How to make visual modeling more attractive to software developers

Andrey Terekhov, Timofey Bryksin, and Yurii Litvinov

Saint Petersburg State University, Saint Petersburg, Russian Federation
{a.terekhov, t.bryksin, y.litvinov}@spbu.ru

Abstract. Visual modeling paradigm is known for a number of decades already but still vast majority of software engineers prefer traditional programming to automated code generation from visual models. In this paper we discuss several ways to make modeling using diagrams more usable for industrial software development. Specialized domain-specific visual languages could be used to make models clearer and more understandable. Also a lot of effort is required for tool developers to make tools supporting these languages less difficult to use in everyday work. We present QReal DSM platform and discuss what we have done to make visual IDEs created on this platform easier and more productive to use, and the process of their creation simpler. In particular we discuss mouse gestures recognition for rapid creation of elements and links on diagrams and a number of special features that increase productivity of modeling process.

Keywords: usability, domain-specific modeling, software engineering

1 INTRODUCTION

Many engineering psychologists believe that people perceive graphic images better than textual information[1]. Graphical diagram languages have already been used for a long time in different fields: in telecommunications (SDL, Specification and Description Language[2]), in aviation and other technically complex areas (SADT, Structured Analysis and Design Technique[3], the most famous part of SADT is IDEF0), in the description of business processes (BPMN, Business Process Model and Notation[4]), and so on. A lot of tools that support a variety of visual languages and models were developed. Model Driven Architecture[5] and Model Driven Engineering become more and more popular.

---

1 For some recent discussion on this topic see, for example, [4]

2 Specification and Description Language, URL: www.itu.int/ITU-T/studygroups/com10/languages/Z.100_1199.pdf

3 Structured Analysis and Design Technique, URL: http://en.wikipedia.org/wiki/Structured_Analysis_and_Design_Technique

4 Business Process Model and Notation, URL: http://www.bpmn.org/

5 Model Driven Architecture, URL: http://www.omg.org/mda/

Nevertheless, software development technologies based on graphical modeling are not so generally used, traditional textual programming using algorithmic languages still remains the mainstream. At the Software Engineering Department of Saint Petersburg State University graphical technologies of software development are being created for many years for both academic research and industrial needs. Therefore, we are very interested in why such tools are not very widely used right now.

We realized that general-purpose languages such as UML[6] are too cumbersome quite a long time ago. It is too complex and expensive to use them as a replacement for textual languages in real industrial practice. Small specialized languages "sharpened" on narrow domain areas (DSLs, Domain Specific Languages) in this sense are much more convenient. They are more expressive and allow automatic generation of effective source or binary code. Domain experts can easily study these languages. There are studies that show 5-10 times productivity increase of mobile application developers who used DSLs compared to those who used traditional programming [7]. Our experience with DSLs for programming robots [15] shows similar results.

Evidently, it is too expensive to implement graphic editors, code generators, run-time environment, debuggers, etc. for each particular DSL. A better solution is to create a meta-technology, or a domain-specific modeling (DSM) platform, that could generate all necessary tools from compact formal description of desired DSL. There are several meta-technologies in the market right now, but some of them are strictly tied to a specific platform (e.g. Visual Studio Visualization and Modeling SDK[7]), others are so overwhelmed with features that are quite difficult to use (e.g. Eclipse Graphical Modeling Project[8]) or expensive and not widely distributed (e.g. MetaEdit+[9]), so we decided to develop our own meta-technology using our experience in this area.

Main contribution of this article is the statement that for a success of visual modeling user-friendliness and convenience of tools are crucial. This statement is supported by an extensive overview of existing empirical research on why visual modeling is not so widely used as it could be. Our own ideas on how to improve user-friendliness of CASE tool and meta-technology that helps to create such CASE tools are presented, their implementation in QReal technology are discussed.

2 RELATED RESEARCH

In 1990s CASE tools market experienced a remarkable growth, but at the same time research papers begin to appear noticing that a lot of acquired tools are almost never used in everyday work or are used very inefficiently. Most researchers highlight

---

6 Unified Modeling Language, URL: http://www.uml.org/

7 Visual Studio Visualization and Modeling SDK, URL: http://archive.msdn.microsoft.com/vsvmsdk

8 Eclipse Graphical Modeling Project, URL: http://www.eclipse.org/modeling/gmp/

9 MetaEdit+, URL: http://www.metacase.com/products.html

economical [13], organizational [1] and technical reasons [5] of massive tool misuse, but also a lot of papers notice substantial usability problems [2, 5, 6, 9].

In [5] Huang noticed that most of the existing tools focus more on particular techniques and formal processes than on its users. According to Huang tools should adjust themselves to different user skill levels: hide too much details and divide complex operations into simpler ones for novices, help them via different wizards and tooltips; provide more low-level functionality and freedom to use it for experts. It could help to make the learning curve less steep and make the tools available for a wide range of developers.

Jarzabek and Huang in [6] comment that a tool that relies on formal methods too much can easily create more problems for its users that it helps to solve. It results in forcing a user to perform not the operations that he or she wants to perform, but operations that must be performed in current context. For instance, strict constraints on created diagrams either brought by a language or by a methodology can result in a lot of discomfort for a developer restricting his or hers natural train of thought. The authors note that a "free-style" graphical editor mode could be very useful: users create diagrams as they like without any semantic rules and later rearrange them according to language semantics (better with the help of the editor). Damm et al. [2] agree: most CASE tools support rather latest stages of development process (design, implementation, and so on) than early ones — e.g. work with requirements and analysis. To support them the tools should be more agile and in a way less formal.

User-friendly tool interface on its own can play a vital part in choosing tools for everyday work. The research by Davis et al. [3] has shown that the more attractive developers find user interface of a tool, not only the more willingly they will use this tool for routine operations, but also they will find this tool more useful than less easy to use but more feature-rich ones. Lending et al. [9] agree on that and add that user-friendliness could be even more strong motivation to use a tool than an increase in productivity that it may give.

According [12] only 15% of software developing companies use model-based tools in their work. Selic believes that the main reason for that is that programming is being taught in traditional ways using textual languages, and such developers later see visual modeling as an auxiliary activity that is often propagated by the top management and distracts from real work. Indeed languages like UML are generally used for documentation purpose and to enable full code generation such diagrams must contain all implementation details. Moreover, this makes such diagrams huge and not easy to understand and maintain. A number of recent studies however show that model-based development can succeed when specialized domain-specific visual languages are used (see, for instance, [12]**)**.

In subsequent section, we describe QReal DSM platform that was originally created keeping in mind such usability problems.

3 QREAL DSM PLATFORM

QReal [8, 16] is a DSM platform that is being developed by research group of Software Engineering Chair of St. Petersburg State University since 2007. It is an open-source project, written in C++ with Qt library. Its main goals are to provide easy to use crossplatform multi-user technology to create domain-specific languages, which also can serve as a base for research in visual modeling, metamodeling, visual languages and related fields. What distinguishes QReal from other existing DSM platforms is that it is designed with a principle "one does not need to know something that he does not use", so it makes possible to create DSLs without special training in metamodeling and visual language formalisms. It also differs from other academic DSM platforms as it provides means to create complete DSM solution, not only visual editors or model transformation tools.

QReal consists of an abstract core, which provides functionality common to all visual languages such as generic user interface, a generic visual editor, a repository with models, property and model editors and so on, and a number of plugins, which implement all language specifics. QReal core has no code that is specific to particular language. Language syntax information, such as list of language elements, their properties and shapes and so on, is provided by plugins and used by generic visual editor. The core also contains utilities needed to create supporting tools such as framework for generators creation or graph transformation engine.

Language syntax is specified in XML-based textual metalanguage or using visual metaeditor, visual and text languages are convertible between each other. A metamodel can be converted automatically into C++ code of an editor, which is then compiled into a shared library and opened in QReal as a plugin. A metamodel can also be opened for interpretation by the core directly, without the need for generation.

Several DSM solutions were created using QReal technology, most notable is QReal:Robots, a development environment for robots programming. It is now used in several Russian schools to teach 5-7th grade students the basics of programming. QReal:Robots has a visual language consisting of about 40 elements, a generator and an interpreter which allow to execute programs on a Lego NXT robot or on a simulator model included in QReal:Robots itself. There are numerous publications describing this case, such as [14]. Second DSM solution used in production is QReal:BP, a tool for business processing modeling. It is based on BPMN modeling language and consists of a visual editor and C#.NET and Python code generators.

4 USABILITY IMPROVEMENTS

There are two main reasons why developers may find a visual modeling tool difficult to use: inconvenience of the tool itself (many routine operations that are needed to be done while working, difficult to understand user interface, tangled process of development etc.) and inconvenience of modeling language(s) the tool is based on. The second problem is partially solved by the DSM paradigm itself: developers create and use specialized languages that consist of well-known entities from target domain,

not abstract classes and objects. Nevertheless, that raises another issue: development of new languages and tools for them is a difficult task on its own, and should be optimized too. The following subsections describe some features that help to simplify modeling and metamodeling tasks.

4.1 "METAMODELING ON THE FLY"

All DSM platforms use some means to formally specify a visual language to be able to automatically generate tool support for it. Almost all of them use some metalanguage (a language to specify other languages). Advanced DSM platforms have visual metalanguage and visual metaeditor, and the process of DSL creation consists of development of its metamodel in such metaeditor, automatic creation of a visual editor from the metamodel, and automated or manual development of other tools like generators, model transformation tools and so on.

However for end users this approach has one major drawback from usability point of view. To be able to create a language, even a small one, one needs to know things that may be beyond the scope of knowledge for even seasoned software engineer. Those things include work with formal syntax of a visual language, the concepts of metamodeling, and so on. So, development of a new DSL in most cases can only be done in collaboration with authors of selected DSM platform, who have knowledge and experience required to do this.

QReal platform also has a visual metaeditor, but as an alternative proposes a different approach, called informally metamodeling on the fly. The main idea of this approach is that a language can be specified right in a process of drawing diagram in that language. For example, a user can add new element to a palette (and a language), change its shape, add properties and so on. The point of this new approach is that all features commonly related to metaeditor, like adding a new element, are implemented as seamless extensions of the diagram editor interface, no metaeditor is needed and a user may not even know about underlying metamodel and related formalism.

This approach is aimed to help inexperienced language developers to quickly create small visual languages without much theoretical knowledge and training with DSM platforms. It also greatly reduces effort in early prototyping phase of DSLs creation, so it is also useful for experienced developers, considering that a prototype created with metamodeling on the fly can be opened in a metaeditor for further language refining. Similar methodology was proposed in Agentsheets platform [11], and its authors proposed to create languages with participation of an end user, when language developer and a user worked on the same computer, the user tries to draw a model, and the developer adds new features to a language and fixes issues right in place. This method is also feasible with metamodeling on the fly, but our technology makes it possible even to develop a language by end users entirely, without help from developers. It is possible because Agentsheets does not have a metaeditor and all possible changes to a language shall be done when drawing a diagram. In QReal metamodeling on the fly allows to create only a prototype of a language with many formal aspects assumed as defaults, which is good enough in many cases and allows

to keep user interface very simple. Then created prototype may be refined and extended in metaeditor if needed.

This technology was implemented in QReal and several small and medium-scale languages were created with it (from only several to nearly 30 entities in a language) [15]. There are some implementation issues worth noting. First, metamodeling on fly is only possible when metamodel is interpreted by the DSM platform's core, so it can be changed dynamically. Second, an ability to change a language when editing a diagram on that language requires some implementation of concurrent model and metamodel evolution features. For example, we may change a type of a property in a metamodel, but existing elements already have those properties filled with some meaningful values. There are many such situations, and also it is possible to open a project created with earlier version of the language, which further complicates implementation (see [15] for details). Current implementation only shows user a warning about those situations, more automated ways of dealing with such problems are considered for further research.

4.2 MOUSE GESTURES RECOGNITION

The effectiveness of each tool is determined by how easy and fast it allows its users to perform operations that this tool is created for. In modeling some of the most frequent operations are creation and deletion of elements on diagrams and relationships between them. In most visual modeling tools to create an element the user should first find it on some kind of palette, toolbar or menu, and then click on it or drag-and-drop it on a diagram canvas. It gets even worse if a tool supports a number of languages with dozens of elements in each. Such basic task as creation of an element results in a sequence of purely mechanical operations that must be repeated again and again. Needless to say that to do them the user should recall what menu, toolbar or palette tab this particular element is located on, switching mental context from hierarchy of models to usage of particular tool. We believe that task of elements creation could and should be automatized, made easy to perform.

As an alternative for traditional element creation mechanism we suggest mouse gestures: trajectories drawn by mouse pointers. Each language element is paired with a so called "ideal gesture", a recorded path of mouse pointer that each new user gesture will be compared to. As a default value each element's ideal gesture is initialized to look similar to this element's visual representation. It happens automatically when visual editor is generated from language metamodel. Custom ideal gesture for any element could be set using QReal's special gestures editor at any time later. While modeling users perform mouse gestures (holding right mouse button pressed to differentiate from regular mouse movements, see Fig. 1) which are compared to ideal gestures of current language elements. If a match is found a new element is created on a diagram (the element that recognized ideal gesture corresponds to).

There are no special mouse gestures for each particular association. To create a link between two elements on a diagram one should just perform a gesture of any kind starting and ending on an existing element. Special QReal component checks what

types of links are possible between these elements according to this language's metamodel, and suggests choosing one of them. If there could only be one of them, it will be created automatically (see Fig. 2).

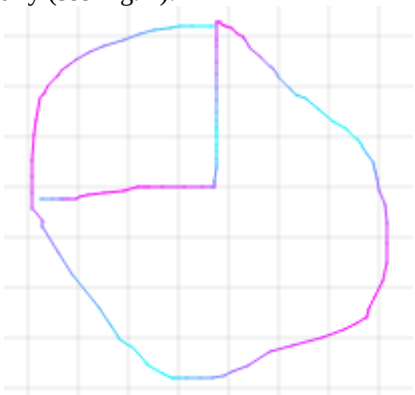

Fig. 1. Creating elements using mouse gestures

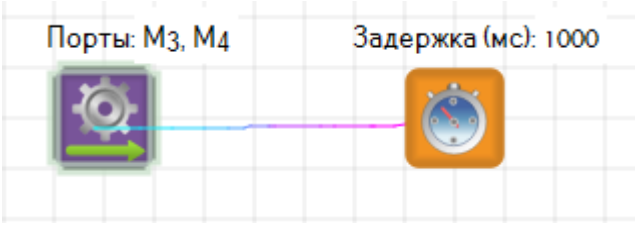

Fig. 2. Creating links using mouse gestures

For more detailed information about implementation of mouse gestures recognition in QReal see [10].

4.3 VISUAL EDITORS FEATURES

Linkers

In addition to mouse gestures recognition there is one more mechanism for rapid creation of relationships in QReal: so called linkers. Linkers are UI elements that are displayed near to elements on diagrams when users select these elements.

Clicking on a linker and moving the mouse while holding left mouse button pressed, one can "drag" a link out of an element. Releasing mouse button on an existing element results in creation of a link connecting these two elements: the one the linker belongs to and the one that the mouse was released on. If a mouse is released on an empty diagram space, a special menu is shown. This menu allows either create a "dangling" links from selected element to this position or create a new element here and connect it with the first one. In case an element can have several types of outgoing links, several linkers are shown near it. Each of them represents one of relationship types.

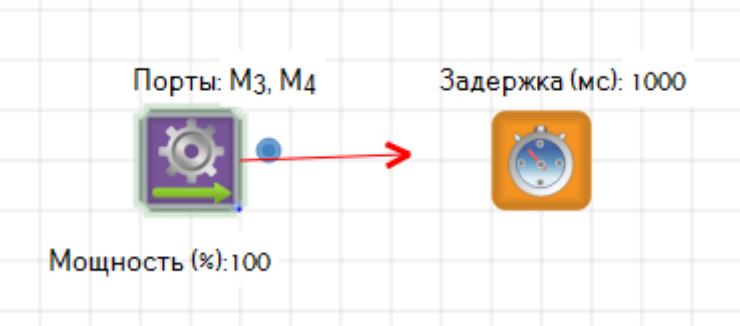

Fig. 3. Creating links using linkers

Embedded control widgets

Any non-trivial language element has a set of properties (or attributes) of various types. To change these attributes development environments traditionally employ property editors or special modal windows that appear when an element is being clicked on. Moreover, property editor or window are often the only way to see values of most of the properties. Switching to such windows or to a property editor multiple times while modeling is another time-consuming operation that we believe could be optimized. In QReal we use so called embedded control widgets — special control elements that are placed within elements' graphical representation and are used to show and manipulate properties values.

Surely, a lot of such widgets can easily make any diagram unreadable, so language developer should find a reasonable balance between informative and compact diagrams. Using QReal's special shape editor language developers can adjust visual representations of language elements: choose properties that will be displayed with control widgets and define position and type of such widgets.

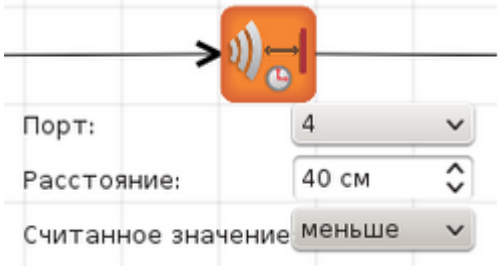

Fig. 4. Embedded control widgets on scene

DRAKON language usability heuristics

Several usability heuristics were taken from the DRAKON visual language[10] that was created in a way to ensure good readability of created diagrams and comfort of

[10] DRAKON visual language, URL: http://en.wikipedia.org/wiki/DRAKON

the modeling process itself. We describe two of them — so called splittable links mechanism and creating elements in groups.

Often existing diagrams should be extended with new elements that should be positioned between existing ones. For instance, this often takes place in control-flow diagrams when inserting elements in the middle of a sequence of elements. It also takes a lot of mechanical operations: create a new element, reconnect existing link to it, create another link from the new element to the following one, reposition all elements and links. To simplify this we implemented the following: when an element is drag-and-dropped on an existing link, all mentioned steps are performed automatically, including repositioning the lowest element (and all elements that are placed lower on a diagram). The reverse action is also very helpful: when the middle element is deleted QReal deletes one of the dangling links and connects another one to the element following the deleted one.

Another useful feature of a visual editor is creation of elements in logical groups. Such feature could be helpful for working with languages that have composite elements. For example, block diagram can have Loop element consisting of a LoopCondition and LoopEnd elements and a number of other elements between them. In QReal we added “Group” metamodel entity that allows to describe such composite elements. When visual editor is ready, each group is represented by a special palette element. When such element is drag-and-dropped on a diagram all elements that were defined for this group in the language metamodel are created at once.

Empirical study

A series of experiments were carried out to determine which way to create elements and links on a diagram is more efficient. We embedded a special module into QReal that logged all low-level user actions and created a tool to search for complex high-level actions within these logs. For instance, “Drag and drop an element from palette to scene” user action consists of several basic actions, such as “Move cursor to the palette”, “Scroll the palette to find target element”, “Select the element”, and so on.

24 participants took part in our study, mainly students and young engineering specialists – main target user groups for QReal:Robots. They were suggested to create a given diagram using different editor features, one at a time: drag-and-drop from palette, linkers, or mouse gestures. We measured and compared what time it takes to create elements and links using each editor feature and how many mistakes users make during diagram creation process. We found out that creating an element using mouse gestures is not much faster than using palette: it took about 2.13 sec to create a single element using the palette, 3.55 sec using linkers, and 1.89 sec using mouse gestures. But when users create a link between two elements, it appears that mouse gestures take the least time: 1.35 sec for mouse gestures, 2.8 for linkers, and from 2 to 5.1 sec for palette (2 sec for creating the link itself and about 1.5 sec for connecting the link to each element). However, when users work with palette they make less mistakes, about 20% of user actions during mouse gestures had to be redone.

Our study also resulted in a number of suggestions that can improve tool usability by decreasing the number of routine user actions and simplify modeling process. Such

improvements include rearranging of linker menu items and palette elements to reduce mouse movements for most frequently used actions and elements, auto aligning of newly created elements, increasing size of linker UI elements to make them easier to click on, and many others. Implementing these suggestions helped to make QReal:Robots more user-friendly according to user feedback.

5 CONCLUSIONS

Despite many advantages that visual languages have over traditional textual languages, they are still not widely used. We believe that this is so because of complexity of general-purpose visual languages like UML and lack of focus on users in current CASE tools. Complexity of general-purpose languages can be avoided by domain-specific modeling paradigm, but it brings even more complexity to the supporting tools, which makes them even less easy to use. As convenience and user-friendliness are a major (if not primary) factor in adoption of a tool or technology, everyone developing such tools should carefully consider it.

In this article, we provided an overview of empirical research that confirms our point of view and presented our own DSM platform that implements some of the features to raise perceived user-friendliness of language creation process, and, more importantly, language usage process. Such features can range from purely technical (like the ability to drag links directly from language elements) to methodological (like "metamodeling on the fly", which allows to hide a huge amount of complexity of language creation process and make formal specification of visual languages a seamless part of diagram drawing). Features like these can significantly lower entry threshold for complex tasks like visual modeling and even language creation and significantly raise productivity of professionals that already use visual modeling in everyday practice.

6 REFERENCES

1. Damm, C.H., Hansen, K.M. An Evaluation of Workspace Awareness in Collaborative, Gesture-based Diagramming Tools. People and Computers XVIII — Design for Life, Springer London, 2005, pp. 35-49
2. Damm, C.H., Hansen, K.M., Thomsen, M., Tyrsted, M. Creative Object-Oriented Modelling: Support for Intuition, Flexibility, and Collaboration in CASE Tools. In Proceeding of ECOOP '00 Proceedings of the 14th European Conference on Object-Oriented Programming, Springer London, 2000, pp. 27-43
3. Davis, E.D., Bagozzi, R.P., Warshaw, P.R. Extrinsic and Intrinsic Motivation to Use Computers in the Workplace. Journal of Applied Social Psychology (22:14), 1992, pp. 1111-1132.
4. Gotel, O., Cleland-Huang, J. Requirements Engineering's Next Top Model. In: IEEE Software, vol. 30, no. 6, pp. 24-29, Nov.-Dec. 2013
5. Huang, R. Making Active CASE Tools - Toward the Next Generation CASE Tools. ACM SIGSOFT Software Engineering Notes, Volume 23, Issue 1, January 1998, pp. 47-50

6. Jarzabek, S., Huang, R. The case for user-centered CASE tools. Communications of the ACM, Volume 41 Issue 8, ACM New York, NY, USA, Aug. 1998, pp 93 – 99
7. Kelly, S., Tolvanen, J.-P., Visual domain-specific modeling: benefits and experiences of using metaCASE tools, in: Bezivin, J., Ernst, J. (Eds.), Proceedings of International workshop on Model Engineering, ECOOP 2000.
8. Kuzenkova, A., Deripaska, A, Bryksin, T., Litvinov, Y., Polyakov, V. QReal DSM Platform: An Environment for Creation of Specific Visual IDEs. In: 8th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2013), SCITEPRESS, 2013, pp. 251-257.
9. Lending, D., Chervany, N. L. The use of CASE tools. In Proceeding of SIGCPR '98 Proceedings of the 1998 ACM SIGCPR conference on Computer personnel research, 1998, pp. 49-58
10. Osechkina, M., Litvinov, Y., Bryksin, T. Multistroke Mouse Gestures Recognition in QReal metaCASE Technology. In: SYRCoSE 2012: 6th Spring/Summer Young Researchers' Colloquium on Software Engineering. pp. 194-200
11. Repenning, A., Sumner, A. Agentsheets: A medium for creating domain-oriented visual languages. In: Computer 28, no. 3 (1995): 17-25.
12. Selic, B. What will it take? A view on adoption of model-based methods in practice. Software and Systems Modeling (SoSyM), Volume 11, Issue 4, Springer-Verlag New York, 2012, 513-526
13. Senn, J.A., Wynekoop, J.L. The Other Side of CASE Implementation: Best Practices for Success, Information Systems Management, (12), 1995, pp. 7-14.
14. Литвинов Ю.В. Реализация визуальных средств программирования роботов для изучения информатики в школах // Компьютерные инструменты в образовании, СПб., 2013, № 1, С. 36-45 (in Russian)
15. Птахина А.И. Разработка метамоделирования "на лету" в системе QReal // Список-2013: Материалы всероссийской научной конференции по проблемам информатики. 2013г ., Санкт-Петербург. — СПб.: Изд-во ВВМ, 2012. С. 28-36 (in Russian)
16. Терехов А.Н., Брыксин Т.А., Литвинов Ю.В. QReal: платформа визуального предметно-ориентированного моделирования. // Программная инженерия, 2013, № 6, С. 11-19 (in Russian)