12 The Essential MOF (EMOF) Model

12.1 Introduction

This chapter defines Essential MOF, which is the subset of MOF that closely corresponds to the facilities found in OOPLs and XML. The value of Essential MOF is that it provides a straightforward framework for mapping MOF models to implementations such as JMI and XMI for simple metamodels. A primary goal of EMOF is to allow simple metamodels to be defined using simple concepts while supporting extensions (by the usual class extension mechanism in MOF) for more sophisticated metamodeling using CMOF. Both EMOF and CMOF (defined in the next chapter) reuse the UML2 InfrastructureLibrary. The motivation behind this goal is to lower the barrier to entry for model driven tool development and tool integration.

The EMOF Model merges the Basic package from UML2 and includes additional language capabilities defined in this specification. The EMOF model merges the Reflection, Identifiers, and Extension capability packages to provide services for discovering, manipulating, identifying, and extending metadata.

EMOF, like all metamodels in the MOF 2 and UML 2 family, is described as a CMOF model. However, full support of EMOF requires it to be specified in itself, removing any package merge and redefinitions that may have been specified in the CMOF model. This chapter provides the CMOF model of EMOF, and the complete, merged EMOF model. This results in a complete, standalone model of EMOF that has no dependencies on any other packages, or metamodeling capabilities that are not supported by EMOF itself.

Note - The abstract semantics specified in "CMOF Abstract Semantics" on page 53 are optional for EMOF.

The relationship between EMOF and InfrastructureLibrary::Core::Basic requires further explanation. EMOF merges Basic with the MOF capabilities and a few extensions of its own that are described below. Ideally, EMOF would just extend Basic using subclasses that provide additional properties and operations. Then EMOF could be formally specified in EMOF without requiring package merge. However, this is not sufficient because Reflection has to introduce Object in the class hierarchy as a new superclass of Basic::Element which requires the merge. As a result of the merge, EMOF is a separate model that merges Basic, but does not inherit from it.

By using PackageMerge, EMOF is directly compatible with Basic XMI files. Defining EMOF using package merge also ensures EMOF will get updated with any changes to Basic. The reason for specifying the complete, merged EMOF model in this chapter is to provide a metamodel that can be used to bootstrap metamodel tools rooted in EMOF without requiring an implementation of CMOF and package merge semantics.

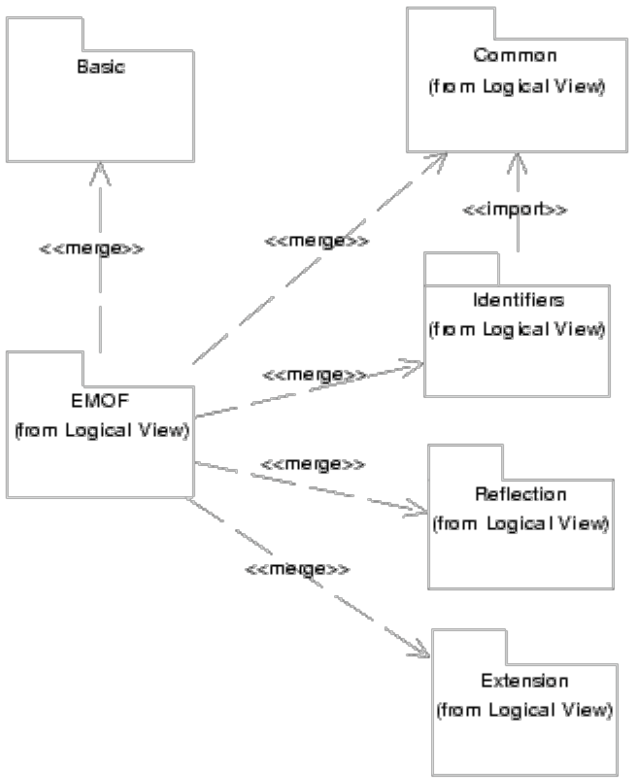


Figure 12.1 - EMOF Model- Overview

The EMOF model provides the minimal set of elements required to model object-oriented systems. EMOF reuses the Basic package from UML 2.0 InfrastructureLibrary as is for metamodel structure without any extensions, although it does introduce some constraints.

12.2 EMOF Merged Model

This section provides the complete EMOF model merged with Basic and the MOF capabilities. It is completely specified in EMOF itself after applying the package merge semantics described in Infrastructure. The description of the model elements is identical to that found in UML 2.0 Infrastructure and is not repeated here.

Basic

EMOF merges InfrastructureLibrary::Core::Basic from UML 2.0 Infrastructure. The results of the merge are given in the following diagrams. The results of merging the capabilities described in the next section are also shown in some of the diagrams.

MultiplicityElement
isOrdered : Boolean = false
isUnique : Boolean = true
lower : Integer = 1
upper : UnlimitedNatural = 1
TypedElement
Type

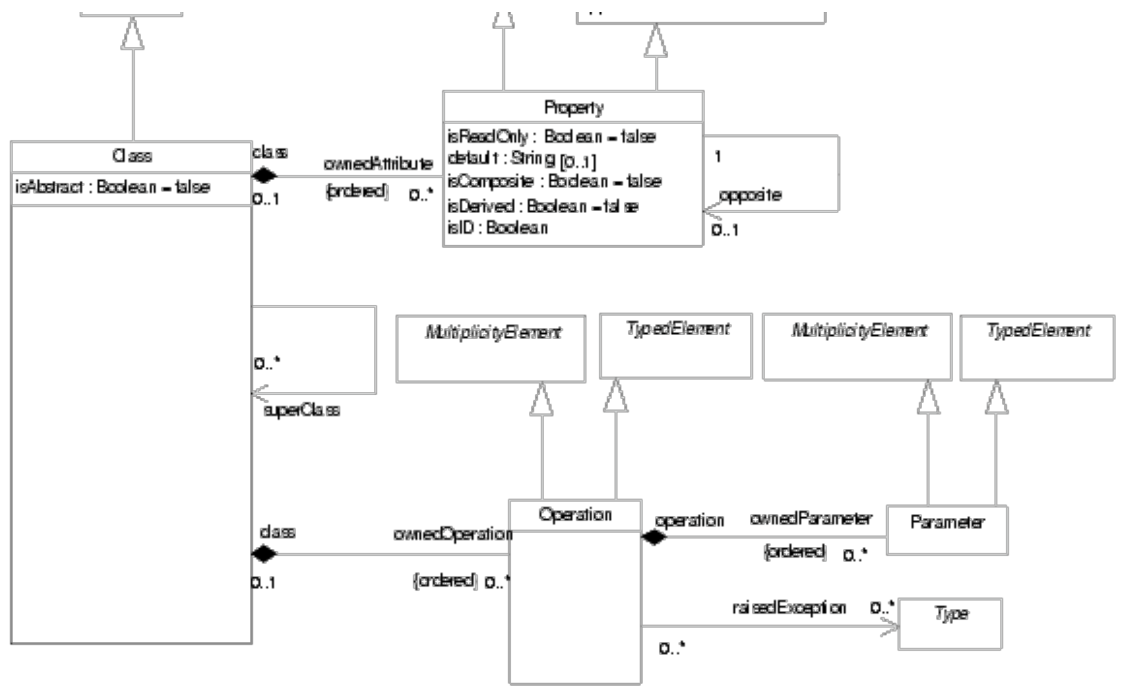


Figure 12.2 - EMOF Classes

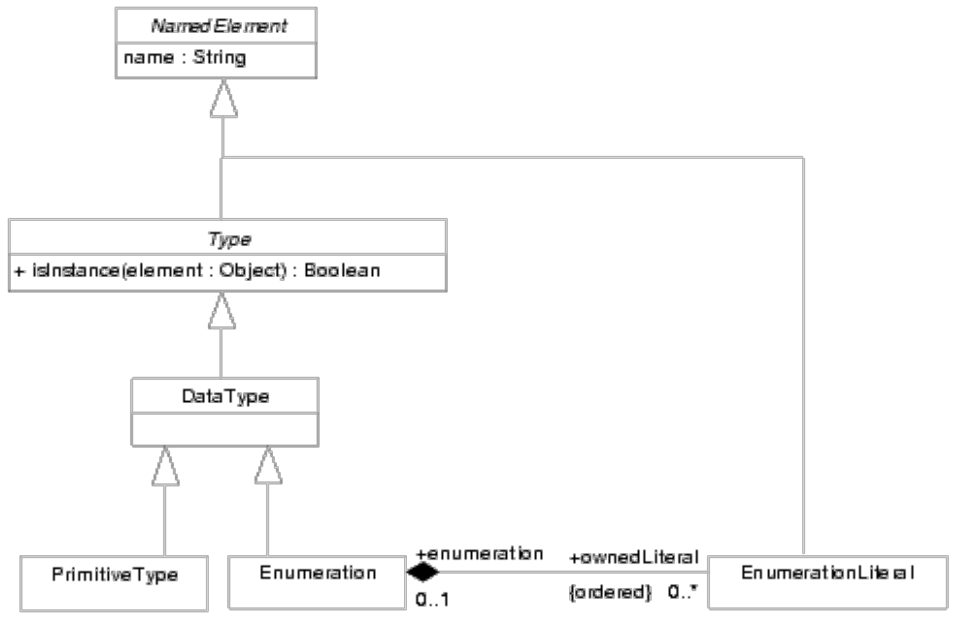
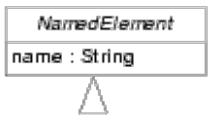


Figure 12.3 - EMOF Data Types



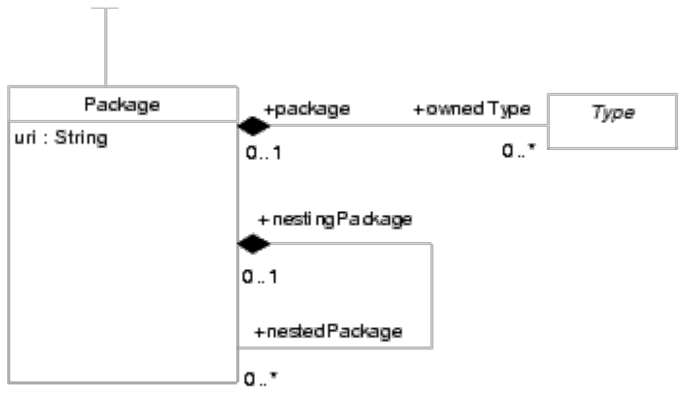


Figure 12.4 - EMOF Packages

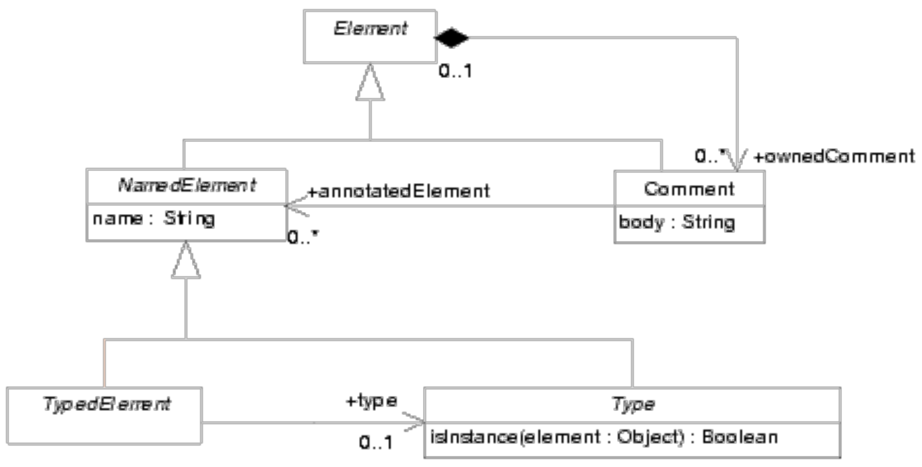


Figure 12.5 - EMOF Types

12.3 Merged Elements from MOF

The EMOF Model merges the following packages from MOF. See the capabilities chapters (9 through 11) for diagrams of the EMOF capabilities.

- Identifiers
- Reflection
- PrimitiveTypes
- Extensions

12.4 EMOF Constraints

1. The type of Operation::raisedException is limited to be Class rather than Type.
2. Notationally, the option is disallowed of suppressing navigation arrows such that bidirectional associations are indistinguishable from non-navigable associations.
3. Names are required for all Types and Properties (though there is nothing to prevent these names being automatically generated by a tool).
4. Core::Basic and EMOF does not support visibilities. All property visibilities

expressed in the UML MOF model will be ignored (and everything assumed to be public). Name clashes through names thus exposed should be avoided.
5. The definitions of Boolean, Integer, and String are consistent with the following implementation definitions:
    - Boolean: http://www.w3.org/TR/xmlschema-2/#boolean
    - Integer: http://www.w3.org/TR/xmlschema-2/#integer
    - String: http://www.w3.org/TR/xmlschema-2/#string
6. All the abstract semantics specified in the Chapter 15, "CMOF Abstract Semantics" are optional for EMOF.
7. X is an object and therefore supports reflection if MOF::Object.isInstance(X)==true.

12.5 EMOF Definitions and Usage Guidelines for the Basic Model

When the EMOF package is used for metadata management the following usage rules apply.

Package

- Although EMOF defines Package and nested packages, EMOF always refers to model elements by direct object reference. EMOF never uses any of the names of the elements. There are no operations to access anything by NamedElement::name. Instances of EMOF models may provide additional namespace semantics to nested packages as needed.

Properties

- All properties are modified atomically.
- When a value is updated, the old value is no longer referred to.
- Derived properties are updated when accessed or when their derived source changes as determined by the implementation. They may also be updated specifically using set() if they are updateable

Type==DataType

- The value of a Property is the default when an object is created or when the property is unset.
- Properties of multiplicity upper bound > 1 have empty lists to indicate no values are set. Values of the list are unique if Property.isUnique==true.
- "Identifier" properties are properties having property.idID==true.

Type==Class

- Properties of multiplicity upper bound == 1 have value null to indicate no object is referenced.
- Properties of multiplicity upper bound > 1 have empty lists to indicate no objects are referenced. Null is not a valid value within the list.
- EMOF does not use the names of the properties, the access is by the Property argument of the reflective interfaces. It does not matter what the names of the Properties are, the names are never used in EMOF. There is no special meaning for

having similar names. The same is true for operations, there is no use of the names, and there is no name collision, override, or redefinition semantics. EMOF does not have an invoke method as part of the reflective interface, so there are no semantics for calling an EMOF operation. The names and types of parameters are never compared and there is no restriction on what they can have singly or in combination. Other instances of EMOF metamodels, or language mappings, such as JMI2, may have additional semantics or find that there are practical restrictions requiring more specific definitions of the meaning of inheritance.

Property::isComposite==true

- An object may have only one container.
- Container properties are always multiplicity upper bound 1.
- Only one container property may be non-null.
- Cyclic containment is invalid.
- If an object has an existing container and a new container is to be set, the object is removed from the old container before the new container is set.
- Adding a container updates both the container and containment properties on the contained and containing objects, respectively. The opposite end is updated first.
- The new value is added to this property.

Property::isComposite==false, Bidirectional

- The object is first removed from the opposite end of the property.
- If the new value's opposite property is of multiplicity upper bound == 1, its old value is removed.
- This object is added to the new value's opposite property.
- The new value is added to this property.

Object

- Everything that may be accessed by MOF is an Object.
- An Object that is not also an Element may be an instance of one DataType.

TOC < PREV NEXT > INDEX