Generalized Sketches and Model-driven Development

Adrian Rutle

Department of Computer Engineering
Faculty of Engineering
Bergen University College

20.08.2007
To be presented at Calco'07-jnr, Bergen, Norway

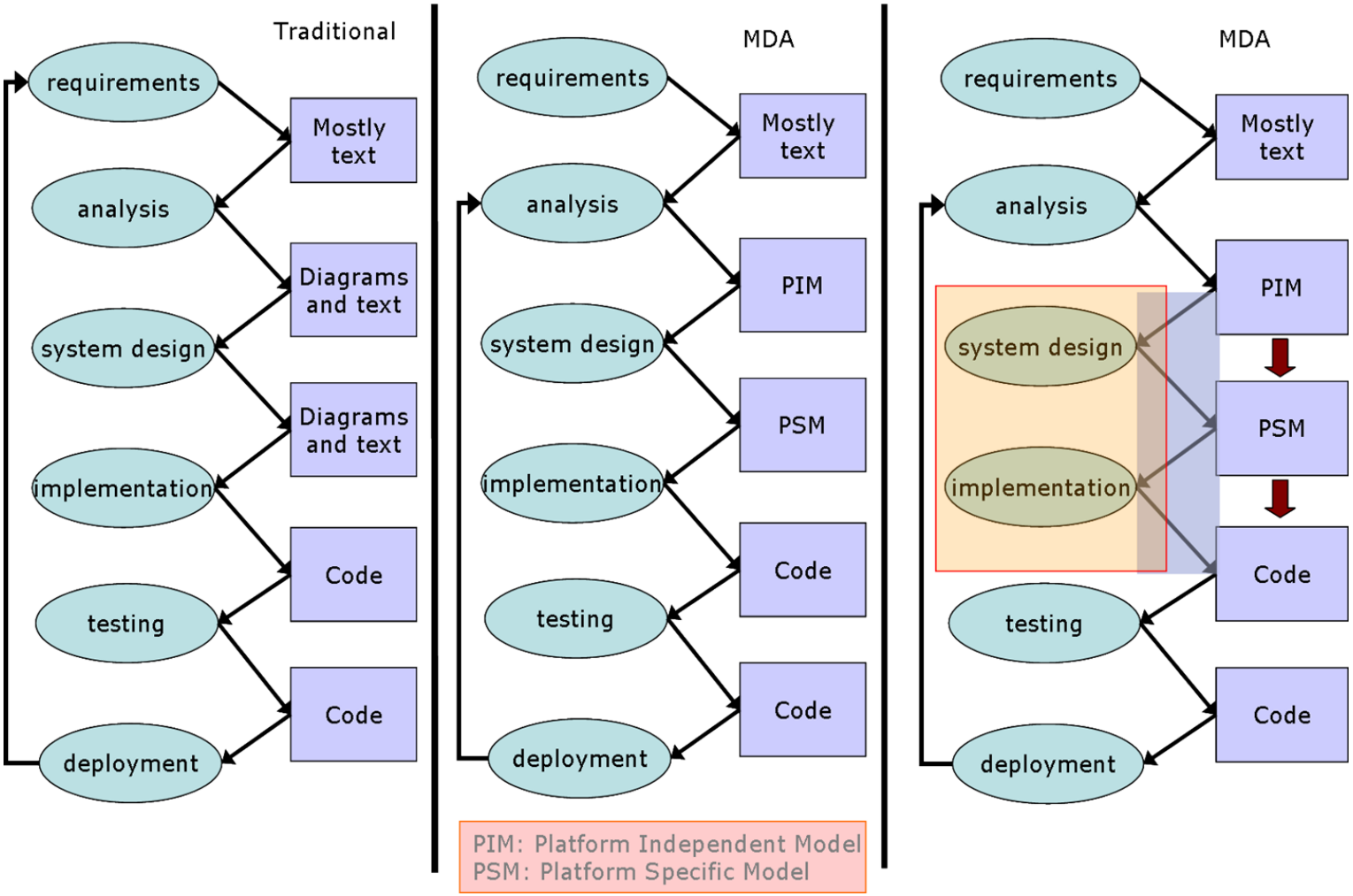
Outline

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

Terminology

- Signature is a graph morphism, $\alpha : \Sigma \rightarrow GRAPH$ where Σ is a graph (nodes: predicate symbols P, edges: dependencies between predicates).
- A $\Sigma - Sketch$ is a pair $S = (G, T)$. G, carrier graph, $T \subset Fm(G)$ is a set of diagrams over G.
- $Fm(G) = \{P(d) | P \in \Sigma, d$ the set of diagrams over G labeled with $\alpha P\}$
- Sketch morphism $s : S \rightarrow S^{'}$, $S = (G, T)$ and $S^{'} = (G^{'}, T^{'})$ is a graph morphism $\phi : G \rightarrow G^{'}$ such that the arity of diagrams are preserved.

Interpretation

- An interpretation of a signature $\alpha : \Sigma \to GRAPH$, is given by a mapping that assigns to each $P \in \Sigma$ a set of graph homomorphisms $[\![P]\!] = \tau : O \to \alpha P$, the set of valid instances of P.
- An instance of a sketch $S = (G, T)$ is a graph I together with a graph morphism $\iota : I \to G$, such that $\iota^* \in [\![P]\!]$, for each diagram $\delta \in Fm(G) : \alpha P \to G$ where ι^* (and O) are given by the following pullback diagram:

$$\begin{tikzcd}
\alpha P \arrow[r, "\delta"'] & G \\
O \arrow[u, "\iota^*"] \arrow[r, "\delta^*"'] \arrow[ur, phantom, "[PB]"] & I \arrow[u, "\iota"']
\end{tikzcd}$$

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

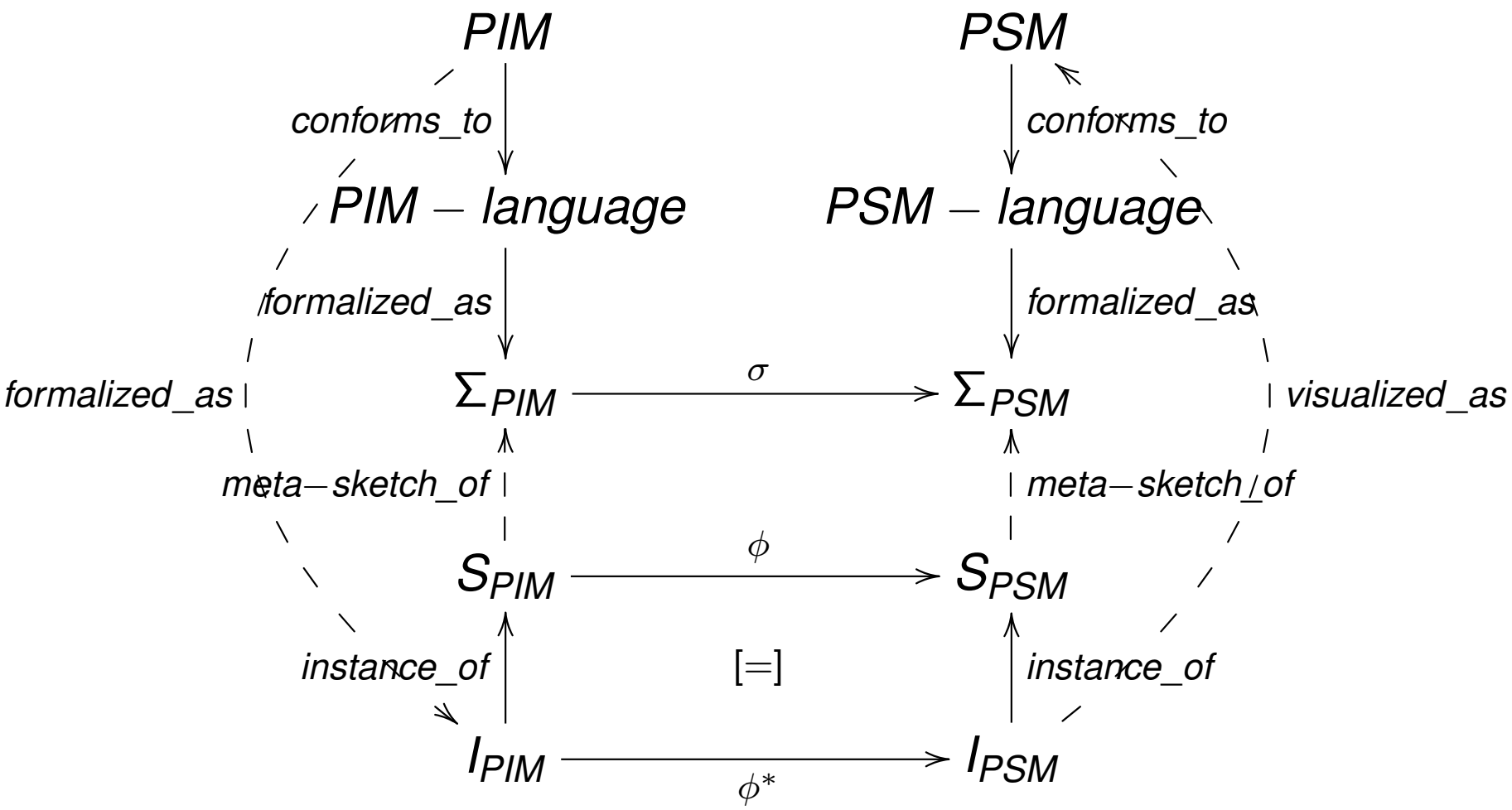
Outline

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

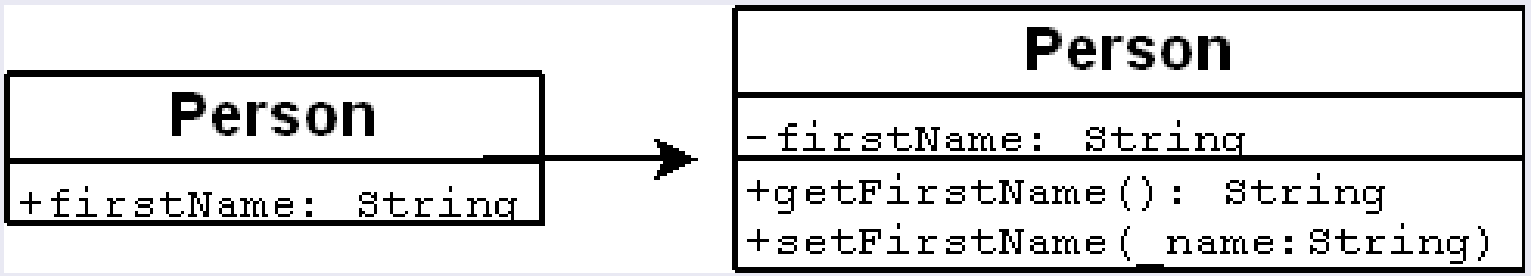
Formalization of Modeling Languages



Outline

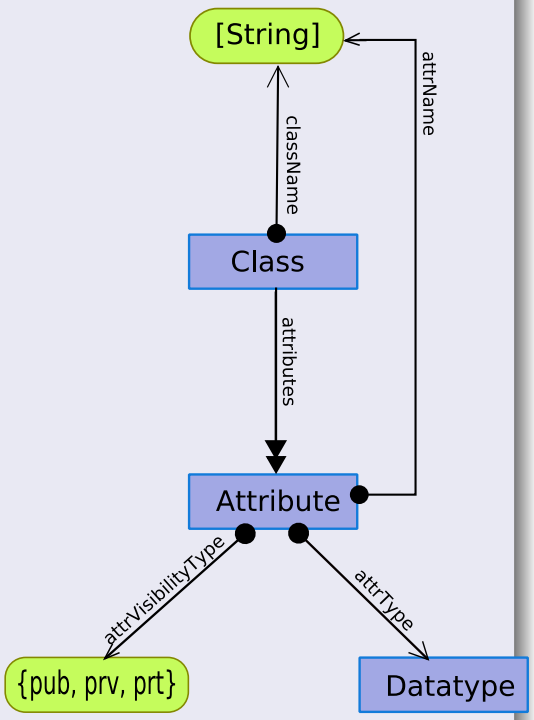
A Simple Model Transformation

Two different UML languages



The PIM Language in GS

PIM's Meta-sketch

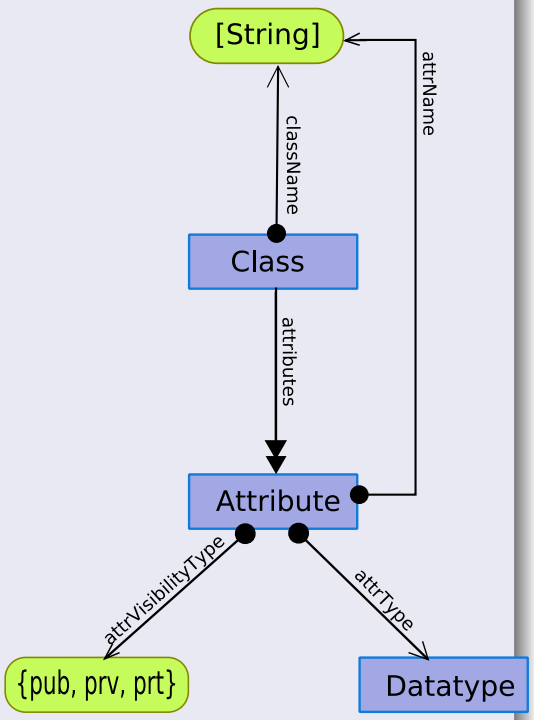


PIM Signature Σ_{PIM}

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

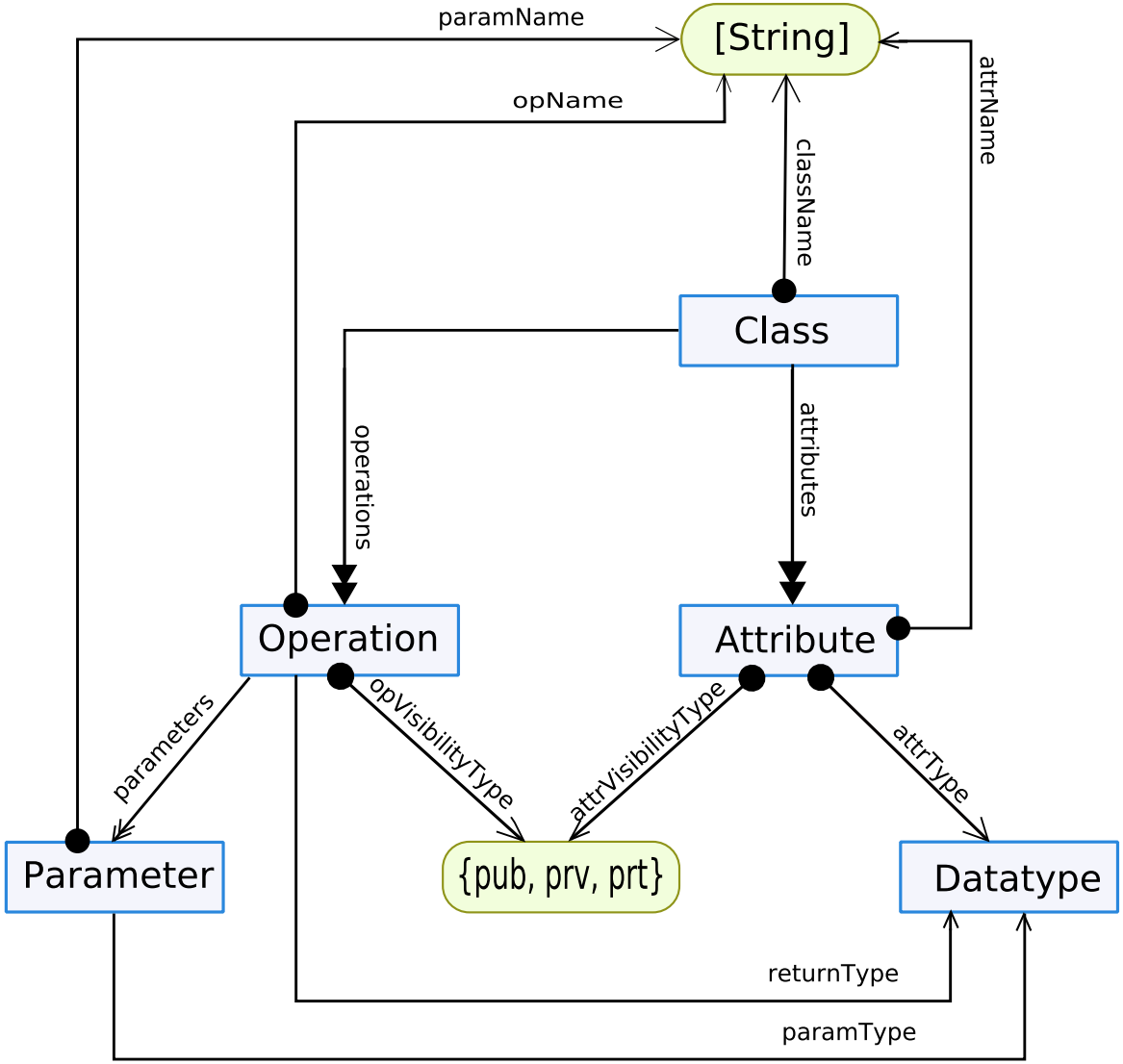
PIM's Meta-sketch



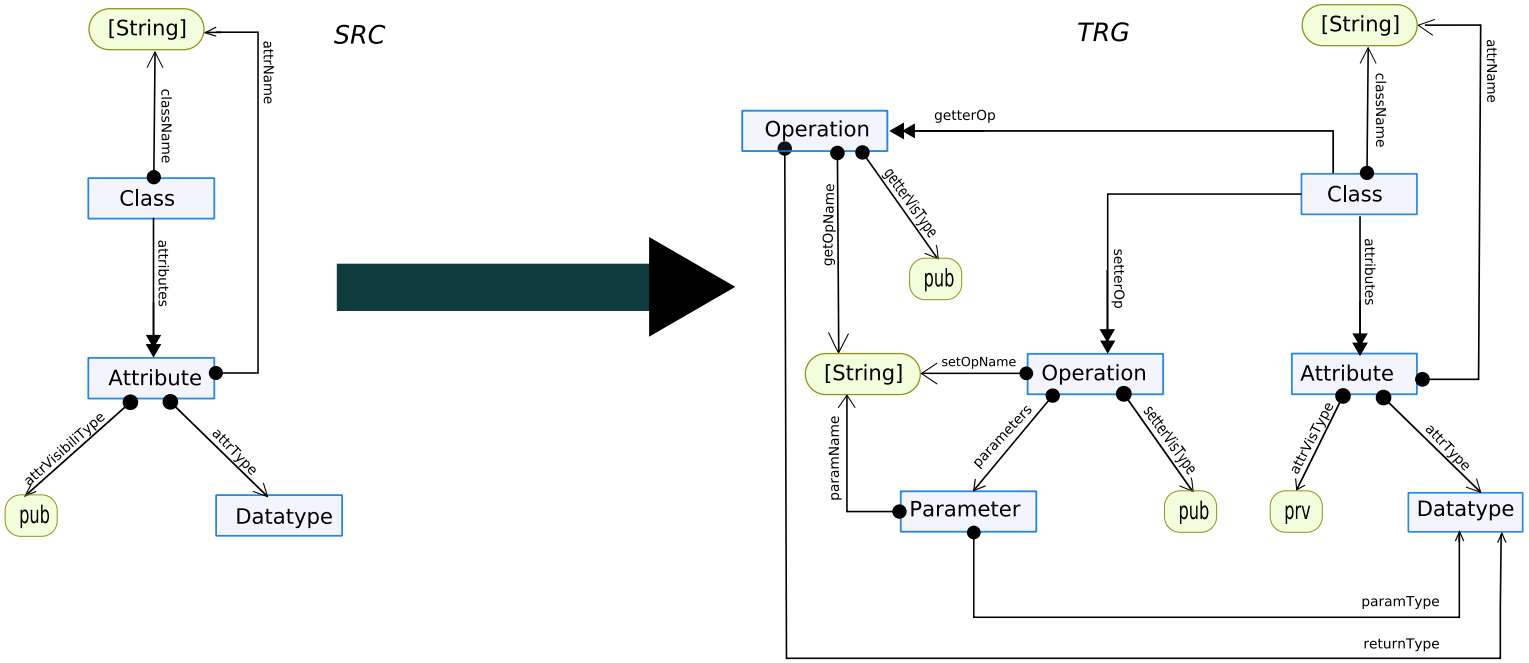
PIM Signature Σ_{PIM}

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]	$\bullet \longrightarrow \bullet$	$\boxed{A} \bullet\!\!-\!\!\xrightarrow{f} \boxed{B}$	$\forall a \in A : \exists b \in B \mid f(a) = b$
[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]	$\bullet \longrightarrow \bullet$, $\bullet \longrightarrow \bullet$	$\boxed{A} \xrightarrow{f} \boxed{B}$, $\boxed{C} \xrightarrow{g} \boxed{B}$	$\{f(a) \mid a \in A\} \cap \{g(c) \mid c \in C\} = \emptyset$

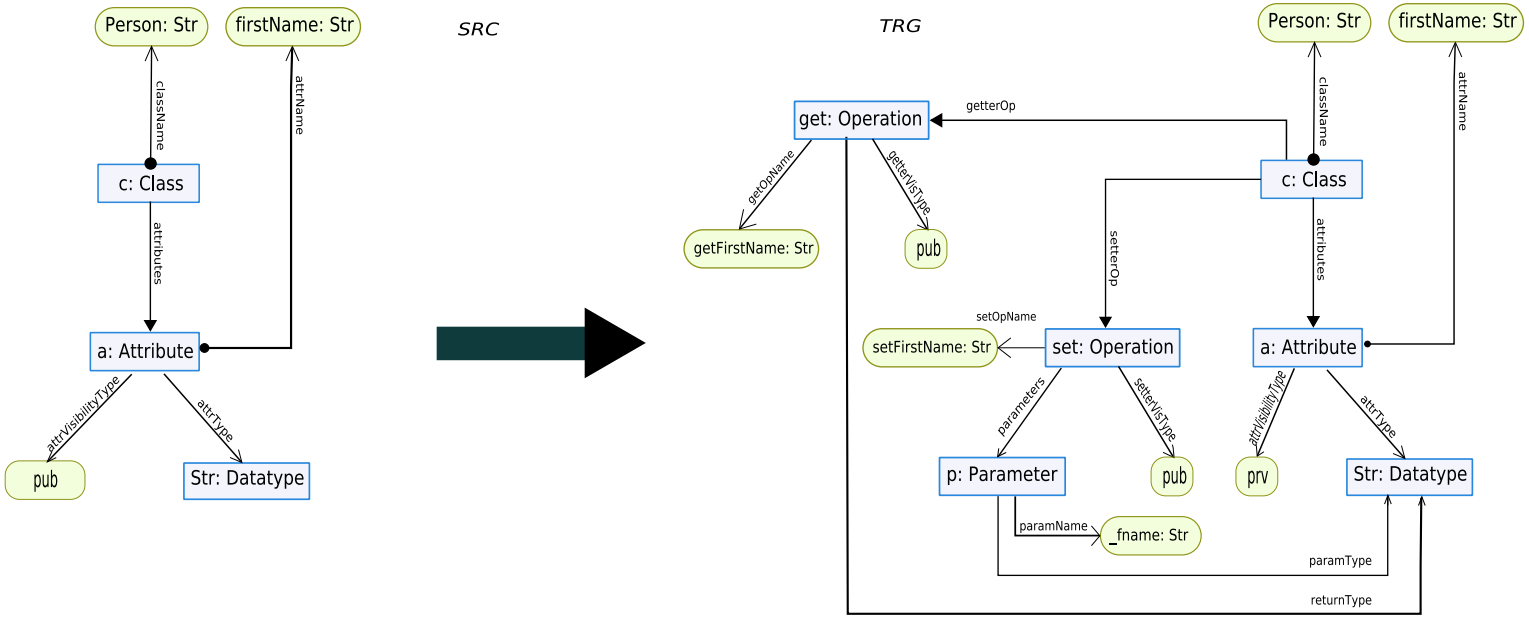
The PSM Language in GS



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