A Formal Approach to Patterns in MDE

Yngve Lamo

Faculty of Engineering, Bergen University College, Norway

21 September 2010
VFfP 2010, Madrid, Spain

Outline

Personal Background

- Master in mathematics NTNU
- Ph.D. in theoretical Computer Science UoB – Algebraic specifications of software, used mathematics such as: universal algebra, logic, category theory
- Get motivated to apply category theory in software engineering
  - In 2000 Uwe Wolter introduced me to the Generalized Sketches framework, created in Latvia in early 90's, by Zinovy Diskin et al
  - In 2003 we started to use GS to formalize diagrammatic specification languages (such as UML, ER)

Why Category Theory?

- Category theory focuses on how objects are related to each other
  - black box view of systems
  - objects (systems) are described by their interface to other objects (systems)
  - quantification over objects (systems)
- Set theory focuses on how objects are created
  - white box view of systems
  - objects described by their internal structure
  - quantification over elements

Model-Driven Engineering (MDE)

- Software models as abstract representations of software systems
  - reasoning at a high level of abstraction
- Model-Driven Engineering (MDE)
  - models as primary artefacts of the software development process
  - generation of systems by means of model transformations

MDE Standards: State-of-the-art

- Modelling languages
  - Unified Modeling Language (UML)
  - Eclipse Modeling Framework (EMF)

  Usually graph-based languages
- Constraint languages
  - Object Constraint Language (OCL)

  Usually text-based languages

Some Advantages of Diagrammatic Modelling

Property	**Advantage**	**Achieved by**
Documentation and communication	Facilitating intuitiveness	*Visual models*
Abstraction	Independence of the implementation platform	*Abstract models* and *model transformations*
Validation and verification	Revealing errors and flaws before the system is implemented	*Formal models* and *model checking*

Some facts about UML

- UML is defacto industry standard for modelling
- UML can only express constraints on binary relations, basically UML only specifies cardinality and typing constraints
- 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 modelling languages

- Nice semantics, several fundamental Software Engineering problems solved by use of formal methods
- Set based semantics (logic, algebras, etc.)
- No concept of meta modelling (index semantics)
- 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

Project History

- 2003– Uwe Wolter and Yngve Lamo started to work with Generalized Sketches
- 2004-2006 Master student Ørjan Hatland implement protoype tool for GS (.NET framework)
- 2006-2010 PhD student Adrian Rutle, MDE, (meta)modeling, model transformation ⇒ DPF
- 2006-2008 Master student Stian Skjerveggen started to implement a graphical editor in Eclipse (GMF) framework
- 2008– PhD student Alessandro Rossini, Version Control for MDE
- 2009– Master Students Øyvind Bech, Dag Viggo Lokøen, tool support for DPF
- 2010– PhD student Florian Mantz, meta model evolution

Diagram Predicate Framework (DPF)

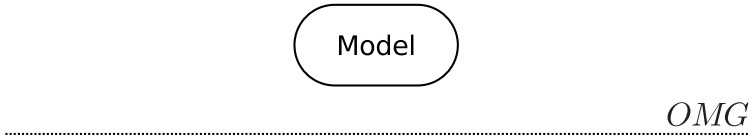
- A generalisation of categorical sketches
- Adaption of generalised sketches to MDE
- Graph based specification framework
  - models: graphs decorated with constraints
  - constraints formalised as categorical diagrams

DPF offers

- Diagrammatic formalisation of metamodelling
- Constraint-aware model transformation
  - constraints affect which structure to create in the target model
  - control which constraints to add to the created structure
- Today patterns in DPF:
  - pattern definition as diagrammatic specifications
  - pattern matching and application by model transformation
- Formalisation of version control of models

(Meta)modelling in OMG and DPF

Modelling approach



Sample model

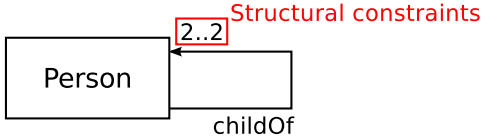
- Model: abstract representation of a software system

(Meta)modelling in OMG and DPF

Modelling approach

OMG

Sample model

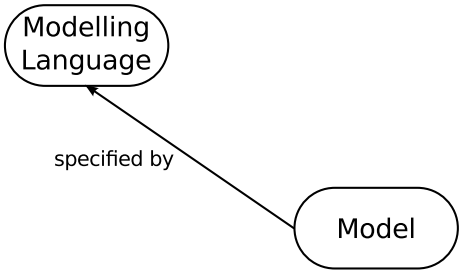


OMG

- Req 1: "a person must be child of exactly two persons"

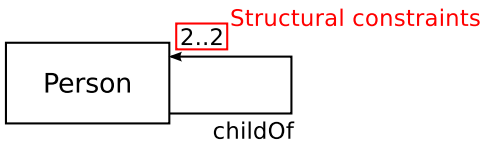
(Meta)modelling in OMG and DPF

Modelling approach



OMG

Sample model

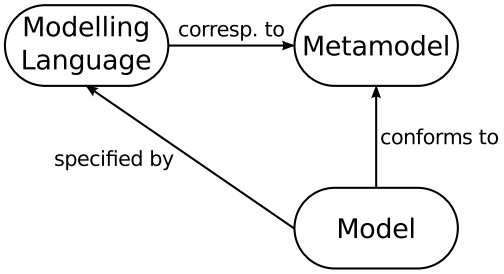


OMG

- Model: specified by a modelling language

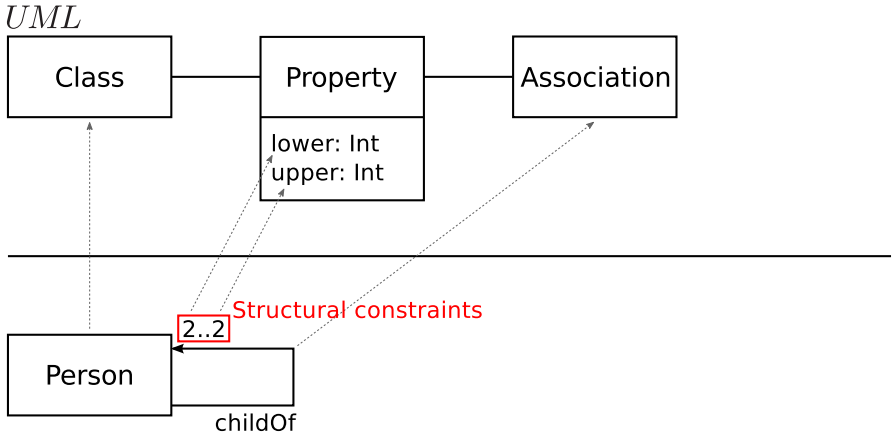
(Meta)modelling in OMG and DPF

Modelling approach



OMG

Sample model

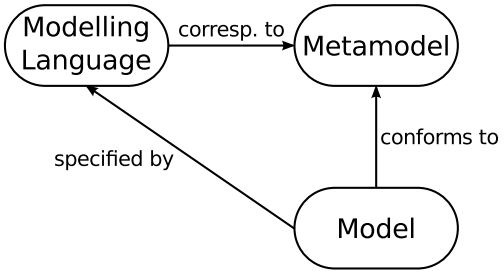


OMG

- Modelling language: corresponding metamodel + semi-formal

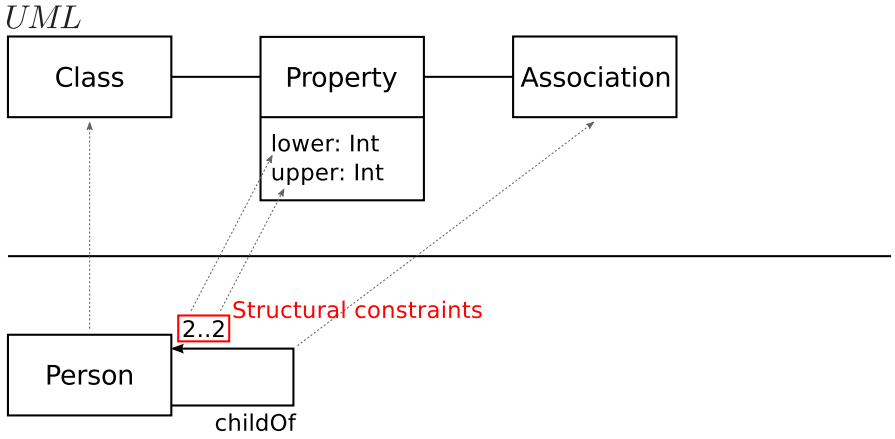
(Meta)modelling in OMG and DPF

Modelling approach



OMG

Sample model

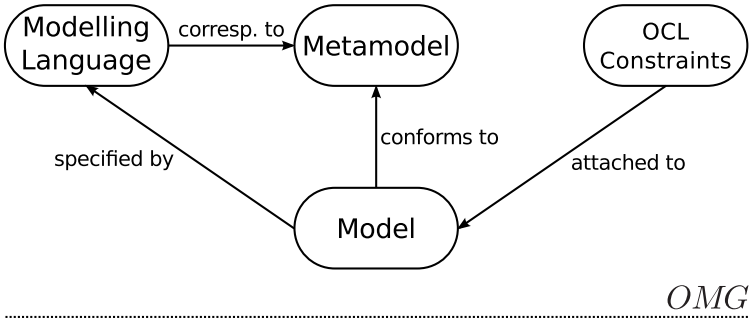


OMG

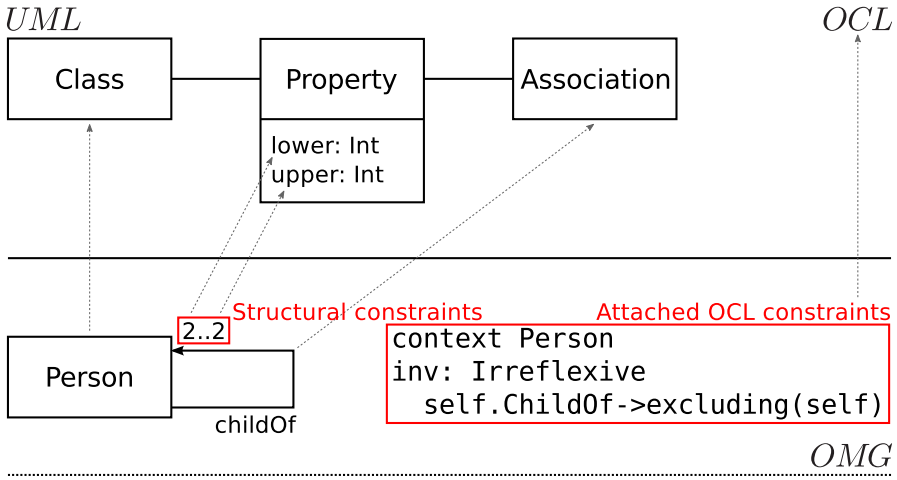
- Req 2: "no person is his/her own child"

(Meta)modelling in OMG and DPF

Modelling approach



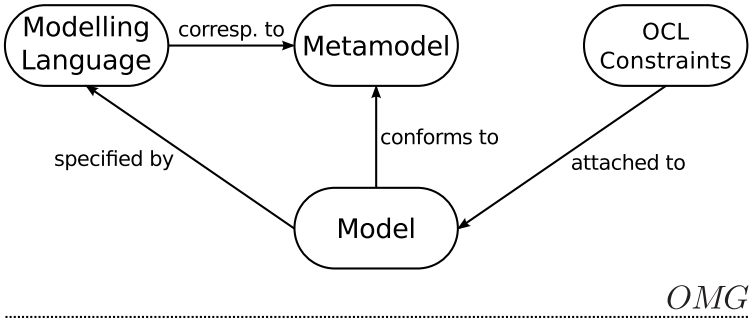
Sample model



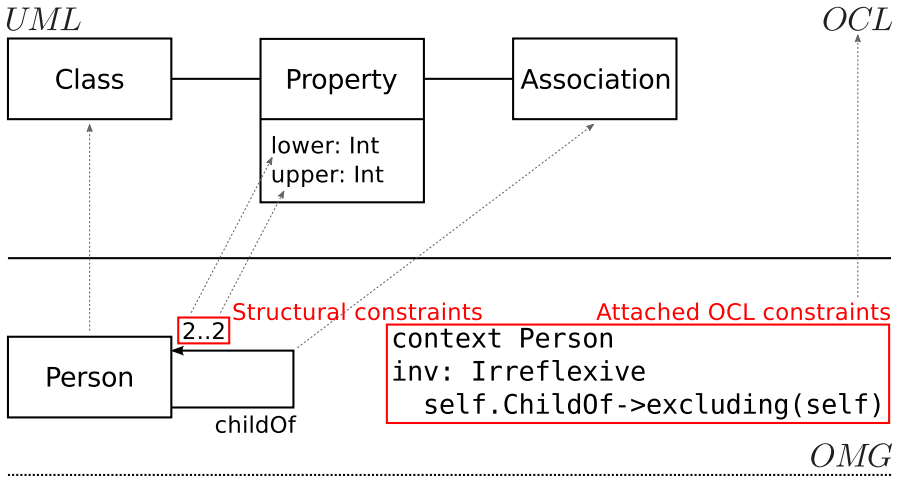
- Attached constraint: specified by means of text-based OCL

(Meta)modelling in OMG and DPF

Modelling approach



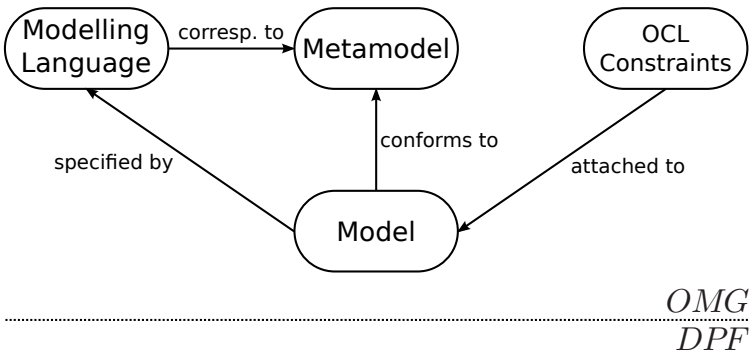
Sample model



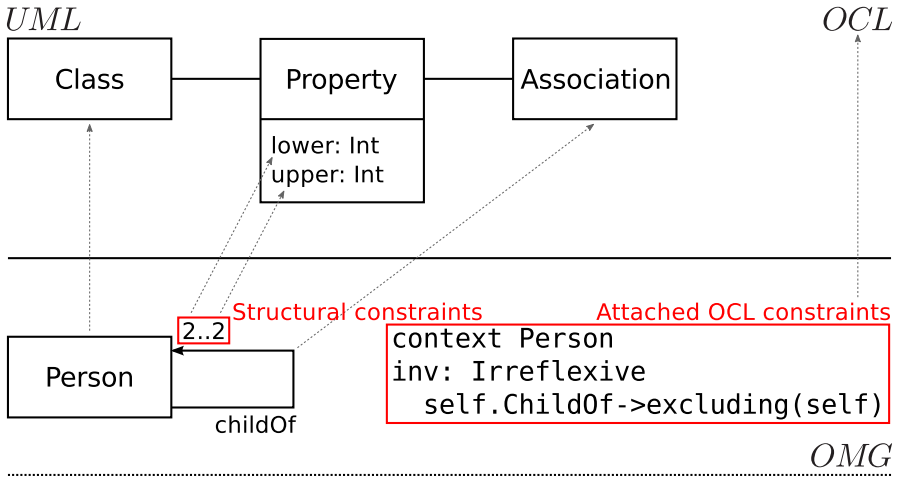
- Diagram Predicate Framework (DPF)

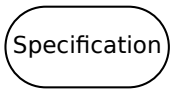
(Meta)modelling in OMG and DPF

Modelling approach



Sample model

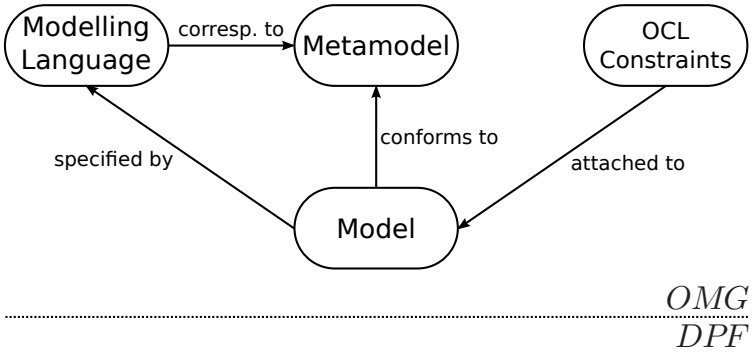




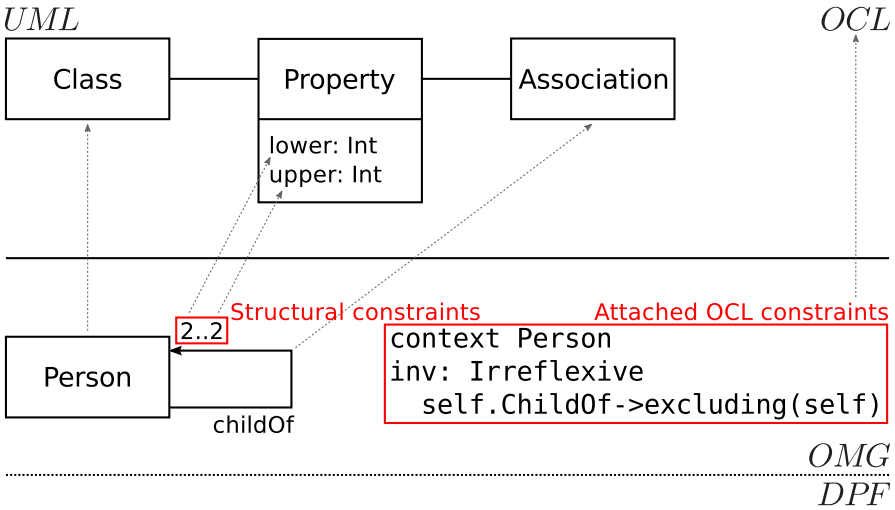
- Specification: graph + set of constraints

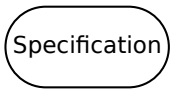
(Meta)modelling in OMG and DPF

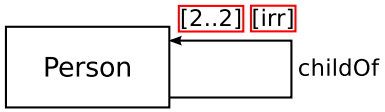
Modelling approach



Sample model



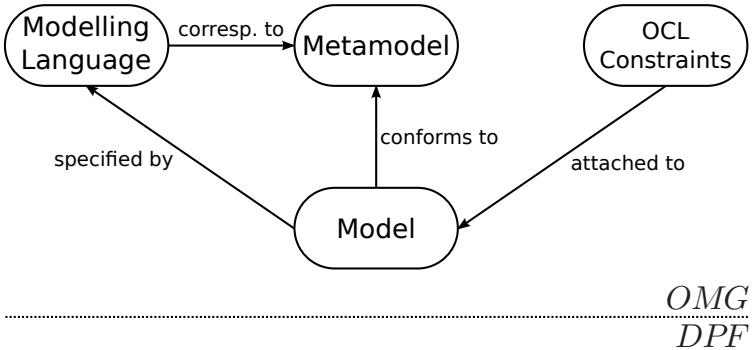




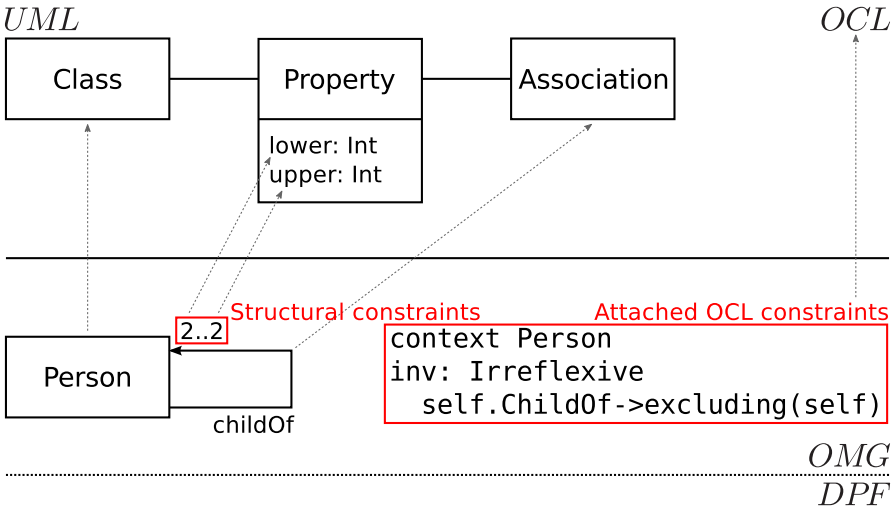
- Specification: graph + set of constraints

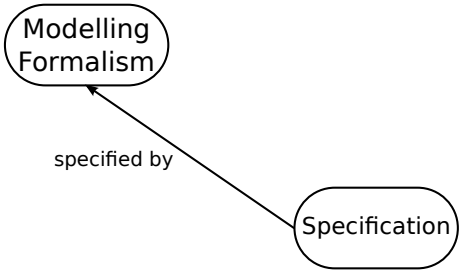
(Meta)modelling in OMG and DPF

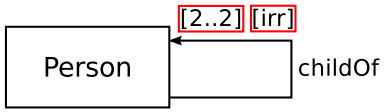
Modelling approach



Sample model



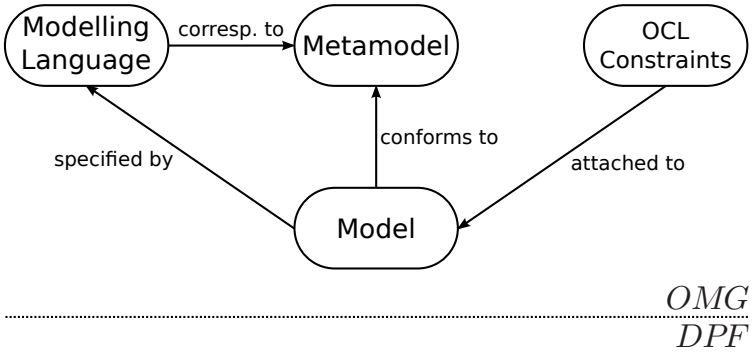


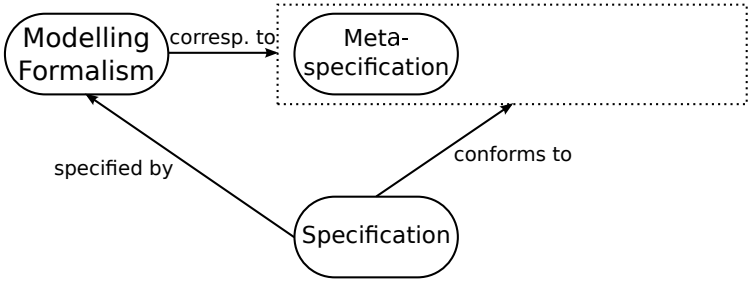


- Specification: specified by a modelling formalism

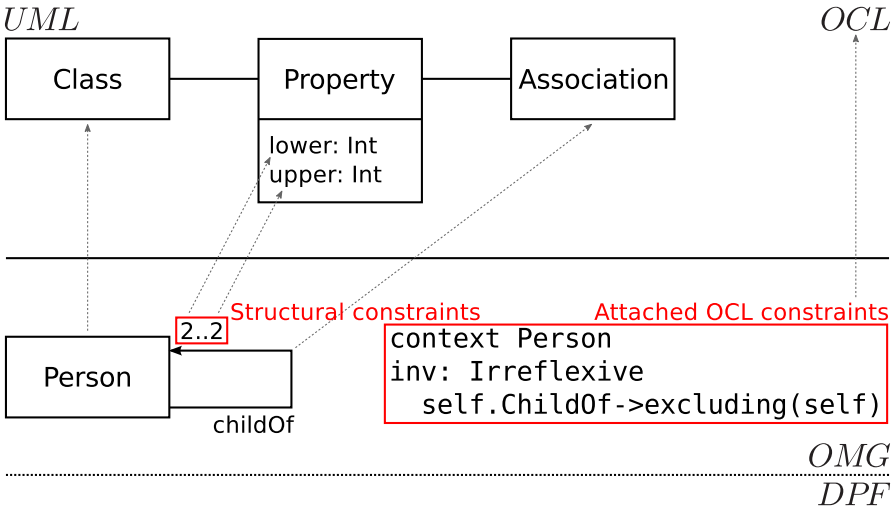
(Meta)modelling in OMG and DPF

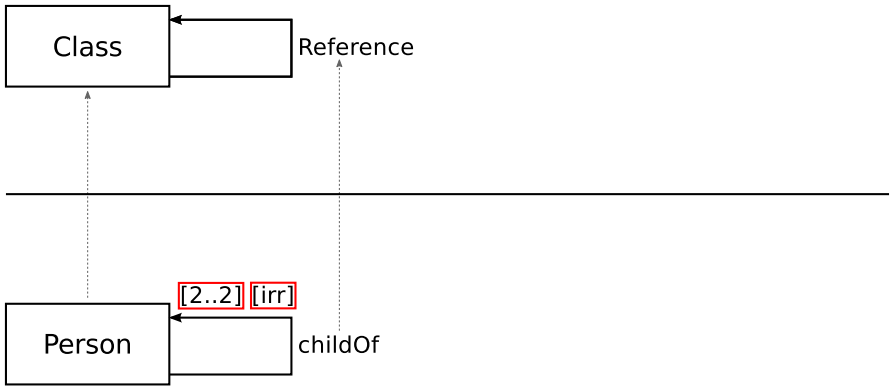
Modelling approach





Sample model

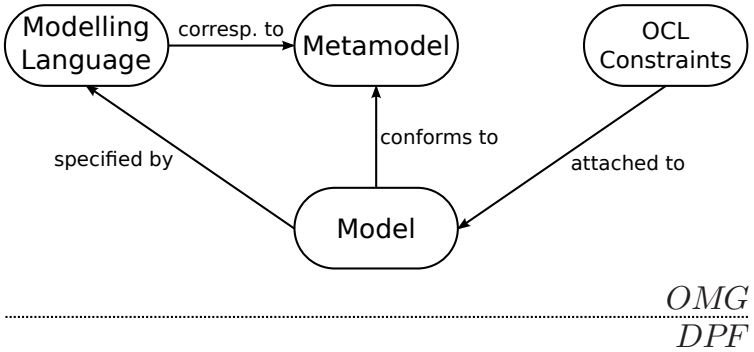


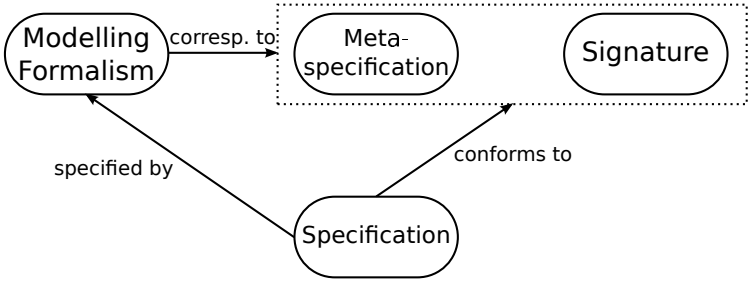


- Modelling formalism: meta-specification + ...

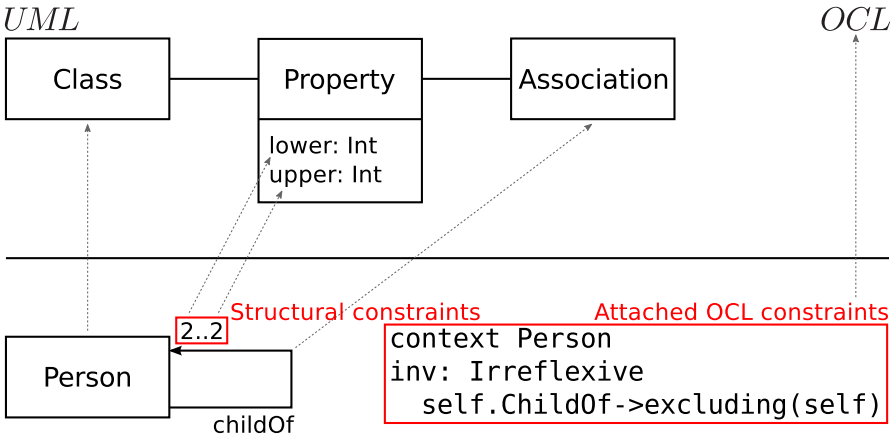
(Meta)modelling in OMG and DPF

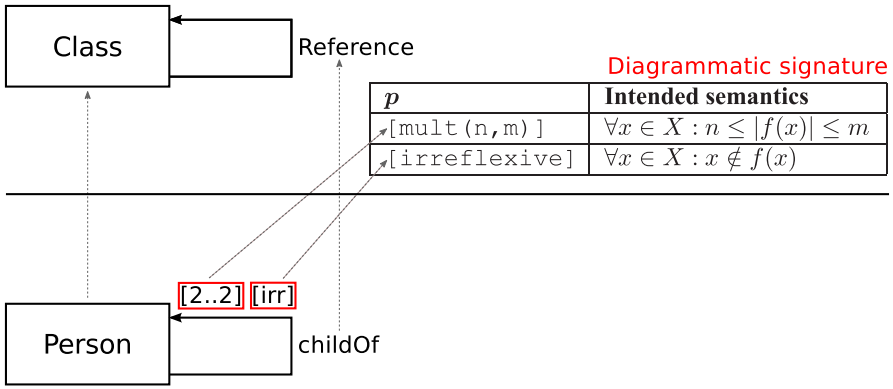
Modelling approach





Sample model

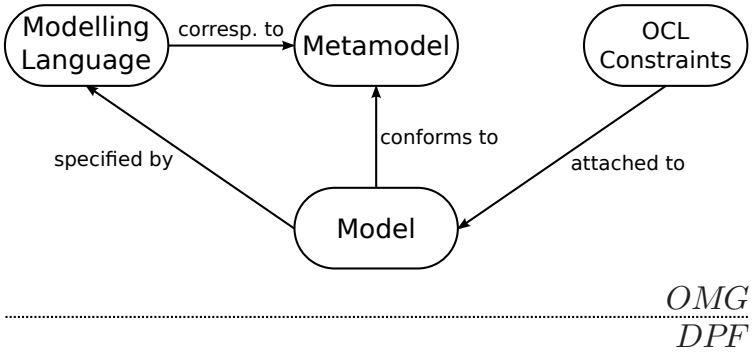


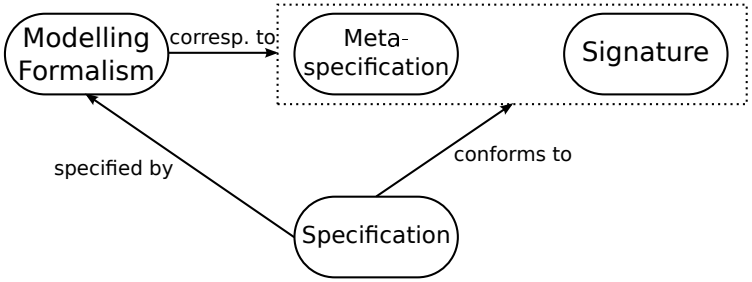


- Modelling formalism: meta-specification + signature

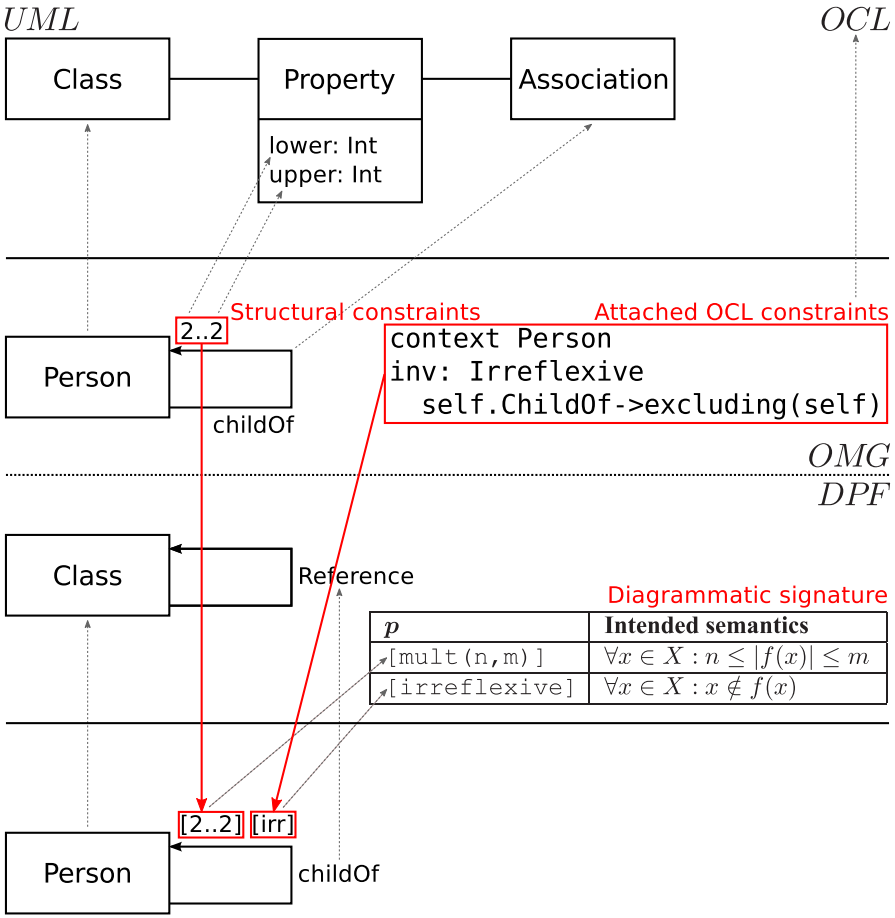
(Meta)modelling in OMG and DPF

Modelling approach



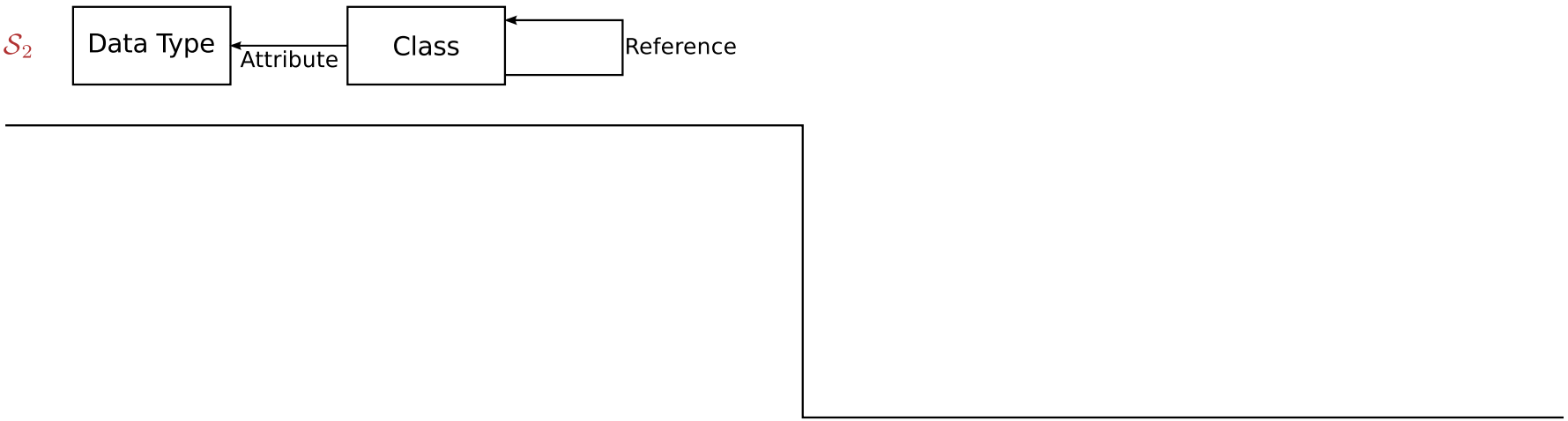


Sample model



- Integration of constraints

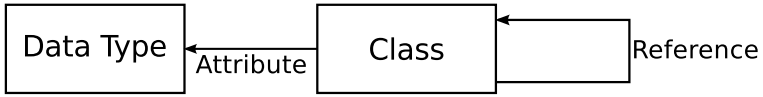
Sample Object-Oriented Modelling Hierarchy



- Modelling formalism: meta-specification + ...

Sample Object-Oriented Modelling Hierarchy

$\mathcal{S}_2$

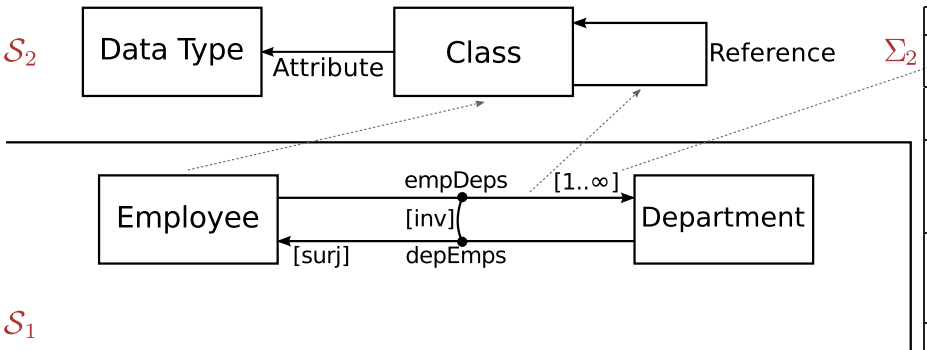


Σ_2

p	$\alpha^{\Sigma_2}(p)$	Proposed vis.	Intended semantics
`[mult(n,m)]`	$1 \xrightarrow{f} 2$	X $\xrightarrow[\text{[n..m]}]{f}$ Y	$\forall x \in X : n \leq \lvert f(x) \rvert \leq m$
`[surjective]`	$1 \xrightarrow{f} 2$	X $\xrightarrow[\text{[surj]}]{f}$ Y	$f(X) = Y$
`[inverse]`	$1 \underset{g}{\overset{f}{\rightleftarrows}} 2$	X $\underset{g}{\overset{f}{\underset{\text{[inv]}}{\rightleftarrows}}}$ Y	$\forall x \in X , \forall y \in Y : y \in f(x)$ iff $x \in g(y)$
`[image-inclusion]`	$1 \underset{g}{\overset{f}{\rightrightarrows}} 2$	X $\underset{g}{\overset{f}{\underset{[\subseteq]}{\rightrightarrows}}}$ Y	$\forall x \in X : f(x) \subseteq g(x)$
`[composition]`	$\begin{tikzcd} 1 \arrow[r, "f"] \arrow[dr, "h"'] & 2 \arrow[d, "g"] \\ & 3 \end{tikzcd}$	$\begin{tikzcd} X \arrow[r, "f"] \arrow[dr, "f;g"'] & Y \arrow[d, "g"] \\ & Z \end{tikzcd}$	$\forall x \in X : f;g(x) = \{g(y) \mid y \in f(x)\}$

- Modelling formalism: meta-specification + signature

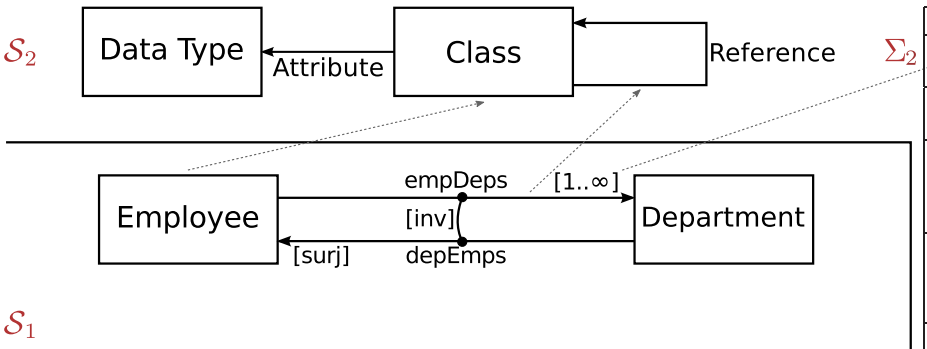
Sample Object-Oriented Modelling Hierarchy



p	$\alpha^{\Sigma_2}(p)$	Proposed vis.	Intended semantics
`[mult(n,m)]`	$1 \xrightarrow{f} 2$	$X \xrightarrow[\text{[n..m]}]{f} Y$	$\forall x \in X : n \leq \lvert f(x) \rvert \leq m$
`[surjective]`	$1 \xrightarrow{f} 2$	$X \xrightarrow[\text{[surj]}]{f} Y$	$f(X) = Y$
`[inverse]`	$1 \underset{g}{\overset{f}{\rightleftarrows}} 2$	$X \underset{g}{\overset{f}{\underset{\text{[inv]}}{\rightleftarrows}}} Y$	$\forall x \in X , \forall y \in Y : y \in f(x)$ iff $x \in g(y)$
`[image-inclusion]`	$1 \underset{g}{\overset{f}{\rightleftarrows}} 2$	$X \underset{g}{\overset{f}{\underset{\text{[}\subseteq\text{]}}{\rightleftarrows}}} Y$	$\forall x \in X : f(x) \subseteq g(x)$
`[composition]`	$\begin{tikzcd} 1 \arrow[r, "f"] \arrow[dr, "h"'] & 2 \arrow[d, "g"] \\ & 3 \end{tikzcd}$	$\begin{tikzcd} X \arrow[r, "f"] \arrow[dr, "f;g"'] & Y \arrow[d, "g"] \\ & Z \end{tikzcd}$	$\forall x \in X : f;g(x) = \{g(y) \mid y \in f(x)\}$

- Req 1: "an employee must work for at least one department"

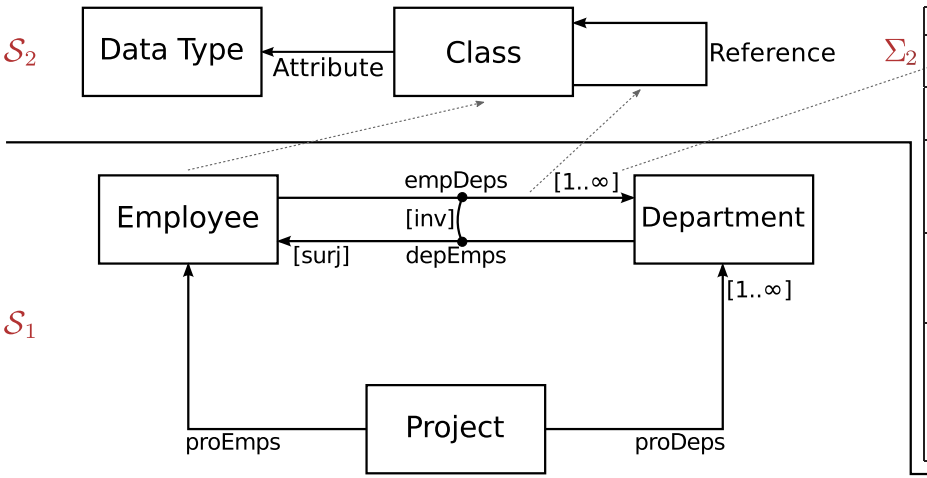
Sample Object-Oriented Modelling Hierarchy



p	$\alpha^{\Sigma_2}(p)$	Proposed vis.	Intended semantics
[mult(n,m)]	$1 \xrightarrow{f} 2$	X $\xrightarrow[\text{[n..m]}]{f}$ Y	$\forall x \in X : n \leq \lvert f(x) \rvert \leq m$
[surjective]	$1 \xrightarrow{f} 2$	X $\xrightarrow[\text{[surj]}]{f}$ Y	$f(X) = Y$
[inverse]	$1 \underset{g}{\overset{f}{\rightleftarrows}} 2$	X $\underset{g}{\overset{f}{\underset{\text{[inv]}}{\rightleftarrows}}}$ Y	$\forall x \in X, \forall y \in Y : y \in f(x)$ iff $x \in g(y)$
[image-inclusion]	$1 \underset{g}{\overset{f}{\rightleftarrows}} 2$	X $\underset{g}{\overset{f}{\underset{\text{[}\subseteq\text{]}}{\rightleftarrows}}}$ Y	$\forall x \in X : f(x) \subseteq g(x)$
[composition]	$\begin{tikzcd} 1 \arrow[r, "f"] \arrow[dr, "h"'] & 2 \arrow[d, "g"] \\ & 3 \end{tikzcd}$	$\begin{tikzcd} X \arrow[r, "f"] \arrow[dr, "f;g"'] & Y \arrow[d, "g"] \\ & Z \end{tikzcd}$	$\forall x \in X : f;g(x) = \{g(y) \mid y \in f(x)\}$

- Req 2: "a department may have none or many employees"

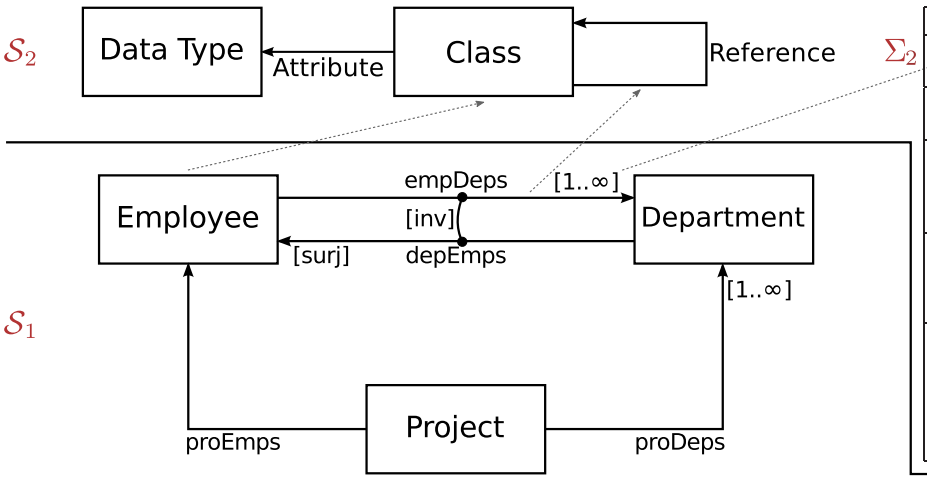
Sample Object-Oriented Modelling Hierarchy



p	$\alpha^{\Sigma_2}(p)$	Proposed vis.	Intended semantics
`[mult(n,m)]`	$1 \xrightarrow{f} 2$	X $\xrightarrow[\text{[n..m]}]{f}$ Y	$\forall x \in X : n \leq \lvert f(x) \rvert \leq m$
`[surjective]`	$1 \xrightarrow{f} 2$	X $\xrightarrow[\text{[surj]}]{f}$ Y	$f(X) = Y$
`[inverse]`	$1 \underset{g}{\overset{f}{\rightleftarrows}} 2$	X $\underset{g}{\overset{f}{\underset{\text{[inv]}}{\rightleftarrows}}}$ Y	$\forall x \in X, \forall y \in Y : y \in f(x)$ iff $x \in g(y)$
`[image-inclusion]`	$1 \underset{g}{\overset{f}{\rightleftarrows}} 2$	X $\underset{g}{\overset{f}{\underset{\text{[}\subseteq\text{]}}{\rightleftarrows}}}$ Y	$\forall x \in X : f(x) \subseteq g(x)$
`[composition]`	$\begin{tikzcd} 1 \arrow[r, "f"] \arrow[dr, "h"'] & 2 \arrow[d, "g"] \\ & 3 \end{tikzcd}$	$\begin{tikzcd} X \arrow[r, "f"] \arrow[dr, "f;g"'] & Y \arrow[d, "g"] \\ & Z \end{tikzcd}$	$\forall x \in X : f;g(x) = \{g(y) \mid y \in f(x)\}$

- Req 3: "a project may involve none or many employees"

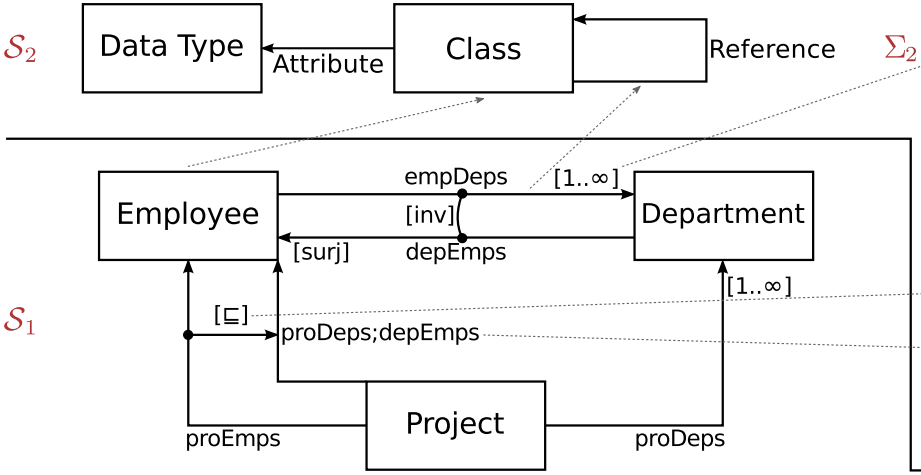
Sample Object-Oriented Modelling Hierarchy



p	$\alpha^{\Sigma_2}(p)$	Proposed vis.	Intended semantics
[mult(n,m)]	$1 \xrightarrow{f} 2$	X $\xrightarrow[\text{[n..m]}]{f}$ Y	$\forall x \in X : n \leq \lvert f(x) \rvert \leq m$
[surjective]	$1 \xrightarrow{f} 2$	X $\xrightarrow[\text{[surj]}]{f}$ Y	$f(X) = Y$
[inverse]	$1 \underset{g}{\overset{f}{\rightleftarrows}} 2$	X $\underset{g}{\overset{f}{\underset{\text{[inv]}}{\rightleftarrows}}}$ Y	$\forall x \in X, \forall y \in Y : y \in f(x)$ iff $x \in g(y)$
[image-inclusion]	$1 \underset{g}{\overset{f}{\rightleftarrows}} 2$	X $\underset{g}{\overset{f}{\underset{\text{[}\subseteq\text{]}}{\rightleftarrows}}}$ Y	$\forall x \in X : f(x) \subseteq g(x)$
[composition]	$\begin{tikzcd} 1 \arrow[r, "f"] \arrow[dr, "h"'] & 2 \arrow[d, "g"] \\ & 3 \end{tikzcd}$	$\begin{tikzcd} X \arrow[r, "f"] \arrow[dr, "f;g"'] & Y \arrow[d, "g"] \\ & Z \end{tikzcd}$	$\forall x \in X : f;g(x) = \{g(y) \mid y \in f(x)\}$

- Req 4: "a project must be controlled by at least one department"

Sample Object-Oriented Modelling Hierarchy

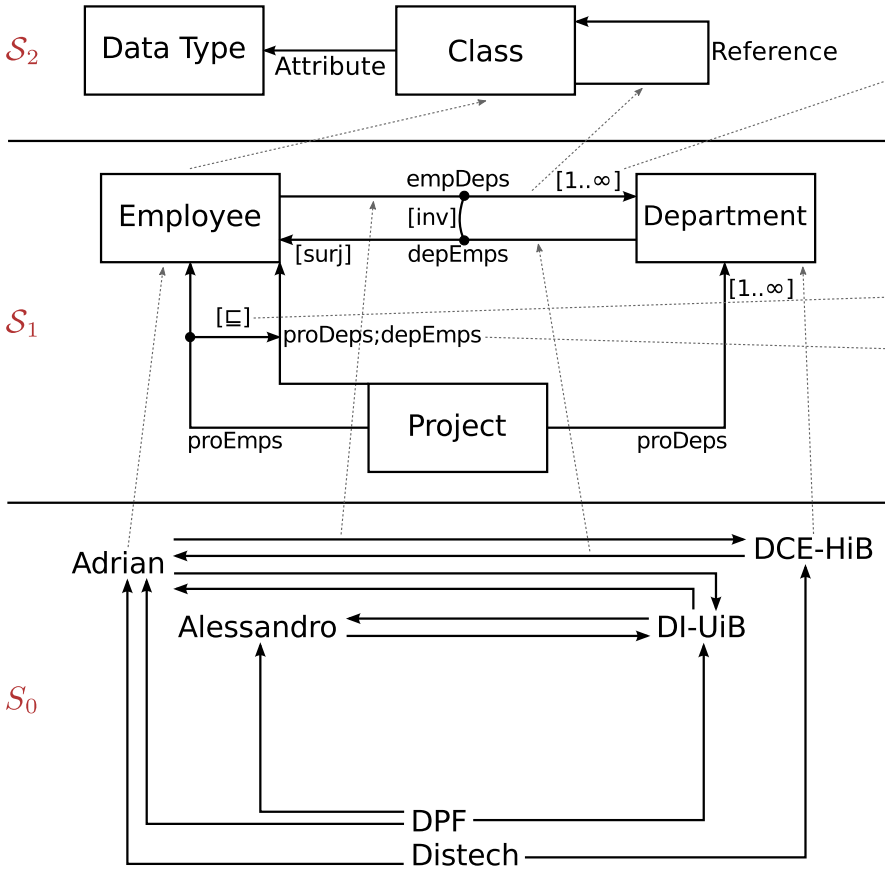


Σ_2

p	$\alpha^{\Sigma_2}(p)$	Proposed vis.	Intended semantics
[mult(n,m)]	$1 \xrightarrow{f} 2$	$X \xrightarrow[{[n..m]}]{f} Y$	$\forall x \in X : n \leq \lvert f(x) \rvert \leq m$
[surjective]	$1 \xrightarrow{f} 2$	$X \xrightarrow[{[surj]}]{f} Y$	$f(X) = Y$
[inverse]	$1 \underset{g}{\overset{f}{\rightleftarrows}} 2$	$X \overset{f}{\underset{g}{\overset{[inv]}{\rightleftarrows}}} Y$	$\forall x \in X, \forall y \in Y : y \in f(x)$ iff $x \in g(y)$
[image-inclusion]	$1 \underset{g}{\overset{f}{\rightrightarrows}} 2$	$X \overset{f}{\underset{g}{\overset{[\sqsubseteq]}{\rightrightarrows}}} Y$	$\forall x \in X : f(x) \subseteq g(x)$
[composition]	$\begin{tikzcd} 1 \arrow[r, "f"] \arrow[dr, "h"'] & 2 \arrow[d, "g"] \\ & 3 \end{tikzcd}$	$\begin{tikzcd} X \arrow[r, "f"] \arrow[dr, "f;g"'] & Y \arrow[d, "g"] \\ & Z \end{tikzcd}$	$\forall x \in X : f;g(x) = \{g(y) \mid y \in f(x)\}$

- Req 5: "an employee involved in a project must work in the controlling department"

Sample Object-Oriented Modelling Hierarchy

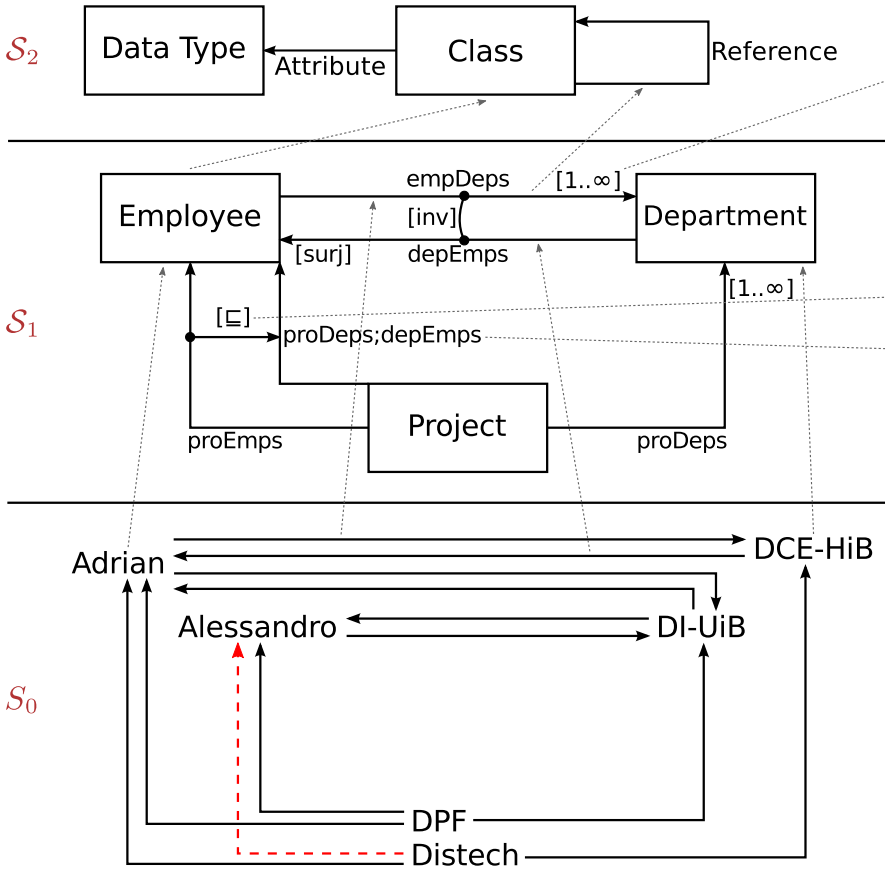


Σ_2

p	$\alpha^{\Sigma_2}(p)$	Proposed vis.	Intended semantics
[mult(n,m)]	$1 \xrightarrow{f} 2$	X $\xrightarrow[\text{[n..m]}]{f}$ Y	$\forall x \in X : n \leq \lvert f(x) \rvert \leq m$
[surjective]	$1 \xrightarrow{f} 2$	X $\xrightarrow[\text{[surj]}]{f}$ Y	$f(X) = Y$
[inverse]	$1 \underset{g}{\overset{f}{\rightleftarrows}} 2$	X $\underset{g}{\overset{f}{\underset{\text{[inv]}}{\rightleftarrows}}}$ Y	$\forall x \in X, \forall y \in Y : y \in f(x)$ iff $x \in g(y)$
[image-inclusion]	$1 \underset{g}{\overset{f}{\rightrightarrows}} 2$	X $\underset{g}{\overset{f}{\underset{\text{[⊑]}}{\rightrightarrows}}}$ Y	$\forall x \in X : f(x) \subseteq g(x)$
[composition]	$\begin{tikzcd} 1 \arrow[r, "f"] \arrow[dr, "h"'] & 2 \arrow[d, "g"] \\ & 3 \end{tikzcd}$	$\begin{tikzcd} X \arrow[r, "f"] \arrow[dr, "f;g"'] & Y \arrow[d, "g"] \\ & Z \end{tikzcd}$	$\forall x \in X : f;g(x) = \{g(y) \mid y \in f(x)\}$

- Instance

Sample Object-Oriented Modelling Hierarchy



Σ_2

p	$\alpha^{\Sigma_2}(p)$	Proposed vis.	Intended semantics
[mult(n,m)]	$1 \xrightarrow{f} 2$	X $\xrightarrow[\text{[n..m]}]{f}$ Y	$\forall x \in X : n \leq \lvert f(x) \rvert \leq m$
[surjective]	$1 \xrightarrow{f} 2$	X $\xrightarrow[\text{[surj]}]{f}$ Y	$f(X) = Y$
[inverse]	$1 \underset{g}{\overset{f}{\rightleftarrows}} 2$	X $\underset{g}{\overset{f}{\underset{\text{[inv]}}{\rightleftarrows}}}$ Y	$\forall x \in X, \forall y \in Y : y \in f(x)$ iff $x \in g(y)$
[image-inclusion]	$1 \underset{g}{\overset{f}{\rightrightarrows}} 2$	X $\underset{g}{\overset{f}{\underset{\text{[⊑]}}{\rightrightarrows}}}$ Y	$\forall x \in X : f(x) \subseteq g(x)$
[composition]	$\begin{tikzcd} 1 \arrow[r, "f"] \arrow[dr, "h"'] & 2 \arrow[d, "g"] \\ & 3 \end{tikzcd}$	$\begin{tikzcd} X \arrow[r, "f"] \arrow[dr, "f;g"'] & Y \arrow[d, "g"] \\ & Z \end{tikzcd}$	$\forall x \in X : f;g(x) = \{g(y) \mid y \in f(x)\}$

- Invalid instance

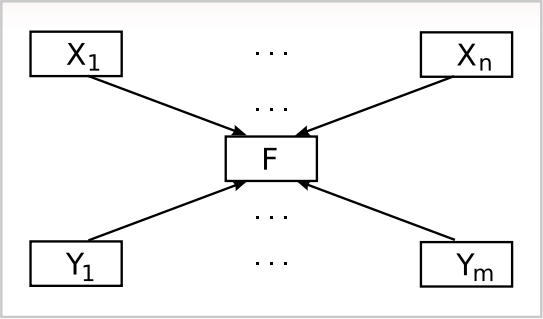
Outline

Patterns in Model Driven Engineering

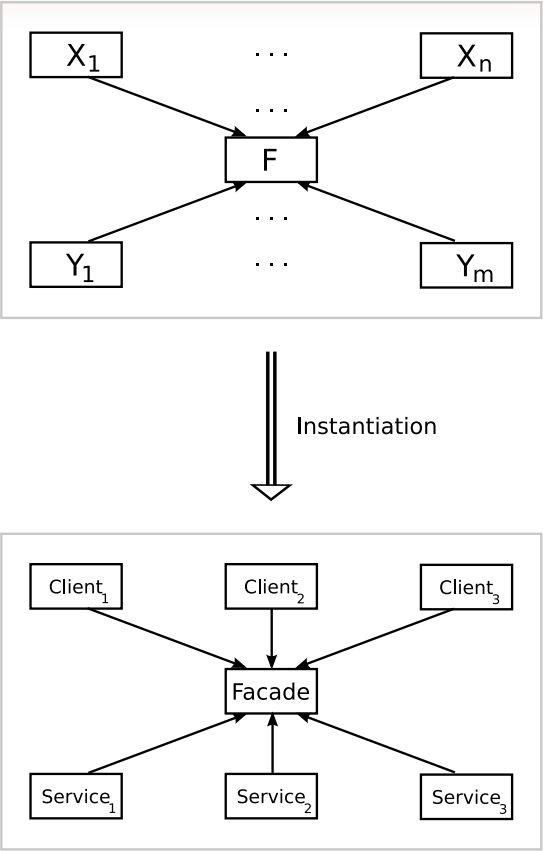
- Patterns are usually explained in a semi formal or informal language
- To be used in MDE patterns should be expressed formally facilitating
  - model transformations and
  - automatic software development steps
- To enhance usability one would like to have a visual language for pattern definition
- Hence natural to start with a graph based approach
- The Diagram Predicate Framework DPF is a graph based framework where one could specify constraints over arbitrary graphs.

Design Patterns

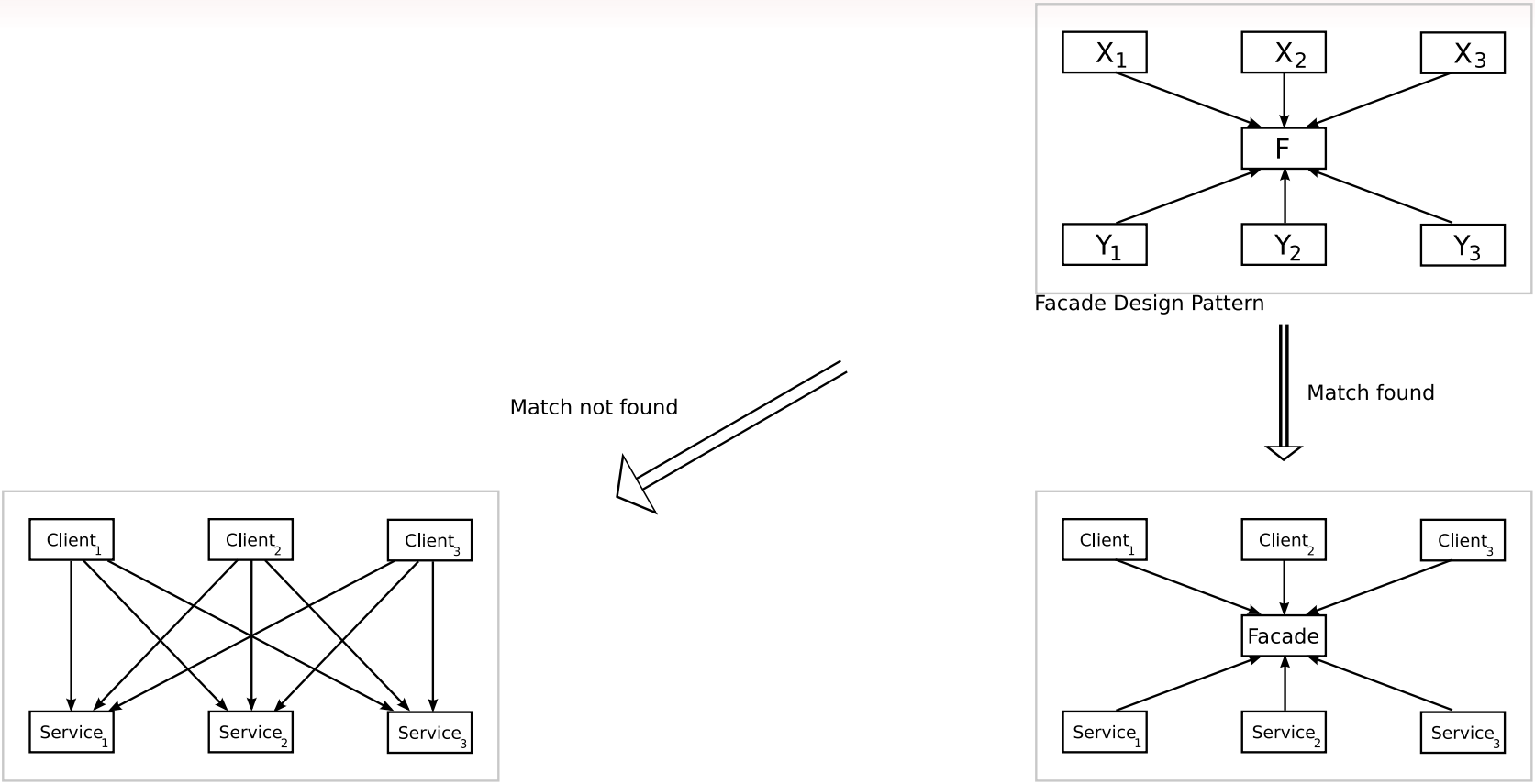
- A solution strategy for a common problem (e.g. facade, decorator, singleton, etc.)
  - Reusable
  - Abstract
  - Conceptually clear
    - the pattern should be understand by referring to it's name
- Often describes a solution for a part of a bigger system.
- Usage scenarios:
  1. guideline for design (specification)
  2. check whether a design conforms to a pattern
  3. if not, change the design (pattern enforcement by transformation/refactoring)



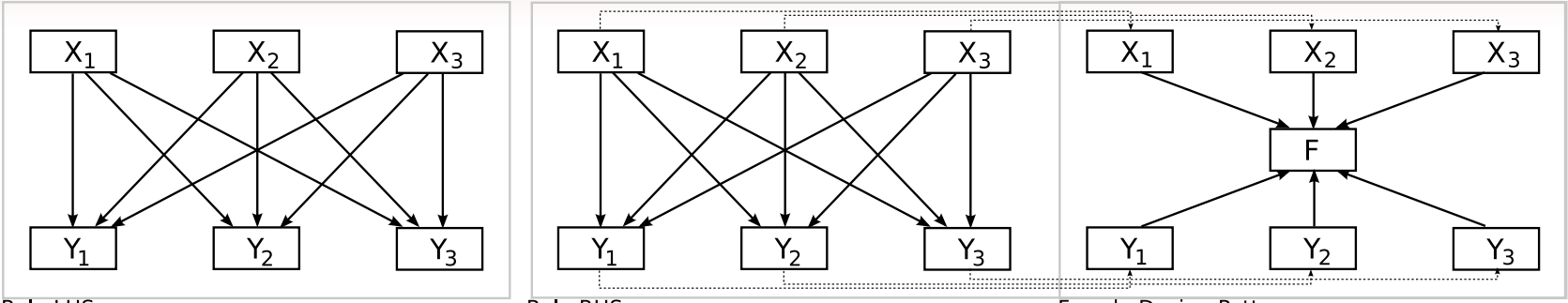
- Facade Design Pattern: Interface for subsystems



- One aspect is to design a specifications that satisfies the pattern

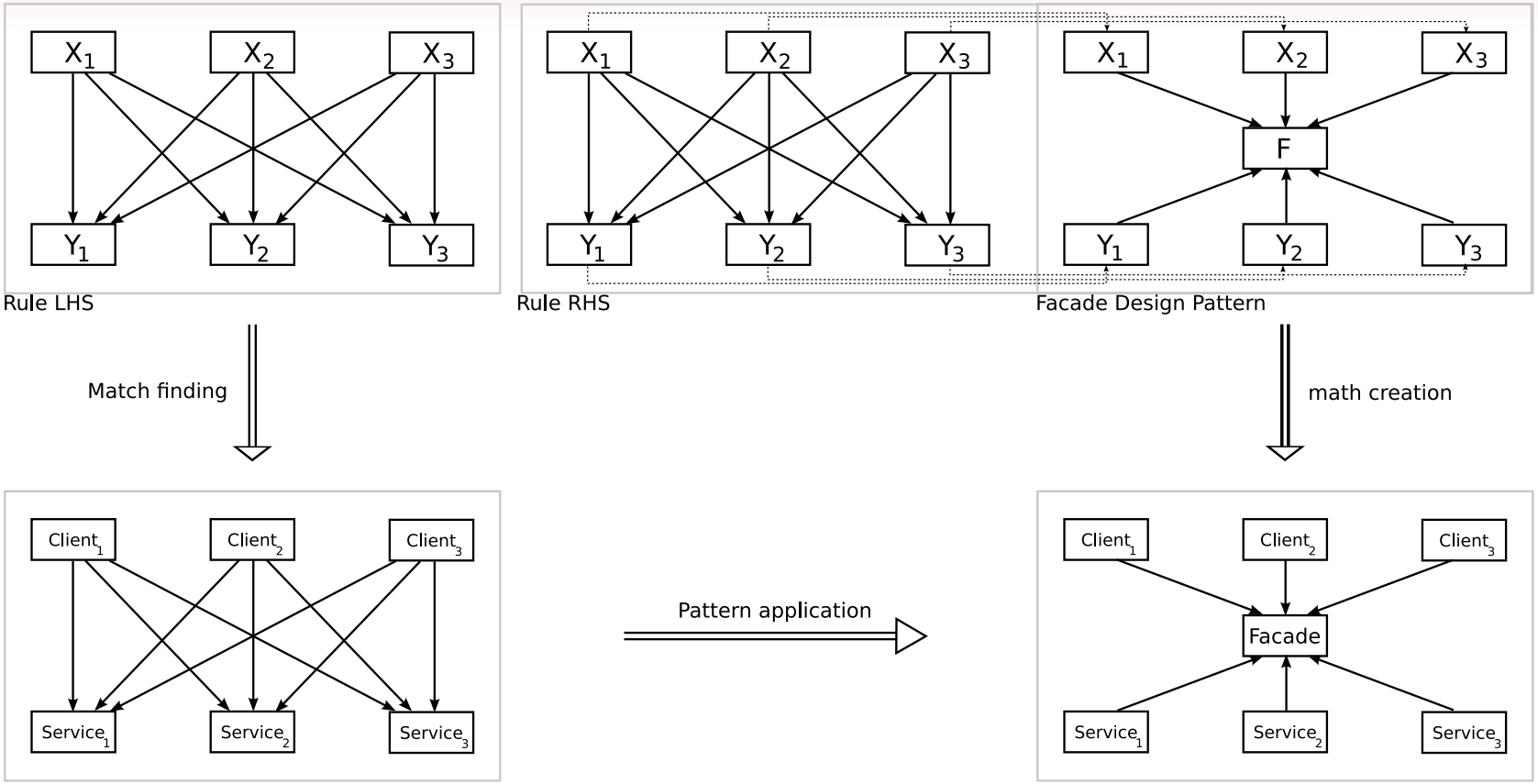


- ... pattern applied to check existing specifications ...

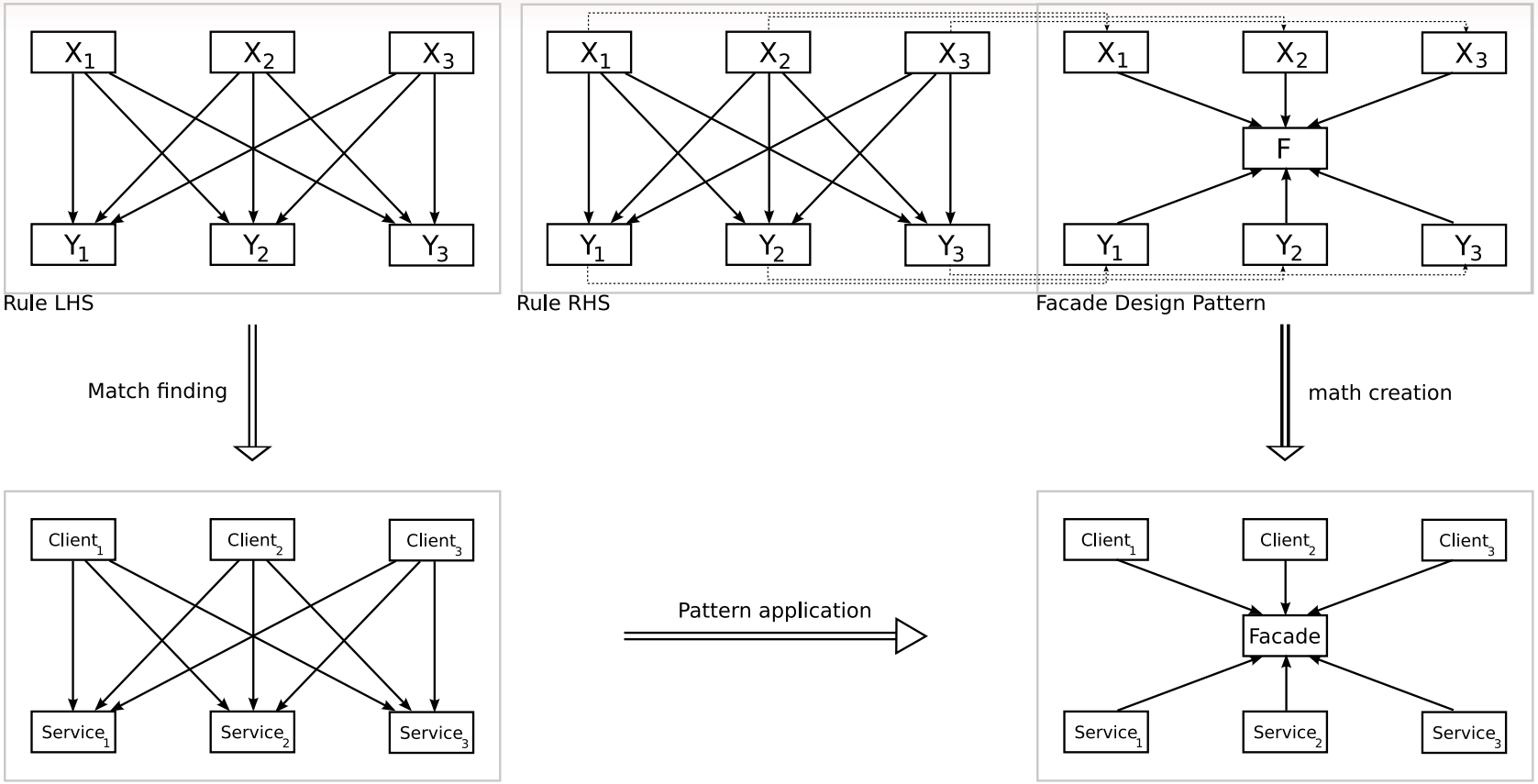


Rule LHS Rule RHS Facade Design Pattern

- ... pattern enforcement by defining a transformation (refactoring) rule which has the design pattern as its RHS



- Match finding: is the pre-condition (LHS) of the rule satisfied?



- Apply pattern: by applying the transformation rule

Pattern as Model Transformation

- Specification of pattern
- Define refactoring rule
- Model refactoring is done "conceptually" in two steps:
  - Step one adding structure by using pushout construction
  - Step two deleting structure by pullback construction

Outline

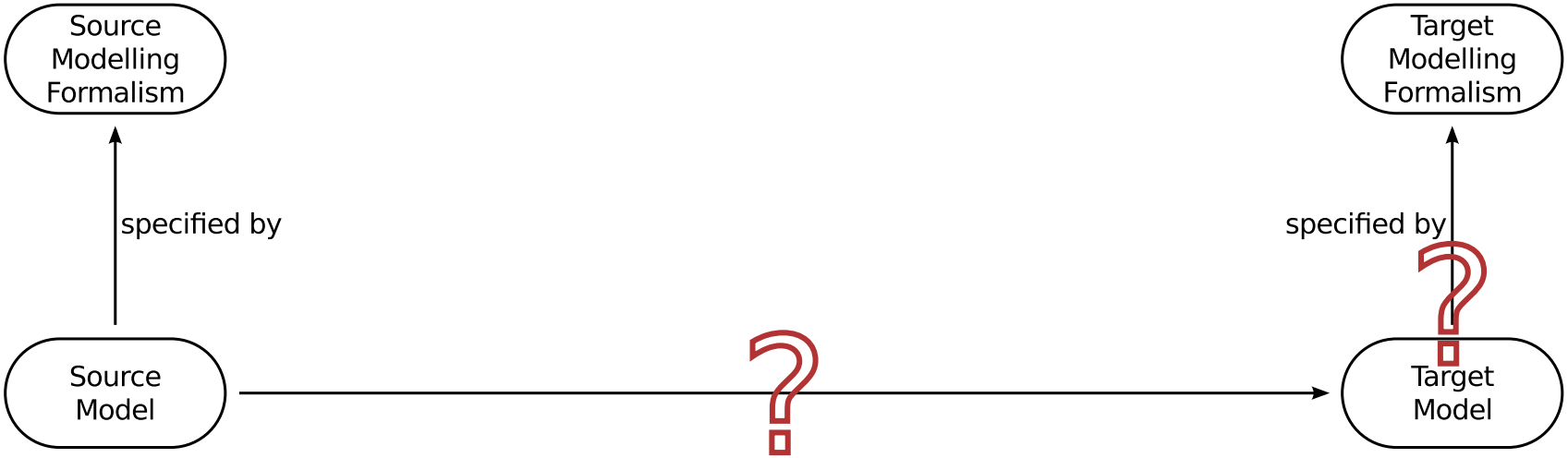
Model Transformation Approach

- Given source and target modelling formalisms and a source model ...

Model Transformation Approach

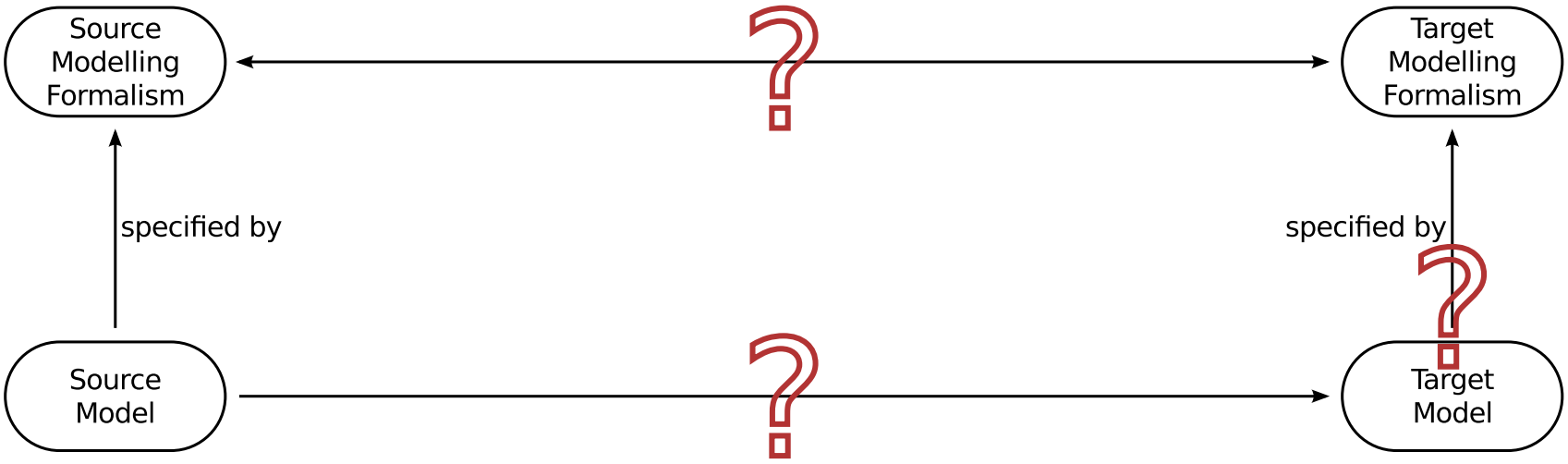
- ... we want to generate a target model

Model Transformation Approach



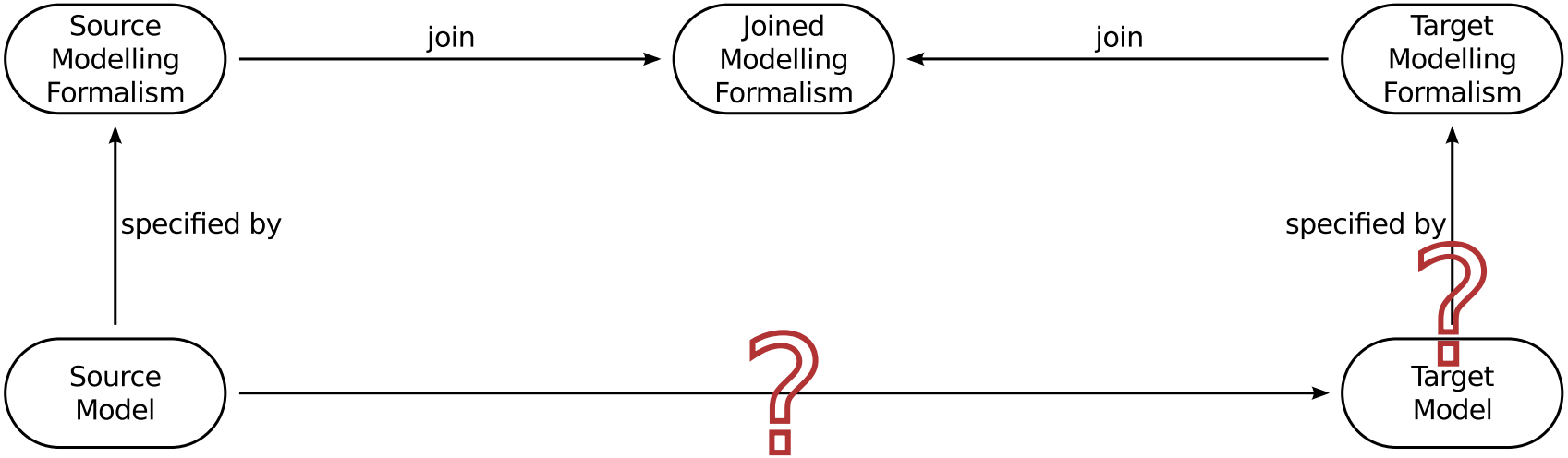
- How we get the target model from the source model?

Model Transformation Approach



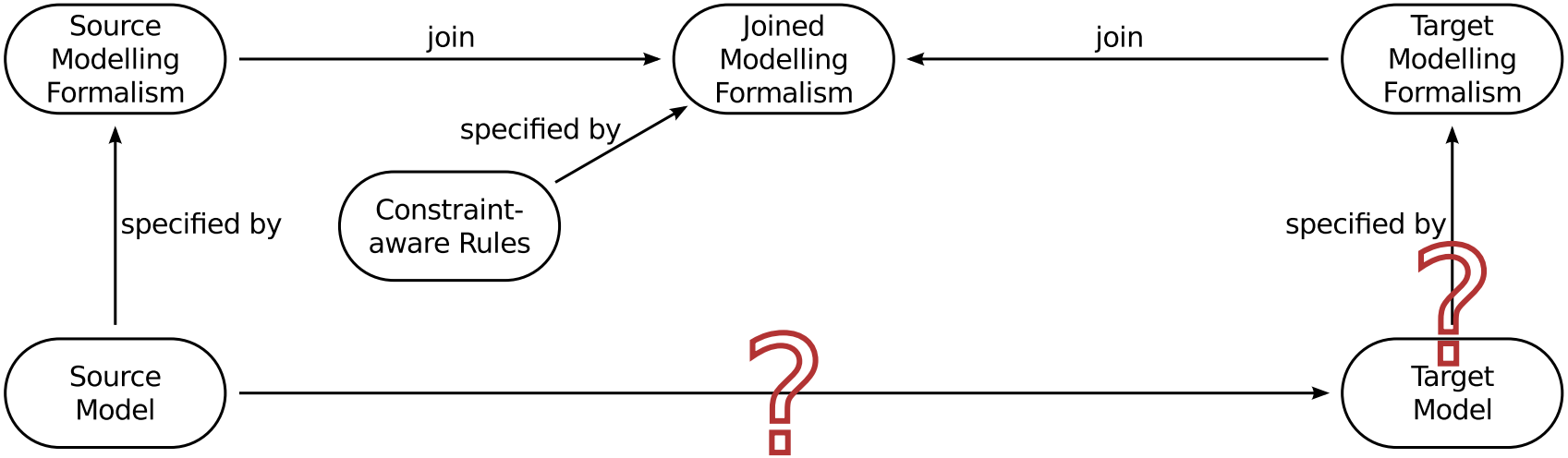
- We have to relate the modelling formalisms

Model Transformation Approach



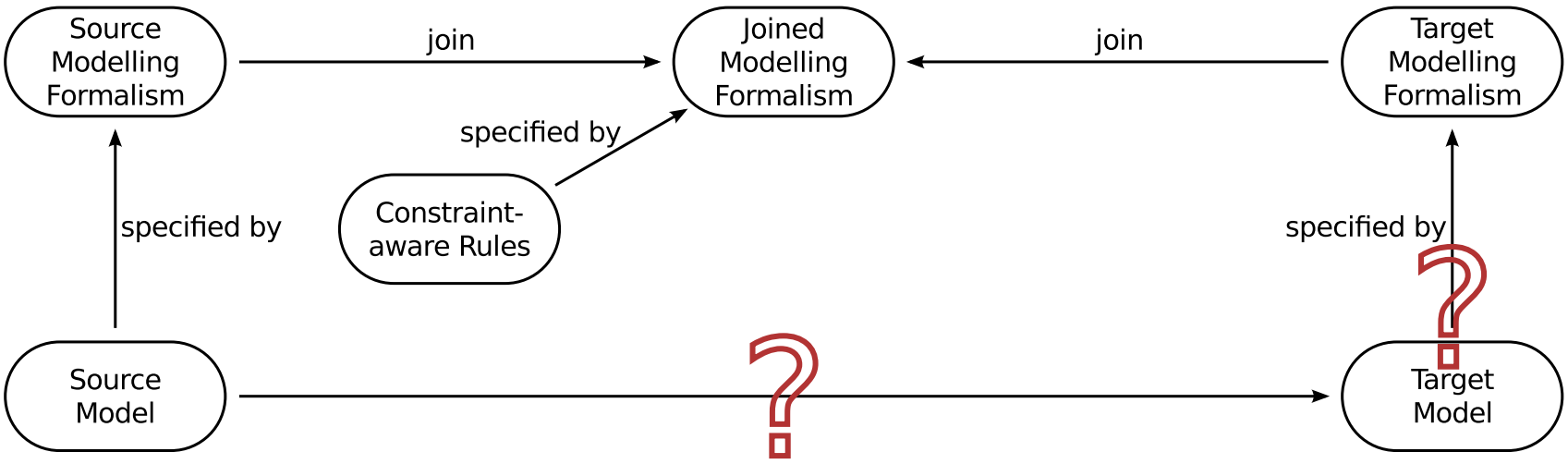
- First: Define morphisms to an appropriate joined modelling formalism

Model Transformation Approach



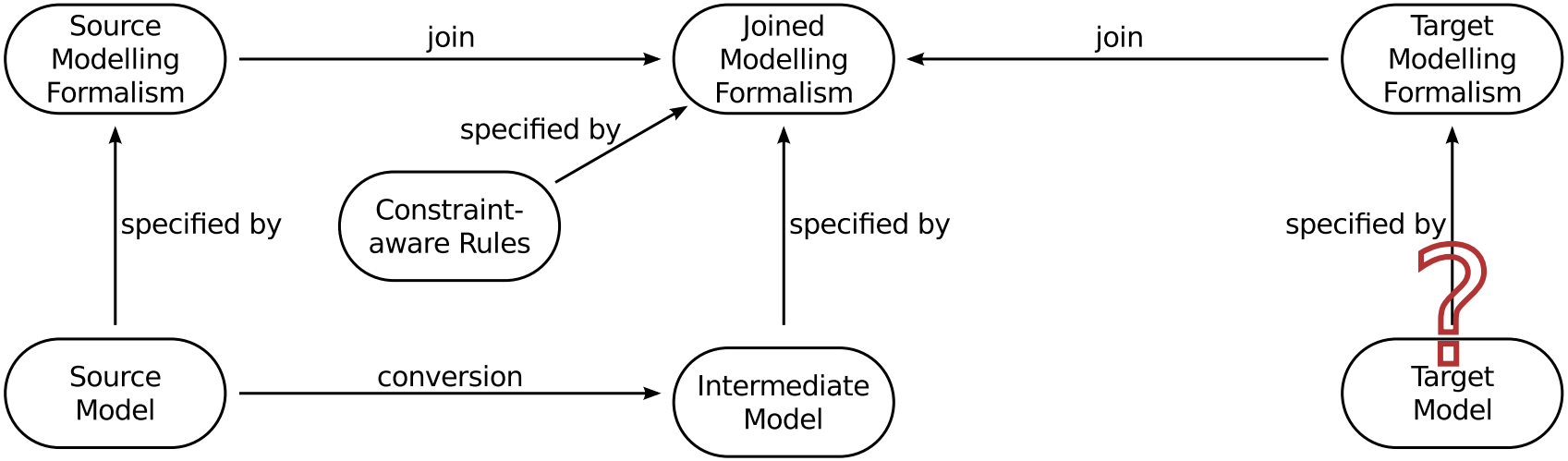
- Second: Define constraint-aware transformation rules

Model Transformation Approach



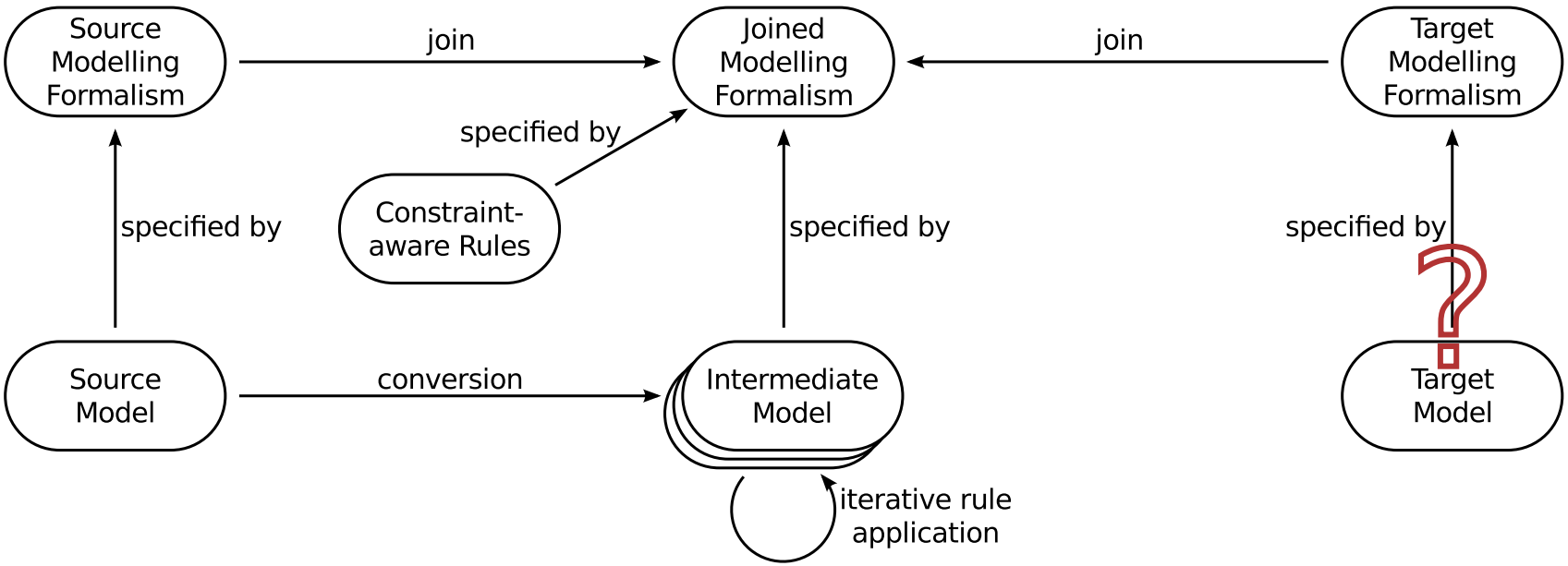
- Third: Apply the model transformation

Model Transformation Approach



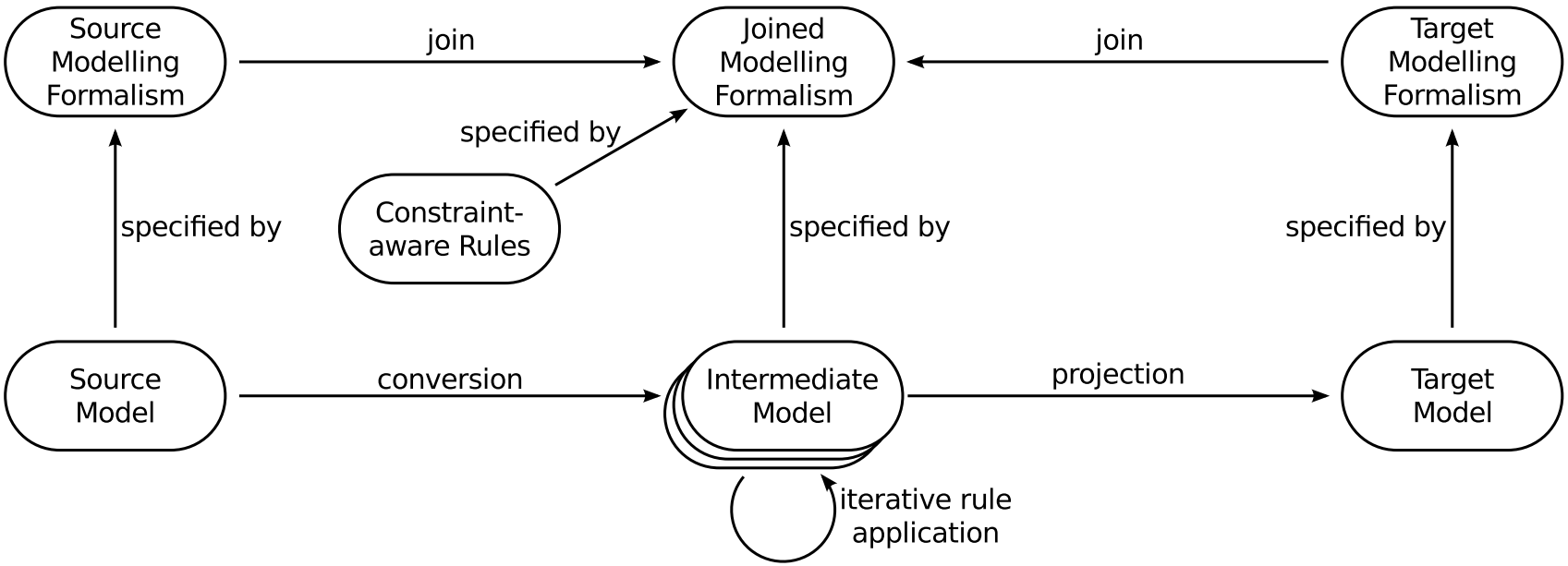
- Third: Apply the model transformation
  - A: Convert the source model to an intermediate model

Model Transformation Approach



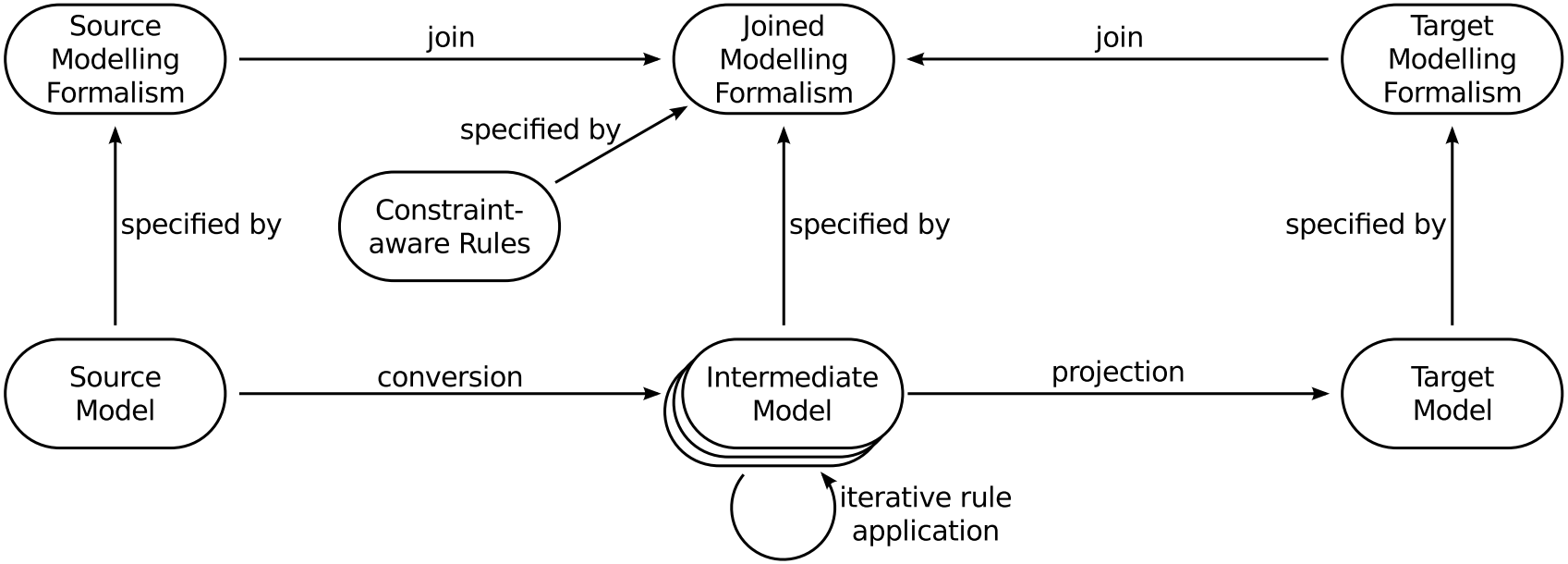
- Third: Apply the model transformation
  - B: Apply the transformation rules (Pushout)

Model Transformation Approach



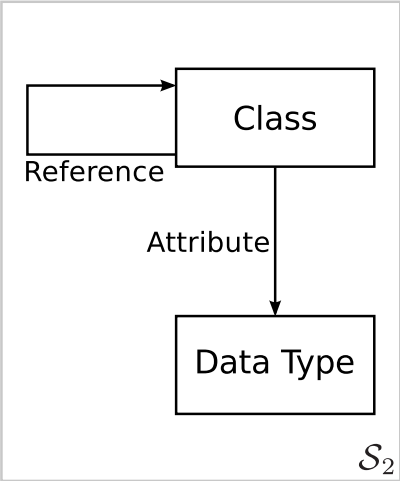
- Third: Apply the model transformation
  - C: Project out the target model (Pullback)

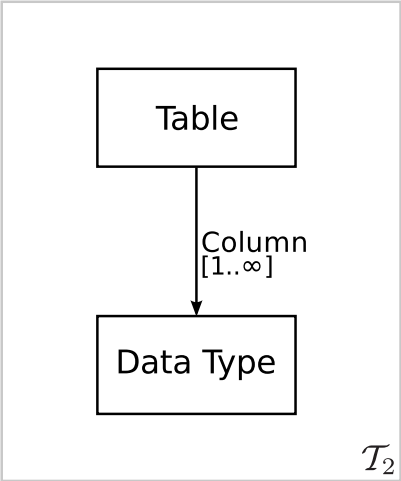
Model Transformation Approach



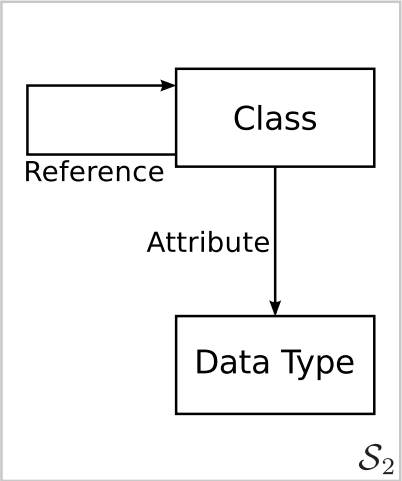
- Heterogeneous, out-place model transformation

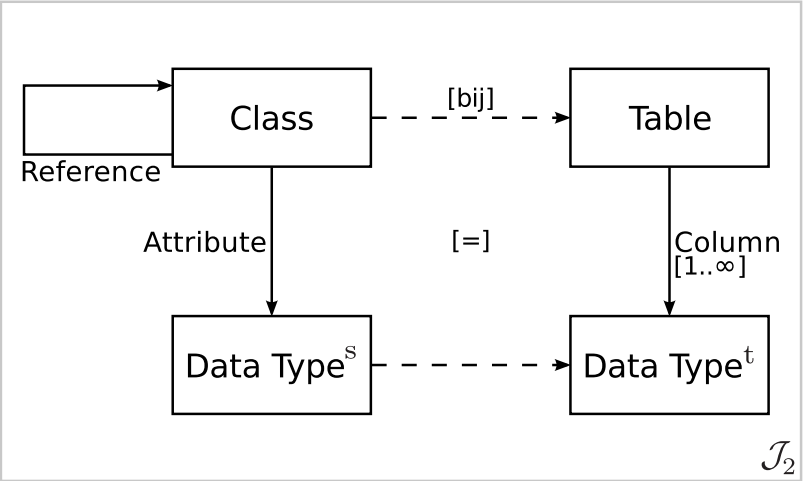
Step 1: Joining Modelling Formalisms

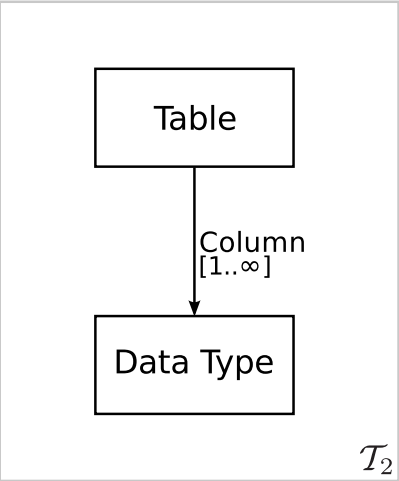




Step 1: Joining Modelling Formalisms







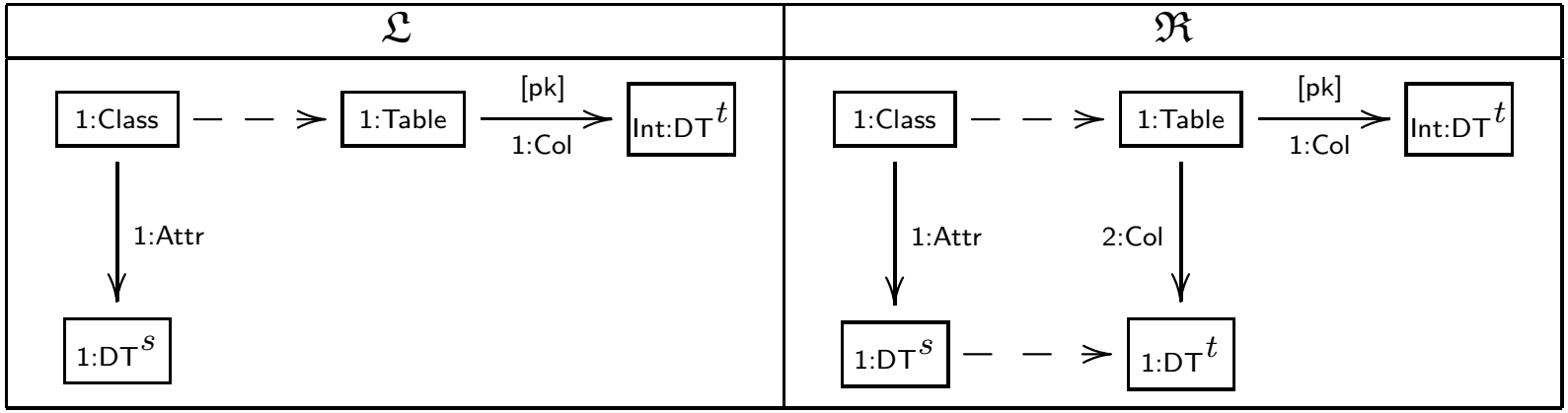
Step 2: Define Constraint-Aware Rules

- Rule r_1. Class to table and primary key

$\mathfrak{L}$	$\mathfrak{R}$
1:Class	1:Class $\dashrightarrow$ 1:Table $\xrightarrow[\text{1:Col}]{\text{[pk]}}$ Int:DT^t

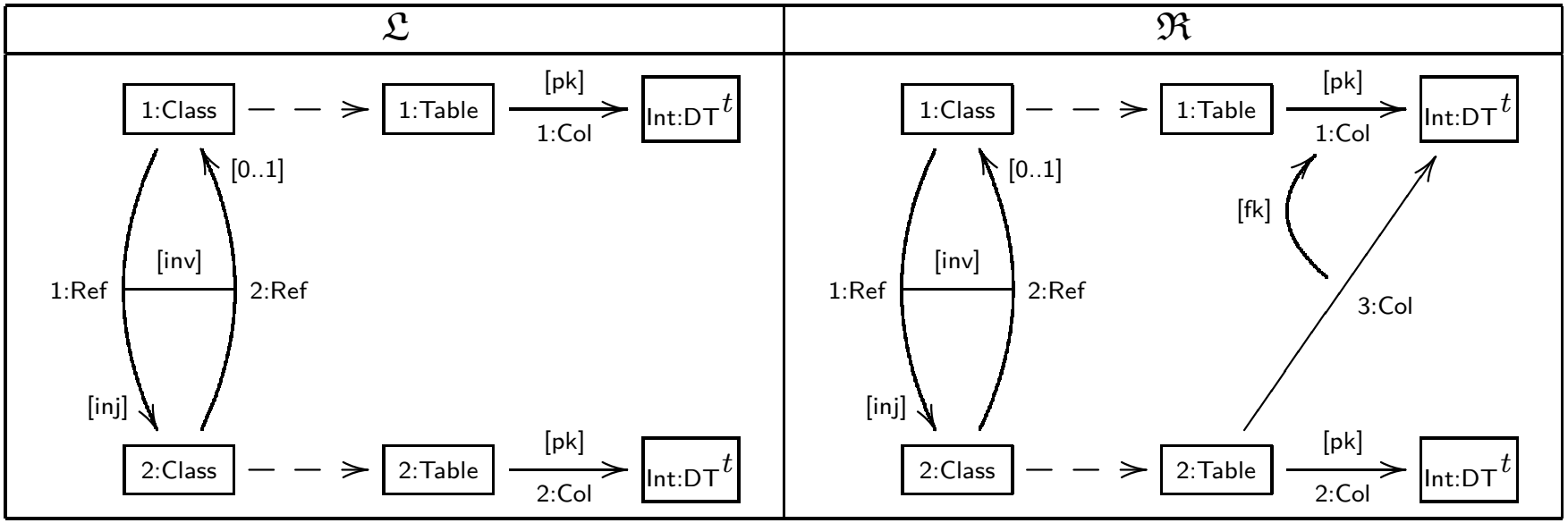
Step 2: Define Constraint-Aware Rules

- Rule r_2. Attribute to column



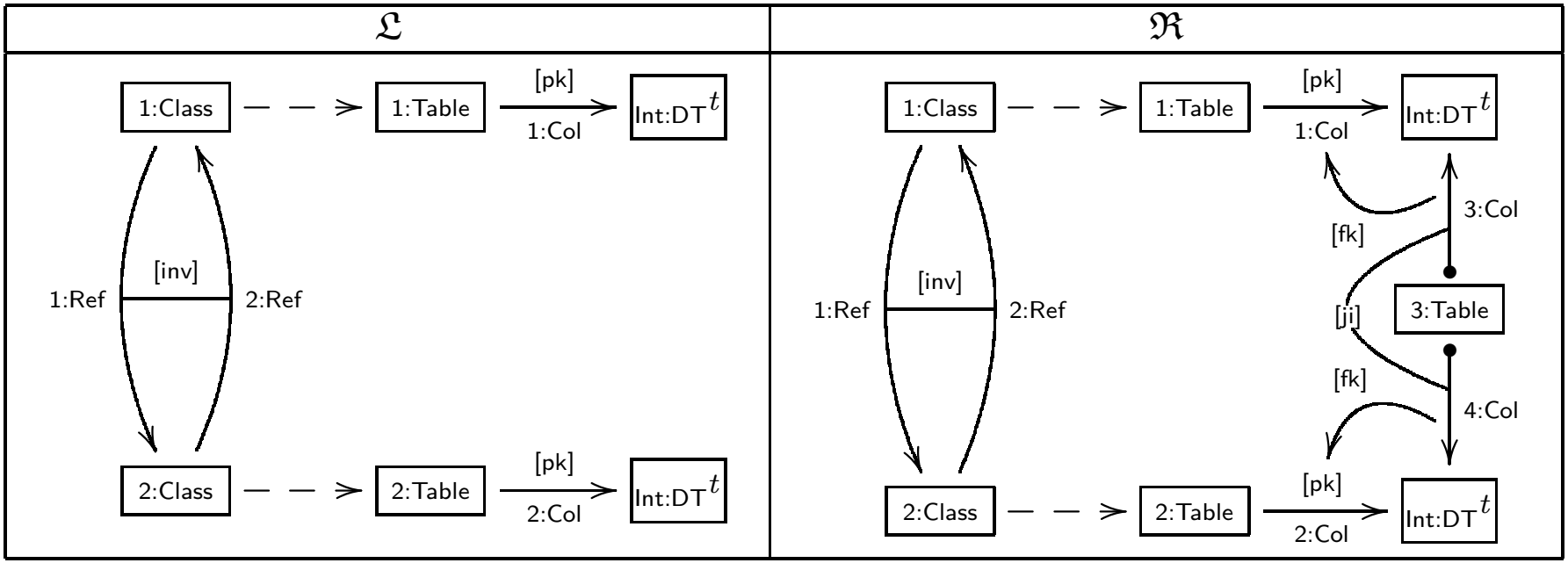
Step 2: Define Constraint-Aware Rules

- Rule r_3. Many-to-one references to foreign key



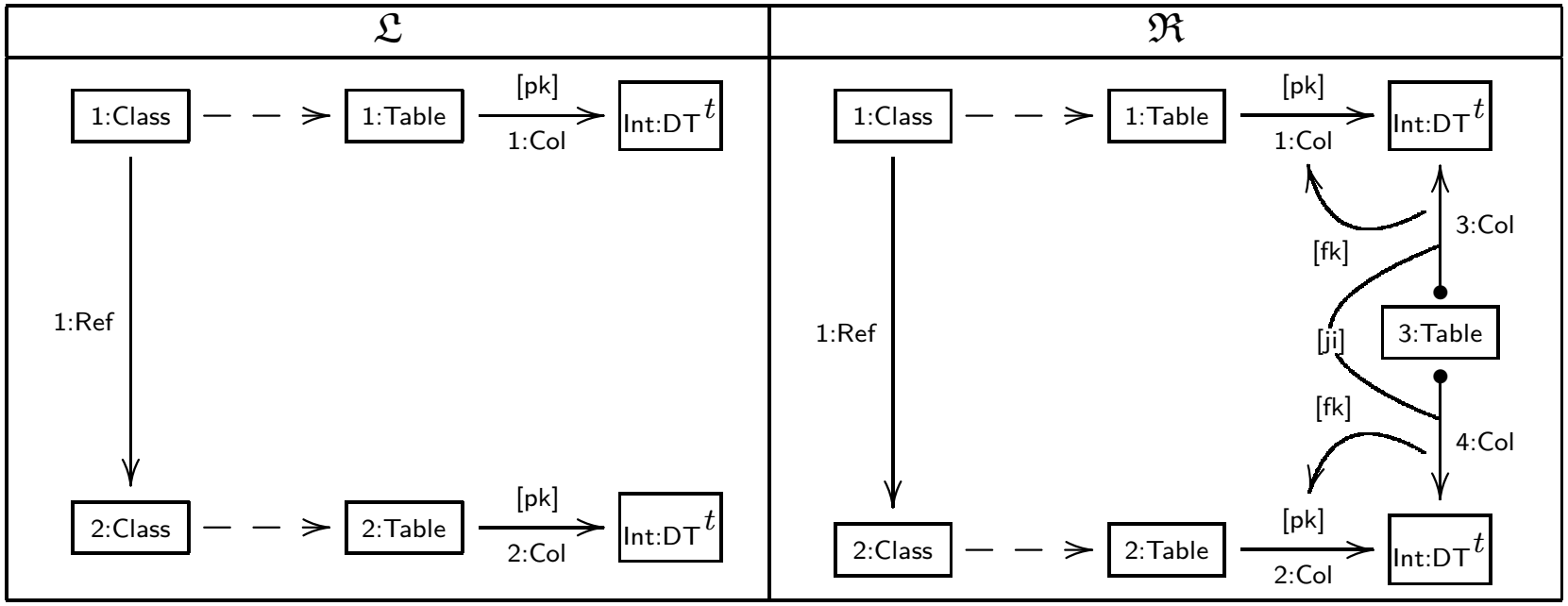
Step 2: Define Constraint-Aware Rules

- Rule r_4. Many-to-many references to link table and foreign key



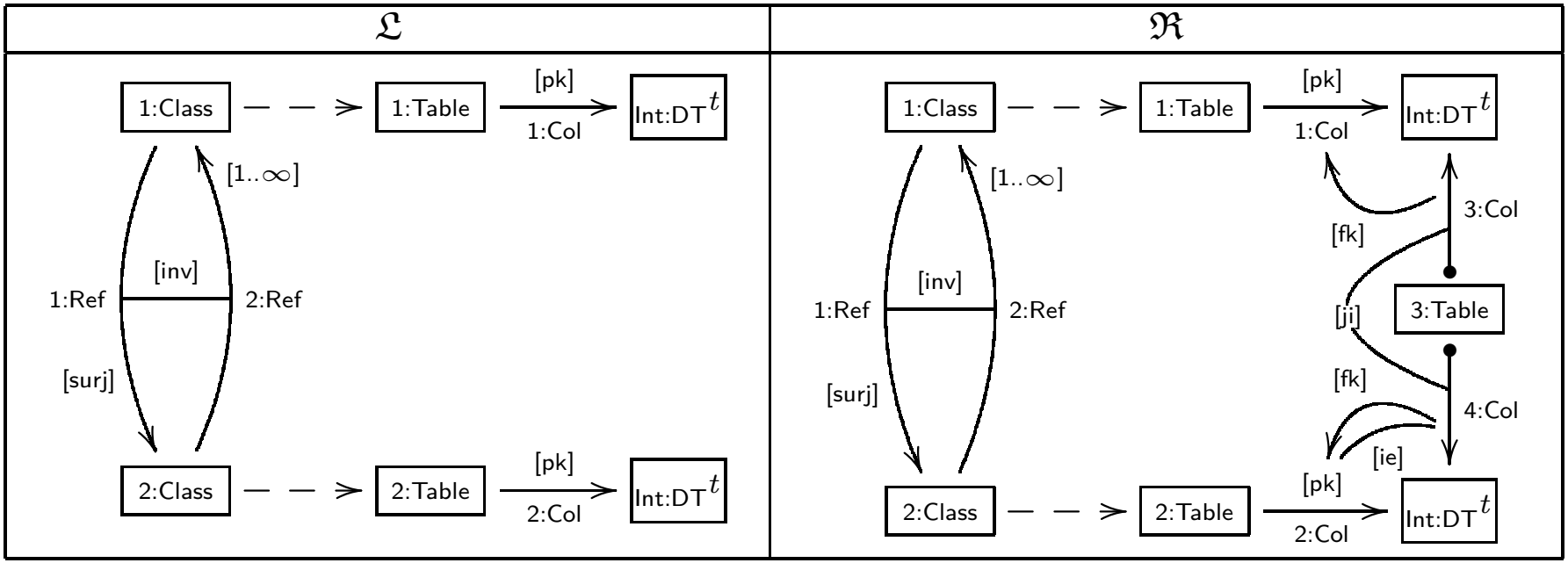
Step 2: Define Constraint-Aware Rules

- Rule r_5. Many-to-many reference to link table and foreign keys

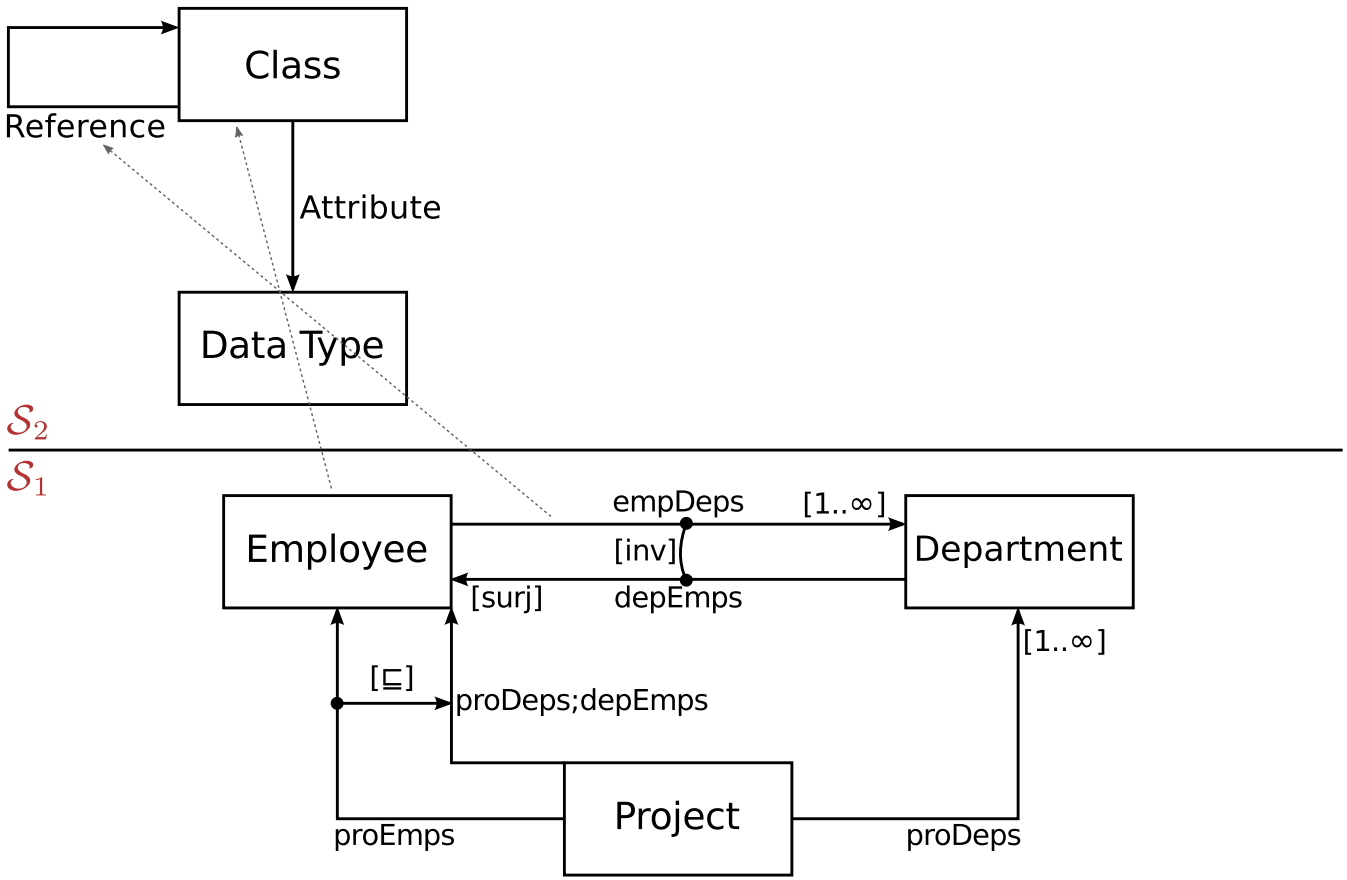


Step 2: Define Constraint-Aware Rules

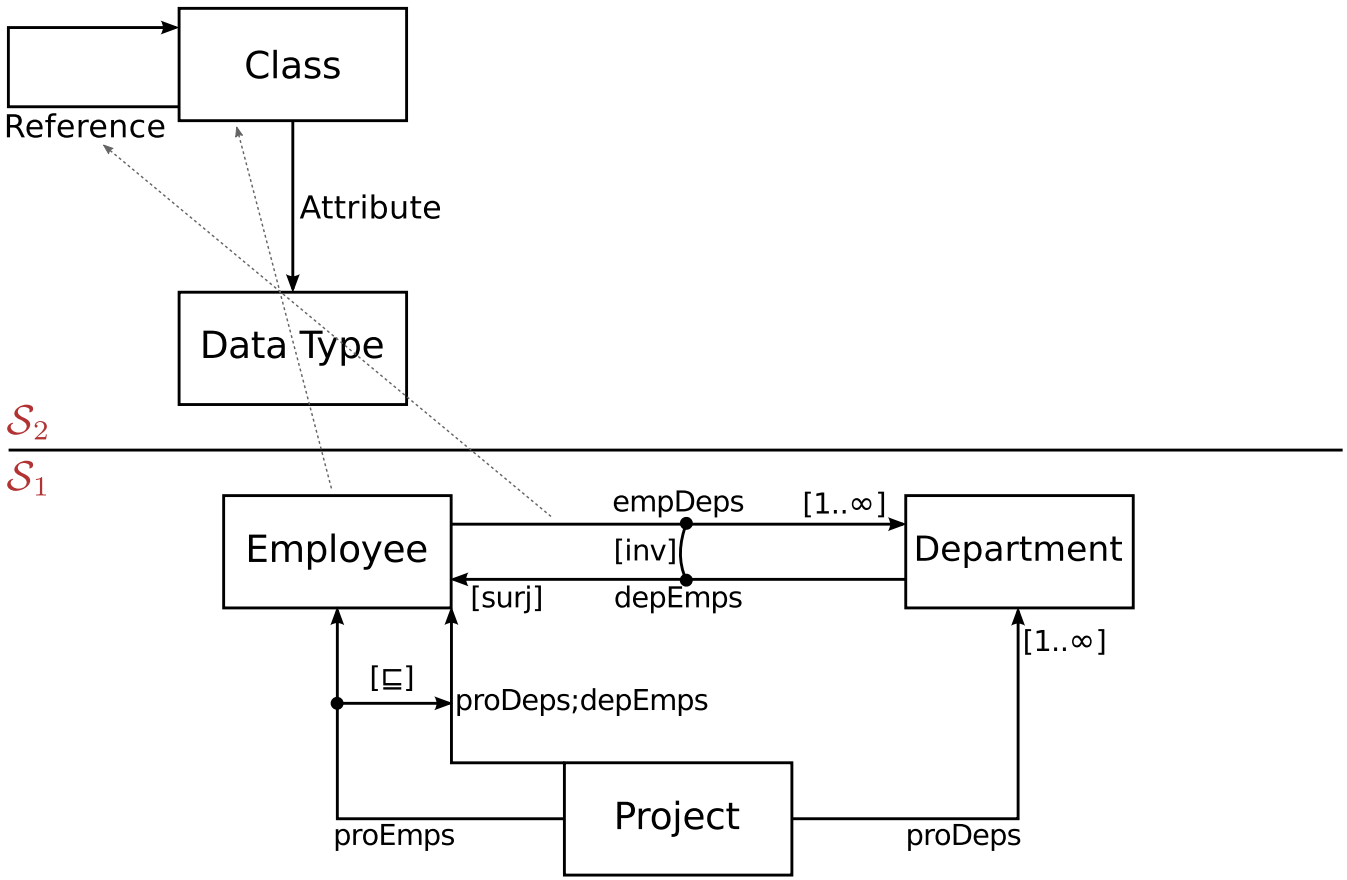
- Rule r_6. `[inverse]` and `[surjective]` to `[foreign-key]`, `[image-equal]` and `[jointly-injective]`



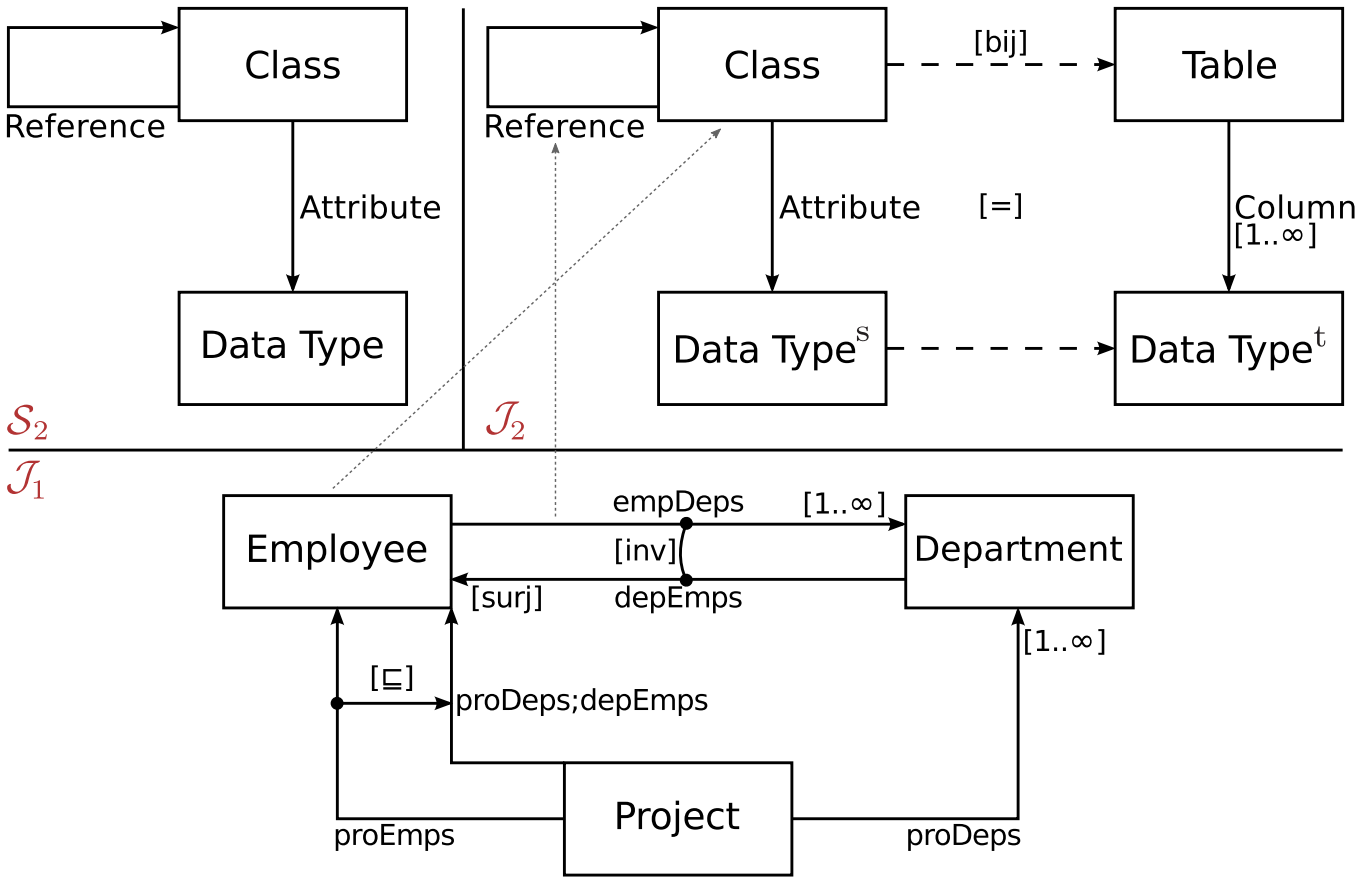
Step 3: Applying Model Transformation



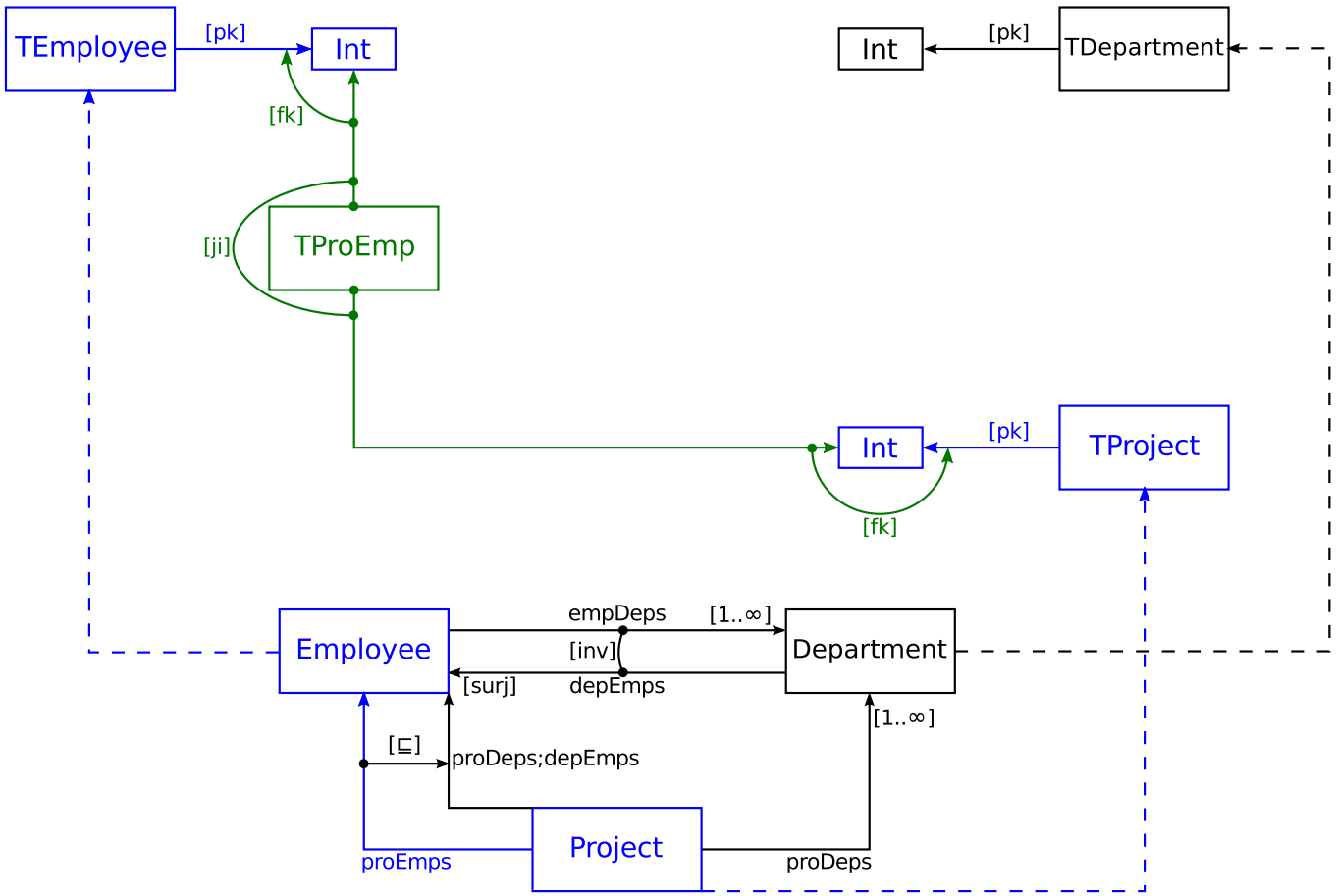
A: Type Conversion of Source Model



A: Type Conversion of Source Model

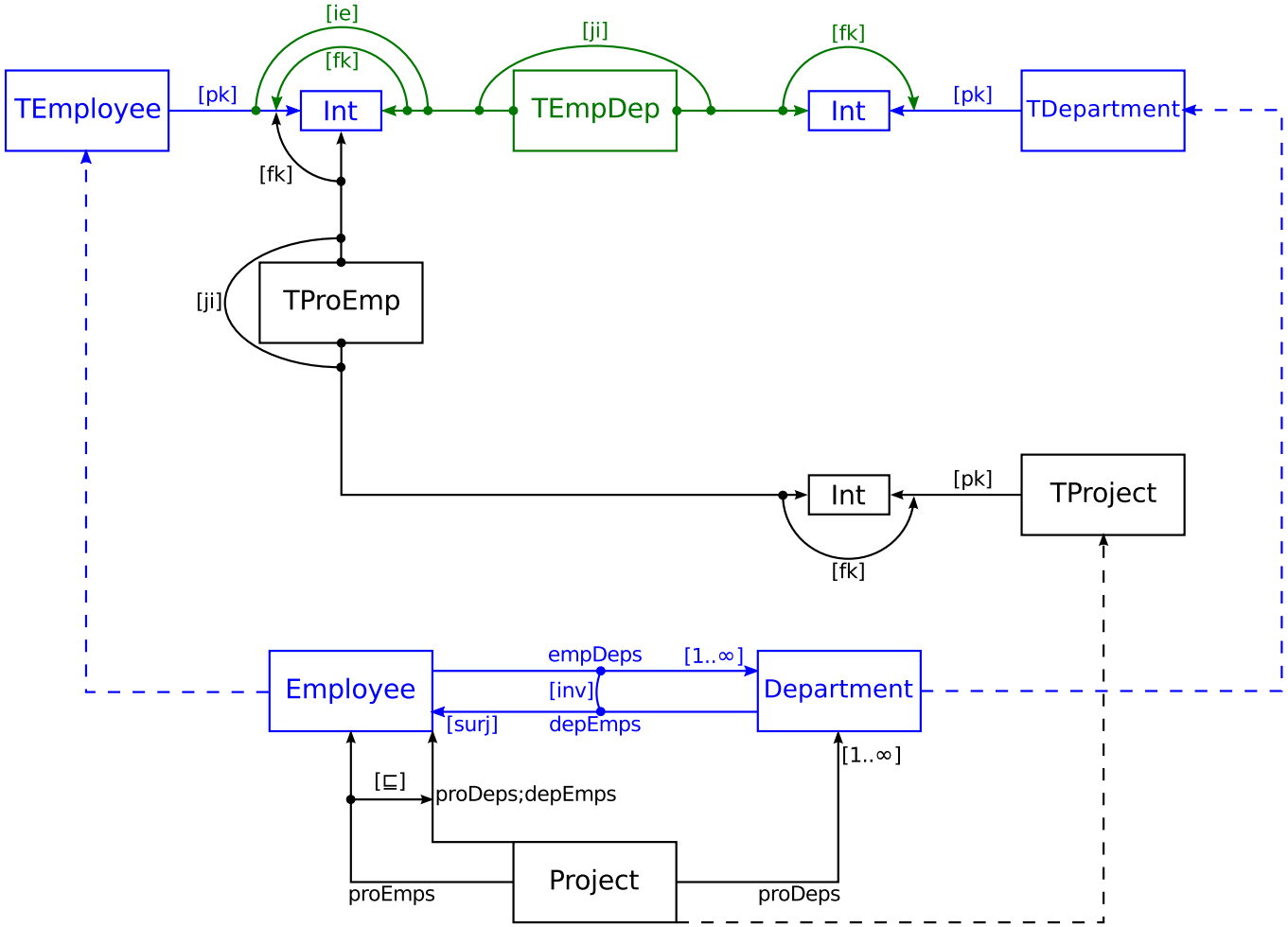


B: Applying the Rules (after r_5)



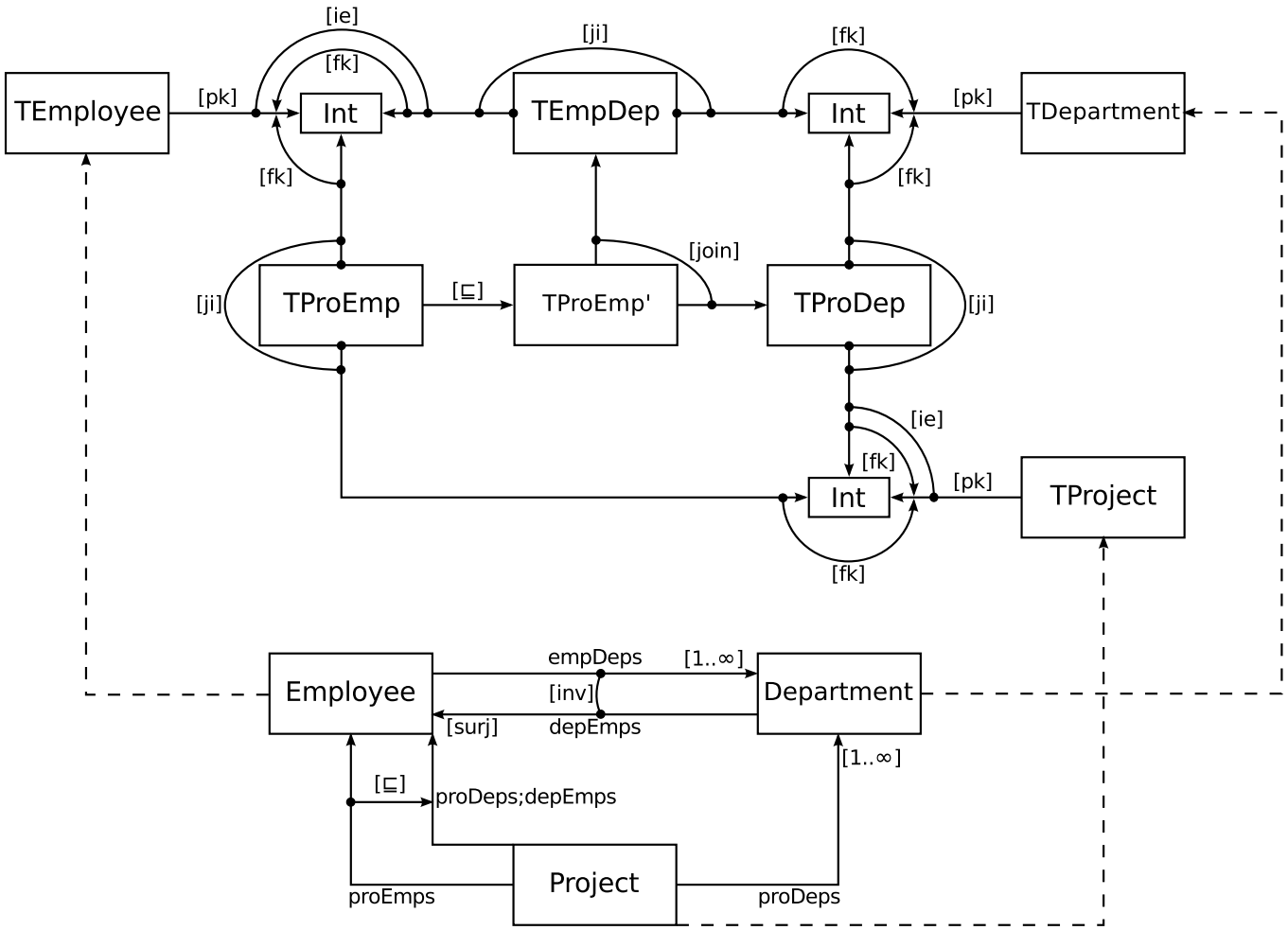
Blue: Matched, Green: Added

B: Applying the Rules (after r_6)



Blue: Matched, Green: Added

Last Intermediate Model



Outline

Summary

- Diagram Predicate Framework (DPF)
  - A "pattern" for definition of diagrammatic specification formalisms
  - Integration of constraints in (meta)modelling
- Pattern as model transformation
- Constraint-aware model transformation
  - Source constraints transformed to target constraints
  - Constraints used to control
    - which structure to create in the target model
    - which constraints to add to the created structure

Summary

- Diagram Predicate Framework (DPF)
  - A "pattern" for definition of diagrammatic specification formalisms
  - Integration of constraints in (meta)modelling
- Pattern as model transformation
- Constraint-aware model transformation
  - Source constraints transformed to target constraints
  - Constraints used to control
    - which structure to create in the target model
    - which constraints to add to the created structure
- Out-place approach to heterogeneous model transformation

Related work

- Graph Transformation System (GTS) *Hartmut Ehrig et al*
  - Adding support for transforming constraints
- Triple Graph Grammar (TGG) *Schürr et al*
  - Adding support for diagrammatic constraints in the joined modelling formalism
- Formalisation of Patterns *De Lara et al*
  - Pattern-based model-to-model transformations
  - GoF design patterns
  - Using Triple Graph Grammars

Further reading

- Diagram Predicate Framework web site, `http://dpf.hib.no`
- A Diagrammatic Formalisation of MOF-Based Modelling Languages, A. Rutle, A. Rossini, Y. Lamo, U. Wolter. LNBIP 33, pp. 37–56. Springer, 2009.
- A Formalisation of the Copy-Modify-Merge Approach to Version Control in MDE, A. Rossini, A. Rutle, Y. Lamo, U. Wolter. Journal of Logic and Algebraic Programming, 2010.
- A Formalisation of Constraint-Aware Model Transformations, A. Rutle, A. Rossini, Y. Lamo, U. Wolter. LNCS 6013, pp. 13–28. Springer, 2010
- Diagram Predicate Framework: A Formal Approach to MDE. A. Rutle. PhD thesis, Department of Informatics, University of Bergen, Norway, September 2010

Thank you!

Questions?

Challenges in modelling

- Mixing graph-based structures with textual constraints
  - Different technical spaces
    - checking models in two different engines/steps
    - model-constraint synchronisation problem
    - violation of "everything-is-a-model" vision of MDE
  - Challenge for domain experts who do not understand OCL

Formalisation approach

- Based on category theory
  - Sketches formalism: define semantics of diagrams (thus graph-based models)
    - models: graphs (nodes and edges)
    - model properties: universal properties (limit, colimit, commutative diagrams)
  - Generalized sketches formalism
    - not only universal properties
    - user-defined diagrammatic predicate signatures
- DPF: specification formalism based on generalized sketches

Sample Signature for Relational Data Models

p	$\alpha^{\Theta_2}(p)$	Proposed vis.	Intended semantics
[primary-key]	$1 \xrightarrow{f} 2$	X $\xrightarrow[\text{[pk]}]{\text{f}}$ Y	f is [total] and [injective]
[foreign-key]	$1 \xrightarrow{f} 2$, $3 \xrightarrow{g} 2$	X $\xrightarrow{\text{f}}$ Y, X $\xrightarrow{\text{[fk]}}$ Y, Z $\xrightarrow{\text{g}}$ Y	$f(X) \subseteq g(Y)$
[image-equal]	$1 \xrightarrow{f} 2$, $3 \xrightarrow{g} 2$	X $\xrightarrow{\text{f}}$ Y, X $\xrightarrow{\text{[ie]}}$ Y, Z $\xrightarrow{\text{g}}$ Y	$f(X) = g(Z)$
[join]	$1 \xrightarrow{f} 2$, $3 \xrightarrow{f'} 4$, $3 \xrightarrow{g'} 1$, $4 \xrightarrow{g} 2$	X $\xrightarrow{\text{f}}$ Y, XZ $\xrightarrow{\text{f}'}$ Z, XZ $\xrightarrow{\text{[join]}}$ X, XZ $\xrightarrow{\text{[join]}}$ Z, Z $\xrightarrow{\text{g}}$ Y, XZ $\xrightarrow{\text{g}'}$ X	$\forall x \in X$, $\forall z \in Z$: $(x, z) \in XZ$ iff $f(x) = g(z)$