Comparison of Textual and Visual Notations of DOMMLite Domain-Specific Language

Igor Dejanović, Maja Tumbas, Gordana Milosavljević, and Branko Perišić

Faculty of Technical Sciences, University of Novi Sad, Serbia
{igord,majat,grist,perisic}@uns.ac.rs

Abstract. This paper presents a comparison of textual and visual syntax notation of Domain-Specific Language (DSL) programs on the example of DOMMLite DSL[3]. Starting from the definition of DOMMLite meta-model, the prototypes of both textual and graphical editors are implemented using tools of the Eclipse Modeling Project[1]. Initial observations in favor and against both syntax notations are summarized and the impact of the chosen concrete syntax on the team development and version control is analyzed.

Key words: language notations, version control, MDE, DSL

1 Introduction

The most powerful weapon we have to fight ever-increasing complexity of software and supporting hardware architecture is abstraction. Although powerful, used abstractions are usually computer-, i.e. solution space-oriented, as opposed to being application domain-, i.e. problem space-oriented [15]. Developers still need to perform the mental mapping of concepts found in the solution domain to concepts found in the problem domain and to apply these mappings manually during the course of implementation [7]. By leveraging abstraction and providing adequate Domain-Specific Languages (DSLs), we strive to reduce the semantic gap between these two domains and eliminate the need for mapping between them as the ultimate goal.

DSLs in use today are mostly based on textual or graphical notatation or the mixture of those two (e.g. visual language with OCL[2] constraints). One type of concrete syntax is usually supported for the same DSL. In this paper we present some initial observation on the differences between graphical and textual syntaxes and their impact on version control based on our experience in developing both notations for DOMMLite DSL. DOMMLite is a declarative language whose aim is the description of statical properties of database-oriented software applications[3]. It has been designed and implemented using Model-Driven Engineering (MDE). DOMMLite builds on the concepts of other ER-like

[1] http://www.eclipse.org/modeling/

[2] Object Constraint Language – http://www.omg.org/technology/documents/modeling_spec_catalog.htm\#OCL

languages such as UML Class Diagrams[13], Meta-Object Facility (MOF)[12], ECore[4], and concepts expressed in Domain-Driven Design[5].

2 Textual vs. Visual Syntaxes

Textual notation and textual editor for DOMMLite language has been fully developed[3] using openArchitectureware framework[3]. Although both visual notation and visual editor for DOMMLite language are in the early phase of development, we will give some initial findings, based on anecdotal evidence, in favor and against both textual and visual syntaxes and supporting tools.

Arguments for visual notations and editors

- **Model structure is easier to comprehend**. It is a popular belief that the graphical notation is better than the textual. However, it has been reported that in some circumstances graphical notation does not perform better, or even that it performs worse than textual[6, 14]. This is usually reported for the languages used for modeling of control or data flow and for languages where "secondary notation"[14](see section 4) has a major impact on model understandability. Nevertheless, for declarative DOMMLite language which is used to express statical properties of the system, based on anecdotal evidence, we are inclined to think that visual notation perform better at comprehension of element relationships and the overall model structure.
- **Easier model navigation**. Using ubiquitous operations pertinent to modern visual editors (like zooming, panning etc.), every part of a DOMMLite model can be visited quickly and with a minimum of effort.
- **Visual languages are easier to learn**. Arguably, this depends on the experience and background of the modeler and the choice of visual and textual representations of language concepts. However, we argue that the learning curve is steeper for visual notations. Presented with the palette of modeling elements, the modeler can almost immediately start placing them on the drawing canvas and connecting them and learn by the means of trial and error. Conversely, with textual syntax, the modeler is presented with an empty file and it is hard to do anything without learning some elements of the language first (i.e. keywords, syntax and semantic rules etc.).

Arguments against visual notations and editors

- **Hard to develop and maintain**. Although building graphical editors is much easier now with the advent of sophisticated tools like GMP[4], DEViL[5] Tiger[6] and VLDesk[7], the amount of work to develop and maintain a full-fledged graphical editor is still considerable, especially in the environment of evolving languages.

[3] `http://www.openarchitectureware.org/`
[4] `http://www.eclipse.org/modeling/gmp/`
[5] `http://devil.cs.upb.de/`
[6] `http://user.cs.tu-berlin.de/~tigerprj/`
[7] `http://www.dmi.unisa.it/people/risi/www/vldesk/index.html`

– **Serialization format is different from the presentation format**. It can be problematic if the need arises to drill down to the serialized representation, which still is the case with current version control systems (see section 3).

Arguments for textual notations and editors

– **Existing tools can be used as a fall-back option**. A plain-text editor can be used to visualize and edit models based on textual notations.
– **Existing text-based version control systems can be used**. See section 3.
– **Programmers are used to textual syntaxes**. Most programmers are used to textual syntaxes, so it is easier to introduce text-based modeling to current software development processes and practices.

Arguments against textual notations and editors

– **Notation verbosity**. Textual notation, despite all precaution, can become quite verbose. No part of model can be hidden and visualized at the will of a modeler (with the sole exception of code folding support).
– **The structure of the model is harder to comprehend**. The modeler needs to provide substantial effort to build a mental model of relationships among model elements. In DOMMLite, for example, references to other entities and inheritance hierarchy are not easy to convey from textual notation.
– **Navigation is not as intuitive as is the case with visual editors**. Navigation can be performed by scrolling sequentially through the text, searching for text patterns, listing and jumping to all usages of certain model element (i.e. all references) etc. For DOMMLite generated editor, the only overview is presented in the form of a tree-like code outline which conveys only language-level containment structure.

3 Impact of the Type of Concrete Syntax on Version Control

The issue of paramount importance when it comes to team development is version control, i.e. identification, preservation, visualization and merging of model differences. Although there are well established tools and techniques for version control of plain textual artifacts (i.e. source code)[16, 9], the version control in the field of model-driven engineering with emphasis on model syntax and semantics is an active field of research[1, 2, 10].

Traditional text-based systems for version control works at file level and considers content of files as an array of lines of text without trying to utilize language syntax or semantics. Using these tools for version control of models can be troublesome. Even if model are serialized in textual XMI format, it is still unwieldy for a modeler to drop down to the verbose and hardly readable XMI format when trying to do the merge of concurrent changes.

Fig. 1. Three-way compare of concurrent changes using XMI format

Modeling tools are rich in visual hierarchy and graphical representation, which are not found in the linear text files of source code representation[8].

Although models can be serialized to textual (e.g. XML) formats, it is hard to visualize differences and resolve conflicts using this form of representation.

We argue that a carefully crafted textual notation has a positive side-effect of the possibility to utilize existing text-based version control systems until the appropriate model-based tools become available.

For example, figure 1 shows the three-way compare of concurrent changes when a model is serialized using XMI format. The same changes, using DOMMLite textual notation are presented in figure 2. It is evident that verbosity of XMI syntax decreases readability of the model and makes conflict resolution of concurrent changes harder in comparison to the DOMMLite textual notation.

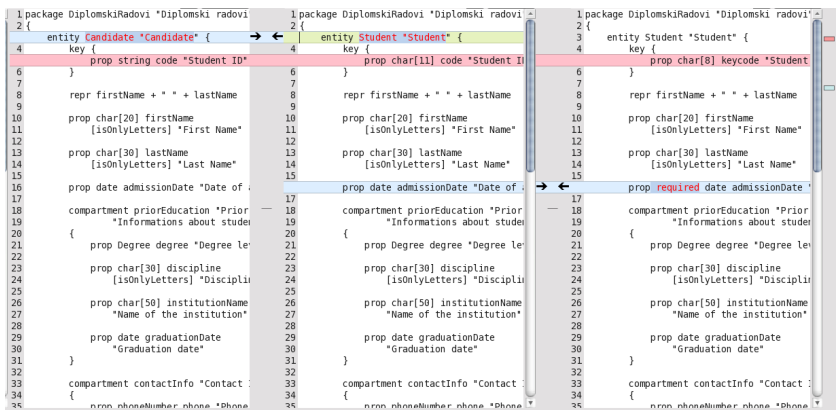

Fig. 2. Three-way compare of concurrent changes using DOMMLite textual notation

4 Related Work

Work related to the topics discussed in this paper includes development of a DSL with various notations and research on the impact of using different language notation on performance of the developers and existing tools utilization.

In [3] DOMMLite language is introduced with it's textual syntax, text-based editor and source code generator.

In [14] the author investigates a so called "secondary notation" (lay-outing, clustering, white-spaces, colors) which is a way that practitioners of visual languages use the non-formal features and techniques to specify information and give hints to the reader.

In [6] authors report comparisons between the comprehensibility of textual and visual programs and paradoxically come to a conclusion that the comprehensibility of the graphical notation is worse than textual.

Visual language and environment for specification of attributed grammars (AG) is presented in [11]. The authors conclude that, by using a visual language, the mental gap between the required text-based AG specification imposed by several compiler generator tools and the habitual way of sketching AGs in the form of a decorated tree has been shortened.

Difficulties and problems related to text-based version control of models are presented in [8, 1].

Native model-based version control approaches are presented in [1, 2, 10]

5 Conclusions and Future Work

There is no definite answer whether textual or graphical syntax is better. There are papers that support either point of view. It is a popular belief that visual languages are easier to understand but, for certain language constructs, it has been empirically proved that comprehensibility can be worse than for textual languages [6]. The graphical notation is also more prone to the "secondary notation" which is not part of the formal system and thus it is left to developers to freely create their own style and ways of encoding additional information and hints. One way to remedy this would be to make elements of "secondary notation" a part of the formal specification.

We have outlined trade-offs between textual and visual notations on the basis of our experience in development of the DOMMLite DSL. If development and maintenance of both syntaxes is not an issue, we suggest using graphical syntax for model overview, navigation and structural changes (e.g. editing relationships) and textual syntax for defining non-structural properties of the model (e.g. for DOMMLite it would be the definition of attribute names and types, constraints etc.). Also, we find that using textual syntax as the canonical serialization format enables utilization of existing, industry proven, text-based version control systems until native model based version control systems mature.

Further research and development will be focused on platforms and tools for supporting development, version control and co-evolution of languages and different concrete syntaxes.

References

1. Alanen, M., Porres, I.: Version control of software models. Advances in UML and XML-based Software Evolution (2005)
2. Altmanninger, K., Kappel, G., Kusel, A., Retschitzegger, W., Schwinger, W., Seidl, M., Wimmer, M.: AMOR–Towards Adaptable Model Versioning. In: 1st International Workshop on Model Co-Evolution and Consistency Management (MODELS '08) (September 2008)
3. Dejanović, I., Milosavljević, G., Perišić, B., Tumbas, M.: A domain-specific language for defining static structure of database applications. Computer Science and Information Systems 7(3), 409–440 (June 2010), http://www.comsis.org/ComSIS/Vol7No3/RegularPapers/paper2.htm
4. Eclipse Foundation: Eclipse Modeling Framework - EMF. Online http://www.eclipse.org/modeling/emf/, http://www.eclipse.org/modeling/emf/, accessed June, 2010
5. Evans, E.: Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley Professional (2004)
6. Green, T., Petre, M.: When visual programs are harder to read than textual programs. In: Human-Computer Interaction: Tasks and Organisation, Proceedings of ECCE-6 (6th European Conference on Cognitive Ergonomics). GC van der Veer, MJ Tauber, S. Bagnarola and M. Antavolits. Rome, CUD. Citeseer (1992)
7. Kelly, S., Tolvanen, J.P.: Domain-Specific Modeling: Enabling Full Code Generation. Wiley-IEEE Computer Society Pr (March 2008)
8. Lin, Y., Zhang, J., Gray, J.: Model comparison: A key challenge for transformation testing and version control in model driven software development. Object Oriented Programing, Systems, Languages and Applications (2004)
9. Mackall, M.: Towards a better scm: Revlog and mercurial. In: Proceedings of the Linux Symposium. vol. 2, pp. 83–90. Ottawa, Ontario, Canada (July 2006)
10. Oliveira, H.L.R., Murta, L.G.P., Werner, C.: Odyssey-vcs: a flexible version control system for uml model elements. In: Proceedings of the 12th International Workshop on Software Configuration Management, SCM 2005. pp. 1–16. ACM, Lisbon, Portugal (September 2005)
11. Oliveira, N., Pereira, M.J.V., Henriques, P.R., da Cruz, D., Cramer, B.: Visuallisa: A visual environment to develop attribute grammars. ComSIS – Computer Science an Information Systems Journal, Special issue on Advances in Languages, Related Technologies and Applications 7(2), 266 – 289 (April 2010)
12. OMG: Meta Object Facility (MOF) Core Specification, Version 2.0 (January 2006)
13. OMG: OMG Unified Modeling Language (OMG UML), Infrastructure, Version 2.1.2 (November 2007), oMG Document formal/2007-11-04
14. Petre, M.: Why looking isn't always seeing: readership skills and graphical programming. Communications of the ACM 38(6), 33–44 (1995)
15. Schmidt, D.C.: Guest editor's introduction: Model-driven engineering. Computer 39(2), 25–31 (2006)
16. The Apache Software Foundation: Apache subversion. Online http://subversion.apache.org/, http://subversion.apache.org/, accessed June, 2010