1. ATL Transformation Example

1.1. Example: Class → Relational

The Class to Relational example describes the simplified transformation of a class schema model to a relational database model. It has been adapted from a paper [1] by the DSTC.

1.1.1. Metamodels

The Class metamodel (see Figure 1 Class) consists of classes having a name which they inherit from the abstract class NamedElts. The principal class is the class Class, which contains a set of attributes of the type Attribute and has the super references pointing to superclasses for modelling inheritance trees. The class DataType models primitive data types. Class and DataType inherit from Classifier which serves to declare the type of Attributes. Attributes can be multivalued, which has an important impact on the transformation.

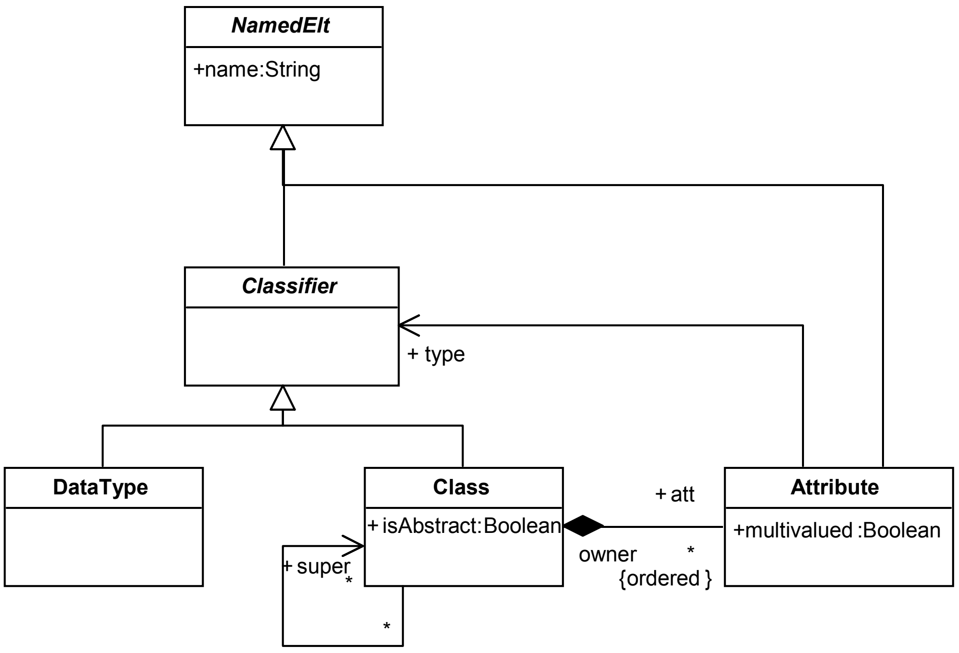

Figure 1 Class Metamodel

The Relational metamodel (see Figure 2) consists of classes having a name which they inherit from the abstract class Named. The principal class Table contains a set of Columns and has a reference to its keys. The class Column has the references owner and keyOf pointing to the Table it belongs to and of which it is part of the key (in case it is a key). Furthermore, Column has a reference to Type.

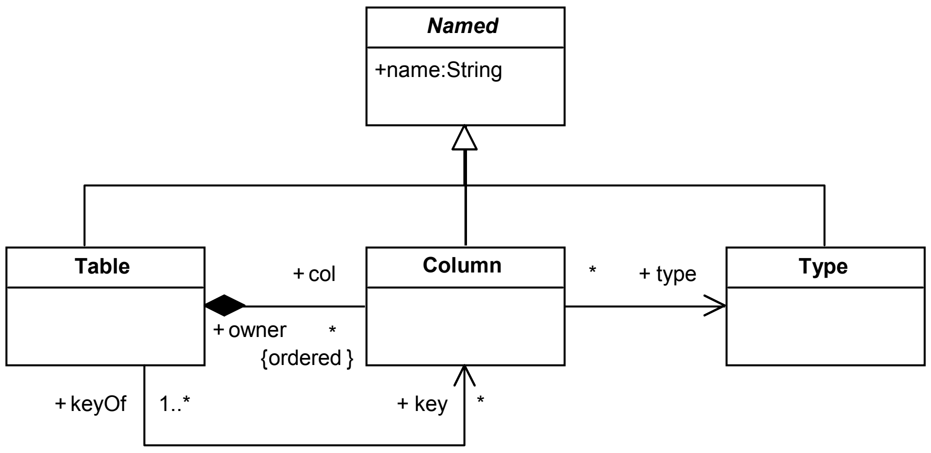

Figure 2 Relational Metamodel

1.1.2. Rules Specification

These are the rules to transform a class model to a relational model:

- For each Class instance, a Table instance has to be created.
    - Their names have to correspond.
    - The col reference set has to contain all Columns that have been created for single-valued attributes and also the key described in the following.
    - The key reference set has to contain a pointer to the key described in the following.
    - An Attribute instance has to be created as key
        - Its name has to be set to 'objectId'
        - Its type reference has to reference a Type with the name Integer which - if not yet existing - has to be created.
- For each DataType instance, a Type instance has to be created.
    - Their names have to correspond.
- For each single-valued Attribute instance of the type DataType, a Column instance has to be created.
    - Their names and their types have to correspond.
- For each multi-valued Attribute instance of the type DataType, a Table instance has to be created.
    - The Table's name is the name of the Attribute's Class concatenated with an underscore and the name of the Attribute.
    - The col reference set has to reference the two Columns described in the following.
    - An identifier Column instance has to be created.

    - Its name has to be set to the Attribute's class name concatenated with 'Id'
    - Its type reference has to reference a Type with the name Integer which - if not yet existing - has to be created.
  - A Column instance has to be created to contain the values of the Attribute.
    - The name and their types have to correspond.
- For each single-valued Attribute of the type Class, a new Column has to be created.
  - Its name has to be set to the attribute's name concatenated with 'id'.
  - Its type reference has to reference a Type with the name Integer which - if not yet existing - has to be created.
- For each multi-valued Attribute of the type Class, a new Table has to be created.
  - The Table's name is the name of the Attribute's Class concatenated with an underscore and the name of the Attribute.
  - The col reference set has to reference the two Columns described in the following.
  - An identifier Column instance has to be created.
    - Its name has to be set to the Attribute's class name concatenated with 'Id'.
    - Its type reference has to reference a Type with the name Integer which - if not yet existing - has to be created.
  - A foreign key Column instance has to be created.
    - Its name has to be set to the Attribute's name concatenated with 'Id'.
    - Its type reference has to reference a Type with the name Integer which - if not yet existing - has to be created.

Simplifications of this rule description:

- It does not take inheritance into account.
- It does not take the classifier isAbstract into account.

1.1.3. ATL Code

This ATL code for the transformation of a UML to Java consists of several functions and rules which are not yet fully implemented. Among the functions, it is important to mention getExtendedName which recursively explores the namespace to concatenate a full path name. Concerning the rules, there are remarks necessary respective the rule O2M. This rule shows how to access sets via OCL expressions. For simplicity of implementation, the (return) type of a Java Method is the first parameter of an UML Operation.

```
module Class2Relational;
create OUT : Relational from IN : Class;
```

```
uses strings;


-- inheritance is not supported yet

-- issue: choose an object-id Type (Integer, String?).
-- We choose Integer here,
-- assuming this type is defined in the source model.
helper def: objectIdType : Relational!Type =
	Class!DataType.allInstances()
		->select(e | e.name = 'Integer')->first();

rule Class2Table {
	from
		c : Class!Class
	to
		out : Relational!Table (
			name <- c.name,
		-- Columns are generated from Attributes in another rule not
		-- explicitly called here !
			col <- Sequence {key}->
					union(c.attr->select(e | not e.multiValued)),
			key <- Set {key}
		),
		key : Relational!Column (
			name <- 'objectId',
			type <- thisModule.objectIdType
		)
}

rule DataType2Type {
	from
		dt : Class!DataType
	to
		out : Relational!Type (
			name <- dt.name
		)
}

rule SingleValuedDataTypeAttribute2Column {
	from
		a : Class!Attribute (
			a.type.oclIsKindOf(Class!DataType) and not a.multiValued
		)
	to
		out : Relational!Column (
			name <- a.name,
			type <- a.type
-- explicit use of implicit tracking links
-- (first expected syntax, then present actual syntax)
--			owner <- [Class2Type.key]a.owner
--			owner <- thisModule.resolveTemp(a.owner, 'key')
		)
}
```

```
rule MultiValuedDataTypeAttribute2Column {
	from
		a : Class!Attribute (
			a.type.oclIsKindOf(Class!DataType) and a.multiValued
		)
	to
		out : Relational!Table (
			name <- a.owner.name + '_' + a.name,
			col <- Sequence {id, value}
		),
		id : Relational!Column (
			name <- a.owner.name.firstToLower() + 'Id',
			type <- thisModule.objectIdType
		),
		value : Relational!Column (
			name <- a.name,
			type <- a.type
		)
}

rule ClassAttribute2Column {
	from
		a : Class!Attribute (
			a.type.oclIsKindOf(Class!Class) and not a.multiValued
		)
	to
		out : Relational!Column (
			name <- a.name + 'Id',
			type <- thisModule.objectIdType
		)
}

rule MultiValuedClassAttribute2Column {
	from
		a : Class!Attribute (
			a.type.oclIsKindOf(Class!Class) and a.multiValued
		)
	to
		out : Relational!Table (
			name <- a.owner.name + '_' + a.name,
			col <- Sequence {id, foreignKey}
		),
		id : Relational!Column (
			name <- a.owner.name.firstToLower() + 'Id',
			type <- thisModule.objectIdType
		),
		foreignKey : Relational!Column (
			name <- a.name + 'Id',
			type <- thisModule.objectIdType
		)
}
```

1.1.4. Typical Test Example

The test example that will be shown in the following has been implemented using KM3. It concerns the package sample with the classes Family and the class Person that will be transformed to the corresponding relational model.

The model of the classes Family and Person:

```
package Sample {

	datatype String;
	datatype Integer;

	class Family {
		attribute name : String;
		attribute members[*] : Person;
	}

	class Person {
		attribute firstName : String;
		attribute closestFriend : Person;
		attribute emailAddresses[*] : String;
	}
}
```

is transformed to the following model of tables and columns:

```
	table Person {
		primary column objectId : Integer;
		column firstName : String;
		column closestFriendId : Integer;
	}

	table Family_members {
		column familyId : Integer;
		column membersId : Integer;
	}

	table Person_emailAddresses {
		column personId : Integer;
		column emailAddresses : String;
	}

	table Family {
		primary column objectId : Integer;
		column name : String;
	}
```

References

[1] Lawley, M., Duddy, K., Gerber, A., Raymond, K. Language Features for Re-Use and Maintainability of MDA Transformations. OOPSLA & GPCE Workshop, 2004