Implementing a MOF-Based Metamodeling Environment Using Graph Transformations

Matthew J. Emerson
(Vanderbilt University, USA
Institute for Software Integrated Systems (ISIS)
mjemerson@isis.vanderbilt.edu)

Janos Sztipanovits
(Vanderbilt University, USA
Institute for Software Integrated Systems (ISIS)
sztipaj@isis.vanderbilt.edu)

September 28, 2004

Abstract

Versatile model-based design demands languages and tools which are suitable for the creation, manipulation, transformation, and composition of domain-specific modeling languages and domain models. The Meta Object Facility (MOF) forms the cornerstone of the OMG's Model Driven Architecture (MDA) as the standard metamodeling language for the specification of domain-specific languages. We have implemented MOF v1.4 as an alternative metamodeling language for the Generic Modeling Environment (GME), the flagship tool of Model Integrated Computing (MIC). Our implementation utilizes model-to-model transformations specified with the Graph Rewriting and Transformation toolsuite (GReAT) to translate between MOF and the UML-based GME metamodeling language. The technique described by this paper illustrates the role graph transformations can play in interfacing MIC technology to new and evolving modeling standards.

1 Introduction

Model-based design has over the last several years grown into an important trend in software and systems engineering. The central vision of the OMG, Model

Driven Architecture (MDA), describes a platform-independent approach to the development of domain-specific applications. It advocates the specification of software systems through modeling and model transformation [1]. MDA builds upon OMG's widely-used UML, which provides a common graphical syntax for object-oriented design. MDA incorporates modeling into every stage of the software development process by describing this process as a sequence of transformations among models.

One theme of MDA is that UML will be the single, universal, platform-independent modeling language used by model translators to generate software artifacts for specific platforms. The basis of this conviction stems from viewing model-based design in the same light as conventional programming, where language standardization has been an important issue. However, the scope of model-based design is in fact much broader. Model-based design encompasses the entire modeling process, which inherently includes the selection of essential domain aspects, careful separation of the modeled and not modeled worlds, and abstraction. Because specifying a universal language which is broad enough to cover all conceivable systems would be extremely difficult, reasonable solutions depend on the use of domain-specific modeling languages (DSML-s).

A step in the right direction is to use a modeling language that is both universal and extendable – an approach captured by the UML profile mechanism. Unfortunately, stereotyping does not change the fundamental syntactic and semantic properties of the modeling languages and tends to create a complex web of interfering standards.

The more radical approach of constructing DSML-s demands an understanding of the fundamentals of constructing modeling languages and creating standards and tool suites for facilitating their specification and composition. The theme of this approach is metamodeling. A metamodel is a model of a DSML expressed using some metamodeling language. The latest developments in UML 2 [2] depend on this approach, as UML 2 has been defined using the Meta Object Facility (MOF). MOF has emerged as the OMG's standard metamodeling language, and one of the MOF use-cases [3] is the specification of DSML-s. In the future, MOF may serve as a widely-adopted tool-independent metamodeling language, allowing model data to be freely transfered between compliant tools using OMG's XML Metadata Interchange (XMI) technology [4].

We must also consider the demand for powerful tool suites which aid in specifying, manipulating, transforming, and composing models [5]. Model Integrated Computing (MIC) is a comprehensive approach to model-based design. In MIC, a modeling language and model interpreter tools are developed for a given domain. Then, the domain-specific language is used for creating and evolving a computer system through modeling and model interpretation [6]. Over the last ten years, MIC metamodeling approaches have been successfully applied in a variety of application domains [7] [8]. The primary MIC development tool is the Generic Modeling Environment (GME), a metaprogrammable model builder for designing and modeling in domain-specific modeling environments. GME supports its own metamodeling language based on UML Class Diagrams with Class stereotypes and OCL constraints called MetaGME. However, as MOF becomes the widely-adopted, industry-standard metamodeling language, GME can evolve to support tool-independent MOF-based metamodels while also maintaining compatibility with technologies based on its own tool-specific metamodeling language.

Using metamodeling and metamodel-based model transformation, we have implemented a MOF v1.4-based alternative metamodeling environment for GME. Our transformation allows the new MOF metamodeling environment to leverage existing tool support for GME modeling environment generation. Our implementation of the MOF sits as an additional layer of abstraction above the existing GME-specific metamodeling facilities.

2 The Generic Metamodeling Environment

GME is the flagship tool of MIC. It provides a graphical UML-based metamodeling language capable of expressing the concrete and abstract syntax of a target graphical modeling language. OCL constraints specify metamodel well-formedness rules. GME provides library import and export facilities and custom model visualizations. There are also facilities for plugging in analysis, verification, and translation tools which interpret domain-specific models. GME's metamodeling language, called MetaGME, predates the OMG's adoption of the MOF specification. It utilizes UML stereotypes to imply the abstract syntax expressed by the metamodel [9]. The meanings of the stereotypes used in MetaGME are:

- *Models* are compound objects which are visualized in GME as containing other model elements.
- *Atoms* are elementary objects which are not visualized in GME as containing other model elements.
- *FCO-s* are first-class objects which must be abstract but can serve as the base type of an element of any other stereotype.
- *References* correspond to pointers in an object-oriented programming language.
- *Connections* are analogous to UML Association Classes.

- *Aspects* provide logical visibility partitioning to present different views of a model.

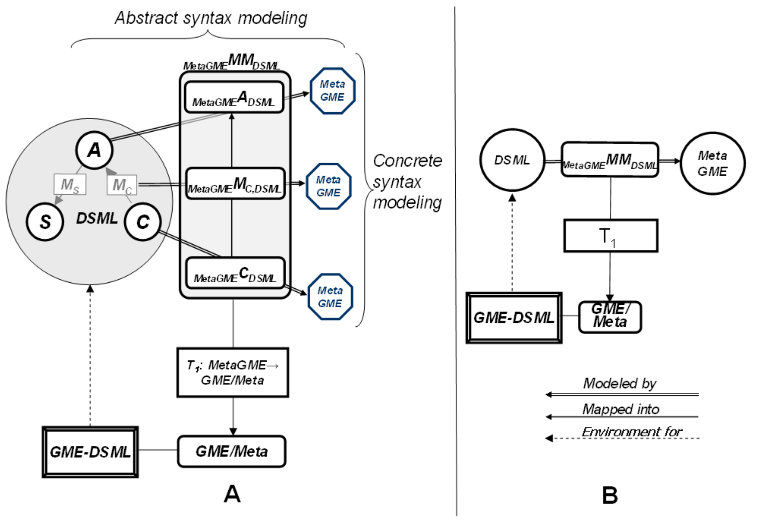

Figure 1: **A)** Metamodeling with GME **B)** Simplified Diagram

GME-based metamodeling is demonstrated in Figure 1. The metamodel ${}_{MetaGME}MM_{DSML}$ of a DSML consists of the abstract syntax ${}_{MetaGME}A_{DSML}$, concrete syntax ${}_{MetaGME}C_{DSML}$, and syntactic mapping ${}_{MetaGME}M_{CDSML}$ specified using the UML constructs of MetaGME. The ${}_{MetaGME}MM_{DSML}$ metamodel is translated by the T_1 translator (called the meta-interpreter) into a configuration file for GME (represented in Figure 1 by the box labelled "GME/Meta"). Using this configuration file, GME can also function as the domain-specific modeling environment for the metamodel domain. (A simplified diagram of the metamodeling process can be seen in Figure 1.B)

There exists a large body of existing GME-based DSML-s [10] [11]. There also exist a number of related modeling tools, including the model transformation tool GReAT [12]. The Graph Rewriting and Transformation toolsuite is a DSML implemented for GME that enables the graphical specification of graph transformation algorithms with formal execution semantics. Because GME essentially represents models as vertex- and edge-labelled multi-graphs where the labels denote the corresponding types defined by some metamodel, GReAT can be used for model-to-model transformation. This paper describes our work to update the MIC metamodeling facilities to incorporate the MOF standard in addition to the UML/OCL-based MetaGME. Because of the large volume of existing DSML-s and tools which depend on the existing GME meta-language, a wholesale replacement of MetaGME with MOF is not desirable. Instead, we use model-to-model transformation technology to implement MOF as an optional layer of abstraction over MetaGME.

3 MOF Metamodeling Constructs

As defined in the v1.4 specification [3], MOF provides the following five basic object-oriented concepts for use in defining metamodels. Our implementation includes each of these constructs. Furthermore, we have specified a mapping between these concepts and the MetaGME concepts outlined above.

- *Classes* are types whose instances have identity, state, and an interface. The state of a Class is expressed by its Attributes and Constants, and its interface is governed by Operations and Exceptions.
- *Associations* describe binary relationships between Classes. They may express composite or non-composite aggregation semantics. Because MOF Associations have no object identity, they lack both state and interface.
- *DataTypes* are types with no object identity. By design, the different MOF DataTypes encompass most of the CORBA IDL primitive and constructed types.
- *Packages* are nestable containers for modularizing and partitioning metamodels into logical subunits. Generally, a non-nested Package contains all of the elements of a metamodel.
- *Constraints* specify the well-formedness rules that govern valid domain models.

4 Implementing MOF for GME

We have implemented a MOF-based metamodeling environment for GME. Our implementation leverages the existing GME metamodeling language and meta-interpreter for the generation of new GME configurations.

As described previously, a complete GME metamodeling language consists of two parts. Part one is a graphical metamodeling environment for the specification of the abstract syntax, concrete syntax, and syntactic mappings. Part two is a translation tool capable of generating from the graphical metamodel of a target domain the configuration file that enables GME to serve as the domain-specific modeling environment for that domain. MetaGME is itself a metamodeling language with sufficient expressive power to fully model MOF. So, while the MOF specification uses MOF to model itself, we have expressed the MOF model with the language of MetaGME. Furthermore, because MetaGME already reflects the full range of configurations which can be realized by GME[1], we found the easiest way to create the necessary translation tool by defining a transformation algorithm from MOF-specified metamodels into MetaGME-specified metamodels.

Having transformed a MOF-specified metamodel to a Meta-GME metamodel, we take advantage of MetaGME's existing meta-interpreter to generate the GME configuration file. The shaded components in Figure 2 represent the facilities we have to provide in implementing MOF for GME: ${}_{MetaGME}MM_{MOF}$ is our MetaGME-specified MOF model and T_2 is our implementation of the transformation algorithm from MOF-specified metamodels to MetaGME-specified metamodels. T_1 is Meta-GME's meta-interpreter, which generates the GME configuration files from the translated metamodels.

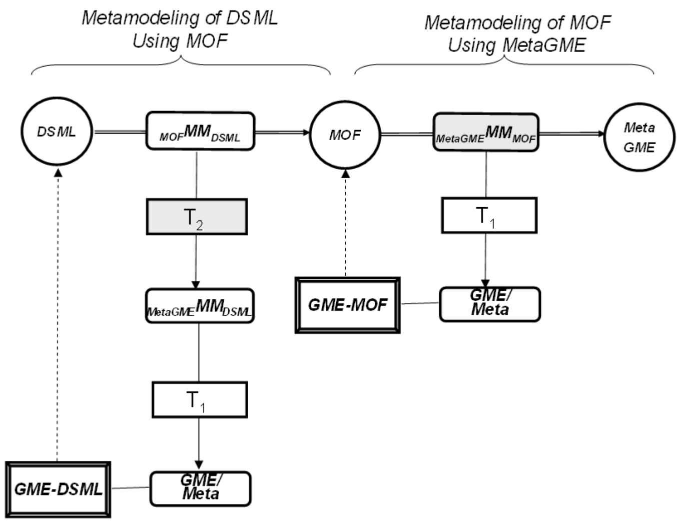

Figure 2: Building the MOF-Based Metamodeling Environment

4.1 The MOF-MetaGME Transformation

The transformation algorithm is quite straightforward because the modeling concepts of both MOF and MetaGME are heavily based on the modeling concepts of UML Class Diagrams. GReAT was the natural choice for implementing our metamodel translation algorithm - we were able to easily and rapidly create our metamodel translation tool. The tool itself is easy to analyze, maintain, and evolve.

[1]In fact, the meta-information in the configuration files directly parallels the MetaGME modeling concepts.

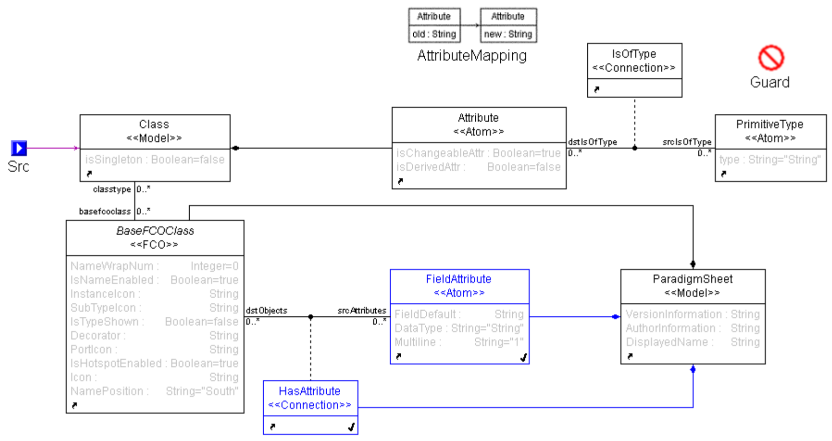

Figure 3: MOF Primitive-Typed Attributes Mapped to MetaGME FieldAttributes

We refer to this tool as MOF2MetaGME.

GReAT transformations are composed of a series of rules which are applied in order to an input model. Each rule attempts to find some pattern in the input model and, if successful, generate some corresponding pattern in an output model. Although we do not wish to delve into the details of GReAT, we explain one example MOF2MetaGME rule visualized in Figure 3.

Figure 3 displays the MOF2MetaGME tranformation rule responsible for mapping String-, Integer-, and Double-typed MOF Attributes into MetaGME Field Attributes. The black Classes represent model patterns which the rule attempts to match, and the blue Classes are those which are to be created in the output model if a match is found[2]. The rule finds any MOF Class containing an Attribute with an IsOfType connection to a PrimitiveType. The guard ensures that only String, Integer, or Double PrimitiveTypes are matched. If such a Class exists, then the rule finds the corresponding MetaGME Class and gives it a Field Attribute of the same type as the matched MOF Attribute.

Table 1 specifies the mapping of MOF constructs to corresponding MetaGME constructs performed by MOF2MetaGME. Notice that MOF Classes and non-aggregate Associations may map to multiple MetaGME constructs. Our implementation of MOF allows metamodelers to specify how each MOF Class or non-aggregate Association should be transformed.

We provide Figures 4 and 5 to illustrate the full function of MOF2MetaGME. Figure 4 is a small part of a MOF-based implementation of UML Class Diagrams for GME used as the input to MOF2MetaGME, and Figure 5 is the corresponding output produced by MOF2MetaGME. Note the high degree of symmetry between

[2]For those viewing this article without color, the blue Classes are HasAttribute and FieldAttribute, while the rest of the Classes are depicted in black.

MOF Construct	**MetaGME Construct**
Package	ParadigmSheet
Class	FCO, Atom, Model, Set, or Reference
Non-Aggregate Association	Connection, SetMembership, or ReferTo
Aggregate Association	Containment
Boolean Attribute	BooleanAttribute
Integer Attribute	Integer FieldAttribute
Double Attribute	Double FieldAttribute
String Attribute	String FieldAttribute
Constraint	Constraint

Table 1: MOF Construct to MetaGME Construct Mapping

the two diagrams.

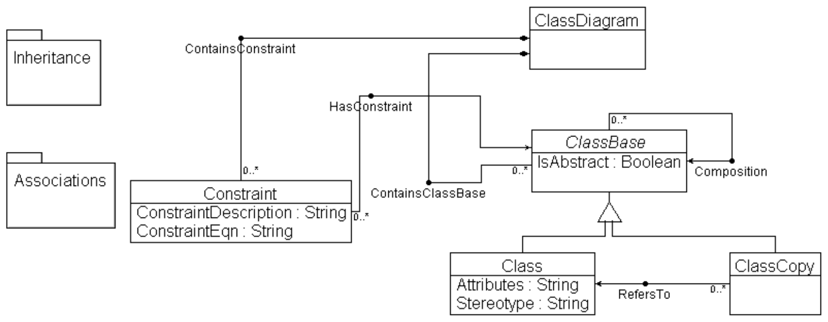

Figure 4: UML Class Diagrams in MOF

4.2 Limitations to our MOF-MetaGME Transformation

Our translation from MOF into MetaGME is not isomorphic – MOF provides some capabilities that MetaGME lacks (and vice-versa). MOF allows a wider range of potential attribute types, the concepts of derived attributes and associations, singleton classes, and classifier-scoped attributes. None of these concepts maps well into MetaGME. MetaGME provides facilities for multi-aspect modeling, concrete syntax specification, and some abstract syntax identifiers which carry special meaning for GME. MOF captures the light-weight Tag extension mechanism which was used to include this GME-specific information in a MOF metamodel. In our implementation of MOF for GME, we have augmented MOF Classes, Associations, Packages, Constraints, and Attributes with

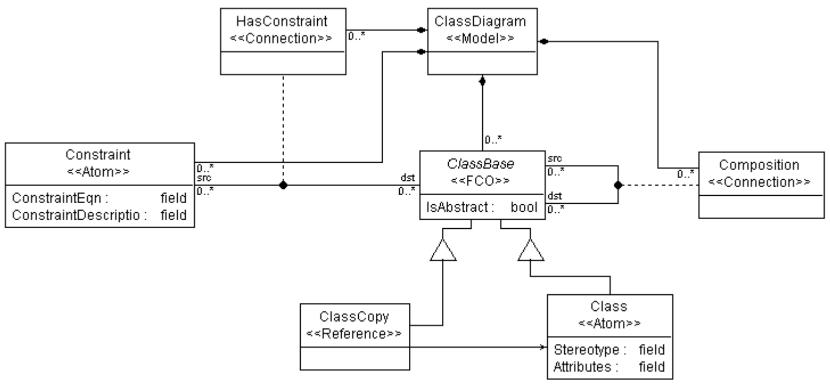

Figure 5: UML Class Diagrams in MetaGME

additional attributes to specify this GME-specific information to facilitate mapping into MetaGME, but these additional attributes may be conceptualized as MOF Tags.

5 Conclusions

A primary insight gained through this work is the recognition that versatile modeling tools like GME need not support MOF as the native metamodeling language in order to be MOF compliant. Using model-to-model transformation, we can take tool-independent metamodels defined using MOF 1.4 and re-interpret them as metamodels in the tool-specific metamodeling language used by GME. This especially useful because the MOF itself lacks some facilities that are necessary for graphical modeling in a tool like GME, including standard mechanisms for describing model concrete syntax. Thanks to the powerful model-to-model transformation technology realized by GReAT, we were able to retain the easy GME modeling environment specification capabilities of MetaGME while also supporting a different industry standard. This work underlines the versatility of metaprogrammable modeling tools such as GME.

References

[1] Object Management Group, “The model driven architecture, 2002”. [Online]. Available: http://www.omg.org/mda

[2] Object Management Group, “Unified Modeling Language: Superstructure version 2.0, 3rd revised submission to OMG RFP”, February 2003. [Online]. Available: http://www.omg.org/docs/ad/00-09-02.pdf

[3] Object Management Group, “Meta Object Facility Specification v1.4”, April 2002. [Online]. Available: http://www.omg.org/docs/formal/02-04-03.pdf

[4] Object Management Group, “XML Metadata Interchange (XMI) Specification v2.0”, May 2003. [Online]. Available: http://www.omg.org/docs/formal/03-05-02.pdf

[5] Greenfield J., Short K., “Software Factories: Assembling Applications with Patterns, Models, Frameworks and Tools”, 2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture, 2003.

[6] 2] Sztipanovits J., Karsai G., “Model-integrated computing”, IEEE Computer, pp. 110112, Apr. 1997.

[7] Long E., Misra A., Sztipanovits J., “Increasing Productivity at Saturn”, IEEE Computer Magazine, August, 1998.

[8] Neema S., Sztipanovits J., Karsai G, “Design-Space Construction and Exploration in Platform-Based Design”, ISIS-02-301, June 24, 2002.

[9] Nordstrom, G., “Metamodeling — Rapid Design and Evolution of Domain-Specific Modeling Environments”, Proceedings of the IEEE ECBS ’99 Conference, 1999.

[10] Neema S., Karsai G., Embedded Control Systems Language for Distributed Processing, ISIS-04-505, May 12, 2004.

[11] Schmidt D., Gokhale A., Natarajan B., Neema S., Bapty T., Parsons J., Gray J., Nechypurenko A., Wan N., “CoSMIC: An MDA Generative Tool for Distributed Real-time and Embedded Component Middleware and Applications”, 2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture, 2002.

[12] Agrawal A., Levendovszky T., Sprinkle J., Shi F., Karsai G., “Generative Programming via Graph Transformations in the Model-Driven Architecture”, 2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture, 2002.