Transformation Rules Based on Meta-modeling

Lemesle Richard
Laboratoire de Recherche
en Sciences de Gestion,
Université de Nantes
2, rue de la Houssinière
BP 92208 44322 Nantes cedex 03, France
email : rlemesle@sodifrance.fr

Abstract

In this paper, we describe how meta-modeling techniques can be used to define rules for model transformation. A meta-model describes the ontology (the semantics) of a model, and the transformation rules from a model to an other can be described using the terms defined in both meta-models of these models. The example presented in this paper is a transformation from an object model, described using the UML semantics [UML1997], to a new one described using a relational paradigm. Both models are represented in a formalism based on semantic nets and called sNets[Bézivin1995]. This work can be compared to graph rewriting techniques as used in PROGRES (Programmed Graph Replacement Systems[Schürr1997]).

1. Introduction

More and more people are working on meta-modeling techniques. Meta-modeling consists in determining concepts and relationships used in a particular model. For example, an UML meta-model describes concepts such as classes, generalizations, associations, etc. while a Petri Net meta-model describes concepts such as places and transitions, and a Relational meta-model describes concepts such as tables, columns and relations.
Meta-modeling has many advantages. First, people using the same meta-model in order to represent their data would be able to share their works easily and to understand each other. They speak the same language. A meta-model can be used to define a modeling language and some associated modeling tools. Meta-models can also facilitate access, use, and sharing of data by describing the content, structure, and semantics of data residing in information systems, databases, or files.

In this paper, we address an other advantage of meta-modeling. If meta-models are formally defined, then it will be possible to define transformation rules in order to transform a model to an other one following a new semantics. We will take the example of a set of rules based on UML and relational semantics that will transform an object model to a relational model. Our aim is not to say that this transformation is obvious because it's far from being true. One may want to define a table for each classes, while an other may want to define a table only for concrete classes. There are a lot of possibilities. Our aim is to show that formal meta-models can be used to formalize such a transformation.

2. Meta-model representation

First, we present the formalism used to describe both models and meta-models. Let us begin with the following example :

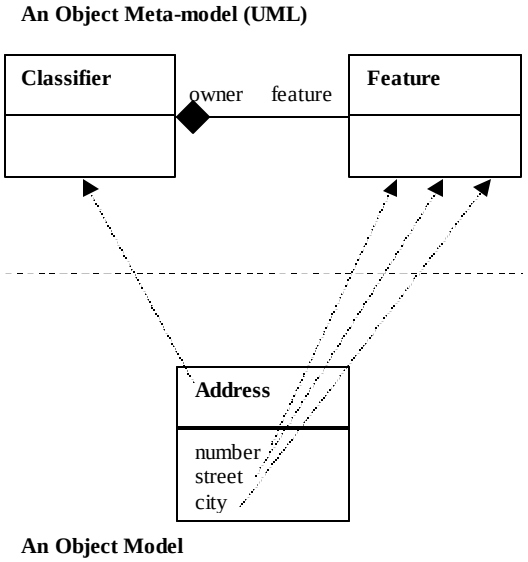

Figure 1. An object model and its meta-model

In this figure, we have represented a part of a model using the UML notation and a part of its meta-model using the same notation.

Dashed arrows represents "**is a**" relationships between the entities of the model and those of the meta-model. Address **is a** classifier while number, street and city are features. This "is a" relationship is also called *meta* relationship. In meta-modeling, each entity of a model must have such link to an entity of the meta-model. If the formalism doesn't allow an explicit representation of this link, then, it exists implicitly.

In the same way, we can define a relational meta-model to represent relational models :

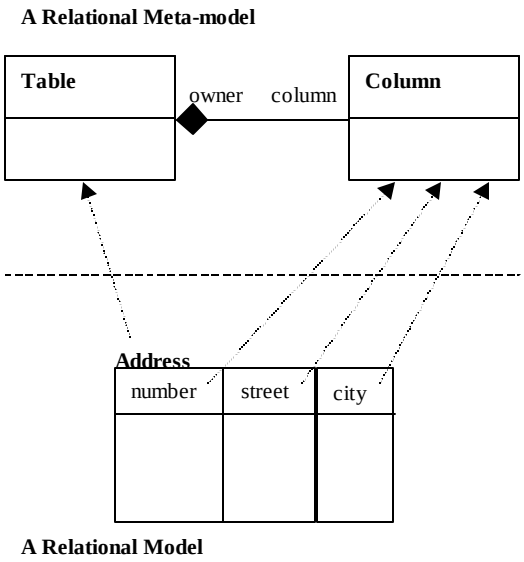

Figure 2. A relational model and its meta-model

Using such a meta-model, concepts represented are tables and columns. There is also *meta* relationships between the entities of the model and those of the meta-model.

Many tools use such meta-models in order to allow queries on models. For example, an OOA&D tool allows the user to access the model via the concept of scripts, and such a script will access the model via the meta-model implemented in the tool.
An other example is a generic application which access relational databases. Such application must query the database to know which are the tables. And then, query these *Meta-Table* or *Table Description* objects in order to know which are the columns, and so on.

Our aim is to show that meta-modeling can also be used for model transformations. But some preliminary work has to be done. In a first time, we will represent all this information (models and meta-models) in a common formalism called sNets which bring us the formal aspect we need.

3. The sNet formalism

This formalism is based on semantic networks. Within this formalism, we only manipulate nodes and links representing concepts and relationships. Every sNet node has at least three links (thereafter, links will be represented in italic font) :

- a link called *name* to his name,
- a link called *meta* to the node describing its type (this is an explicit representation of the "**is a**" relationship) and
- a link called *partOf* to a node representing his context (such a context is called universe in sNets and is used to address modularity concerns).

The Address class represented in Figure 1 can be represented in the sNet formalism using the following notation:

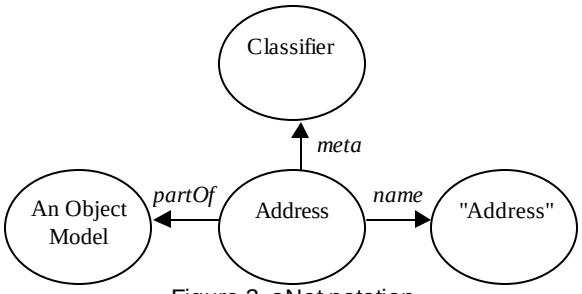

Figure 3. sNet notation

In this figure, there are four entities (or nodes) :

- the Address node which is our Address class representation,
- the "Address" node which is its name,
- the Classifier node which describes its type and
- the node called "An Object Model" which represents its context (the object model).

So, in the sNet formalism, each concept may be represented by a node or a graph (a set of nodes with their associated links). But each of these nodes must be an sNet node and then, must have the *name*, *meta* and *partOf* links.

In order to simplify this notation, the following one can also be used :

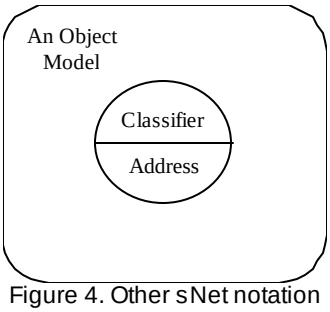

Figure 4. Other sNet notation

Models, in the sNets formalism, are represented by nodes of type **Universe** (thereafter, nodes of type **Node** will be represented in bold font). And meta-models are represented by nodes of type **SemanticUniverse** which is a subtype of **Universe**. So, models and meta-models are always represented by nodes of type **Universe**, and then, *partOf* links always lead to a node of this type (or a subtype of **Universe** such as **SemanticUniverse**).

In a semantic Universe, we find the description of a meta-model. Such a description is based on nodes of type **Node** which describe the concepts defined in the meta-model, and nodes of type **Link** which describe the relationships between these concepts.

So a concept type is always represented by a node of type **Node**, and then, *meta* links always lead to a node of this type.

In the following figure, we have represented a **Universe** node named "An Object Model" which is our model and a **SemanticUniverse** node named "An Object Meta-model" which is our meta-model. In the meta-model, we have represented a node of type **Node** named "Classifier" which is our Classifier meta-entity, and in the model we have represented a node of type **Classifier** named "Address" which is our Classifier entity. The "is a" relationships between the entities of the model and those of the meta-model are also represented here by the *meta* link between the node "Address" and the node "Classifier".

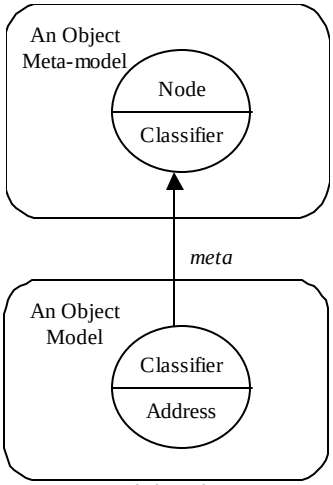

Figure 5. sNet model and sNet meta-model

A meta-model is also a model with a meta-model. In the sNet formalism, this meta-meta-model is defined in a universe called "semantic" which is the sNet's core. This universe defines the concepts of **Node** and **Link**. So we have also the following scheme :

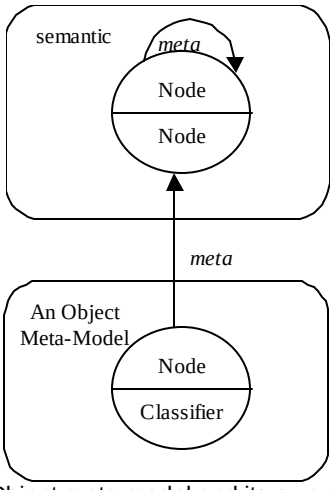

Figure 6. Object meta-model and its own meta-meta-model

The "semantic" universe is its own meta-model. It is composed of nodes of type **Node** and **Link** in order to describe **Node** and **Link** types (as it is shown, the node named "Node" has a *meta* link to itself).

Relationships between nodes are represented by labelled and directed links. Such relationships are defined by meta-relationships entities in the meta-model. In the sNet formalism, meta-relationships are represented by nodes of type **Link**. An sNet meta-model itself is defined using **Node** and **Link** nodes and relationships between these nodes. Principal relationships are *inComing* and *outGoing* links :

- an *outGoing* link between a node "X" of type **Node** and a node "l" of type **Link** means that a node of type **X** may be the source of a link *l*, and
- an *inComing* link between a node "l" of type **Link** and a node Y of type **Node** means that a node of type **Y** may be the target of a link *l*.

These relationships *inComing* and *outGoing* are defined by nodes of type **Link** in the "semantic" universe.

For example, the meta-model described using the UML formalism in Figure 1 can be described by the following scheme in sNets :

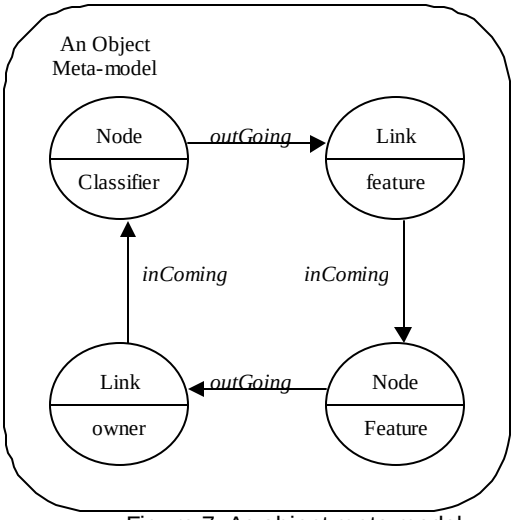

Figure 7. An object meta-model

This means :

- An object model can only contains nodes of type **Classifier** and **Feature**,
- nodes of type **Classifier** can only have *feature* links to nodes of type **Feature**,
- nodes of type **Feature** can only have *owner* links to nodes of type **Classifier**.

Using such a semantic (or ontology or meta-model), we can define the object model described in the Figure 1.

In this figure, there is a classifier called "Address" and three features called "number", "city" and "street". These features are the Address's features so there is also *owner* links between the features and the "Address" classifier and *feature* links between the "Address" classifier and the features.

In the sNet formalism, we also represent the relation between a model and its meta-model. This relation is represented by a link called "*sem*" between the model and its meta-model.

So, the following figure represents the object model and its meta-model already described in figure 1 :

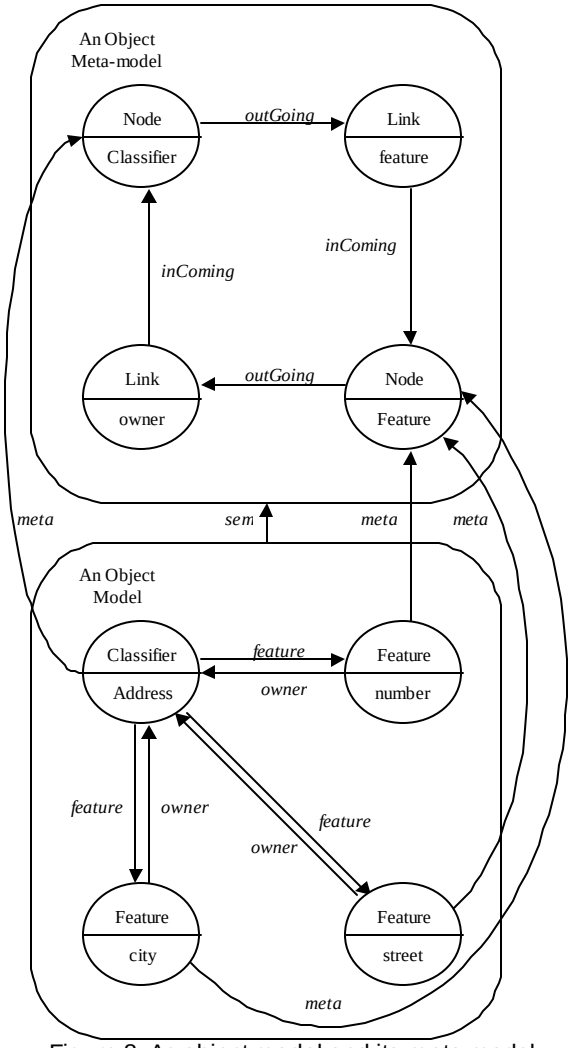

Figure 8. An object model and its meta-model

Using sNets, all meta-models are based on the semantics defined in the "semantic" universe. So, we can define a relational meta-model which follows this semantic. And then, define relational models in universes which follow the semantics defined in the relation meta-model universe.
In order to make model transformation, we must be able to represent several models. And each of these models may follow its own semantics.

As described below, a sNet can contain several models and several meta-models :

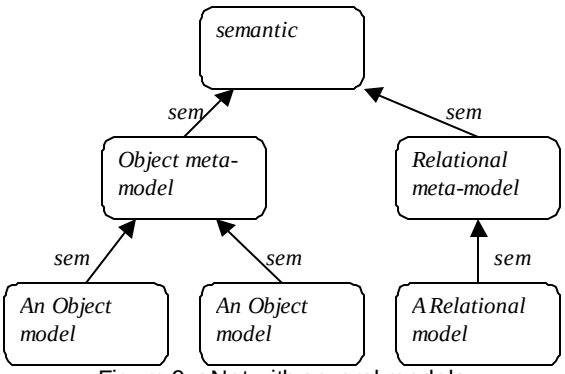

Figure 9. sNet with several models

Then, any model and meta-model represented using sNets may also be seen as a directed graph with labelled links. And any model transformation can be seen as a kind of graph rewriting process.

4. Transformation rules

Transformation rules can be defined using the semantics of the meta-models. The following scheme presents the example of transformation rules based on the object meta-model and the relational meta-model and defined to transform an object model to a relational one :

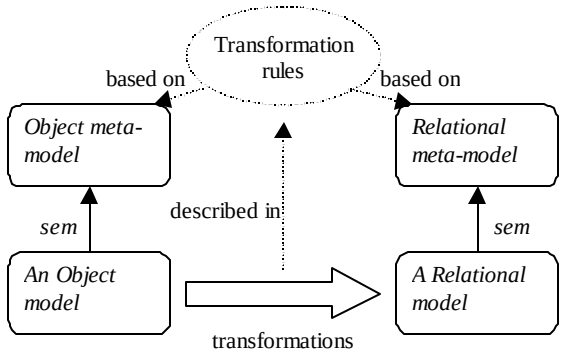

Figure 10. Transformation scheme

In such a transformation, we may want to state that a classifier becomes a table and a feature relationship between a classifier and a feature becomes a column relationship between a table and a column. Such transformation rules described in a textual form are clearly based on meta-models. And a formal description of meta-models allows a formal description of these transformations.

The transformation rules are currently defined using a text format. A transformation consists of the semantics of the source model, the semantics of the target model, and the set of transformation rules. Each rule is composed of a set of conditions, a set of conclusions and a set of variables. A condition may be a type condition on a variable or a link condition between two variables. A conclusion may be a node creation assigned to a variable or a link creation between two variables. Variables are identified by their names.

We have defined the following BNF grammar :

<RulesDesc> ::= <Header> **:** <Rules>
<Header> ::= <SourceSem> **->** <TargetSem>
<Rules> ::= <Rule> *
<Rule> ::= <Conditions> **->** <Conclusions> **;**
<Conditions> ::= <Condition> *
<Conclusions> ::= <Conclusion> *

4.1. Rule Conditions

A condition may be a type condition on a variable or a link condition between two variables.

A type condition has the following form :

<TypeName> **(** <Variable> **)**

This is true if and only if the node assigned to the variable called <Variable> has a *meta* link to the node of type **Node** called <TypeName> in the meta-models. This can be formalized by:

? v / meta(v, vt)

Where v is the variable and vt is a constant which represents the node of type **Node** named <TypeName> in the meta-models.

A link condition has the following form :

<linkName> **(** <SourceVariable>**,** <TargetVariable> **)**

This is true if and only if there is a link named <LinkName> between the node referenced by the variable called <SourceVariable> and the node referenced by the variable called <TargetVariable>. This can be formalized by :

? sv,tv / linkName(sv, tv)

Where sv and tv are respectively the variables called <SourceVariable> and <TargetVariable>.

Some examples of conditions :

- Classifier(class) which means :
  ? class / meta (class, Classifier)

- Table(t) which means :
  ? t / meta (t, Table)

- owner(a,c) which means :
  ? a,c / owner (a, c)

- name(c,"Address") which means :
  ? c / name(c,"Address")

In these examples, Classifier is a constant which represents the node of type **Node** named "Classifier" in the Object meta-model and Table is a constant which represents the node of type **Node** named "Table" in the Relational meta-model.

4.2. Rule Conclusions

A conclusion may be a node creation assigned to a variable or a link creation between two bound variables.

A node creation assigned to a variable has the following form :

<TypeName> **(** <Variable> **)**

This means that we have to create a new node in the target universe. This node is assigned to the variable called <Variable> and his type is <TypeName> (so, the node has a *meta* link to the node of type **Node** called <TypeName> in the target meta-model). This can be formalized by:

? v / meta(v, vt) partOf(v, tm)

where v is the variable, vt is a constant which represents the node of type **Node** named <TypeName> in the target meta-model and tm is the node which represents the target model.

A link creation between two variables has the following form :

<linkName> **(** <SourceVariable>**,** <TargetVariable> **)**

This means that we have to create a new link named <linkName> between the node referenced by the variable called <SourceVariable> and the node referenced by the variable called <TargetVariable>. This can be formalized by :

linkName(sv,tv)

where sv and tv are respectively the variables called <SourceVariable> and the variable called <TargetVariable>.

Some examples of conclusions :

- Classifier(class) which is the creation of a node of type **Classifier** assigned to the variable class in the target model (tm) and which means :

  ? class / meta(class,Classifier) partOf(class, tm)

- Table(t) which is the creation of a node of type **Table** assigned to the variable t in the target model (tm) and which means :

  ? t / meta(t, Table) partOf(t, tm)

- owner(a,c) which is the creation of a link *owner* between the node referenced by the variable a and the node referenced by the variable c and which only means :

  owner(a,c)

- name(c,"Address") is the creation of a link *name* between the node referenced by the variable c and the constant "Address" and which only means :

  name(c,"Address")

In these examples, Classifier is a constant which represents the node of type **Node** named "Classifier" in the Object meta-model.

4.3. Rule description

A rule is composed of a set of conditions and a set of conclusions and has the following form :

<Conditions> **->** <Conclusions> **;**

This means that if all conditions are verified, then all conclusions must be verified too (these conclusions must be applied to be verified). This can be formalized by :

<conditions> ? <conclusions>

An example of rule is the following one :

Classifier(c) -> Table (t);

which means :

? c / meta(c, Classifier) ? (? t / meta(t, Table) partOf(t, tm))

In this sentence, c and t are variables. Classifier is a constant which represents the node of type **Node**

named "Classifier" in the object meta-model. Table is a constant which represents the node of type **Node** named "Table" in the relational meta-model. And tm is a constant which represents the node of type **Universe** which is the target model. This rule means that for all node of type **Classifier** in the source model, there is node of type **Table** in the target model. And the application of the rule means the creation of a **Table** node in the target model for any **Classifier** node in the source model.

4.4. The transformation process

Then, when applying a set of rules, the transformation is done when no more rule can be applied. In the prototype, a rule cannot be applied more than once with the same set of bounded variables.

In this prototype, a set of rules is described in a text format which start with a header of the following form :

<source sem> -> <target sem> :

For example, a set of rules which transform an object model to a relational model is described in a text file starting with :

ObjectMetaModel -> RelationMetaModel :

This ensure that concepts used in rules belongs to these meta-models.

Then, a simple transformation which defines a table concept in a relational model for each classifier concept in an object model may be the following :

```
ObjectMetaModel -> RelationalMetaModel :

Classifier(c) -> Table(t);
```

In such a transformation, the created nodes of type **Table** are named "aTable" (default name for a node of type **X** is aX in sNet). If one wishes that the tables bear the same names as the classes, the previous rule just have to be replaced by the following one :

```
Classifier(c) name(c, n) -> Table(t) name(t, n);
```

The type of nodes and links created during the rule's application must be defined in the target meta-model. This ensure that the target model always follows the semantics defined in its meta-model. But sometimes we have to create temporary links used only during the transformation and removed when the transformation is done.

For example, suppose that a rule creates a **Table** node for each **Classifier** node and an other rule creates a **Column** node for each **Feature** node. Then, how can we create the *column* links between the tables and the columns ?

We have the following rules :

```
Classifier(c) name(c,n) -> Table(t) name(t,n);
Feature(f) name(f,n) -> Column(c) name(c,n);
```

And we want to write something like this :

```
Classifier(c) Feature(f) feature(c,f) -> column(t,col);
```

But we must state that t is the table which corresponds to c and col is the feature which corresponds to f.

A solution consists to create a temporary link between the created tables and their corresponding classifiers. And the same is done for features and columns.

The rules are replaced by the following ones :

```
Classifier(c) name(c,n) ->
        Table(t) name(t,n) _coref(t,c) ;

Feature(f) name(f,n) ->
        Column(c) name(c,n) _coref(c,f);
```

Temporary links are identified in our prototype by names starting with the underscore character. So, *_coref* link is a temporary link. A temporary link doesn't have to be defined in meta-models.

Then, the column link can be created by the following rule :

```
Classifier(c) Table(t) _coref(t,c) Feature(f) Column(col)
_coref(col,f) feature(c,f) ->

        column(t,col);
```

These temporary links must be removed when the transformation is done because they are not defined in the meta-models. They only have sense during the transformation. And then, when the transformation is done, the remaining links are only those defined in the meta-models.

Then, the following example shows a more complete set of transformation rules that can be defined :

```
ObjectMetaModel -> RelationalMetaModel :

Classifier(c) name(c,n) ->

        Table(t) name(t,n) _coref(c,t) Column(id)
        name(id,"id") column(t,id);

Classifier(c) generalization(c,g) supertype(g,sc) ->

        _super(c,sc);

Classifier(c) Classifier(sc1) Classifier(sc2)
_super(c,sc1) _super(sc1,sc2) ->

        _super(c,sc2);

Classifier(c) feature(c,a) ->

        _attribute(c,a);

Classifier(c) _super(c,sc) _attribute(sc,a) ->

        _attribute(c,a);

Classifier(c) _coref(c,t) _attribute(c,a) name(a,n) ->

        Column(col) name(col,n) column(t,col);
```

The first rule creates a table with a column named "id" for each classifier.
The second and the third creates a temporary link *_super* in order to access directly all the superclasses for a classifier. This is used for the next temporary link.
The fourth and fifth creates a temporary link *_attribute* between classifiers and all inherited attributes.
And the last rule creates the *column* links between the tables and the columns.

So, we have demonstrated that formal transformation rules can benefit from formal meta-modeling techniques. Then, our ongoing work is to use sNet in order to represent such rules.

5. sNet representation for transformation rules

A set of transformation rules can be described in a universe. In this universe, we represent concepts such as Rules, Nodes (variables) and Links. For example, we may want to represent the following rule in a universe called "A Rules Set" :

Classifier(c) -> Table(t) _coref(c,t)

Then, such a universe can be defined as described in the following scheme :

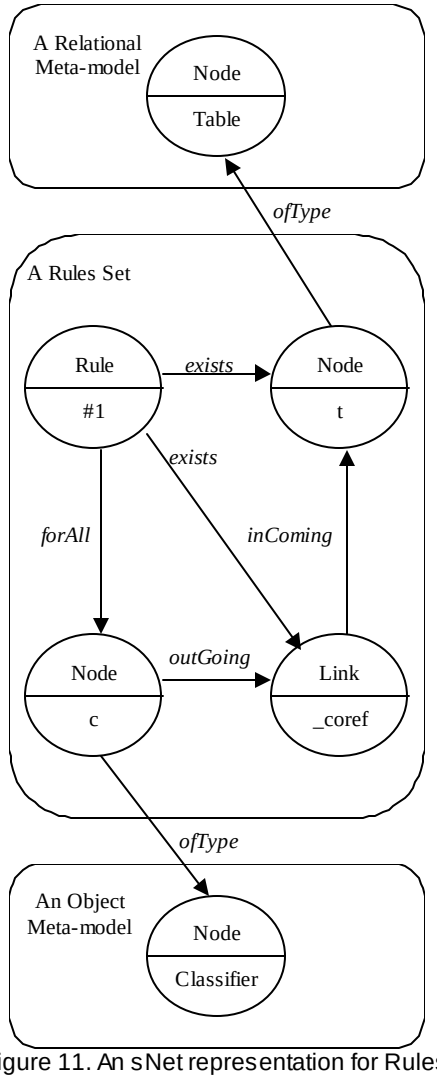

Figure 11. An sNet representation for Rules

In order to do this, we just have to define the meta-model of such a universe. Then, this universe is closely related to its based meta-models by links called *ofType* there. And the same formalism is used to represent models, meta-models and transformation rules.

6. Conclusion

Meta-modeling techniques can bring much more than what is currently done. Nowadays, modeling tools are more and more based on well defined meta-models, but cannot handle more than one meta-model at a time. And more and more processes will have to handle several models based on several meta-models. This paper presents such a process. Our aim is to show that a formalism like sNets makes this possible.

We are also working on process modeling. And if products are represented by models (using UML semantics for example), then processes can be represented by transformation rules.

7. References

[Banach1997] Banach, R. & Papadopoulos; G.A. A study of two graph rewriting formalisms: Interaction Nets and MONSTR, Journal of Programming Languages, vol 05, issue 01, p. 201-231.

[Bauderon1987] Bauderon, M. & Courcelle B. Graph expressions and graph rewritings, Mathematical Systems Theory, vol 20, p. 83-127.

[Bézivin1997] Bézivin, J. & Lemesle, R. Ontology-based Layered Semantics for Precise OA&D Modeling, ECOOP'97 Workshop on Precise Semantics for Object-Oriented Techniques.

[Bézivin1995] Bézivin, J. Lanneluc, J. & Lemesle, R. sNets: The Core Formalism for an Object-Oriented CASE Tool, Object-Oriented Technology for Database and Software Systems, V.S. Alagar & R. Missaoui ed., World Scientific Publishers, 1995, p. 224-239.

[Blostein1997] Blostein, D. & Schürr, A. Computing with Graphs and Graph Rewriting. Technical Report AIB 97-8, Fachgruppe Informatik, RWTH Aachen, Germany.

[Booch1995] Booch, G. & Rumbaugh, J. Unified Method for Object-Oriented Development Documentation Set Version 0.8, Rational Software Corporation, (Oct. 1995).

[Courcelle1990] Courcelle, B. Graph rewriting : An algebraic and logic approach, Handbook of Theorical Computer Science, vol B, p. 193-242.

[Ernst1997] Ernst, J. Introduction to CDIF, Integrated Systems, Inc., (Jan. 1997).

[Habel1992] Habel, A. Hyperedge replacement : grammars and languages. LNCS, vol 643.

[Kilov1994] Kilov, H. James Ross. Information Modeling: an Object-oriented Approach. Prentice-Hall, Englewood Cliffs, NJ, (1994).

[Lamb1996] Lamb, D.A. Editor, Studies of Software Design, Lecture Notes in Computer Science, Volume 1078, Springer Verlag, (1996).

[Odell1995] Odell, J. Meta-modeling, (1995).

[Schürr1997] Schûrr, A. Programmed Graph Replacement Systems, G. Rozenberg, Handbook on Graph Grammars: Foundations, Vol. 1, Singapure: World Scientific (1997), p 479-546

[UML1997] UML Semantics, version 1.1, http://www.rational.com/uml (septembre 1997).