Generalized Sketches and Model-driven Architecture

Adrian Rutle[1], Yngve Lamo[1] and Uwe Wolter[2]
[1]Bergen University College, NORWAY
[2]Dept. of Informatics, University of Bergen, NORWAY

ABSTRACT

Software developers increasingly recognize the need for portable, scalable and distributed applications which are reliable, secure and run at high performance. These applications are also required to be produced both faster and at a lower cost. An approach to achieve this is the Object Management Group's Model Driven Architecture which involves automatic development processes and model management in addition to executable models. This makes the importance of formal modeling languages more recognizable since the automatization of processes requires formal models. However, current modeling languages are either semi-formal, ambiguous, or both, therefore, a generic framework for formal, diagrammatic software specification is inevitable. In this presentation, Generalized Sketches will be introduced as a generic framework for the specification of modeling languages and transformations between them. Then an example will be presented to show how Generalized Sketches may be used in Model Driven Architecture.

Introduction and Motivation

Evidently, formalization of modeling languages and transformations between them is an important step in Model Driven Architecture (MDA) [10]. In contrast to the traditional software development processes where models are used only for documentation purposes, in MDA, models are considered first-class citizens. In MDA, building applications is started by the construction of abstract, platform-independent models (PIM) of system properties and behavior. These models are automatically transformed into one or more platform-specific models (PSM) which are used by code-generators to generate application code [7]. The transformation processes are specified by transformation definition languages and are executed by transformation tools. The involvement of these tools in the development processes requires that the models and the transformations between them should be defined formally. This implies the necessity of techniques which can be used to specify formal models and formal transformation definitions.

In addition, using formal modeling techniques provides mechanisms for model de-composition and integration; mechanisms for verification of correctness, consistency and validity of models and transformation definitions; as well as mechanisms for reasoning about models and transformations. Thus the basis for executable models will be a step nearer accomplishment.

Related Work

Currently, the Unified Modeling Language (UML) is the most used diagrammatic modeling language in software engineering. A huge effort is done by the Object Management Group (OMG) to formalize UML. This effort has resulted in special languages such as the Meta Object Facility (MOF) [8], which can be used to specify all other OMG languages; and the Queries, Views and Transformations (QVT) language which is used to specify transformations between MOF-compliant languages [9]. The QVT's approach of transformation requires that modeling languages are MOF-compliant in order to enable model transformation and integration. This approach might solve some of the problems concerning model definition and transformation, however, it is not the optimal solution for the problem because of the continuous development of modeling languages that are not necessarily MOF-compliant. In fact, MOF only provides the syntax for modeling languages, hence the semantics is still needed to be formalized.

Another language which is used for definition of transformations is the ATLAS Transformation Language (ATL) [6]. ATL is currently available as an open source project under the Eclipse Modeling subproject. The ATL framework consists of a transformation language (ATL), a virtual machine, and an IDE for writing transformation definitions. ATL is a hybrid language –both declarative and imperative. The declarative style is used to specify rules for matching source and target patterns. During application of the rules, a target pattern will be created in the target model whenever a source pattern is found. ATL presumes that the source and target languages are well-formed without providing mechanisms to check the well-formedness of the languages. ATL defines the semantics of transformation rules, however, this semantics is not graph-based.

Goals and Approaches

The purpose of our research is to investigate and apply the potentials of Category Theory (CT) and the ideas which propose Generalized Sketches (GS) as a mathematical formalism for formalization of modeling languages [1, 2, 4] and their transformations [3]. GS is a graph-based specification format which borrows its main ideas from both first order logic and categorical logic [4]. GS provides mechanisms for specification and manipulation of models in an abstract, generic and formal way.

In GS, a modeling language L is represented by a meta-sketch S_L which in turn is based on a diagrammatic signature Σ_L. A Σ_L-sketch can be seen as an instance of S_L,

$L-$models correspond to visualizations of Σ_L-sketches, and model transformations correspond to sketch operations. Σ_L is a collection of diagrammatic *predicate symbols*, and, each predicate symbol corresponds to a constraint or construct which can be set by L. Σ_L-sketches are categorical structures which consist of nodes, arrows and diagrams which are marked with predicate labels from Σ_L. By having a graph-based logic in which the arrow-thinking style is provided, the relationship between the syntax and the semantics of diagrams in Σ_L-sketches is made formal and more compact.

Heterogenous model transformations can be formalized in the following way. For the Language $L^{'}$, we can define the signature $\Sigma_{L'}$ and the meta-sketch $S_{L'}$. The transformation between constructs of L and $L^{'}$ can now be specified formally as a sketch operation which is given by a sketch morphism $\phi : S_L \rightarrow S_{L'}$ (Figure 1). Then a Σ_L-sketch, I_L, which is an instance of the meta-sketch S_L can be transformed to a $\Sigma_{L'}-$sketch, $I_{L'}$, by applying the sketch operation as shown in the figure and explained in the example below. The commutativity requirement of the diagram in the figure is to assure that the transformation is correct. For homogenous model transformation, we use the same procedure for $\Sigma_L = \Sigma_{L'}$.

$$\begin{tikzcd}
S_L \arrow[r, "\phi"] & S_{L'} \\
I_L \arrow[u, "\iota_1"] \arrow[r, "\phi^*"'] \arrow[ur, phantom, "{[=]}"] & I_{L'} \arrow[u, "\iota_2"']
\end{tikzcd}$$

Figure 1: Generic Model Transformation

Example

A simplified meta-sketch, S_{PIM}, of a subset of a PIM language is shown in Figure 2. The figure explains the relations between classes and attributes. In this language, attributes must have exactly one kind of visibility: public, private or protected. This constraint is given by the total mapping, $attrVisibility$, and the curly braces including the possible visibility types, $\{pub, prv, prt\}$. Associations are omitted for briefness, and operations are not part of this PIM language.

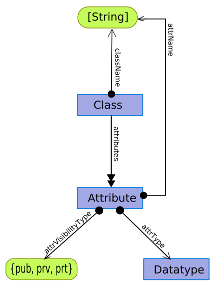

Figure 2: S_{PIM}, the meta-sketch of a PIM language

In Table 1, some constraints (or predicates) from Σ_{PIM} with their semantics in terms of sets and mappings are shown. Predicates can be combined to set the necessary constraints,

name	arity	visualization	semantic
"node"	$\bullet$	$\boxed{A}$	set
[cover]	$\bullet \longrightarrow \bullet$	$\boxed{A} \xrightarrow{f} \boxed{B}$	$\forall b \in B : \exists a \in A \mid f(a) = b$
[total]	$\bullet \longrightarrow \bullet$	$\boxed{A} \bullet\!\xrightarrow{f} \boxed{B}$	$\forall a \in A : \exists b \in B \mid f(a) = b$
[partial]	$\bullet \longrightarrow \bullet$	$\boxed{A} \circ\!\xrightarrow{f} \boxed{B}$	$\exists a \in A \mid \nexists b \in B \mid f(a) = b$
[multivalued]	$\bullet \longrightarrow \bullet$	$\boxed{A} \xrightarrow{f} \!\!\!\!\rightarrow \boxed{B}$	$\forall a \in A \mid a \in Dom(f) : f(a) \subseteq \mathsf{P}(B)$

Table 1: Signatures Σ_{PIM} and Σ_{PSM}

e.g. the mapping between *Class* and *Attribute* in S_{PIM} is marked with two predicates: *[cover]*, meaning that all attributes must exist as a field in some class, and *[multivalued]*, meaning that each class may have *0...** attributes.

Figure 3 shows a simplified meta-sketch, S_{PSM}, of a subset of a PSM language. Operations are part of this PSM language. There is a semantic overlap between the signatures of the PIM and the PSM languages, therefore, there is no need for a signature mapping. In most cases, especially when very different languages are concerned, a signature mapping is necessary to make the alignment of the two languages possible. A signature mapping is given by a signature morphism which is a mapping between predicate labels such that the shape graphs of the predicate labels are preserved.

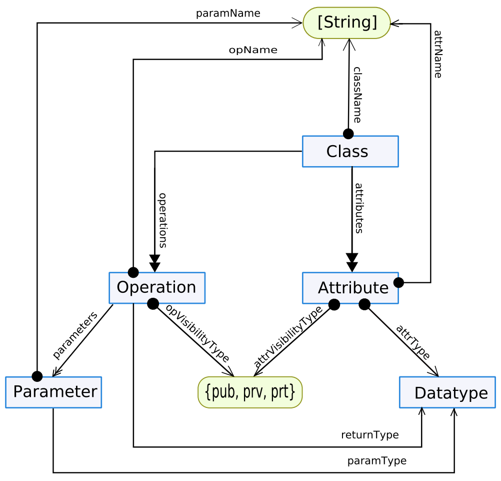

Figure 3: S_{PSM}, the meta-sketch of a PSM language

In Figure 4, we show a very simple transformation definition as it is defined by the GS formalism. This transformation is often used in the transformation of PIMs to PSMs, where a low level model –including operations– is generated from the high level model. For each public attribute *attr* from the PIM, we will generate a private attribute and two public operations –a getter and a setter– to access *attr*. In the transformation, we declare that for any match of the diagram SRC in the source model, the diagram TRG will be generated in the target model. This transformation can be specified diagrammatically and, we can put any kind of formal constraints on the transformation and its source and target models.

Examples of constraints which must be defined for this transformation are:

- $getterOp; returnType = attr; attrType$, stating that the return type of the getter of each attribute must be

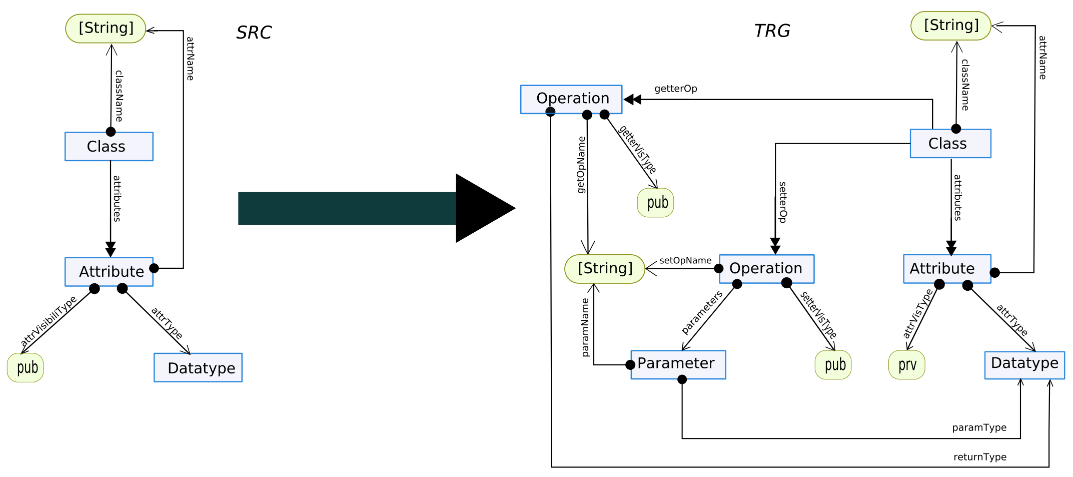

Figure 4: Transformation definition.

the same as the type of the attribute.

- $setterOp; param; paramType = attr; attrType$, stating that the type of the parameter of the setter of each attribute must be the same as the type of the attribute.
- $getterOp; op1Name = "get" + attr; attrName$, stating that the name of the getter of each attribute must be the same as the name of the attribute prefixed by "get".
- $setterOp; op2Name = "set" + attr; attrName$, stating that the name of the setter of each attribute must be the same as the name of the attribute prefixed by "set".

Concluding Remarks

The example shows how a transformation between two modeling languages (which are specified in GS) is defined. That kind transformation may be achieved also by using ATL or QVT. However, in GS, this process is both diagrammatic, allowing us to define the transformation visually; formal, allowing us to compose transformations and verify easily that the target model is an instance of the target metamodel; as well as language-independent, the transformation can be applied to the metamodel of any source language which has an occurrence of *SRC*, and any target language in which *TRG* can be expressed.

Currently, the focus of our research is on accomplishing the theories of GS and the design of tools for the application of these theories. Our tools will be implemented as plugins to Eclipse and will be proposed as a subproject of the Eclipse Modeling project [5]. The tools will be intended for three groups of users.

- Those who are interested in formalization of languages by designing diagrammatic signatures for those languages.
- Those who are interested in comparison and alignment of languages by definition of transformations between those languages.
- Those who are interested in using the signatures and transformations to define domain-specific models and then transform them to models in other modeling or programming languages, i.e. automatic code-generation.

1. REFERENCES

[1] Zinovy Diskin. Generalized sketches as an algebraic graph-based framework for semantic modeling and database design. Technical Report 9701, University of Latvia, Riga, Latvia, August 1997.

[2] Zinovy Diskin. *Practical foundations of business system specifications*, chapter Mathematics of UML: Making the Odysseys of UML less dramatic, pages 145–178. Kluwer Academic Publishers, 2003.

[3] Zinovy Diskin. *Encyclopedia of Database Technologies and Applications*, chapter Mathematics of Generic Specifications for Model Management I and II, pages 351–366. Information Science Reference, 2005.

[4] Zinovy Diskin and Uwe Wolter. A Diagrammatic Logic for Object-Oriented Visual Modeling. In *ACCAT 2007: 2nd Workshop on Applied and Computational Category Theory*, volume 203 of *ENTCS*, pages 19–41, Amsterdam, The Netherlands, 2008. Elsevier Science Publishers B. V.

[5] Eclipse Modeling Framework. *Project Web Site.* `http://www.eclipse.org/emf/`.

[6] Frédéric Jouault and Ivan Kurtev. On the architectural alignment of ATL and QVT. In Hisham Haddad, editor, *SAC 2006: 21nd ACM Symposium on Applied Computing*, pages 1188–1195. ACM, 2006.

[7] Anneke G. Kleppe, Jos Warmer, and Wim Bast. *MDA Explained: The Model Driven Architecture: Practice and Promise.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[8] Object Management Group. *Meta-Object Facility Specification*, January 2006. `http://www.omg.org/cgi-bin/doc?formal/2006-01-01`.

[9] Object Management Group. *Query/View/Transformation Specification*, April 2008. `http://www.omg.org/cgi-bin/doc?formal/2008-04-03`.

[10] OMG Model Driven Architecture. *Web Site.* `http://www.omg.org/mda/`.