

Model To Model Transformations

Philip Liew



Introduction

- Model driven development of systems
- Necessary to transform models into other models
- Tools needed to ease writing of complex transformations
- Graph Grammars and Graph Transformations technique

Introduction

- Need method to precisely specify operation of transformers on categories of models
- Generate model transformer code from specification

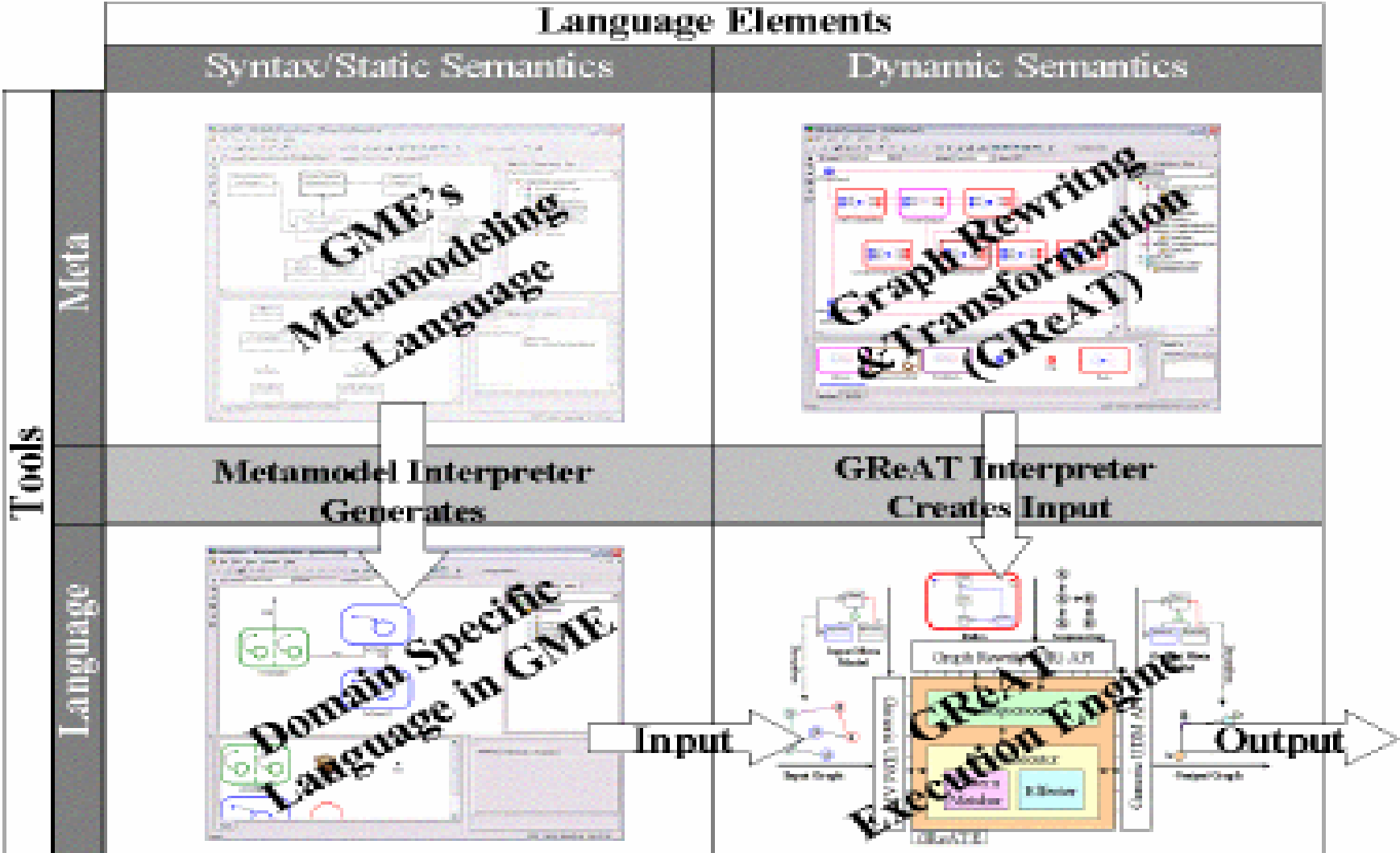
Specifications for Languages

- Language should provide user with methods to specify the different graph domains being used
- Should support transformation to domains different than input domain (heterogeneous transforms)
- Implementation for model transformer should exhibit acceptable performance
- Language should be usable by software engineers of average experience

GReAT

- Graphical Rewrite and Translation
- Intended to meet set specifications
- Emphasis on usability
- Works on the existing Generic Modeling Environment (GME) framework

GReAT Overview



Brief Overview

- Introduction to the Generic Modeling Environment (GME)
- Introduction to the Graph Rewriting and Transformation system (GReAT)
- Introduction to UML to RDBMS Model Transformation
- Review of software

Generic Modeling Environment

- Domain specific design environments use graphical methods to capture specifications. (i.e. Matlab/Simulink or Labview)
- Configurable toolset for creation of domain-specific modeling and program synthesis environments

GME Modeling Concepts

- Paradigm based on Unified Modeling Language (UML)
- Syntactic definitions are modeled using pure UML Class diagrams
- Static semantics are specified using Object Constraint Language (OCL)

MetaGME Paradigm - FCO

- Atoms – Basic limited type entity that has no contained objects. Relationship expressed through name, attributes, and relations it participates in
- Models – Similar to atoms but can contain other atoms, models, or objects

MetaGME Paradigm - FCO

- Connections:
 - Primary concepts that represent relationships.
 - Normally describe relationship between two objects.
 - Can have attributes
- References is an entity with a built-in association to another object. (i.e. alias or pointer)
- Sets help define a collection of objects

MetaGME Paradigm

- Attributes are bound to first class objects (FCO) which are used to store information in the object
- Aspects – used to represent different views of a model. Which objects are chosen to be accessible for certain views
- Constraints – validity rules applied to a model, expressed in OCL predicate language

GReAT

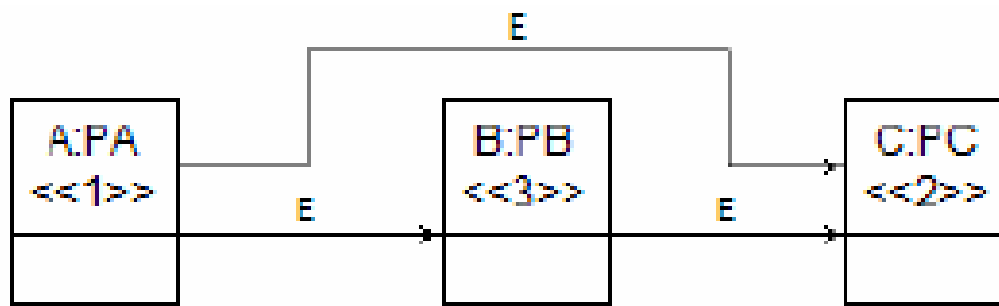
- Language divided into 3 distinct parts:
 1. Pattern Specification Language
 2. Graph Transformation Language
 3. Control Flow Language

Pattern Specification Language

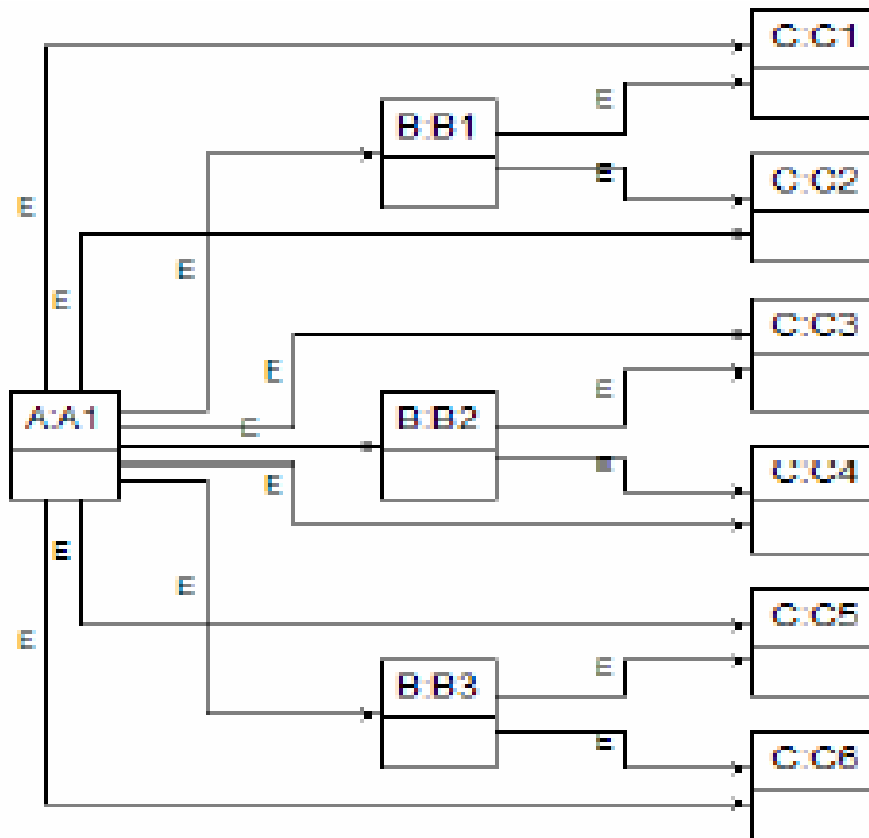
- Introduced language easy to use and tightly coupled to UML class diagrams
- Patterns in language have a one-to-one correspondence with host graph

Pattern Specification Language

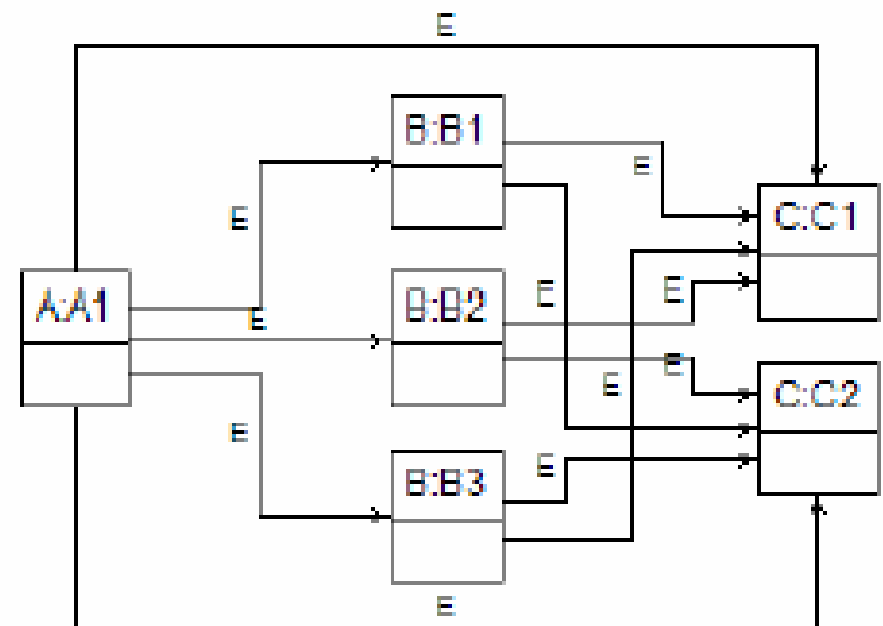
- Cardinality can be specified for each pattern vertex
- Pattern must match n host graph vertices (n represents cardinality)
- Cardinality semantics is ambiguous



(a) Pattern with three vertices



(b) Tree semantics



(c) Set semantics

Pattern Specification Language

- Use of set semantics in GReAT due to:
 - Does not depend on factors such as starting point
 - Set semantics will always return a match of third structure
- GReAT has pattern-matching algorithms for single cardinality and fixed cardinality of vertices

Graph Transformation Language

- Concerns:
 - Specification of static structural constraint in graphs
 - Ensuring constraints are maintained throughout transformation
- Problems:
 - Specify and maintain two different models conforming to two different meta-models
 - Maintaining references between two models

Graph Transformation Language

- Basic Transformation entity consists of a production (or rule)
- Contains pattern graph consisting of pattern vertices and edges
- Pattern vertices are objects that conform to type from metamodel

Graph Transformation Language

- Pattern can play three roles:
 - Bind: used to match objects in graph
 - Delete: when specified pattern is matched, proceed to delete object
 - New: Create object when pattern is matched

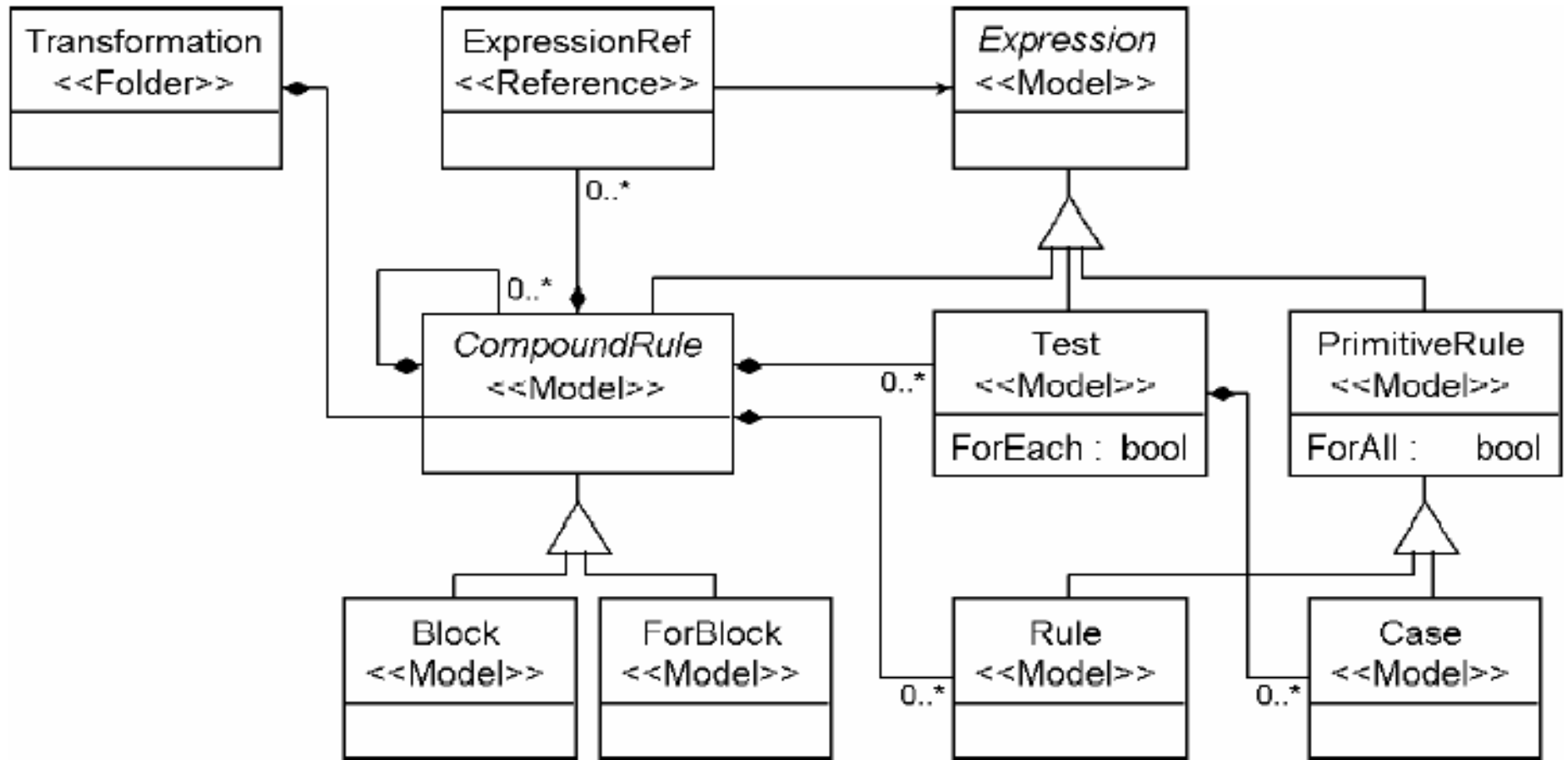
Graph Transformation Language

- Necessary to match non-structural constraints (i.e. values of attributes)
- Use of Guards and Attribute Mappings
- Constraints or preconditions described using Object Constraint Language (OCL)

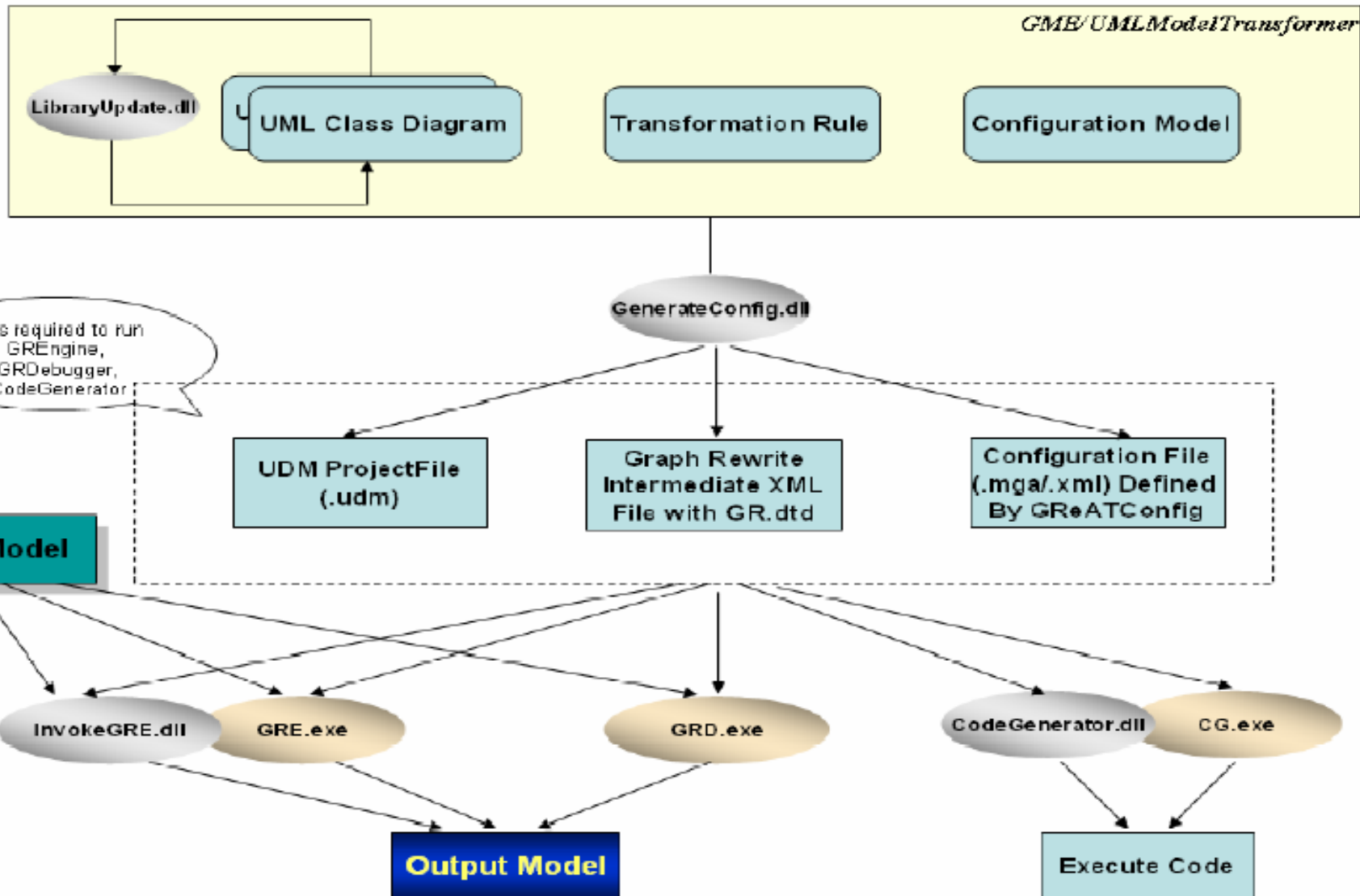
Control Flow Language

- High Level control flow language supports:
 - Sequencing
 - Non-determinism
 - Hierarchy
 - Recursion
 - Test/Case

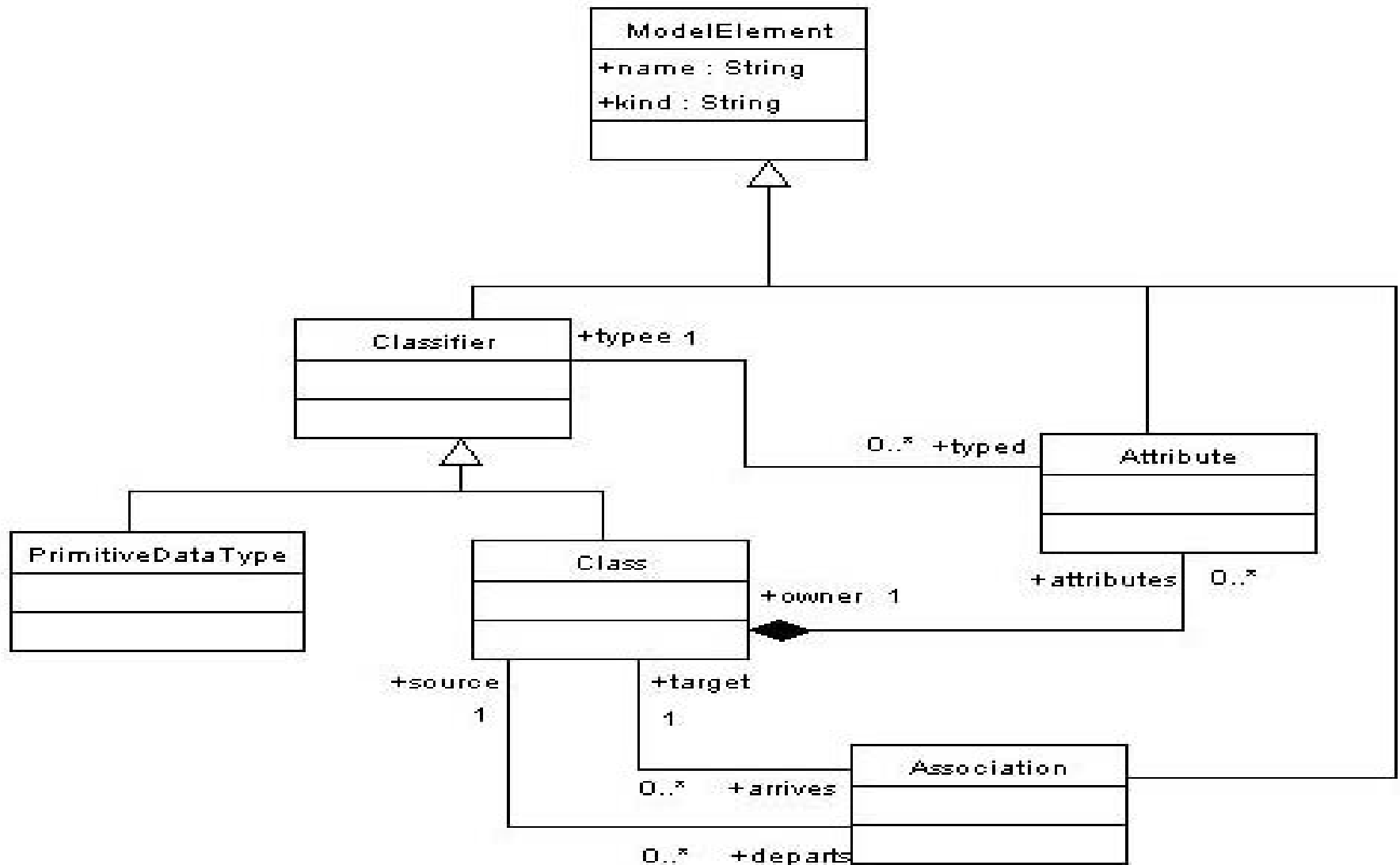
Transformation Language Syntax



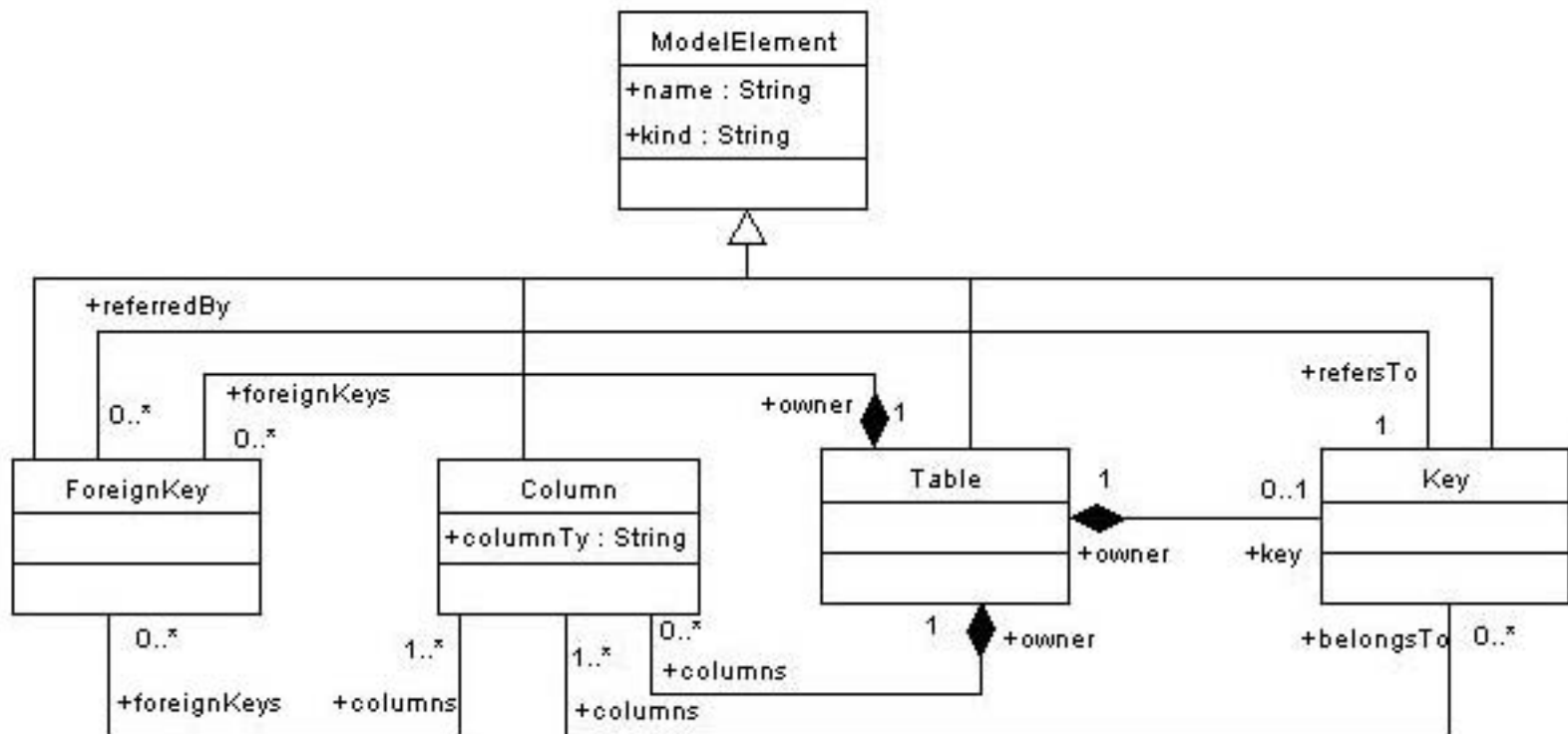
Tool Chain Overview



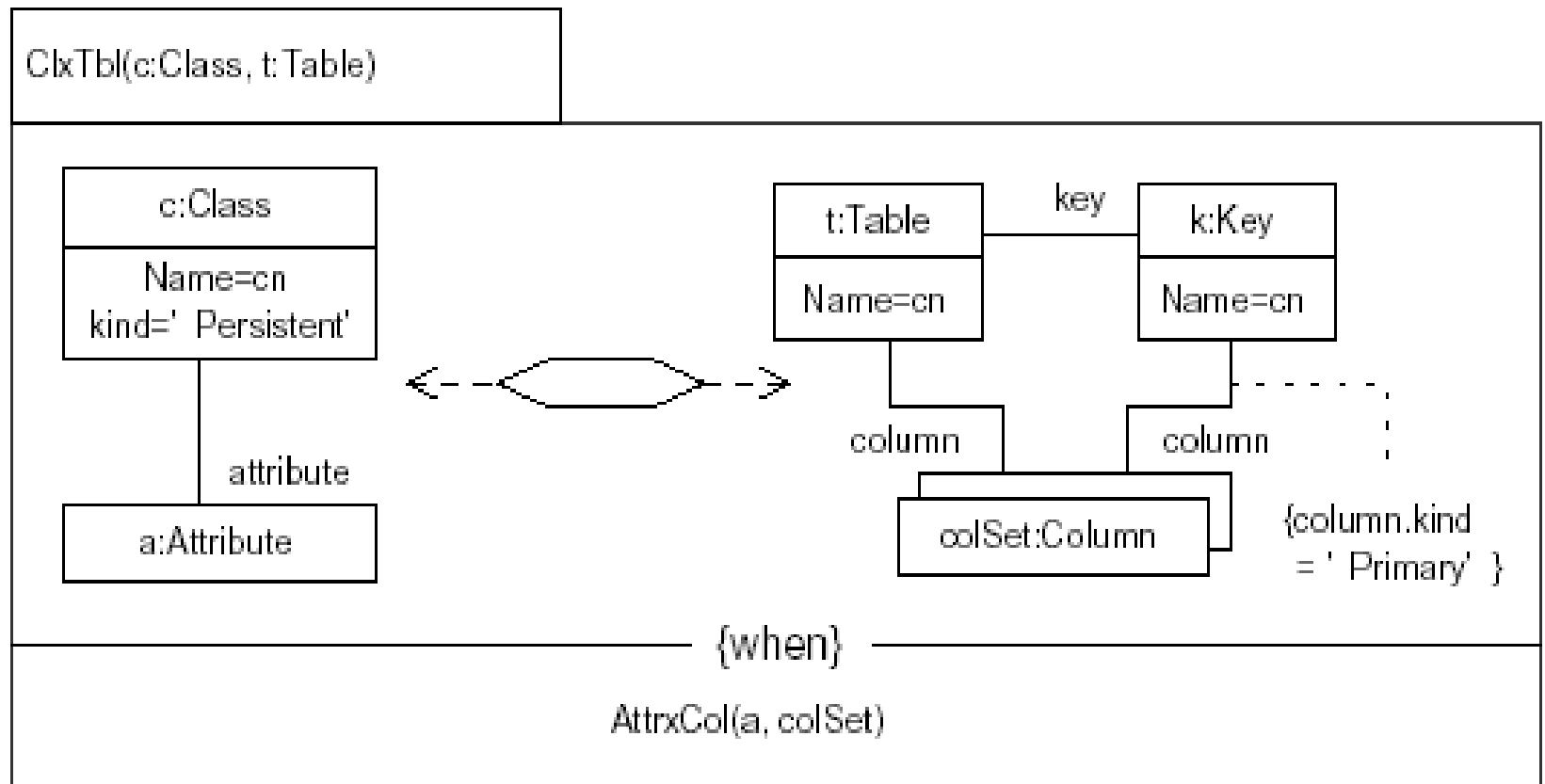
Representation of Simple UML Model



Representation of an RDBMS Model

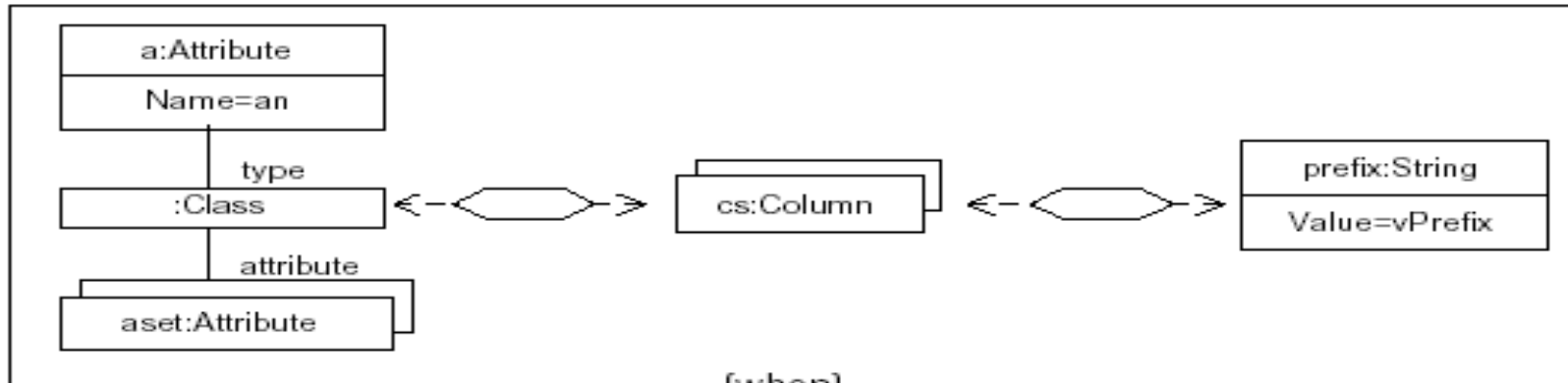


Class to Table Transformation



Attribute to Column Transformation

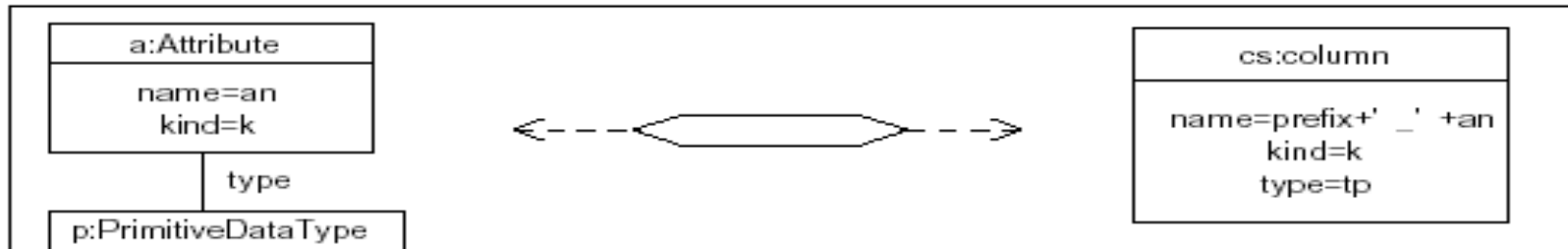
AttrCol(a:Attribute, cs:Set(Column), prefix:String)



{when}

let nPrefix = vPrefix+'_' +an in
 aset->forall(aa | AttrCol(aa, acset, nPrefix) and cs->includesAll(acset))

{or}

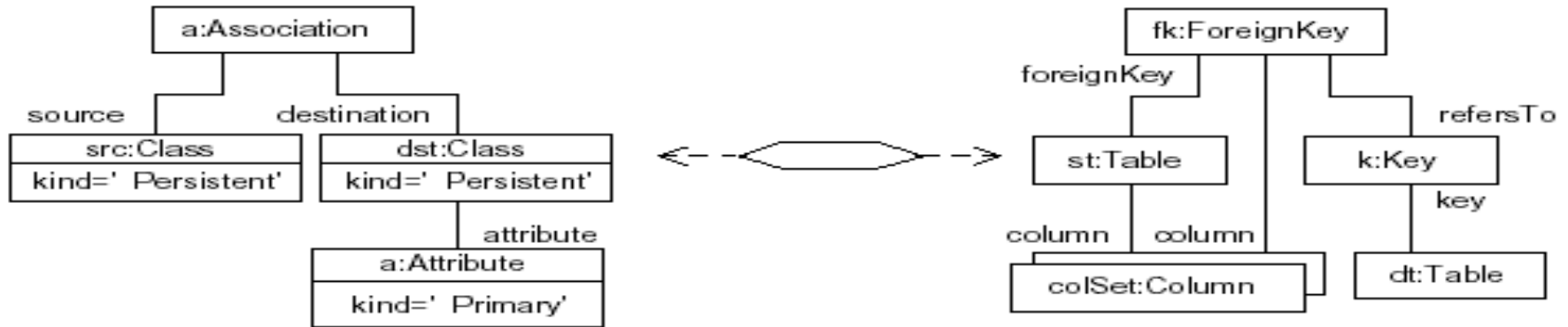


{when}

tp = if p.oclIsTypeOf(Integer) ' number' else ' varchar'

Association to Foreign Key Xform

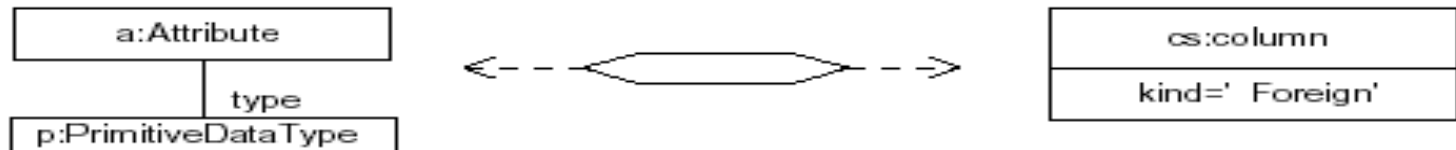
AssocxFKKey(a:Association, fk:ForeignKey)



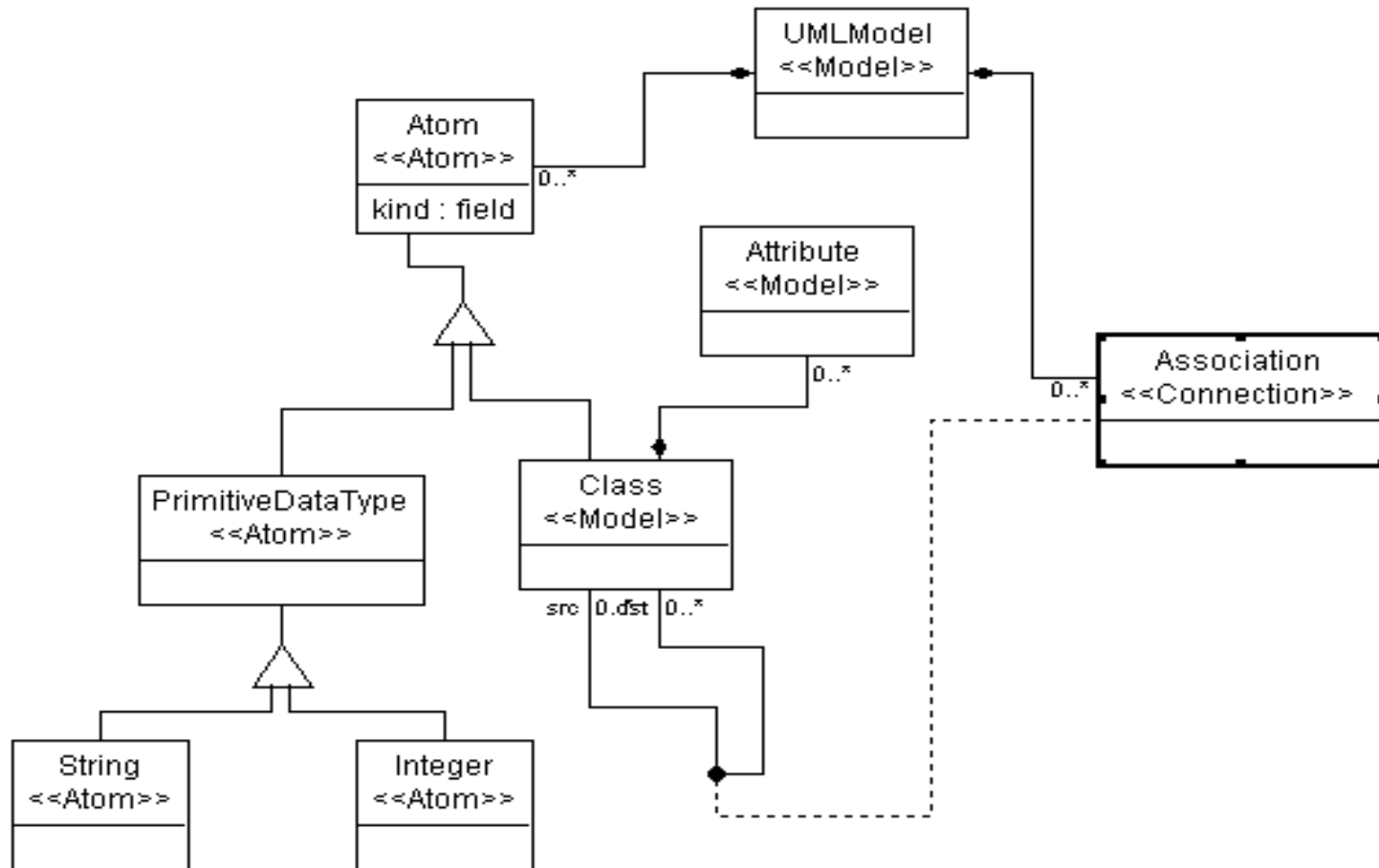
{when}

AttrxFool(a, colSet)
and
ClxTbl(src, st)
and
ClxTbl(dst, dt)

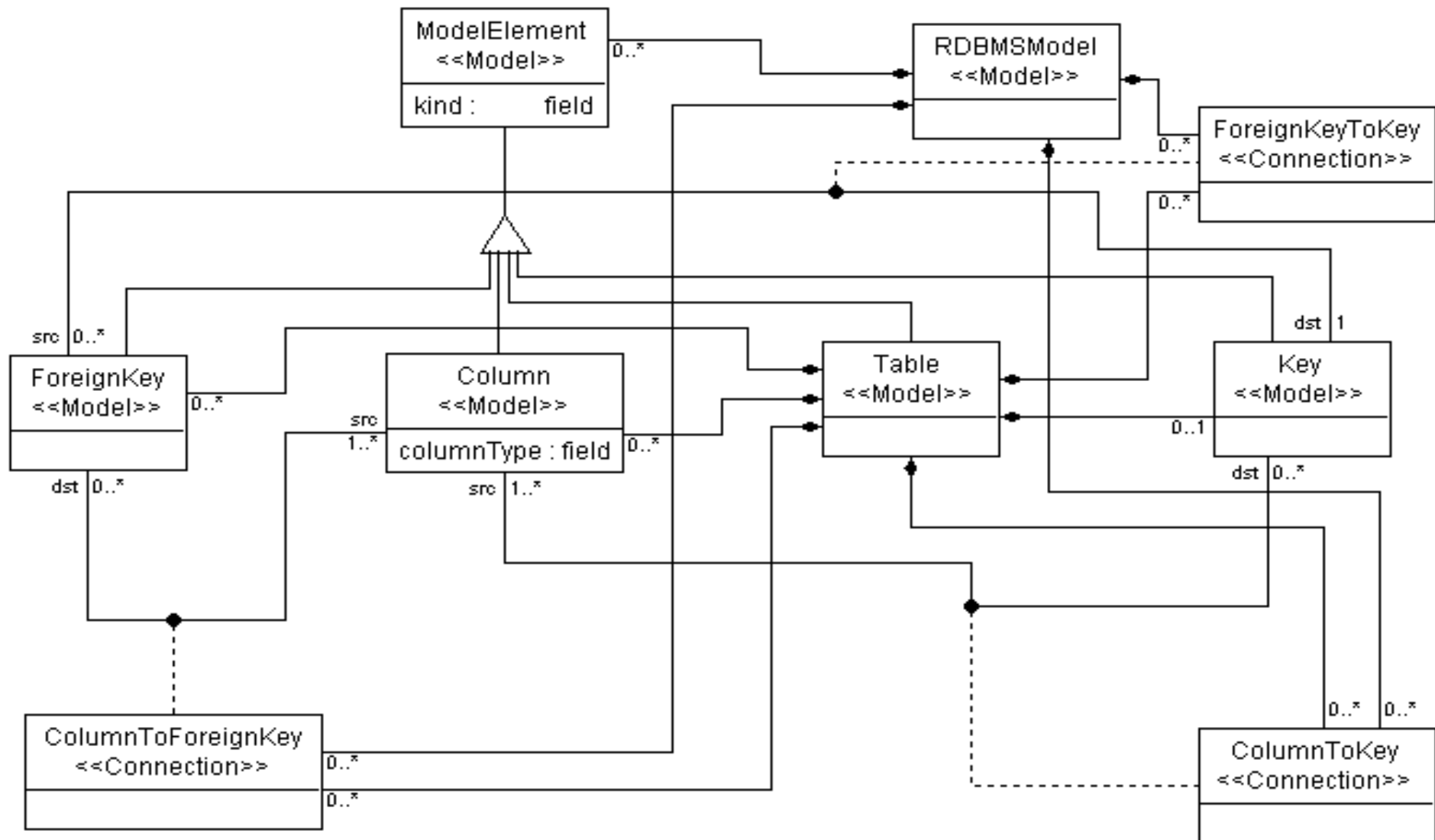
AttrxFool(a:Attribute, cs:Set(Column), prefix:String) extends AttrxCol



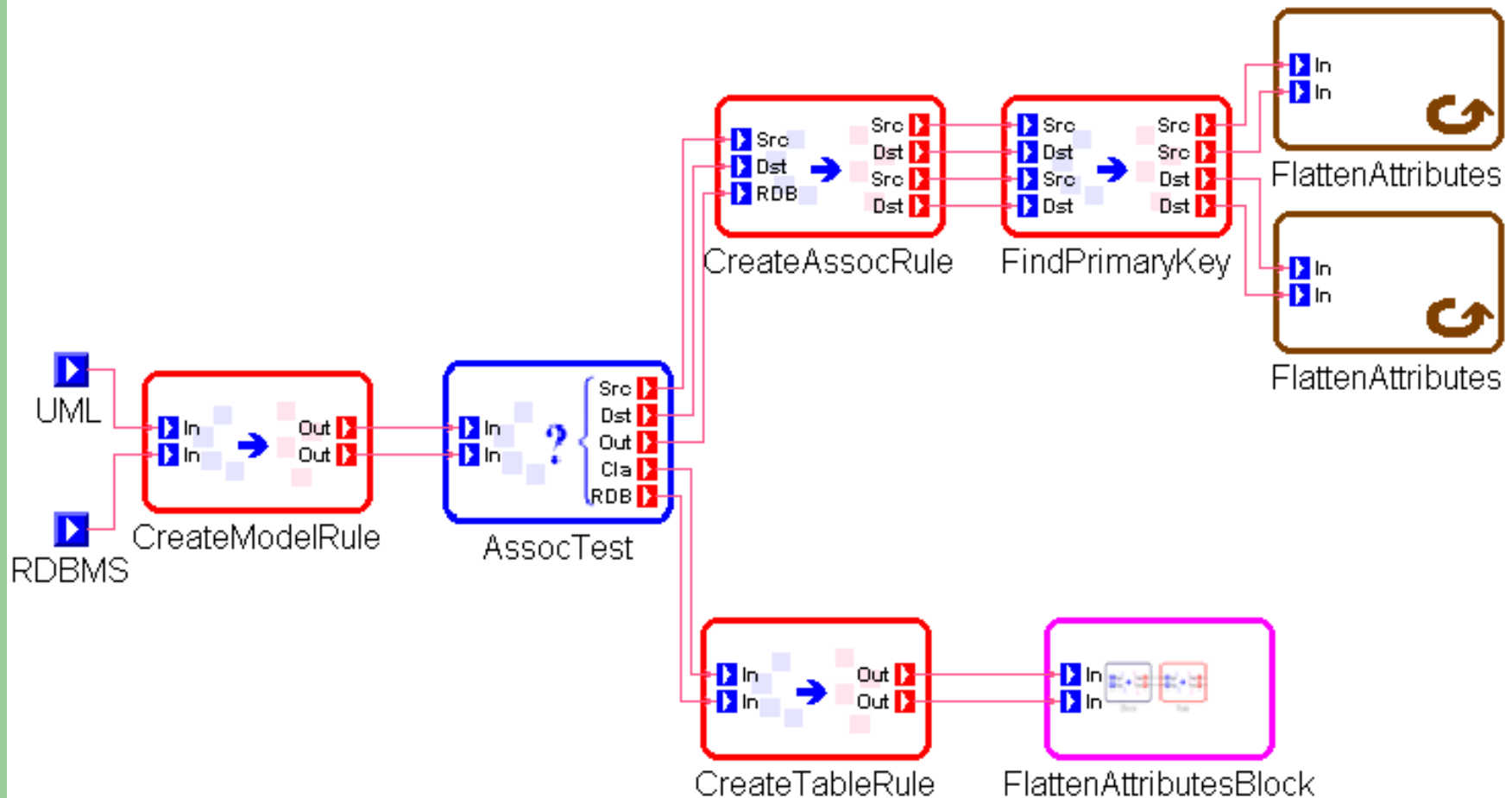
MetaGME UML Model



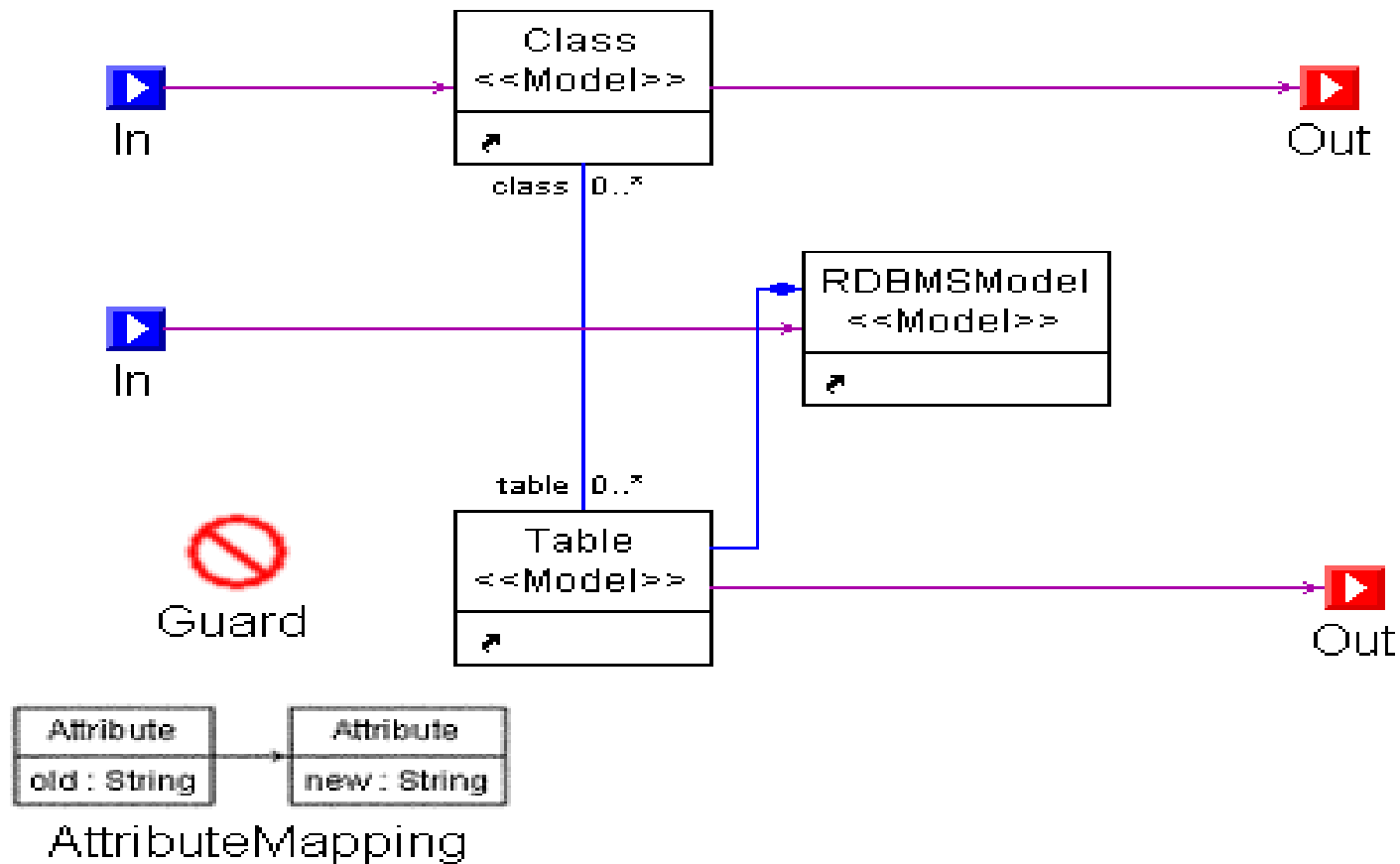
MetaGME RDBMS Model



Transformations



Transformations



Transformations

- Guard Code:

```
cint_string attrib_type(20);  
Class.GetStrVal("kind", attrib_type);  
if (attrib_type == "persistent") return true;  
else return false;
```

Sample UML Input

The screenshot displays a UML modeling application window titled "UMLModel - UMLModel5 - [Employee - /UMLModel5/NewUMLModel/]". The main workspace contains several colored boxes representing UML elements:

- Integer** (brown box) labeled *EmployeeNumber*
- String** (purple box) labeled *FirstName*
- String** (purple box) labeled *LastName*
- Integer** (brown box) labeled *PhoneNumber*
- Integer** (brown box) labeled *Address*
- Class** (blue box) labeled *CoWorker*

At the bottom left, an "Aspect" palette shows three selected elements: **Class**, **Integer**, and **String**.

At the bottom right, a property editor for the *EmployeeNumber* element is open, showing the "Attributes" tab with the following details:

| Attributes | Preferences | Properties |
|------------|-------------|------------|
| kind | | primary |

On the right side, a "Components" tree view shows the model structure:

- UMLModel5
 - NewUMLModel
 - Employee
 - Address
 - CoWorker
 - Address
 - EmployeeNumber
 - FirstName
 - LastName
 - PhoneNumber
 - EmployeeNumber
 - FirstName
 - LastName
 - PhoneNumber
 - Manager
 - Address
 - EmployeeNumber
 - FirstName
 - LastName
 - PhoneNumber

Output RDBMS Transformation

RDBMSModel - RDBMSModel5 - [t_Manager - //NewUMLModel/]

File Edit View Window Help

Name: t_Manager Table Aspect: Aspect Base: N/A

Aggregate Inheritance Meta

t_Manager

- RDBMSModel5
 - NewUMLModel
 - t_Employee
 - Address
 - CoWorker
 - CoWorker_Address
 - CoWorker_EmployeeNumber
 - CoWorker_FirstName
 - CoWorker_LastName
 - CoWorker_PhoneNumber
 - EmployeeNumber
 - EmployeeNumber
 - FirstName
 - LastName
 - PhoneNumber
 - foreign_EmployeeNumber
 - t_Manager
 - t_Manager
 - Address
 - EmployeeNumber
 - EmployeeNumber
 - FirstName
 - LastName
 - PhoneNumber

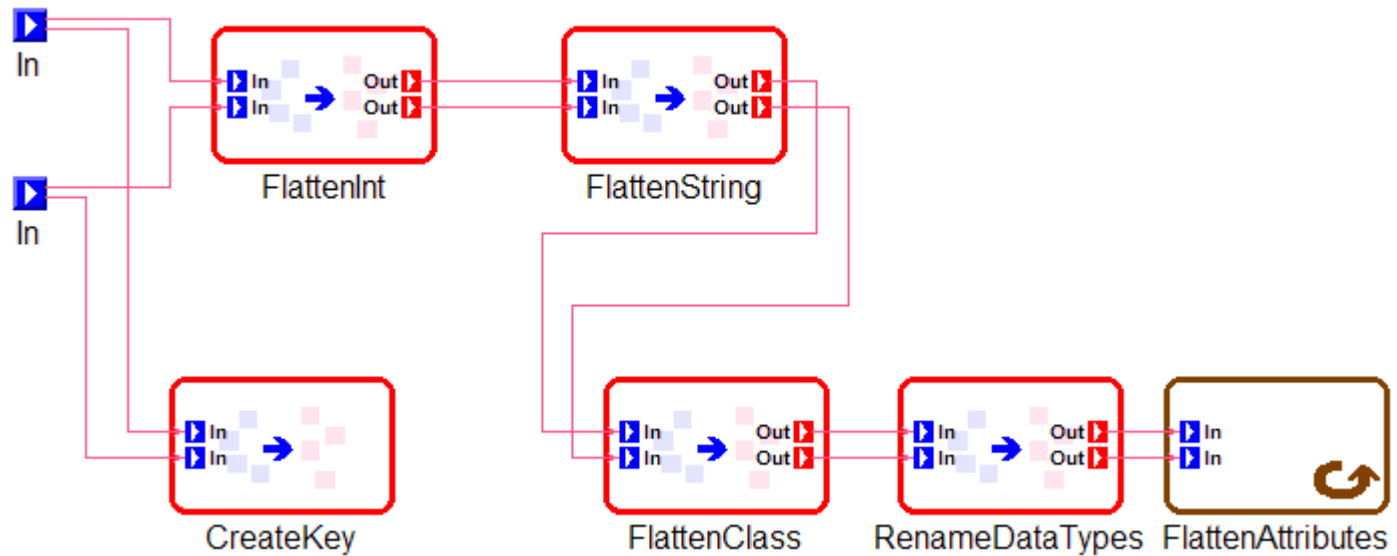
Column ForeignKey Key

EmployeeNumber for Kind

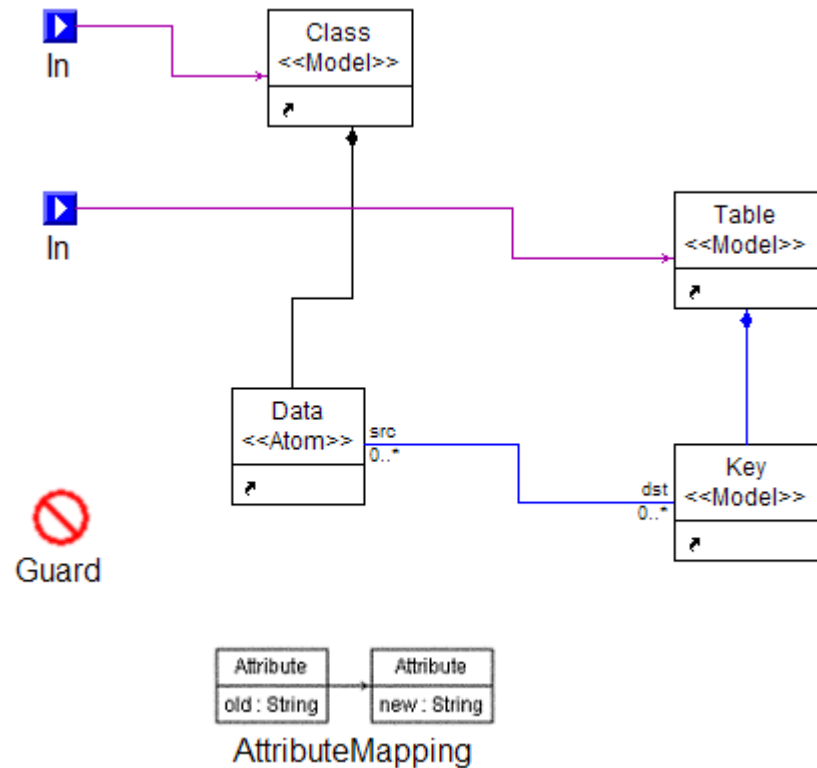
| Attributes | Preferences | Properties |
|------------|-------------|------------|
| kind | | primary |
| columnType | | NUMBER |

Ready EDIT 100% RDBMSModel 09:36 PM

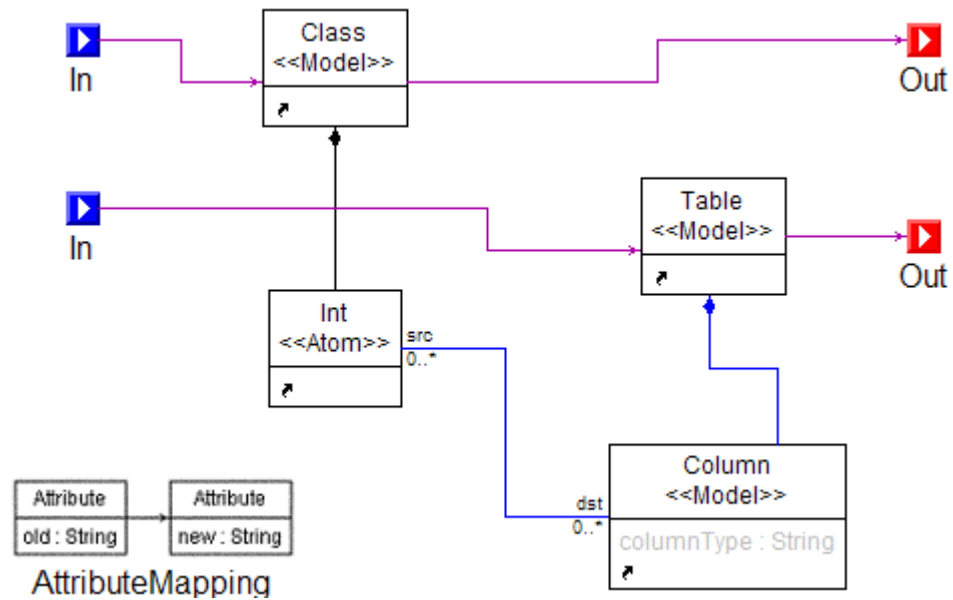
Flatten Attributes Rule



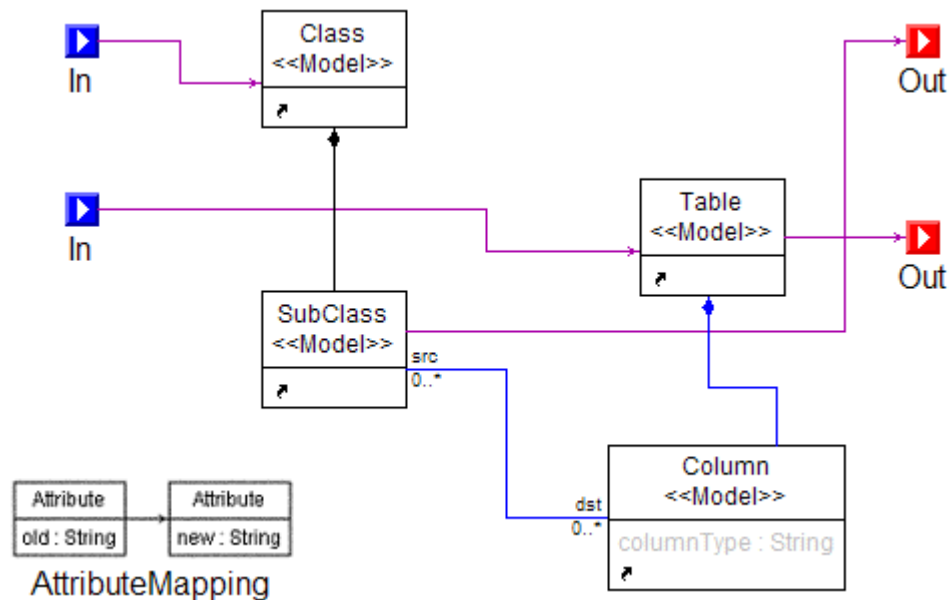
Flatten Attributes (CreateKey)



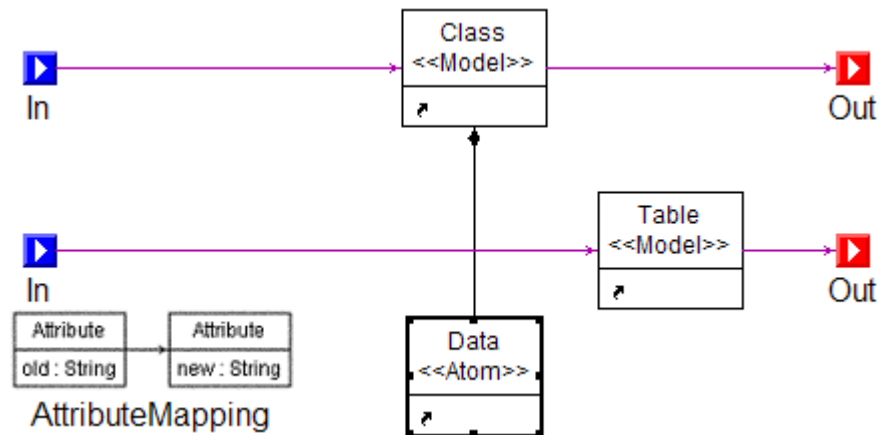
Flatten Attributes (FlattenString)



Flatten Attributes (FlattenClass)



Flatten Attributes (RenameDataTypes)



Analysis of Software - Cons

- Scoping of variables
 - No way of making globally accessible variables
 - Can speed up implementation and logic
- Unable to determine object type from OCL
- Complicated to pattern match attributes with objects
- Use of GME as meta language rather than standardized MOF
- Extra time incurred through use of GME
- Set semantics logic
- No Negative pattern matching operator

Analysis of Software - Pros

- Simple and easy to use
- Very intuitive transformation language
 - Can start with simple
 - Can become quite complex
- Transform models adhere to meta model specifications

Suggestions

- Modeling language for OCL constraints
 - Implementation of OCL had all associated problems of coding (not MDA)
- Improve documentation on non-pattern objects

References

- UML and RDBMS models from Simon 😊
- A. Agrawal, G. Karsai, and A. Ledeczi, “An End-to-End Domain-Driven Software Development Framework”
- A. Agrawal, Z. Kalmar, G. Karsai, F. Shi, A. Vizhanyo, “GReAT User Manual”
- A. Ledeczi, M. Maroti, A. Bakay, and G. Karsai, “The Generic Modeling Environment”

Thanks

- Professor Czarnecki
- Simon
- Aditya Agrawal