Model Driven Engineering (MDE) and Diagrammatic Predicate Logic (DPL)

Yngve Lamo

Department of Computer Engineering
Faculty of Engineering
Bergen University College
NORWAY

06.06.2008
Institute of Mathematics and Informatics, Vilnius, LITHUANIA

Project goal

- To apply and adapt formal methods to software engineering
  - Actual research on MDE and MDE-related standards and approaches
  - Develop a formal framework to express software models and model transformations
  - Develop tools that justifies the framework

Outline

History of our project:

- Ph.d in Theoretical Computer Science 2003, algebraic specifications, logic and category theory
- Discussions with professor Uwe Wolter since 2000 about how to use theory in practice
- In 2000 Wolter introduced me to the Generalized Sketches framework, created in Latvia in the 90's, by Diskin ++
- 2003, we decided to try to use the GS approach for practical software engineering

...

- 2004 master student Ørjan Hatland started to implement a tool for GS, based on the .NET framework, (graduated 2006)
- Early 2006 Ph.d student Adrian Rutle started to investigate the subject and we get interested in MDE and the Eclipse framework
- 2006 Master student Stian Skjerveggen started to implement a graphical editor in Eclipse framework, (will graduate June 2008)

Outline

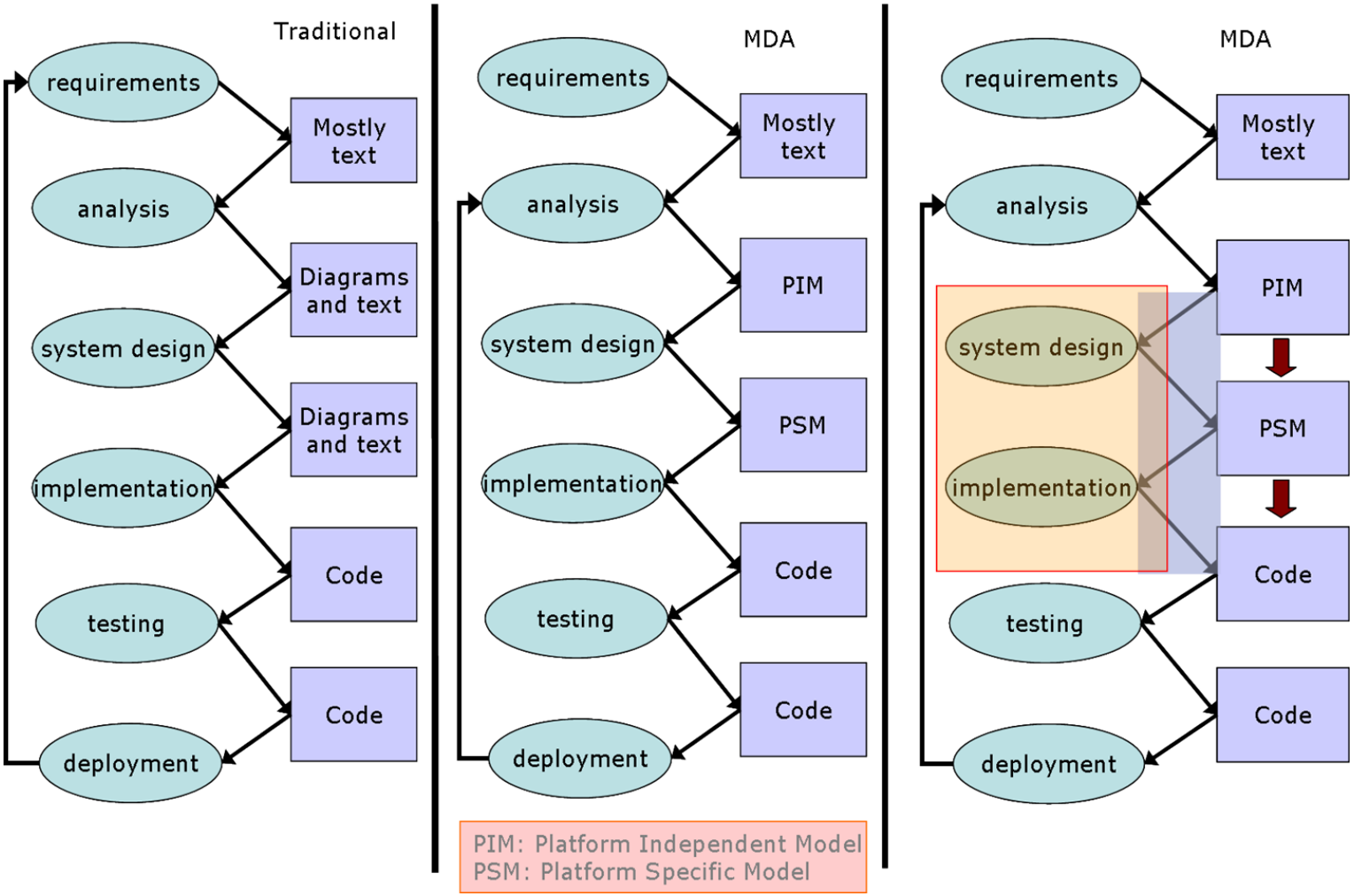
MDE

- Engineering techniques where models are first-class entities
- Evolved from the popularity of diagrammatic languages such as UML and ER and their popularity for specification and documentation of software systems
- Aims to raise the abstraction level of software development from text to models
- The development process is based on software models, not on plain text

Industry "standard" for MDE

- Model Driven Architecture (MDA) approach from Object Management Group (OMG) where:
  - First step is building platform independent domain models, PIM's
  - PIM's are refined to platform specific models PSM's
  - Refinement process more or less automatized as model transformations
  - Code generation and model refinement can be seen as model transformations
  - Reuse of software models by using model transformations

MDD vs Traditional Development Processes



Key point for success for MDE

Models are no longer only for documentation

- Models should be **both**:
  - intuitive for **software engineers** to intercept and work with
  - formal such that **machines** could have precise understanding and reason about the models

⇒ Diagrammatic approach enhance human understanding
⇒ Challenge to combine intuition with formalization

Key point for success for MDE

Models are no longer only for documentation

- Models should be **both**:
  - intuitive for **software engineers** to intercept and work with
  - formal such that **machines** could have precise understanding and reason about the models

⇒ Diagrammatic approach enhance human understanding

⇒ Challenge to combine intuition with formalization

Key point for success for MDE

Models are no longer only for documentation

- Models should be **both**:
  - intuitive for **software engineers** to intercept and work with
  - formal such that **machines** could have precise understanding and reason about the models

⇒ Diagrammatic approach enhance human understanding
⇒ Challenge to combine intuition with formalization

Outline

Some facts about UML

- UML is defacto industry standard for modeling
- UML can only express constraints on binary relations
- UML has serious issues regarding:
  - Semantics; UML models may be ambiguous and have semi-formal semantics
  - Complexity; UML uses 13 different types of diagrams
  - Expressibility; To express constraints over higher order relations one need to use string based logic, (Object Constraint Logic, OCL)

Formal modeling languages

- Nice semantics, proven abilities and several fundamental Software Engineering problems solved by use of formal methods
- Set based semantics (logic, algebras etc...)
- Hard for software engineers to apply in practice
- Lack of good software development tools based on formal approaches
- Only used by well trained experts
- High cost low productivity

Outline

DPL

- Aims to combine the intuition from graphical modeling languages with the semantic rigor of formal methods
- Based on Generalized Sketches
- Sketches is a graphical representation of category theory, constraints expressed by universal constructions (limits and co-limits) of diagrams
- Generalized Sketches is sketches extended with constraints on arbitrary diagrams, not only universal constructions
- In DPL, sketches are generalized and adapted to model driven software engineering

More on DPL

- DPL will be used as a framework for defining modeling languages and transformations between them in a formal graphical way
- We work continuously on Investigating, Analyzing, Adapting, Verifying and Evaluating DPL
- Potential to use the machinery from category to implement MDE tools

Outline

Diagrammatic specifications

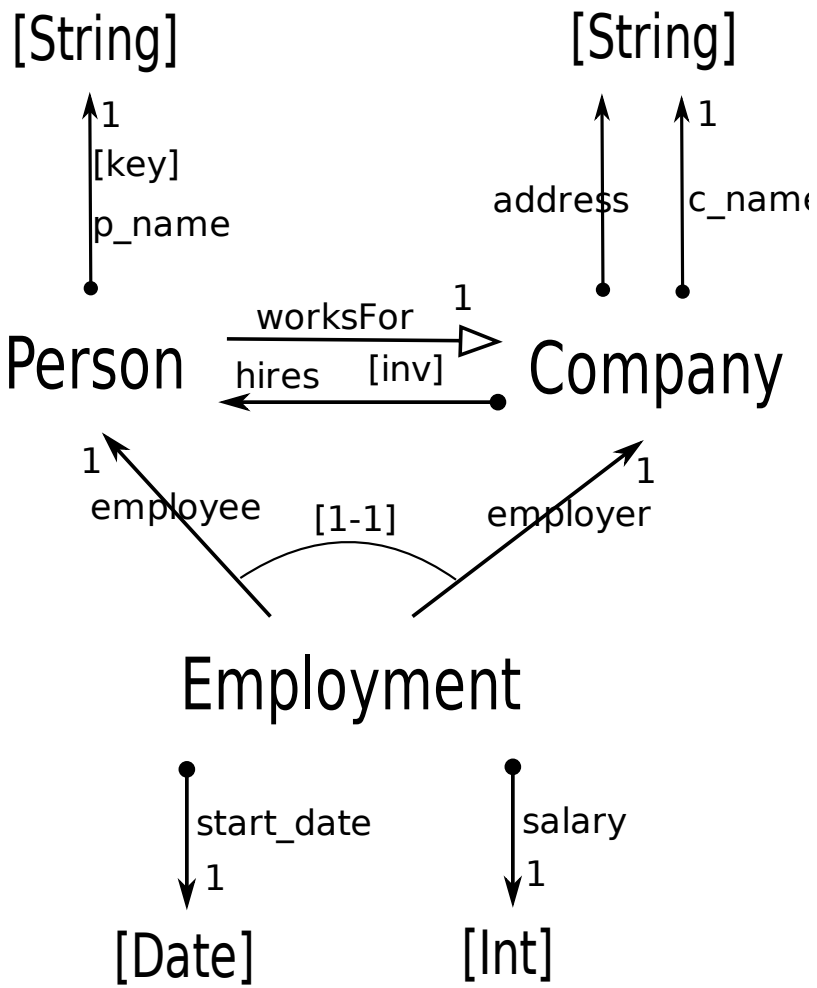
- Signature
  - Set of constraint (predicate) names Π with
  - An arity function $\alpha : \Pi \rightarrow$ **graph**
- Diagrammatic Specifications consist of
  - the underlying graph $\mathbf{G}(\mathbf{S})$ together with a
  - set of constraints $\mathbf{C_S}$ i.e. diagrams $\delta : \alpha(c_S) \rightarrow \mathbf{G}$, where $c_S \in \Pi$

Outline

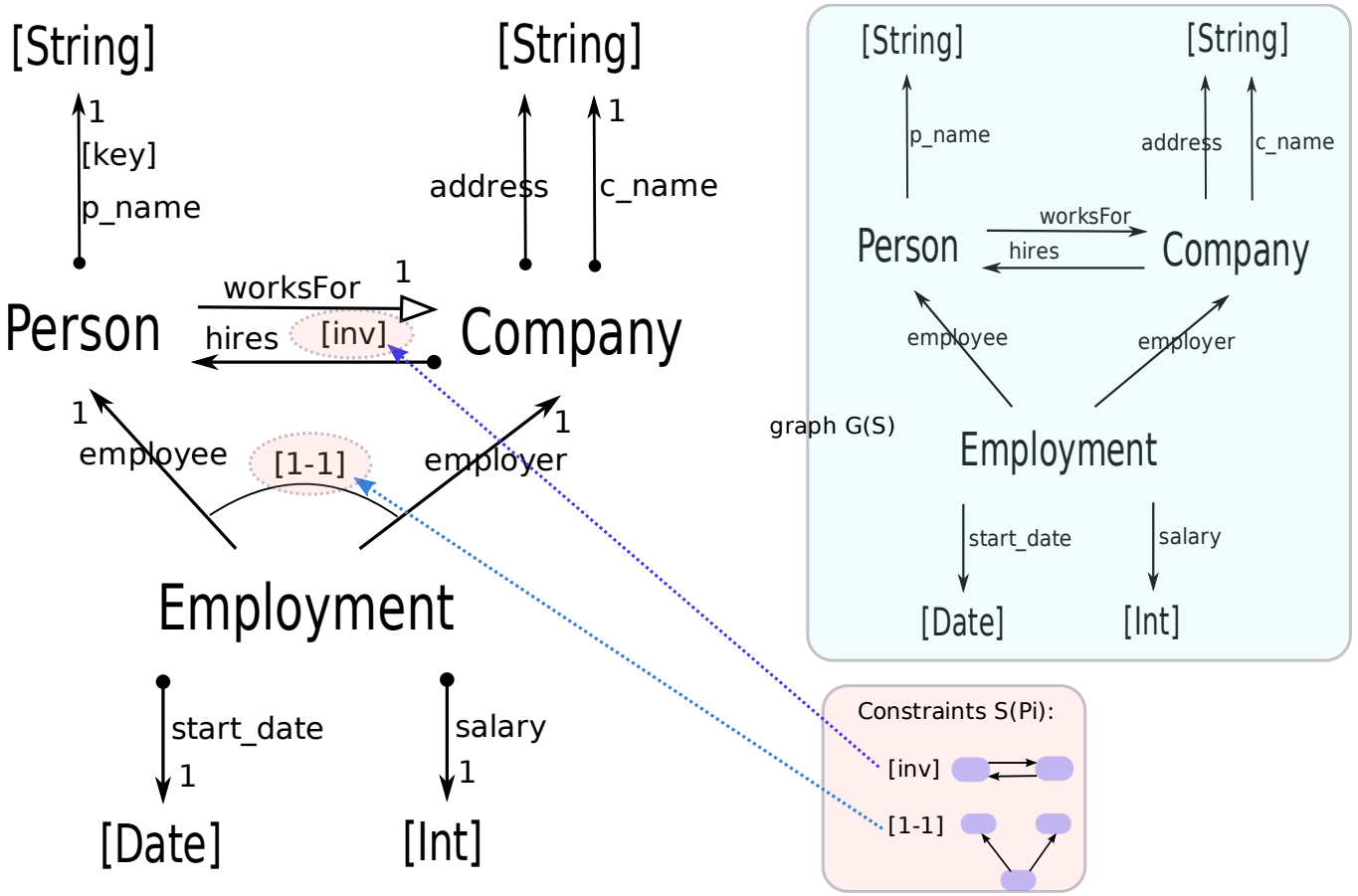
Signature – Example

Π	α	visualization	intended semantics
[total]	$1 \xrightarrow{x} 2$	$\boxed{A} \xrightarrow{f} \boxed{B}$	$\forall a \in A : \lvert f(a) \rvert \geq 1$
[key]	$1 \xrightarrow{x} 2$	$\boxed{A} \xrightarrow{f\ [key]} \boxed{B}$	$\forall a, a' \in A : a \neq a'$ implies $f(a) \neq f(a')$
[inv]	$1 \underset{y}{\overset{x}{\rightleftarrows}} 2$	$\boxed{A} \underset{g}{\overset{f}{\underset{[inv]}{\rightleftarrows}}} \boxed{B}$	$\forall a \in A$, $\forall b \in B : b \in f(a)$ iff $a \in g(b)$
[jointly-key] or [1-1]	$\begin{tikzcd} 1 \arrow[r, "x"] \arrow[d, "y"] & 2 \\ 3 & \end{tikzcd}$	$\begin{tikzcd} \boxed{A} \arrow[r, "f"] \arrow[d, "g"'] & \boxed{B} \arrow[dl, bend left, "{[1-1]}"] \\ \boxed{C} & \end{tikzcd}$	$\forall a, a' \in A :$ $a \neq a'$ *implies* $f(a) \neq f(a')$ or $g(a) \neq g(a')$

Diagrammatic Specification – Example

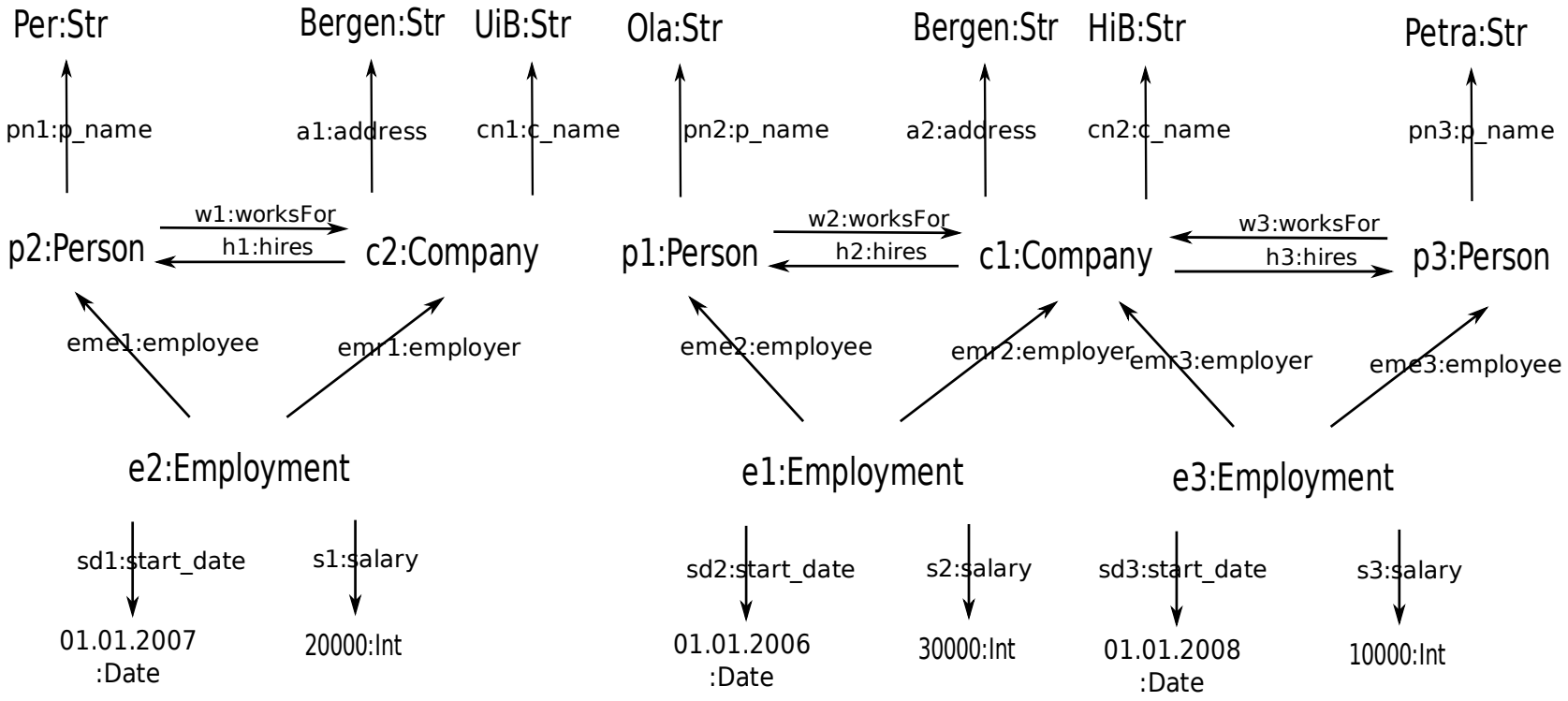


Diagrammatic Specification – Example



Outline

Instance of Diagrammatic Specification – Example



Instance of specification

- The semantic interpretation of $\Sigma := (\Pi, \alpha)$ assigns to each $p \in \Pi$ a set $[\![p]\!]$ of graph homomorphisms $\tau : O \to \alpha(p)$, where O may vary over all graphs.
- An instance of a specification S is a graph I together with a graph homomorphism $\iota : I \to G(S)$, such that $\iota^* \in [\![p]\!]$, i.e. ι^* is a valid instance of p for each constraint (p, δ) where $\delta : \alpha(p) \to G(S)$.
  ι^* is given by the pullback diagram:

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

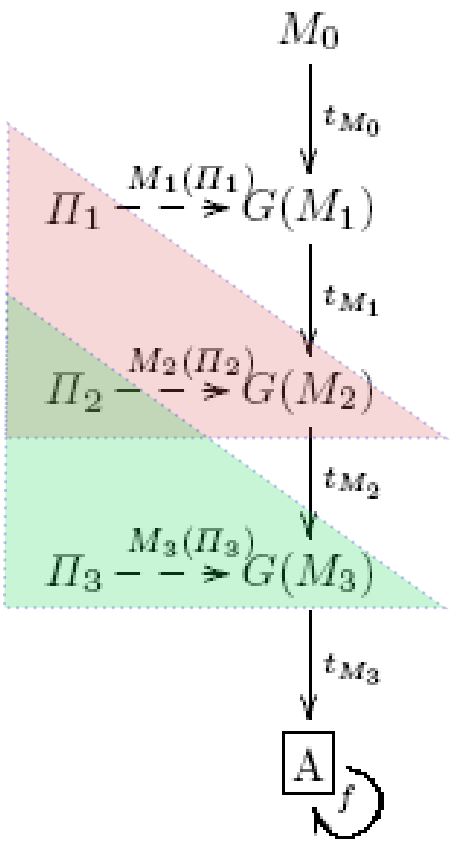
Outline

OMG Metamodel Levels

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

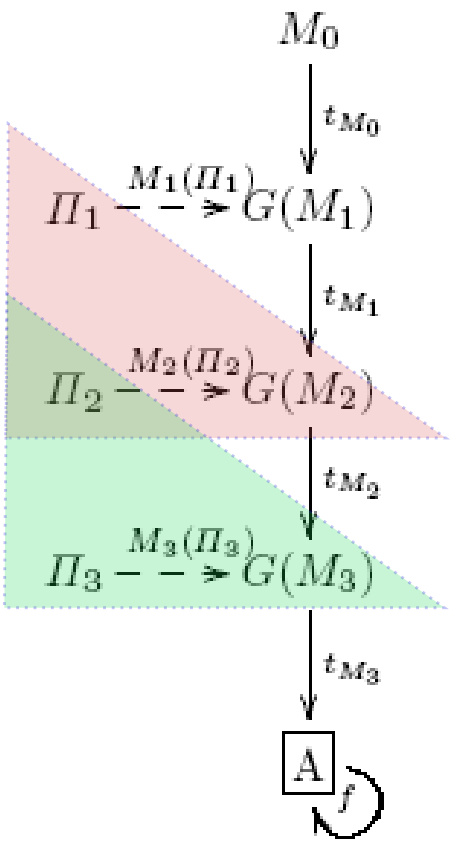
Outline

DPL Methodology and Modeling Formalisms



- Modeling Formalism $F_2 = (\Sigma_1, M_2, \Sigma_2)$
- Meta-formalism $F_3 = (\Sigma_2, M_3, \Sigma_3)$
- F_3 is reflexive iff $(\Sigma_3 \sqsubseteq \Sigma_2)$

DPL Methodology and Modeling Formalisms

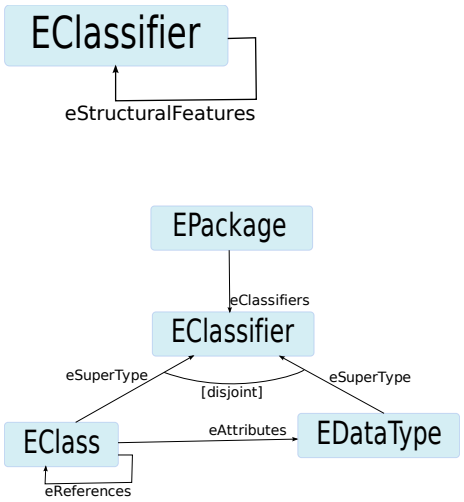


- Modeling Formalism $F_2 = (\Sigma_1, M_2, \Sigma_2)$
- Meta-formalism $F_3 = (\Sigma_2, M_3, \Sigma_3)$
- F_3 is reflexive iff $(\Sigma_3 \sqsubseteq \Sigma_2)$

Outline

Ecore model expressed in DPL

Ecore is the metamodel of the Eclipse Modeling Framework, EMF

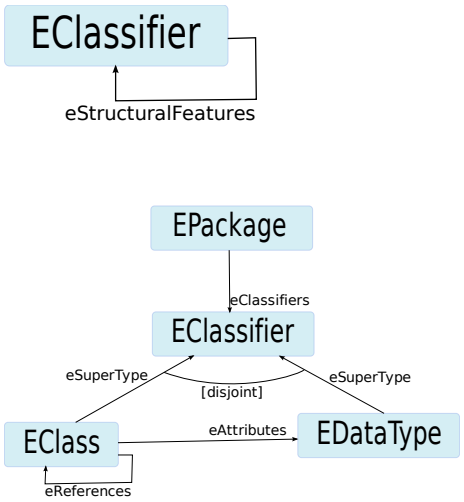


- Meta-formalism E_3 (EClassifier, EReference), essentially **Graph**
- Modeling Formalism, E_2 (EPackage), typed over E_3:
  $t(\texttt{EPackage}) = t(\texttt{EClassifier}) = t(\texttt{EClass}) = t(\texttt{EDataType}) = \texttt{EClassifier}$,

  $t(\texttt{eClassifiers}) = t(\texttt{eType}) = t(\texttt{eAttribues}) = t(\texttt{eReferences}) = \texttt{eStructuralFeatures}$

Ecore model expressed in DPL

Ecore is the metamodel of the Eclipse Modeling Framework, EMF

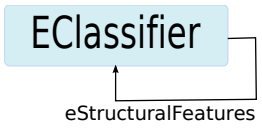


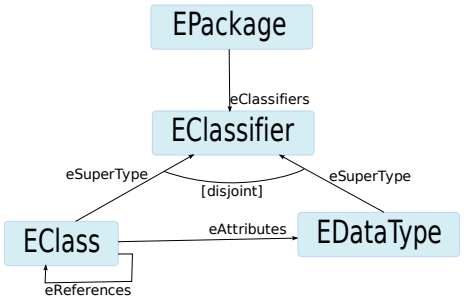
- Meta-formalism E_3 (EClassifier, EReference), essentially **Graph**
- Modeling Formalism, E_2 (EPackage), typed over E_3:
  $t(\texttt{EPackage}) = t(\texttt{EClassifier}) = t(\texttt{EClass}) = t(\texttt{EDataType}) = \texttt{EClassifier}$,

  $t(\texttt{eClassifiers}) = t(\texttt{eType}) = t(\texttt{eAttribues}) = t(\texttt{eReferences}) = \texttt{eStructuralFeatures}$

Ecore model expressed in DPL

Ecore is the metamodel of the Eclipse Modeling Framework, EMF





- Meta-formalism E_3 (EClassifier, EReference), essentially **Graph**
- Modeling Formalism, E_2 (EPackage), typed over E_3:
  $t(\texttt{EPackage}) = t(\texttt{EClassifier}) = t(\texttt{EClass}) = t(\texttt{EDataType}) = \texttt{EClassifier}$,

  $t(\texttt{eClassifiers}) = t(\texttt{eType}) = t(\texttt{eAttribues}) = t(\texttt{eReferences}) = \texttt{eStructuralFeatures}$

Tools

- Tools implemented as plug-ins to the Eclipse platform
- Need tools for:
  - Drawing of models, partly finished June 2008
  - Serialization of models, work started
  - Validation of models, future research project
  - Model transformation tools, reasearch started

Drawing tool

- Built over Eclipse Graphical Modeling Framework, GMF
- Will build separate tool for building signatures, conforms to a modeling language
- Tool for creating models over signatures, i.e. modeling tool in the chosen language
- Today drawing tool still some issues with diagrams with more than 3 nodes

Summary

- The industry have a lot of hidden knowledge, they have no need for formalization
- Modeling community work with fibrations, not index as we are trained to in mathematics
- Practical problems seems easy in theory but you really get dirt on your fingers
- We have obtained good understanding on modeling, meta modeling and model transformation

Further work

- Work on classification of model transformations, what properties has different class of transformations?
- Implement different formalisms in DPL (UML, XML, ...)
- Implement model transformation tool with code generation and cross platform translation

Model Transformation Concepts

- Model Transformation = set of Transformation Rules + Coordinations ($\Rightarrow$ operational)
- Model Transaction = declarative definition of the transformation
- Transformation Engine = executes the rules

Transformation Rules

$$
\begin{tikzcd}
 & P \arrow[dl, "\iota_P"'] \arrow[d, "m"] \arrow[r, mapsto] \arrow[dr, phantom, "[=]"] & P' \arrow[d, "[\![t]\!](m)"] \arrow[dr, "\iota_{P'}"] & \\
G(M_1) & I \arrow[l, "\iota"] \arrow[r, dotted, "trace"'] & I' \arrow[r, "\iota'"'] & G(M_2)
\end{tikzcd}
$$

- Input and output patterns P and P'
- Input and out put instances I and I'
- The models M_1 and M_2
- The matches m and $[\![t]\!](m)$

Transformation Rules

$$
\begin{tikzcd}
 & P \arrow[dl, "\iota_P"'] \arrow[d, "m"] \arrow[r, mapsto] \arrow[dr, phantom, "[=]"] & P' \arrow[d, "[\![t]\!](m)"] \arrow[dr, "\iota_{P'}"] & \\
G(M_1) & I \arrow[l, "\iota"] \arrow[r, dotted, "trace"'] & I' \arrow[r, "\iota'"'] & G(M_2)
\end{tikzcd}
$$

- Input and output patterns P and P'
- Input and out put instances I and I'
- The models M_1 and M_2
- The matches m and $[\![t]\!](m)$