A Graph Diagram Engine for the Transformation-Driven Architecture

Janis Barzdins Karlis Cerans Sergejs Kozlovics
Edgars Rencis Andris Zarins
Institute of Mathematics and Computer Science, University of Latvia
Raina blvd. 29, LV-1459, Riga, Latvia
{janis.barzdins, karlis.cerans, sergejs.kozlovics, edgars.rencis, andris.zarins}@lumii.lv

ABSTRACT
The transformation driven architecture (TDA) is a system building (in particular, tool building) approach that is based on model transformations, interface metamodels with corresponding engines, and event/command mechanism. This paper describes a metamodel and the corresponding engine for graph diagram presentations within TDA. The facilities of the metamodel and the engine include static diagram presentations, as well as graph diagram animations.

Keywords
Transformation-Driven Architecture, model transformations, metamodels, graph diagrams, diagram animation, modeling tools.

1. INTRODUCTION
The increasing variety of metamodel-based tools such as MetaEdit [1], Eclipse GMF [2], Microsoft DSL Tools [3], DiaGen/DiaMeta [4] and METAclipse [5] has lead to study of principles behind tool architecture. Such tools allow domain data to be represented in a graphical form according to some (perhaps, implicit) presentation metamodel. In [6] we have developed an approach called the *Transformation-Driven Architecture* (TDA), where not just one, but several presentation metamodels are allowed. And the link between domain and presentation models within a modeling tool is established by means of model transformations.

Since a presentation model is not yet the end interface that can be shown to the user, some engine is needed to construct the corresponding diagram itself from the instance of the presentation metamodel. Such engines form an essential part of the TDA.

Developing a presentation engine and the corresponding metamodel may be a non-trivial job. But once implemented, the corresponding engine can be reused in several tools built upon the TDA.

In this paper a metamodel for graph diagram presentations within TDA and the corresponding engine for drawing/editing graph diagrams is presented. The metamodel along with the engine is a development over previous authors' work [7] by fully elaborating the metamodel and putting it within the context of TDA. The graph diagram animation facilities are also newly sketched here.

The paper is organized as follows. The next section lists some ideas of the TDA and explains how the proposed Graph Diagram Engine can be integrated within the TDA Framework. In Sect. 3 the Graph Diagram Metamodel and the Graph Diagram Engine are explained. Sect. 4 presents a way of implementing animation mechanism for graph diagrams. Finally, Sect. 5 concludes the paper.

2. THE ESSENCE OF THE TRANSFORMATION-DRIVEN ARCHITECTURE (TDA)
The Transformation-Driven Architecture (TDA) [6] is a metamodel-based system (in particular, tool) building approach, where the system metamodel consists of one or more presentation metamodels served by the corresponding engines and the (optional) Domain Metamodel. There is also the Core Metamodel (fixed) with the corresponding Head Engine. Model transformations are used for linking instances of the mentioned metamodels (see Fig. 1).

The Head Engine is a special engine, whose role is to provide services for transformations as well as for presentation engines. For instance, when in some presentation engine a user event (such as a mouse click) occurs, the Head Engine may be asked to call the corresponding transformation for handling this event. Also, a transformation may give commands to presentation engines. Thus, the Core Metamodel contains classes *Event* and *Command*, and the Head Engine is used as an event/command manager.

The TDA has its own framework that comes with the built-in Head Engine (serving the Core Metamodel) and some predefined pluggable engines (the Graph Diagram Engine is one of them). Other presentation engines may also be written and plugged-in, when needed. The TDA framework is common to all the tools built upon the TDA. The framework is brought to life by means of transformations. One can choose between writing different transformations for different tools and writing one configurable transformation covering several tools.

3. GRAPH DIAGRAM METAMODEL AND GRAPH DIAGRAM ENGINE
The visual elements of the presentation (see Fig. 2) correspond to the classes *GraphDiagram*, *Box*, *Line*, and *Port* along with *Compartment* corresponding to text fields that may be placed inside boxes or attached to lines and ports. Instances of the classes just mentioned will be diagrams and graphical elements created by the user. Every element, compartment and graph diagram has its style (see the *ElementStyle* class with its subclasses as well as classes *CompartmentStyle* and *GraphDiagramStyle*) being a

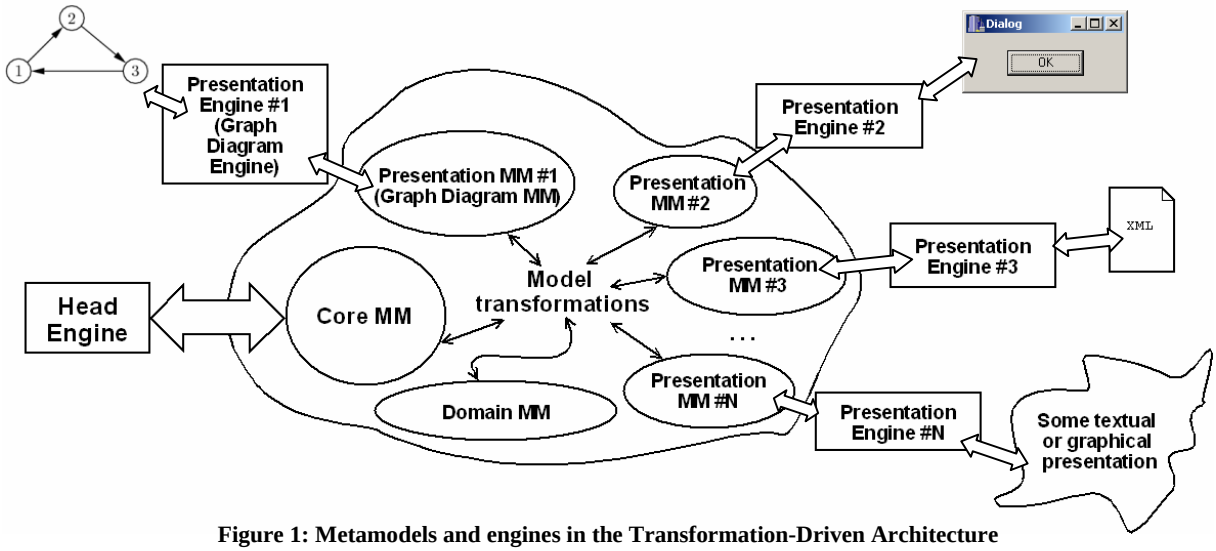

Figure 1: Metamodels and engines in the Transformation-Driven Architecture

composition of various style attributes. For example, the element style attributes supported by the graph diagram engine include line color, background color, shape, line width, attached pictures, etc. The compartment style attributes include such items as alignment, adornment, visibility, text font, etc.

Every *GraphDiagram* has its *Palette* consisting of *PaletteElement*s, each of them being a line, a box or a port. *Toolbar*s* consisting of *ToolbarItem*s* can also be associated with the *GraphDiagram*. Additional *MenuItem*s* could also be associated with the graph diagram. When the graph diagram is being activated, the corresponding toolbars and menu items are made visible. The currently active diagram is pointed to by the *ActiveDiagramPointer* singleton instance. The elements currently selected are linked to a *CollectionOfActiveElements*.

The metamodel allows the user to specify also a *PopUpMenu** consisting of *PopUpMenuItem*s**. Such a menu is usually activated when the user clicks the right mouse button.

The Graph Diagram Metamodel defines engine-specific events and commands that are subclasses of *Event** and *Command**. Every event and every command during tool runtime is placed within the context defined by the metamodel. For example, the *NewBoxEvent* is attached to the *PaletteBox* with which it is being created, and the *Box* in which it is being put (see associations from class *NewBoxEvent*).

The singleton class *GraphDiagramEngine* contains attributes that correspond to engine's events. In the beginning a transformation can assign values for these attributes, each value representing the name of the transformation that has to be called when the particular event occurs.

The graph diagram engine is responsible for visualizing instances of the Graph Diagram Metamodel. The engine is developed on the basis of graphical engines for GRADE tools family [8]. The engine relies on advanced graph layout algorithms [9, 10] as well as effective internal diagram representation structures allowing to handle the visualization tasks efficiently even for large diagrams.

4. GRAPH DIAGRAM ANIMATION

Although there are several approaches for metamodel-based handling of dynamic multimedia objects that include animations (see, for instance, [11]), our goal here is more specific — to provide simple animation facilities for graph diagrams explained in Sect. 3. Complex interactive animations (such as animations that can be created in *Microsoft Silverlight* [12] or *Adobe Flash* [13]) are beyond our scope.

In Fig. 3 we extend the metamodel of Fig. 2 by classes for describing graph diagram animations. The animation of graph diagrams is based on the notion of token that is associated with some element (box or line) in a graph diagram. Tokens do not imply any semantics, they are used only for managing the animation process. The semantics is up to transformations.

A token is started by *StartTokenCmd* that specifies also its *duration* (how long the token "lives"). There are also commands for starting, stopping, pausing and resuming a token in the diagram, as well as pausing, resuming and stopping all tokens in the diagram. The "end of life" of a token is observed by the presentation engine — at that time it creates a corresponding *EndTokenEvent*. There can be several tokens living concurrently in the diagram.

An *explicit token* is able to simulate the activity of the associated element for the given duration. The visual effect of the simulation is determined by *TokenStyle* instance associated with the token. If an *ElementStyle* instance is associated with the token style, then the animation consists of just changing the element style for the token's lifetime. Other options of animation consist of moving a bullet of certain size or some image along the line in the diagram, or animating a box by a line flowing over it in certain direction, with or without leaving the trailing part in the specified color. In the case of *AUTOMATIC* direction, the actual line flowing direction is determined by the presentation engine on the basis of the placement of the actual outgoing line from the box. A *hidden token* doesn't animate any element, but just "lives" for the

* From the Core Metamodel.

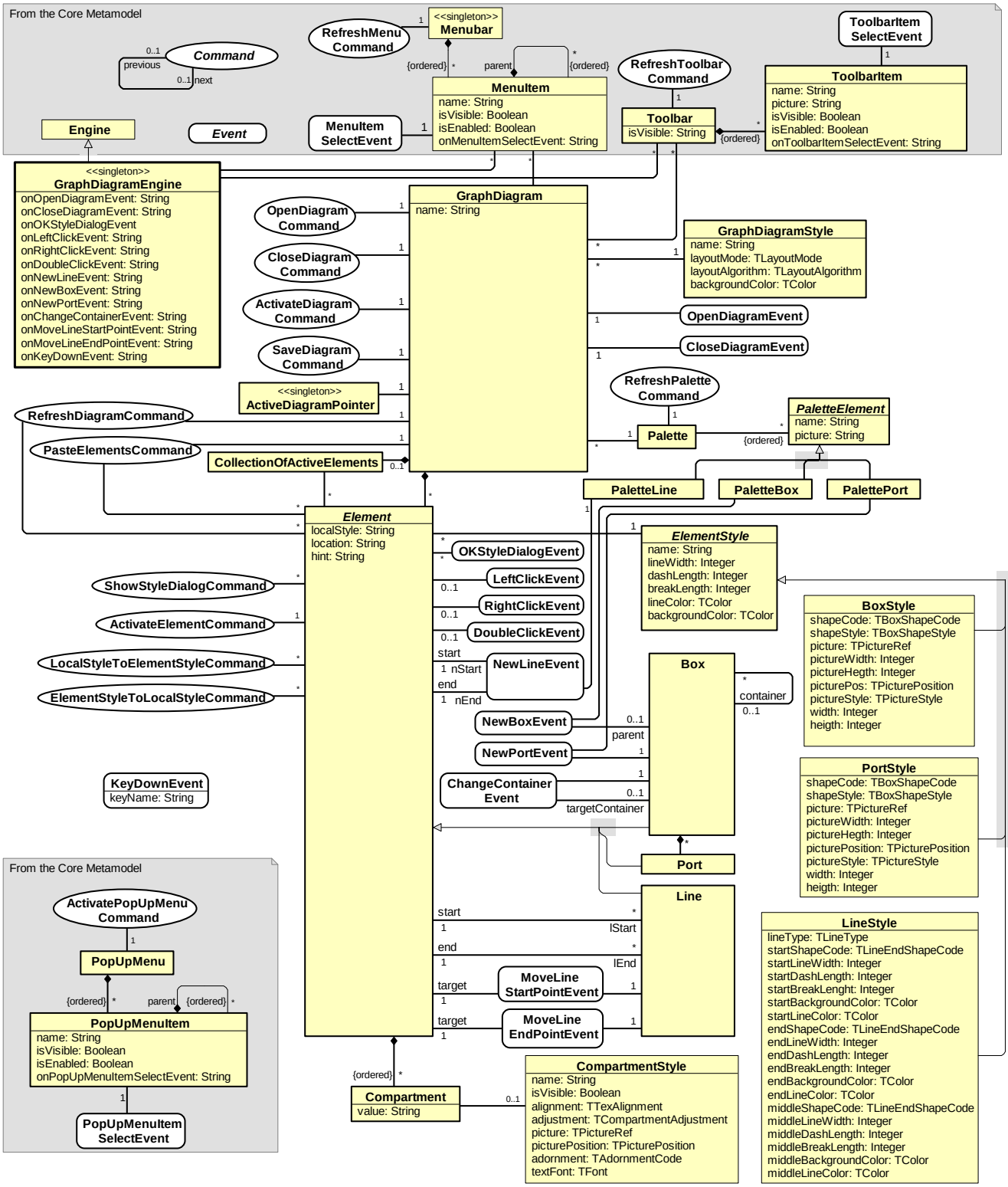

Figure 2: The Graph Diagram Metamodel. Ellipses are *Command* subclasses, while rounded rectangles are *Event* subclasses.

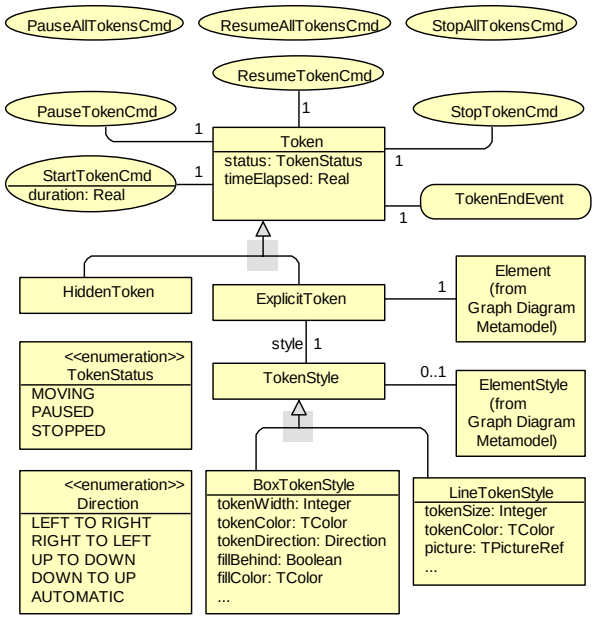

Figure 3: Adding animation capabilities to the Graph Diagram Metamodel

specified amount of time. Hidden tokens can be useful, e.g., for accounting the global animation time, or for creating certain breakpoints during the animation when the control is transferred to transformations for some semantic actions.

The implementation of animation facilities in our graph diagram engine is currently under development.

5. CONCLUSIONS

The Graph Diagram Engine has been successfully implemented in recent version of transformation-based tool building platform GrTP [7]. The GrTP tool is now being transformed to the TDA framework, which should become publicly available soon. At the moment, the TDA framework consists of two predefined engines (one of them is the Graph Diagram Engine and the other is the Dialog Engine), and the interaction between these engines and model transformations performed by means of commands and events works well. We are working on ameliorating the TDA framework and its engines. One of the research topics here is adding advanced graph diagram layout capabilities to the Graph Diagram Engine. We are working also on implementing diagram animations within the Graph Diagram Engine for TDA.

Several diagram editors (such as class diagram editor and activity diagram editor) have been successfully built using the Graph Diagram Engine. This engine has been also used in [14]. We are looking forward for applying the TDA and its engines in the Semantic Web domain.

6. ACKNOWLEDGMENTS

The authors would like to acknowledge (alphabetically) A. Kalnins, L. Lace, R. Liepins, A. Sprogis, R. Zarits and M. Zviedris for their efforts in materializing the ideas presented in this paper.

7. REFERENCES

[1] MetaEdit+ (http://www.metacase.com).

[2] A. Shatalin and A. Tikhomirov. Graphical Modeling Framework Architecture Overview. *Eclipse Modeling Symposium*, 2006.

[3] S. Cook, G. Jones, S. Kent and A. C. Wills. *Domain-Specific Development with Visual Studio DSL Tools*, Addison-Wesley, 2007.

[4] DiaGen/DiaMeta (http://www.unibw.de/inf2/DiaGen).

[5] A. Kalnins, O. Vilitis, E. Celms, E. Kalnina, A. Sostaks and J. Barzdins. Building Tools by Model Transformations in Eclipse. *Proceedings of DSM'07 Workshop of OOPSLA 2007*, Montreal, Canada, Jyvaskyla University Printing House, pp. 194–207, 2007.

[6] J. Barzdins, S. Kozlovics, E. Rencis. The Transformation-Driven Architecture. *Proceedings of DSM'08 Workshop of OOPSLA 2008*, Nashville, USA, pp. 60–63, 2008.

[7] J.Barzdins, A.Zarins, K.Cerans, A,Kalnins, E.Rencis, L.Lace, R.Liepins and A.Sprogis. GrTP: Transformation Based Graphical Tool Building Platform. *Proceedings of MDDAUI Workshop of MoDELS 2007*, Nashville, USA, 2007.

[8] GRADE tools (http://www.gradetools.com).

[9] P. Kikusts and P. Rucevskis. Layout Algorithms of Graph-Like Diagrams for GRADE Windows Graphic Editors. *Proceedings of Graph Drawing '95*, LNCS, vol. 1027, pp. 361–364, Springer-Verlag, 1996.

[10] K. Freivalds and P. Kikusts. Optimum Layout Adjustment Supporting Ordering Constraints in Graph-Like Diagram Drawing. *Proceedings of The Latvian Academy of Sciences*, Section B, vol. 55, No. 1, pp. 43–51, 2001.

[11] A. Pleuss, A. Vitzthum, H. Hussmann. Integrating Heterogeneous Tools into Model-Centric Development of Interactive Application. *MoDELS 2007*, LNCS, vol. 4735, pp. 241–355, Springer-Verlag, 2007.

[12] Silverlight Animation Overwiew. *MSDN*, Microsoft Corp. (http://msdn.microsoft.com/en-us/library /cc189019(vs.95).aspx).

[13] Adobe Flash (http://www.adobe.com/products/flash).

[14] G. Barzdins, E. Liepins, E., M. Veilande, M. Zviedris. Semantic Latvia Approach in the Medical Domain. In Haav, H-M., & Kalja, A. (eds), *Proceedings of the 8th International Baltic Conference* (Baltic DB&IS2008), June 2-5, Tallin, Estonia, Tallinn University of Technology Press, pp. 89–102, 2008.