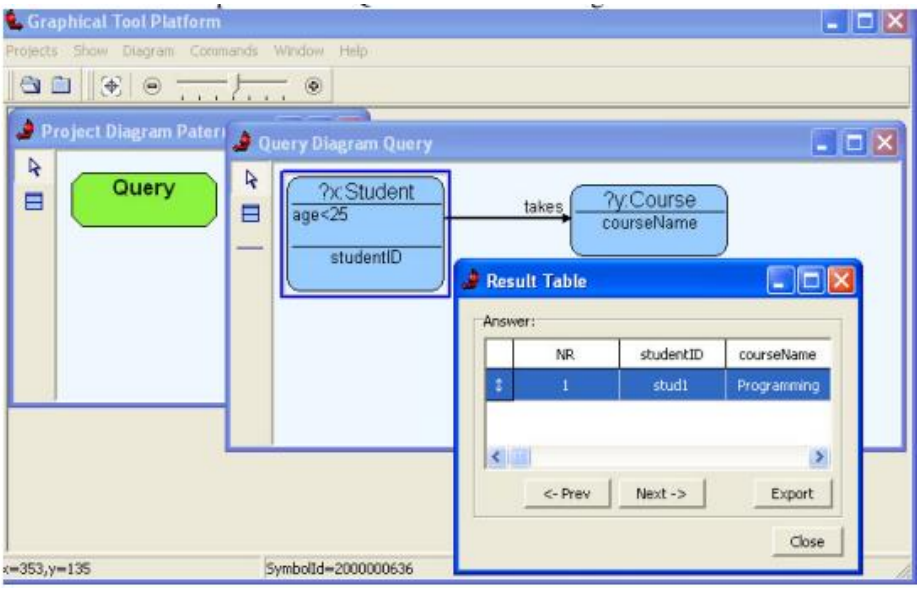

Fig. 7. Viziquer tool (Barzdins et al., 2009b)

5.3. VSAA

Business process editor for the State Social Insurance Agency (Barzdins et al., 2009a) - another DSL tool that was created with GrTP platform. This tool is quite similar to a BPMN editor. But the tool comes with 3 relatively specific services:

1. Online collaboration with a relational database – the searching for information in a database was to be combined with the graphical tool. The use case of that was a possibility to browse for normative acts during the diagram design phase – the normative acts are stored in a database and need to be accessed from the tool.
2. Users wanted to start using the tool as soon as possible – even before the language definition has been fully completed. That means it was needed to assure the preservation of user-made models while the language can still change slightly. So, the DSL evolution over time is an issue to be considered.
3. The tool had to provide the ability to create textual reports from graphical information.

5.4. Graphical editor for Project Assessment Diagrams

Graphical editor for Project Assessment Diagrams (Barzdins et al., 2009a) is an editor for visualizing business processes regarding review and assessment of submitted projects. This editor is based on UML activity diagrams and thus contains means for modeling business processes. Yet, some new attributes and some new elements have been added in order to handle the specific needs. This editor is a part of a bigger information system for document flow management (a simplified BPM suite), so services providing interconnection between the system and the editor have been provided for it. For example, the import of the model of the project evaluation diagram to the database is provided, where the external information systems are able to interact with the models created in the editor.

6. Use in the teaching process at the University of Latvia

The L0 language was used to demonstrate model transformation concepts in the Systems Modeling course of the Master's program at the University of Latvia. Within the Systems Modeling course, students were taught UML modeling languages – class diagrams, activity diagrams and other types of diagrams. While talking about class diagrams, model transformations were also discussed.

The example introducing the concept of model transformations was graph metamodel transformation. In this example, two metamodels are defined, both representing directed graphs. One (metamodel A) defines a graph as a set of vertices and edges (each edge has a starting and ending vertex). The other (metamodel B) adds additional details to each edge, specifying the starting and ending points, which are in turn attached to vertices. Both these metamodels and a special mapping link can be seen in the figure below.

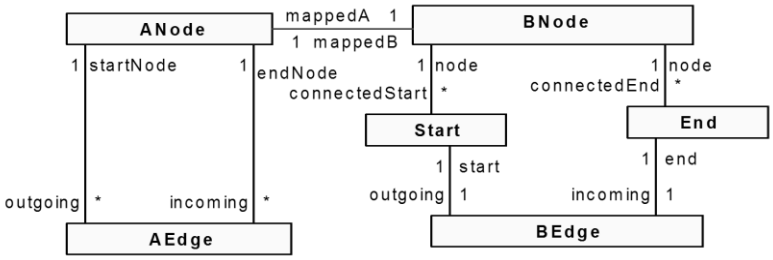

Fig. 8. Metamodel for oriented graphs with a mapping association

The basic idea of a transformation taking a graph model corresponding to metamodel A and producing a graph model corresponding to metamodel B is to create one BNode for every ANode and to transform every AEdge to BEdge with the corresponding Start and

End. To simplify this transformation, we add a mapping association to the metamodel between classes ANode and BNode. Transformation program in L0 implementing this algorithm can be found below.

```
transformation Graphs;
useMM "graphs.mmd";
main procedure Graph2Graph();
  pointer a : ANode;
  pointer b : BNode;
  pointer aEd : AEdge;
  pointer bEd : BEdge;
  pointer edgeStart  : Start;
  pointer edgeEnd    : End;
  pointer aEdgeStNode : ANode;
  pointer aEdgeEnNode   : ANode;
  pointer mapBNode : BNode;
begin;
    //copy nodes;
    first a : ANode else aNodeProcessed;
    label loopANode;
       addObj  b : BNode;
       addLink a . mappedB . b;
    next a else aNodeProcessed;
    goto loopANode;
    label aNodeProcessed;
    //copy edges;
    first aEd : AEdge else aEdgesProcessed;
    label loopAEdge;
        addObj bEd : BEdge;
        addObj edgeStart : Start;
        addObj edgeEnd : End;
        addLink bEd.start.edgeStart;
        addLink bEd.end.edgeEnd;
        //quit if not found;
        first aEdgeStNode : ANode  from aEd by startNode else aEdgesProcessed;
        first mapBNode : BNode  from aEdgeStNode by mappedB else aEdgesProcessed;
          addLink edgeStart.node.mapBNode;
        first aEdgeEnNode : ANode  from aEd by endNode else aEdgesProcessed;
        first mapBNode : BNode  from aEdgeEnNode by mappedB else aEdgesProcessed;
        addLink edgeEnd . node. mapBNode;
        next aEd else aEdgesProcessed;
        goto loopAEdge;
    label aEdgesProcessed;
end;
endTransformation;
```

Fig. 9. Transformation for oriented graphs

In the teaching field, the language L0 demonstrated itself well because its constructs correspond to elementary transformation rules. Students were also invited to create L0 transformations themselves and have been generally successful within the task. We observed that the mistakes made by students were purely syntactical rather than semantical.

In summary, the language has provided an important contribution to the Systems Modeling course regarding the explanation of the concept of model transformations.

7. Conclusions

In this paper we presented a review of practical applications of model transformation language L0. We reviewed how L0 has been used in the following areas:

- **implementing higher-level model transformation languages** – L0 has been used as a base transformation language in process of implementation of model transformation language MOLA by bootstrapping method (Barzdins at el., 2008; Sostaks and Kalnins, 2008 ; Rencis, 2008; Šostaks, 2010).
- **developing the GrTP tool building platform** - L0 was used in the development of a graphical tool building platform GrTP (Bārzdiņš et al., 2007) developed at LU IMCS. In GrTP, specification of platform internal logic is done by model transformations written in L0.
- **specifying tools within the GrTP platform** - likewise, by using L0, several DSL tools were developed (in this case by tool development we understand specification of model transformations that process changes in the interface model and accordingly update domain model), which are used in practice (Zviedris and Barzdins, 2011; Barzdins et al., 2009a; Barzdins et al., 2009b; Web, d).
- **student teaching** – L0 has been used in System modeling course at the University of Latvia in which the students have been presented with main ideas of model transformation principles.

Overall, although the first planned use case for the L0 language was to be used as a base language in the process of implementing higher-level languages, the further experience of its practical application has demonstrated that it has also had a number of equally successful applications in other areas.

By generalizing the observations presented above, we can state that using the highest-level language possible is not always the only correct path. There are quite a few situations where desired results can be achieved by using sufficiently expressive low-level languages that contain only the basic model transformation means.

Acknowledgements

This work has been carried through at Institute of Mathematics and Computer Science, University of Latvia, and has been partially supported by Latvian Science Council grant lzp-2021/1-0389 “Visual Queries in Distributed Knowledge Graphs” and project “Strengthening of the capacity of doctoral studies at the University of Latvia within the framework of the new doctoral model”, identification No. 8.2.2.0/20/I/006 (S. Rikačovs).

References

Bārzdiņš, J., Zariņš, A., Čerāns, K., Kalniņš, A., Rencis, E., Lāce, L., Liepiņš, R., Sproģis,A. (2007) GrTP: Transformation Based Graphical Tool Building Platform. In: MoDELS`07,Workshop: Model Driven Development of Advanced User Interfaces (MDDAUI-2007), available at http://ceur-ws.org, Vol 297.

Barzdins, J., Kalnins, A., Rencis, E., Rikacovs, S. (2008). Model Transformation Languages and their Implementation by Bootstrapping Method. Lecture Notes in Computer Science, Springer Verlag, 2008, pp. 130-145.

Barzdins, J., Cerans, K., Grasmanis, M., Kalnins, A., Kozlovics, S., Lace, L., Liepins, R., Rencis, E., Sprogis, A. and Zarins, A. (2009a). Domain specific languages for business process management: a case study. In Proceedings of DSM (Vol. 9, pp. 34-40).

Barzdins, G., Rikacovs, S., Zviedris, M. (2009b). Graphical query language as SPARQL frontend, ADBIS 2009, Local Proceedings, 2009, pp. 93-107.

Cerans, K., Sostaks, A., Bojars, U., Ovcinnikova, J., Lace, L., Grasmanis, M., Romane, A., Sprogis, A., Barzdins, J. (2018). ViziQuer: a Web-based Tool for Visual Diagrammatic Queries over RDF Data In The Semantic Web: ESWC 2018 Satellite Events, Springer Verlag Lecture Notes in Computer Science, Vol.11155, pp.158-163.

Estévez, J. Padrón, E. V. Sánchez, J. L. Roda. (2006). ATC: A Low-Level Model Transformation Language. In: MDEIS 2006: Proceedings of the 2nd International Workshop on Model Driven Enterprise Information Systems, May 2006, pp 64 - 74. ISBN 978-972-8865-56-6.

Jouault, F., Allilaire, F. An introduction to the ATL Virtual MachineV1.0 draft (2006), avilable at http://www.eclipse.org/m2m/atl/doc/ATL_VM_Presentation_%5B1.0%5D.pdf http://www.eclipse.org/atl/documentation/old/ATL_VM_Presentation_[1.0].pdf

Jouault, F., Kurtev, I. (2006). Transforming Models with ATL. Proc. of the Satellite Events at the MoDELS2005 Conference. LNCS, Vol. 3844, Springer-Verlag, 2006, 128–138.

Kalnins, A., Barzdins, J., Celms, E. (2004a) Model Transformation Language MOLA. – *Proc. MDAFA 2004 (Model-Driven Architecture: Foundations and Applications 2004)*, Linkoeping, Sweden, pp.14-28

Kalnins, A., Barzdins, J., Celms, E. (2004b). Basics of Model Transformation Language MOLA. - ECOOP 2004 (Workshop on Model Transformation and execution in the context of MDA), Oslo, Norway, June 14-18, 2004

Kalnins, A., Barzdins, J., Celms, E. (2004c). Efficiency Problems in MOLA Implementation. 19th International Conference, OOPSLA'2004 (Workshop "Best Practices for Model-Driven Software Development"), Vancouver, Canada, October 2004, p. 14.

Kalnins, A., Celms, E., Sostaks, A. (2006). Simple and Efficient Implementation of Pattern Matching in MOLA Tool. *Proceedings of the 7th International Baltic Conference on Databases and Information Systems (Baltic DB&IS'2006)*, 2006, pp. 159-167.

Kleppe, A., Warmer, J., Bast, W. (2003) MDA Explained: The Model Driven Architecture -- Practice and Promise, Addison Wesley, 2003

Kolovos, D., Paige, R., Polack, F. (2006). The epsilon object language (eol)., Model Driven Architecture–Foundations and Applications LNCS 4066(2006), pp 128-142.

Kolovos, D.S., Paige, R.F., Polack, F.A.C. (2008). The epsilon transformation language. In: Vallecillo, A., Gray, J., Pierantonio, A. (eds.) ICMT 2008. LNCS, vol. 5063, pp.46–60. Springer, Heidelberg (2008)

Opmanis, M., Cerans, K. (2010). JR: A Multilevel Data Repository. Proceedings of the 9th International Baltic Conference on Databases and Information Systems (Baltic DB&IS 2010), Riga, Latvia, July 5-7, 2010, pp. 375-390

Rencis, E. (2008). Model Transformation Languages L1, L2, L3 and their Implementation, Scientific Papers. University of Latvia, "Computer Science and Information Technologies", 2008.

Rikacovs, S. (2008). The base transformation language L0+ and its implementation, Scientific Chapters, University of Latvia, “Computer Science and Information Technologies”, 2008, pp. 75–102.

Sostaks, A., Kalnins, A. (2008). The implementation of MOLA to L3 compiler, Scientific Papers University of Latvia, “Computer Science and Information Technologies”, 2008.

Šostaks, A. (2010). Implementation of Model Transformation Languages. Ph.D. thesis, University of Latvia (2010)

Zviedris, M., Barzdins, G. (2011). ViziQuer: A Tool to Explore and Query SPARQL Endpoints, Lecture Notes in Computer Science, 2011, Volume 6644/2011, pp. 441-445.

WEB (a). OMG. MOF 2.0 Query/View/Transformation Specification. Available at http://www.omg.org/docs/ptc/07-07-07.pdf

WEB (b). ATL, available at http://www.sciences.univ-nantes.fr/lina/atl/

WEB (c). Specification of the ATL Virtual Machine, available at http://www.eclipse.org/atl/documentation/old/ATL_VMSpecification[v00.01].pdf

WEB (d). GradeTwo, available at http://gradetwo.lumii.lv/

WEB (e). SPARQL Query Language for RDF, available at http://www.w3.org/TR/rdf-sparql-query

WEB (f). Eclipse Modelling Framework URL, available at http://www.eclipse.org/emf/

WEB (g). Metadata Repository (MDR), available at http://mdr.netbeans.org/

Received December 1, 2023, accepted December 29, 2023