

MTRANS: A general framework, based on XSLT, for model transformations.

Mikaël Peltier^(*), Jean Bézivin^(#), Gabriel Guillaume^(*)

mikael.peltier@rd.francetelecom.fr
gabriel.guillaume@francetelecom.fr
Jean.Bezivin@sciences.univ-nantes.fr

^(*) FT R&D DIH/HDM/DEI,
4, rue du Clos Courtel, BP 59
35000 Cesson-Sévigné, France

^(#) Université de Nantes, LRS
2, rue de la Houssinière, B.P. 92208
44322 Nantes cedex 3, France

Abstract

The MTRANS project aims to supply a general framework for expressing model transformations. We want this framework to be the most general possible. To achieve this, MTRANS is based on a meta-modeling approach (a meta-model is used to define the semantics of each model). The MTRANS framework is supplying a language and an environment to write models transformations. The language is composed by a fixed instruction set (conditional, loop, etc.) plus a part depending on the particular meta-models used. Thanks to, the meta-modeling approach, MTRANS can be used to transform all MOF-compliant models.

Keys-words

Transformation, XML, MOF, XMI, Meta-model, XSLT, UML

Introduction

Within the software engineering community, XML [*XML*] is becoming the “lingua franca” for data communication between applications. Consequently, XML is the common and standard projection for models of any kinds. The XMI [*XMI*] specification is used to represent all models and meta-models, which are MOF compliant. The MOF specification [*MOF*] defines a meta-meta-model, which is used in turn to define meta-models, such as a UML meta-model, an EJB meta-model or a Data Warehouse meta-model. All these meta-models consequently share a common “semantics”. The precision of this semantics may be adapted since the OCL assertion language may be applied at the meta-model or at the meta-meta-model level. Consequently, the transformation between models of any kinds is easier because we have meta-meta-model used to define meta-models. Moreover, the physical representation of these models and meta-models are defined by the XMI specification.

XML comes with a set of specifications. One of these is XSLT [*XSLT*], which allows transforming a XML document into another format. This format can be HTML, XML, or anything else. Consequently, XSLT could be used to transform models, which are represented in the XMI formalism. In the next section, we will see that directly using XSLT is not the best way to express models transformations.

Models transformation can be useful for different reasons; we can use it to optimize models, to reorganize models [BP96, GR98], to reuse data between different models, but also to simplify the evolution of meta-models. For instance, if we have a meta-model with a specific concept, if this concept is divided into two concepts in a future version, all models of the previous version are unusable unless we have some tool to convert your models. A language to express models transformation describes how to transform element of the first model into elements of the second and generates automatically a system which can convert your old model toward models which are compliant with the new meta-model.

XSLT vs. MTRANS language

As we saw in introduction, it seems that XSLT can be used to express model transformation if models are represented by XMI. Nevertheless, It appears difficult to use directly XSLT on a real system for some reasons:

- Writing an XSLT program is long and painful. For instance, at France Telecom R&D, we use XSLT to transform a model into another and this program takes about 7000 lines and it is unreadable. One important problem with XSLT is its poor readability and the high cost of maintenance for associated programs
- Writing an XSLT program implies good skills in the MOF and XMI specification, because when using XPATH in XSLT, we must take into account the deep structure of models that depends on meta-models which are themselves highly dependent on MOF and XMI. Few people which are experts with MOF and XMI and we would welcome a more straightforward way of expressing these transformations.
- Executing an XSLT program is not user-friendly for models transformation. There are no error messages that depend on the application domain. For example, if we want to transform a concept that does not exist, the processor XSLT does not inform the user.
- Finally it is interesting to have the meta-models in line when generating code. This will allow the compiler to generate more clever patterns of XSLT code.

At France Telecom R&D, we are working on a framework to express model transformations. This framework must guide and help a user to write these transformations.

The MTRANS framework

The framework uses XSLT to transform models, but we develop an abstraction level above XSLT, which is more compact and easier to understand than XSLT. The architecture of the framework is depicted in Figure 1.

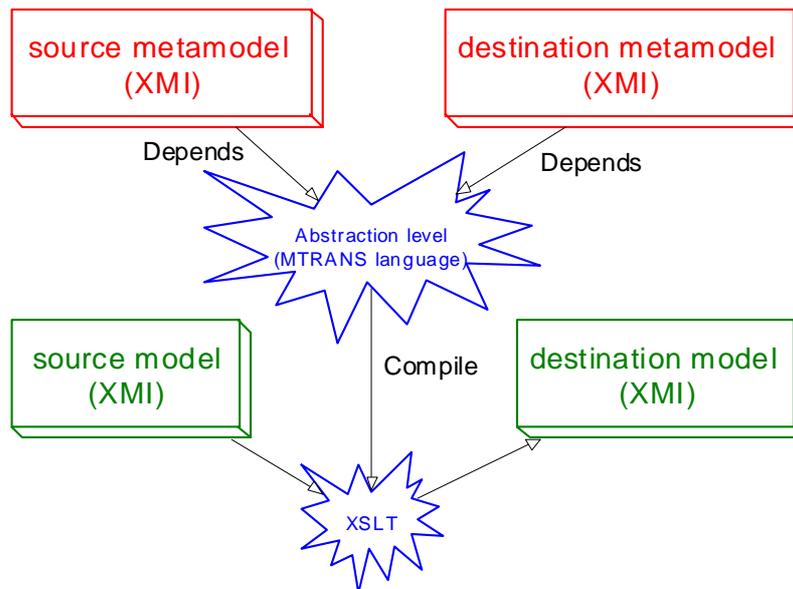


Figure1. Architecture of the MTRANS framework developed at France Telecom R&D

In this architecture, we see that the MTRANS language depends on the source meta-models and on the destination models (they can be different). This architecture is natural, because when we want to transform models, what we want is to transform concepts, properties or relations which appear in models. All of model characteristics are defined in the meta-model.

The MTRANS framework is composed by an editor of MTRANS program and by two browsers of meta-models, one for the source meta-model and another for the destination model. Those browsers are very important to create a user-friendly transformation framework. When we write a transformation, we want to transform concepts defined in a meta-model and we are using associations between those concepts to navigate inside a specific model. Consequently, those browsers give to the user, the name of concepts, their properties and their relations. We can also know the destination type of a relation, and so on.

The MTRANS language associated with this framework is a language based on rules. A rule describes either a concept transformation (A to B) or a concept creation. Each rule is decomposed into two parts : one to specify attributes, and another to specify roles. This decomposition comes from the metamodel definition, indeed, a metamodel is defined by a set of concepts which contain attributes and which are linked by roles. Consequently, to obtain a valid instance of a meta-model we must fill these characteristics.

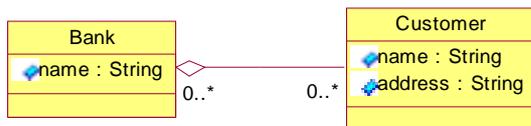
UML manipulation

As a simple example of model transformation, we are going to transform a class diagram into another by applying the transformation below:

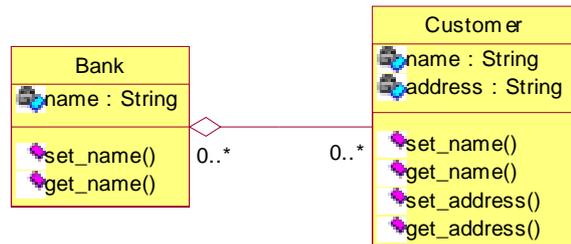
“All attributes which are publics are transformed into private attributes and we generate access methods for these new attributes”

This example is very naïve but is used here to illustrate the global characteristics of the MTRANS system. As we mentioned previously, a MTRANS program is composed by a fixed set of instructions (conditional, loop, etc.) and by a set of key words depending on the source meta-model. This second set is primarily use to navigate into the model by using the concepts

defined in the meta-model. For instance, we can use the type reference define in the UML meta-model to access the type of an attribute.



Source model



Destination model

```

sourceMeta-model : "miniuml_correct"
destinationMeta-model : "miniuml_correct"
mode : "modification"
setOfRules
{
  Attribute [visibility=="public"] to Attribute {
    attributes:
      visibility = "private"
    roles:
      type = type
  }
  and Operation {
    attributes:
      name = "set_" # name,
      visibility = "public",
      ownerScope = "instance",
      isQuery = "false",
      specification = "none",
      isPolymorphic = "false",
      concurrency = "sequential"
    roles:
      parameter = new [value] Parameter(type)
  }
  and Operation {
    attributes:
      name = "get_" # name,
      visibility = "public",
      ownerScope = "instance",
      isQuery = "false",
      specification = "none",
      isPolymorphic = "false",
      concurrency = "sequential"
    roles:
      parameter = new [return] Parameter(type)
  }
}

```

```

[value] Parameter {
  param:
    att
  attributes:
    name = "value",
    visibility = "private",
    defaultValue = new Expression (),
    kind = "in"
  roles:
    type = att
}

[return] Parameter{
  param:
    att
  attributes:
    name = "return",
    visibility = "private",
    defaultValue = new Expression (),
    kind = "return"
  roles:
    type = att
}

Expression {
  attributes :
    language = "none",
    body = "none"
}

```

MTRANS program to convert the source model into the destination model

Conclusion

Compared to an approach such as UMLAUT [HJGP99], the MTRANS framework with its meta-modeling approach is more generic. We can transform all kinds of models, which are MOF compliant. The MTRANS language has been used to manipulate UML documents. The example presented is simple but we can use MTRANS for manipulations which are much more complex. For instance, we used MTRANS to apply the visitor design pattern to a UML class diagram. We are presently providing a more rigorous description of the MTRANS language. Another aspect is that we can prove formally that a MTRANS program terminate (infinite loop or non-termination is not possible).

To finish, writing a model transformation with the MTRANS framework is fast because the MTRANS language is small, declarative and also because the MTRANS framework is a

guideline to write exact transformations rules (errors messages consistent, meta-model browser).

Bibliography

- [MOF]** OMG, Meta-Object Facility (MOF) specification version 1.3, Document ad/00-04-03, March 2000.
- [XMI]** OMG, eXtensible Metadata Interchange (XMI) specification version 1.0, Document ad/00-06-01, June 2000.
- [XML]** W3C, eXtensible Markup Language (XML) specification version 1.0 (second edition), October 2000.
- [XSLT]** Crane Softwrights, Introduction to XSLT and XPATH, Tutorial XML'99.
- [BP96]** Michael Blaha and William Premerlani, A Catalog of Object Model Transformations, Proceedings of the 3rd Working Conference on Reverse Engineering (Monterey, California), November 1996.
- [GR98]** M. Gogolla and M. Richters, Transformation rules for UML class diagrams, Proceedings of the Unified Modeling Language (Berlin), November 1998, pp. 92-106.
- [HJGP99]** W. M. Ho, J-M Jézéquel, A. Le Guennec and F. Pennaneac'h, UMLAUT : an extensible UML transformation framework, Technical Report 3775, INRIA, Octobre 1999.
- [Lem98b]** R. Lemesle, Transformation Rules Based on Meta-Modeling, Proceedings of the Second International Enterprise Distributed Object Computing Workshop (San Diego), November 1998