



**HAL**  
open science

## Towards a Traceability Framework for Model Transformations in Kermeta

Jean-Rémy Falleri, Marianne Huchard, Clémentine Nebut

► **To cite this version:**

Jean-Rémy Falleri, Marianne Huchard, Clémentine Nebut. Towards a Traceability Framework for Model Transformations in Kermeta. J. Aagedal; T. Neple; J. Oldevik. ECMDA-TW'06: ECMDA Traceability Workshop, Jul 2006, Bilbao, Spain. Sintef ICT, Norway, pp.31-40, 2006. lirmm-00102855

**HAL Id: lirmm-00102855**

**<https://hal-lirmm.ccsd.cnrs.fr/lirmm-00102855v1>**

Submitted on 2 Oct 2006

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Towards a traceability framework for model transformations in Kermeta

Jean-Rémy Falleri, Marianne Huchard and Clémentine Nebut

LIRMM, CNRS and Université de Montpellier 2

9 juillet 2006

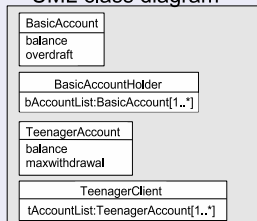
# Outline

- 1 Introduction
- 2 Example
- 3 An intuitive model definition
- 4 Kermeta implementation
- 5 Conclusion and future work

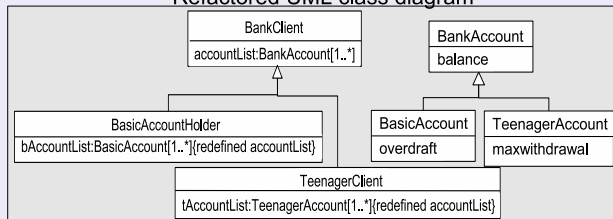
## Introduction

## Context

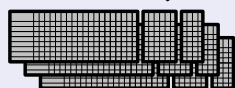
## UML class diagram



## Refactored UML class diagram



## UML2contexts

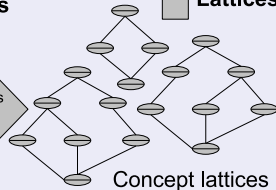


Set of contexts +  
relational contexts

## InitialContexts2FinalLattices



## Lattices2UML



# Introduction

## Context

- Definition of complex model transformations
- Usage of Kermeta language (imperative syntax)
- Transformations using information from previously applied transformations

# Introduction

## Context

- Definition of complex model transformations
- Usage of Kermeta language (imperative syntax)
- Transformations using information from previously applied transformations

## Goals

- **Need for a traceability framework**

# Kermeta

## Description

- Developed at the *IRISA/INRIA (Triskell Team)*, Rennes, France
- Object-oriented language
- Object model close to *MOF*
- Meta reflection
- Definition of structure and operational semantics of metamodels
- Integrated in *Eclipse*, compatible with *EMF*

## Drawbacks

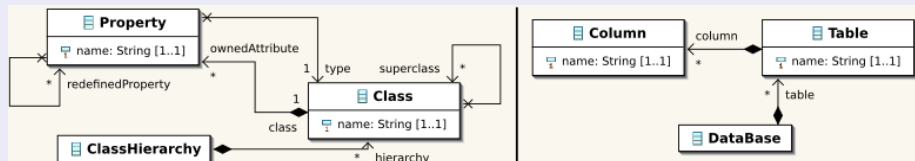
- No natural way to trace model transformations
- Impossible to avoid trace generating code

# Example

## Principe

A *UML* class model is transformed into a *Database*

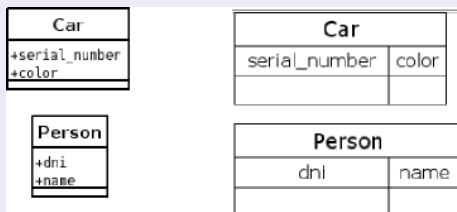
## Metamodels



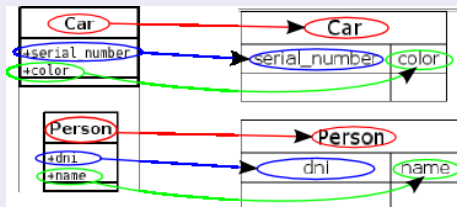


# Example

## Models



## Trace

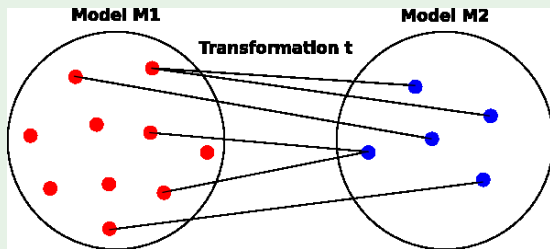


# An intuitive model definition

## Definition

- A model  $M$  is a set of elements.
- Let  $M1$  and  $M2$  be two models. A model transformation is a relation  $t, t \subseteq M1 \times M2$ .

## Example

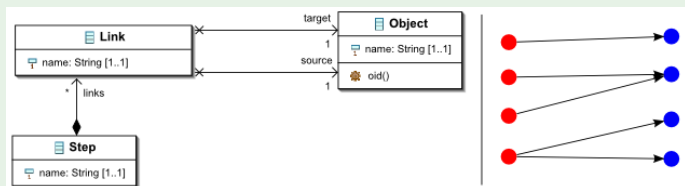


# Trace definition

## Definition

A step of a trace is a bipartite graph. The nodes are partitioned into two categories : source nodes and target nodes.

## Example

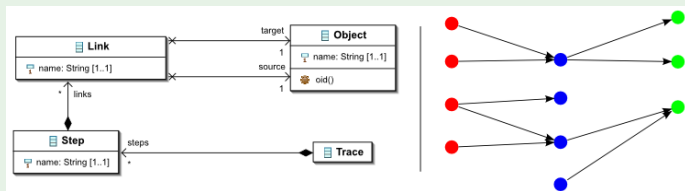


# Trace definition

## Definition

A transformation trace is a set of bipartite graphs.

## Example

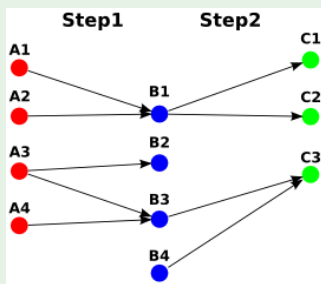


# Basic operations

## Definition

- $parents(C3) = \{B3, B4\}$
- $allparents(C3) = \{B3, B4, A4, A3\}$

## Example



# Implementation

## Features

- Trace serialization (in *XMI 2.0*, thanks to *EMF*)
- Simple transformation from a trace to Graphviz's dot language (*AT-T*), in order to allow trace visualization

## Constraints

- Trace generating code should be as short as possible, and only a small part of it should be placed in the transformation code
- Developers must be able to access to the elements of their choice through the trace
- Developers must be able to select the elements they want to trace

# Serialization

## Problems

- An element can't be contained by two elements
- During execution, developers need references

## Current approach

- Use of references of model elements during execution
- Elements referenced are stored in the models
- Reduced clones are stored in the serialized trace

## Transformation code

```

/**
 * Transform a minuml model to a mindb model
 */
operation transform(source: ClassHierarchy): DataBase is do
    result := DataBase.new // Initialize the target model

    trace.initStep("minuml2mindb") // Trace Generating Code

    source.hierarchy.each{ cls | // Iterate on every class of the source model
        var table: Table init Table.new // Create a Table
        table.name := String.clone(cls.name) // Copy the name of the Class to the table
        result.table.add(table) // Add the table in the target model

        trace.add_link("minuml2mindb", "class2table", cls, table) // Trace Generating Code

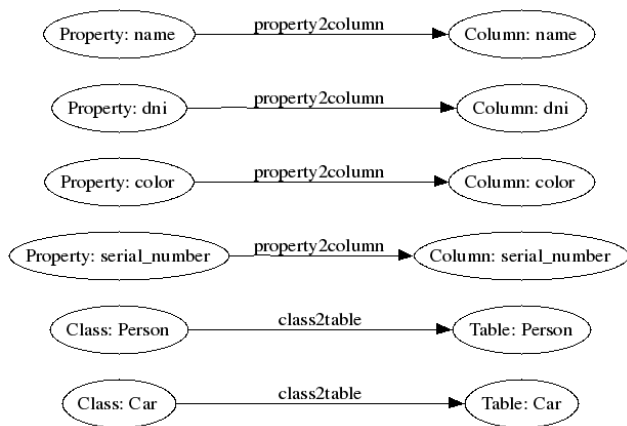
        cls.ownedAttribute.each{ prop | // Iterate on every Property of the Class
            var col: Column init Column.new // Create a new Column
            col.name := String.clone(prop.name)
            table.column.add(col) // Add the Column to the relative Table

            trace.add_link("minuml2mindb", "property2column", prop, col) // Trace Generating Code
        }
    }
end

```



## Sample trace



# Conclusion and future work

## Conclusion

- An intuitive model definition
- A framework in the Kermeta language
- Difficulty to trace with an imperative syntax

## Future work

- Composite links
- Graph-based trace definition

## Open question

- Traceability is a cross-cutting concern
- Using aspects to cope with traceability in imperative transformation languages?