Generating Model-Specific Editors for MDA

Anna Gerber[1] and Michael Lawley[1]

Co-operative Research Centre for Enterprise Distributed Systems (DSTC),
University of Queensland, Brisbane 4072, Australia
{agerber, lawley}@dstc.edu.au

Abstract. We propose a model-based approach for the generation of graphical tools for MDA. Our approach is to define the appearance and behaviour of graphical editors by specifying mappings from MDA-related domain model constructs to graphical representations and constraints on those representations. From these mappings, we may generate the view and controller aspects of Model-View-Controller-based editors. In this paper, we discuss the merits of this tool-focused approach and describe how our prototype, JANE, generates graphical editors for Eclipse Modelling Framework (EMF) models.

1 Introduction

The OMG's Model-Driven Architecture (MDA) provides a framework for the generation of information systems through transformation and weaving of Platform Independent Models (PIMs) that represent various aspects of each system. While UML is often used to model these system aspects in light-weight approaches to MDA, the focus is shifting to domain-specific modelling for PIMs. In this approach, each platform independent metamodel can be considered analogous to an abstract syntax for a domain-specific language. The EDOC [2] standard, with domain-specific metamodels for component collaboration and business process modelling, is an example of the new wave of standards coming from the OMG that are shaping MDA in this way. Thus, model-based domain-specific languages and tools to support their definition and use play an important role in MDA.

Technologies such as the Human Useable Textual Notation (HUTN) [3] provide mechanisms for the specification of model-based textual languages and tools like TokTok [4], for the manipulation of instances of those languages. However, many of the models used in MDA have graphical or diagrammatic notations in addition to or instead of textual notations. Even for models without an existing concrete syntax, a diagrammatic notation is natural because their graph-like structure usually maps more directly to a graphical notation than to linear text. Thus, we identify a need for tools to allow model-based languages to be represented and manipulated via graphical tools.

In this paper, we propose a tool-focused approach for the generation of model-specific graphical editors for MDA. Our approach is based on MDA principles:

We use models to represent the model, view and controller aspects of each graphical tool, and use mappings (or transformations) between these models to express how to parameterise the code generation performed by our prototype editor generator, JANE. We discuss the benefits of focusing on tool issues over a purely language-based approach in Section 2. Then, in Section 3, we describe how we realise these benefits through our prototype.

2 A Tool-Focused Approach for Model-Specific Editors

Initially, we focused on specifying graphical notations for models, with model-specific editors being artefacts generated using existing Domain-Specific Language (DSL) tools such as Diagen [6] and Grace [5]. Editors generated using these tools provided basic model editing functionality, however, prototyping indicated that in order to generate tools flexible enough to be used within the context of MDA, some degree of parameterisation of the generation is required. This is primarily because graphical languages that are used within the context of MDA are almost always used within the context of tools, and hence need to take into considerations issues related to tool-based graphical languages, while many graphical DSL tools tend to be language-focused; defining graphical languages of the type that people might draw on flip-charts, or whiteboards, and do not focus on tool-related issues.

2.1 Characteristics of existing tools

Typically, graphical languages considered by existing DSL tools are small, purpose-built languages that represent a single aspect or facet of a problem domain. Many of the tools generated by such approaches are intended to be used by a mostly homogeneous user base, with common expectations of the UI. There is little room for customisation of the UI of these generated tools.

Such tools often share the following characteristics:

- Based on graph grammars, not models in the MDA sense.
- Creation of diagrams or language instances is the main goal of generated tools: Integration or interoperability with other tools or other languages is typically not a consideration.
- Grammars that define the graphical languages represented by such tools can be considered analogous to a model, however they are languages that are isomorphic with their representation.

2.2 MDA Tool Considerations

When generating tools for model-specific languages within the context of MDA, we believe that the following considerations need to be made:

Conformance to standards: The MDA vision relies on a number of core technologies from the OMG. These include the Meta Object Facility (MOF) [8], XML Metadata Interchange (XMI) [7] and MOF Queries/Views/Transformations (QVT) [9]. Hence, support for thse standards is vital for model-specific editors for MDA, so that models and language definitions can be exchanged between MDA tools and so that MDA-based generation techniques can be applied to the models produced by these editors.

Tool evolution: One of the main goals of MDA is to facilitate evolution of systems based on a model-driven approach. The same principles apply to evolution of the tools used to manipulate MDA-based languages themselves. As the underlying models change, or as requirements for the MDA tools change, it should be possible to use MDA techniques such as model transformation to enact the appropriate changes in the model-specific tools.

Metamodel expressiveness: The MOF metamodel includes concepts such as *containment*, *reference*, *association*, and *cardinality constraints* that can be valuable in selecting appropriate visual representations and controllers for an editor. We want to be able to use the same concepts with the same level of expressive power from the modelling framework used to define each domain-specific model to also define the notation for that model. In a generalised graph-based metamodel these concepts generally do not exist, and so are not available for use in generation of an editor.

Reuse: The graphical languages used to represent models within MDA tools will often reuse graphical primitives and relationships for similar or related concepts. For example, UML reuses the rectangular notation used for its static structural concepts (Class, Interface, Datatype) for instances in object diagrams, collaboration diagrams and interaction diagrams. Such reuse will usually occur in the presence of inheritance, however, it may also occur due to other relationships in the metamodel, or due to meta-relationships such as the concept of *containment*.

Scalability: A scalability concern that arises for all graphical tools is how to deal with large model instances. Visual representations may need to be split into parts in order to be able to be displayed, and some objects may need to appear more than once for presentation or layout reasons.

Non-isomorphic representations: Sometimes the graphical language defined for a model is not isomorphic with the model. For example, a single icon in the graphical language may indicate the presence of a large chunk of *boiler plate* in the model. This may be because the model has been retrofitted to the language, as may be the case when an established graphical language is modelled for the purposes of using it within the context of MDA. Alternatively, it may be because the model is designed for more than one purpose, and the graphical notation corresponds to only a subset of those purposes.

Decoupling of views: A related issue is that although a language may correspond closely to its model, there may be several views on the same model (for example, outline vs expanded vs properties view). Views may hide some detail for each object, may represent only a subset of a model, or may em-

phasise particular parts of the model, especially those likely to be changed most frequently by the intended users of the tool.

Customisation of View and Control There may be many different users of an MDA-related language, with different requirements for model-specific tools. Hence, it may be necessary to generate multiple configurations of a particular tool, such as an editor, for a given model-based language. Typically this will require some customisation of both the View and the Control within such tools. The MDA way is to generate all configuration of a particular tool from a model, using parameterisation to configure these aspects of the generated tools.

Extended command set: DSL tools typically provide commands based on creating instances of a language, usually based on the structure of the language, for example, to create or delete nodes or create, re-target or delete links between nodes (ie editing commands). In order to generate tools other than editors, we will need to be able to parameterise MDA-tool generation from models to include extended command sets. (Note that some existing DSL tools, it is already possible to define some additional commands.)

Domain model vs MVC model: The models that are manipulated by and interchanged between MDA tools may differ from the model implemented as the model in the MVC sense. This is because typically the M in MVC represents state that will be persisted as domain model instances as well as transient state that is used by the tool, often for the purposes of layout. MDA models to be persisted by generated tools may require addition of default values, or information that is tool-specific and not part of the actual MDA model will need to be persisted in such a way so that it can be restored, but does not pollute the MDA-model instances. Approaches to represent this additional information include persisting it as a model in its own right (for example, layout models), or to use different XML namespaces within an XMI representation of a model instance this extra information.

While existing MetaCASE tools demonstrate many of the features described in this section, we believe that the first three considerations are particularly important for seamless integration of domain-specific modelling tools with existing MDA standards and practises. Through our prototype editor generator, JANE, we aim to address these considerations by providing specific support for MDA-enabling technologies and tool evolution through use of MDA techniques.

3 JANE: A Model-Specific Editor Generator

We have developed a prototype, JANE, to demonstrate how MDA techniques can be used to generate customised model-specific editors. JANE is a plug-in for the Eclipse platform, which generates plug-ins based on the Eclipse Modeling Framework (EMF) [10] and the Eclipse Graphical Editing Framework (GEF) [11]. JANE uses EMF's JET framework, described in [12] to generate code representing the view and controller classes for a particular model, based on the

input mappings. Currently, the input is very basic, and the languages generated by JANE are based on boxes and lines, much like a UML Class diagram. However we are expanding it to take models of MVC as parameterisations to the generation of editors. These MVC models will eventually be mapped from (domain) MDA models using QVT [9].

Our project is investigating how to model different types of controllers and to identify features that may be parameterised in the generation of graphical editors for model-specific notations. The controller model is being based on a survey of existing tools and paradigms for the manipulation of graphical notations. The JANE prototype is currently being modified to allow mappings between the domain model and the controller model to be used as additional inputs to the editor generation.

4 Future work

JANE is currently focused on single-model-based editors. We intend to investigate how to generate tools that make use of multiple model-based languages. An example is a graphical model transformation tool. Such a tool could potentially make use of several graphical languages - the languages used to represent the source model(s), languages used to represent the target model(s) and a graphical representation of the transformation between them.

We also intend to evaluate and test the capabilities of the prototype. We will create test cases for JANE based on large domain models with corresponding graphical notations. The generated editors can then be acceptance tested by users. The default 'mapping' that is currently hardcoded in JANE will also be improved and formalised, so that editors can be generated for models without an existing graphical notation.

5 Concluding Remarks

Using the approach described in this paper, we automate the generation of model-specific editors for use within MDA. One of the goals of MDA is to simplify system evolution by evolving the models that specify the structure and behaviour of a system, and then using transformation and code generation techniques, to evolve the system itself. As model-based languages evolve, the tools that manipulate them must also evolve rapidly, while retaining customisations to make manipulating model instances with the tools easier for their intended users. Our approach brings us closer to this goal and demonstrates how MDA techniques can be used to achieve it.

6 Acknowledgements

The work represented in this paper has been funded in part by the Co-operative Center for Enterprise Distributed Systems Technology (DSTC) through the Aus-

tralian Federal Government's CRC Programme (Department of Education, Science and Training).

References

1. OMG Architecture Board MDA Drafting Team: Model Driven Architecture - A Technical Perspective. OMG Document: ormsc 01-07-01, July 2001.
2. OMG. UML Profile for EDOC (EDOC). OMG Document: formal/04-02-01.
3. OMG. Human-Usable Textual Notation (HUTN). OMG Document: formal/04-08-01.
4. DSTC: TokTok - The Language Generator. http://www.dstc.edu.au/Research/Projects/Pegamento/TokTok/index.html
5. Klein, G.: Generating graphical editors for graph-like datastructures. Chair of Computer Science II, Technische Universitat Munchen, 1999. http://www.doclsf.de/grace/index.html
6. Minas, M.: Visual specification of visual editors with DiaGen. Proc. Application of Graph Transformations with Industrial Relevance (AGTIVE'03), pages 407-414, Sept./Oct., 2003, Charlottesville/Virgina, USA.
7. OMG. XML Metadata Interchange. OMG Document: formal/2003-05-02.
8. OMG. Meta Object Facility (MOF) v 1.4. OMG Document: formal/2002-04-03, April 2002.
9. OMG Request for Proposal: MOF 2.0 Query / Views / Transformations RFP, OMG Document: ad/02-04-10, April 2002.
10. Budinsky, F., Steinberg, D., Merks, E., Ellersick, R., Grose, T.: Eclipse Modeling Framework. Addison-Wesley (2003)
11. Graphical Editing Framework. http://www.eclipse.org/gef
12. Moore, W., Dean, D., Gerber, A., Wagenknecht, G., Vanderheyden, P.: Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework. IBM Redbook SG24-6302-00 (2004)