Rules for Edit Operations in Class Diagrams

Technical Report

Michaela Rindt, Timo Kehrer, Udo Kelter, Pit Pietsch
Software Engineering Group
University of Siegen

rev. 2012-01-06
Contents

1 Introduction 6
  1.1 Design Rules for Edit Operations . . . . . . . . . . . . . . . . . . . . . . 6
  1.2 Notations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

2 Conditions 7
  2.1 Precondition: checkOtherNames . . . . . . . . . . . . . . . . . . . . . . . 7
  2.2 Precondition: checkOppositeAggregationKind . . . . . . . . . . . . . . . . 8
  2.3 Precondition: checkInheritanceCycle . . . . . . . . . . . . . . . . . . . . . 8

3 Packages 9
  3.1 createPackageInModel . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
  3.2 createPackageInPackage . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
  3.3 deletePackage . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
  3.4 editPackageName . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
  3.5 editPackageVisibility . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
  3.6 movePackageFromModelToPackage . . . . . . . . . . . . . . . . . . . . . . 14
  3.7 movePackageFromPackageToModel . . . . . . . . . . . . . . . . . . . . . . 15
  3.8 movePackageFromPackageToPackage . . . . . . . . . . . . . . . . . . . . . 16

4 Classes 18
  4.1 createClass . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
  4.2 deleteClass . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
  4.3 editClassName . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
  4.4 editClassIsAbstract . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
  4.5 editClassIsFinal . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
  4.6 editClassVisibility . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
  4.7 moveClass . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22

5 Interfaces 24
  5.1 createInterface . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24
  5.2 deleteInterface . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
  5.3 editInterfaceName . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
  5.4 editInterfaceVisibility . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26
  5.5 moveInterface . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27

6 Enumerations 29
  6.1 createEnumeration . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29
  6.2 deleteEnumeration . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30
  6.3 editEnumerationName . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30
  6.4 editEnumerationVisibility . . . . . . . . . . . . . . . . . . . . . . . . . . . 31
  6.5 moveEnumeration . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32
7 Literals
7.1 createLiteral ................................................. 34
7.2 deleteLiteral .................................................. 34
7.3 editLiteralName .............................................. 35
7.4 moveLiteral ................................................... 36

8 Primitive Types
8.1 createPrimitiveType ......................................... 38
8.2 deletePrimitiveType ......................................... 38
8.3 editPrimitiveTypeName ....................................... 39
8.4 movePrimitiveType ........................................... 40

9 Stereotypes
9.1 createStereotype ............................................. 42
9.2 deleteStereotype ............................................. 42
9.3 editStereotypeName .......................................... 43

10 Associations
10.1 createAssociationBetweenClasses ......................... 45
10.2 createAssociationBetweenInterfaces ...................... 46
10.3 deleteAssociation ........................................... 48
10.4 editAssociationName ........................................ 50

11 Association Classes

12 Association Ends
12.1 createAssociationEndIsClass .............................. 53
12.2 createAssociationEndIsInterface ......................... 54
12.3 deleteAssociationEnd ......................................... 56
12.4 editAssociationEndFromClassToClass ..................... 58
12.5 editAssociationEndFromClassToInterface ................. 59
12.6 editAssociationEndFromInterfaceToClass ................ 60
12.7 editAssociationEndFromInterfaceToInterface .......... 60
12.8 editAssociationEndIsNavigatable ......................... 61
12.9 editAssociationEndIsOrdered .............................. 62
12.10 editAssociationEndName ................................. 63
12.11 editAssociationEndKind ................................. 63
12.12 editAssociationEndMultiplicity ......................... 64

13 Generalizations
13.1 createGeneralizationBetweenAssociations ............... 66
13.2 createGeneralizationBetweenClasses ..................... 66
13.3 createGeneralizationBetweenInterfaces .................. 67
13.4 deleteGeneralizationBetweenAssociations ............... 68
13.5 deleteGeneralizationBetweenClasses ..................... 69
13.6 deleteGeneralizationBetweenInterfaces 
13.7 editGeneralizationGeneralElementFromAssociationToAssociation 
13.8 editGeneralizationGeneralElementFromClassToClass 
13.9 editGeneralizationGeneralElementFromInterfaceToInterface 
13.10 editGeneralizationSpecialElementFromInterfaceToInterface 
13.11 editGeneralizationSpecialElementFromClassToClass 
13.12 editGeneralizationSpecialElementFromAssociationToAssociation 

14 Operations 
14.1 createOperationInClass 
14.2 createOperationInInterface 
14.3 deleteOperation 
14.4 editOperationName 
14.5 editOperationIsAbstract 
14.6 editOperationIsStatic 
14.7 editOperationVisibility 
14.8 editOperationReturnTypeFromClassToClass 
14.9 editOperationReturnTypeFromClassToPrimitive 
14.10 editOperationReturnTypeFromPrimitiveToClass 
14.11 editOperationReturnTypeFromPrimitiveToPrimitive 
14.12 moveOperationBetweenClasses 
14.13 moveOperationBetweenInterfaces 
14.14 moveOperationFromClassToInterface 
14.15 moveOperationFromInterfaceToClass 

15 Parameters 
15.1 createParameter 
15.2 deleteParameter 
15.3 editParameterName 
15.4 editParameterKind 
15.5 editParameterTypeFromClassToClass 
15.6 editParameterTypeFromClassToPrimitive 
15.7 editParameterTypeFromPrimitiveToClass 
15.8 editParameterTypeFromPrimitiveToPrimitive 
15.9 moveParameter 

16 Attributes 
16.1 createAttribute 
16.2 deleteAttribute 
16.3 editAttributeName 
16.4 editAttributeIsReadOnly 
16.5 editAttributeIsStatic 
16.6 editAttributeIsFinal 
16.7 editAttributeVisibility
16.8 editAttributeTypeFromClassToClass ........................................... 108
16.9 editAttributeTypeFromClassToPrimitive .................................... 109
16.10 editAttributeTypeFromPrimitiveToClass ................................... 110
16.11 editAttributeTypeFromPrimitiveToPrimitive ............................... 111
16.12 moveAttribute ................................................................. 112

17 References ................................. 114
1 Introduction

This technical report specifies edit operations for UML class diagrams. We assume that the class diagrams are represented as Ecore objects using the EMF technology. Each operation is specified as follows:

1. by a short informal description from a users’ point of view
2. by a suggested signature of a method which implements this edit operation, and an informal description of the parameters in this signature
3. by the preconditions (informally specified) which must hold to successfully perform the edit operation
4. by an implementation of this edit operation as a Henshin transformation rule, which acts as a formal, precise specification.

The main emphasis is on the formal specifications using Henshin. These edit rules can be translated into software which detects applications of these edit operations in a difference between two models, s. [3] for details.

Coverage of the UML Metamodel. This catalog of edit rules covers only the more important concepts of UML class diagrams. For example, details such as ValueSpecifications in multiplicities are not covered. Future versions of this document will extend the set of covered modelling concepts.

Tooling. The edit operations were designed with EMF-Henshin and EMF-Refactor. EMF-Henshin provides a graphical user interface for defining such edit operation, whereas EMF-Refactor is an extension with which one can define preconditions to the actual transformation process.

More Information about EMF-Henshin can be found under their project page [1]. To find more information about EMF-Refactor visit the EMF-Refactor project page [2].

1.1 Design Rules for Edit Operations

Some edit operations are obviously needed, e.g. operations for creating and deleting model elements. This catalog contains many additional operations, mostly “move operations”, which are not strictly necessary, but convenient. An example of this is the operation which moves a class into another package: instead of moving the class, it could be deleted and reconstructed from scratch in the other package.

Whenever possible, we have avoided to use undefined or default values. For example, when creating a class, its name must be specified immediately in a parameter.

Another design decision was that each edit operation should be implementable by one Henshin transformation. There are some model modifications which could be implemented in different ways.

Since the creation of Henshin transformations for UML is still work in progress, this catalog of edit operations is not yet complete and sometimes discretionary.
1.2 Notations

In general, every transformation consists of two files: the (graphical) diagram file and the (more technical) Henshin-graph file. In the latter, one can create TransformationUnits which will define the execution sequence of rules in different ways. In some cases of the following transformations you will find excerpts of such unit graphs for a better understanding.

Notations like 'nameValue/newName' show first the parameter name used in the main-Unit of the graph-file and secondly the placeholder name of the element in the diagram-file.

Most conditions are very similar and are therefore just listed once under the section 'Conditions'. You will find a reference to these under relevant sections.

Please note that the conditions are not yet complete.

2 Conditions

With EMF Refactor and Henshin conditions can be designed either as 'initialCheck', 'finalcheck' or as 'execute'.

Where initial- and finalcheck serve as preconditional check, the 'execute'-type is the actual transformation which does the modification.

Note, with finalchecks one designs the 'negative' case which is not wanted for the execution to be run - or in other words: the finalcheck must NOT match for the 'execute' to be run.

2.1 Precondition: checkOtherNames

This precondition is a 'finalcheck'. checkOtherNames will match if an element (here: UMLClass) with the same name like in the propagated value of 'newName' exists under the selected context (here: UMLPackage) and therefore prevent the actual transformation process.

In other words: the finalcheck mustn’t match if one wants the corresponding 'execute' (here e.g. createClass or editClassName) to be run.
2.2 **Precondition: checkOppositeAggregationKind**

still to be implemented

2.3 **Precondition: checkInheritanceCycle**

still to be implemented
3 Packages

3.1 createPackageInModel

Adds a package in a model / under the root element

Signature:

createPackageInModel(Model selectedEObject, String idValue, String nameValue)

Context element (selectedEObject):

The model providing the container for the newly created package.

Additional parameters:

- idValue/newID: The identifier of the newly created package
- nameValue/newName: The name of the newly created package

Precondition:

There is no package whose name equals the parameter-value of 'newName’ (see 2.1)

Implementation in EMF Henshin:

A new package and references to and from the model will be created. 'selectedModel', as it is shown in the picture, is a placeholder for the concrete Model which will be delivered via mappings of the selectedEObject. Also a default value 'public' for the visibility feature will be set.

Figure 2: createPackageInModel
3.2 createPackageInPackage

Adds a package in a package

**Signature:**

createPackageInPackage(Package selectedEObject, String idValue, String nameValue)

**Context element (selectedEObject):**

The package providing the container for the newly created package.

**Additional parameters:**

- idValue/newID: The identifier of the newly created package
- nameValue/newName: The name of the newly created package

**Precondition:**

There is no package whose name equals the parameter-value of 'newName' (see 2.1)

**Implementation in EMF Henshin:**

A new package and references to and from the super package will be created. 'selectedPackage', as it is shown in the picture, is a placeholder for the concrete super package which will be delivered via mappings of the selectedEObject. Also a default value 'public' for the visibility feature will be set.

![Figure 3: createPackageInPackage](image)

Figure 3: createPackageInPackage
3.3 deletePackage

Deletes a package

Signature:

deletePackage(Package selectedEObject)

Context element (selectedEObject):

The package which should be deleted

Additional parameters:

Precondition:

Implementation in EMF Henshin:

For a better readability this is a simplified version of the 'deletePackage'-transformation and will only cover cases where the package has no containments and no references to other elements. Such a complex transformation rule exits but won’t be listed here.

In this simple version we first of all have to find out from which context type the selected package should be deleted. For this we have the rule 'containerIsModel', which matches if the container is a model. If not we can assume that the context is a package since no other model element can contain a package according to the meta model specification. After we know the context type one of the two variants of the delete-rule (deletePackageFromModel and deletePackageFromPackage) can be applied.

Below you can also see the graph of units which explains this process with an if-then-else structure called 'Conditional Unit'. The selectedEObject is the placeholder for the context and it will be propagated to the according parameters in the three rules via the so called 'Parameter Mappings'.
Figure 4: deletePackage

```
 conditional Unit mainUnit
  If
    ⇒ Rule containerIsModel
  Then
    ⇒ Rule deletePackageFromModel
  Else
    ⇒ Rule deletePackageFromPackage
    Parameter selectedEObject
    Parameter Mapping selectedEObject -> selectedPackage[containerIsModel]
    Parameter Mapping selectedEObject -> selectedPackage[deletePackageFromModel]
    Parameter Mapping selectedEObject -> selectedPackage[deletePackageFromPackage]
```

Figure 5: deletePackage(UnitView)
3.4 **editPackageName**

edits the name of a package

**Signature:**

\[
\text{editPackageName(Package selectedEObject, String nameValue)}
\]

**Context element (selectedEObject):**

The package whose name should be renamed.

**Additional parameters:**

- nameValue/newName: The new name

**Precondition:**

There is no package whose name equals the parameter-value of 'newName' (see 2.1)

**Implementation in EMF Henshin:**

The \texttt{<<create>>}-symbol in the image means that even if the attribute exists its value will be overwritten. 'newName' is the placeholder for the input name.

![editPackageName](image)

Figure 6: editPackageName

3.5 **editPackageVisibility**

edits the visibility of a package

**Signature:**

\[
\text{editPackageVisibility(Package selectedEObject, Visibility visibilityValue)}
\]

**Context element (selectedEObject):**

The package whose visibility should be edited.
Additional parameters:

- visibilityValue/visibility: The new visibility

Precondition:

Implementation in EMF Henshin:

The `<<create>>`-symbol in the image means that even if the attribute exists its value will be overwritten. 'visibility' is the placeholder for the input visibility.

![Diagram](image)

Figure 7: editPackageVisibility

3.6 `movePackageFromModelToPackage`

moves a package from a model into package

**Signature:**

`movePackageFromModelToPackage(Package selectedEObject, Package tgt)`

**Context element (selectedEObject):**

The package which should be moved.

**Additional parameters:**

- tgt/targetPackage: the target package

**Precondition:**

There is no package with the same name in the target context (see 2.1)

Implementation in EMF Henshin:

Only the references change
3.7 movePackageFromPackageToModel

moves a package from a package directly under the model

**Signature:**

movePackageFromPackageToModel(Package selectedEObject, Model tgt)

**Context element (selectedEObject):**

The package which should be moved.

**Additional parameters:**

- tgt/targetModel: the target model

**Precondition:**

There is no package with the same name in the target context (see 2.1)

**Implementation in EMF Henshin:**

Only the references change
3.8 movePackageFromPackageToPackage

moves a package from a package into another package

**Signature:**

movePackageFromPackageToPackage(Package selectedEObject, Package tgt)

**Context element (selectedEObject):**

The package which should be moved.

**Additional parameters:**

- tgt/targetPackage: the target package

**Precondition:**

There is no package with the same name in the target context (see 2.1)

**Implementation in EMF Henshin:**

Only the references change
Figure 10: movePackageFromPackageToPackage
4 Classes

4.1 createClass

creates a new class

Signature:

createClass(Package selectedEObject, String nameValue)

Context element (selectedEObject):

The package providing the container for the newly created class.

Additional parameters:

- nameValue/newName: The name of the newly created class
- idValue/newID: The id of the newly created class

Precondition:

There is no class whose name equals the parameter-value of 'newName' (see 2.1)

Implementation in EMF Henshin:

Only the name and the id will be set via input data. Visibility, isAbstract and isFinal will be set with default values as defined with the diagram editor in the image below.

Figure 11: createClass
4.2 deleteClass
Deletes a class

**Signature:**

del**eteClass(Class selectedEObject)**

**Context element (selectedEObject):**
The class which should be deleted

**Additional parameters:**
-

**Precondition:**
-

**Implementation in EMF Henshin:**
For a better readability this is a simplified version of the 'deleteClass'-transformation and will only cover cases where the class has no containments and no references to other elements. Such a complex transformation rule exits but won’t be listed here.

![diagram](image)

Figure 12: deleteClass

4.3 editClassName
edits the name of a class
Signature:
editClassName(Class selectedEObject, String nameValue)

Context element (selectedEObject):
The class whose name should be renamed.

Additional parameters:
• nameValue/someName: The new name

Precondition:
There is no class in the same package whose name equals the parameter-value of 'newName' (see 2.1)

Implementation in EMF Henshin:
The <<create>>-symbol in the image means that even if the attribute exists its value will be overwritten. 'someName' is the placeholder for the input name.

4.4 editClassIsAbstract
edits the isAbstract-value of a class

Signature:
editClassIsAbstract(Class selectedEObject, boolean booleanValue)

Context element (selectedEObject):
The class whose isAbstract-value should be edited.

Additional parameters:
• booleanValue/bool: The new isAbstract-value
Precondition:

- 

Implementation in EMF Henshin:

The \texttt{<<create>>}-symbol in the image means that even if the attribute exists its value will be overwritten. 'bool' is the placeholder for the input boolean value.

Figure 14: editClassIsAbstract

4.5 editClassIsFinal

edits the isFinal-value of a class

Signature:

\texttt{editClassIsFinal(Class selectedEObject, boolean booleanValue)}

Context element (selectedEObject):

The class whose isFinal-value should be edited.

Additional parameters:

- booleanValue/bool: The new isFinal-value

Precondition:

- 

Implementation in EMF Henshin:

The \texttt{<<create>>}-symbol in the image means that even if the attribute exists its value will be overwritten. 'bool' is the placeholder for the input boolean value.
4.6 editClassVisibility

edits the visibility of a class

**Signature:**

`editClassVisibility(Class selectedEObject, Visibility visibilityValue)`

**Context element (selectedEObject):**

The class whose visibility should be edited.

**Additional parameters:**

- `visibilityValue/visibility`: The new visibility

**Precondition:**

-

**Implementation in EMF Henshin:**

The `<create>`-symbol in the image means that even if the attribute exists its value will be overwritten. ‘visibility’ is the placeholder for the input visibility value.

4.7 moveClass

moves a class from a package to another package
Signature:
moveClass(Class selectedEObject, Package tgt)

Context element (selectedEObject):
The class which should be moved.

Additional parameters:
• tgt/tgt[moveClass]: the target package

Precondition:
There is no class with the same name in the target context (see 2.1)

Implementation in EMF Henshin:
Only references will change.

Figure 17: moveClass
5 Interfaces

5.1 createInterface

creates a new interface

Signature:

createInterface(Package selectedEObject, String nameValue, String idValue)

Context element (selectedEObject):
The package providing the container for the newly created interface.

Additional parameters:

• nameValue/newName: The name of the newly created interface
• idValue/newID: The id of the newly created interface

Precondition:

There is no interface whose name equals the parameter-value of ‘newName’ (see 2.1)

Implementation in EMF Henshin:

Only the name and the id will be set via input data. Visibility will be set with a default value as defined with the diagram editor in the image below.

![Diagram of createInterface](image)

Figure 18: createInterface
5.2 deleteInterface

Deletes an interface

**Signature:**

```
deleteInterface(Interface selectedEObject)
```

**Context element (selectedEObject):**

The interface which should be deleted

**Additional parameters:**

- 

**Precondition:**

- 

**Implementation in EMF Henshin:**

For a better readability this is a simplified version of the 'deleteInterface'-transformation and will only cover cases where the interfac has no containments and no references to other elements. Such a complex transformation rule exits but won’t be listed here.

![Figure 19: deleteInterface](image)

5.3 editInterfaceName

edits the name of an interface

**Signature:**

```
editInterfaceName(Interface selectedEObject, String nameValue)
```
Context element (selectedEObject):

The interface whose name should be renamed.

Additional parameters:

- nameValue/newName: The new name

Precondition:

There is no interface in the same package whose name equals the parameter-value of 'newName' (see 2.1)

Implementation in EMF Henshin:

The <<create>>-symbol in the image means that even if the attribute exists its value will be overwritten. 'newName' is the placeholder for the input name.

5.4 editInterfaceVisibility

edits the visibility of an interface

Signature:

editInterfaceVisibility(Interface selectedEObject, Visibility visibilityValue)

Context element (selectedEObject):

The interface whose visibility should be edited.

Additional parameters:

- visibilityValue/visibility: The new visibility

Precondition:

-
Implementation in EMF Henshin:

The <<create>>-symbol in the image means that even if the attribute exists its value will be overwritten. ‘visibility’ is the placeholder for the input visibility value.

![Diagram](image)

Figure 21: editInterfaceVisibility

5.5 moveInterface

moves an interface from a package to another package

**Signature:**

moveInterface(Interface selectedEObject, Package tgt)

**Context element (selectedEObject):**

The interface which should be moved.

**Additional parameters:**

- tgt/tgt[moveInterface]: the target package

**Precondition:**

There is no interface with the same name in the target context (see 2.1)

**Implementation in EMF Henshin:**

Only references will change
Figure 22: moveInterface
6 Enumerations

6.1 createEnumeration
creates a new enumeration

Signature:
createEnumeration(Package selectedEObject, String nameValue, String idValue)

Context element (selectedEObject):
The package providing the container for the newly created enumeration.

Additional parameters:
- nameValue/newName: The name of the newly created enumeration
- idValue/newID: The id of the newly created enumeration

Precondition:
There is no enumeration whose name equals the parameter-value of 'newName' (see 2.1)

Implementation in EMF Henshin:
Only the name and the id will be set via input data. Visibility will be set with a default value as defined with the diagram editor in the image below.

![createEnumeration Diagram](image)

Figure 23: createEnumeration
6.2 deleteEnumeration

Deletes an enumeration

**Signature:**

deleteEnumeration(Enumeration selectedEObject)

**Context element (selectedEObject):**

The enumeration which should be deleted

**Additional parameters:**

-

**Precondition:**

-

**Implementation in EMF Henshin:**

For a better readability this is a simplified version of the 'deleteEnumeration'-transformation and will only cover cases where the enumeration has no containments and no references to other elements. Such a complex transformation rule exits but won’t be listed here.

![Figure 24: deleteEnumeration](image)

6.3 editEnumerationName

edits the name of an enumeration

**Signature:**

editEnumerationName(Enumeration selectedEObject, String nameValue)
Context element (selectedEObject):
The enumeration whose name should be renamed.

Additional parameters:
- nameValue/newName: The new name

Precondition:
There is no enumeration in the same package whose name equals the parameter-value of ‘newName’ (see 2.1)

Implementation in EMF Henshin:
The \textless \textgreater symbol in the image means that even if the attribute exists its value will be overwritten. ’newName’ is the placeholder for the input name.

Figure 25: editEnumerationName

6.4 editEnumerationVisibility
edits the visibility of an enumeration

Signature:
editEnumerationVisibility(Enumeration selectedEObject, Visibility visibilityValue)

Context element (selectedEObject):
The enumeration whose visibility should be edited.

Additional parameters:
- visibilityValue/visibility: The new visibility

Precondition:

Implementation in EMF Henshin:

The "<<create>>"-symbol in the image means that even if the attribute exists its value will be overwritten. 'visibility' is the placeholder for the input visibility value.

![Figure 26: editEnumerationVisibility](image)

6.5 moveEnumeration

moves an enumeration from a package to another package

**Signature:**

moveEnumeration(Enumeration selectedEObject, Package tgt)

**Context element (selectedEObject):**

The enumeration which should be moved.

**Additional parameters:**

- tgt/tgt[moveEnumeration]: the target package

**Precondition:**

There is no enumeration with the same name in the target context (see 2.1)

**Implementation in EMF Henshin:**

Only references will change.
Figure 27: moveEnumeration
7 Literature

7.1 createLiteral

creates a new literal

**Signature:**

createLiteral(Package selectedEObject, String nameValue)

**Context element (selectedEObject):**

The package providing the container for the newly created literal.

**Additional parameters:**

- nameValue/newName: The name of the newly created literal
- idValue/newID: The id of the newly created literal

**Precondition:**

There is no literal whose name equals the parameter-value of ‘newID’ (see 2.1)

**Implementation in EMF Henshin:**

The name and the id will be set via input data.

![createLiteral](image)

Figure 28: createLiteral

7.2 deleteLiteral

Deletes a literal
Signature:

\texttt{deleteLiteral(Literal selectedEObject)}

Context element (selectedEObject):

The literal which should be deleted

Additional parameters:

-

Precondition:

-

Implementation in EMF Henshin:

For a better readability this is a simplified version of the 'deleteLiteral'-transformation and will only cover cases where the literal has no references to other elements. Such a complex transformation rule exits but won’t be listed here.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{deleteLiteral}
\caption{deleteLiteral}
\end{figure}

7.3 \textbf{editLiteralName}

edits the name of a literal

Signature:

\texttt{editLiteralName(Literal selectedEObject, String nameValue)}
**Context element (selectedEObject):**

The literal whose name should be renamed.

**Additional parameters:**

- `nameValue/newName`: The new name

**Precondition:**

There is no literal in the same package whose name equals the parameter-value of 'newName' (see 2.1)

**Implementation in EMF Henshin:**

The `<<create>>`-symbol in the image means that even if the attribute exists its value will be overwritten. 'newName' is the placeholder for the input name.

![editLiteralName](image)

Figure 30: editLiteralName

### 7.4 moveLiteral

moves a literal from an enumeration to another

**Signature:**

`moveLiteral(Literal selectedEObject, Enumeration tgt)`

**Context element (selectedEObject):**

The literal which should be moved.

**Additional parameters:**

- `tgt/tgt[moveLiteral]`: the target enumeration

**Precondition:**

There is no literal with the same name in the target context (see 2.1)
Implementation in EMF Henshin:

Only references will change.

Figure 31: moveLiteral
8 PrimitiveTypes

8.1 createPrimitiveType

creates a new primitiveType

Signature:

createPrimitiveType(Package selectedEObject, String nameValue, String idValue)

Context element (selectedEObject):

The package providing the container for the newly created primitiveType.

Additional parameters:

- nameValue/newName: The name of the newly created primitiveType
- idValue/newID: The id of the newly created primitiveType

Precondition:

There is no primitiveType whose name equals the parameter-value of 'newName' (see 2.1)

Implementation in EMF Henshin:

The name and the id will be set via input data.

![Diagram of createPrimitiveType](image)

Figure 32: createPrimitiveType

8.2 deletePrimitiveType

Deletes a primitiveType
**Signature:**
```
deletePrimitiveType(PrimitiveType selectedEObject)
```

**Context element (selectedEObject):**
The primitiveType which should be deleted

**Additional parameters:**
- 

**Precondition:**
- 

**Implementation in EMF Henshin:**
For a better readability this is a simplified version of the 'deletePrimitiveType'-transformation and will only cover cases where the primitiveType has no references to other elements. Such a complex transformation rule exits but won’t be listed here.

![Figure 33: deletePrimitiveType](image)

**8.3 editPrimitiveTypeName**

edits the name of a primitiveType

**Signature:**
```
editPrimitiveTypeName(PrimitiveType selectedEObject, String nameValue)
```

**Context element (selectedEObject):**
The primitiveType whose name should be renamed.
Additional parameters:

- nameValue/newName: The new name

Precondition:

There is no primitiveType in the same package whose name equals the parameter-value of 'newName' (see 2.1)

Implementation in EMF Henshin:

The <<create>>-symbol in the image means that even if the attribute exists its value will be overwritten. 'newName' is the placeholder for the input name.

8.4 movePrimitiveType

moves a primitiveType from a package to another package

Signature:

movePrimitiveType(PrimitiveType selectedEObject, Package tgt)

Context element (selectedEObject):

The primitiveType which should be moved.

Additional parameters:

- tgt/tgt[movePrimitiveType]: the target package

Precondition:

There is no primitiveType with the same name in the target context (see 2.1)

Implementation in EMF Henshin:

Only references will change
Figure 35: movePrimitiveType
9 Stereotypes

9.1 createStereotype

creates a new stereotype

Signature:

createStereotype(Package selectedEObject, String nameValue)

Context element (selectedEObject):
The model providing the container for the newly created stereotype.

Additional parameters:

• nameValue/newName: The name of the newly created stereotype
• idValue/newID: The id of the newly created stereotype

Precondition:

There is no stereotype whose name equals the parameter-value of ‘newName’ (see 2.1)

Implementation in EMF Henshin:

Only the name and the id will be set via input data.

Figure 36: createStereotype

9.2 deleteStereotype

Deletes a stereotype
**Signature:**

deleteStereotype(Stereotype selectedEObject)

**Context element (selectedEObject):**

The stereotype which should be deleted

**Additional parameters:**

-

**Precondition:**

-

**Implementation in EMF Henshin:**

For a better readability this is a simplified version of the ‘deleteStereotype’-transformation and will only cover cases where the stereotype has no containments and no references to other elements. Such a complex transformation rule exits but won’t be listed here.

![Figure 37: deleteStereotype](image)

**9.3 editStereotypeName**

edits the name of a stereotype

**Signature:**

editStereotypeName(Stereotype selectedEObject, String nameValue)

**Context element (selectedEObject):**

The stereotype whose name should be renamed.
Additional parameters:

- nameValue/newName: The new name

Precondition:

There is no stereotype in the same package whose name equals the parameter-value of 'newName' (see 2.1)

Implementation in EMF Henshin:

The "<<create>>"-symbol in the image means that even if the attribute exists its value will be overwritten. 'newName' is the placeholder for the input name.

Figure 38: editStereotypeName
10 Associations

10.1 createAssociationBetweenClasses

creates a new association between classes

**Signature:**

createAssociationBetweenClasses(Package selectedEObject, String nameValue, String idValue, String srcMultiplicity)

**Context element (selectedEObject):**

The package providing the container for the newly created association.

**Additional parameters:**

- nameValue/newName: The name of the newly created association
- idValue/newID: The id of the newly created association
- srcValue/newSrcName: The name of the newly created source association end
- srcIdValue/newSrcID: The id of the newly created source association end
- srcMultiplicity/newMultiplicity: The multiplicity of the newly created source association end
- tgtValue/newTgtName: The name of the newly created target association end
- tgtIdValue/newTgtID: The id of the newly created target association end
- tgtMultiplicity/newMultiplicity: The multiplicity of the newly created target association end

**Precondition:**

The 'newSrcName' and 'newTgtName' must differ (see 2.1)

**Implementation in EMF Henshin:**

This rule firstly creates an association under the propagated package with input data for id and name. Secondly it will create association ends with input data for id, name and multiplicity (isNavigatable, kind and isOrdered will receive default values). Afterwards it will append them to the association created beforehand via references (association and associationEnds). Lastly the association ends will either get a reference (target) to an input class.

Note that the reference from a class and from the selected association point to the same
unknown package. This will make sure that there won’t be created associations between classes of different packages.

![Diagram: createAssociationBetweenClasses]

Figure 39: createAssociationBetweenClasses

### 10.2 createAssociationBetweenInterfaces

creates a new association between interfaces

**Signature:**

`.createAssociationBetweenInterfaces(Package selectedEObject, String nameValue, String idValue, String srcMultiplicity)`
Context element (selectedEObject):

The package providing the container for the newly created association.

Additional parameters:

- nameValue/newName: The name of the newly created association
- idValue/newID: The id of the newly created association
- srcValue/newSrcName: The name of the newly created source association end
- srcIdValue/newSrcID: The id of the newly created source association end
- srcMultiplicity/newMultiplicity: The multiplicity of the newly created source association end
- tgtValue/newTgtName: The name of the newly created target association end
- tgtIdValue/newTgtID: The id of the newly created target association end
- tgtMultiplicity/newMultiplicity: The multiplicity of the newly created target association end

Precondition:

The 'newSrcName' and 'newTgtName' must differ (see 2.1)

Implementation in EMF Henshin:

This rule firstly creates an association under the propagated package with input data for id and name. Secondly it will create association ends with input data for id, name and multiplicity (isNavigatable, kind and isOrdered will recieve default values). Afterwards it will append them to the association created beforehand via references (association and associationEnds). Lastly the association ends will either get a reference (target) to an input interface.

Note that the reference from an interface and from the selected association point to the same unknown package. This will make sure that there won’t be created associations between interfaces of different packages.
10.3 deleteAssociation

Deletes an association

**Signature:**

`deleteAssociation(Association selectedEObject)`

**Context element (selectedEObject):**

The association which should be deleted including all its containments and all its references
Additional parameters:

- 

Precondition:

- 

Implementation in EMF Henshin:

For this transformation three rules are needed. The first two will be used to delete the target reference of an association end depending on the type of the target. The third rule will delete the selected association from its container package.

In the second picture you can find the TransformationUnits. It contains a Sequential 'mainUnit' which will firstly run the Counted Unit 'LOOP_deleteAssociationEndIsClass'. After there is no association end under the selected association left which points to a class, the next Counted Unit 'LOOP_deleteAssociationEndIsInterface' will be run. Both Counted Units behave similarly. When there are no available association ends left, the deleteAssociation-Rule will be called.

Figure 41: deleteAssociation
10.4 editAssociationName

dits the name of an association

Signature:

editAssociationName(Association selectedEObject, String nameValue)

Context element (selectedEObject):

The association whose name should be renamed.

Additional parameters:

- nameValue/newName: The new name

Precondition:

Implementation in EMF Henshin:

The "<<create>>"-symbol in the image means that even if the attribute exists its value will be overwritten. 'newName' is the placeholder for the input name.
Figure 43: editAssociationName
11 AssociationClasses

Since the AssociationClass-Transformations are build the same like the Association-Transformations with the only difference that the selectedEObject of the edit- and delete-Transformations is an AssociationClass instead of an Association they won’t be covered here.
12 AssociationEnds

12.1 createAssociationEndIsClass

Creates a new association end under the selected association and links it to a class that lies in the same package as the association.

**Signature:**

```java
createAssociationEndIsClass(Association selectedEObject, String nameValue, String idValue, String srcMultiplicity)
```

**Context element (selectedEObject):**

The association providing the container for the newly created association end.

**Additional parameters:**

- `nameValue/newName`: The name of the newly created association end
- `idValue/newID`: The id of the newly created association end
- `srcMultiplicity/newMultiplicity`: The multiplicity of the newly created source association end

**Precondition:**

There mustn’t be an association end under the selected association with the same name like in 'newName' (see 2.1)

**Implementation in EMF Henshin:**

Only the name, id and multiplicity will be set via input data. `isNavigable`, `kind` and `isOrdered` will be set with a default value as defined with the diagram editor in the image below. Note that the reference from a class and from the selected association point to the same unknown package. This will make sure that there won’t be created associations between classes of different packages.
12.2 createAssociationEndIsInterface

Creates a new association end under the selected association and links it to an interface that lies in the same package as the association.

**Signature:**

createAssociationEndIsInterface(Association selectedEObject, String nameValue, String idValue, String srcMultiplicity)

**Context element (selectedEObject):**

The association providing the container for the newly created association end.

**Additional parameters:**

- nameValue/newName: The name of the newly created association end
- idValue/newID: The id of the newly created association end
• `srcMultiplicity/newMultiplicity`: The multiplicity of the newly created source association end

**Precondition:**

There mustn’t be an association end under the selected association with the same name like in 'newName' (see 2.1).

**Implementation in EMF Henshin:**

Only the name, id and multiplicity will be set via input data. isNavigable, kind and isOrdered will be set with a default value as defined with the diagram editor in the image below. Note that the reference from an interface and from the selected association point to the same unknown package. This will make sure that there won’t be created associations between interfaces of different packages.

![Diagram](image.png)

Figure 45: createAssociationEndIsInterface
12.3 deleteAssociationEnd

Deletes an association end and the reference to its actual target, but not the target itself

**Signature:**

```
deleteAssociationEnd(AssociationEnd selectedEObject)
```

**Context element (selectedEObject):**

The association end which should be deleted including its reference

**Additional parameters:**

- 

**Precondition:**

- 

**Implementation in EMF Henshin:**

This transformation is designed with three rules. The first two rules will delete an association end depending on which type it points to. The third one will simply do nothing but is still required in the TransformationUnits.

You can see that the 'mainUnit' is a Conditional Unit this time. It will try to run the rule 'deleteAssociationEndIsClass' and in case it will match this means that the modification has worked and there is nothing left to do (that’s why a 'doNothing-Rule' is following in the 'Then'-part). Otherwise, if our selected association end does not point to a class, we assume that it can only point to an interface and therefore 'deleteAssociationEndIsInterface' will be called.
Figure 46: deleteAssociationEnd
12.4 editAssociationEndFromClassToClass

Edits the target of an association end to another class

**Signature:**

```
editAssociationEndFromClassToClass(AssociationEnd selectedEObject, Class tgt)
```

**Context element (selectedEObject):**

The association end whose target should be changed

**Additional parameters:**

- tgt/targetClass: The new class

**Precondition:**

- 

**Implementation in EMF Henshin:**

Only references will change.
12.5 editAssociationEndFromClassToInterface

Edits the target of an association end from a class to an interface

**Signature:**

\[
\text{editAssociationEndFromClassToInterface(AssociationEnd selectedEObject, Interface tgt)}
\]

**Context element (selectedEObject):**

The association end whose target should be changed

**Additional parameters:**

- tgt/targetInterface: The new interface

**Precondition:**

- 

**Implementation in EMF Henshin:**

Only references will change.
12.6 editAssociationEndFromInterfaceToClass
Edits the target of an association end from an interface to a class

**Signature:**

editAssociationEndFromInterfaceToClass(AssociationEnd selectedEObject, Class tgt)

**Context element (selectedEObject):**
The association end whose target should be changed

**Additional parameters:**
- tgt/targetClass: The new class

**Precondition:**

- 

**Implementation in EMF Henshin:**
Only references will change.

![Diagram](image)

Figure 50: editAssociationEndFromInterfaceToClass

12.7 editAssociationEndFromInterfaceToInterface
Edits the target of an association end to another interface

**Signature:**

editAssociationEndFromInterfaceToInterface(AssociationEnd selectedEObject, Interface tgt)

**Context element (selectedEObject):**
The association end whose target should be changed
Additional parameters:

- tgt/targetInterface: The new interface

Precondition:

- 

Implementation in EMF Henshin:

Only references will change.

![Figure 51: editAssociationEndFromInterfaceToInterface](image)

12.8 editAssociationEndIsNavigatable

Edits the isNavigatable-value of an association end

Signature:

editAssociationEndIsNavigatable(AssociationEnd selectedEObject, boolean booleanValue)

Context element (selectedEObject):

The association end whose isNavigatable-value should be edited.

Additional parameters:

- booleanValue/bool: The new isNavigatable-value

Precondition:

-
Implementation in EMF Henshin:

The <<create>>-symbol in the image means that even if the attribute exists its value will be overwritten. 'bool' is the placeholder for the input bool value.

![Figure 52: editAssociationEndIsNavigatable](image)

12.9 editAssociationEndIsOrdered

Edits the isOrdered-value of an association end

**Signature:**

```java
editAssociationEndIsOrdered(AssociationEnd selectedEObject, boolean booleanValue)
```

**Context element (selectedEObject):**

The association end whose isOrdered-value should be edited.

**Additional parameters:**

- booleanValue/bool: The new isOrdered-value

**Precondition:**

- 

**Implementation in EMF Henshin:**

The <<create>>-symbol in the image means that even if the attribute exists its value will be overwritten. 'bool' is the placeholder for the input bool value.

![Figure 53: editAssociationEndIsOrdered](image)
12.10 editAssociationEndName
Edits the name of an association end

Signature:
editAssociationEndName(AssociationEnd selectedEObject, String nameValue)

Context element (selectedEObject):
The class whose name should be renamed.

Additional parameters:
• nameValue/newName: The new name

Precondition:
There is no association end under the same association whose name equals the parameter-
value of 'newName' (see 2.1)

Implementation in EMF Henshin:
The <<create>>-symbol in the image means that even if the attribute exists its value will be overwritten. 'newName' is the placeholder for the input name.

Figure 54: editAssociationEndName

12.11 editAssociationEndKind
Edits the kind (none, shared or composit) of an association end

Signature:
editAssociationEndKind(AssociationEnd selectedEObject, Kind kindValue)

Context element (selectedEObject):
The class whose name should be renamed.
Additional parameters:

- kindValue/newKind: The new kind

Precondition:

The opposit ending mustn’t also be composit if the new kind is composit (see 2.2)

Implementation in EMF Henshin:

The <<create>>-symbol in the image means that even if the attribute exists its value will be overwritten. 'newKind' is the placeholder for the input kind value.

![Image of editAssociationEndKind]

Figure 55: editAssociationEndKind

12.12 editAssociationEndMultiplicity

Edits the multiplicity of an association end

Signature:

editAssociationEndMultiplicity(AssociationEnd selectedEObject, String srcMultiplic-
ity)

Context element (selectedEObject):

The class whose name should be renamed.

Additional parameters:

- srcMultiplicity/newMultiplicity: The new multiplicity

Precondition:

-

Implementation in EMF Henshin:

The <<create>>-symbol in the image means that even if the attribute exists its value will be overwritten. 'newMultiplicity' is the placeholder for the input multiplicity value.
Figure 56: editAssociationEndMultiplicity

- <<preserve>>
- selectedAssociationEnd:UMLAssociationEnd
- <<create>> multiplicity=newMultiplicity
13 Generalizations

13.1 createGeneralizationBetweenAssociations

Creates a new generalization relationship between two associations

Signature:

createGeneralizationBetweenAssociations(Association selectedEObject, Association tgt)

Context element (selectedEObject):

The association which will become the general element

Additional parameters:

- tgt/targetAssociation: the association which will become the special element and inherit from the general element

Precondition:

The new generalization relationship must not create an inheritance-cycle (see 2.3)

Implementation in EMF Henshin:

A generalization and references to the input selected association (the parent) and the target association (the child) are created.

![Diagram of createGeneralizationBetweenAssociations](image)

Figure 57: createGeneralizationBetweenAssociations

13.2 createGeneralizationBetweenClasses

Creates a new generalization relationship between classes

Signature:

createGeneralizationBetweenClasses(Class selectedEObject, Class tgt)
Context element (selectedEObject):
The class which will become the general element

Additional parameters:

• tgt/targetClass: the class which will become the special element and inherit from the general element

Precondition:
The new generalization relationship must not create an inheritance-cycle (see 2.3)

Implementation in EMF Henshin:
A generalization and references to the input selected class (the parent) and the target class (the child) are created.

![createGeneralizationBetweenClasses Diagram](image)

Figure 58: createGeneralizationBetweenClasses

13.3 createGeneralizationBetweenInterfaces
Creates a new generalization relationship between interfaces

Signature:
createGeneralizationBetweenInterfaces(Interface selectedEObject, Interface tgt)

Context element (selectedEObject):
The interface which will become the general element

Additional parameters:

• tgt/targetInterface: the interface which will become the special element and inherit from the general element
Precondition:

The new generalization relationship must not create an inheritance-cycle (see 2.3).

Implementation in EMF Henshin:

A generalization and references to the input selected interface (the parent) and the target interface (the child) are created.

![Figure 59: createGeneralizationBetweenInterfaces](image)

13.4 deleteGeneralizationBetweenAssociations

Deletes a generalization between associations

**Signature:**

deleGeneralizationBetweenAssociations(Generalization selectedEObject)

**Context element (selectedEObject):**

The generalization which should be deleted

**Additional parameters:**

- 

**Precondition:**

- 

Implementation in EMF Henshin:

First the references from the associations to the generalization are deleted and then the selected generalization itself.
13.5 **deleteGeneralizationBetweenClasses**

Deletes a generalization between classes

**Signature:**

\[ \text{deleteGeneralizationBetweenClasses}(\text{Generalization selectedEObject}) \]

**Context element (selectedEObject):**

The generalization which should be deleted

**Additional parameters:**

- 

**Precondition:**

- 

**Implementation in EMF Henshin:**

First the references from the classes to the generalization are deleted and then the selected generalization itself.
13.6 deleteGeneralizationBetweenInterfaces

Deletes a generalization between interfaces

**Signature:**

```java
deleteGeneralizationBetweenInterfaces(Generalization selectedEObject)
```

**Context element (selectedEObject):**

The generalization which should be deleted

**Additional parameters:**

- 

**Precondition:**

- 

**Implementation in EMF Henshin:**

First the references from the interfaces to the generalization are deleted and then the selected generalization itself.
13.7 editGeneralizationGeneralElementFromAssociationToAssociation

Replaces the general element of a generalization from an association by another

**Signature:**

```
eeditGeneralizationGeneralElementFromAssociationToAssociation(Generalization selectedEObject, Association src, Association tgt)
```

**Context element (selectedEObject):**

The generalization whose general element should be changed.

**Additional parameters:**

- `src/oldAssociation`: The old association as general element
- `tgt/newAssociation`: The new association as general element

**Precondition:**

The new generalization relationship must not create an inheritance-cycle (see [2.3](#)).

**Implementation in EMF Henshin:**

Only references change.
13.8 editGeneralizationGeneralElementFromClassToClass

Replaces the general element of a generalization from an class by another

**Signature:**

```
editGeneralizationGeneralElementFromClassToClass(Generalization selectedEObject, Class src, Class tgt)
```

**Context element (selectedEObject):**

The generalization whose general element should be changed.

**Additional parameters:**

- `src/oldClass`: The old class as general element
- `tgt/newClass`: The new class as general element

**Precondition:**

The new generalization relationship must not create an inheritance-cycle (see 2.3).

**Implementation in EMF Henshin:**

Only references change.
13.9 editGeneralizationGeneralElementFromInterfaceToInterface

Replaces the general element of a generalization from an interface by another

**Signature:**

```
editGeneralizationGeneralElementFromInterfaceToInterface(Generalization selectedEObject, Interface src, Interface tgt)
```

**Context element (selectedEObject):**

The generalization whose general element should be changed.

**Additional parameters:**

- `src/oldInterface`: The old interface as general element
- `tgt/newInterface`: The new interface as general element

**Precondition:**

The new generalization relationship must not create an inheritance-cycle (see 2.3)

**Implementation in EMF Henshin:**

Only references change.
13.10 editGeneralizationSpecialElementFromInterfaceToInterface

Edits the special element of a specialization from an interface to another

**Signature:**

editGeneralizationSpecialElementFromInterfaceToInterface(Generalization selectedEObject, Interface src, Interface tgt)

**Context element (selectedEObject):**

The specialization whose special element should be changed.

**Additional parameters:**

- src/oldInterface: The old interface as special element
- tgt/newInterface: The new interface as special element

**Precondition:**

The new generalization relationship must not create an inheritance-cycle (see 2.3)

**Implementation in EMF Henshin:**

Only references change.
13.11 editGeneralizationSpecialElementFromClassToClass

Edits the special element of a specialization from one class to another.

**Signature:**

```java
editGeneralizationSpecialElementFromClassToClass(Generalization selectedEObject, Class src, Class tgt)
```

**Context element (selectedEObject):**

The specialization whose special element should be changed.

**Additional parameters:**

- `src/oldClass`: The old class as special element
- `tgt/newClass`: The new class as special element

**Precondition:**

The new generalization relationship must not create an inheritance-cycle (see 2.3).

**Implementation in EMF Henshin:**

Only references change.
13.12 editGeneralizationSpecialElementFromAssociationToAssociation

Edits the special element of a specialization from an association to another

**Signature:**

editGeneralizationSpecialElementFromAssociationToAssociation(Generalization selectedEObject, Association src, Association tgt)

**Context element (selectedEObject):**

The specialization whose special element should be changed.

**Additional parameters:**

- src/oldAssociation: The old association as special element
- tgt/newAssociation: The new association as special element

**Precondition:**

The new generalization relationship must not create an inheritance-cycle (see 2.3)

**Implementation in EMF Henshin:**

Only references change.
Figure 68: editGeneralizationSpecialElementFromAssociationToAssociation
14 Operations

14.1 createOperationInClass

creates a new operation in a class

Signature:

createOperationInClass(Class selectedEObject, String nameValue, Class typeValue)

Context element (selectedEObject):

The class providing the container for the newly created operation.

Additional parameters:

- nameValue/newName: The name of the newly created operation
- idValue/newID: The id of the newly created operation
- typeValue/newType: The type of the newly created operation

Precondition:

There is no operation in the same context whose name equals the parameter-value of 'newName' (see 2.1)

Implementation in EMF Henshin:

Only the name and the id will be set via input data. Visibility, isAbstract adn isStatic will be set with a default value as defined with the diagram editor in the image below. The 'newType' input data is a concrete propagated class.
14.2 `createOperationInInterface`

creates a new operation in an interface

**Signature:**

`createOperationInInterface(Interface selectedEObject, String nameValue, Class typeValue)`

**Context element (selectedEObject):**

The interface providing the container for the newly created operation.

**Additional parameters:**

- `nameValue/newName`: The name of the newly created operation
- `idValue/newID`: The id of the newly created operation
- `typeValue/newType`: The type of the newly created operation
**Precondition:**

There is no operation in the same context whose name equals the parameter-value of 'newName' (see 2.1)

**Implementation in EMF Henshin:**

Only the name and the id will be set via input data. Visibility, isAbstract adn isStatic will be set with a default value as defined with the diagram editor in the image below. The 'newType' input data is a concrete propagated interface.

![Diagram](image)

**Figure 70: createOperationInInterface**

### 14.3 deleteOperation

Deletes an operation

**Signature:**

`deleteOperation(Operation selectedEObject)`

**Context element (selectedEObject):**

The operation which should be deleted
**Additional parameters:**

- 

**Precondition:**

- 

**Implementation in EMF Henshin:**

For a better readability this is a simplified version of the ‘deleteOperation’-transformation and will only cover cases where the operation has no containments and no references to other elements. Such a complex transformation rule exits but won’t be listed here.

In this simplified version we have three rules. One that checks if the container of the operation is a model and the other rules that deletes the operation depending on the found container type. In the following image of the units you can see the case distinction done with a Conditional Unit. We assume the container is an interface if not a class.

![Figure 71: deleteOperation](image-url)
14.4 editOperationName

edits the name of an operation

**Signature:**

editOperationName(Operation selectedEObject, String nameValue)

**Context element (selectedEObject):**

The operation whose name should be renamed.

**Additional parameters:**

- nameValue/newName: The new name

**Precondition:**

There is no operation in the same class whose name equals the parameter-value of 'newName' (see 2.1)

**Implementation in EMF Henshin:**

The <<create>>-symbol in the image means that even if the attribute exists its value will be overwritten. 'someName' is the placeholder for the input name.
14.5 editOperationIsAbstract

edits the isAbstract-value of an operation

**Signature:**

\[
\text{editOperationIsAbstract(Operation selectedEObject, boolean booleanValue)}
\]

**Context element (selectedEObject):**

The operation whose isAbstract-value should be edited.

**Additional parameters:**

- booleanValue/bool: The new isAbstract-value

**Precondition:**

The \(<\text{create}>\)-symbol in the image means that even if the attribute exists its value will be overwritten. ‘bool’ is the placeholder for the input boolean value.

**Implementation in EMF Henshin:**

safsdfsd

14.6 editOperationIsStatic

edits the isStatic-value of an operation

**Signature:**

\[
\text{editOperationIsStatic(Operation selectedEObject, boolean booleanValue)}
\]
Context element (selectedEObject):
The operation whose isStatic-value should be edited.

Additional parameters:
- booleanValue/bool: The new isStatic-value

Precondition:

Implementation in EMF Henshin:
The <<create>>-symbol in the image means that even if the attribute exists its value will be overwritten. 'bool' is the placeholder for the input boolean value.

![Figure 75: editOperationIsStatic](image)

14.7 editOperationVisibility

edits the visibility of an operation

Signature:

editOperationVisibility(Operation selectedEObject, Visibility visibilityValue)

Context element (selectedEObject):
The operation whose visibility should be edited.

Additional parameters:
- visibilityValue/visibility: The new visibility

Precondition:


Implementation in EMF Henshin:

The <<create>>-symbol in the image means that even if the attribute exists its value will be overwritten. 'visibility' is the placeholder for the input visibility value.

```
<table>
<thead>
<tr>
<th>editOperationVisibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;preserve&gt;&gt;</td>
</tr>
<tr>
<td>selectedOperation:UMLOperation</td>
</tr>
<tr>
<td>☐ &lt;&lt;create&gt;&gt; visibility=visibility</td>
</tr>
</tbody>
</table>
```

Figure 76: editOperationVisibility

14.8 editOperationReturnTypeFromClassToClass

edits the type of an operation from a class to another

**Signature:**

```
editOperationReturnTypeFromClassToClass(Operation selectedEObject, Class typeValue)
```

**Context element (selectedEObject):**

The operation whose type should be edited.

**Additional parameters:**

- `typeValue/newType`: The new type

**Precondition:**


Implementation in EMF Henshin:

Only references will change.
14.9 editOperationReturnReturnTypeFromClassToPrimitive

edits the type of an operation from a class to a primitiveType

**Signature:**

\[
\text{editOperationReturnReturnTypeFromClassToPrimitive}(\text{Operation selectedEObject, PrimitiveType typeValue})
\]

**Context element (selectedEObject):**

The operation whose type should be edited.

**Additional parameters:**

- \(\text{typeValue/newType}\): The new type

**Precondition:**

-

**Implementation in EMF Henshin:**

Only references will change.
14.10 editOperationReturnTypeFromPrimitiveToClass

edits the type of an operation from a primitiveType to a class

**Signature:**

editOperationReturnTypeFromPrimitiveToClass(Operation selectedEObject, Class typeValue)

**Context element (selectedEObject):**

The operation whose type should be edited.

**Additional parameters:**

- typeValue/newType: The new type

**Precondition:**

- 

**Implementation in EMF Henshin:**

Only references will change.
14.11 editOperationReturnTypeFromPrimitiveToPrimitive

edits the type of an operation from a primitiveType to a primitiveType

**Signature:**

```java
editOperationReturnTypeFromPrimitiveToPrimitive(Operation selectedEObject, PrimitiveType typeValue)
```

**Context element (selectedEObject):**

The operation whose type should be edited.

**Additional parameters:**

- `typeValue/newType`: The new type

**Precondition:**

- 

**Implementation in EMF Henshin:**

Only references will change.
14.12 moveOperationBetweenClasses

moves an operation from a class to another class

**Signature:**

moveOperationBetweenClasses(Operation selectedEObject, Class tgt)

**Context element (selectedEObject):**

The operation which should be moved.

**Additional parameters:**

- tgt/tgtClass: the target class

**Precondition:**

There is no operation with the same name in the target context (see 2.1)

**Implementation in EMF Henshin:**

Only references change.
14.13 moveOperationBetweenInterfaces

moves an operation from an interface to another

**Signature:**

moveOperationBetweenInterfaces(Operation selectedEObject, Interface tgt)

**Context element (selectedEObject):**

The operation which should be moved.

**Additional parameters:**

- tgt/targetInterface: the target interface

**Precondition:**

There is no operation with the same name in the target context (see 2.1)

**Implementation in EMF Henshin:**

Only references change.
14.14 moveOperationFromClassToInterface

moves an operation from a class to an interface

**Signature:**

moveOperationFromClassToInterface(Operation selectedEObject, Interface tgt)

**Context element (selectedEObject):**

The operation which should be moved.

**Additional parameters:**

- tgt/targetInterface: the target interface

**Precondition:**

There is no operation with the same name in the target context (see §2.1)

**Implementation in EMF Henshin:**

Only references change.
14.15 moveOperationFromInterfaceToClass

moves an operation from an interface to a class

**Signature:**

moveOperationFromInterfaceToClass(Operation selectedEObject, Class tgt)

**Context element (selectedEObject):**

The operation which should be moved.

**Additional parameters:**

- tgt/targetClass: the target class

**Precondition:**

There is no operation with the same name in the target context (see 2.1)

**Implementation in EMF Henshin:**

Only references change.
Figure 84: moveOperationFromInterfaceToClass
15 Parameters

15.1 createParameter
creates a new parameter

Signature:
createParameter(Operation selectedEObject, String nameValue)

Context element (selectedEObject):
The operation providing the container for the newly created parameter.

Additional parameters:
- nameValue/newName: The name of the newly created parameter
- idValue/newID: The id of the newly created parameter

Precondition:
There is no parameter in the same context whose name equals the parameter-value of 'newName' (see 2.1)

Implementation in EMF Henshin:
Only the name and the id will be set via input data. kind will be set with a default value as defined with the diagram editor in the image below.

![Diagram of createParameter](image-url)

Figure 85: createParameter
15.2 deleteParameter

Deletes an parameter

Signature:

deleteParameter(Parameter selectedEObject)

Context element (selectedEObject):

The parameter which should be deleted

Additional parameters:

- 

Precondition:

- 

Implementation in EMF Henshin:

For a better readability this is a simplified version of the `deleteParameter`-transformation and will only cover cases where the parameter has no references to other elements. Such a complex transformation rule exits but won’t be listed here.

![Figure 86: createParameter](image)

15.3 editParameterName

edits the name of an parameter

Signature:

editParameterName(Parameter selectedEObject, String nameValue)
Context element (selectedEObject):
The parameter whose name should be renamed.

Additional parameters:
- nameValue/newName: The new name

Precondition:
There is no parameter in the same operation whose name equals the parameter-value of 'newName' (see 2.1)

Implementation in EMF Henshin:
The <<create>>-symbol in the image means that even if the attribute exists its value will be overwritten. 'someName' is the placeholder for the input name.

Figure 87: editParameterName

15.4 editParameterKind
Edits the kind-value (in;out) of a parameter

Signature:
editParameterKind(Parameter selectedEObject, Kind newParameterKindValue)

Context element (selectedEObject):
The parameter whose kind-value should be edited.

Additional parameters:
- newParameterKindValue/parameterKind: The new kind-value

Precondition:
-
Implementation in EMF Henshin:

The \texttt{create}-symbol in the image means that even if the attribute exists its value will be overwritten. 'parameterKind' is the placeholder for the input parameterKind.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{editParameterKind}
\caption{editParameterKind}
\end{figure}

15.5 \texttt{editParameterTypeFromClassToClass}

edits the type of an parameter from a class to another

\textbf{Signature:}

\texttt{editParameterTypeFromClassToClass(Parameter selectedEObject, Class typeValue)}

\textbf{Context element (selectedEObject):}

The parameter whose type should be edited.

\textbf{Additional parameters:}

- typeValue/newType: The new type

\textbf{Precondition:}

- 

\textbf{Implementation in EMF Henshin:}

Only references will change.
15.6 *editParameterTypeFromClassToPrimitive*

edits the type of an parameter from a class to a primitiveType

**Signature:**

```
editParameterTypeFromClassToPrimitive(Parameter selectedEObject, PrimitiveType typeValue)
```

**Context element (selectedEObject):**

The parameter whose type should be edited.

**Additional parameters:**

- `typeValue/newType`: The new type

**Precondition:**

- 

**Implementation in EMF Henshin:**

- Only references will change.
15.7 editParameterTypeFromPrimitiveToClass

edits the type of an parameter from a primitiveType to a class

**Signature:**

\[
\text{editParameterTypeFromPrimitiveToClass}(\text{Parameter } \text{selectedEObject}, \text{Class } \text{typeValue})
\]

**Context element (selectedEObject):**

The parameter whose type should be edited.

**Additional parameters:**

- typeValue/newType: The new type

**Precondition:**

- Only references will change.
15.8 editParameterTypeFromPrimitiveToPrimitive

edits the type of an parameter from a primitiveType to a primitiveType

**Signature:**

```
editParameterTypeFromPrimitiveToPrimitive(Parameter selectedEObject, PrimitiveType typeValue)
```

**Context element (selectedEObject):**

The parameter whose type should be edited.

**Additional parameters:**

- `typeValue/newType`: The new type

**Precondition:**

- 

**Implementation in EMF Henshin:**

- Only references will change.
15.9 moveParameter

moves a parameter from a operation to another operation

**Signature:**

moveParameter(Parameter selectedEObject, Operation tgt)

**Context element (selectedEObject):**

The parameter which should be moved.

**Additional parameters:**

- tgt/tgt[moveParameter]: the target operation

**Precondition:**

There is no parameter with the same name in the target context (see 2.1)

**Implementation in EMF Henshin:**

Only references will change.
Figure 93: moveParameter
16 Attributes

16.1 createAttribute

creates a new attribute

Signature:

cREATE_ATTRIBUTE(Class selectedEObject, String nameValue)

Context element (selectedEObject):

The class providing the container for the newly created attribute.

Additional parameters:

- nameValue/newName: The name of the newly created attribute
- idValue/idName: The id of the newly created attribute

Precondition:

There is no attribute in the same context whose name equals the parameter-value of 'newName' (see 2.1)

Implementation in EMF Henshin:

Only the name and the id will be set via input data. Visibility, isStatic, isFinal and isReadOnly will be set with a default value as defined with the diagram editor in the image below.

Figure 94: createAttribute
16.2 deleteAttribute

Deletes an attribute

**Signature:**

`deleteAttribute(Attribute selectedEObject)`

**Context element (selectedEObject):**

The attribute which should be deleted

**Additional parameters:**

- 

**Precondition:**

- 

**Implementation in EMF Henshin:**

For a better readability this is a simplified version of the 'deleteAttribute'-transformation and will only cover cases where the attribute has no references to other elements. Such a complex transformation rule exits but won’t be listed here.

![Figure 95: deleteAttribute](image)

16.3 editAttributeName

edits the name of an attribute

**Signature:**

`editAttributeName(Attribute selectedEObject, String nameValue)`
Context element (selectedEObject):
The attribute whose name should be renamed.

Additional parameters:
- nameValue/newName: The new name

Precondition:
There is no attribute in the same class whose name equals the parameter-value of 'newName' (see 2.1)

Implementation in EMF Henshin:
The “<<create>>” symbol in the image means that even if the attribute exists its value will be overwritten. ‘newName’ is the placeholder for the input name.

Figure 96: editAttributeName

16.4 editAttributeIsReadOnly
edits the isReadOnly-value of an attribute

Signature:
editAttributeIsReadOnly(Attribute selectedEObject, boolean booleanValue)

Context element (selectedEObject):
The attribute whose isReadOnly-value should be edited.

Additional parameters:
- booleanValue/bool: The new isReadOnly-value

Precondition:
Implementation in EMF Henshin:

The \(<\text{create}>\)-symbol in the image means that even if the attribute exists its value will be overwritten. 'bool' is the placeholder for the input boolean value.

\begin{figure}[h]
\centering
\includegraphics[width=0.4\textwidth]{editAttributeIsReadOnly.png}
\caption{editAttributeIsReadOnly}
\end{figure}

16.5 editAttributeIsStatic

edits the isStatic-value of an attribute

**Signature:**

\[
\text{editAttributeIsStatic} (\text{Attribute selectedEObject}, \text{boolean booleanValue})
\]

**Context element (selectedEObject):**

The attribute whose isStatic-value should be edited.

**Additional parameters:**

- booleanValue/bool: The new isStatic-value

**Precondition:**

- 

Implementation in EMF Henshin:

The \(<\text{create}>\)-symbol in the image means that even if the attribute exists its value will be overwritten. 'bool' is the placeholder for the input boolean value.

\begin{figure}[h]
\centering
\includegraphics[width=0.4\textwidth]{editAttributeIsStatic.png}
\caption{editAttributeIsStatic}
\end{figure}
16.6 editAttributeIsFinal

edits the isFinal-value of an attribute

**Signature:**

```
editAttributeIsFinal(Attribute selectedEObject, boolean booleanValue)
```

**Context element (selectedEObject):**

The attribute whose isFinal-value should be edited.

**Additional parameters:**

- booleanValue/bool: The new isFinal-value

**Precondition:**

```
-
```

**Implementation in EMF Henshin:**

The `<<create>>`-symbol in the image means that even if the attribute exists its value will be overwritten. `bool` is the placeholder for the input boolean value.

![editAttributeIsFinal](image)

Figure 99: editAttributeIsFinal

16.7 editAttributeVisibility

edits the visibility of an attribute

**Signature:**

```
editAttributeVisibility(Attribute selectedEObject, Visibility visibilityValue)
```

**Context element (selectedEObject):**

The attribute whose visibility should be edited.
**Additional parameters:**

- visibilityValue/visibility: The new visibility

**Precondition:**

- 

**Implementation in EMF Henshin:**

The `<<create>>`-symbol in the image means that even if the attribute exists its value will be overwritten. 'visibility' is the placeholder for the input visibility value.

![editAttributeVisibility](image)

Figure 100: editAttributeVisibility

### 16.8 editAttributeTypeFromClassToClass

Edits the type of an attribute from a class to another

**Signature:**

```java
editAttributeTypeFromClassToClass(Attribute selectedEObject, Class typeValue)
```

**Context element (selectedEObject):**

The attribute whose type should be edited.

**Additional parameters:**

- typeValue/newType: The new type

**Precondition:**

- 

**Implementation in EMF Henshin:**

Only references will change.
16.9 editAttributeTypeFromClassToPrimitive

edits the type of an attribute from a class to a primitiveType

Signature:

\[
\text{editAttributeTypeFromClassToPrimitive}(\text{Attribute } \text{selectedEObject}, \text{ PrimitiveType } \text{typeValue})
\]

Context element (selectedEObject):

The attribute whose type should be edited.

Additional parameters:

- \(\text{typeValue/newType}\): The new type

Precondition:

- 

Implementation in EMF Henshin:

Only references will change.
16.10 **editAttributeTypeFromClassToPrimitive**

edits the type of an attribute from a primitiveType to a class

**Signature:**

```plaintext
editAttributeTypeFromClassToPrimitive(Attribute selectedEObject, Class typeValue)
```

**Context element (selectedEObject):**

The attribute whose type should be edited.

**Additional parameters:**

- `typeValue/newType`: The new type

**Precondition:**

- 

**Implementation in EMF Henshin:**

Only references will change.
16.11 editAttributeTypeFromPrimitiveToPrimitive

edits the type of an attribute from a primitiveType to a primitiveType

**Signature:**

editAttributeTypeFromPrimitiveToPrimitive(Attribute selectedEObject, PrimitiveType typeValue)

**Context element (selectedEObject):**

The attribute whose type should be edited.

**Additional parameters:**

- typeValue/newType: The new type

**Precondition:**

- 

**Implementation in EMF Henshin:**

Only references will change.
16.12 moveAttribute

moves an attribute from a class to another class

**Signature:**

moveAttributeBetweenClasses(Attribute selectedEObject, class tgt)

**Context element (selectedEObject):**

The attribute which should be moved.

**Additional parameters:**

- tgt/tgt[moveAttribute]: the target class

**Precondition:**

There is no attribute with the same name in the target context (see 2.1)

**Implementation in EMF Henshin:**

Only references will change.
Figure 105: moveAttribute
17 References

References

