

MOF to EMF: There and Back Again

Anna Gerber and Kerry Raymond

{agerber, kerry}@dstc.edu.au

Cooperative Research Centre for Enterprise Distributed Systems (DSTC)

University of Queensland, Brisbane 4072, Australia

Abstract

The OMG's Meta-Object Facility (MOF) and the open source Eclipse Modelling Framework (EMF) are two popular meta-modelling frameworks, created to meet similar (but not identical) requirements. A means of translating MOF to EMF and vice versa is required to enable the two communities to leverage one another's specifications. This paper explains the relationship between the MOF and EMF meta-models and describes the XSLT-based translation tool, E-MORF, to convert between MOF and EMF.

1. Introduction

The Meta-Object Facility (MOF) [1] [2] is the common minimal meta-modelling framework that is used to define other modelling frameworks within the Object Management Group (OMG), e.g. UML [3], Common Warehouse Metamodel [4].

This paper describes a mapping between the MOF and EMF [5] meta-modelling frameworks by first comparing the MOF and EMF meta-models in Section 2. An XSLT-based translation from MOF to EMF is described in Section 3. The issues involved in round tripping are discussed in Section 4. Finally Section 5 concludes with summary and future work.

To prevent confusion, this paper uses the following terminology and notation. When talking specifically about a concept in MOF or EMF, we use the proper name, e.g. "Class" or "EClass" (all EMF concepts have the "E" prefix). When talking about a concept in a general O-O sense, we use lower case, e.g. "class".

Similarly, when working with meta-models, it is easy to become confused between the different layers of "meta-ness". Figure 1 illustrates how the conversion between MOF and EMF exists at three different levels;

- the comparison of the meta-models,
- the mapping between models defined by those meta-models,
- the mapping of the instances of those models.

2. Comparison of MOF and EMF

Historically, the MOF is the older of the two modelling frameworks, and the design of EMF was influenced by MOF. However, the goals and processes of the OMG differs from those of Eclipse, resulting in different design decisions.

2.1. Goals

OMG is an open standards forum with formal processes to produce standards for object-oriented distributed systems which are independent of programming language. The resulting specifications are intended to be suitable for subsequent independent implementation by different vendors, and are very stable, being updated infrequently. The MOF defines an IDL mapping for the provision of repositories for models defined in MOF; MOF products usually provide language-specific implementations for these interfaces.

In contrast, the Eclipse project is an open source project that provides a shared code base for public use, capable of rapid evolution, but the specification may lag behind the implementation. EMF is intended as a low-cost tool to obtain the benefits of formal modelling and Java code generation (and hence is neither language independent nor distributed). As a consequence, EMF tends to take a bottom-up approach whereas MOF tends to take a top-down approach.

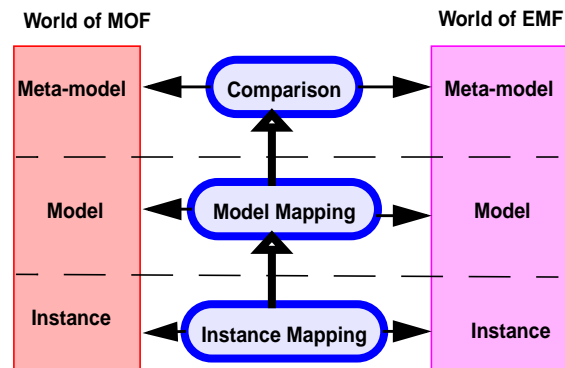


Fig 1. Mapping between MOF and EMF at different meta-layers

2.2. Concepts

MOF and EMF are very similar conceptually, both based on the concept of classes with typed attributes and operations with parameters and exceptions, supporting reuse through multiple inheritance. Both frameworks use packages as a grouping mechanism and support nested packages.

The main conceptual difference lies in their treatment of relationships between classes. MOF has the first-class concept of Association as a binary relationship between two classes (the AssociationEnds, which have a navigability property). In MOF, there is a distinction between relationships that are fundamental to the definition of a class (the References in a class accesses its related classes) versus those that are observations about a class (and not fundamental to the definition of the class). For example, a person's parents are fundamental to the nature of each person, whereas their next-door neighbour is merely an observation about a person. MOF also supports the use of class-typed attributes, which enable one object to refer to another. Through these constructs, MOF is able to describe relationships between classes with a number of subtle semantic flavours.

EMF (reflecting its bottom-up more code-centric approach) has only EReferences (which can be thought of as class-typed attributes), without Association Ends or Associations. Two EMF EReferences can be defined to be "opposite" of one another, forming a de-facto two-way relationship between the classes. There are advantages and disadvantages to these two approaches to modelling relationships. In MOF, symmetric associations (e.g. "spouse of") cannot be easily expressed (as both X-Y and Y-X must be consistently maintained). However, in EMF, an EReference can be its own "opposite" which precisely captures the semantics of a symmetric association. However, for asymmetric associations, the MOF provides a single place to describe that relationship, as opposed to distributing this information into the various endpoint classes.

The other significant difference is the treatment of data types. MOF 1.3 [1] re-uses the CORBA TypeCode, while MOF 1.4 [2] incorporates concepts like CollectionType, StructureType etc as subtypes of DataType within the MOF itself. However both are language-independent mechanisms. EMF takes a hybrid approach to EDataType by re-using Java types, but introduces an EEnum subtype

of EDataType in EMF to handle enumerations (not directly supported by Java). Unfortunately, as the datatype aggregation is done in Java, it is only possible to use the EEnums as part of any aggregated data type through reuse of the EMF-generated Java code created for these EEnums.

Both MOF and EMF support the reuse of concepts in other packages. MOF supports a variety of package reuse mechanisms, such as explicit package import and package inheritance, not available in EMF. However, EMF allows mutually recursive definitions between packages (provided both are loaded together), whereas MOF prohibits cyclic dependencies between packages due to the practical problems in maintaining their mutual consistency in a distributed environment.

Generally, MOF is the larger richer model with explicit support for more concepts. MOF has an explicit concept of Constant, whereas in EMF, these must be defined as unchangeable EAttributes (and hence can only be defined within an EClass). EMF has no equivalent to MOF's Constraint.

2.3. Properties

Generally MOF concepts have more properties than the equivalent EMF concept. In MOF, MofAttributes and Operations can be defined as having instance or classifier scope, whereas EMF has only instance scope. Support for multi-valued attributes is available in both MOF and EMF, but only MOF permits parameters to be multi-valued (within the model). Also MOF supports the notion of ordering within multi-values whereas EMF does not.

Although EMF does have some properties without equivalents in MOF, such properties are generally "advice" to the EMF code generation. In contrast, MOF does not allow code-generation information to form part of the model but instead supports a system of Tags (application-specific name/value pairs) which can be added to each model element for such purposes. Early versions of EMF had no similar concept to Tag, but the recent introduction of EAnnotation into EMF provides a similar construct.

A complete description of the mapping between MOF and EMF is given in [6].

3. E-MORF: translating with XSLT

The XMI standard [7] defines how MOF models and instances of MOF models can be interchanged using XML. Both MOF and EMF support the use

of XMI enabling the interchange of models and model instances through XML based on DTDs/XMLSchemas generated from the corresponding models. However, as the underlying models are different, MOF/XMI and EMF/XMI are not interchangeable. Therefore, mapping between MOF and EMF was most easily done by translating XML using XSLT [8].

The E-MORF tool consists of two XSLT files (centre ovals in Figure 2). The user starts with a MOF model M expressed in MOF/XMI. In Step 1, using M as input, the MOF-model-to-EMF-model XSLT translation produces two results:

- an EMF/XMI file that contains the corresponding EMF model E(M)
- an XSLT file that translates from instances of MOF model M (expressed in M/MOF/XMI) to corresponding instances of EMF model E(M) expressed in E(M)/EMF/XMI.

Having created the translation for instances of M into instances of E(M), then in Step 2, MM, an instance of the MOF model M expressed in M/MOF/XMI, is translated into EE(MM), an instance of the EMF model E(M) expressed in E(M)/EMF/XMI.

If it is then desired to reverse the process, Step 3 translates the EMF model back to a MOF model and creates an XSLT file to translate instances of that EMF model into instances of the correspond-

ing MOF model (Step 4).

It is important to understand that while the XSLT files that translate between the MOF and EMF models are fixed (and hand-programmed), the XSLT files that translate between corresponding instances of MOF and EMF models are generated.

3.1. Java data types

Figure 2 does not present a complete view of the transformations. As explained in Section 2, data types are not completely modelled in EMF but must be expressed as Java types. Therefore, the translation between the MOF model and the EMF model must also produce a set of Java data types. Some data types are used by all models and can be placed in a standard library. However, data types derived from user-defined DataTypes in the MOF model must be generated for each model.

Java datatypes are also required for “helpers” needed to translate some MOF concepts into EMF concepts. For example, the “inout” Parameters of MOF Operations are passed by reference whereas all EParameters of EOperations in EMF are passed by value. Therefore a MOF “inout” Parameter of type T must be mapped to an EParameter whose type is a Java class which holds an attribute of the Java-equivalent of T (enabling pass-by-reference).

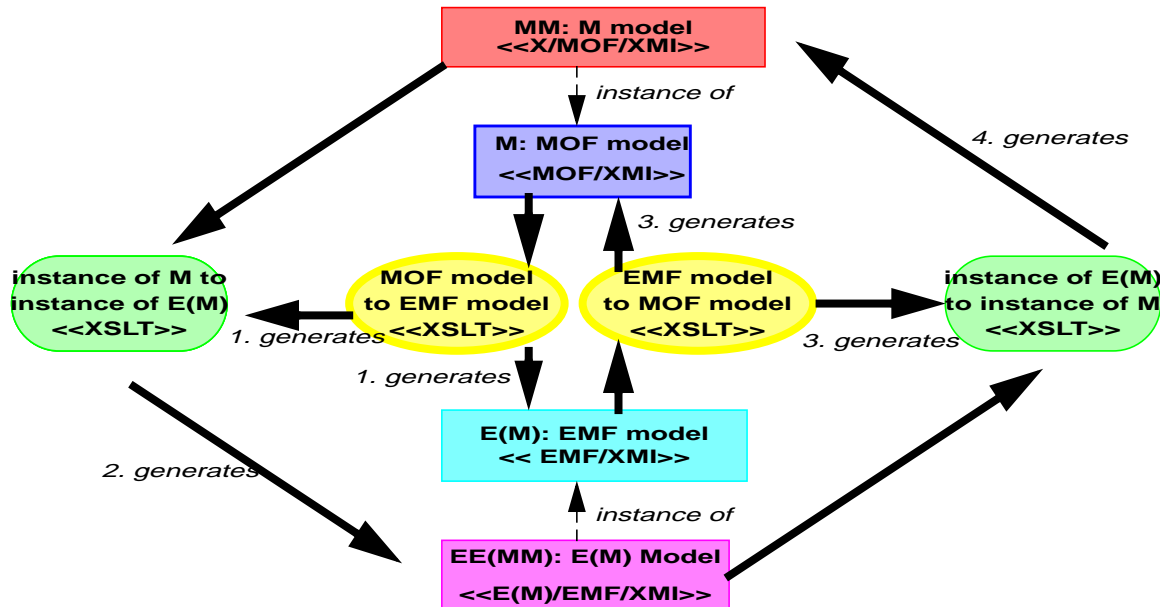


Fig 2. Translating between MOF and EMF by applying XSLT to XMI

Consequently the inverse mapping (EMF model to MOF model) must use the knowledge of the Java datatypes to create MOF Datatypes. As XSLT requires its input to be XML, JavaML [9] is used to translate Java datatypes into XML, as input to the EMF-to-MOF model translation.

3.2. Hierarchical Structure

EMF has a very simple fixed containment structure in its meta-model. The EPackage can contain sub-packages and classifiers (EClass and EDataType). EClasses can contain EAttributes, EReferences, and EOperations. EOperations can contain EParameters, and so on. With the exception of EAnnotation, the type of the container is always known for every EMF model element.

However, MOF permits more complex containment rules in its meta-model, in particular Classes are allowed to be nested within other Classes to an unlimited depth. Therefore, the translation from MOF model to EMF model must flatten out any deeply nested MOF model elements by promoting them to some higher point in their containment hierarchy. For example, if a MOF Package P contains a MOF Class C1 which contains a MOF Class C2 which in turn contains MOF Class C3, then in EMF, the EPackage P will contain three EClasses C1, C2 and C3.

3.3. EAnnotations

Because the MOF has a richer meta-model than EMF, there is some information loss that inevitably occurs, e.g. there is no EMF equivalent of MOF Constraint. However, to support round-trip translation (MOF model to EMF model to MOF model), such information must be preserved within the EMF model through the use of EAnnotations. The general principle is that if a MOF element has no equivalent in EMF, then an EAnnotation is generated and attached to the EMF-equivalent of the “leaf-most” ancestor element in MOF for which there is an EMF-equivalent. That is, whole subtrees of MOF concepts may be collapsed into a set of EAnnotations, which are then attached at the lowest appropriate point in the EMF containment tree.

For example, MOF Associations are translated to several EAnnotations on the EPackage mapped from the containing MOF Package, one for each attribute of the class MOF Association, such as “isDerived”. Similarly, the corresponding Associa-

tionEnds and their attributes are translated to EAnnotations on the EPackage.

In addition to information loss, EAnnotations are also generated to:

- hold the “annotation” Attribute possessed by every MOF model element
- be the equivalent of the MOF Tag concept
- record the “rule” and “sources” which generated each element for round-trip purposes

For example, a MofAttribute whose type is a Class becomes an EReference in EMF, while a MOF Reference also translates to an EReference. Therefore, it must be recorded whether an EReference arose from a MofAttribute or a Reference in the original MOF model, if round-tripping is to succeed.

3.4. Name mangling

A common problem throughout the translation of MOF models to EMF models is the need to create names for elements in the EMF model. In general, the goal is to preserve names. However the generation of Java datatypes, EAnnotations and hierarchy flattening all result in multiple elements being generated within the same EMF naming scope, which must be separately named to avoid name clashes. For example, if MOF Class C contains Class C, the naive mapping to EMF would create two EClasses both called “C” within the same EPackage. Also, hierarchical names had to be flattened into a single string using “punctuation” symbols (e.g. “C1::C2::C3”).

Considerable care was required to develop naming schemes that are meaningful to the reader while avoiding collisions within the same scope.

3.5. EMF to MOF translation

As the EMF meta-model is simpler than the MOF meta-model in terms of its concepts, properties and containment structure, the mapping of EMF’s concepts into MOF’s concepts is relatively straightforward and is mostly 1-to-1 translations. The most notable exception is when a pair of “opposite” EReferences in an EMF model must be mapped to an Association, two AssociationEnds, and two References in the MOF.

Any property in the EMF meta-model that has no equivalent in the MOF model is mapped onto MOF Tags. While there are some name mangling considerations here, they are simpler than those in the translation from MOF model to EMF model.

There is one absolute show-stopper for translating EMF models to MOF models: the creation of dependency cycles between “root” packages which MOF does not support.

4. Round-tripping: and back again!

While translating “native” EMF models to MOF models is relatively straightforward, round-tripping from MOF model to EMF model to MOF model is more complex, as the goal is not just to produce any equivalent MOF model but one as close as possible to the original, e.g. by exploiting the additional information embedded in the EAnnotations.

For example, consider an EReference in EMF which has no opposite EReference. The simplest translation into MOF is to create a class-typed MofAttribute. However, a more complex alternative is that this EReference should be translated into a Reference belonging to an Association for which the “other end” has no Reference. If there are generated EAnnotations contained in the EReference, then they indicate that this model was previously represented in MOF and whether the EReference was originally a MofAttribute or a Reference, which determines the reverse translation to be applied.

Therefore, the translation from EMF model to MOF model becomes quite complex as all translations must be developed in the following style:

- in the absence of evidence of “round-tripping”, apply the simplest translation rule (the “native” rule)
- in the presence of evidence of “round-tripping”, analyse what translation was originally applied and reverse it, *provided everything is self-consistent*
- if the round-tripping information is inconsistent with the EMF model, ignore the round-tripping information and apply the simplest transformation rule.

4.1. EMF to MOF to EMF round-tripping

Since most aspects of the EMF-to-MOF translation are 1-to-1, the round-tripping from EMF to MOF and back to EMF is quite straightforward, as the round-trip translation is the same as the simplest “native” translation in most situations.

5. Conclusions

Apart from some pathological cases, the E-MORF tool can translate between MOF and EMF in both directions, including support for round-tripping. This enables models defined for either the CORBA or Eclipse environment to be easily ported to the other platform.

The next challenge is to support the translation of the Java programs which are clients of the generated interfaces for MOF to work with the equivalent generated interfaces of EMF and vice versa.

Acknowledgements

The work reported in this paper has been funded in part by the Co-operative Centre for Enterprise Distributed Systems Technology (DSTC) through the Australian Federal Government’s CRC Programme (Department of Education, Science and Training) and in part by an IBM Eclipse Innovation Award.

About the Authors

Anna Gerber is a Research Scientist at DSTC with special interests in Eclipse and XSLT. Kerry Raymond is a Distinguished Research Fellow at DSTC and is one of the authors of MOF.

References

- [1] “Meta Object Facility (MOF) v1.3.1”. OMG Document: formal/01-11-02, Nov. 2001.
- [2] “Meta Object Facility (MOF) v 1.4”. OMG Document: formal/2002-04-03, Apr. 2002.
- [3] “Unified Modeling Language v1.4”. OMG Document: formal/01-09-67, Sept. 2001.
- [4] “Common Warehouse Metamodel (CWM) Specification”, OMG Documents: ad/01-02-{01, 02, 03}, Feb. 2001.
- [5] “EMF Developer FAQ”, www.eclipse.org/emf
- [6] Anna Gerber & Kerry Raymond, “Mapping MOF to EMF”, DSTC Technical Report, March 2003.
- [7] “XML Metadata Interchange v 1.2”, OMG Document: formal/2002-01-01, Jan. 2002.
- [8] “XSL Transformations (XSLT) v1.0”. W3C Recommendation: www.w3.org/TR/xslt, Nov. 1999.
- [9] Greg J. Badros, “JavaML”, www.cs.washington.edu/homes/gjb/JavaML/