Generalized Sketches and Model-driven Development

Adrian Rutle

Department of Computer Engineering
Faculty of Engineering
Bergen University College

12.10.2007
To be presented at NWTP 07, Oslo, Norway
Outline

1 Introduction and Motivation
   - Model-driven Development (MDD)
   - Generalized Sketches (GS)

2 Project Goals and Approaches
   - Project Goals
   - Examples

3 Tools and Contributions
Outline

1. Introduction and Motivation
   - Model-driven Development (MDD)
   - Generalized Sketches (GS)

2. Project Goals and Approaches
   - Project Goals
   - Examples

3. Tools and Contributions
What is MDD?

- Software development methodology in which models are first-class entities
- Development of software is started by building platform-independent models
- Refinement of software can be achieved by model transformations
- Models are used as input to code-generation tools
- Code generation can be seen as model transformation
- Model integration and de-composition can be achieved by using tools
MDD vs Traditional Development Processes

**Traditional**
- Requirements: Mostly text
- Analysis: Diagrams and text
- System Design: Diagrams and text
- Implementation: Code
- Testing: Code
- Deployment: Code

**MDA**
- Requirements: Mostly text
- Analysis: PIM
- System Design: PSM
- Implementation: Code
- Testing: Code
- Deployment: Code

**Notes:**
- PIM: Platform Independent Model
- PSM: Platform Specific Model
Benefits of MDD

- Raises the abstraction level of programming languages
- Software interoperability by platform independence of the models
- Increase in productivity by automatic code generation
- Flexibility
- Separation of business logic from application logic
Outline

1. Introduction and Motivation
   - Model-driven Development (MDD)
   - Generalized Sketches (GS)

2. Project Goals and Approaches
   - Project Goals
   - Examples

3. Tools and Contributions
What is GS?

- Graph-based specification format
- Graph-based logic based on FOL and Category Theory
- Can be used as a mathematical formalism to formalize modeling languages and transformations between them
In GS, categorical sketches are generalized and adapted for modeling in software engineering.

Categorical sketches are representations of mathematical categories.

Sketches in GS are graphes in which some diagrams are marked with predicates from a predefined signature.
Definitions

Definition
A signature $\Sigma := (\Pi, ar)$ is an abstract structure consisting of a collection of predicate symbols $\Pi$ with a mapping that assigns a shape (arity) graph $ar(p)$ to each predicate symbol $p \in \Pi$.

Definition
A diagram labeled with $p$ in a graph $G(S)$ is a graph morphism $\delta : ar(p) \rightarrow G(S)$ where $ar(p)$ is the shape of $p$. 
Definitions (cont.)

A \(\Sigma\)-sketch \(S := (G(S), D(\Pi))\), is a graph \(G(S)\) with a set \(D(\Pi)\) of diagrams in \(G(S)\) labeled with predicates from the signature \(\Sigma\). \(\text{Ske}(\Sigma)\) is used to denote the collection of all \(\Sigma\)-sketches.

A \(\Sigma\)-sketch morphism \(\phi : S_1 \rightarrow S_2\) between two \(\Sigma\)-sketches \(S_1 = (G(S_1), D_1(\Pi))\) and \(S_2 = (G(S_2), D_2(\Pi))\) is a graph homomorphism \(\phi : G(S_1) \rightarrow G(S_2)\) compatible with marked diagrams, i.e., \((\delta : ar_1(p_1) \rightarrow G(S_1)) \in D_1(\Pi)\) implies \((\delta ; \phi : ar_1(p_1) \rightarrow \phi(G(S_1))) \in D_2(\Pi)\) for all \(p \in \Sigma\).

Now \(\text{Ske}(\Sigma)\) is the category of all \(\Sigma\)-sketches, where objects are sketches and morphisms are sketch-morphisms.
Definitions (cont.)

**Definition**

The semantic interpretation of a signature \( \Sigma := (\Pi, \text{ar}) \) is given by a mapping that assigns to each \( p \in \Pi \) a set of graph homomorphisms \( \llbracket P \rrbracket = \tau : O \rightarrow \text{ar}(p) \) which is called the set of valid instances of \( p \).
An instance of a sketch $S$ is a graph $I$ together with a graph morphism $\iota : I \rightarrow G(S)$, where $G(S)$ is the carrier graph of $S$, such that $\iota^* \in [P]$, i.e. $\iota^*$ is a valid instance of $p$ for each diagram $\delta : ar(p) \rightarrow G(S)$ where $\iota^*$ (and $O$) are given by the following pullback diagram:

$$
\begin{array}{c}
ar(p) \xrightarrow{\delta} G(S) \\
\uparrow^{\iota^*} \quad [PB] \quad \uparrow^{\iota} \\
O \xrightarrow{\delta^*} I
\end{array}
$$

(1)
## OMG Metamodel Levels

<table>
<thead>
<tr>
<th>OMG levels</th>
<th>OMG Standards/examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_3$</td>
<td>MOF</td>
</tr>
<tr>
<td>$M_2$</td>
<td>UML language</td>
</tr>
<tr>
<td>$M_1$</td>
<td>A UML model: Class &quot;Person&quot; with attributes &quot;name&quot; and &quot;address&quot;</td>
</tr>
<tr>
<td>$M_0$</td>
<td>An instance of &quot;Person&quot;: &quot;Ola Nordmann&quot; living in &quot;Sotraveien 1, Bergen&quot;</td>
</tr>
</tbody>
</table>
## GS Methodology and Metamodelling

### Definition

A $\Sigma^M$-sketch $S^M = (G(S^M), D^M(\Pi))$ is the metamodel of $\text{Ske}(\Sigma)$ iff for all $\Sigma$-sketches $S = (G(S), D(\Pi))$ there exists an instance $i^S : G(S) \rightarrow G(S^M)$ of $S^M$ and for any instance $\iota : I \rightarrow G(S^M)$ of $S^M$ there exists a $\Sigma$-sketch $S^\iota = (I, D(\Pi))$.

<table>
<thead>
<tr>
<th>OMG levels</th>
<th>OMG standards</th>
<th>GS methodology</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_3$</td>
<td>MOF</td>
<td>$\Sigma^M$</td>
</tr>
<tr>
<td>$M_2$</td>
<td>UML</td>
<td>Signature $\Sigma_{UML}$ represented by a $\Sigma^M$-sketch $S_{UML}$</td>
</tr>
<tr>
<td>$M_1$</td>
<td>A UML model</td>
<td>$\Sigma_{UML}$-sketch $S = (G(S), D(\Pi))$</td>
</tr>
<tr>
<td>$M_0$</td>
<td>An instance</td>
<td>An instance $\iota : I \rightarrow G(S)$ of $S$</td>
</tr>
</tbody>
</table>
Generic Model Transformation

$S_1 \xrightarrow{\phi} S_2$

$\text{Inst}(S_1) \xleftarrow{\phi} \text{Inst}(S_2)$

$\text{Inst}(S_1) \xleftarrow{\phi} \text{Inst}(S_2)$
Some model transformations can be achieved by the reduct transformation.

However, most model transformations in software engineering are used for model extension.

Thus require a morphism in the other direction

\[ \phi^\circ : \text{Inst}(S_1) \to \text{Inst}(S_2) \] which is persistent

\[ \phi^\circ \circ \phi^\bullet = \text{id}_{\text{Inst}(S_1)}. \]

Our hypothesis is that in most practical cases \( \phi^\circ \) is even a left-adjoint to \( \phi^\bullet \)
GS and MDA

MDA procedure

- Model_1 ...
- Model_n

Transformation Tools

Software System

needs

- Specification techniques
- Techniques for definition of transformations
- Techniques for automation of transformations

We use

Generalized Sketches to:
1. Specify the spec techniques
2. Specify transformations between spec techniques
3. "Automatically" transform models and their instances (by construction of Pullback)
Benefits/Features of GS

- Formalization of (graphical) modeling languages
- Formalization of transformation definitions
- Diagrammatic formalism
- Solid mathematical foundation
- Language-independent transformations
- Graph-based logic $\Rightarrow$ compact relation between syntax and semantics
- Support for model integration and de-composition
- Support for reasoning about models and transformations
About Formal Modeling Languages

Some advantages:
- Understandable by tools
- Support for checking model consistency and correctness
- Support for reasoning and verification of models and model transformations
- Automatization of model transformations

Some Disadvantages:
- Most formal modeling languages are text-based, complex and error-prone
- Not appealing for software engineers and programmers
Diagrammatic languages are easier to use and more appealing to software engineers than text-based languages.

However, diagrammatic languages are more challenging to formalize.
Outline

1. Introduction and Motivation
   - Model-driven Development (MDD)
   - Generalized Sketches (GS)

2. Project Goals and Approaches
   - Project Goals
   - Examples

3. Tools and Contributions
Our Goals

- Investigate GS as a framework for formalization of (diagrammatic) modeling languages and transformations between them
- Develop and adapt the theories of GS for generic and diagrammatic model management
- Sketching/formalizing common modeling languages (e.g. UML) as case-studies
- Implementation of tools for MDD which exploit the capabilities of GS
Outline

1. Introduction and Motivation
   - Model-driven Development (MDD)
   - Generalized Sketches (GS)

2. Project Goals and Approaches
   - Project Goals
   - Examples

3. Tools and Contributions
A Simple Model Transformation

Two different UML languages

```
Person
+firstName: String

Person
-firstName: String
+getFirstName(): String
+setFirstName(_name: String)
```
The PIM Language in GS

PIM’s Meta-sketch

PIM Signature $\Sigma_{PIM}$

<table>
<thead>
<tr>
<th>name</th>
<th>arity</th>
<th>visualization</th>
<th>semantic</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;node&quot;</td>
<td>☐</td>
<td>$A$</td>
<td>set</td>
</tr>
<tr>
<td>[cover]</td>
<td>☐ →</td>
<td>$A \xrightarrow{f} B$</td>
<td>$\forall b \in B : \exists a \in A \mid f(a) = b$</td>
</tr>
<tr>
<td>[total]</td>
<td>☐ →</td>
<td>$A \xrightarrow{f} B$</td>
<td>$\forall a \in A : \exists b \in B \mid f(a) = b$</td>
</tr>
<tr>
<td>[partial]</td>
<td>☐ →</td>
<td>$A \xrightarrow{f} B$</td>
<td>$\exists a \in A \mid \exists b \in B \mid f(a) = b$</td>
</tr>
<tr>
<td>[multivalued]</td>
<td>☐ →</td>
<td>$A \xrightarrow{f} B$</td>
<td>$\forall a \in A \mid a \in \text{Dom}(f) : f(a) \subseteq P(B)$</td>
</tr>
<tr>
<td>[disjoint]</td>
<td>☐ →</td>
<td>$A \xrightarrow{f} B$</td>
<td>${ f(a) \mid a \in A } \cap { g(c) \mid c \in C } = \emptyset$</td>
</tr>
</tbody>
</table>
### The PIM Language in GS

#### PIM’s Meta-sketch

#### PIM Signature $\Sigma_{PIM}$

<table>
<thead>
<tr>
<th>name</th>
<th>arity</th>
<th>visualization</th>
<th>semantic</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;node&quot;</td>
<td>●</td>
<td>$A \longrightarrow B$</td>
<td>$\forall b \in B : \exists a \in A \mid f(a) = b$</td>
</tr>
<tr>
<td>[cover]</td>
<td>●</td>
<td>$A \longrightarrow B$</td>
<td>$\forall a \in A : \exists b \in B \mid f(a) = b$</td>
</tr>
<tr>
<td>[total]</td>
<td>●</td>
<td>$A \longrightarrow B$</td>
<td>$\exists a \in A \mid \nexists b \in B \mid f(a) = b$</td>
</tr>
<tr>
<td>[partial]</td>
<td>●</td>
<td>$A \longrightarrow B$</td>
<td>$\forall a \in A \mid a \in Dom(f) : f(a) \subseteq P(B)$</td>
</tr>
<tr>
<td>[multivalued]</td>
<td>●</td>
<td>$A \longrightarrow B$</td>
<td>${f(a) \mid a \in A } \cap {g(c) \mid c \in C } = \emptyset$</td>
</tr>
</tbody>
</table>
The PSM Language in GS

Diagram showing the relationships between classes, operations, attributes, and datatypes.
The Simple Model Transformation in GS
The Simple Transformation for Instances
Tools

- Our tools will be implemented as plugins to Eclipse.
- Formalization of languages by designing diagrammatic signatures for those languages.
- Comparison and alignment of languages by definition of transformations between those languages.
- 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.
Summary

- The MDD is a promising approach for future software development processes
- The MDD requires tools for automatic model management
- Automatic model management requires formal modeling and model transformation languages
- GS may be used as a generic framework for formalization of modeling and model transformation languages