Towards a Formal Diagrammatic Framework for Model Driven Architecture (MDA)

Adrian Rutle

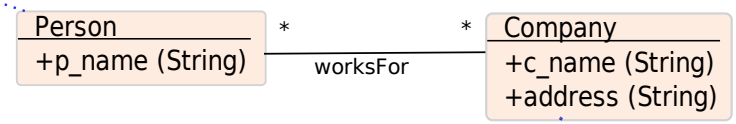
Department of Computer Engineering
Faculty of Engineering
Bergen University College
Norway

08.07.2008
ECOOP 2008, Paphos, Cyprus

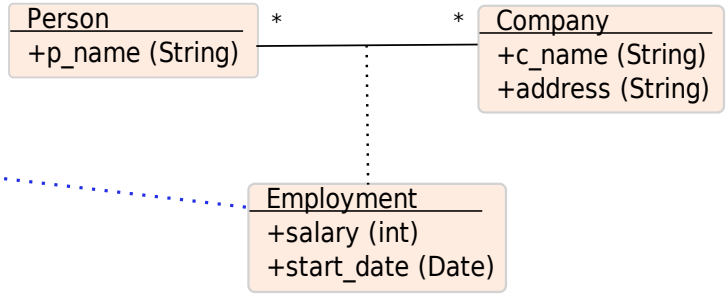
Motivation

- Raise of abstraction level of programming languages ⇒ Modeling vs coding ⇒ MDA ⇒ Models and Model Transformations

```
Public class Person{
  public String p_name;
  public Company worksFor;
  //constraints here
  public Person(...){...}
}
```




```
Public class Company{
  public String c_name;
  public String address;
  public List <Person> hires;
  //constraints here
  public Company(...){...}
  }
```

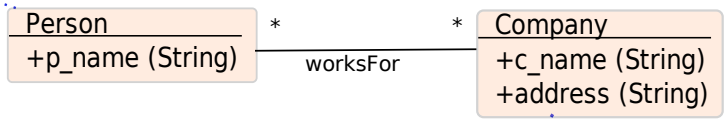



```
Public class Employment{
  public Date start_date;
  public int salary;
  public Company employer;
  public Person employee;
  //constraints here
  public Employment(...){...}
}
```

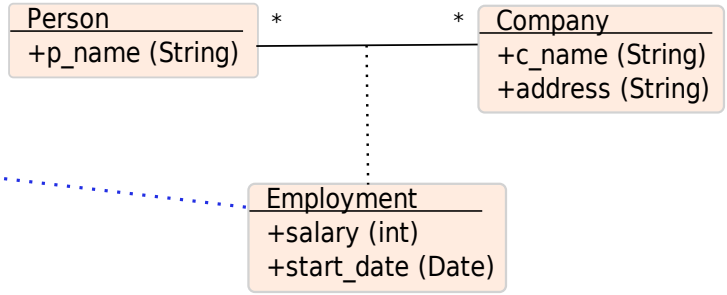
Motivation

- Raise of abstraction level of programming languages ⇒ Modeling vs coding ⇒ MDA ⇒ Models and Model Transformations

```
Public class Person{
  public String p_name;
  public Company worksFor;
  //constraints here
  public Person(...){...}
}
```




```
Public class Company{
  public String c_name;
  public String address;
  public List <Person> hires;
  //constraints here
  public Company(...){...}
  }
```

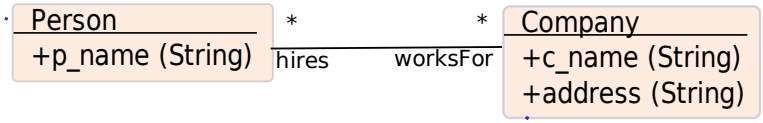



```
Public class Employment{
  public Date start_date;
  public int salary;
  public Company employer;
  public Person employee;
  //constraints here
  public Employment(...){...}
}
```

Motivation

- Raise of abstraction level of programming languages ⇒ Modeling vs coding ⇒ MDA ⇒ Models and Model Transformations

```
Public class Person{
  public String p_name;
  public List<Company> worksFor;
  //constraints here
  public Person(...){...}
}
```




```
Public class Company{
  public String c_name;
  public String address;
  public List <Person> hires;
  //constraints here
  public Company(...){...}
  }
```

Instances:

p1	c1
p2	c2
p2	c1
p1	c1

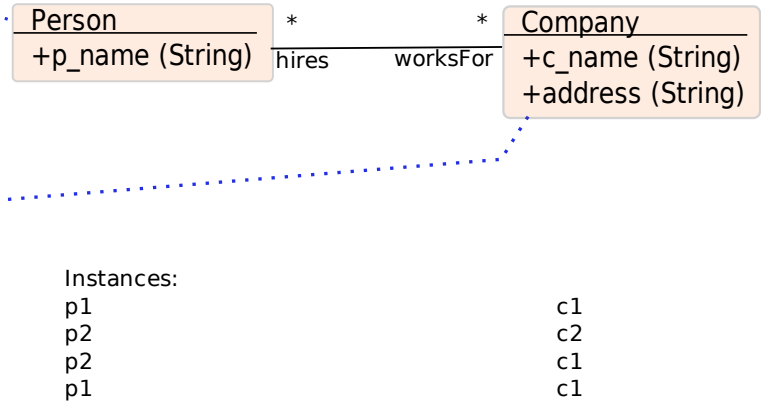
- Need for OCL to put constraints ⇒ Mixing text and diagrams

Motivation

- Raise of abstraction level of programming languages ⇒ Modeling vs coding ⇒ MDA ⇒ Models and Model Transformations

```
Public class Person{
  public String p_name;
  public List<Company> worksFor;
  //constraints here
  public Person(...){...}
}
```

```
Public class Company{
  public String c_name;
  public String address;
  public List <Person> hires;
  //constraints here
  public Company(...){...}
  }
```



- Need for OCL to put constraints ⇒ Mixing text and diagrams

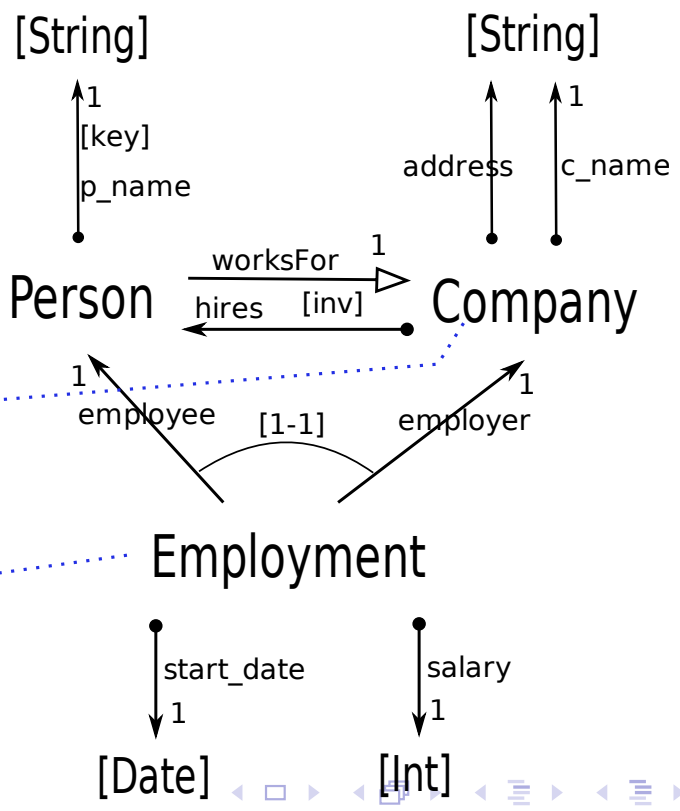
Motivation

- Raise of abstraction level of programming languages ⇒ Modeling vs coding ⇒ MDA ⇒ Models and Model Transformations

```
Public class Person{
   public String p_name;
   public Company worksFor;
   //constraints here
   public Person(...){...}
}
```

```
Public class Company{
   public String c_name;
   public String address;
   public List <Person> hires;
   //constraints here
   public Company(...){...}
   }
```

```
Public class Employment{
   public Date start_date;
   public int salary;
   public Company employer;
   public Person employee;
   //constraints here
   public Employment(...){...}
}
```



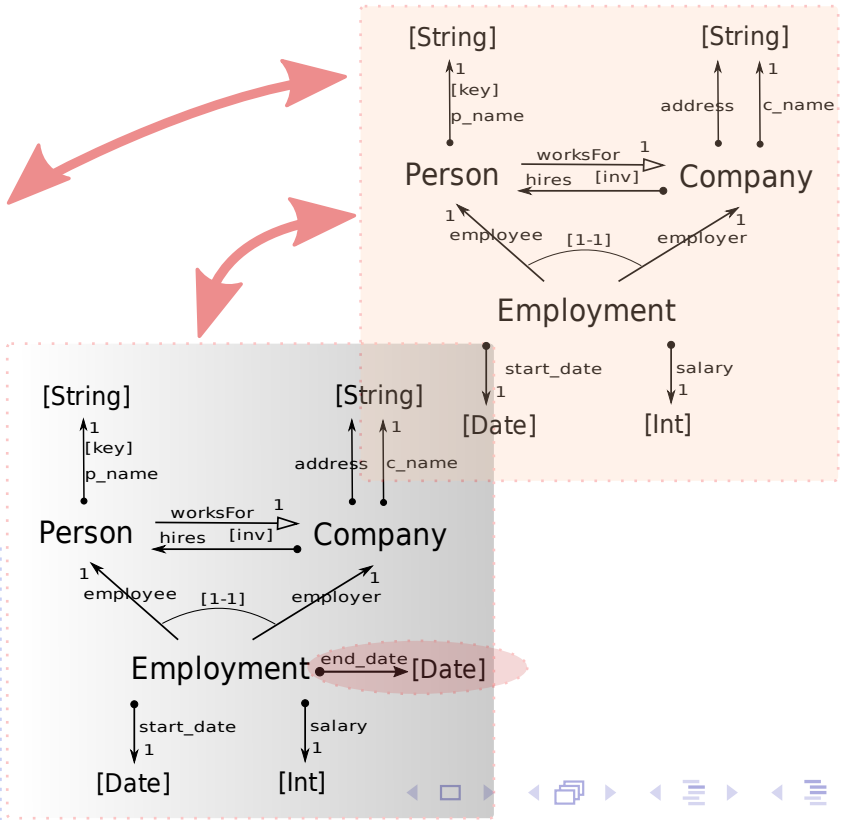
Motivation

- MDA ⇒ Models and Model Transformations ⇒ Code generation, model integration, model versioning, reverse engineering ...

```
Public class Person{
   public String p_name;
   public Company worksFor;
   //constraints here
   public Person(...){...}
}
```

```
Public class Company{
   public String c_name;
   public String address;
   public List <Person> hires;
   //constraints here
   public Company(...){...}
   }
```

```
Public class Employment{
   public Date start_date;
   public int salary;
   public Company employer;
   public Person employee;
   //constraints here
   public Employment(...){...}
}
```



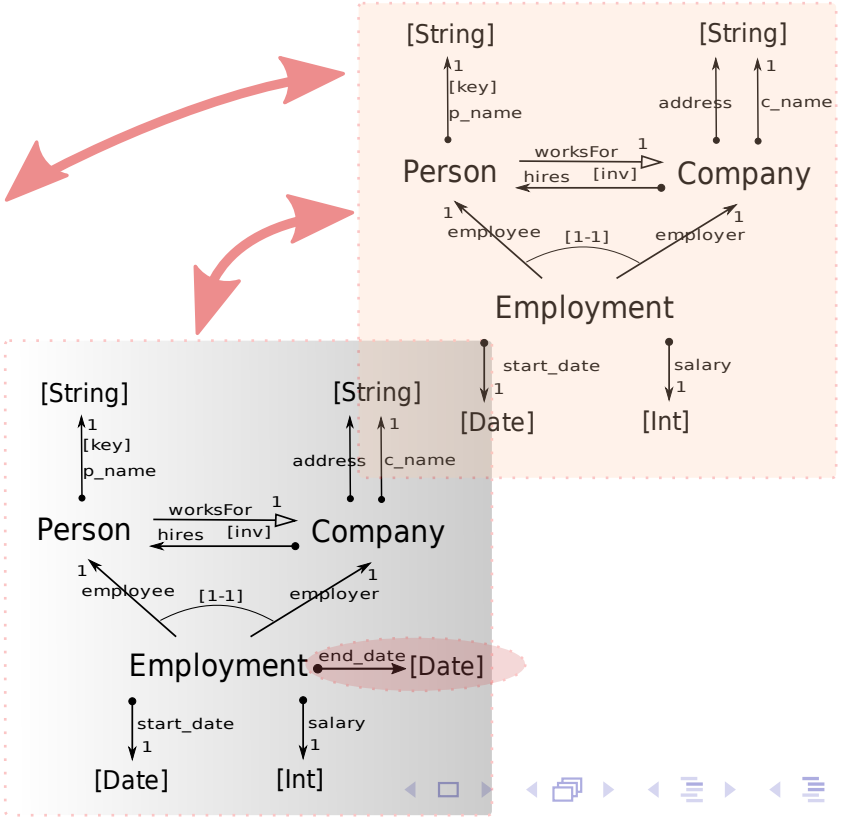
Motivation

- MDA ⇒ Models and Model Transformations ⇒ Code generation, model integration, model versioning, reverse engineering ...

```
Public class Person{
   public String p_name;
   public Company worksFor;
   //constraints here
   public Person(...){...}
}
```

```
Public class Company{
   public String c_name;
   public String address;
   public List <Person> hires;
   //constraints here
   public Company(...){...}
   }
```

```
Public class Employment{
   public Date start_date;
   public int salary;
   public Company employer;
   public Person employee;
   //constraints here
   public Employment(...){...}
}
```



Key-points for the Success of MDA

Models are no longer only for documentation

- Models should be **both**:
  - intuitive to ease their interception by **software engineers** and **domain experts**
  - formal such that **machines** can have a precise understanding of (and able to reason about) the models

⇒ Diagrammatic approach enhances human understanding
⇒ One challenge is to combine intuition with formalization
⇒ Another challenge is to implement the formalization approach

Key-points for the Success of MDA

Models are no longer only for documentation

- Models should be **both**:
  - intuitive to ease their interception by **software engineers** and **domain experts**
  - formal such that **machines** can have a precise understanding of (and able to reason about) the models

⇒ Diagrammatic approach enhances human understanding

⇒ One challenge is to combine intuition with formalization

⇒ Another challenge is to implement the formalization approach

Key-points for the Success of MDA

Models are no longer only for documentation

- Models should be **both**:
  - intuitive to ease their interception by **software engineers** and **domain experts**
  - formal such that **machines** can have a precise understanding of (and able to reason about) the models

⇒ Diagrammatic approach enhances human understanding
⇒ One challenge is to combine intuition with formalization
⇒ Another challenge is to implement the formalization approach

Key-points for the Success of MDA

Models are no longer only for documentation

- Models should be **both**:
  - intuitive to ease their interception by **software engineers** and **domain experts**
  - formal such that **machines** can have a precise understanding of (and able to reason about) the models

⇒ Diagrammatic approach enhances human understanding
⇒ One challenge is to combine intuition with formalization
⇒ Another challenge is to implement the formalization approach

Outline

UML vs Formal Modeling Languages

UML ...

- is considered *de facto* industry standard for modeling
- has serious issues regarding semantics, complexity, expressibility (uses OCL for constraints on higher order relations)

Formal Modeling Languages ...

- has precise semantics and proven abilities
- has set based semantics, while models are graph based
- is hard for software engineers to apply in practice
- lacks good software development tools

Outline

Our Approach: Diagrammatic Predicate Logic (DPL)

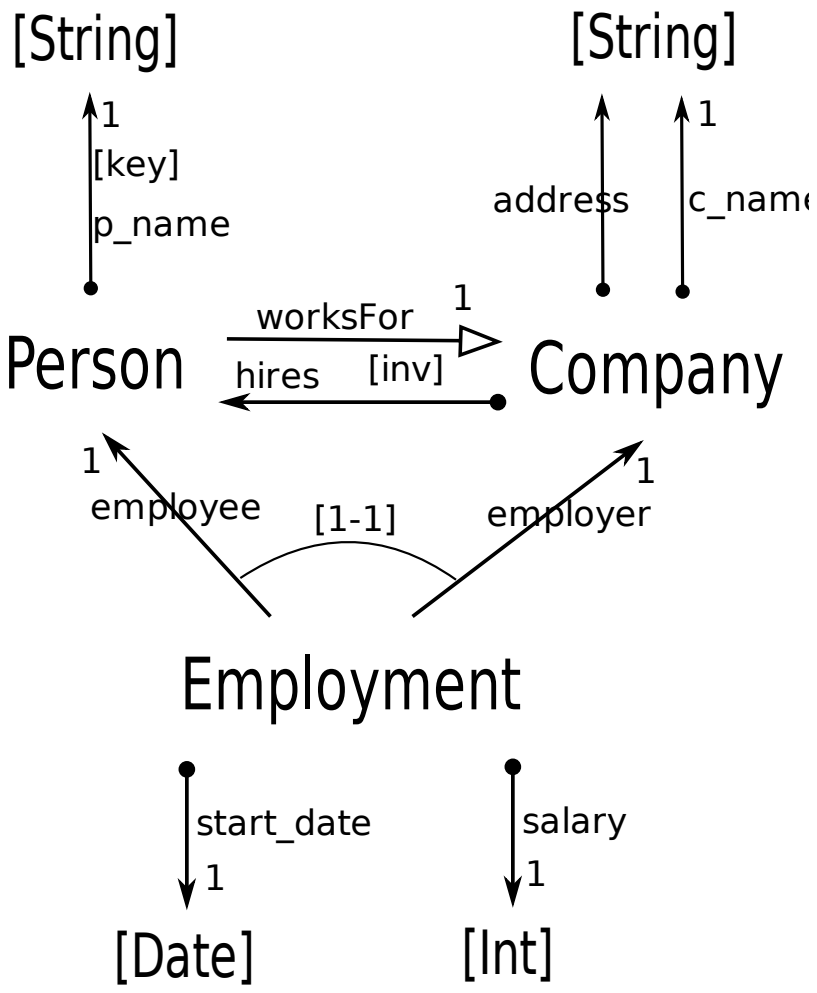
- Aims to combine the intuition from graphical modeling languages with the semantic rigor of formal methods
- Based on Generalized Sketches/Category Theory
- In DPL, sketches are generalized and adapted for use in MDA
- Potentials to use the machinery from category theory to implement MDE tools

Outline

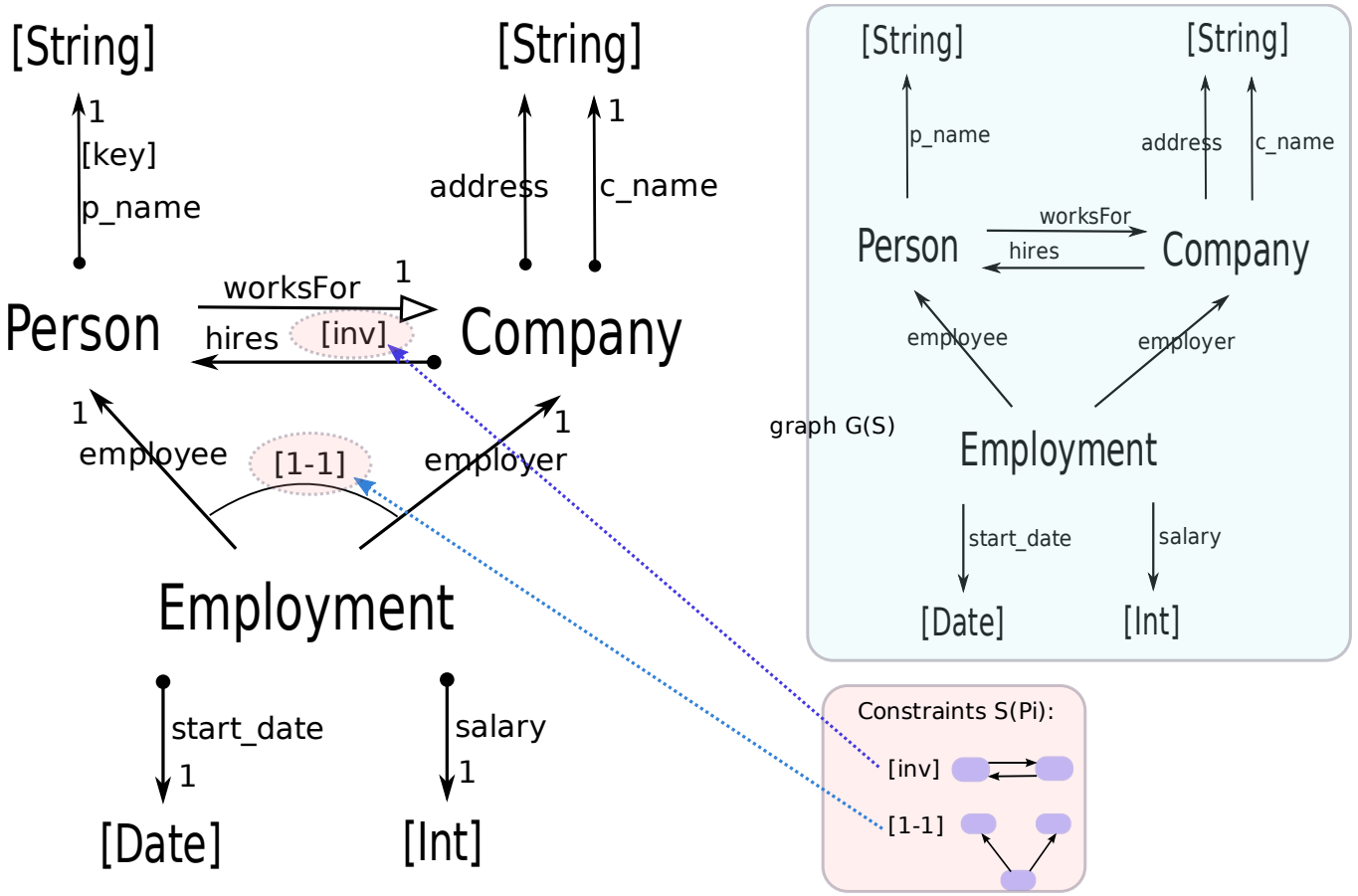
Signature – Example

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

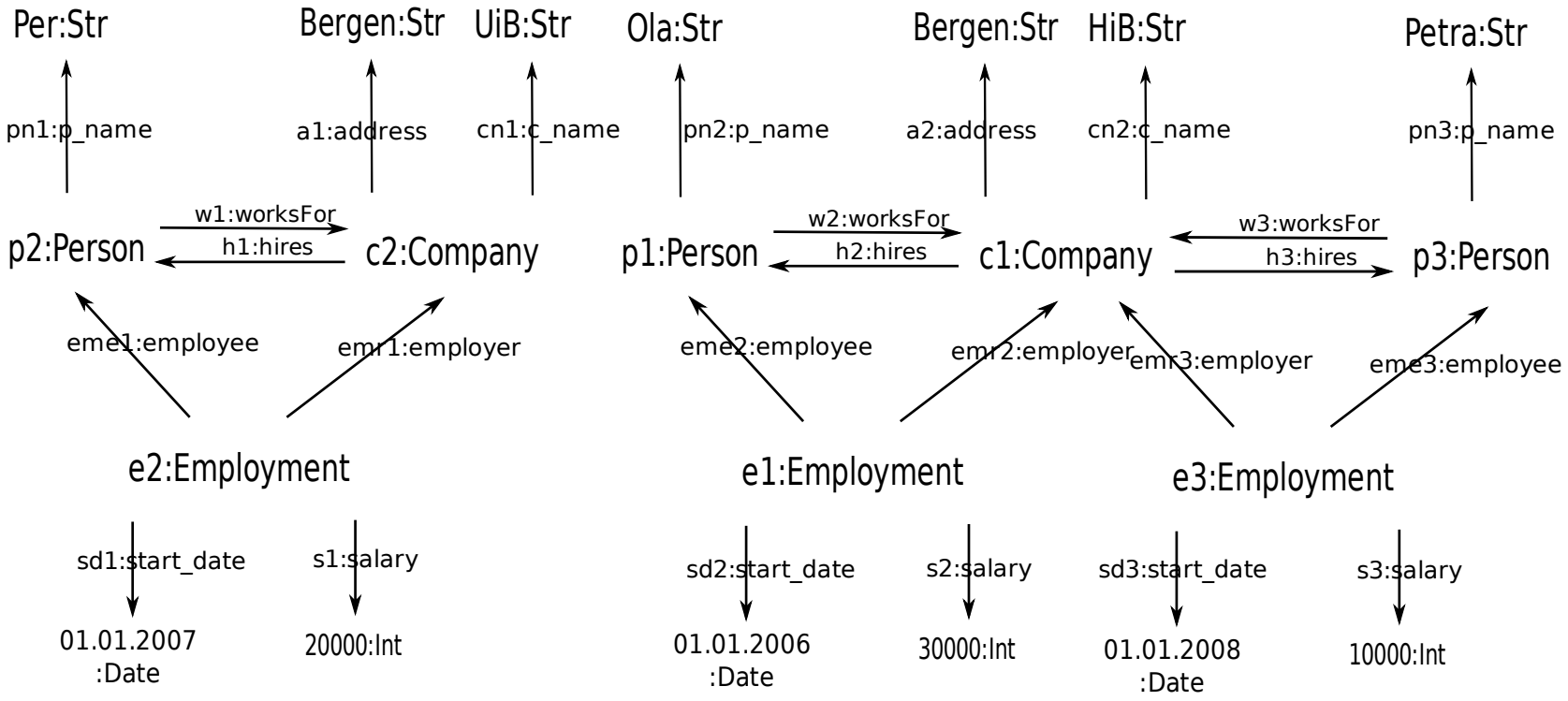
Diagrammatic Specification – Example



Diagrammatic Specification – Example

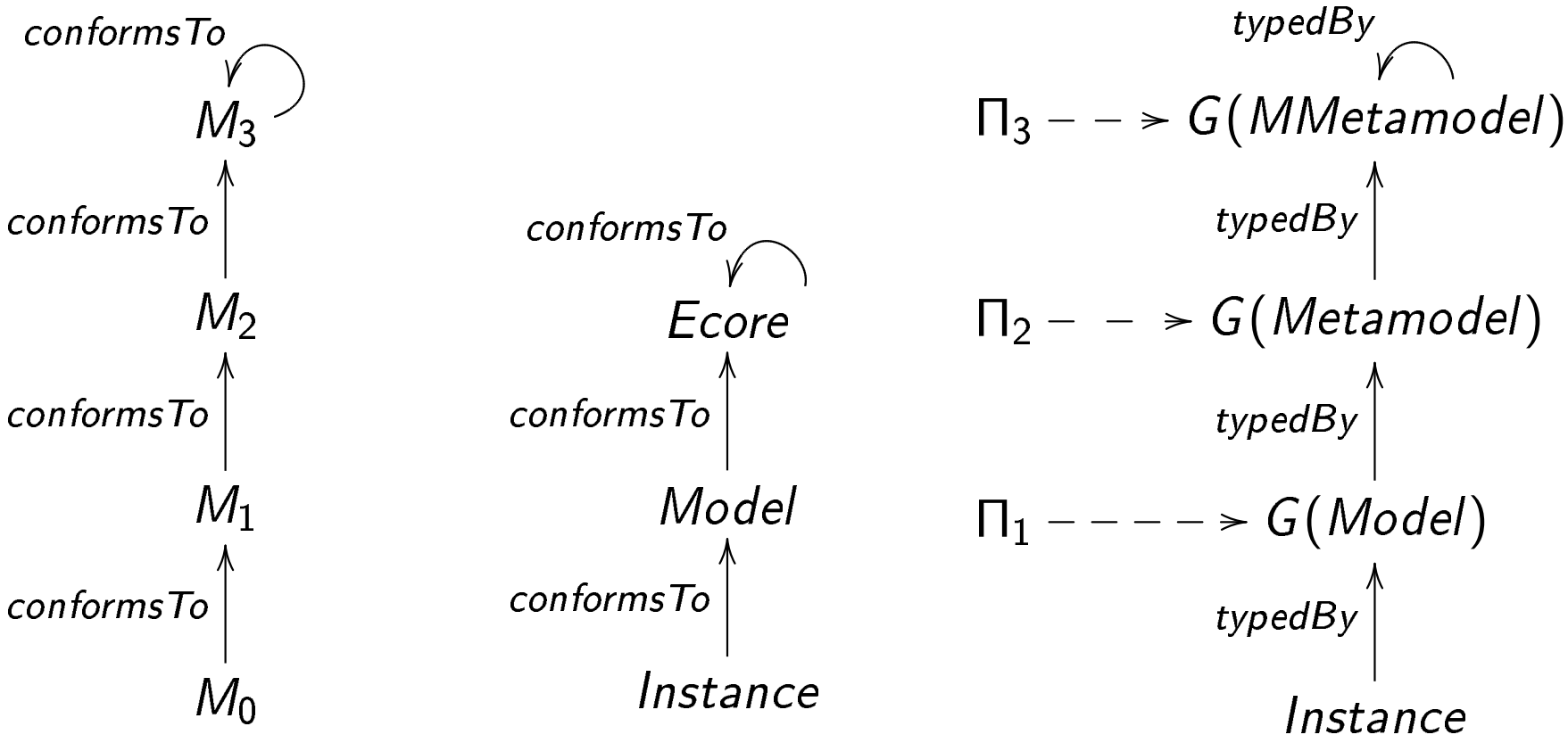


Instance of Diagrammatic Specification – Example



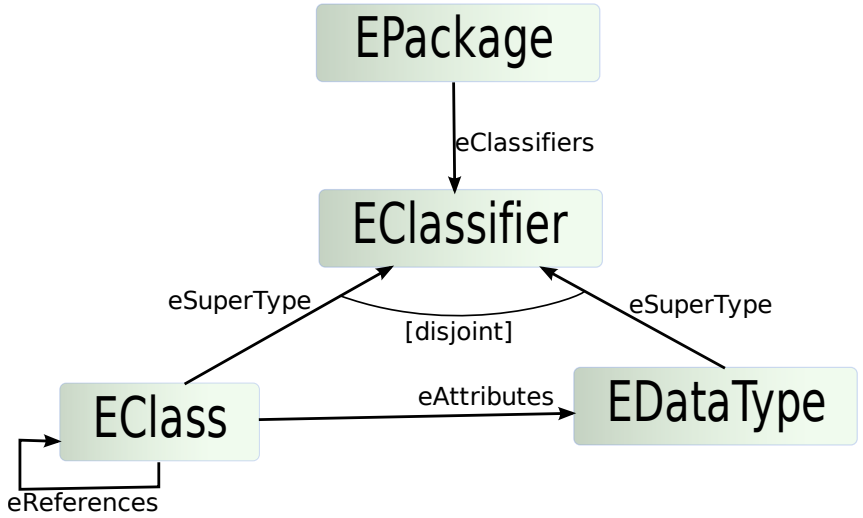
Outline

DPL Methodology and Modeling Formalisms

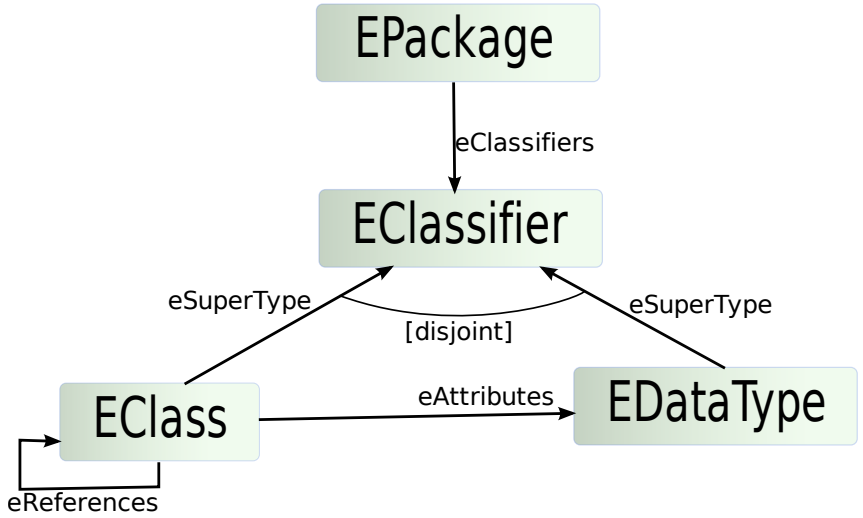


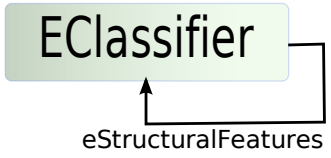
Outline

Ecore model expressed in DPL

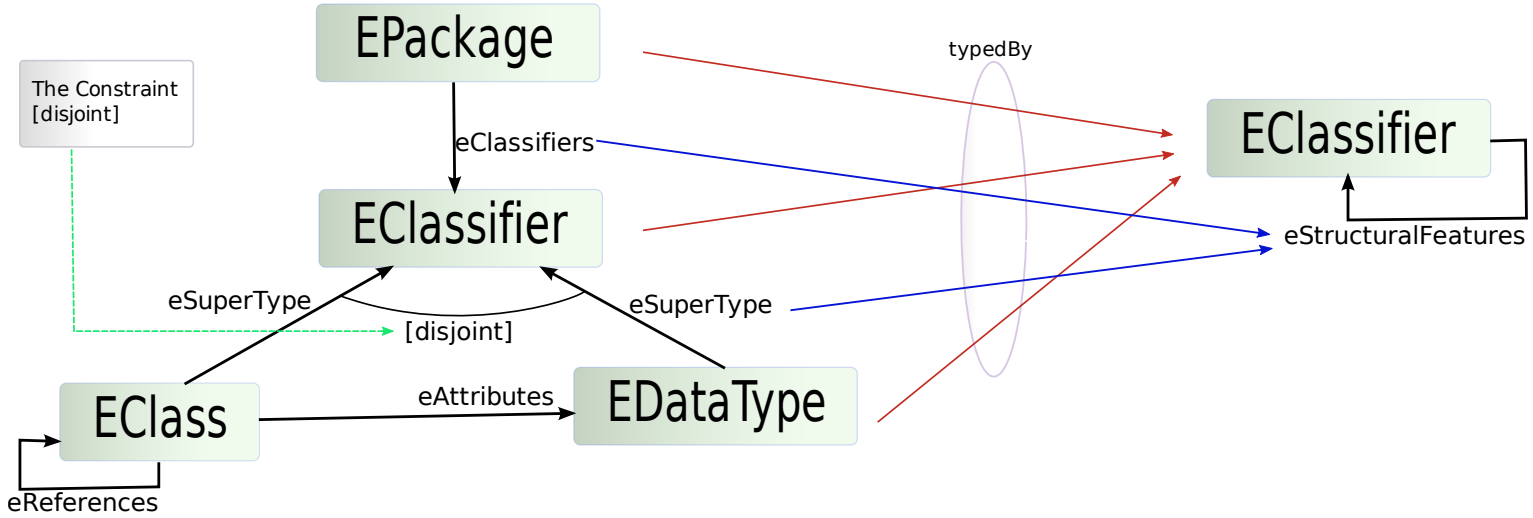


Ecore model expressed in DPL

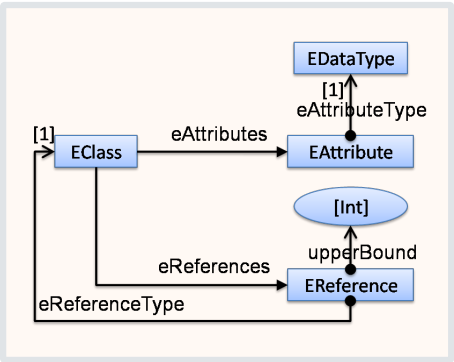




Ecore model expressed in DPL

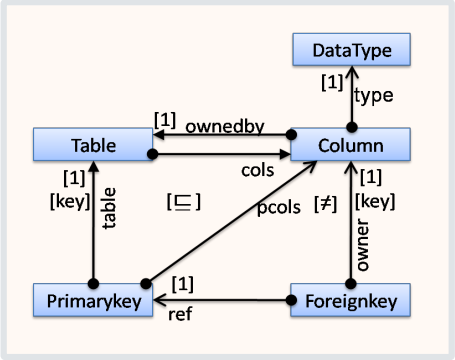


Model Transformation

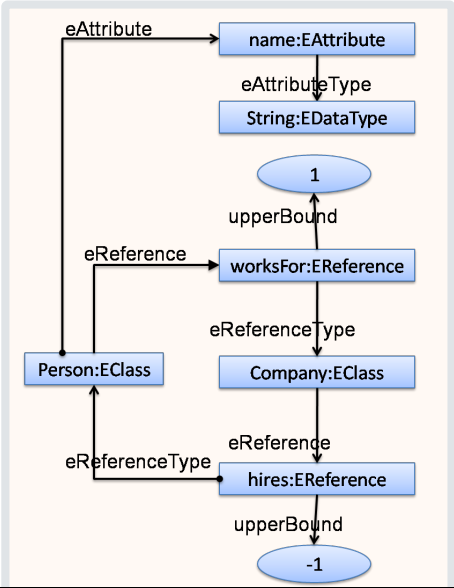


MT Language

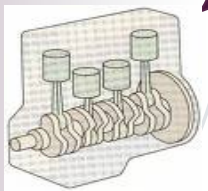
MT Definition



conformsTo

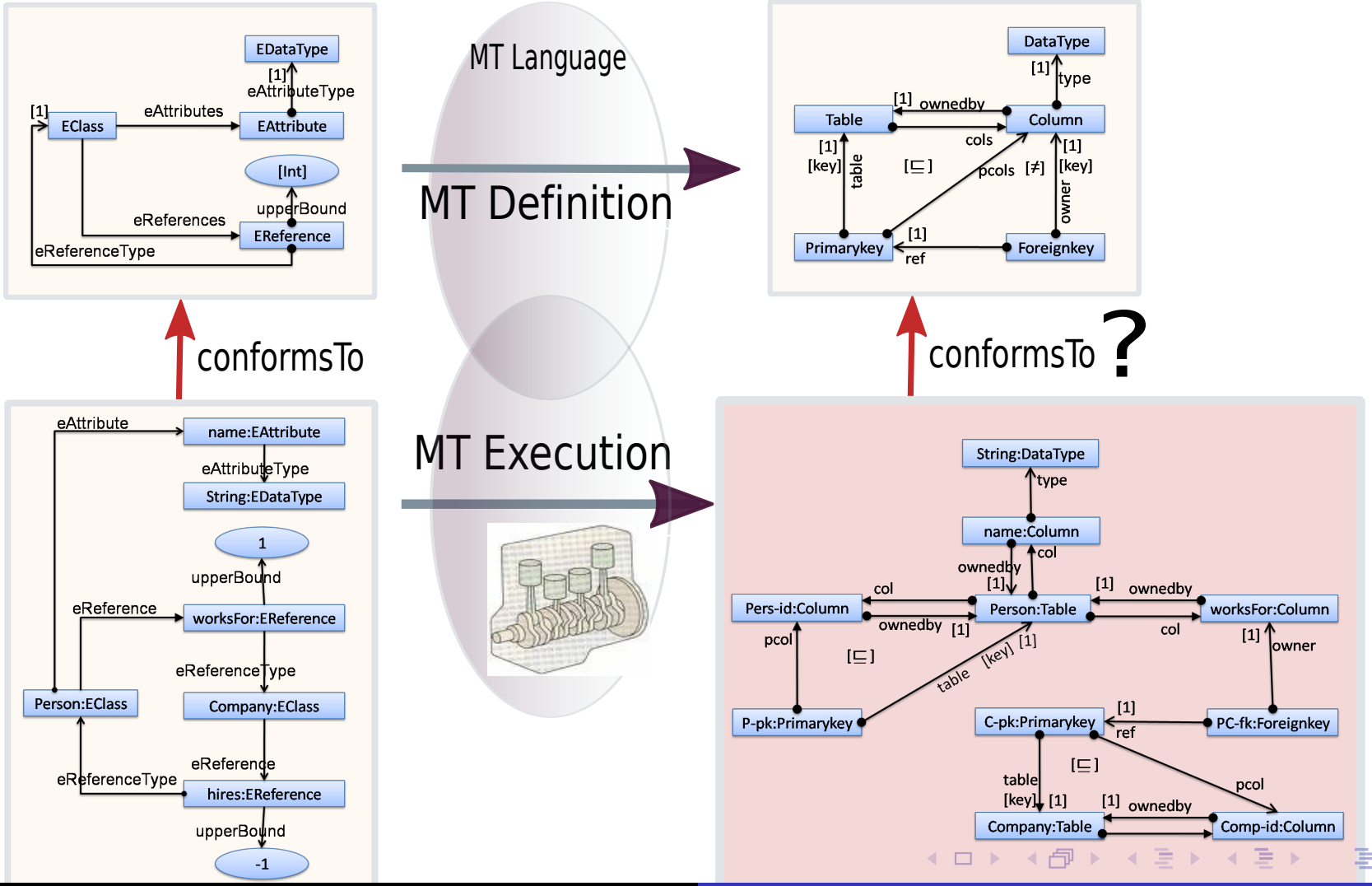


MT Execution



?

Model Transformation



Outline

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

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

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

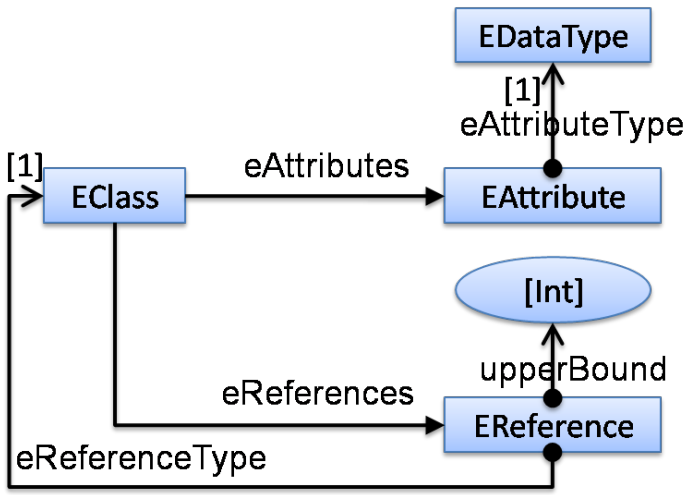
Transformation Rules

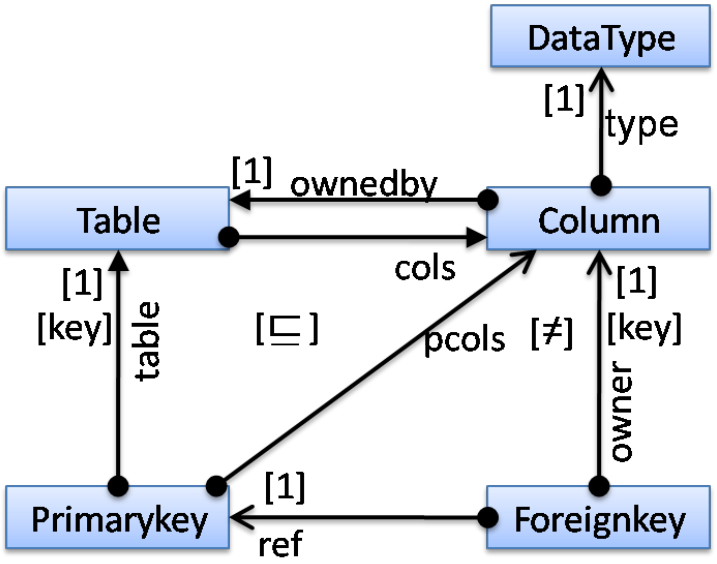
$$
\begin{tikzcd}
 & P \arrow[dl, "\iota_P"'] \arrow[d, "m"] \arrow[r, mapsto] \arrow[dr, phantom, "[=]"] & P' \arrow[d, "[\![t]\!](m)"] \arrow[dr, "\iota_{P'}"] & \\
G(M_1) & I \arrow[l, "\iota"] & I' \arrow[l, dotted, "trace"] \arrow[r, "\iota'"'] & G(M_2)
\end{tikzcd}
$$

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

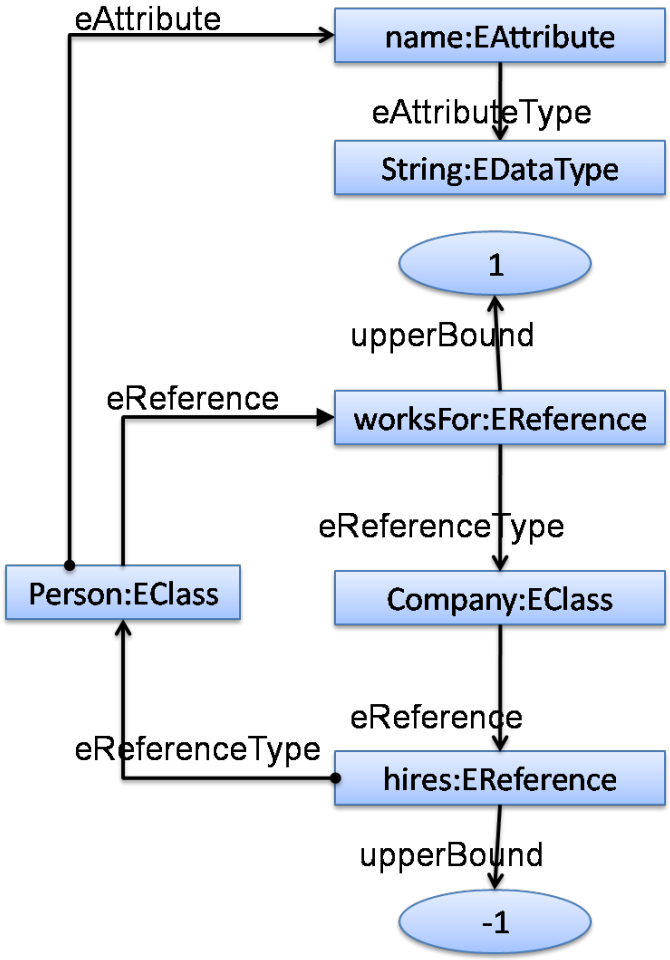
Outline

Model Transaction: Ecore to RDB

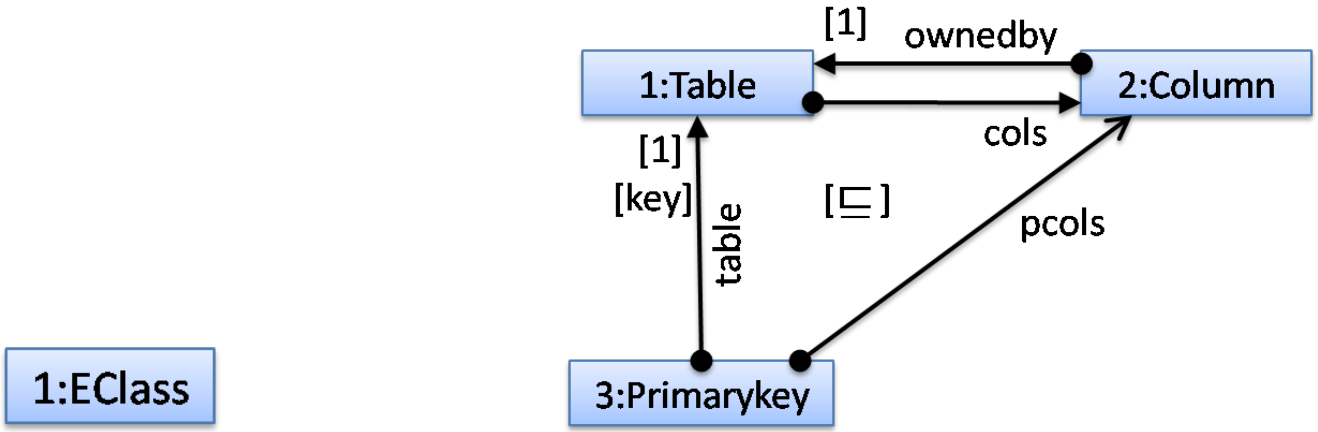




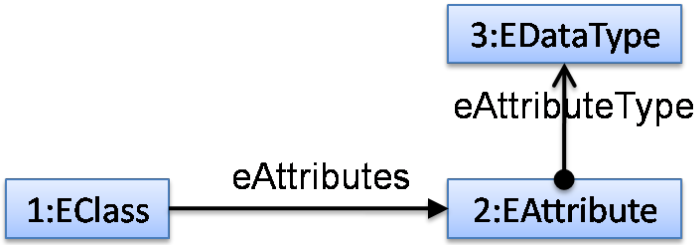
The Input EMF Model

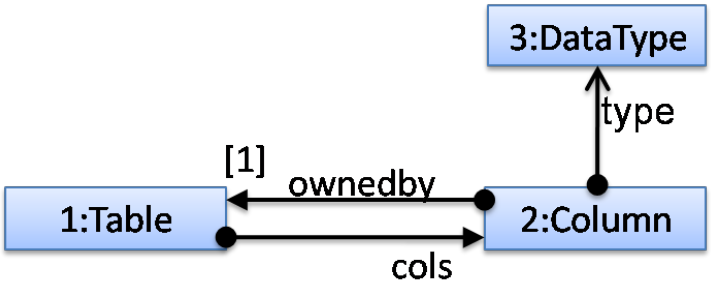


Rule1: EClass to Table + PrimaryKey

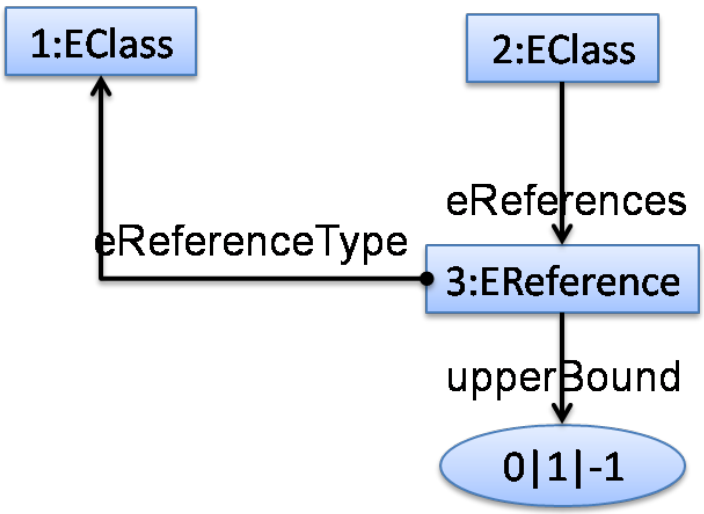


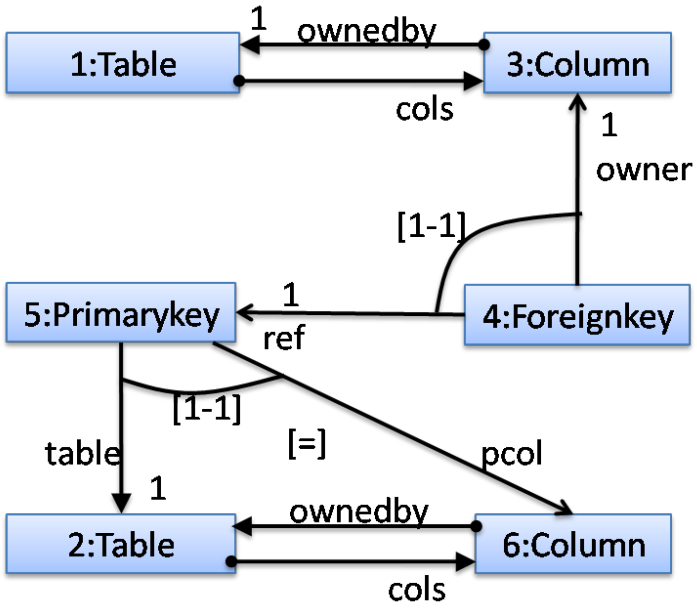
Rule2: EAttribute to Column



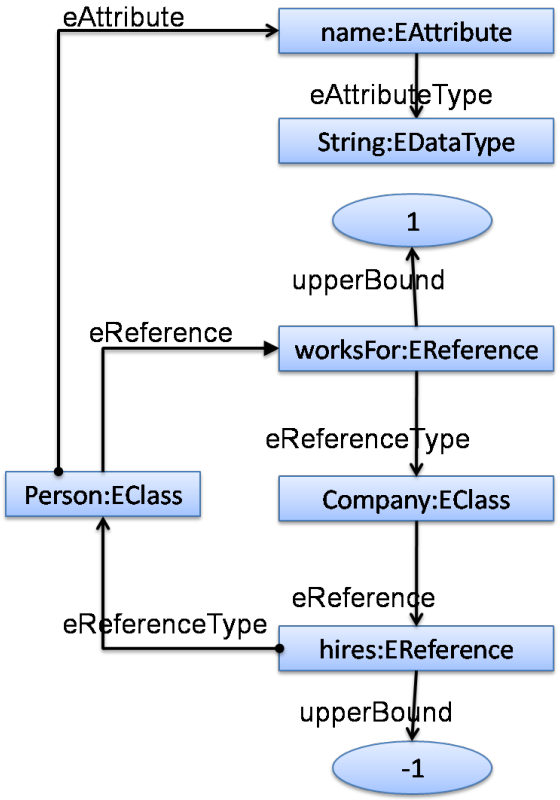


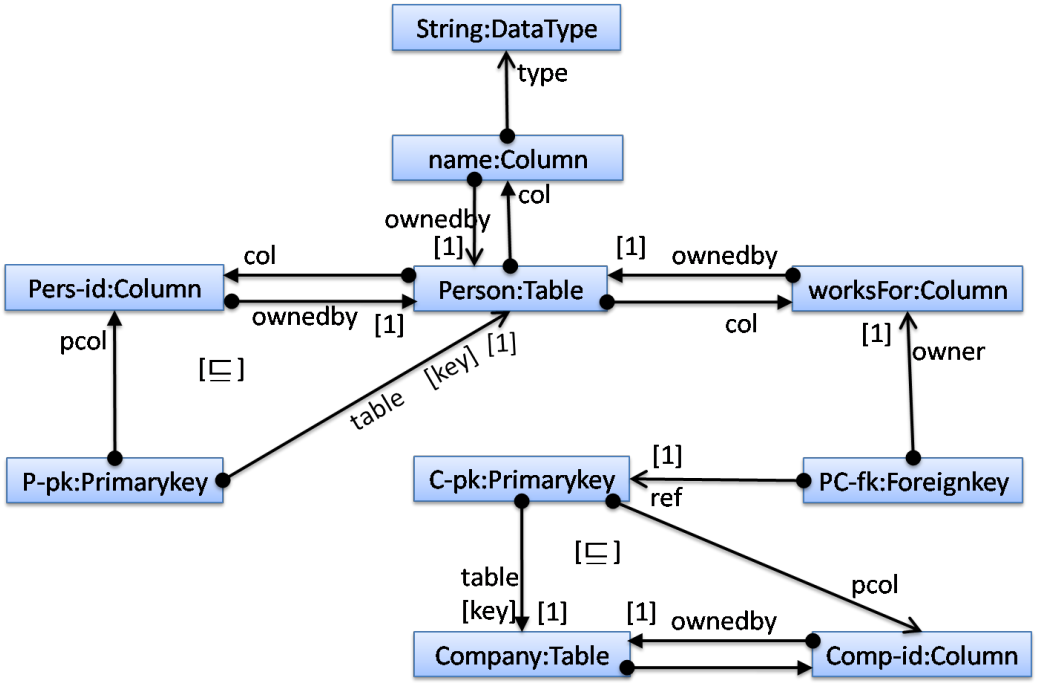
Rule3 EReference to ForeignKey



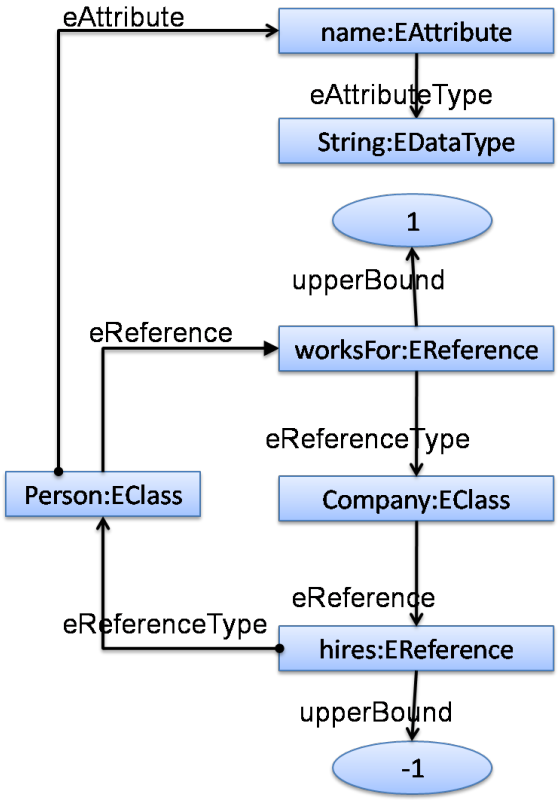


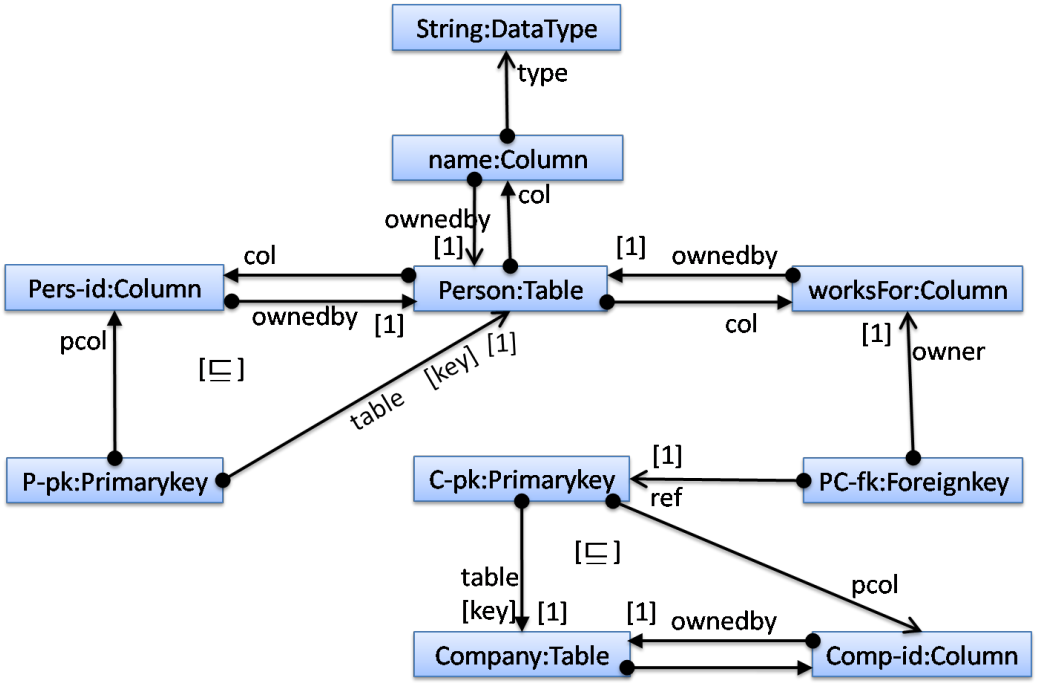
The Output RDB Model





The Output RDB Model

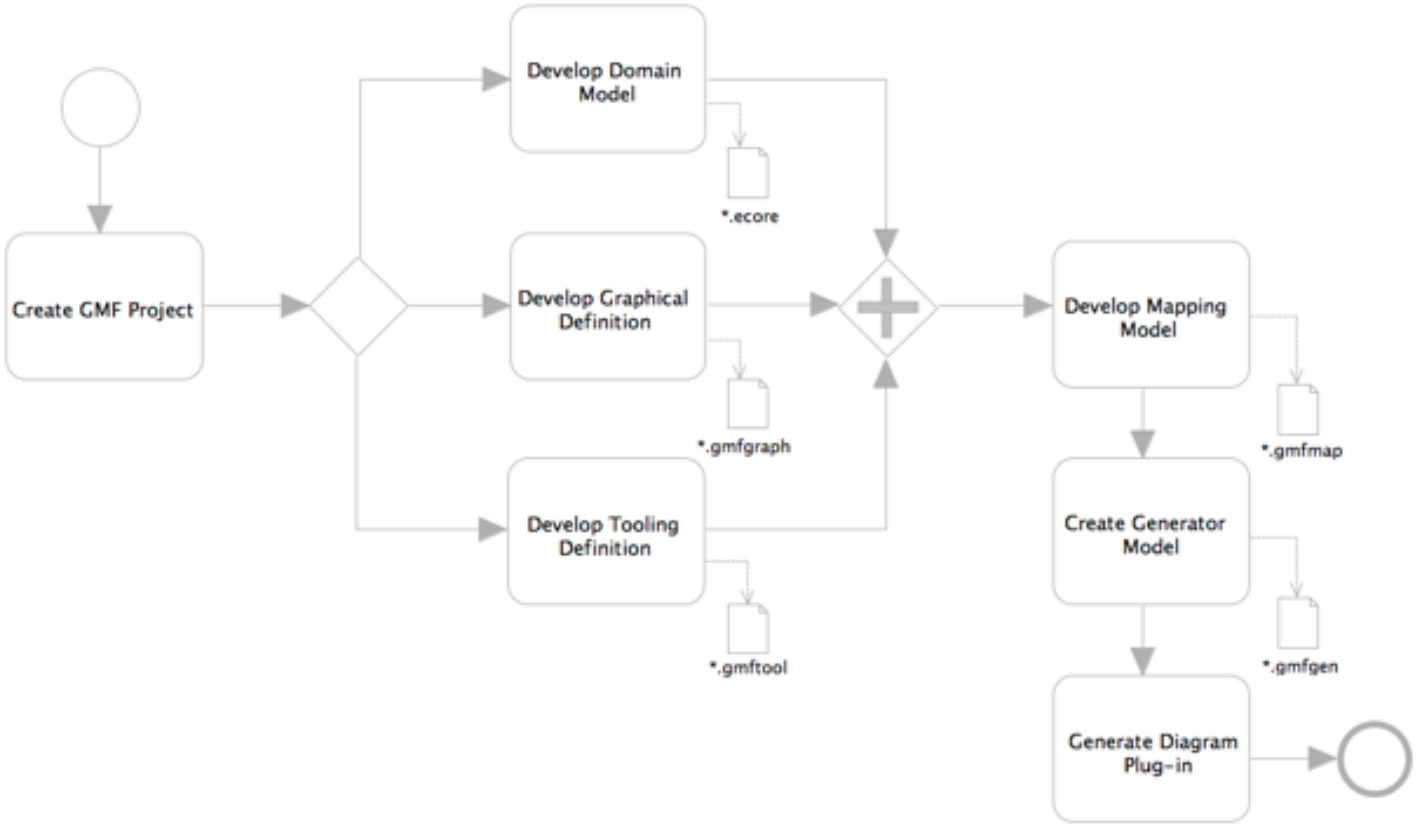




Tools

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

Generation of GMF-based Editors



Summary

- The MDD is a promising approach for software development processes
- The MDD requires tools for automatic model management
- Automatic model management requires formal modeling and model transformation languages
- Using DPL as a generic framework for formalization of modeling and model transformation languages