Departamento de Ingeniería de Sistemas y Computación
Universidad de los Andes

EnAr-Fusion. A Tool for Metamodel Composition

Héctor Flórez, Mario Sánchez, Jorge Villalobos

Technical Report ISIS-01-2012

2012

Departamento de Ingeniería de Sistemas y Computación
Universidad de los Andes

EnAr-Fusion. A Tool for Metamodel Composition

Héctor Flórez, Mario Sánchez, Jorge Villalobos
Departamento de Ingeniería de Sistemas y Computación
Universidad de los Andes
Bogotá, Colombia
{ha.florez39, mar-san1, jvillalo}@uniandes.edu.co

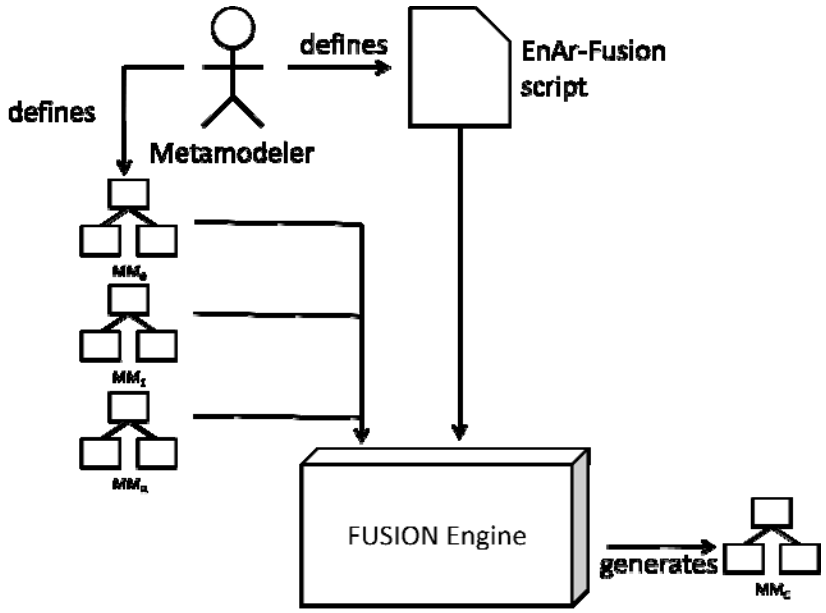
Technical Report ISIS-01-2012

2012

Abstract
This technical report presents EnAr-Fusion that is a domain specific language for metamodels composition. EnAr-Fusion provides a language with several instructions to adapt and compose elements based from existing metamodels.

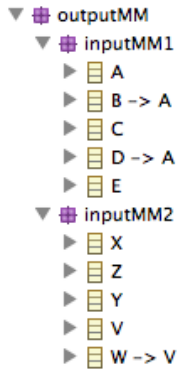
1. Introduction
EnAr-Fusion is a tool to perform metamodels composition based on a domain specific language that offers instructions to adapt input metamodels and compose these metamodel generating a new composite metamodel.

2. EnAr-Fusion
EnAr-Fusion is a domain specific language (DSL) to compose and refine our Enterprise Architecture domain metamodels. However, before starting to use this language, it is necessary to specify these metamodels by using a metamodeling language. In this case, we used Ecore to specify metamodels. Then, these metamodels are specified using a common set of concepts from the metamodeling language.

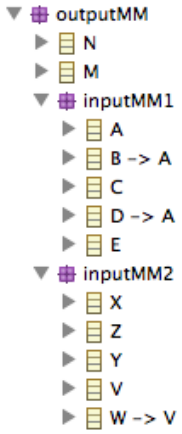


The language was created through XTEXT. Each instruction requires several transformations on the input metamodels that have been perform using EMF.

2.1. Execution
Once the script has been written, the engine can execute it. The execution process imports the metamodels required for the composition. Those metamodels are organized in the correspondent package inside the composite metamodel package. Next figure shows the distribution of the elements in two imported metamodels.



The EClasses created in execution time, will be placed in the composite metamodel package. For instance, next figure represents the distribution of elements after the creation of the EClasses N and M



The composition process ends when the composite metamodel is exported. The engine removes the imported metamodel packages placing all elements in the composite metamodel package. After this operation, the composite metamodel is created.

2.2. Language

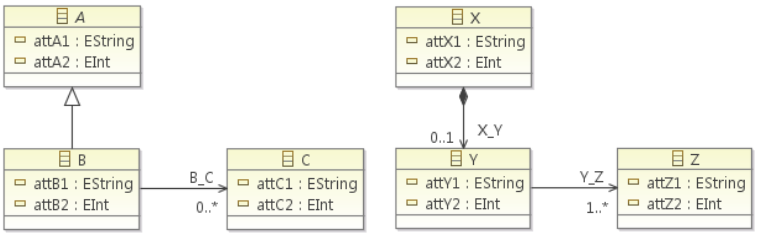
The language allows in a single transformation, adapting metamodels and the composition of adapted metamodels. To achieve this, EnAr-Fusion offers 18 operations that can act on the input metamodels and on the composite metamodel. These operations are classified in two groups: 1) declarations and 2) Instructions. Declarations allow importing input metamodels and exporting an output metamodel. Instructions allow performing the composite metamodel. The instructions can affect entities, which are equivalent to EClasses; attributes, which are equivalent to EAttributes; and links, which are equivalent to EReferences.

The set of operations supported by EnAr-Fusion is presented in the next table.

Type	Operation
Declarations	import
	export
Instructions	newEntity
	deleteEntity
	renameEntity

	setAbstractEntity
	setNonAbstractEntity
	addAttribute
	removeAttribute
	updateAttribute
	renameAttribute
	newLink
	deleteLink
	updateLink
	newInheritanceLink
	deleteInheritanceLink
	mergeEntities
	splitEntity

In order to explain the language, we use the next two metamodels called mm1 and mm2. We assume that the packages containing the Ecore metamodels are called exampleMM1 and exampleMM2.



The language operations are:

1. `import <metamodel path>`
   Imports the metamodel located in the path `<metamodel path>`.
   The next script imports the metamodel mm1.ecore. The package of the metamodel mm1, may not exist in the metamodels imported before.

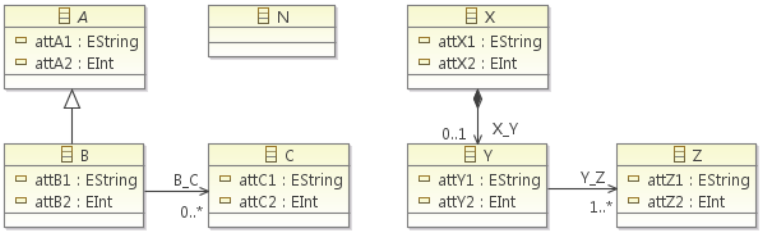
```
import "./models/mm1.ecore"
```

2. `export <metamodel name> <metamodel URI>`
   Exports the composite metamodel named `<metamodel name>` with the URI `<metamodel URI>`
   The next script exports the composite metamodel named mm3.

```
export "mm3" "http://uniandes.edu.co/exampleMM3"
```

3. `newEntity <entity name>`
   Creates a new entity named `<entity name>` in the composite metamodel. This entity has neither attributes nor relationships with other elements.
   The next script creates an entity called “N”. The entity “N” may not exist in the composite metamodel. The composite metamodel generated after this instruction using the metamodels mm1 and mm2 described above is presented in the next figure.

```
newEntity N
```

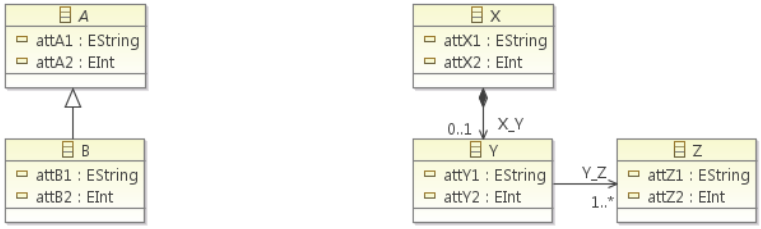


4. `deleteEntity <entity name>`

   Deletes the entity named `<entity name>` in the composite metamodel.

   The next script deletes the entity called “C” from the metamodel mm1. The entity “C” must exist in the metamodel mm1. The composite metamodel generated after this instruction using the metamodels mm1 and mm2 described above is presented in the next figure.

```
deleteEntity exampleMM1.C
```

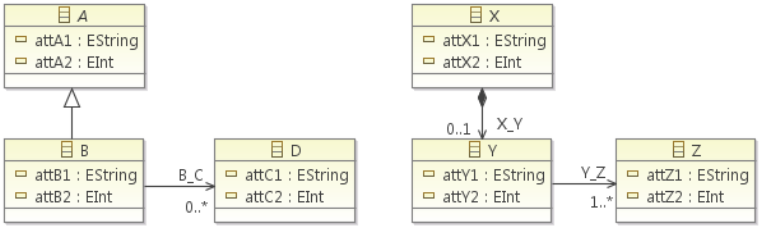


5. `renameEntity <entity name> <new entity name>`

   Renames the entity named `<entity name>` with `<new entity name>` in the composite metamodel.

   The next script renames the entity called “C” from the metamodel mm1 setting the new name “D”. The entity “C” must exist in the metamodel mm1. The entity “D” may not exist in the composite metamodel. The composite metamodel generated after this instruction using the metamodels mm1 and mm2 described above is presented in the next figure.

```
renameEntity exampleMM1.C “D”
```

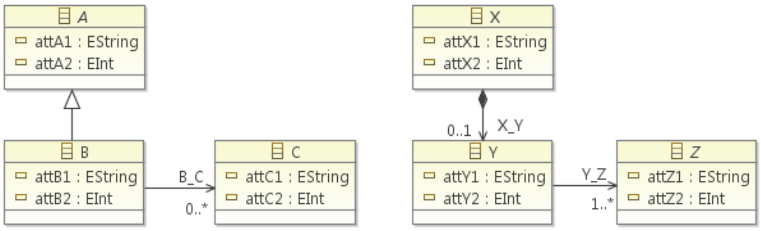


6. `setAbstractEntity <entity name>`

   Converts the entity `<entity name>` in an abstract entity in the composite metamodel.

   The next script sets abstract the entity called “Z” from the metamodel mm2. The entity “Z” must exist in the metamodel mm2. The composite metamodel generated after this instruction using the metamodels mm1 and mm2 described above is presented in the next figure.

```
setAbstractEntity exampleMM2.Z
```

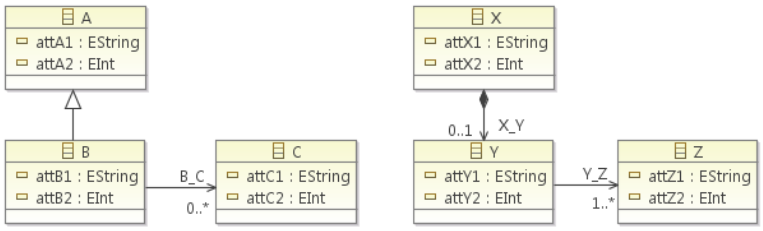


7. `setNonAbstractEntity <entity name>`

   Converts the abstract entity `<entity name>` in non abstract entity in the composite metamodel.

   The next script sets non abstract the entity called “A” from the metamodel mm1. The entity “A” must exist in the metamodel mm1. The composite metamodel generated after this instruction using the metamodels mm1 and mm2 described above is presented in the next figure.

```
setNonAbstractEntity exampleMM1.A
```

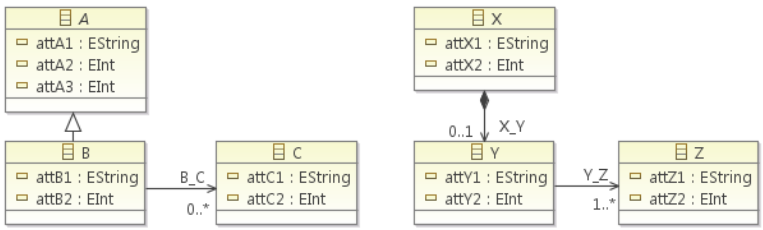


8. `addAttribute <attribute name>{`
   `source=<entity name>`
   `type=<attribute type>`
   `optional=<true|false>`
   `}`

   Adds the attribute `<attribute name>` into the entity `<entity name>` in the composite metamodel. This attribute must have type and the optional property.

   The next script adds the attribute called “attA3” with type “EInt” into the entity called “A” from the metamodel mm1. The entity “A” must exist in the metamodel mm1. The attribute “attA3” may not exist in the entity “A”. The composite metamodel generated after this instruction using the metamodels mm1 and mm2 described above is presented in the next figure.

```
addAttribute attA3{
  source=exampleMM1.A
  type=EInt
  optional=true
}
```

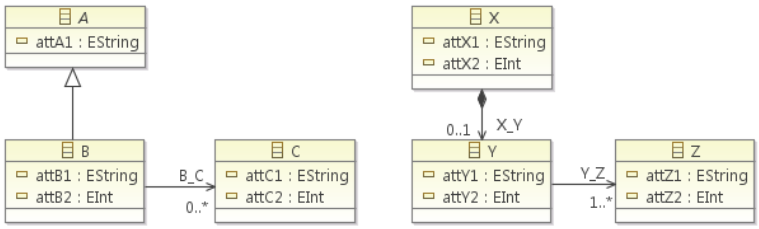


9. `removeAttribute <entity name>.<attribute name>`

Removes the attribute `<attribute name>` from the entity `<entity name>` in the composite metamodel.

The next script removes the attribute called “attA2” from the entity called “A” from the metamodel mm1. The entity “A” must exist in the metamodel mm1. The attribute “attA2” must exist in the entity “A”. The composite metamodel generated after this instruction using the metamodels mm1 and mm2 described above is presented in the next figure.

```
removeAttribute exampleMM1.A.attA2
```

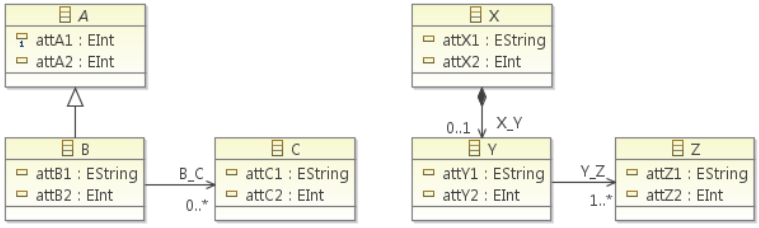


10. `updateAttribute <entity name>.<attribute name>`
    `type=<attribute type>`
    `optional=<true|false>`
    `}`

    Updates the attribute `<attribute name>` from the entity `<entity name>` in the composite metamodel. This attribute must have type and the optional property.

    The next script updates the attribute called “attA1” from the entity called “A” from the metamodel mm1. The entity “A” must exist in the metamodel mm1. The attribute “attA1” must exist in the entity “A”. The composite metamodel generated after this instruction using the metamodels mm1 and mm2 described above is presented in the next figure.

```
updateAttribute exampleMM1.A.attA1{
  type=EInt
  optional=false
}
```

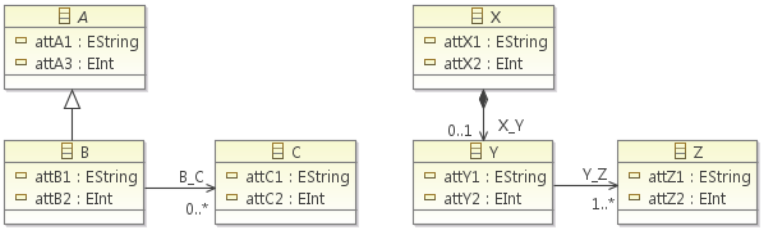


11. `renameAttribute <entity name>.<attribute name> <new attribute name>`

    Renames the attribute `<attribute name>` from the entity `<entity name>` with `<new attribute name>` in the composite metamodel.

    The next script renames the attribute called “attA2” from the entity called “A” from the metamodel mm1. The entity “A” must exist in the metamodel mm1. The attribute “attA2” must exist in the entity “A”. The attribute “attA3” may not exist in the entity “A”. The composite metamodel generated after this instruction using the metamodels mm1 and mm2 described above is presented in the next figure.

```
renameAttribute exampleMM1.A.attA2 “attA3”
```

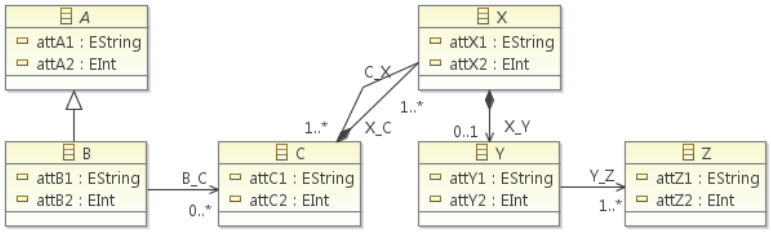


12. `newLink <link name>{`
    `source=<source entity name>`
    `target=<target entity name>`
    `containment=<true|false>`
    `minCard=<min card value>`
    `maxCard=<max card value>`
    `associatedLink=<associated link name>`
    `}`

    Establishes a new link named `<min card value>` between the entities `<source entity name>` and `<target entity name>` in the composite metamodel. The link can be containment and has a `<min card value>` and `<max card value>`. Also, it can be created with an opposite link named `<associated link name>`.

    The next script creates a link called “C_X” from the entity called “C” from the metamodel mm1 to the entity called “X” from the metamodel mm2. This link also has containment true, minimum cardinality 1 maximum cardinality -1 that is equivalent to *. Finally, it creates an opposite link with the same features called “X_C”. The entity “C” must exist in the metamodel mm1. The entity “X” must exist in the metamodel mm2. The Link “C_X” may not exist in the composite metamodel. Then minCard must be greater or equal than 0. The maxCard must be greater or equal than -1. The composite metamodel generated after this instruction using the metamodels mm1 and mm2 described above is presented in the next figure.

```
newLink C_X{
  source=exampleMM1.C
  target=exampleMM2.X
  containment=true
  minCard=1
  maxCard=-1
  associatedLink=X_C
}
```



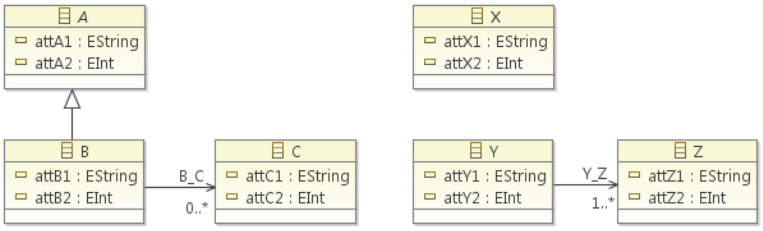
13. `deleteLink <source entity name>.<link name>`

    Deletes the link `<link name>` placed in the entity `<source entity name>` the composite metamodel.

    The next script deletes a link called “X_Y” from the entity called “X” from the metamodel mm2. The link “X_Y” must belong to the entity X. The entity X must exist in the

metamodel mm2. The composite metamodel generated after this instruction using the metamodels mm1 and mm2 described above is presented in the next figure.

```
deleteLink exampleMM2.X.X_Y
```

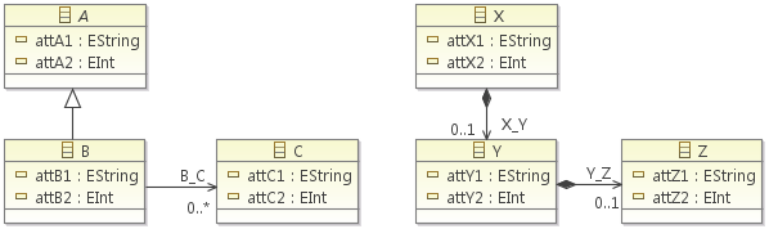


14. `updateLink <entity name>.<link name>{`
    `containment=<true|false>`
    `minCard=<min card value>`
    `maxCard=<miax card value>`
    `}`

    Updates the link `<link name>` places in the entity `<entity name>` in the composite metamodel. The link can be containment and has a `<min card value>` and `<max card value>`.

    The next script updates a link called “Y_Z” from the entity called “Y” from the metamodel mm2. This link also has containment true, minimum cardinality 0 maximum cardinality 1. The link “Y_Z” must belong to the entity Y. The entity Y must exist in the metamodel mm2. The minCard must be greater or equal than 0. The maxCard must be greater or equal than -1. The composite metamodel generated after this instruction using the metamodels mm1 and mm2 described above is presented in the next figure.

```
updateLink exampleMM2.Y.Y_Z{
  containment=true
  minCard=0
  maxCard=1
}
```



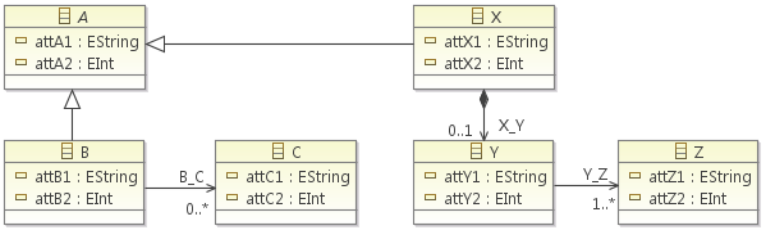
15. `newInheritanceLink{`
    `subEntity=<sub entity name>`
    `superEntity=<super entity name>`
    `}`

    Creates an inheritance link between the sub entity `<sub entity name>` and the super entity `<super entity name>` in the composite metamodel.

    The next script creates an inheritance link from the sub entity called “X” from the metamodel mm2 to the super entity called “A” from the metamodel mm2. The entity “X” must exist in the metamodel mm2. The entity “A” must exist in the metamodel mm1. The composite metamodel generated after this instruction using the metamodels mm1 and mm2 described above is presented in the next figure.

```
newInheritanceLink{
  subEntity=exampleMM2.X
  superEntity=exampleMM1.A
```

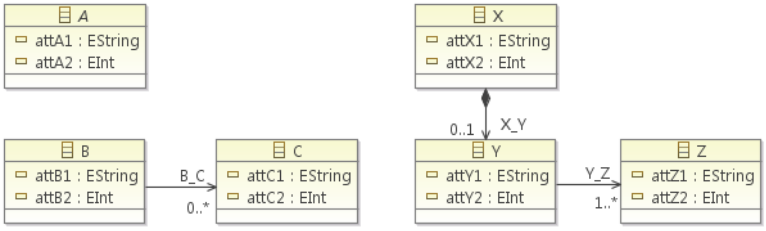
```
}
```



16. `deleteInheritanceLink <sub entity name>`

    Deletes an inheritance link of the sub entity `<sub entity name>` in the composite metamodel. The next script deletes the inheritance link from the sub entity called “B” from the metamodel mm1. The entity “B” must exist in the metamodel mm1. The entity “B” should have an inheritance link. The composite metamodel generated after this instruction using the metamodels mm1 and mm2 described above is presented in the next figure.

```
deleteInheritanceLink exampleMM1.B
```

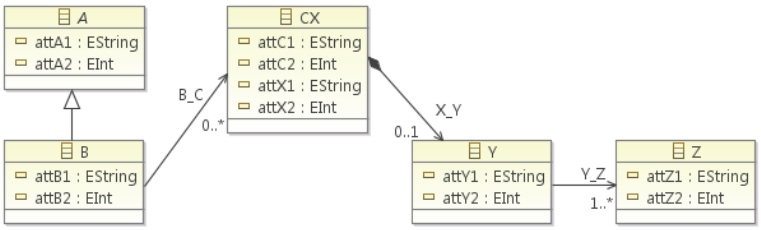


17. `mergeEntities <new entity name>{`
    `entity1=<entity name 1>`
    `entity2=<entity name 2>`
    `}`

    Creates the new entity `<new entity name>` from the existing entities `<entity name 1>` and `<entity name 2>` in the composite metamodel. The new entity is a mixture of the two previous entities and groups all attributes and links that existed before.

    The next script creates an entity based on the entity called “C” from the metamodel mm1 and the entity called “X” from the metamodel mm2. The entity “C” must exist in the metamodel mm1. The entity “X” must exist in the metamodel mm2. The name of the attributes in the entity “C” must be different than the name of the attributes in the entity “X”. The composite metamodel generated after this instruction using the metamodels mm1 and mm2 described above is presented in the next figure.

```
mergeEntities CX{
  entity1=exampleMM1.C
  entity2=exampleMM2.X
}
```



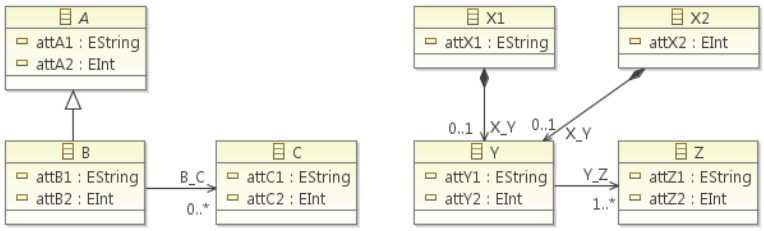
18. `splitEntity <entity name>{`
    `entity <entity name 1>{`

```
    Attributes{
      <List of attributes 1>
    }
    Links{
      <List of links 1>
    }
  }
  entity <entity name n>{
    Attributes{
      <List of attributes n>
    }
    Links{
      <List of links n>
    }
  }
}
```

Creates the new entities `<entity name 1>` and `<entity name 2>` in the composite metamodel, from an existing entity `<entity name>`. The entity's attributes and entity's links should be shared between the created entities.

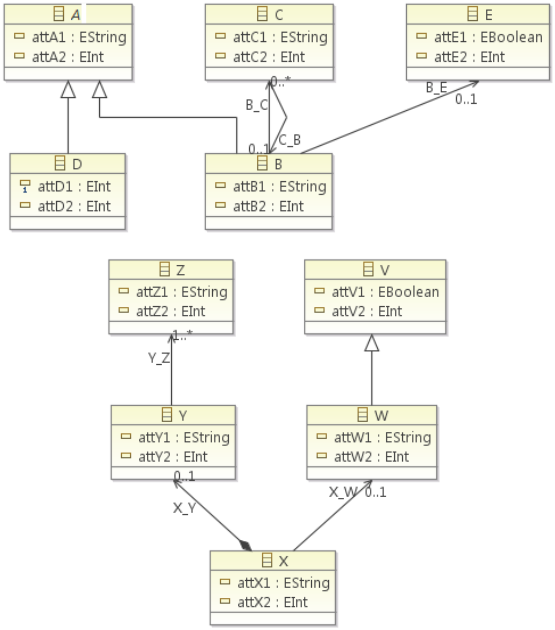
The next script creates an entity called “X1” based on the entity called “X” from the metamodel mm2 and includes the attribute called “attX1” and the link called “X_Y”. Also, creates another entity called “X2” based on the entity called “X” from the metamodel mm2 and includes the attribute called “attX2” and the link called “X_Y”. The entity “X” must exist in the metamodel mm2. The entity “X1” may not exist in the composite metamodel. The entity “X2” may not exist in the composite metamodel. At least must be implemented two split entities. All attributes in the entity “X”, must be assigned in at least one split entity. All links over the entity “X”, must be assigned in at least one split entity. The composite metamodel generated after this instruction using the metamodels mm1 and mm2 described above is presented in the next figure.

```
splitEntity exampleMM2.X{
  entity X1{
    Attributes{
      attX1
    }
    Links{
      X_Y
    }
  }
  entity X2{
    Attributes{
      attX2
    }
    Links{
      X_Y
    }
  }
}
```



3. Example

In order to illustrate how EnAr-Fusion works, we consider the next two metamodels.

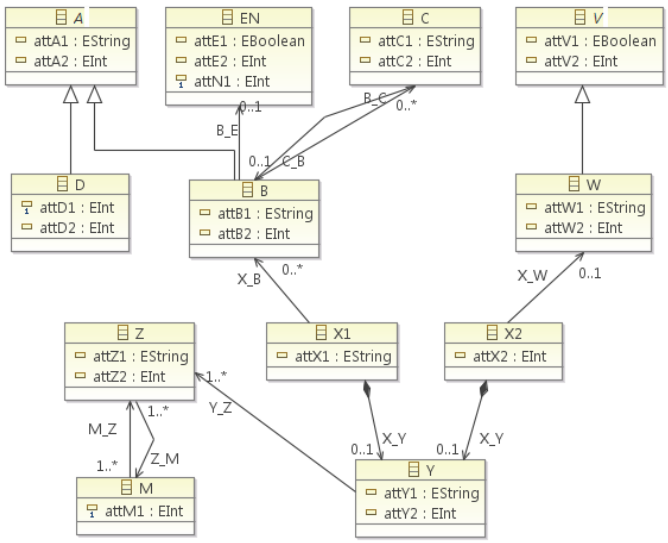


And the next EnAr-Fusion source code

```
import "./models/mm1.ecore"
import "./models/mm2.ecore"
export "mm3" "http://uniandes.edu.co/exampleMM3"

newEntity N

addAttribute attN1 {
  source = N
  type = EInt
  optional = false
}

newEntity M

addAttribute attM1 {
  source = M
  type = EInt
  optional = false
}

setAbstractEntity exampleMM2.V

mergeEntities EN {
  entity1 = exampleMM1.E
  entity2 = N
}

newLink M_Z {
  source = M
  target = exampleMM2.Z
  containment = false
  minCard = 1
  maxCard = -1
  associatedLink = Z_M
```

```
}

newLink X_B {
  source = exampleMM2.X
  target = exampleMM1.B
  containment = false
  minCard = 0
  maxCard = -1
}

splitEntity exampleMM2.X {
  entity X1{
    Attributes {
      attX1
    }
    Links {
      X_Y
      X_B
    }
  }
  entity X2 {
    Attributes {
      attX2
    }
    Links {
      X_Y
      X_W
    }
  }
}
```

The result of the execution of this script having the previous input metamodels is presented in next figure.



4. EnAr-Fusion Grammar

The following specification describes the concrete syntax of EnAr-Fusion. The notation used to describe this syntax is extended Backus-Naur-Form (EBNF) through the framework XTEXT.

```
grammar co.edu.uniandes.enar.Fusion with org.eclipse.xtext.common.Terminals

import "http://www.eclipse.org/emf/2002/Ecore" as ecore
```

```
generate fusion "http://www.edu.co/uniandes/enar/Fusion"

Model returns Model:
	{Model}
	(imports+=Import)*
	export=Export
	(actions+=Action)*
;

Import returns Import:
	{Import}
	'import ' importURI=STRING
;

Export returns Export:
	{Export}
	'export ' prefix=STRING ' ' exportURI=STRING
;

Action:
	EntityAction|
	AttributeAction|
	LinkAction|
	CompositionAction
;

EntityAction:
	NewEntity|
	DeleteEntity|
	RenameEntity|
	SetAbstractEntity|
	SetNonAbstractEntity
;

AttributeAction:
	AddAttribute|
	RemoveAttribute|
	UpdateAttribute|
	RenameAttribute
;

LinkAction:
	NewLink|
	DeleteLink|
	UpdateLink|
	NewInheritanceLink|
	DeleteInheritanceLink
;

CompositionAction:
	MergeEntities|
	SplitEntity
;

NewEntity returns NewEntity:
	{NewEntity}
	'newEntity ' entityName=ID
;

DeleteEntity returns DeleteEntity:
	{DeleteEntity}
	'deleteEntity ' (entityPackage=ID '.')? entityName=ID
;

RenameEntity returns RenameEntity:
	{RenameEntity}
	'renameEntity ' (entityPackage=ID '.')? entityName=ID ' ' newEntityName=STRING
;

SetAbstractEntity returns SetAbstractEntity:
	{SetAbstractEntity}
```

```
        'setAbstractEntity ' (entityPackage=ID '.')? entityName=ID
;

SetNonAbstractEntity returns SetNonAbstractEntity:
        {SetNonAbstractEntity}
        'setNonAbstractEntity ' (entityPackage=ID '.')? entityName=ID
;

AddAttribute returns AddAttribute:
        {AddAttribute}
        'addAttribute ' attributeName=ID '{'
                'source' '=' (entityPackage=ID ".")? entityName=ID
                'type' '=' type=Domain
                'optional' '=' optional=Bool
        '}'
;

RemoveAttribute returns RemoveAttribute:
        {RemoveAttribute}
        'removeAttribute ' (entityPackage=ID '.')? entityName=ID '.' attributeName=ID
;

UpdateAttribute returns UpdateAttribute:
        {UpdateAttribute}
        'updateAttribute ' (entityPackage=ID '.')? entityName=ID '.' attributeName=ID '{'
                'type' '=' type=Domain
                'optional' '=' optional=Bool
        '}'
;

RenameAttribute returns RenameAttribute:
        {RenameAttribute}
        'renameAttribute ' (entityPackage=ID '.')? entityName=ID '.' attributeName=ID ' '
newAttributeName=STRING
;

NewLink returns NewLink:
        {NewLink}
        'newLink ' linkName=ID '{'
                'source' '=' (sourceEntityPackage=ID ".")? sourceEntityName=ID
                'target' '=' (targetEntityPackage=ID ".")? targetEntityName=ID
                'containment' '='  containment=Bool
                'minCard' '='  minCard=INT
                'maxCard' '='  maxCard=SIGNED_INT
                ('associatedLink' '='  associatedLink=ID)?
        '}'
;

DeleteLink returns DeleteLink:
        {DeleteLink}
        'deleteLink ' (entityPackage=ID '.')? entityName=ID '.' linkName=ID
;

UpdateLink returns UpdateLink:
        {UpdateLink}
        'updateLink ' (entityPackage=ID '.')? entityName=ID '.' linkName=ID '{'
                'containment' '='  containment=Bool
                'minCard' '=' minCard=INT
                'maxCard' '=' maxCard=SIGNED_INT
        '}'
;

NewInheritanceLink returns NewInheritanceLink:
        {NewInheritanceLink}
        'newInheritanceLink' '{'
                'subEntity' '=' (subEntityPackage=ID ".")? subEntityName=ID
                'superEntity' '=' (superEntityPackage=ID ".")? superEntityName=ID
        '}'
;

DeleteInheritanceLink returns DeleteInheritanceLink:
```

```
	{DeleteInheritanceLink}
	'deleteInheritanceLink ' (entityPackage=ID '.')? entityName=ID
;

MergeEntities returns MergeEntities:
	{MergeEntities}
	'mergeEntities ' mergedEntityName=ID '{'
		'entity1' '=' (entityPackage1=ID ".")? entityName1=ID
		'entity2' '=' (entityPackage2=ID ".")? entityName2=ID
	'}'
;

SplitEntity returns SplitEntity:
	{SplitEntity}
	'splitEntity ' entityPackage=ID '.' entityName=ID '{'
		(entities += SplitEntityEntities)*
	'}'
;

SplitEntityEntities returns SplitEntityEntities:
	{SplitEntityEntities}
	'entity ' entityName=ID '{'
		attributes=SplitEntityAttributes
		links=SplitEntityLinks
	'}'
;

SplitEntityAttributes returns SplitEntityAttributes:
	{SplitEntityAttributes}
	'Attributes' '{'
		(attributesNames+=ID)*
	'}'
;

SplitEntityLinks returns SplitEntityLinks:
	{SplitEntityLinks}
	'Links' '{'
		(linksNames+=ID)*
	'}'
;

SIGNED_INT: ('+'|'-')? INT;

enum Domain:
	EBoolean|EDate|EDouble|EInt|EString
;

enum Bool :
	false|true
;
```

5. References

[1]. Steinberg Dave, Budinsky Frank, Paternostro Marcelo, Merks Ed. EMF: Eclipse Modeling Framework, 2nd Edition. Second Edition. 2009.

[2]. Clayberg Eric, Rubel Dan. Eclipse Plug-ins. Third Edition. 2008.

[3]. Eclipse Modeling Framework Project (EMF). http://www.eclipse.org/modeling/emf/

[4]. Xtext. http://www.eclipse.org/Xtext/