Definition of Visual Language Editors Using Declarative Languages

Torbjörn Lundkvist

TUCS Turku Centre for Computer Science
SoSE Graduate School on Software Systems and Engineering
Department of Information Technologies,
Åbo Akademi University
Joukahaisenkatu 3–5 A, FIN-20520 Turku, Finland
e-mail:`torbjorn.lundkvist@abo.fi`

1 Introduction

Developing large software systems is a complex process that involves the integration of many different artifacts. Managing all these software artifacts and their complex relationships is a difficult task to carry out and requires sophisticated tool support. Still, to be able to understand all the aspects of a software system, there is a need to represent and visualize the software system at different scales and levels of detail. Creating meaningful visualizations of large amounts of data is a central problem in software modeling. A software model can be used to describe different aspects of a software system, such as the structural and behavioral aspects. A modeling language is a language that defines the concepts that can be used to describe a particular part of a software system. This requires tools for defining, representing and manipulating visual languages.

The Unified Modeling Language (UML) [15] has become the de facto language for software modeling in the software industry. UML is a general-purpose modeling language that can be used for describing a wide variety of application domains. Domain-specific languages are modeling languages targeted at a specific domain. A domain-specific modeling language can be designed to improve the understanding and description of a specific application domain.

In technologies such as Model Driven Engineering (MDE) [8], models rather than actual source code are the primary artifacts of a software system. A software system developed using an MDE approach may consist of several models describing the structure and behavior of the software system, written in several languages, including both general-purpose and domain-specific modeling languages. In order to successfully realize an MDE process, specialized methods and tools for manipulating, analyzing and transforming models and diagrams constructed using several domain specific modeling languages, with complex dependencies and relationships, are required.

In my dissertation research I aim to study techniques that reduce the effort of designing customized interactive visual editors with support for MDE processes. This research aims to show that this can be achieved with an editor architecture constructed based on loosely coupled general components that are configurable for a specific language by using a set of declarative languages.

2 Background and Related Work

The problem of providing tool support for customized visual languages is a well studied problem and there exists many different approaches and architectures, which allow the definition of new visual languages. Examples of these are architectures are DiaGen [9], AToM^3 [5] and GenGed [3] that are based on graph grammars. Among the commercial environments there are MetaEdit+ [7] and the Microsoft DSL Tools [4], and community-based Eclipse Graphical Modeling Framework [6].

One of the main problems with some of the existing tools is that there is not a clear separation between the abstract and concrete syntax. We consider this separation important since it allows multiple representations of the same abstract elements in different diagram types. In addition, we think that the definition of modeling languages and diagrams are tool independent artifacts that should be defined based on common standards, to ensure interoperability between modeling tools. The graph grammar based approaches use a declarative approach for defining transformations used in interactive editors. One of the most important benefits of this approach is that transformation definitions are independent of the underlying implementation, and could therefore be exchanged between different tools. We would like to extend the use of declarative approaches to define an editor architecture configurable using only definitions in declarative languages.

3 Research Problem

The aim of this research work is to investigate how to reduce the effort of designing visual interactive editors that can be customized for several domain-specific visual languages. In the context of this research work, a high level of reuse of configurable general editor components is considered to reduce the effort of designing editors for domain-specific environments. This research work aims to show that this can be achieved by defining a general language independent editor architecture that is configured to a specific language notation by the use of *declarative languages*. A declarative language can be used to describe *what* a system should be like, not *how* to implement it. We believe this brings many benefits, as the information expressed in a declarative language can be reused by many different components in a tool.

The focus of this research work is finding methods that allow the definition of a visual language editor based on declarative languages. This problem can be decomposed into several related areas, including the definition of languages and visual notations, how to edit and manipulate structures expressed in these languages, and the definition of query and model transformation languages.

4 Expected Results and Research Method

This research work will address some of the issues related to the development of configurable interactive editors for visual languages. Such editors are essential in an MDE process, as there is a need to rigorously define new modeling languages that have an abstract and a visual notation in the form of diagrams, to support both human users and computer programs, such as various model analyzers and model transformation tools. Such tools need to be based on standards, to ensure interoperability and long-term support.

In this research project we already started to tackle the problem of defining diagrams based on the abstract syntax, based on the OMG [13] standards for diagram interchange [16] and modeling language definition [17]. We expect that this work is significant since such a mechanism targeted at the OMG standards have been lacking, and is required in order to create diagrams that can be interchanged between modeling tools. In addition, our preliminary results, discussed in more detail in Section 5, show that this diagram definition mechanism can be used to automate the reconciliation of diagrams based on changes in the abstract models. Future research based on this work will include investigating how these diagram definintions can be reused or extended to be used in other components of an interactive editor for visual languages. Examples of such possible extensions are the inclusion of layouter constraints and renderer definitions. Since diagrams are important means for human users to manipulate models, diagram definitions could also include rules that declare the behavior and interactions of a diagram editor.

In our research, we like to base the representation and manipulation of models primarily on graphs and graph transformation, since this gives a solid foundation for designing model query and transformation components in a modeling tool. Model query is a fundamental element for model transformation components. We believe that such components should be generic and reusable for several languages and several purposes, to define large sequences of transformations in a MDE process and small transformations that are executed interactively on the abstract model data in a diagram editor. In this research work we are also studying the definition of declarative query and model transformation languages. We have in previous research addressed the need for a *star operator* [11] in query languages for describing recursive or hierarchical structures in query patterns. In our future work we plan to investigate the need for other graph matching operators and how to use the star operator in a model transformation language.

As an important result of this research work, we aim to define an architecture for defining interactive editors for visual languages that are configured using declarative languages. This architecture will be based on standards, which we believe will help in ensuring interoperability between tools. Although this architecture is targeted towards domain-specific languages, we believe the presented approach can also successfully be applied to general-purpose languages as well. There are plans for publishing a deeper analysis of this architecture and how it can be used to define a customized editor environment based on declarative languages.

This research work will be carried out using a practical approach based on a theoretical foundation to address the issues raised in this research plan. This means that proposed solutions will also be evaluated using a reference implementation in a modeling tool.

5 Preliminary Results and Evaluation

In this section we will present and discuss some of the preliminary results that are related to this research project.

The Object Management Group (OMG) [13] maintains a series of modeling standards such as the UML. The abstract syntax of a modeling language can be defined using the Meta Object Facility (MOF) [17]. To represent and interchange diagrams, OMG provides the UML 2.0 Diagram Interchange (DI) [16]. However, the OMG standards do not specify the relation between the abstract and concrete syntax. This information is needed to be able to construct new diagrams in the DI standard and to ensure interoperability between different modeling tools. The OMG has recently published a request for proposals for a *model view to diagram* language [14].

To address this issue, we have introduced the Diagram Interchange Mapping Language (DIML) [1]. DIML is a language that defines the relation between the abstract syntax of a modeling language to the concrete syntax as DI Diagrams as a set of *mappings*. A DIML mapping declares which diagrams are possible to construct based on the abstract models. In [12], DIML mappings for a subset of UML 1.4 can be found. Perhaps the most important application of DIML is a *diagram reconciliation component*. Diagram reconciliation is based on the idea that after a model transformation component has been executed and modified the abstract models, an independent diagram reconciliation component analyzes the changes made in the abstract model, and calculates which parts of the existing diagrams have been invalidated by the changes and needs to be updated. Using this technique, it is possible to decouple transformations made on abstract models from the necessary updates on diagrams, simplifying model transformation rules that need to preserve the consistency of existing diagrams.

Query languages are used to find parts of a model that fulfill some given constraint, and are important when defining model transformation languages. In [11], we present an approach for matching recursive or hierarchical structures based on an extension of *subgraph isomorphism*, by introducing the concept of a *star operator* in a query language. The star operator resembles the Kleene star operator and allows a part of a pattern to match repeatedly zero or more times to a target graph. We are currently investigating the definition of a model transformation language based on this query language, and we believe that the star operator increases the expressiveness of model transformation definitions.

In [10], we present a tool environment for defining peripherals for mobile phones called MICAS. This environment was built as a customization of the Coral Modeling Framework [2], including a domain-specific modeling language with a concrete syntax defined with DIML, a constraint evaluation component to detect potential violations of *well-formedness rules*, a model transformation engine to perform PIM to PSM transformations and a component for generating code for a simulation environment. We believe the MICAS tool is an important study as it highlights the need for domain-specific editors that support MDE processes.

The work presented in this research plan is currently in its second year out of a four year period. We believe that as a result of this research project we will find methods for defining the most essential components of an editor for visual languages entirely using artifacts defined using declarative languages. We have presented some preliminary results at international conferences and in international journals and we plan to continue publishing results of this research for rigorous review. The ideas described in the preliminary results have been implemented within the Coral Modeling Framework, available at `http://mde.abo.fi`.

References

1. Marcus Alanen, Torbjörn Lundkvist, and Ivan Porres. Creating and Reconciling Diagrams After Executing Model Transformations. *Accepted for publication in Elsevier journal Science of Computer Programming*, 2007.
2. Marcus Alanen and Ivan Porres. The Coral Modelling Framework. In Johan Lilius Kai Koskimies, Ludwik Kuzniarz and Ivan Porres, editors, *Proceedings of the 2nd Nordic Workshop on the Unified Modeling Language NWUML 2004*, number 35 in TUCS General Publications. TUCS Turku Centre for Computer Science, Jul 2004.
3. R. Bardohl, H. Ehrig, J. de Lara, and G. Taentzer. Integrating Meta Modelling with Graph Transformation for Efficient Visual Language Definition and Model Manipulation. In Springer, editor, *Proceedings of the Fundamental Aspects of Software Engineering (FASE)*, pages 214–228, 2004.
4. Microsoft Corporation. Microsoft Domain-Specific Language Tools. Available at http://msdn.microsoft.com/vstudio/DSLTools/.
5. Juan de Lara Jaramillo, Hans Vangheluwe, and Manuel Alfonseca Moreno. Using Meta-Modelling and Graph Grammars to Create Modelling Environments. *Electronic Notes in Theoretical Computer Science*, 72(3), 2003.
6. The Eclipse Graphical Modeling Framework website. `http://www.eclipse.org/gmf`.
7. Steven Kelly. Comparison of Eclipse EMF/GEF and MetaEdit+ for DSM. In *19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications, Workshop on Best Practices for Model Driven Software Development*, October 2004.
8. Stuart Kent. Model Driven Engineering. In *Proc. of IFM International Formal Methods 2002*, volume 2335 of *LNCS*. Springer-Verlag, 2002.
9. Oliver Köth and Mark Minas. Structure, Abstraction, and Direct Manipulation in Diagram Editors. *LNCS*, 2317:290–304, 2002.
10. Johan Lilius, Tomas Lillqvist, Torbjörn Lundkvist, Ian Oliver, Ivan Porres, Kim Sandström, Glenn Sveholm, and Asim Pervez Zaka. An Architecture Exploration Environment for System on Chip Design. *Nordic Journal of Computing*, 12(4):361–378, 2005.
11. Johan Lindqvist, Torbjörn Lundkvist, and Ivan Porres. A Query Language With the Star Operator. In *Proceedings of the 6th International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2007)*, volume 6(2007) of *Electronic Communications of the EASST*, Braga, Portugal, March 2007. EASST.
12. Torbjörn Lundkvist. Diagram Reconciliation and Interchange in a Modeling Tool. Master's Thesis in Computer Science, Department of Computer Science, Åbo Akademi University, Turku, Finland, November 2005.
13. Object Management Group website. `http://www.omg.org/`.
14. OMG. MOF model view to diagram request for proposals. OMG Document ad/2006-11-07. Available at www.omg.org.
15. OMG. UML 2.0 Superstructure Specification, August 2003. Document ptc/03-08-02, available at `http://www.omg.org/`.
16. OMG. Diagram Interchange version 1.0, April 2006. OMG document formal/06-04-04. Available at `http://www.omg.org`.
17. OMG. Meta Object Facility (MOF) Core Specification, version 2.0, January 2006. Document formal/06-01-01, available at `http://www.omg.org/`.