Generalized Sketches and Model-driven Development

Adrian Rutle

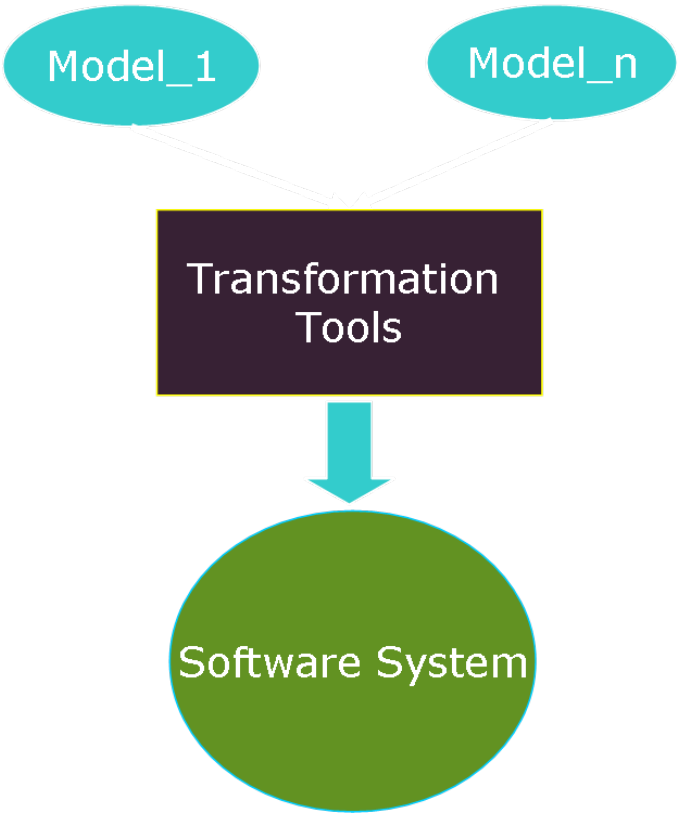
Department of Computer Engineering
Faculty of Engineering
Bergen University College

12.10.2007
To be presented at NWTP 07, Oslo, Norway

Outline

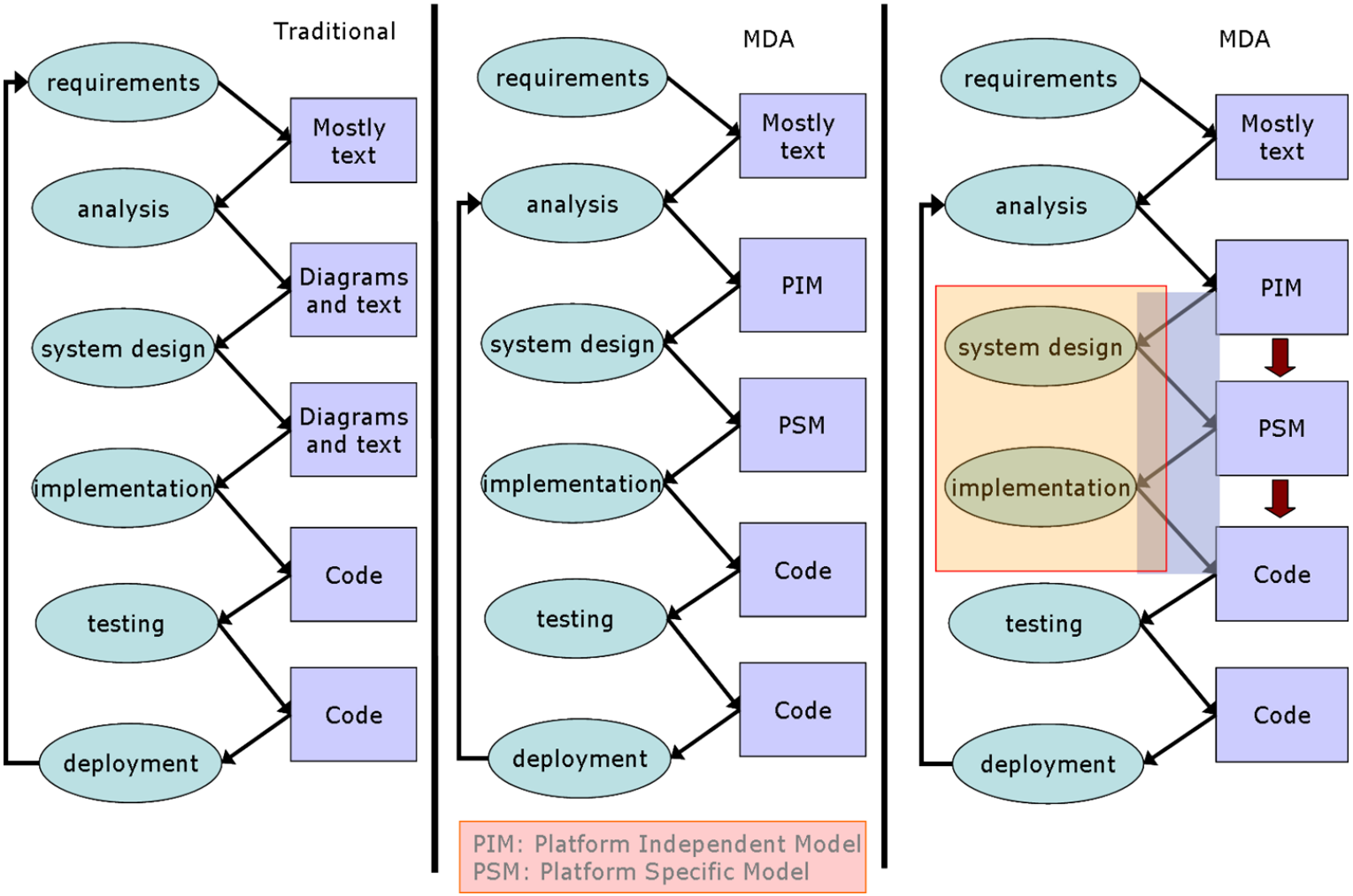
Outline

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



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

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

More About GS

- 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 Π 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.)

Definition

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 Σ. $Ske(\Sigma)$ is used to denote the collection of all $\Sigma - sketches$.

Definition

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 $Ske(\Sigma)$ 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 $\Sigma := (\Pi, ar)$ is given by a mapping that assigns to each $p \in \Pi$ a set of graph homomorphisms $[\![P]\!] = \tau : O \rightarrow ar(p)$ which is called the set of valid instances of p.

Definitions (cont.)

Definition

An instance of a sketch S is a graph I together with a graph morphism $\iota : I \to G(S)$, where $G(S)$ is the carrier graph of S, such that $\iota^* \in [\![P]\!]$, i.e. ι^* is a valid instance of p for each diagram $\delta : ar(p) \to G(S)$ where ι^* (and O) are given by the following pullback diagram:

$$\begin{tikzcd} ar(p) \arrow[r, "\delta"'] & G(S) \\ O \arrow[u, "\iota^*"] \arrow[r, "\delta^*"'] \arrow[ur, phantom, "[PB]"] & I \arrow[u, "\iota"'] \end{tikzcd} \qquad (1)$$

OMG Metamodel Levels

OMG levels	OMG Standards/examples
M_3	MOF
M_2	UML language
M_1	A UML model: Class "Person" with attributes "name" and "address"
M_0	An instance of "Person": "Ola Nordmann" living in "Sotraveien 1, Bergen"

GS Methodology and Metamodeling

Definition

A Σ^M-sketch $S^M = (G(S^M), D^M(\Pi))$ is the metamodel of $Ske(\Sigma)$ 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 $\Sigma - sketch\ S^\iota = (I, D(\Pi))$.

OMG levels	**OMG standards**	**GS methodology**
M_3	MOF	Σ^M
M_2	UML	Signature Σ_{UML} represented by a Σ^M-sketch S_{UML}
M_1	A UML model	Σ_{UML}-sketch $S = (G(S), D(\Pi))$
M_0	An instance	An instance $\iota : I \rightarrow G(S)$ of S

Generic Model Transformation

$$S_1 \xrightarrow{\phi} S_2$$

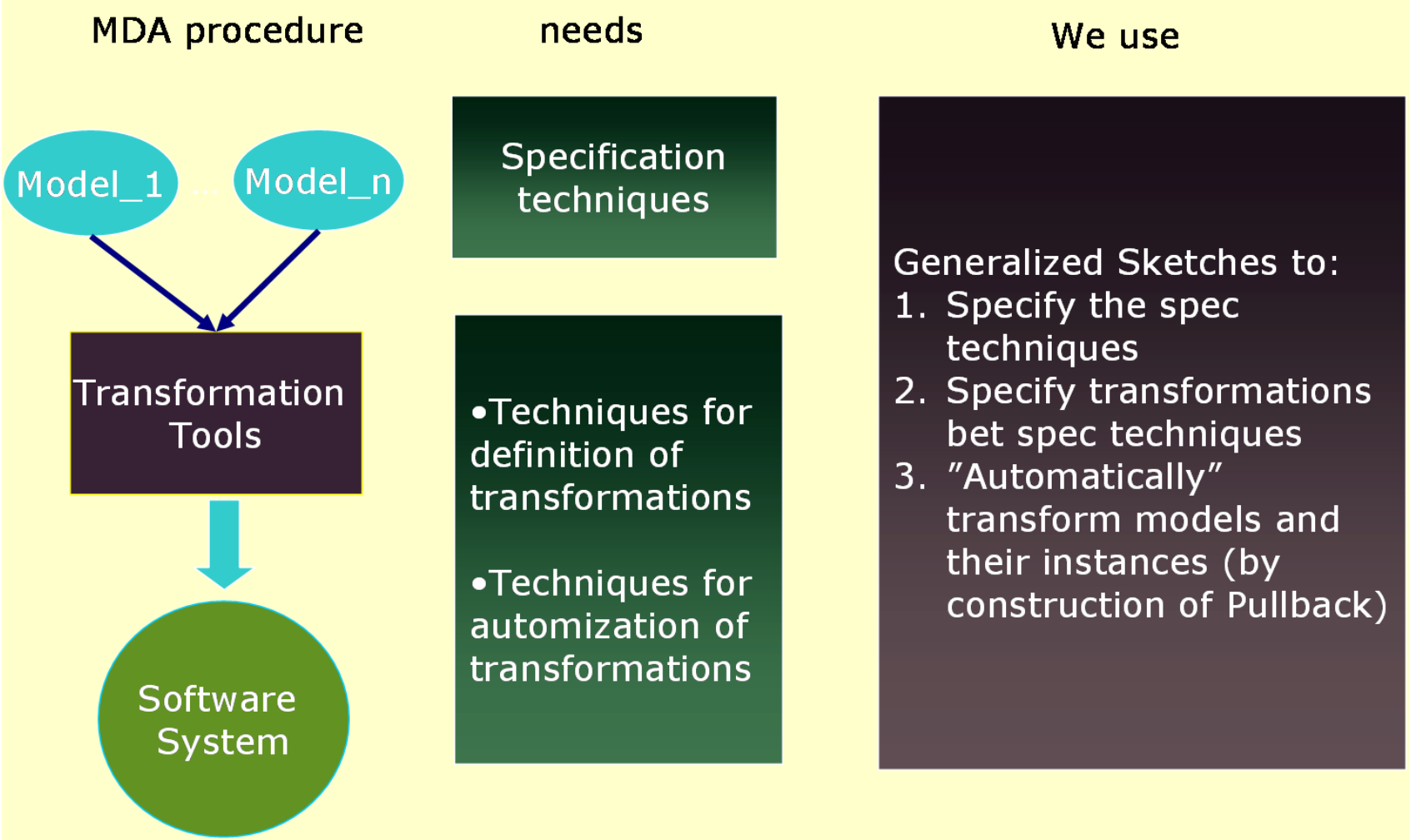
$$Inst(S_1) \xleftarrow[\phi^\bullet]{} Inst(S_2)$$

$$\begin{tikzcd}
ar_1(p_1) \arrow[r, "\delta"] \arrow[rr, bend left, "\delta;\phi"] & G(S_1) \arrow[r, "\phi"] & G(S_2) \\
O \arrow[u, "\iota"] \arrow[r, "\delta^*"] \arrow[rr, bend right, "\delta^*;\phi^*"'] \arrow[ur, phantom, "[PB_2]"] & I_1 \arrow[u, "\iota_1"] \arrow[r, "\phi^*"] \arrow[ur, phantom, "[PB_1]"] & I_2 \arrow[u, "\iota_2"]
\end{tikzcd}$$

Model Transformation in SE

- 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 $\phi^\circ : Inst(S_1) \rightarrow Inst(S_2)$ which is persistent $\phi^\circ ; \phi^\bullet = id_{Inst(S_1)}$.
- Our hypothesis is that in most practical cases ϕ° is even a left-adjoint to $\phi^\bullet$

GS and MDA


MDA procedure
needs
We use
Model_1
...
Model_n
Transformation Tools
Software System
Specification techniques
•Techniques for definition of transformations
•Techniques for automization of transformations
Generalized Sketches to:
1. Specify the spec techniques
2. Specify transformations bet 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

Solution: Formal, Diagrammatic Modeling Languages

- 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

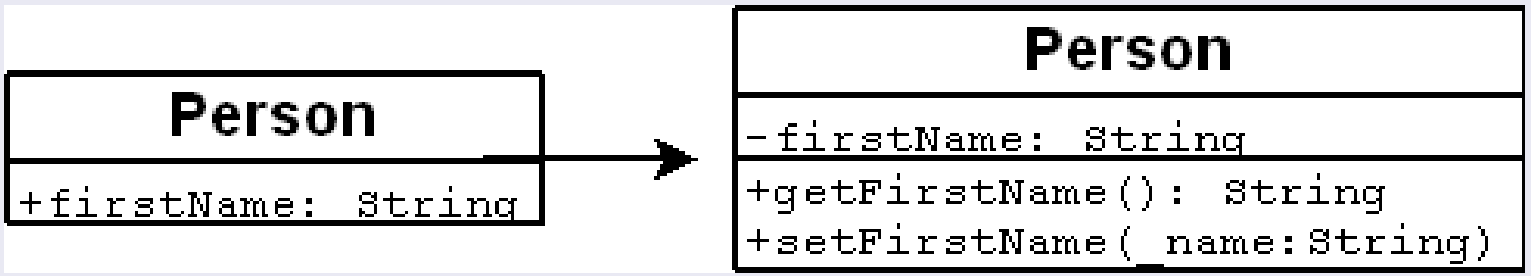
Our Goals

- 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

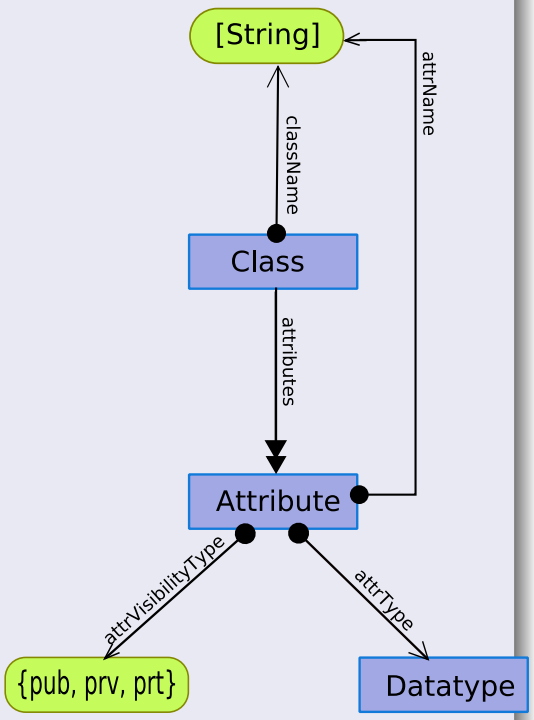
A Simple Model Transformation

Two different UML languages



The PIM Language in GS

PIM's Meta-sketch

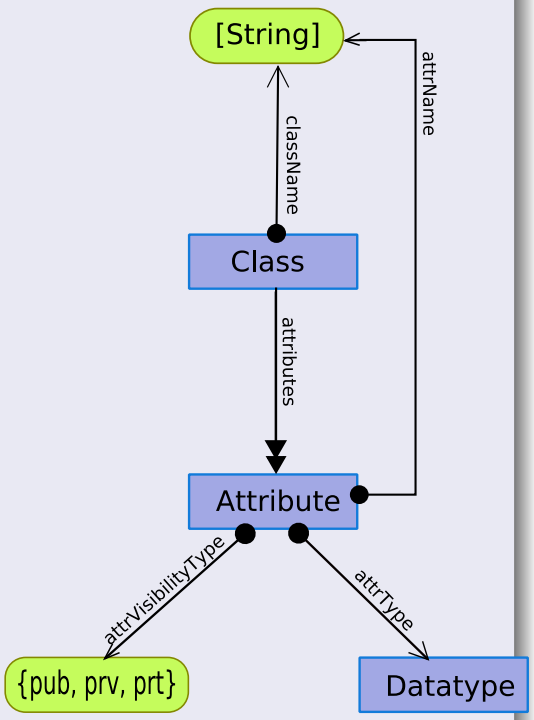


PIM Signature Σ_{PIM}

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]	$\bullet \longrightarrow \bullet$	$\boxed{A} \xrightarrow{f} \boxed{B}$	$\forall a \in A \mid a \in Dom(f) : f(a) \subseteq \mathrm{P}(B)$
[disjoint]	$\bullet \longrightarrow \bullet$, $\bullet \longrightarrow \bullet$	$\boxed{A} \xrightarrow{f} \boxed{B}$, $\boxed{C} \xrightarrow{g} \boxed{B}$	$\{f(a) \mid a \in A\} \cap \{g(c) \mid c \in C\} = \emptyset$

The PIM Language in GS

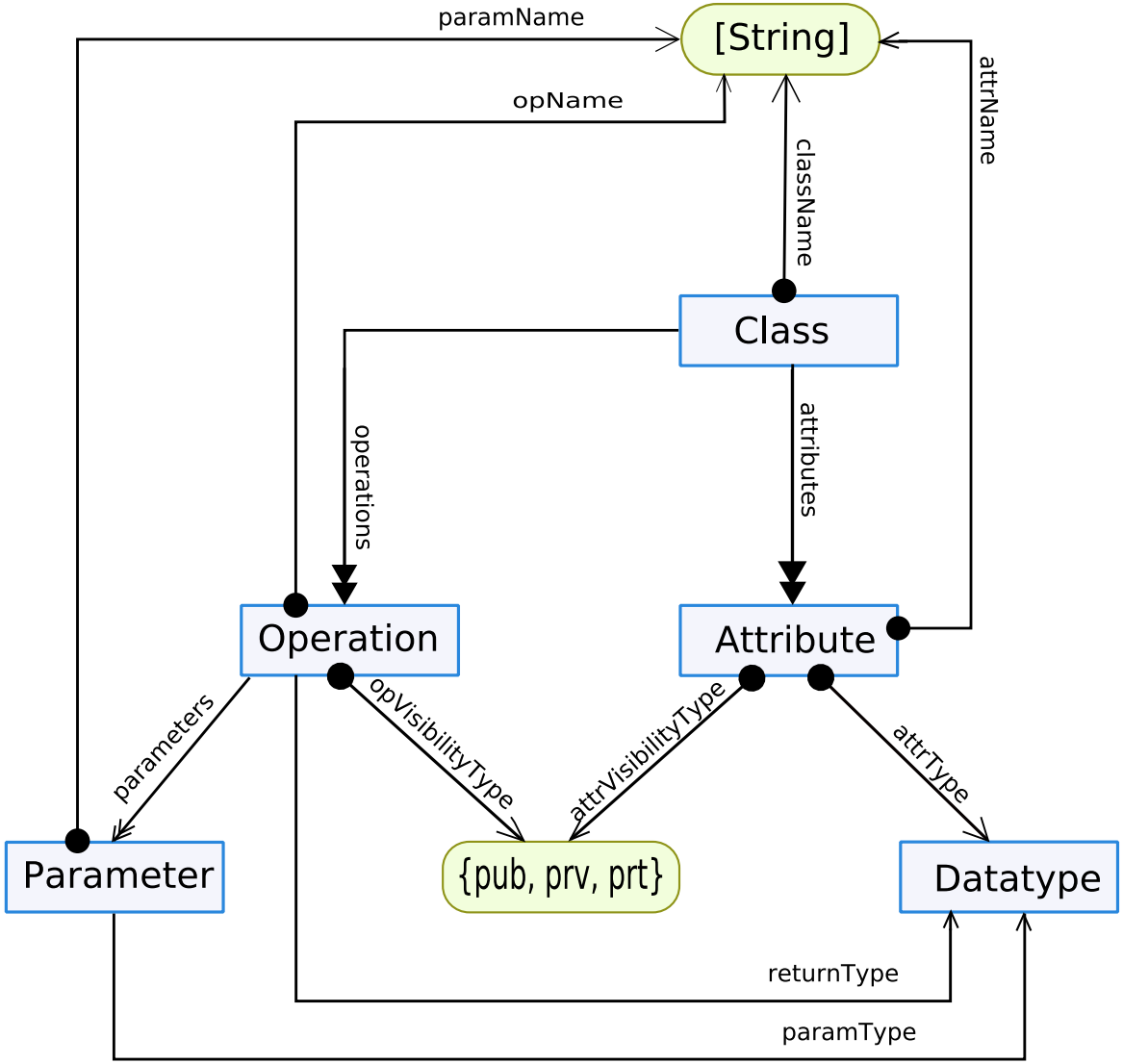
PIM's Meta-sketch



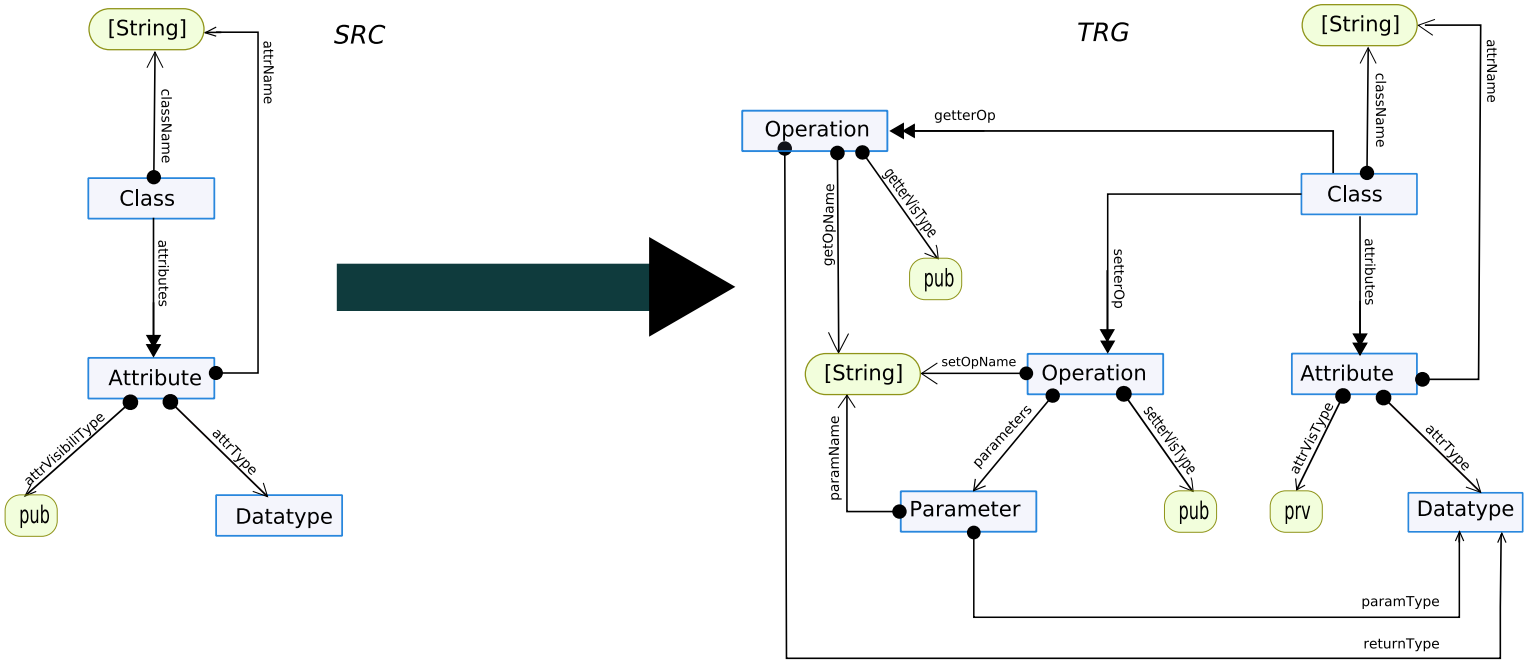
PIM Signature Σ_{PIM}

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]	$\bullet \longrightarrow \bullet$	$\boxed{A} \bullet\!\!-\!\!f\!\!\rightarrow \boxed{B}$	$\forall a \in A : \exists b \in B \mid f(a) = b$
[partial]	$\bullet \longrightarrow \bullet$	$\boxed{A} \circ\!\!-\!\!f\!\!\rightarrow \boxed{B}$	$\exists a \in A \mid \nexists b \in B \mid f(a) = b$
[multivalued]	$\bullet \longrightarrow \bullet$	$\boxed{A} \xrightarrow{f} \boxed{B}$	$\forall a \in A \mid a \in Dom(f) : f(a) \subseteq \mathrm{P}(B)$
[disjoint]	$\bullet \longrightarrow \bullet$, $\bullet \longrightarrow \bullet$	$\boxed{A} \xrightarrow{f} \boxed{B}$, $\boxed{C} \xrightarrow{g} \boxed{B}$	$\{f(a) \mid a \in A\} \cap \{g(c) \mid c \in C\} = \emptyset$

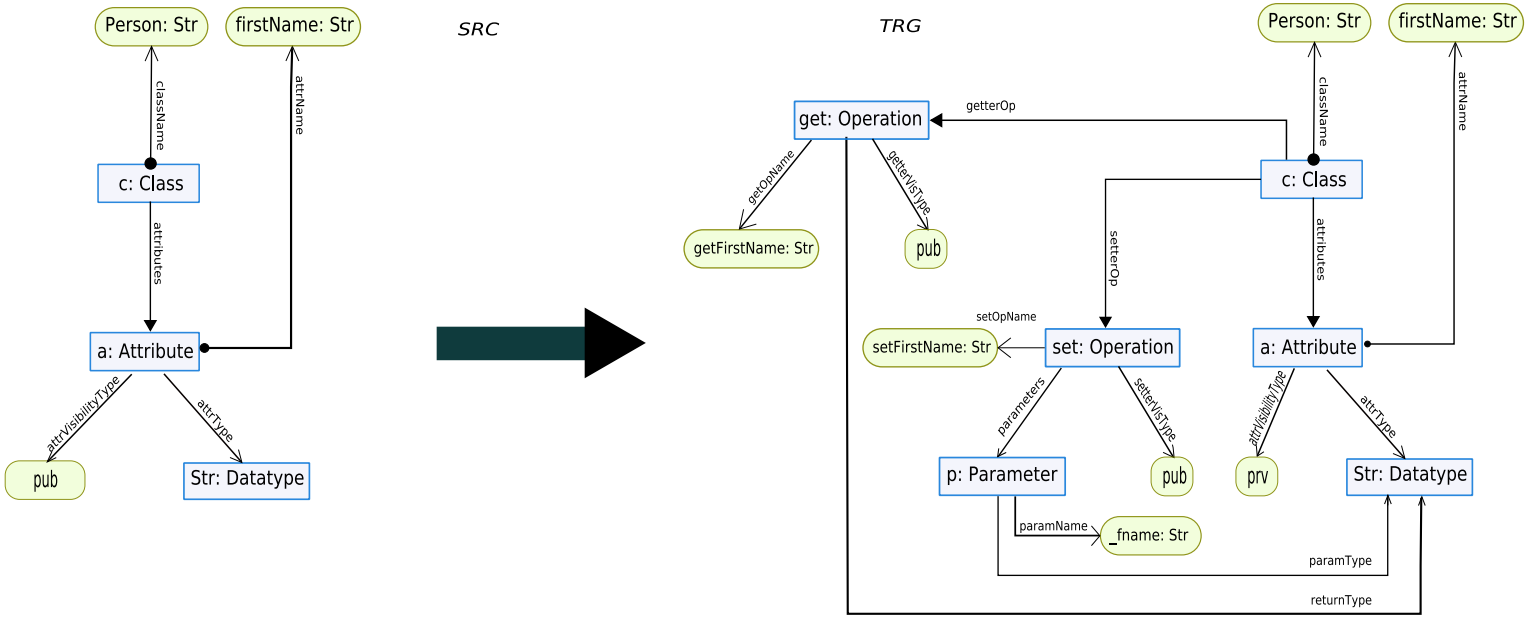
The PSM Language in GS



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