December 2019

OBJECT MANAGEMENT GROUP

Metamodel Extension Facility (MEF)

Version 1.0 - Beta 1

OMG Document Number:	ptc/2020-03-01
Specification URL:	https://www.omg.org/spec/MEF/1.0/
Machine-readable Files:	
(normative)	https://www.omg.org/spec/MEF/20190901/MEF.xmi
(informative / ancillary)	ad/2019-08-03 Ancillary Files - OMG Members only

---

This OMG document replaces the submission document (ad/19-08-01) and associated errata document (ad/19-09-04). It is an OMG Adopted Beta Specification and is currently in the finalization phase. Comments on the content of this document are welcome, and should be directed to issues@omg.org by January 24, 2020.

You may view the pending issues for this specification from the OMG revision issues web page: https://issues.omg.org/issues/lists.

The FTF Recommendation and Report for this specification will be published in December 2019. If you are reading this after that date, please download the available specification from the OMG Specifications Catalog.

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification. Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means–graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems–without permission of the copyright owner.

OMG's Issue Reporting Procedure

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page, under Documents, Report a Bug/Issue (https://www.omg.org/report_issue.)

Contents

Preface

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG's specifications include: UML®(Unified Modeling Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Meta-model); and industry-specific standards for dozens of vertical markets. More information on the OMG is available at https://www.omg.org.

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. All OMG Specifications are available from the OMG website at: http://www.omg.org/spec

Specifications are organized by categories; listing selected major categories below:

Fundamental Information Modeling Technologies

- Unified Modeling Langue (UML)
- Systems Engineering Modeling Language (SysML)
- Interface Definition Language (IDL)

Domain-Specific Technologies

- Business and Enterprise Modeling
- Industrial Engineering and Manufacturing
- Space, Control and Transportation Technologies
- Robotics
- Software and Systems Modernization
- Information Security

- Healthcare
- Retail

Middleware and Real-Time Technologies

- Common Object Request Broker (CORBA)
- Data Distribution Service (DDS)
- Middleware Services
- Real-Time and Embedded Systems
- Signal Processing and Communication

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarter
109 Highland Avenue
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: pubs@omg.org

Certain OMG specifications are also available as ISO standards. Please consult http://www.iso.org

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Times/Times New Roman - 10 pt.: Standard body text

Helvetica/Arial - 10 pt. Bold: OMG Interface Definition Language (OMG IDL) and syntax elements.

`Courier - 10 pt. Bold:` Programming language elements.

Helvetica/Arial - 10 pt: Exceptions

NOTE: Terms that appear in italics are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

Issues

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page https://www.omg.org, under Documents, Report a Bug/Issue.

1 Scope

The Metamodel Extension Facility (MEF), part of the Meta Object Facility (MOF) family of specifications, provides structure, operations and interchange for adding extended structured capabilities to existing metamodels.

It builds on, and provides more control compared to, the Semantic Structures for MOF (SMOF) specification: specifically it allows the same capabilities provided by UML Profiles to be applied to any other metamodel. For example, it makes explicit the application of metamodels to a model and allows the application of additional metaclasses to be constrained.

And, for UML, it provides a more flexible and convenient Profile application, manipulation and interchange capability, while remaining compatible with existing UML (2.5.x) Profile definitions. By making use of SMOF multiple classification it avoids the need to create an additional element and link for each attached Stereotype.

2 Conformance

This specification defines the following conformance points:

- *UML Profile MEF Conformance*: Conformant products must provide exactly the profile definition and application modeling capabilities specified by UML, though they must also provide import and export of models in the XMI format specified in this MEF specification (this may be instead of or in addition to the UML format).
- *Non-filtering UML Profile MEF Conformance*: As for UML Profile MEF Conformance but without support for the profile filtering capability specified by UML. In other words the isStrict parameter to applyProfile() is assumed to be false.
- *Non-Profile MEF Conformance*: Conformant products need only support the ability to apply and unapply metamodels to models: they do not need to support Profiles and Stereotypes.
- *Full MEF Conformance*: Conformant products shall implement all parts of this specification.

3 References

3.1 Normative References

The following normative documents contain provisions which, through reference in this text, constitute provisions of this specification. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply.

[MOF] Meta Object Facility (MOF) Core, version 2.5.1, OMG Specification
https://www.omg.org/spec/MOF/2.5.1

[SMOF] MOF Support for Semantic Structures (SMOF), version 1.0, OMG Specification
https://www.omg.org/spec/SMOF/1.0

[UML] Unified Modeling Language (UML), version 2.5.1, OMG Specification
https://www.omg.org/spec/UML/2.5.1

[XMI] XML Metadata Interchange (XMI), version 2.5.1, OMG Specification
https://www.omg.org/spec/XMI/2.5.1

3.2 Non-normative References

None.

4 Terms and Definitions

For the purposes of this specification, the following terms and definitions apply.

Profile
A metamodel designed to extend a reference metamodel with the purpose of adapting the metamodel to a specific platform or domain. As such it is more constrained than a general metamodel.

Stereotype
A metaclass designed to be added to an element already classified with another metaclass (from a specified set).

5 Symbols

The following symbols and/or abbreviations are used throughout this specification.

None.

6 Additional Information

(informative)

6.1 How to read this Specification

Clause 7 is an informative overview of the principles of operation. Clause 8 defines the abstract architecture. As such it shows also the positioning of the Metamodel Extension Facility in relation to the MOF Facility and Object Lifecycle and MOF Versioning and Development Lifecycle specifications, even though those specifications are *not* used or referenced by this specification. Clause 9 specifies the new metaclasses and operations. Clause 10 provides a step-by-step example of how to use the Metamodel Extension Facility.

All clauses of this document are normative unless explicitly marked "(informative)". The marking "(informative)" of a particular clause also applies to all contained sub-clauses of that clause.

6.2 Acknowledgments

The following organizations submitted this specification:

- 88solutions Corporation
- Adaptive, Inc
- Microsoft Corporation
- Model-Driven Solutions, Inc
- No Magic, Inc
- SOFTEAM Group

7 Principles of Operation

(informative)

Creating information models using a modeling language, like the Unified Modeling Language (UML), means to create an arrangement of representations of modeling language elements. In a UML Class Diagram, this would be an arrangement of class boxes, connected by representations of relationship elements, like associations, dependencies, or generalizations. Such a model is logically situated at the model level, also referred to as the "M1 metalevel". The set of available model elements to create M1 models is defined by another model logically situated above the model level at the "M2 metalevel", and is usually called the (modeling language) metamodel. Metamodels are comparably small, use only a limited set of modeling elements and are usually closed, except for dedicated extension mechanisms, if any. OMG provides the Meta Object Facility (MOF) as the common modeling language and modeling environment to specify metamodels. This MOF modeling language, though it reuses a subset of UML, is logically situated at the "M3 metalevel", and is self-defining (reflective), so no further metalevel is needed.

Each model element at a certain metalevel, for example the definition (often called "type") of a user-created UML element of type Class or type Association at M1 level is defined by (or is an instance of) a class named "Class" or a class named "Association" at the M2 level. These classes are typically referred to as "metaclasses" and are defined by the MOF language ("metametamodel") at level M3. While the model management capabilities of the MOF Core and its extension by the MOF Support for Semantic Structures (SMOF) are primarily intended to construct and maintain metamodels, the provided capabilities can be equally applied to any metalevel. The capabilities of MOF can be used to manipulate and/or extend any metamodel, however not every metamodel is prepared to tolerate such manipulations without becoming inconsistent or invalid. A common approach to control alterations and/or extensions of metamodels is the use of "Profiles". A profile is a collection of metamodel elements, together with related rules, constraints, and a well defined mechanism for applying the profile to the base model to achieve the intended alteration or extension effect. This mechanism is most prominent in UML and UML-derived languages like the Systems Engineering Modeling Language (SysML), but is also a common approach used in many modeling systems. MOF itself lacks a Profile concept, this is now provided by the Metamodel Extension Facility (MEF).

To freely allow the application and un-application (removal) of Profiles, UML provides a form of metaclass extension called "Stereotype". Stereotypes are the UML-way to add additional classifications to regular UML elements, however burdened with limitations and restrictions resulting from the underlying mechanisms of UML. Among others, Stereotypes can only *extend existing* metaclasses, they cannot represent new independent metaclasses. Even if they are defined to extend multiple metaclasses, they can extend only one of those metaclasses at any given point of time. Also, Stereotypes can only subclass other Stereotypes, not regular metaclasses, they can only participate in binary associations, and more restrictions. See the Profile clause of the UML specification [UML] for the full list of restrictions. The root of these restrictions is the fact that Stereotypes are defined at the M1 level, using M1 level capabilities for creation and management of M2 level constructs. SMOF, on the other hand, can freely alter the classifications of any element at any metalevel with nearly no restrictions. MEF uses the the power of SMOF while closely resembling the UML Profile structure and semantics. MEF Profiles therefore replace UML Profiles to eliminate the Stereotype-born restrictions while preserving the specific profile semantics. Due to the close resemblance, and identical abstract syntax, MEF Profiles may reuse the graphical notation, and UML tooling, of Stereotype definition as a pure matter of convenience, while the actual metaclass extension will be the result of a MEF or SMOF operation. MEF Profiles are pure MOF and SMOF metamodels, and have no dependency on UML (besides the portion of the metamodel shared between MOF and UML, see the MOF Core specification [MOF] for details). MEF Profiles may therefore be used with any MOF metamodel.

8 Abstract Syntax Architecture

8.1 Introduction

MOF shares part of the UML metamodel, adding reflection, factory, and extended model management. Further detail on this is laid out in the MOF Core specification [MOF]. SMOF extends MOF Core with the ability to dynamically reclassify elements.

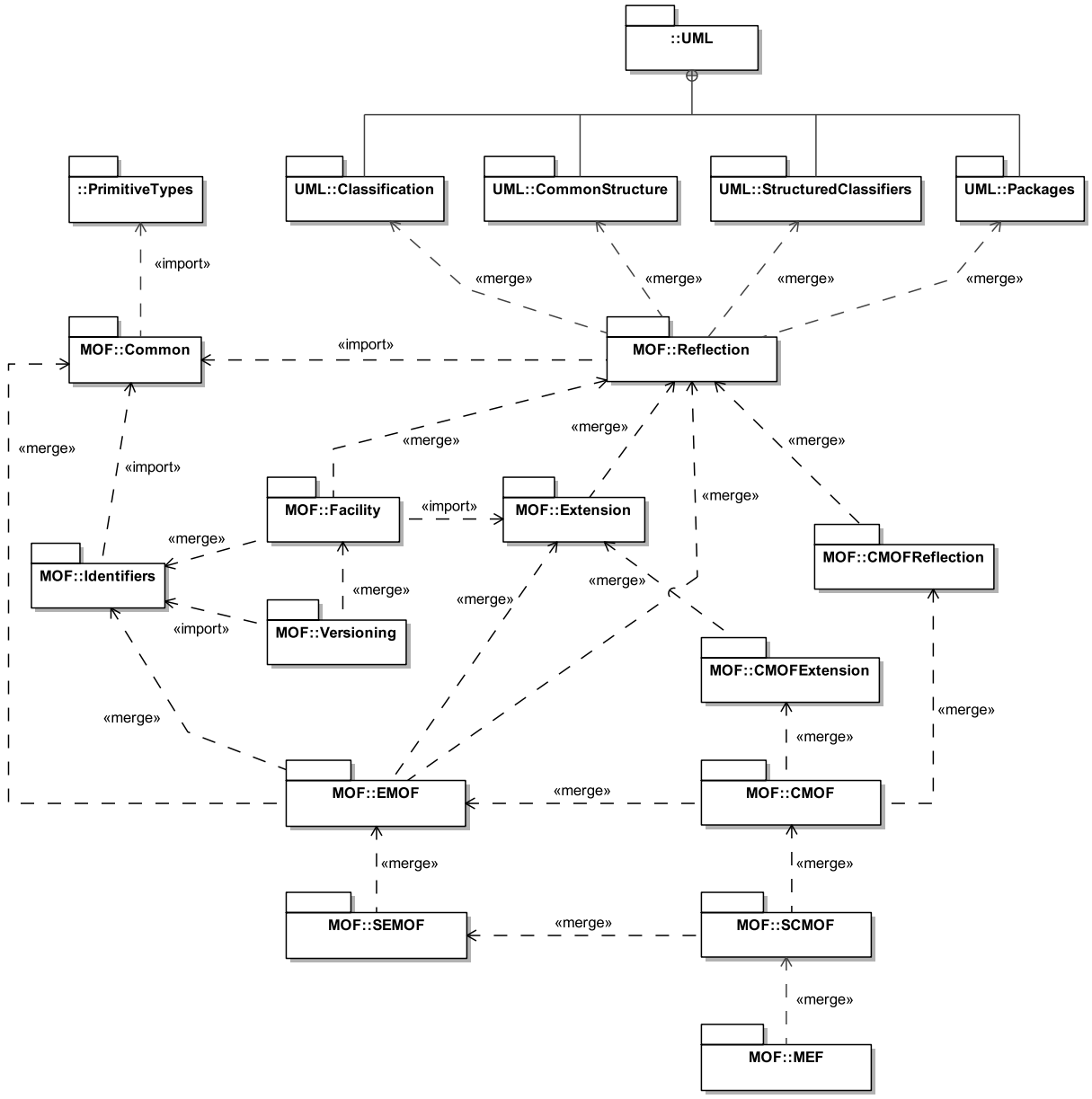

Figure 8.1: All MOF Packages

This specification provides the new package MOF::MEF, which merges with SCMOF. This new package provides a version of Profile that complements the implicit profile application and removal operations with a pair of explicit

operations that perform profile application and removal by executing a sequence of SMOF reclassification operations. Package MOF::MEF also extends MOF::SEMOF::Element with new operations required by the explicit profile application and removal operations. The extensions to Profile and Element are incorporated using PackageMerge.

8.2 MOF Family of Specifications

The Package diagram in Figure 8.1 provides an architectural overview of the family of MOF specifications, and their interrelationships. Since version 2.4, MOF and UML share the underlying metamodel. Details about this sharing can be found in the description of the Essential MOF (EMOF) and Complete MOF (CMOF) within the MOF Core specification [MOF]. UML 2.5 eliminated the split of UML into Infrastructure and Superstructure and simplified the package structure of the UML metamodel. In addition, all UML sub-packages are imported into the top-level package UML. The Package diagram in Figure 8.1 shows that only the four sub-packages UML::Classification, UML::CommonStructure, UML::StructuredClassifiers, and UML::Packages contain metaclasses shared with, and extended by MOF. The extensions provided by MOF are introduced into the shared metamodel by applying PackageMerge.

8.2.1 MOF Core

The packages MOF::Common, MOF::Reflection, MOF::Identifiers, and MOF::Extension constitute the inner core of MOF, and lead to the basic MOF functionality, the Essential MOF (package MOF::EMOF). EMOF is then expanded to the full capabilities of the Complete MOF (package CMOF) by merging the packages MOF::CMOFReflection and MOF::CMOFExtension into EMOF. See the MOF Core specification [MOF] for details.

8.2.2 SMOF

The MOF Support for Semantic Structures (SMOF) specification extends EMOF with the ability to dynamically reclassify and/or multiple-classify any element. These capabilities are added to EMOF via a single structural change and additional operations on Element. This leads to package SEMOF, and with the additional CMOF capabilities merged in, to package SCMOF. See the MOF Support for Semantic Structures specification [SMOF] for details.

8.2.3 MEF

This document contains the Metamodel Extension Facility specification, which extends SCMOF with additional operations on Element, Package and Factory to provide the metamodel management and extension capabilities using the combined capabilities of MOF, SMOF and MEF.

8.2.4 Facility and Versioning

The packages for these two members of the MOF family of specification are shown in Figure 8.1 only for completeness and for their positioning relative to other MOF packages. Facility and Versioning are *not directly used or referenced* by the Metamodel Extension Facility (MEF).

9 Metamodel Extension Facility

9.1 Introduction

The Metamodel Extension Facility (MEF) replicates the metamodel extension mechanism provided by UML, using Profiles and Stereotypes, for any MOF metamodel, including, but not limited to, UML. Replacing the traditional stereotype application method of UML by SMOF reclassification operations, nearly all restrictions on Stereotype can be removed. MEF is independent of UML, reusing the Stereotype definition abstract and concrete syntaxes is a pure convenience. From MEF perspective, a Stereotype is a regular metaclass with a specific association to one or multiple base-metaclasse(s).

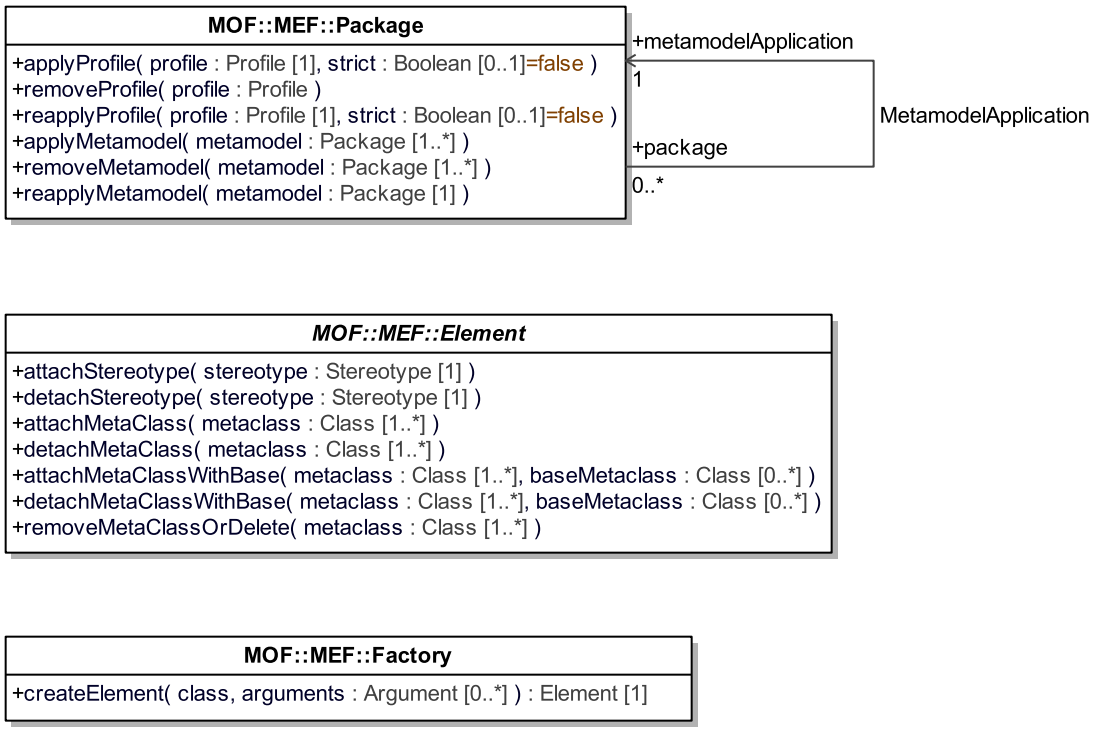

Figure 9.1: MEF extension to MOF and SMOF

9.2 Class Descriptions

9.2.1 MOF::MEF::Package

MEF::Profile extents UML::Package (from the metamodel shared between UML and MOF) with a pair of operations for the explicit application and removal of Profiles following the UML extension semantics, and a second generic pair of operations to add or remove MOF metamodels. These four operations make Profile-based and Profile-alike extensions available to all MOF metamodels. If used with UML, they relax restrictions imposed by the native UML Profile operations.

applyProfile()

```
applyProfile( profile : Profile[1], strict : Boolean[0..1] = false )
```

The applyProfile() operation is defined on MOF::MEF::Package and applies the Profile specified in the profile parameter to the current Package. If the optional parameter "strict" is present and set to "strict = true", then the profile is applied in strict mode, which will invoke the filtering rules as specified in the UML specification [UML]. The profile and the package are connected via an instance of ProfileApplication, with links to the profile and the package. Note, the Profile is not automatically applied to nested and imported packages, it must be applied to these individually. Reapplying the same profile has no effect, even if the profile has changed. To correctly reapply a profile, the profile must be reapplied using the reapplyProfile operation (see below), which performs additional consistency checks. For each Stereotype owned by the profile that has an Extension with "isRequired = true", the whole package is searched for elements classified by the base-metaclass(es) associated with the Stereotype. For each element found, the Stereotype classification is then added to that element using the MEF operation attachStereotype() with the Stereotype as argument.

removeProfile()

```
removeProfile( profile : Profile[1] )
```

The removeProfile() operation is defined on MOF::MEF::Package and removes the Profile specified in the profile parameter from this Package. All instances of the stereotypes owned by the specified profile are detached from elements in this package using the detachStereotype() operation, which will in turn invoke the removeMetaClass() SMOF operation to remove the classification associated with the Stereotype. All non-Stereotype elements inserted into the package during Profile application are removed. The ProfileApplication linkage between the profile and package is destroyed.I

reapplyProfile()

```
reapplyProfile( profile : Profile[1], strict : Boolean[0..1] = false )
```

The reapplyProfile() operation is defined on MOF::MEF::Package and applies the Profile specified in the profile parameter to the current Package. If the optional parameter "strict" is present and set to "strict = true", then the profile is applied in strict mode, which will invoke the filtering rules as specified in the UML specification [UML]. The reapplyProfile operation will compare the profile provided in the profile argument with the profile discovered through the connected instance of ProfileApplication, and perform actions to preserve consistency, if required. Note, the Profile is not automatically applied to nested and imported packages, it must be applied to these individually. Reapplying the same profile has no effect, even if the profile had changed. For each Stereotype owned by the profile that has an Extension with "isRequired = true", the whole package is searched for elements classified by the base-metaclass(es) associated with the Stereotype. For each element found, the Stereotype classification is then added to that element using the MEF operation attachStereotype() with the Stereotype as argument.

applyMetamodel()

```
applyMetamodel( metamodel : Package[1])
```

The applyMetamodel() operation is defined on MOF::MEF::Package and applies the metamodel specified in the metamodel parameter to the current Package. The metamodel and the package are connected via an instance of the MetamodelApplication Association, creating a link from the target package to the package containing the metamodel to be applied. Note, the metamodel is not automatically applied to nested and imported packages, it must be applied to these individually. Reapplying the same metamodel has no effect, even if the metamodel had changed. To correctly reapply a metamodel, the metamodel must be reapplied using the reapplyMetamodel operation, which performs additional consistency checks.

removeMetamodel()

```
removeMetamodel( metamodel : Package[1] )
```

The removeMetamodel() operation is defined on MOF::MEF::Package and removes the metamodel specified in the profile parameter from this Package. All classifications by metaclasses contained in the metamodel to be removed are detached from elements in this package using the removeMetaClassOrDelete operation, which will remove the classification associated with the metaclass, or will deep delete the element and all its dependent elements if the removed classification was the final classification of the element. All non-metaclass elements inserted into the package during metamodel application are removed. The link between the metamodel and package is destroyed.

reapplyMetamodel()

```
reapplyMetamodel( metamodel : Package[1] )
```

The reapplyMetamodel() operation is defined on MOF::MEF::Package and reapplies the metamodel specified in the metamodel parameter to the current Package. The reapplyMetamodel operation will compare the metamodel provided in the metamodel argument with the metamodel discovered through the established link as instance of the MetamodelApplication Association, and perform actions to preserve consistency, if required. Note, the metamodel is not automatically applied to nested and imported packages, it must be applied to these individually.

9.2.2 MOF::MEF::Element

MOF::MEF::Element extends MOF::SEMOF::Element with additional classification operations related to the profile-based extension mechanism.

attachStereotype()

```
attachStereotype( stereotype : Stereotype[1] )
```

This operation adds the classification by a Stereotype to the Element. Stereotypes consisting of one or more base-metaclasses and the Stereotype metaclass. The element must already be classified by the base-metaclass, or one of the set of base-metaclasses if the Stereotype extends multiple base-metaclasses. For MEF, a Stereotype is a regular metaclass, the restrictions on UML Stereotypes do not apply. As the first step, the attachStereotype() operation verifies that the element is in fact classified by the, or one of the, base-metaclass(es) of the Stereotype. If this is not the case, then that is an error situation and no action is performed. If the element is classified by the base-metaclass, or one of the

base metaclasses, then the classification by the Stereotype is added using the addNewMetaClass() SMOF operation. In case the Stereotype classification is already present at the element, then invocation of addNewMetaClass() by the attachStereotype() operation causes no effect.

detachStereotype()

```
detachStereotype( stereotype : Stereotype[1] )
```

This operation removes the classification by a Stereotype from the Element. As first step, the classification by the Stereotype *and* the base-metaclass of the Stereotype, or one of the base-metaclasses, is verified to ensure that the element is in fact classified by the Stereotype. Then the classification introduced by the Stereotype is removed using the SMOF operation removeMetaClass. The classification by the applicable base-metaclass is *not* removed.

attachMetaClass()

```
attachMetaClass( metaclass : Class[1..*] )
```

This operation adds the classification by one or multiple metaclass(es) to the element. Every Metaclass listed in the metaclass parameter is tested if it is the base-metaclass of a {required} Stereotype. If so, then the classification by that Stereotype is also added. The actual reclassification of the element is performed by invoking the SMOF operation addNewMetaClass(). In case the element is already classified by any of the metaclasses listed in the metaclass parameter of attachMetaClass(), or by any of the implicit Stereotype classifications, than those classifications are ignored and not added again by invoking the addNewMetaClass() SMOF operation.

attachMetaClassWithBase()

```
attachMetaClass( metaclass : Class[1..*], baseMetaclass : Class[1..*]  )
```

This operation is identical to attachMetaClass(), except that the new classifications are only added to elements that are already classified by one of the base-metaclasses listed in the baseMetaclass parameter. This initial filtering step allows targeted bulk reclassification of elements. The basemetaclass classifications are not affected in any way.

detachMetaClass()

```
detachMetaClass( metaclass : Class[1..*] )
```

This operation removes the classification by one or multiple metaclass(es) from the element. Every Metaclass listed in the metaclass parameter is tested if it is the base-metaclass of a {required} Stereotype. If so, then the classification by that Stereotype is also removed. The actual reclassification of the element is performed by invoking the SMOF operation removeMetaClassOrDelete(), which is a variant of the SMOF operation removeMetaClass(). While the original SMOF operation removeMetaClass() operation does not allow the removal of *all* classifications from an element, and would signal an error in that attempt, the remove MetaClassOrDelete() operation performs a deep delete of the element when its last classification is removed.

detachMetaClassWithBase()

```
detachMetaClassWithBase( metaclass : Class[1..*], baseMetaclass : Class[1..*] )
```

This operation is identical to detachMetaClass(), except that the removal of classifications is only performed on elements that are classified by one of the base-metaclasses listed in the basemetaclass parameter. This initial filtering step allows targeted bulk reclassification of elements. There is no restriction to include metaclasses used for filtering (listed in the basemetaclass parameter) also in the list of metaclasses provided to parameter metaclass, specifying the classifications to be removed. The removal of classifications is performed by the removeMetaClassOrDelete() SMOF operation, causing a deep delete of an element after all its classifications are removed.

removeMetaClassOrDelete()

```
removeMetaClassOrDelete( metaclass : Class[1..*] )
```

This operation is a variation of the removeMetaClass() SMOF operation. Like the original operation, it will remove the classifications by the Metaclasses listed in the metaclass parameter from the element it is executing on. However, instead of refusing to remove the last remaining classification of the element, and signalling an error condition on that attempt, removeMetaClassOrDelete() will initiate a deep delete of the element, removing the element itself and all compositely contained elements from the model.

9.3 MOF::MEF::Factory

MEF extends the semantics of the createElement operation defined by MOF::CMOFReflection to make it aware of potential Stereotypes that have "isRequired = true" set on their Extension.

9.3.1 createElement

```
createElement( class, arguments : Argument [0..*] ) : Element [1]
```

This operation extends the MOF::CMOFReflection::Factory::createElement() operation with a test checking if the specified metaclass in the class parameter is the base-metaclass of a {required} Stereotype. If so, then the classification by that Stereotype is also added. The actual reclassification of the element after creation is performed by invoking the SMOF operation addNewMetaClass().

10 Example

(informative)

10.1 Introduction

This informative clause demonstrates a step-by-step example of the use of MEF capabilities, utilizing a test profile developed by the Model Interchange Working Group of the Object Management Group. The model is shown in Figure 10.1 below, the application of the MEF capabilities is then detailed in several steps. Prefix "TP" will be used for the test profile.

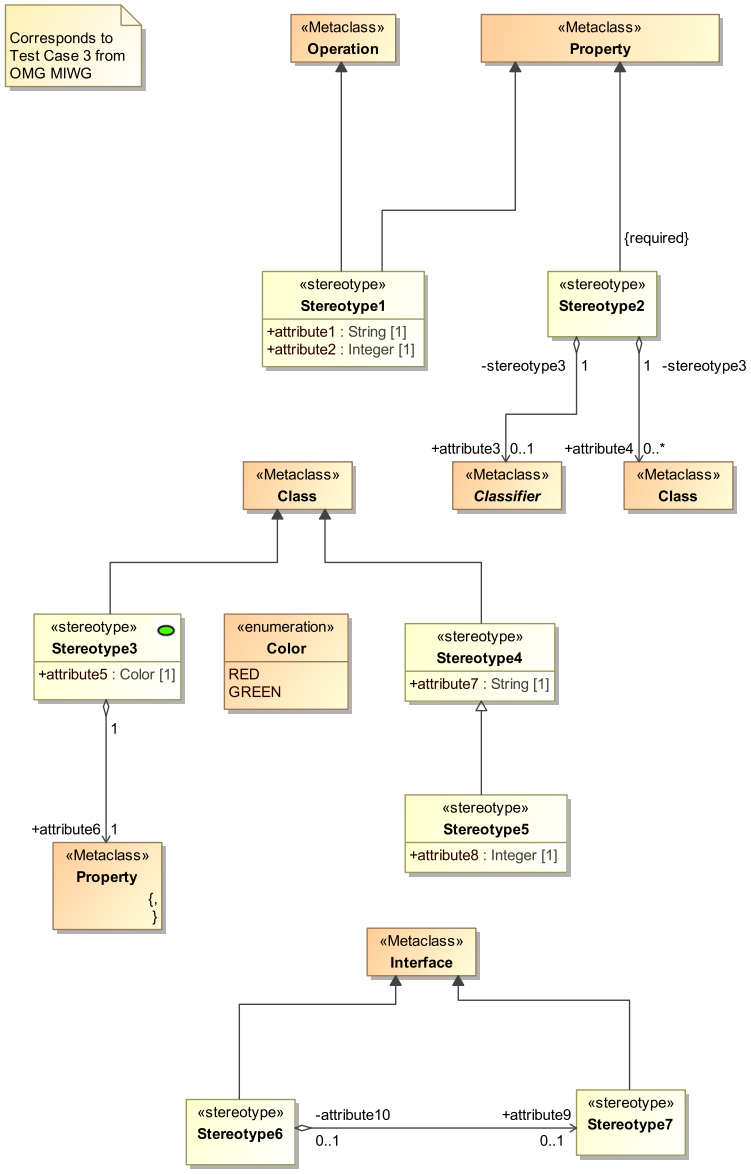

Figure 10.1: MIWG Test Case 3: A Profile

For each step, the user action is shown first, then the resulting model as instance diagram, and as the XMI serialization. Changes in the XMI resulting from the corresponding step are shown in bold font.

10.2 Step-by-Step Example

10.2.1 Step 1

Create a Package and apply the Test Profile TP.

```
f a MOF::Factory
pkg = f.createElement(UML::Package)
pkg.applyProfile(TP)
```

MEF and SMOF Operations for Step 1

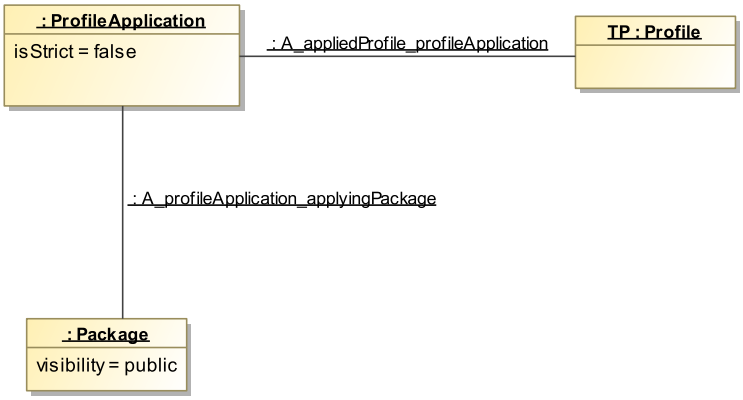

Figure 10.2: Resulting model for Step 1

```
<uml:Package xmi:id="pkg1" xmi:uuid="omg.org/mef/example/pkg1"
             xmi:type="uml:Package"/>
    <packagedElement xmi:id="pa1" xmi:uuid="omg.org/mef/example/pkg1/pa1"
                     xmi:type="uml:ProfileApplication">
        <appliedProfile href="omg.org/mef/example/TP"/>
    </packagedElement>
</uml:Package>
```

XMI Serialization for Step 1

10.2.2 Step 2

Create a Property (instance of the UML metaclass)(Stereotype2 is automatically added)

```
f a MOF::Factory
p = f.createElement(UML::Property)
(implicit p.attachStereotype(TP::Stereotype2))
```

MEF and SMOF Operations for Step 2

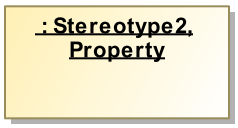

Figure 10.3: Resulting model for Step 2

(Note: the top level Package is not shown in the following and subsequent XMI listings)

```
<packagedElement xmi:id="e1" xmi:uuid="omg.org/mef/example/e1"
                 xmi:type="uml:Property"/>
<tp:Stereotype2 xmi:id="p1" xmi:uuid="omg.org/mef/example/e1"
                xmi:type="tp:Stereotype2"/>
```

XMI Serialization for Step 2

10.2.3 Step 3

Set value for attribute4 to a new random class named Class1

```
c = f.createElement(UML::Class)
c.setValue(name, "Class1")
```

MEF and SMOF Operations for Step 3

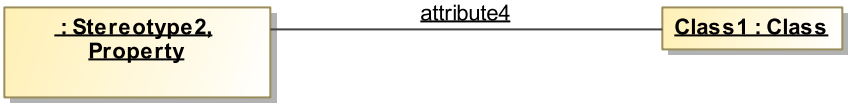

Figure 10.4: Model for Step 3

```
<packagedElement xmi:id="e1" xmi:uuid="omg.org/mef/example/e1"
                 xmi:type="uml:Property"/>
<packagedElement xmi:id="c1" xmi:uuid="omg.org/mef/example/c1"
                 xmi:type="uml:Class" name="C1ass1" />
<tp:Stereotype2 xmi:id="p1" xmi:uuid="omg.org/mef/example/e1"
                xmi:type="tp:Stereotype2">
  <attribute4 xmi:idref="c1"/>
</tp:Stereotype2>
```

XMI Serialization for Step 3

10.2.4 Step 4

Add Stereotype1 as an additional stereotype

```
p.attachStereotype(Stereotype1)
```

MEF and SMOF Operations for Step 1

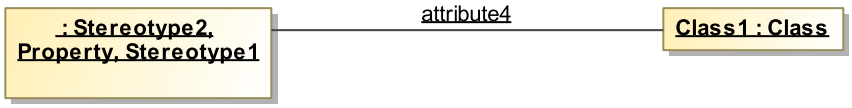

Figure 10.5: Model for Step 4

```
<packagedElement xmi:id="e1" xmi:uuid="omg.org/mef/example/e1"
                 xmi:type="uml:Property"/>
<packagedElement xmi:id="c1" xmi:uuid="omg.org/mef/example/c1"
                 xmi:type="uml:Class" name="C1ass1" />
 <tp:Stereotype2 xmi:id="p1" xmi:uuid="omg.org/mef/example/e1"
                 xmi:type="tp:Stereotype2">
   <attribute4 xmi:idref="c1"/>
</tp:Stereotype2>
<tp:Stereotype1 xmi:id="p2" xmi:uuid="omg.org/mef/example/e1"
                xmi:type="tp:Stereotype1"/>
```

XMI Serialization for Step 4

10.2.5 Step 5

Set attribute1 of Stereotype1

```
p.setValue(attribute1, "a string")
```

MEF and SMOF Operations for Step 5

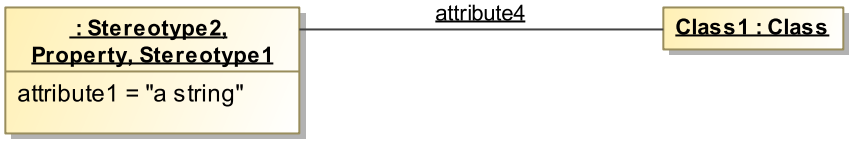

Figure 10.6: Model for Step 5

```
<packagedElement xmi:id="e1" xmi:uuid="omg.org/mef/example/e1"
                 xmi:type="uml:Property"/>
<packagedElement xmi:id="c1" xmi:uuid="omg.org/mef/example/c1"
                 xmi:type="uml:Class" name="C1ass1" />
<tp:Stereotype2 xmi:id="p1" xmi:uuid="omg.org/mef/example/e1"
                xmi:type="tp:Stereotype2">
  <attribute4 xmi:idref="c1"/>
</tp:Stereotype2>
<tp:Stereotype1 xmi:id="p2" xmi:uuid="omg.org/mef/example/e1"
                xmi:type="tp:Stereotype1" attribute1="a string"/>
```

XMI Serialization for Step 5

10.2.6 Step 6

Set name of the Property to E1

```
p.setValue(name, "E1")
```

MEF and SMOF Operations for Step 6

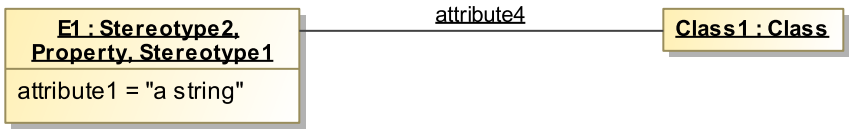

Figure 10.7: Model for Step 6

```
<packagedElement xmi:id="e1" xmi:uuid="omg.org/mef/example/e1"
                 xmi:type="uml:Property" name="E1"/>
<packagedElement xmi:id="c1" xmi:uuid="omg.org/mef/example/c1"
                 xmi:type="uml:Class" name="C1ass1" />
<tp:Stereotype2 xmi:id="p1" xmi:uuid="omg.org/mef/example/e1"
                xmi:type="tp:Stereotype2">
  <attribute4 xmi:idref="c1"/>
</tp:Stereotype2>
<tp:Stereotype1 xmi:id="p2" xmi:uuid="omg.org/mef/example/e1"
                xmi:type="tp:Stereotype1" attribute1="a string"/>
```

XMI Serialization for Step 6

10.2.7 Step 7

Change the base class to Operation

```
p.reclassify(oldMetaClass=uml:Property, newMetaClass=uml:Operation)
```

MEF and SMOF Operations for Step 7

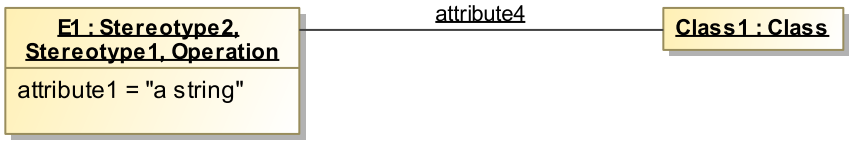

Figure 10.8: Model for Step 7

```
<packagedElement xmi:id="e1" xmi:uuid="omg.org/mef/example/e1"
                 xmi:type="uml:Property" xmi:type="uml:Operation" name="E1"/>
<packagedElement xmi:id="c1" xmi:uuid="omg.org/mef/example/c1"
                 xmi:type="uml:Class" name="C1ass1" />
<tp:Stereotype2 xmi:id="p1" xmi:uuid="omg.org/mef/example/e1"
                xmi:type="tp:Stereotype2">
  <attribute4 xmi:idref="c1"/>
</tp:Stereotype2>
<tp:Stereotype1 xmi:id="p2" xmi:uuid="omg.org/mef/example/e1"
                xmi:type="tp:Stereotype1" attribute1="a string"/>
```

XMI Serialization for Step 7

10.2.8 Step 8

Detach Stereotype2 (it's no longer required)

```
p.reclassify(oldMetaClass=uml:Property, newMetaClass=uml:Operation)
```

MEF and SMOF Operations for Step 8

Figure 10.9: Model for Step 8

```
<packagedElement xmi:id="e1" xmi:uuid="omg.org/mef/example/e1"
                 xmi:type="uml:Property" xmi:type="uml:Operation" name="E1"/>
<packagedElement xmi:id="c1" xmi:uuid="'omg.org/mef/example/c1"
                 xmi:type="uml:Class" name="C1ass1" />
 <tp:Stereotype2 xmi:id="p1" xmi:uuid="omg.org/mef/example/e1"
                 xmi:type="tp:Stereotype2">
  <attribute4 xmi:idref="c1"/>
</tp:Stereotype2>
<tp:Stereotype1 xmi:id="p2" xmi:uuid="omg.org/mef/example/e1"
                xmi:type="tp:Stereotype1" attribute1="a string"/>
```

XMI Serialization for Step 8