

**Eclipse Modelling Framework (EMF)
import/export from MOF / JMI
Status Report: 3 May 2003**

**Keith Duddy, Anna Gerber, Kerry Raymond
Pegamento Project, DSTC
{dud, agerber, kerry}@dstc.edu.au**

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 report 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. Based on this specific translation, general principles for transformation between modelling frameworks are discussed and used to reflect on our current standardisation work on MOF Query/View/Transformations (QVT) within the OMG's Model Driven Architecture initiative.

1. Introduction

1.1. Background

The OMG's Model Driven Architecture (MDA) [1] provides a set of guidelines for structuring specifications expressed as models and the mappings between those models. The MDA classifies technologies as either Platform Independent Models (PIMs), or Platform Specific Models (PSMs). The MDA approach and the standards that support it allow the same PIM to be realized on multiple platforms through mappings from the PIM to PSMs that implement the functionality described in the PIM.

The Meta-Object Facility (MOF) [2] [3] is the standard facility used by the OMG to define modelling frameworks used to express both PIMs and PSMs. Mappings between PIMs and PSMs (or any other models expressed using a framework described using MOF) can be expressed at the same level of abstraction as the models' meta-models. MOF Query/Views/Transformations (QVT) [4] is currently undergoing standardisation through the OMG to provide the mechanism for expressing these mappings.

EMF is the Eclipse Modelling Framework used by IBM's open source Eclipse project [6]. EMF represents the next generation of meta-modelling and object repository technology that started with the Meta Object Facility standard of OMG, and was progressed within the Java Community Process to produce the Java Metadata Interface standard.

1.2. Objective

A number of standard meta-models are currently published by OMG and other standards bodies, using the MOF 1.3 and MOF 1.4/JMI standards as their expression language. This project aims to produce mappings from MOF/JMI to EMF and vice versa, which will facilitate the interchange of MOF and EMF models and their corresponding instances. The plan is to wrap these XSLT implementations into plug-ins for the Eclipse environment.

1.3. Methodology

The differences between the EMF and MOF meta-models will be investigated, and mappings between them will be devised and documented. The primary consideration will be to produce a uniform and automatable way of importing existing MOF meta-models and their instances into Eclipse. A secondary goal is to provide round-tripping of meta-models and instances between Eclipse and MOF/JMI repositories. Any possible information loss between models will be documented, and may be compensated for by tagging of models. Preliminary investigation leads us to believe that mappings and tags can facilitate round tripping of meta-models and model instances between MOF/JMI and EMF.

The cleanest integration path is via the XML Model Interchange, which differs in style from MOF to EMF due to the variations in their meta-meta-models. XML translation techniques, using XSLT will be investigated as a bridge for model interchange between MOF/JMI implementations and Eclipse.

We are using DSTC's dMOF suite [18] as a reference implementation of MOF1.3 and Sun Metadata Repository (MDR) [19] as a reference implementation of MOF 1.4. We are currently using Eclipse 2.1 and EMF 1.1.0 (but due to the open source nature of Eclipse and EMF, we expect to continue to be upgrading to newer versions as they become available).

1.4. Milestones

- Initial draft of MOF/JMI to EMF meta-model mapping devised - 5 person weeks
- Complete Mappings from MOF -> EMF and EMF -> MOF with roundtrip tagging will be specified - 2 person weeks
- Work begins on XMI transformation stylesheet to import MOF models and instances into Eclipse - 10 person weeks
- Working XMI translation for roundtrip import/export between Eclipse and MOF/JMI repositories - 4 person weeks
- Implementation of import/export integrated into Eclipse code base. - 3 person weeks

1.5. Structure of this report

This report documents the mapping between the MOF and EMF [5] meta-modelling frameworks. The comparison of the MOF and EMF meta-models is presented in Section 2. An XSLT-based translation from MOF to EMF is described in Section 3 with Section 4 containing a detailed table outlining the translation. Section 5 discusses some of the more interesting or troublesome aspects of the mapping. The issues involved in round tripping from MOF to EMF and then back to MOF are discussed in Section 6, while successive round-trip translations are discussed in Section 7.

While EMF is not a framework described using MOF, the mapping between MOF and EMF provides a significant transformation example, from which we were able to identify a number of general transformation principles that we use to reflect on our current QVT Transformation model [7]. We describe the general transformation principles learnt from mapping MOF to EMF and back again in Section 8, and use these principles to reflect on DSTC’s QVT submission in Section 9. Finally Section 10 concludes with summary and future work.

1.6. Terminology and notation

To prevent confusion, this report uses the following terminology and notation. When talking specifically about a concept in MOF or EMF, we will 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 will use lower case, e.g. “class”. When converting between text-based representations, we use the term “translation”. When mapping between object-based representations, we use the term “transformation”. When converting from an object-based representation to a text-based representation, we use the term “rendering”. We use the term “mapping” to cover all possibilities.

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.

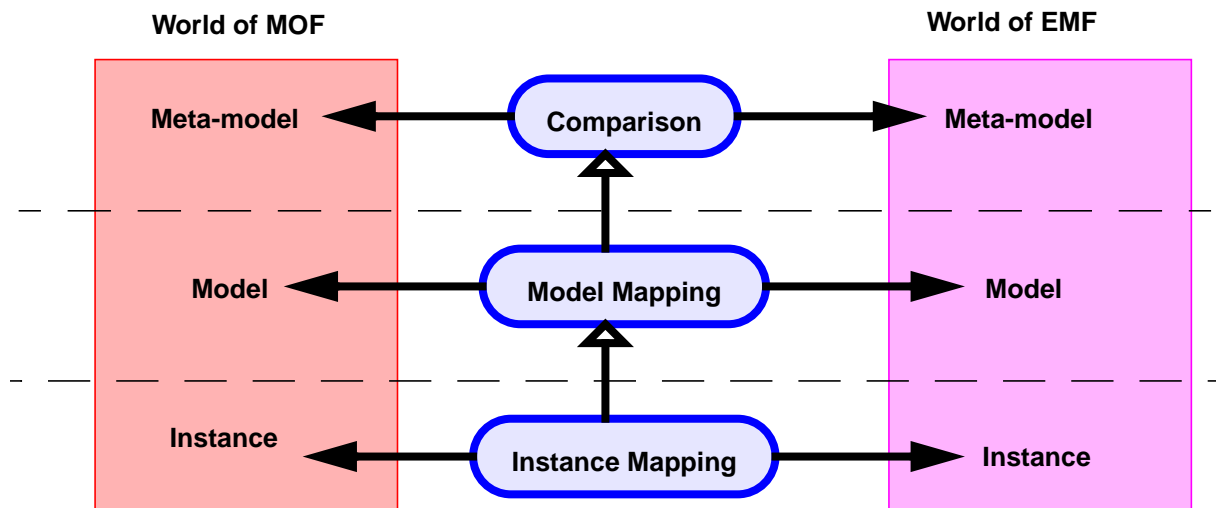


Figure 1. Mapping between MOF and EMF at different meta-layers

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 two organisations involved are significantly different and this accounts for some difference in design decisions.

2.1. Goals

OMG is an open standards forum with well-defined 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 by a well-defined process. The MOF is the common minimal meta-modelling framework that is used to define other modelling frameworks within the OMG (e.g. UML [8], Common Warehouse Metamodel [9]). 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/documentation 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.

2.2. Concepts

Not surprisingly, MOF and EMF are very similar conceptually. The key concept in both is the notion 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 general 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 [2] re-uses the CORBA TypeCode, while MOF 1.4 [3] incorporates concepts like CollectionType, StructureType etc as subtypes of DataType within the MOF itself. However, in both MOF versions, these are language-independent mechanisms. EMF (reflecting its links to Java) takes a hybrid approach to EDataType by re-using Java types, but introduces an EEnum subtype of EDataType to handle enumerations which are 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 types through reuse of the 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 and interpretation 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. 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 (February 2003) introduction of EAnnotation into EMF provides a similar construct. We anticipate that some of the more implementation-specific properties of EMF may be refactored as EAnnotations in future revisions to produce a "cleaner" EMF model

3. Translating between MOF and EMF with XSLT

The XMI standard [10] 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

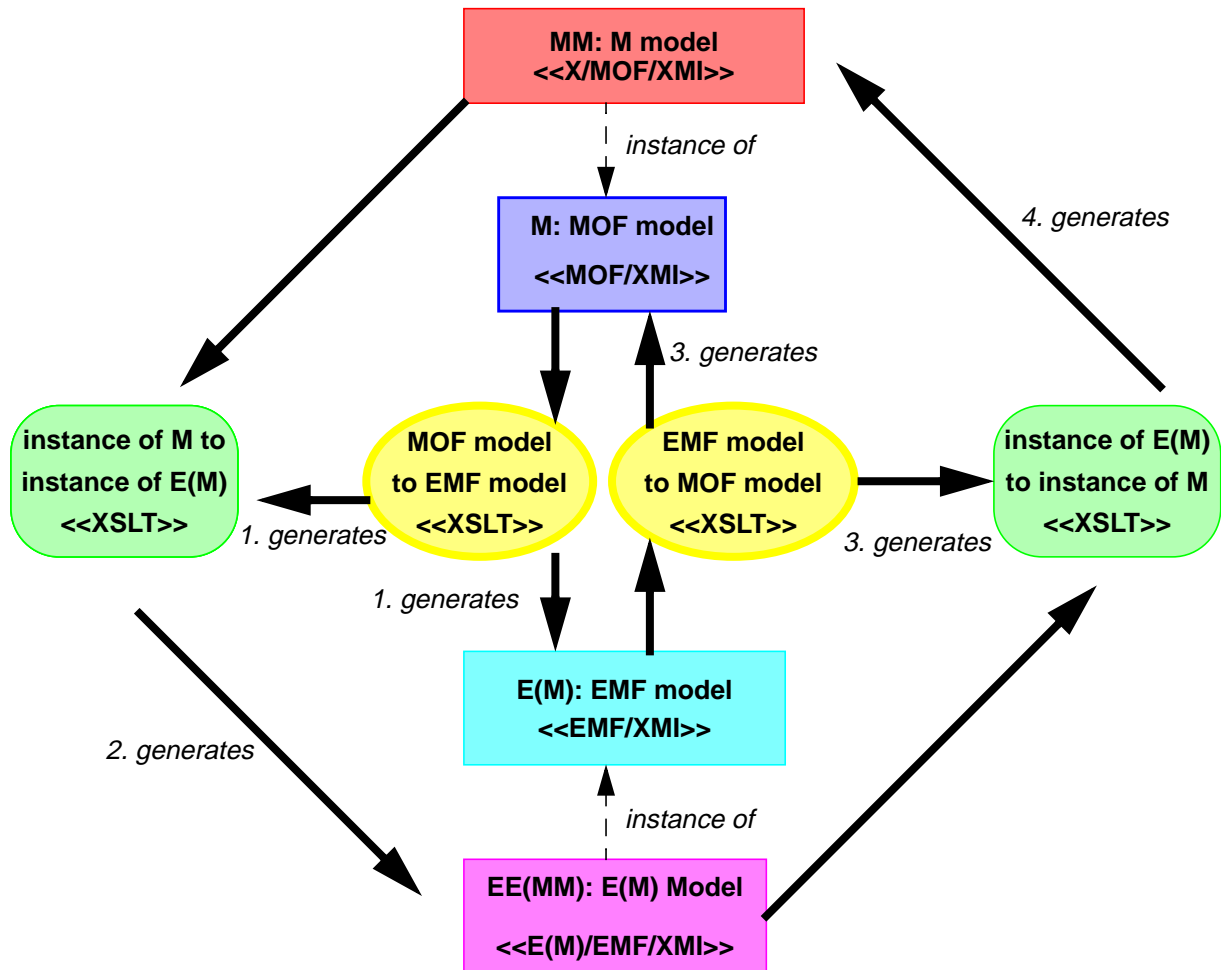


Figure 2. Translating between MOF and EMF applying XSLT to XMI

corresponding models. However, as the underlying models are different, MOF/XMI and EMF/XMI are not interchangeable.¹ Therefore, mapping between MOF and EMF was most easily done by translating XML using XSLT [11].

Figure 2 illustrates the translation. The E-MORF tool consists of two XSLT files (centre ovals in Figure 2).

Suppose 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.

1. There is widespread misunderstanding that XMI is a single format that can be interchanged between any XMI-compliant systems. XMI is in fact a family of formats, each generated from a model. Only XMI generated for the same model is interchangeable.

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 corresponding 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 2.2, data types are not completely modelled in EMOF but must often be expressed as Java types. Therefore, the translation between the MOF model and the EMF model must also produce a set of Java files containing definitions of the necessary data types in Java. 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. Also, (to be discussed below), additional datatypes must also be produced to compensate for missing concepts in the EMF model.

Consequently the inverse mapping (EMF model to MOF model) must use the knowledge of the Java datatypes to create MOF DataTypes. Although XSLT can produce output in any text-based format, its input must be XML (and not Java type definitions). Therefore, an existing tool (JavaML [12]) is used to translate Java datatypes into XML, which can then be used as input to the EMF-to-MOF model translation. Only the structure of the Java datatype need be considered as neither EMF DataTypes nor MOF DataTypes can support Operations at the modelling level (obviously both EMF and MOF uses methods in the implementation of the models but these are produced by EMF and MOF code generation tools).

In addition to translating MOF DataTypes, Java datatypes are also required for “helpers” needed to translate some MOF concepts into EMF concepts. For example, the Parameters of MOF Operations are directional (“in” or “out” or “inout” as used in CORBA), whereas EParameters of EOperations in EMF are assumed to be passed by value (i.e. “in” direction). Therefore a MOF “in” Parameter of type T can be mapped to an EMF EParameter of the corresponding type “T”, but an “out” or “inout” Parameter must be mapped to an EParameter whose type is a Java class which holds an attribute of the corresponding type T (enabling it to be passed by reference).

3.2. 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. Since any EMF model element can contain EAnnotations, the general principle established for handling information loss between the MOF meta-model and the EMF meta-model is as follows:

- If a MofAttribute or Reference of a Class in the MOF meta-model has no equivalent in the EMF meta-model, then an EAnnotation is added to the corresponding class in EMF meta-model to hold the attribute/reference values as key-value pairs. For example, the isLeaf attribute of the MOF Class is translated to an EAnnotation with key name set to “isLeaf” and value set to the value of the isLeaf attribute.
- If a Class in the MOF meta-model has no equivalent in the EMF meta-model, then all of the information contained by that class (and its contents) is added to the EPackage (or other container) as EAnnotations. 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 AssociationEnds and their attributes are translated to EAnnotations on the EPackage.

In addition to information loss, EAnnotations are required for EMF model elements for the following purposes:

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

In many situations, it is not necessary to explicitly record the “rule” if there is only one rule that might have generated such an EMF model element. However, in ambiguous situations, the rule used must be explicitly recorded. 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.

3.3. 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.

3.4. 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. Where the mapping is 1-to-1 and the containment structure is preserved, the name of the source element can be used for the target element as it will remain sufficiently unique.

However, as illustrated by the section above, 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.

Considerable care was required to develop naming schemes for Java datatypes, EAnnotation keys and deeply-nested elements that are meaningful to the reader (i.e. embed the original source element name) but which avoid collisions with other user-defined or generated names within the same scope.

In particular, the naming schemes for EAnnotation keys and deeply-nested elements must allow for qualified names to be flattened into a single string, introducing the need for “punctuation” symbols within these flattened names (e.g. “C1::C2::C3”). In theory, there is the added complication that these punctuation symbols may themselves appear within the individual names and require escape conventions, as neither MOF nor EMF explicitly restrict the allowed symbols used to form names. However, in practice, users do not embed such symbols in individual names, as such names are usually illegal in the programming languages commonly used in conjunction with MOF and EMF (and the users rapidly learn to avoid such symbols).

3.5. EMF to MOF translation

This section has so far focused on the translation of MOF models to EMF models. 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 considerably 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. We do not address this issue currently.

4. Concept Mapping

The following table shows the coarse-level equivalence between MOF1.4 model concepts and EMF model concepts. As the EMF model was substantially influenced by MOF, it is hardly surprising that there is a large subset of similar concepts. Abstract concepts are shown in italics; the equivalence of abstract concepts is interesting intellectually but relatively unimportant in practical terms.

Table 1: Equivalent concepts in MOF1.4/JMI and EMF

MOF1.4/JMI concept	EMF concept	Comments
<i>ModelElement</i>	<i>EModelElement</i> , <i>ENamedElement</i>	ModelElement and EModelElement are the base class in both universes. ModelElement has a name (and a derived qualified name) but EModelElement does not; in EMF, it is the ENamedElement that has the name attribute.
<i>Namespace</i>	--	Namespace implements the naming hierarchy in MOF, and provides name resolution operations. In EMF, only certain specific classes (e.g. EPackage, EClass, EEnum) contain specific operations to lookup some of their content by name, but there appears to be no operation to lookup-by-name EOperations within EClass or EParameters within EOperations. EMF does not appear to have a concept of fully-qualified names or operations to resolve such names.
<i>GeneralizableElement</i>	--	Only the concrete class EClass is capable of generalisation, so no abstract equivalent is needed in EMF. MOF has attributes for visibility, isRoot, and isLeaf for which there is no equivalent in EMF. The notion of abstract GeneralizableElement in MOF is equivalent to the notion of “interface” in EClass (and not to “abstract” in EClass).
Package (without generalisation or importing)	EPackage	A grouping mechanism in both universes. In EMF, there are a number of attributes considered with the corresponding XML namespaces and generated Java, which do not derive from any information in the MOF and for which we will have to establish a set of standard conventions (similar to the Eclipse defaults).
Package (with generalisation), Import	--	EPackages cannot support generalisation, except through copying of the inherited content. There is a mechanism for import through the EMF “XMI” document#//fully-scoped-name.
<i>Classifier</i>	<i>EClassifier</i>	things that have “type” in both universes.
Class (with or without generalisation)	EClass	used to define “things” in both universes. MOF Classes have the attribute “isSingleton” for which there is no corresponding EMF concept.

Table 1: Equivalent concepts in MOF1.4/JMI and EMF

MOF1.4/JMI concept	EMF concept	Comments
<i>DataType</i>	EDataType	MOF has a built-in type model to make it language neutral, whereas EMF uses Eclipse-specific types. EMF has the notion of default values for EDataTypes, but there is no equivalent concept in MOF.
PrimitiveType	EDataType	The EDataType nominates EInteger, etc.
StructureType, Structure-Field	EDataType	The EDataType nominates a Java class with the corresponding structure.
CollectionType	EDataType	The EDataType nominates a Java class with an attribute “value” of the appropriate base type and multiplicity.
AliasType where aliased type is PrimitiveType	EDataType	The EDataType nominates a Java class which is a subtype of the Java class type nominated by the EDataType corresponding to the PrimitiveType.
AliasType where aliased type is StructureType	EDataType	as above
AliasType where aliased type is CollectionType	EDataType	as above
AliasType where aliased type is AliasType	EDataType	as above
AliasType where aliased type is EnumerationType	EDataType	as above
EnumerationType	EEnum, EEnumLiteral	The EEnum nominates an introduced Java class with final attributes for each of the EEnumLiterals.
<i>TypedElement</i>	<i>ETypedElement</i>	Things that have an associated type in both universes
<i>Feature</i>	--	There are fewer concrete types in EMF so this abstraction is not required. There is no concept in EMF of scope and visibility used for MOF Features (and damn good thing too).

Table 1: Equivalent concepts in MOF1.4/JMI and EMF

MOF1.4/JMI concept	EMF concept	Comments
<i>BehaviouralFeature</i>	--	There is only one concrete behaviour feature in EMF, EOperation, so no abstract type is required.
<i>StructuralFeature</i>	<i>EStructuralFeature</i>	Data held by a thing in both universes.
Parameter (multi-valued)	EParameter, EDataType	EParameter must be of an EDataType for an appropriate Java collection. Single-valued Parameters have the corresponding type to the EParameter.
Parameter (direction in)	EParameter	All EParameters are “by value”
Parameter (direction out or inout)	EParameter	EParameter must be of an EDataType which nominates a Java class which wraps the data (if it is not already a class).
Operation, Parameter (direction return)	EOperation	Unlike Operations which optionally have a named return parameter, EOperations have a (unnamed) return type by virtue of being an ETypedElement. How do we handle Operations without return types? A. The EOperation has no associated type (it is optional). There is no EMF equivalent to the MOF concept of isQuery Operations.
Exception and its contained Parameters	EClass, EAttributes/EReferences	The Exception is mapped to an EClass and each of its Parameters is mapped to an EAttribute or EReference (depending on whether the type of the Parameter is a DataType or Class respectively). There is no problem here with multi-valued parameters as EAttributes/EReferences can be multi-valued.
Attribute where the type is a DataType	EAttribute	Attributes can have type which is a Class but EAttributes can only have type which is EDataType, as EClass-valued contents of a an EClass are modelled as EReferences. The following attribute mappings apply: <ul style="list-style-type: none"> • MOF isChangeable -> EMF changeable • MOF isDerived -> EMF volatile, transient • MOF Multiplicity -> EMF unique, lowerBound, upperBound (there is no concept of ordering of multi-valued attributes) • There is no concept in MOF of default values • EMF many = upperBound > 1 • EMF required = lowerBound >= 1 • EMF unsettable = lowerBound == 0

Table 1: Equivalent concepts in MOF1.4/JMI and EMF

MOF1.4/JMI concept	EMF concept	Comments
Attribute where the type is a Class	EReference	as above, plus: <ul style="list-style-type: none"> • container = true (as MOF Classes contain their class-typed attributes) • containment = false • eOpposite = null • resolveProxies = false (as contained elements in MOF must be within the same extent)
Reference, Association, Association End	EReference	<p>A 2-way-navigable Association in MOF has 2 AssociationEnds and 2 References. This map to a pair of “opposite” EReferences. A 1-way-navigation Association in MOF has 2 Association Ends and 1 Reference, and makes to 1 EReference.</p> <p>A non-navigable Association in MOF has no equivalent in EMF, as EMF does not distinguish between “relationship between” and “relationship to”.</p> <p>As navigation is not required in the MOF world to support querying of associations, it may be that the MOF modeller has intended the association to be accessible despite the absence of explicit References. Therefore, we have 2 alternate mappings (will be selected by a parameter). The first makes all non-navigable AssociationEnds navigable (as if there was a Reference). The second makes all non-navigable Associations navigable in both directions, but does nothing if the Association was defined as 1-way navigable.</p> <p>Mappings of attributes as follows:</p> <ul style="list-style-type: none"> • MOF Reference.name (if available) else MOF AssociationEnd.name concatenated with MOF Association.name (for uniqueness) -> EMF.EReference.name • MOF Association.isDerived -> EMF.transient, volatile • MOF AssociationEnd.aggregation = composite -> the corresponding EReference is container and the eOpposite EReference is containment • MOF AssociationEnd.aggregation = none, both EReferences are neither container or containment • MOF AssociationEnd.aggregation = shared --> there is no equivalent (treat as aggregation = none) • MOF AssociationEnd Multiplicity -> EMF EReference unique, lowerBound, upperBound (there is no concept of ordering of multi-valued attributes) • MOF AssociationEnd isChangeable -> MOF EReference changeable • MOF AssociationEnd otherEnd -> EMF EReference.eOpposite

Table 1: Equivalent concepts in MOF1.4/JMI and EMF

MOF1.4/JMI concept	EMF concept	Comments
Constant	EDataType	The EDatatype nominates a Java class with only a single attribute “constant” which is of the appropriate type and value and is final.
Constraint	--	There is no equivalent concept in EMF which makes life a lot easier :-)
Tag	--	There is no equivalent concept in EMF which makes it difficult to pass in information that may not be interpreted (or interpreted only by specific tools).
--	EFactory	The package factory in MOF does not modelled but is generated for the implementation.
Shared aggregation (white-diamond) on Associations		No comparable concept, only black-diamond containment supported

5. Issues in the mapping

5.1. What is EMF?

One of the issues in mapping MOF 1.x to EMF is the lack of up-to-date documentation on EMF. Often it is necessary to experiment with the code or to locate files in the distribution to determine the precise definition of EMF concepts.

5.2. Nesting in MOF

In MOF 1.x, it is permitted to declare classes, datatypes and exceptions within a class. EMF does not support such nesting, and the corresponding EMF constructs must be declared at the package level. This requires name mangling.

5.3. Classifier-level attributes and operations in MOF

In MOF 1.x, attributes and operations can be declared as being at the classifier-level (i.e. shared by all instances of that class), similar to “static” attributes in Java. There is no equivalent concept in EMF.

5.4. Complex data types in EMF

There is a problem with EEnum in EMF in that it appears that you cannot construct aggregate types (e.g. structures, collections) from it. Therefore, we map simple enumerations into EEnum but aggregate data types must be mapped into Java classes with no obvious linkage to the EEnum in EMF. This is EMF’s problem!

5.5. Creation of Java classes

Note where we say “nominates a Java class”, this requires the definition of this Java class. A naming convention will be employed to provide suitably mnemonic names for these Java classes.

5.6. Name Mangling in the mapping

Name mangling can cause problems in the mapping. For example, a Class X may require the creation of a XList datatype. This name potentially clashes with any XList already defined within the MOF model. However, as similar name mapping takes place in the OMG IDL mapping, it can hopefully be assumed that MOF models will not generally define names that cause clashes in the OMG IDL mapping and therefore there should be no problem in the EMF mapping.

6. Roundtripping: and back again!

While translating “native” EMF models to MOF models is relatively straightforward, roundtripping from MOF model to EMF model to MOF model is far more complex for two reasons:

- unlike a “native EMF model”, a generated EMF model is rich in EAnnotations, complex flattened names, etc (described earlier in this section) which have semantic implications in the MOF model and must be interpreted
- the goal of round-tripping is not to produce just any MOF model that is semantically equivalent to the EMF model, but to try to reverse the original transformation and recover the original MOF model as closely as possible (by exploiting the information contained in the generated EAnnotations)

6.1. Reversing a previous translation

To illustrate the point, 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 firstly that this model was previously represented in MOF and should indicate 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.

6.2. Inconsistency

The problem that can occur when attempting to reverse a translation, that there may be conflicts between the EMF model and the information contained in the generated EAnnotations. For example, a MOF Class C may have the “isLeaf” attribute set to true, meaning that this Class may not have sub-classes. However, as there is no corresponding concept in EMF, the “isLeaf: information will appear in the EMF model as a generated EAnnotation. When attempting to reverse this mapping, it may be discovered that the corresponding EClass EC does indeed have a sub-class ESC, presumably added to the EMF model after its original translation from the MOF model. In this particular case, there are four alternative choices:

- 7) add a subclass in the MOF model and set the IsLeaf status of the parent Class to false
- 8) retain the isLeaf status of the parent Class and not create the subclass
- 9) add a subclass in the MOF model *and* retain the IsLeaf status of the parent Class
- 10) the translation fails

Option 10) is not acceptable to users; the expectation is that the tool should make its best effort at translation and provide warnings rather than fail/refuse/abort.

Option 9) creates an inconsistent MOF model which may be rejected by MOF tools.

Option 2) may create consequential problems in that the new subclass may be referenced elsewhere in the EMF model and hence all just usages would have to be elided.

Option 1) is therefore in our experience, the only viable choice.

This demonstrates our general principle that when there is an inconsistency the EMF model and the generated EAnnotations, the EMF model overrides the round-tripping information held in the generated EAnnotations. Since a well-formed EMF model will (almost always¹) translate to a well-formed MOF model, the “worst-case” scenario for “reversing” in the presence of inconsistent generated EAnnotations is to simply to translate as if it was a “native” EMF model.

6.3. EMF to MOF to EMF round-tripping

Again, this section has assumed that the original model was expressed in MOF. If the original model was expressed in EMF, the simpler nature of the EMF model compared with the MOF model makes it very easy to reverse for EMF to MOF to EMF round-tripping. Since most aspects of the EMF-to-MOF translation are 1-to-1, the reversal is quite straightforward in most situations. The only introduced model elements from the EMF model to MOF model translation (apart from Tags to contain information not present in the MOF model and to hold information about the EMF to MOF generation) will be Associations and AssociationEnds to support “opposite” EReferences for which the native MOF model to EMF mapping reduces back to two “opposite” EReferences with EAnnotations to capture the information about the Association and AssociationEnds that existed in the MOF universe. Thus, round-tripping from EMF to MOF and back to EMF is not made more complex by the presence of Tags describing the translation from EMF to MOF, as the simplest translation is effectively also the reverse translation.

6.4. Other issues

Another round-tripping problem is the use of ordering within associations within the MOF, a concept which has no meaning in EMF. Therefore, e.g. the attributes of a class could be reordered through the round-tripping, which is Bad News. So, the ordering information must be handled in a similar manner to annotations/Tags. Another one is the name of the return Parameter of a MOF Operation. Another big one is the information about the Association and its AssociationEnds.

7. Round-tripping: around and around and around

Obviously, a model and its instances may undergo many successive round-trip translations. The round-trip translations as described in Section 6 are not sufficient to cope with this. All translations from MOF to EMF and EMF to MOF produce additional information (at a minimum, the introduction of EAnnotations to document the translation). Therefore, there are important trade-offs to be made about preserving such information through every successive translation.

1. In the absence of cyclic dependencies between root packages.

If it is desired to retain all information generated in each successive translation, there is the benefit that the complete history of any model element or instance is retained and this may be important for audit purposes. However, against that, there is the complexity of maintaining this historical information as MOF Tags and EMF EAnnotations so that:

- the sequence of translations is maintained, and in particular there is no confusion between the most recent translation and any previous translation in the same “direction” (i.e. MOF to EMF or EMF to MOF), e.g. through the use of time stamps or sequence numbers
- inconsistent Tags/EAnnotations (which would have been ignored in a subsequent translation) can be recognised as such
- there is no reliance on the continued existence of earlier versions of the model in either MOF or EMF form which would make object references or names used in the Tags/EAnnotation incapable of resolution (implies information is copied into Tags/EAnnotations rather than referenced)

Due to these complexities, our current E-MORF tool does not attempt to maintain the complete history, but retains only Tags/EAnnotations associated with the most recent translation (sufficient to support the reversal of that translation). We believe that the complete evolution of models and instances should be addressed by some (or all) of the OMG’s current Requests for Proposals:

- MOF 2.0 Query/Views/Transformations [4]
- MOF 2.0 Versioning and Development Lifecycle [13]
- MOF 2.0 Facility and Object Lifecycle [14]

8. General principles for transformation

As a consequence of tackling the specific problem of translating MOF models and instances to EMF and vice versa, we have derived the following general principles about models and transformations. Fundamental to the MDA approach is the explicit modelling of relevant information that is too often hidden within an implementation. By examining E-MORF, we concluded that there were many models and principles implicit in our E-MORF translations, which deserved independent elaboration.

8.1. Having an escape hatch

The very nature of modelling is to abstract from the full complexity of the real world to some smaller set of concepts that capture one’s universe of discourse sufficient to the current purpose. The temptation for the modeller (or meta-modeller) is to simply ignore the parts of the real world outside the universe of discourse and believe that the model does not need to preserve the rest of the world in any form whatsoever. Our general principle is that every model needs an escape hatch, a part of the model where information outside of the universe of discourse can be “harmlessly” retained if desired. In MOF, this is the Tag element and in EMF, it is the EAnnotation element. When this project commenced, the EAnnotation element was not part of the EMF model, making it very difficult to store additional information “harmlessly” within an EMF model.

By “harmless”, we mean stored in such a way that it is invisible to the tools that manipulate that model (meta-model) in a “native” form. For example, in MOF and EMF, it would be harmful to store such information where it would alter the code generation of these systems. For example, without EAnnotations in EMF, the key/data pairs required would have had to be encoded in some “least harmful” way, e.g. on an EClass, one can define readonly EAttributes with default values (where the name of the EAttribute is the key and the default value contains the data (encoded as a string). Although the code generation of EMF would have resulted in unnecessary operations to get the value of these EAttributes, there would have been no impact on the overall model semantics, hence minimising the damage caused by the introduction of “out-of-model” information.

8.2. Extra information about the source and target models

Ancillary source and target models may be involved in the transformation, for example, in E-MORF, we make use of the Java model, as discussed in Section 3.1. Hence, the transformation model must support multiple models, both as sources and as targets of the transformation.

As we observed in the translation between MOF and EMF, considerable care must be taken to devise naming schemes and name mangling algorithms. For example, in order to make use of the “helpers” needed to translate MOF concepts such as “inout” Parameters to EMF, naming rules for both the Java type model and the EMF model were used implicitly.

The naming rules specific to each model involved in a transformation may need to be represented as ancillary source models to a transformation, and a mechanism employed for identifying “name” attributes within source and target models.

A criticism of MOF 1.X that was observed in the IDL generation rules as well as in the preparation of the XMI and HUTN [15] standards is the failure to distinguish “name” attributes within the model. Currently in MOF 1.X, an attribute whose purpose is to uniquely name an object within some scope is not annotated to reflect this (an omission hopefully to be corrected by MOF 2.0 [16]). When performing transformations, there is a strong and frequent desire to exploit and propagate existing naming systems in order to:

- provide a linkage between related elements in the source and target models
- retain the “natural language” semantics associated with those names

As a consequence, XMI and HUTN were forced to introduce an additional system of artificial identifiers to ensure elements were uniquely identifiable. While some models do not possess a system of naming (unique identification), many models do possess a system of naming and it is disappointing that this cannot be exploited. In HUTN, there is scope for user customisation and one aspect of this customisation is the identification of attributes that have “naming semantics” which HUTN will use in preference to its introduced identifiers to improve human readability.

In addition to knowing which attributes are “names” and what the scope of those names are, it is also important to know what are the syntactic constraints upon names in both source and target models. For example, if names are case-sensitive in the source model but not in the target model, then name mangling will have to prevent the propagation of names into the target model which differ only by case, e.g. “foo” and “FOO”. Also, it is necessary to know how to compose a qualified name, e.g. “foo::X” or “foo/X” or “X@foo” to name things in other scopes.

As well as name-mangling rules, rules for data-encoding, specific to each model, may also need to be represented as ancillary source models to a transformation, so that these rules can be applied from within transformation rules. This is exemplified by the way in which E-MORF made implicit use of a model of escape-hatch encoding rules for EMF, by using a consistent naming scheme for keys within the EAnnotations used to encode round-tripping and rule correspondence information.

8.3. Reflection in the transformation model

In translating between MOF and EMF, we observed that many translation rules were “self-aware”, suggesting the need for a reflective capability within the Transformation model.

As discussed in Section 3.2, EAnnotations are used to record the rule and sources which generated each target element. Therefore, the Transformation model should support the identification of its rules, and make this information available at runtime through introspection.

As discussed in Section 6.2, when reversing a transformation, round-tripping information may be inconsistent with the native model, hence, the Transformation model must also support the expression of “preconditions” or consistency rules that allow the inconsistencies between round-tripping information and the native model to be resolved.

One of the uses of these “preconditions” is to choose between “native” and “reversing” rules. To do this the Transformation model should support the notion of alternative or “overriding” rules.

8.4. Managing transformation history

As discussed in Section 6 and Section 7, information added to record how the translation was performed and how round-tripping should be reversed can become complex to manage. Although the escape hatches or external mechanisms used to provide support for this information will usually be model-specific, there may be a general model of annotation management that can reduce the burden on the writer of the transformation.

8.5. Building transformations

When constructing the translation between MOF and EMF, we observed that there were several approaches and styles for building a transformation.

The translation expressed by E-MORF represents a top-down style transformation, with rules matching top level packages, and working down the containment hierarchy. A top-down approach is typical of XSLT transformations as it mirrors the containment structure of an XML document. For some transformations, a bottom-up approach may be more appropriate, for example, when coalescing multiple source elements into a single target element, such as merging the many EAnnotations with single key/value pairs (described in Section 3.2) into a single EAnnotation with multiple key/value pairs. This coalescing is difficult to express using a single XSLT translation; it is easier to coalesce the EAnnotations using a subsequent XSLT translation.

It is also possible to consider the transformation from a number of orientations: source-driven, target-driven or aspect-driven. As with most XSLT-based transformations, E-MORF is a source-driven translation, with rules based on the structure of the source XMI. Each rule matches a single element from the source model, and produces a number of elements in the target models as a result. There is a strong correlation between top-down and source-driven approaches.

A target-driven transformation is structured around elements from the target models. Rules typical of this style will produce a target model element by matching a number of source model elements under associated conditions; this is also often encountered in the bottom-up style. An aspect-driven transformation may be structured according to semantic concepts rather than around the models involved in the transformation. For example, rules that generate fully-qualified names are aspect-driven as they are not specific to any one type in the source or target models.

In addition to supporting different styles and orientations, the transformation model should also allow modular transformations to be built, so that transformations may be re-used or composed to produce new transformations.

9. Implications for QVT

As DSTC's research has also involved working with generic transformation models culminating in our OMG submission [7] to the RFP on MOF Query/View/Transformation, it is illuminating to examine our generic transformation model in the light of the principles of Section 8.

9.1. Needs an escape hatch

Our QVT model does not incorporate an escape hatch mechanism. However, since there is no reason why our QVT model should not be the source or target of transformations, it should be extended to support a flexible escape hatch in the key/data pair style.

9.2. Additional information about the source and target models

Our model allows multiple source and target models to a transformation. Hence, additional information about the source and target model can be represented by additional source models to a transformation. Our submission does not define models to represent naming or escape-hatch encoding rules. However, as this information is likely to be required for most transformations, it may be prudent to standardise such models to provide a consistent mechanism for representing and accessing this additional meta-information.

9.3. Reflection in the transformation model

Our QVT model supports the following capabilities:

- identification of transformation rules
- recording the sources and targets for rule application through the use of tracking functions
- the definition of “preconditions” or consistency rules
- extension and superseding of transformation rules based on “preconditions”

However, while this information is in the transformation model, there is no direct support for reflection. A poor man's solution is to introduce the transformation model as an additional source model to the transformation, but this approach will lead to needlessly complex transformation rules compared with direct support for introspection.

9.4. Managing transformation history

The tracking functions in our QVT model are intended to record the source elements on which a target element is dependent. The tracking functions do not have an explicit identification of the transformation rule itself, but in practice there is often a correspondence between the names of the tracking function and the transformation rules (although it is not always 1-to-1). It is also left to the transformation writer to choose which aspects of the history are sufficiently interesting to record.

Although our QVT model supports the tracking functions during execution of the transformation, our current implementation does not store the tracking functions persistently after the transformation is complete. Future implementations are likely to use the tracking functions to construct a "correspondence" MOF model [17] which will be used to save the tracking data as an independent MOFlet. There is no plan at this stage to support embedding of the tracking data into the target model elements.

Our tracking functions are only concerned with the immediate past transformation; no consideration has been given to maintaining any longer history, other than as a sequence of correspondence MOFlets. Note that the correspondence MOFlet approach depends on the continued existence of all earlier versions of the source data.

9.5. Building transformations

Both source-driven, top-down and target-driven, bottom-up styles of transformation are able to be represented using our transformation model. Our use of tracking functions naturally leads to a bottom-up approach. A containment-based top-down approach is less implicitly supported by our model, as containment relationships are not treated any differently to other relationships. However, it is possible to explicitly construct a top-down transformation based on any relationship, for example, on type hierarchy. A transformation may include general rules that apply to a supertype that may be extended or superseded with more specific rules that apply to its subtypes.

Our transformation model is particularly suited to an aspect-driven approach to transformation, because object creation is implicit, allowing us to describe a particular aspect, without having to worry about the existence of the object. An aspect-driven approach typically applies to non-containment associations, in particular many-to-many relationships, such as the MOF CanRaise association between Operations and Exceptions.

Our submission does not currently address how transformations may be re-used or composed, but this is a planned area of future work. We anticipate that the notion of extending and superseding of individual transformation rules will also apply to transformations as a whole.

10. Conclusions

In addition to developing E-MORF, a practical XSLT-based tool to facilitate the interworking of the OMG and Eclipse communities, the translations between MOF and EMF have given us valuable practical insight into the requirements of a generic transformation language.

The immediate future challenge is to fully incorporate these requirements into our QVT model, for use in the OMG world. Then, using E-MORF, we intend to migrate this revised transformation model into the EMF framework to enable EMF to be used as a true model driven architecture.

Acknowledgements

The work reported in this report 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). The work was also funded in part by an IBM Eclipse Innovation Award to investigate EMF import/export from MOF and JMI.

References

- [1] OMG. “Model Driven Architecture A Technical Perspective”. OMG Document: ormsc/01-07-01, July 2001.
- [2] OMG. “Meta Object Facility (MOF) v1.3.1”. OMG Document: formal/01-11-02, Nov. 2001.
- [3] OMG. “Meta Object Facility (MOF) v 1.4”. OMG Document: formal/2002-04-03, Apr. 2002.
- [4] OMG. “MOF 2.0 Query/Views/Transformations RFP”, OMG Document: ad/02-04-10, Apr. 2002.
- [5] “EMF Developer FAQ”, www.eclipse.org/emf
- [6] “Eclipse Platform”, www.eclipse.org
- [7] DSTC, IBM, CBOP, “MOF 2.0 Query/Views/Transformations RFP”, OMG Document: ad/03-02-03, Mar. 2003.
- [8] OMG. “Unified Modeling Language v1.4”. OMG Document: formal/01-09-67, Sept. 2001.
- [9] CWM Partners, “Common Warehouse Metamodel (CWM) Specification”, OMG Documents: ad/01-02-{01, 02, 03}, Feb. 2001.
- [10] OMG, “XML Metadata Interchange v 1.2”, OMG Document: formal/2002-01-01, Jan. 2002.
- [11] W3C. “XSL Transformations (XSLT) v1.0”. W3C Recommendation: www.w3.org/TR/xslt, Nov. 1999.
- [12] Greg J. Badros, “JavaML”, www.cs.washington.edu/homes/gjb/JavaML/
- [13] OMG. “MOF 2.0 Versioning and Development Lifecycle RFP”. OMG Document: ad/02-06-23, June, 2002.
- [14] OMG. “MOF 2.0 Facility and Object Lifecycle RFP”. OMG Document: ad/03-01-35, Jan. 2003.
- [15] OMG. “Human-Usable Textual Notation”. OMG Document: ad/02-03-02, Apr. 2002.
- [16] OMG. “MOF 2.0 Core RFP”. OMG Document: ad/01-11-05, Nov. 2001.
- [17] Kerry Raymond, “The Fundamental Interconnectedness of All Things”, DSTC Symposium Paper, September 2001.
- [18] DSTC, “dMOF 1.1: an OMG Meta-Object Facility Implementation”, www.dstc.edu.au/Products/CORBA/MOF
- [19] Sun Microsystems, “Metadata Repository (MDR)”, mdr.netbeans.org

