UNIVERSITÀ DEGLI STUDI DELL'AQUILA

Adapting Acceleo transformations after BNR metamodel changes

Model Driven Engineering Project

Manuel Dell'Elce
ID: 230340
mh025r0x@gmail.com

22/01/2015

Table of contents

Case study

Metamodel

The metamodel that has been built as a base for the transformation process outlines the domain of an HTML5 web application.

The main class, **WebApp**, has a router and a set of views, models, collections, controller, templates and styles.

A **Router** redirects user requests to controllers as defined in its set of **RouterBinding** elements, each of them binding a particular url pattern to a specific Controller.

Controller is an abstract class extended by two specific types of controllers: PageController and ServiceController; both of them using a set of parameters (Attribute), which should be passed along by the router through extraction from the user request, to act accordingly.

The **PageController** displays a page (View), while the **ServiceController** is meant to execute services that does not require user intervention, such as API calls, by using the API endpoint specified in its definition.

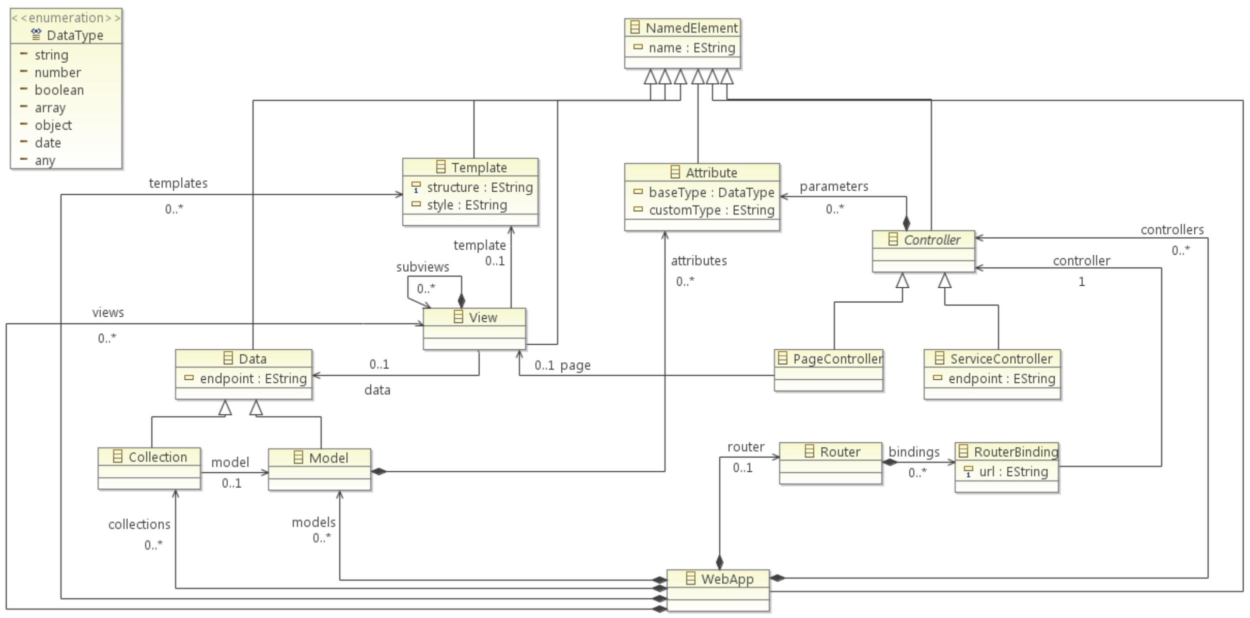
A **View** task is to manage a certain region of the DOM, optionally by delegating some subregions to subviews, by displaying a Template filled with Data passed along by the PageController.

The **Template** contains the structure, e.g. HTML, by using some syntax for defining placeholders to fill with data, and the style rules as a string to apply to the structure, e.g. by using CSS.

A **Data** represents the M part in the MVC model, associated with a remote backend instance by means of an endpoint, is extended by Collection and Model to represent an abstract representation of the WebApp domain.

A **Collection** is used to display models of a given Model type, while a **Model** itself is a domain entity having a set of attributes representing its content.

The **Attribute** is a general way to specify a value that can be either base, i.e. of a type of the **DataType** enum, or custom, by extending the base type.

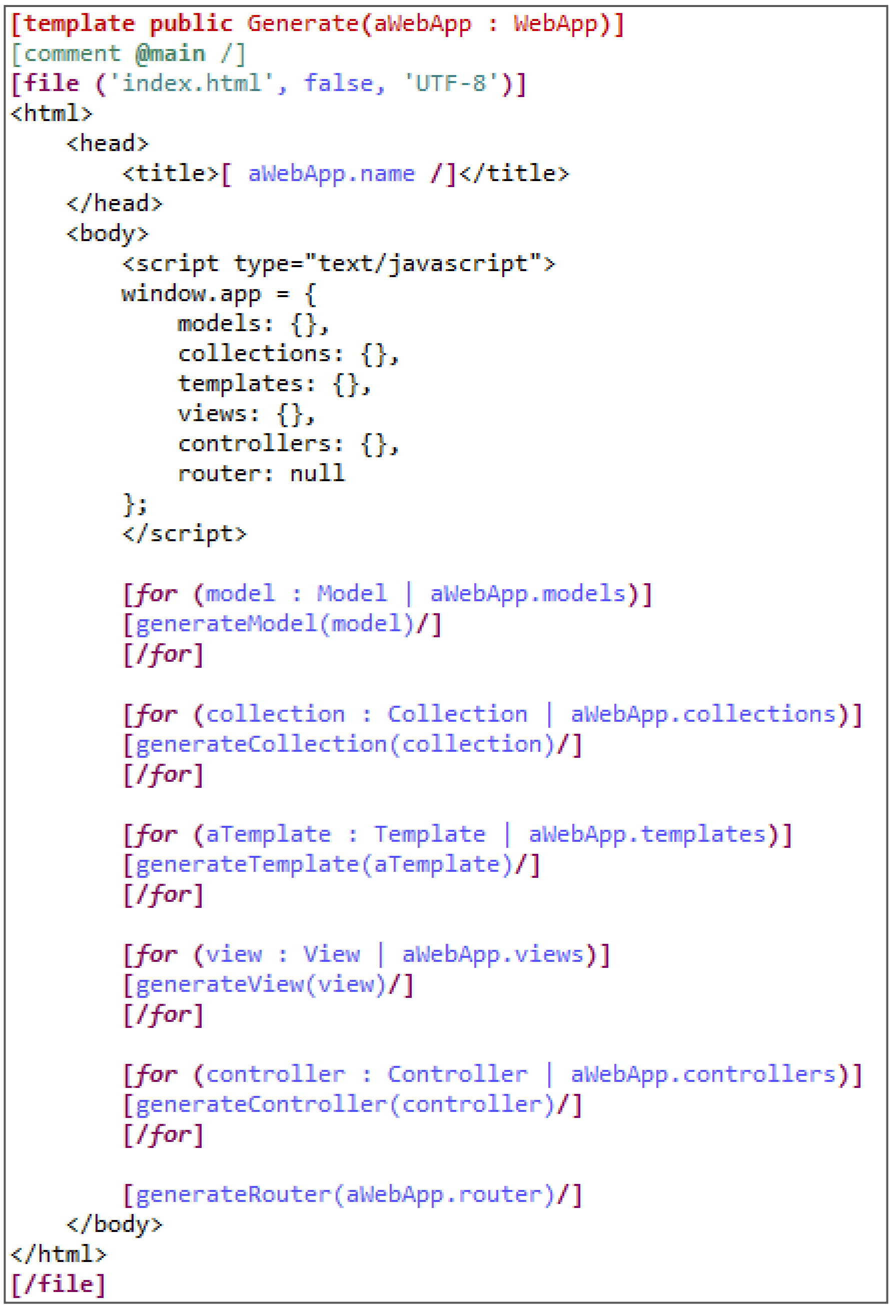


Metamodel diagram

Acceleo transformation

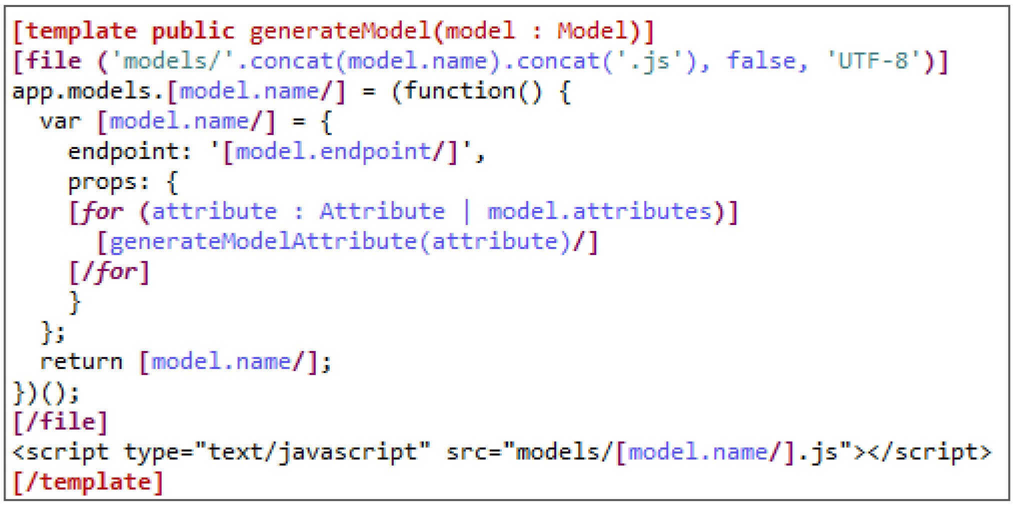
An Acceleo module has been built to generate the HTML+JS files of the Web Application, i.e the index.html used as a main entry-point and JS files for each component: models, collections, templates, views, controller and the router.

Please note that the generated components are meant to represent the WebApplication modules, or types/classes, that can be used with the help of a framework or library to further define the application logic.



```
[template public Generate(aWebApp : WebApp)]
[comment @main /]
[file ('index.html', false, 'UTF-8')]
<html>
    <head>
        <title>[ aWebApp.name /]</title>
    </head>
    <body>
        <script type="text/javascript">
        window.app = {
            models: {},
            collections: {},
            templates: {},
            views: {},
            controllers: {},
            router: null
        };
        </script>

        [for (model : Model | aWebApp.models)]
        [generateModel(model)/]
        [/for]

        [for (collection : Collection | aWebApp.collections)]
        [generateCollection(collection)/]
        [/for]

        [for (aTemplate : Template | aWebApp.templates)]
        [generateTemplate(aTemplate)/]
        [/for]

        [for (view : View | aWebApp.views)]
        [generateView(view)/]
        [/for]

        [for (controller : Controller | aWebApp.controllers)]
        [generateController(controller)/]
        [/for]

        [generateRouter(aWebApp.router)/]
    </body>
</html>
[/file]
```

Acceleo main template

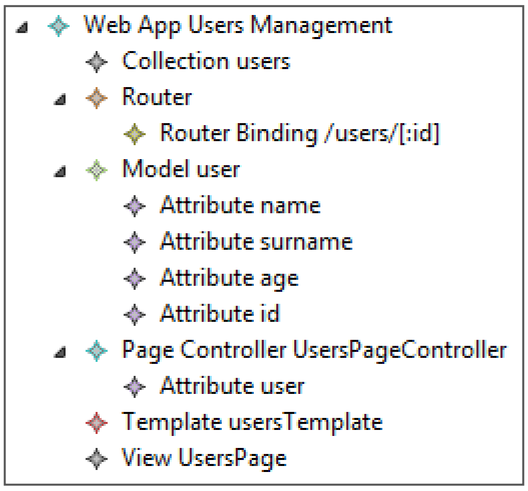


```
[template public generateModel(model : Model)]
[file ('models/'.concat(model.name).concat('.js'), false, 'UTF-8')]
app.models.[model.name/] = (function() {
  var [model.name/] = {
    endpoint: '[model.endpoint/]',
    props: {
    [for (attribute : Attribute | model.attributes)]
      [generateModelAttribute(attribute)/]
    [/for]
    }
  };
  return [model.name/];
})();
[/file]
<script type="text/javascript" src="models/[model.name/].js"></script>
[/template]
```

Acceleo generateModel template

Model

Consider the following model representing a fragment of a web application that displays a page with the details of a given user:

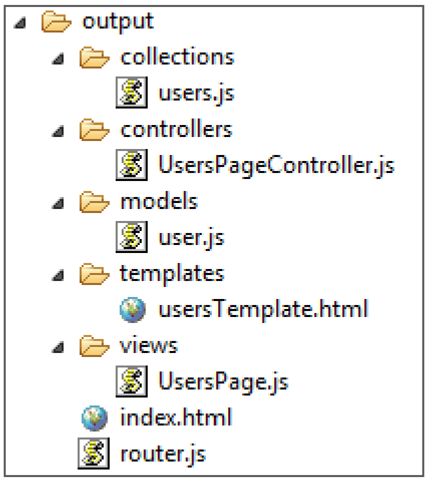


```
Web App Users Management
    Collection users
    Router
        Router Binding /users/[:id]
    Model user
        Attribute name
        Attribute surname
        Attribute age
        Attribute id
    Page Controller UsersPageController
        Attribute user
    Template usersTemplate
    View UsersPage
```

Model hierarchical representation

- The router binds the “/users/[:id]/ url with the UsersPageController, where [:id] is a pattern representing the user id as a string, e.g “/users/12/”.
- The UsersPageController, given the id of the user as an Attribute, displays the UsersPage.
- The UsersPage displays the user details by using the user Model and the usersTemplate.
- An user has a name, a surname, an age and an id, all of base type string except the age being a number.
- The usersTemplate contains html and css of the users page.

By giving this model as input to the Acceleo transformation, the generated files are the following:

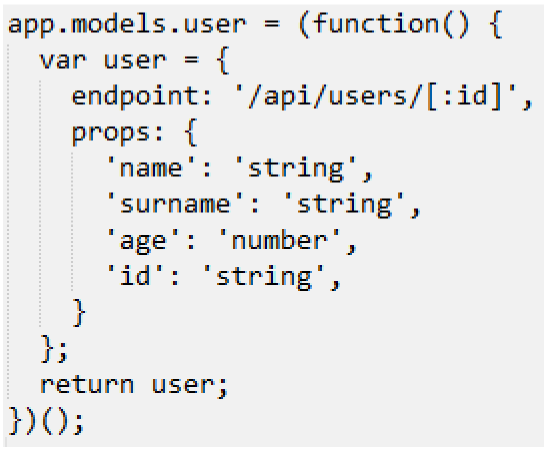


Generated HTML+JS app files

Where the generated file content for the main template and the generateModel template seen before is shown below:

```html
<html>
    <head>
        <title>Users Management</title>
    </head>
    <body>
        <script type="text/javascript">
        window.app = {
            models: {},
            collections: {},
            templates: {},
            views: {},
            controllers: {},
            router: null
        };
        </script>

        <script type="text/javascript" src="models/user.js"></script>

        <script type="text/javascript" src="collections/users.js"></script>

        <script type="text/javascript" src="templates/usersTemplate.js"></script>

        <script type="text/javascript" src="views/UsersPage.js"></script>

        <script type="text/javascript" src="controllers/UsersPageController.js"></script>

        <script type="text/javascript" src="router.js"></script>
    </body>
</html>
```

Generated index.html

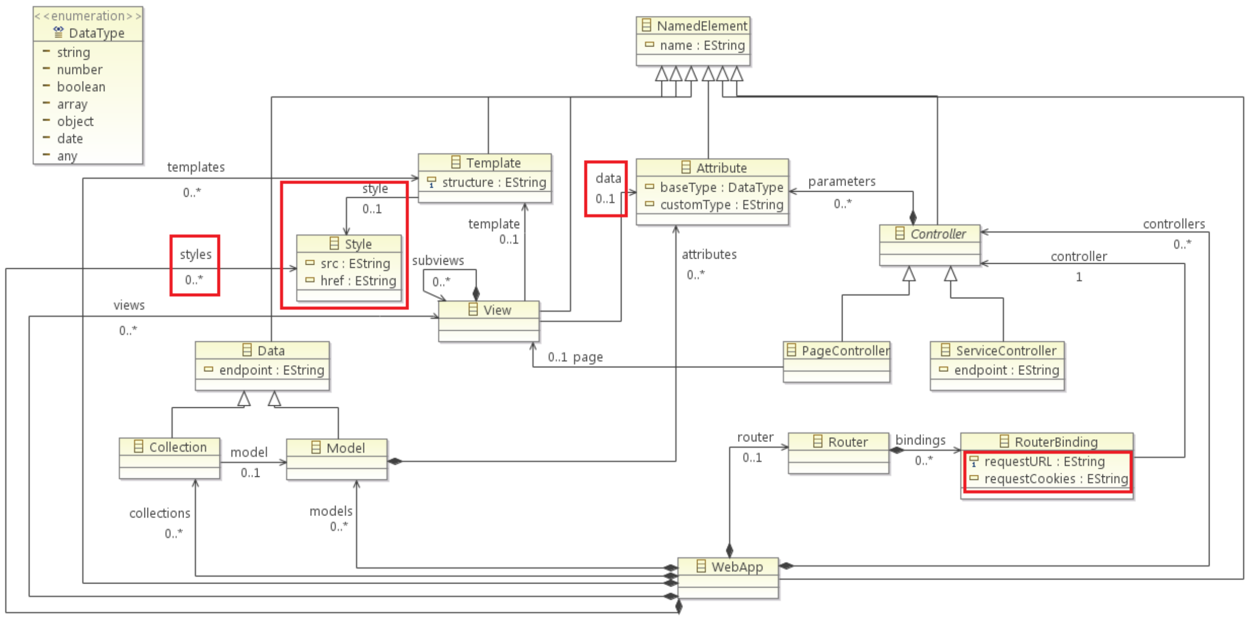


```
app.models.user = (function() {
  var user = {
    endpoint: '/api/users/[:id]',
    props: {
      'name': 'string',
      'surname': 'string',
      'age': 'number',
      'id': 'string',
    }
  };
  return user;
})();
```

Generated models/user.js

Evolution

Evolved metamodel



Evolved metamodel diagram

The following atomic changes to the metamodel have been performed:

- Changed View “data” EReference referenced class from Data to Attribute.

- Removed RouterBinding “url” EAttribute.
- Added RouterBinding “requestURL” and “requestCookies” features, both EAttribute of type EString.

Note: this means to replace the “url” EAttribute with the two EAttribute “requestURL” and “requestCookies”.

- Added “Style” EClass with “src” and “href” features, both EAtrribute of type EString, to represent an optionally external stylesheet.

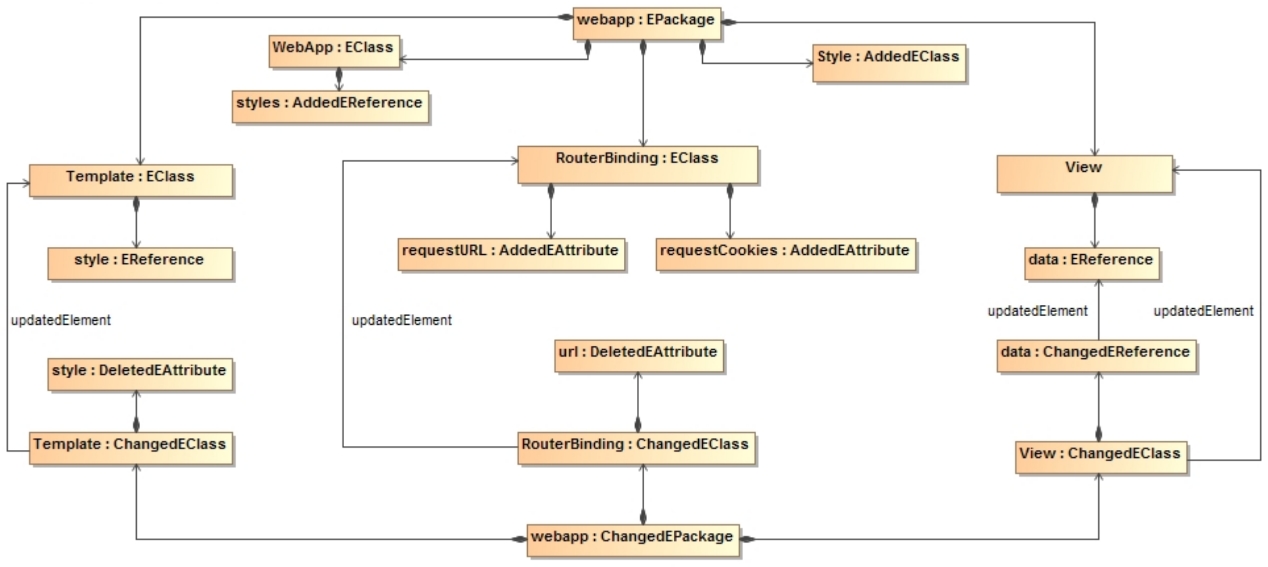
- Removed Template “style” EAttribute.
- Added Template “style” EReference with referenced class Style.

Note: this means to change the Template “style” feature kind: from EAttribute to EReference.

- Added WebApp “styles” EReference with type Style.

Difference model

The changes performed on the metamodel are clearly visible by executing a diff between the original and the evolved metamodel:



Difference model diagram

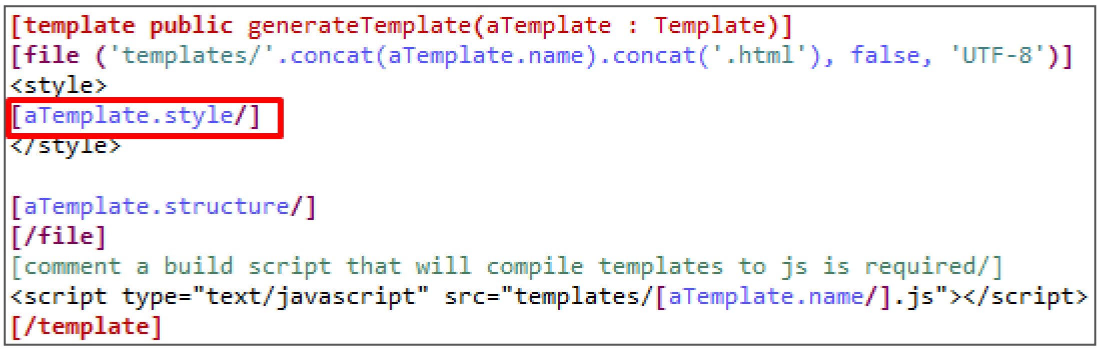
Note that the difference model will be crucial when adapting the broken Acceleo transformation, since it will help the system in detecting If a breaking change applies to the evolved metamodel.

Acceleo transformation Co-evolution

Breaking-not-resolvable (BNR) changes

Given the changes performed during the evolution of the metamodel and the related Acceleo transformation, the following changes that made the transformation unusable and that can't be automatically adapted without user intervention have been detected:

1. **Referenced EAttribute changed into EReference.**



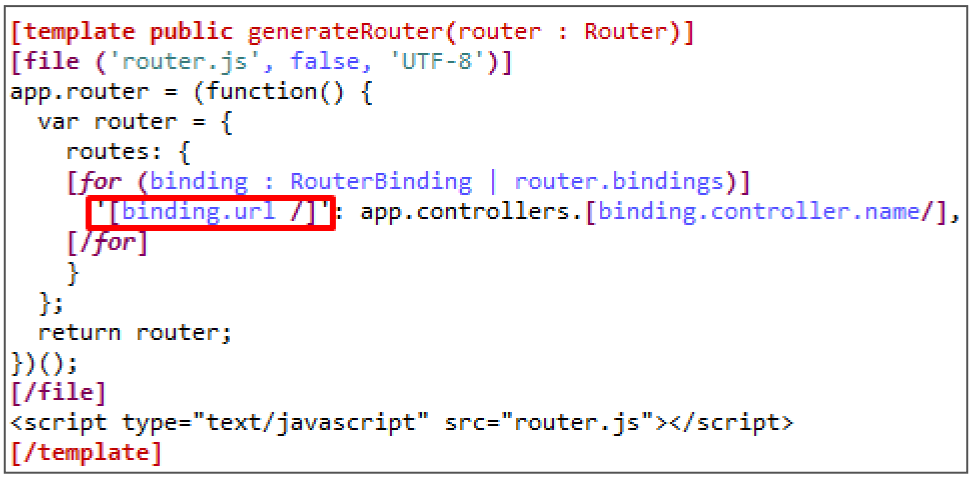
```
[template public generateTemplate(aTemplate : Template)]
[file ('templates/'.concat(aTemplate.name).concat('.html'), false, 'UTF-8')]
<style>
[aTemplate.style/]
</style>

[aTemplate.structure/]
[/file]
[comment a build script that will compile templates to js is required/]
<script type="text/javascript" src="templates/[aTemplate.name/].js"></script>
[/template]
```

Printed Template "style" attribute, deleted during evolution

Changing the feature kind to EReference breaks the transformation and requires the choosing of the referenced class attribute to use.

This is clearly not automatically resolvable, since the change is fully semantic and requires domain and vocabulary knowledge that can't be easily inferred, even less with absolute certainty, thus requiring user intervention.

2. **Referenced EAttribute replaced by other EAttribute features.**

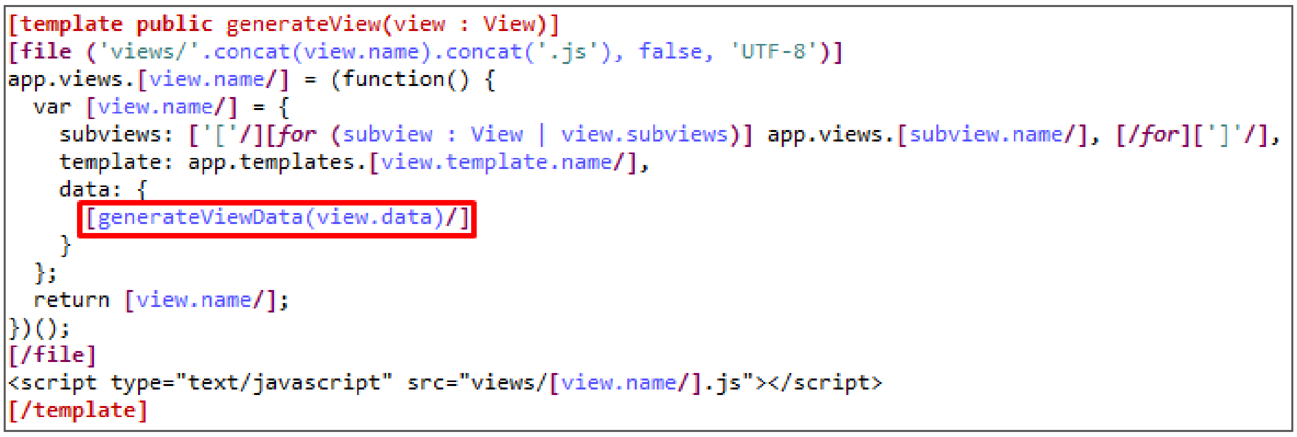


```
[template public generateRouter(router : Router)]
[file ('router.js', false, 'UTF-8')]
app.router = (function() {
  var router = {
    routes: {
    [for (binding : RouterBinding | router.bindings)]
      '[binding.url /]': app.controllers.[binding.controller.name/],
    [/for]
    }
  };
  return router;
})();
[/file]
<script type="text/javascript" src="router.js"></script>
[/template]
```

Printed RouterBinding "url" attribute, replaced during evolution

The referenced attribute has been removed, leaving the Acceleo transformation with nothing to reference, however, new attributes have been added, thus the user can fix the problem by choosing the new attribute to reference; also in this case an automated system can't know by itself what attribute to choose, making this breaking change unresolvable.

3. **EReference used for a template invocation changed referenced class, making the invocated template unusable**.

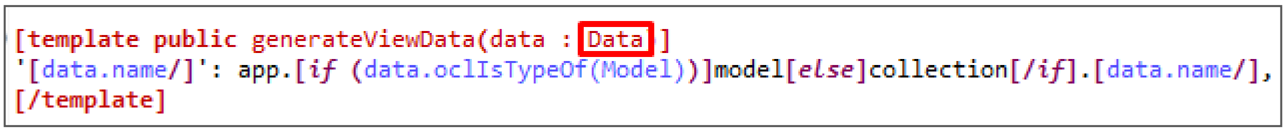


```
[template public generateView(view : View)]
[file ('views/'.concat(view.name).concat('.js'), false, 'UTF-8')]
app.views.[view.name/] = (function() {
  var [view.name/] = {
    subviews: ['['/][for (subview : View | view.subviews)] app.views.[subview.name/], [/for][']'/],
    template: app.templates.[view.template.name/],
    data: {
      [generateViewData(view.data)/]
    }
  };
  return [view.name/];
})();
[/file]
<script type="text/javascript" src="views/[view.name/].js"></script>
[/template]
```

View “data” EReference used for a template invocation

If an EReference is used as an argument in a template invocation, than it must be compatible with the template, i.e. having the same type of the template parameter or being one of its child types.

During the evolution the EReference changed the referenced EClass to a new one that is not compatible anymore with the called template.



```
[template public generateViewData(data : Data)]
'[data.name/]': app.[if (data.oclIsTypeOf(Model))]model[else]collection[/if].[data.name/],
[/template]
```

View “data” now refers “Attribute” which is not of kind “Data”

The issue can be solved by the user by choosing what template among the compatible ones to call.

ETL Acceleo BNR-adaptor

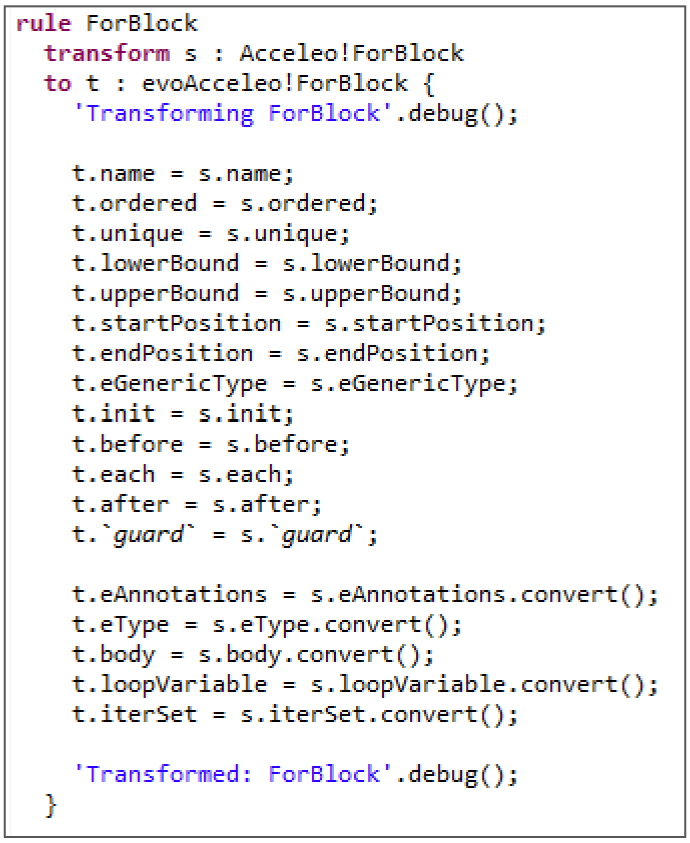
An ETL transformation has been created to adapt the broken Acceleo transformation, in fact, thanks to the ETL interactive features, dialogs can be created to ask for user intervention in fixing unresolvable changes.

The ETL transformation is logically divided into two main parts: a conservative copy section and a BNR-changes fixing section.

Conservative copy

The role of this section is simply to clone the source Acceleo model by copying the object properties in order to create the target Acceleo model, with the exception of the Metamodel objects that must be mapped with their already existing counterparts in the Evolved Metamodel.

An example of a conservative copy transformation rule is the following:

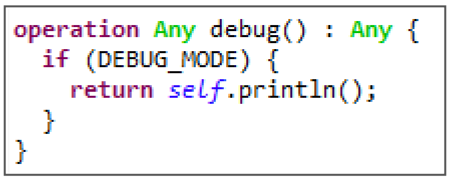


```
rule ForBlock
  transform s : Acceleo!ForBlock
  to t : evoAcceleo!ForBlock {
    'Transforming ForBlock'.debug();

    t.name = s.name;
    t.ordered = s.ordered;
    t.unique = s.unique;
    t.lowerBound = s.lowerBound;
    t.upperBound = s.upperBound;
    t.startPosition = s.startPosition;
    t.endPosition = s.endPosition;
    t.eGenericType = s.eGenericType;
    t.init = s.init;
    t.before = s.before;
    t.each = s.each;
    t.after = s.after;
    t.`guard` = s.`guard`;

    t.eAnnotations = s.eAnnotations.convert();
    t.eType = s.eType.convert();
    t.body = s.body.convert();
    t.loopVariable = s.loopVariable.convert();
    t.iterSet = s.iterSet.convert();

    'Transformed: ForBlock'.debug();
  }
```

A conservative copy transformation rule

What's interesting in this rule is the usage of the debug and convert operations.

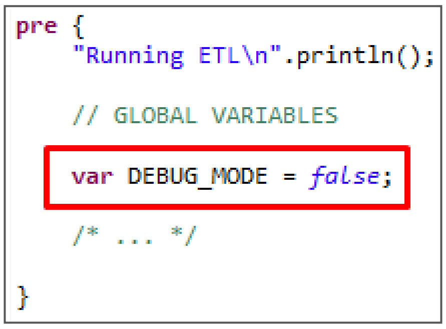
The debug operation



```
operation Any debug() : Any {
  if (DEBUG_MODE) {
    return self.println();
  }
}
```

The debug operation

The debug operation is a simple helper that prints the context it has been called onto only if the debug mode has been activated, i.e. if the DEBUG_MODE constant has been setted to true in the “pre” ETL block:



```
pre {
    "Running ETL\n".println();

    // GLOBAL VARIABLES

    var DEBUG_MODE = false;

    /* ... */

}
```

The DEBUG_MODE constant

That’s quite useful during development to log all the transformation steps, while hiding them in production usage.

The convert operation

The convert operation is a very important operation among the ones that have been created, since allows to retrieve the transformed target object given the source object in an intelligent way; in fact, often is not enough to just copy or clone the object as is, we need to reference the converted version, even by calling the apposite transformation rule if it has not been executed yet.

It extends the ETL built-in equivalent operation, by adding support for the following:

- If called on a Collection, recursively calls convert on each of its items.
- If no matching rule for the given context can be found, then instead that returning undefined like the equivalent operation, it returns the context itself.
  This is quite useful when working with transformations from and to the same metamodel, since it allows writing only the transformation rules that are really needed, automatically keeping the objects as they are if no changes are required.
- ETL has good support for polymorphism in operations, thus custom converts can be defined that are triggered only for specific contexts, like EClass, EAttribute and so on.
  When a custom convert matches the current context, it is immediately called instead that searching for matching transformation rules, allowing for a consistent way of mapping objects with other existing objects, without creating them anew.

```
// Custom equivalent function:
// is recursive and returns the element itself if there is no matching rule for it.
operation Any convert() {

  var equivalent;

  if (self.isKindOf(Collection)) {
    // collection -> recursion
    'Converting Collection '.concat(self.toString()).debug();
    equivalent = self->collect( element | element.convert() );
    'Converted: Collection '.concat(equivalent.toString()).debug();
  } else {
    // single element
    equivalent = self.equivalent();
    if (equivalent.isUndefined()) {
      // no matching rule(s)
      equivalent = self;
    }
  }

  return equivalent;
}
```

The convert operation

Follows a list of custom convert operations for the source metamodel objects.

Convert EPackage

The package evolved counterpart is found by simply iterating over all the evolved metamodel packages, selecting the one that has the same name.

```
operation Metamodel!ecore::EPackage convert() : evoMetamodel!ecore::EPackage {
  'Converting EPackage '.concat(self.name).debug();

  // get the relative package from the evolved metamodel
  var evoPackage = evoMetamodel!ecore::EPackage.allInstances()->selectOne(package | package.name == self.name);

  'Converted: EPackage '.concat(self.name).debug();

  return evoPackage;
}
```

The EPackage convert operation

Convert EClass

We assumes the non-existence of different packages containing metaclasses with the same names, thus we use the previous method: select the evolved class with the same name.

```
operation Metamodel!ecore::EClass convert() : evoMetamodel!ecore::EClass {
  'Converting EClass '.concat(self.name).debug();

  // get the relative class from the evolved metamodel
  var evoClass = evoMetamodel!ecore::EClass.allInstances()->selectOne(metaclass | metaclass.name == self.name);

  'Converted: EClass '.concat(self.name).debug();

  return evoClass;
}
```

The EClass convert operation

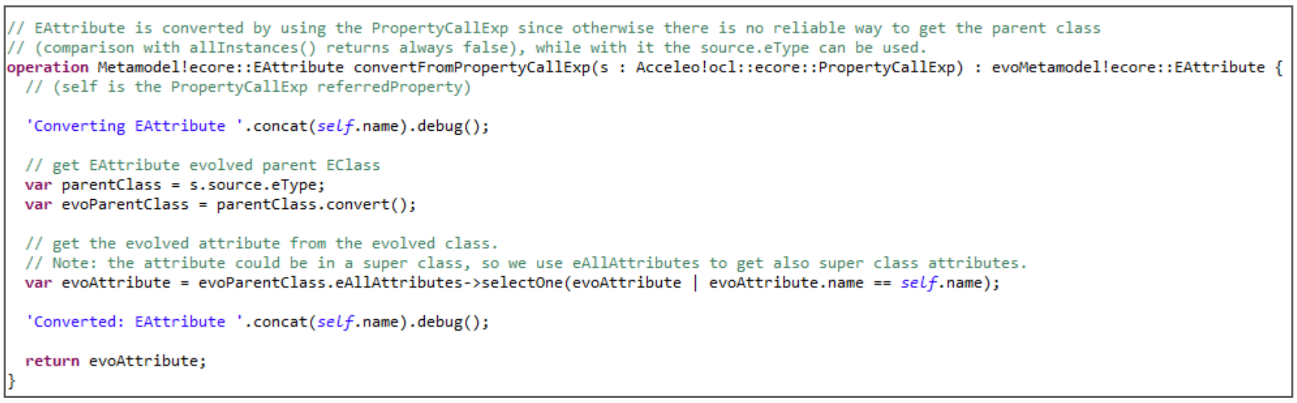
Convert EAttribute from PropertyCallExp

This convert operation is not a real override of the original convert, in fact it has a different name: convertFromPropertyCallExp.

The reason is that it requires an argument stating the OCL::ECORE::PropertyCallExp object in the source Acceleo transformation that contains the EAttribute parent EClass to convert.

In fact, differently than ATL, ETL does not seem to provide a way of getting the parent EClass of a structural feature, thus it must be extracted directly from the model(s).

The evolved attribute is the one with the same name in the evolved parent class.

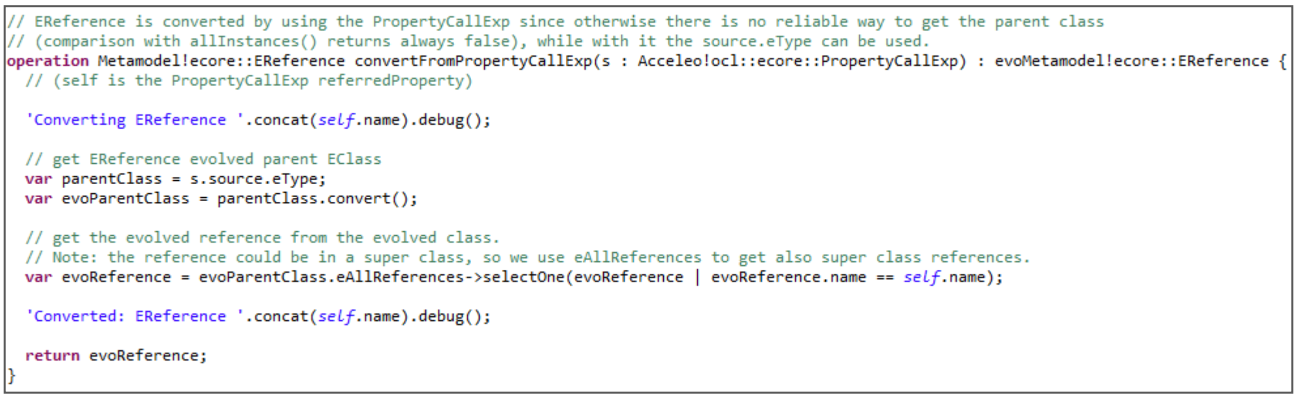


```
// EAttribute is converted by using the PropertyCallExp since otherwise there is no reliable way to get the parent class
// (comparison with allInstances() returns always false), while with it the source.eType can be used.
operation Metamodel!ecore::EAttribute convertFromPropertyCallExp(s : Acceleo!ocl::ecore::PropertyCallExp) : evoMetamodel!ecore::EAttribute {
  // (self is the PropertyCallExp referredProperty)

  'Converting EAttribute '.concat(self.name).debug();

  // get EAttribute evolved parent EClass
  var parentClass = s.source.eType;
  var evoParentClass = parentClass.convert();

  // get the evolved attribute from the evolved class.
  // Note: the attribute could be in a super class, so we use eAllAttributes to get also super class attributes.
  var evoAttribute = evoParentClass.eAllAttributes->selectOne(evoAttribute | evoAttribute.name == self.name);

  'Converted: EAttribute '.concat(self.name).debug();

  return evoAttribute;
}
```

The EAttribute convert operation

Convert EReference from PropertyCallExp

Identical to the previous, but for PropertyCallExp where the referenced property is an EReference instead that an EAttribute.

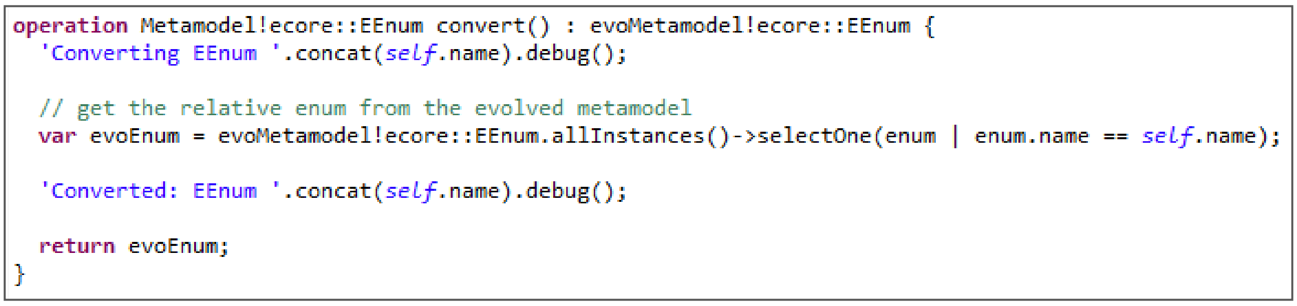


```
// EReference is converted by using the PropertyCallExp since otherwise there is no reliable way to get the parent class
// (comparison with allInstances() returns always false), while with it the source.eType can be used.
operation Metamodel!ecore::EReference convertFromPropertyCallExp(s : Acceleo!ocl::ecore::PropertyCallExp) : evoMetamodel!ecore::EReference {
  // (self is the PropertyCallExp referredProperty)

  'Converting EReference '.concat(self.name).debug();

  // get EReference evolved parent EClass
  var parentClass = s.source.eType;
  var evoParentClass = parentClass.convert();

  // get the evolved reference from the evolved class.
  // Note: the reference could be in a super class, so we use eAllReferences to get also super class references.
  var evoReference = evoParentClass.eAllReferences->selectOne(evoReference | evoReference.name == self.name);

  'Converted: EReference '.concat(self.name).debug();

  return evoReference;
}
```

The EReference convert operation

Convert EENum

Straightforward convert function similar to the ones for EClass and EPackage:



```
operation Metamodel!ecore::EEnum convert() : evoMetamodel!ecore::EEnum {
  'Converting EEnum '.concat(self.name).debug();

  // get the relative enum from the evolved metamodel
  var evoEnum = evoMetamodel!ecore::EEnum.allInstances()->selectOne(enum | enum.name == self.name);

  'Converted: EEnum '.concat(self.name).debug();

  return evoEnum;
}
```

The EENum convert operation

Fixing BNR changes

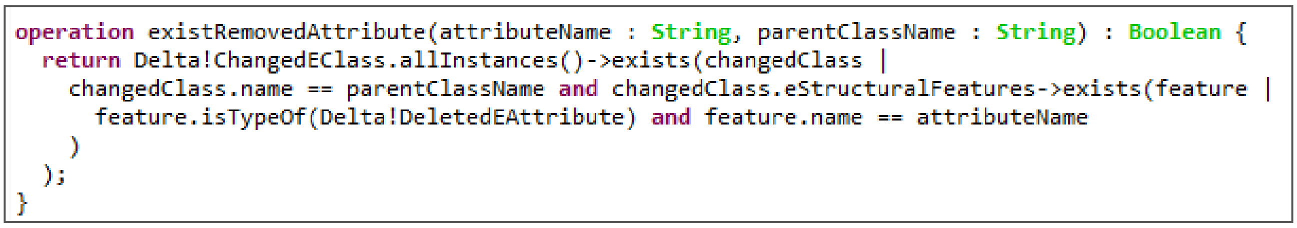
The fixing part of the transformation specifically targets only those objects that are affected by the BNR changes, specifically PropertyCallExp and TemplateInvocation objects.

Of course, not every object of those classes is actually broken, thus we need a way of detecting when this happens, by helping ourselves with the difference model previously created.

For this reason, some preparatory Delta helpers have been created, in order to have a convenient way of querying the difference model for interesting changes in the original metamodel.

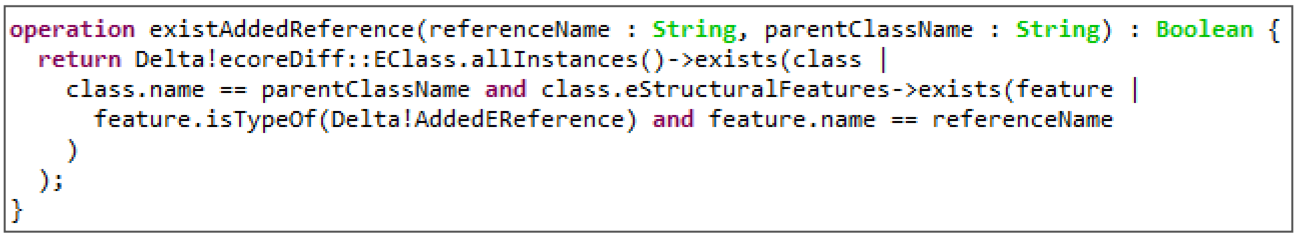
Delta utilities

All of this helpers simply use first-order-logic built-in operations to navigate the model; the reader can refer to the difference model diagram for better understanding.



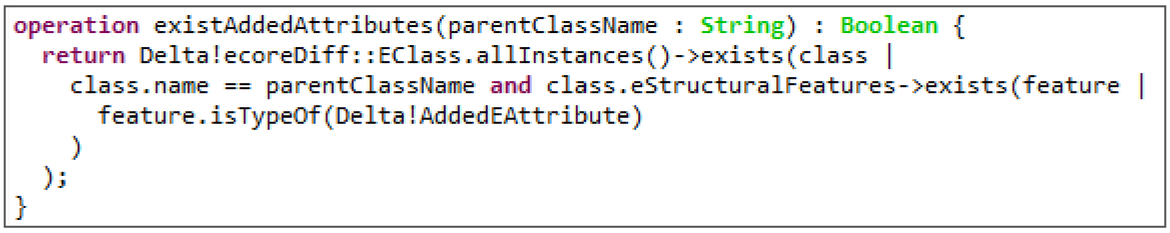
```
operation existRemovedAttribute(attributeName : String, parentClassName : String) : Boolean {
  return Delta!ChangedEClass.allInstances()->exists(changedClass |
    changedClass.name == parentClassName and changedClass.eStructuralFeatures->exists(feature |
      feature.isTypeOf(Delta!DeletedEAttribute) and feature.name == attributeName
    )
  );
}
```

Check for EAttribute removal (BNR1, BNR2)



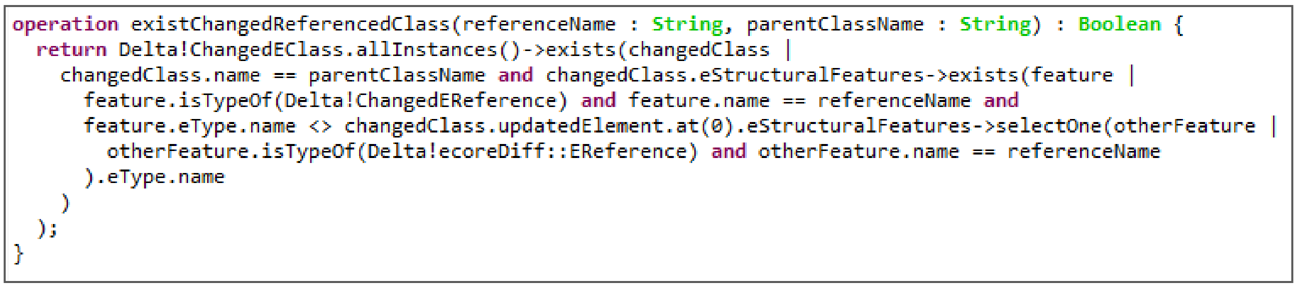
```
operation existAddedReference(referenceName : String, parentClassName : String) : Boolean {
  return Delta!ecoreDiff::EClass.allInstances()->exists(class |
    class.name == parentClassName and class.eStructuralFeatures->exists(feature |
      feature.isTypeOf(Delta!AddedEReference) and feature.name == referenceName
    )
  );
}
```

Check for EReference addition (BNR1)



```
operation existAddedAttributes(parentClassName : String) : Boolean {
  return Delta!ecoreDiff::EClass.allInstances()->exists(class |
    class.name == parentClassName and class.eStructuralFeatures->exists(feature |
      feature.isTypeOf(Delta!AddedEAttribute)
    )
  );
}
```

Check for addition of one or more EAttribute (BNR2)



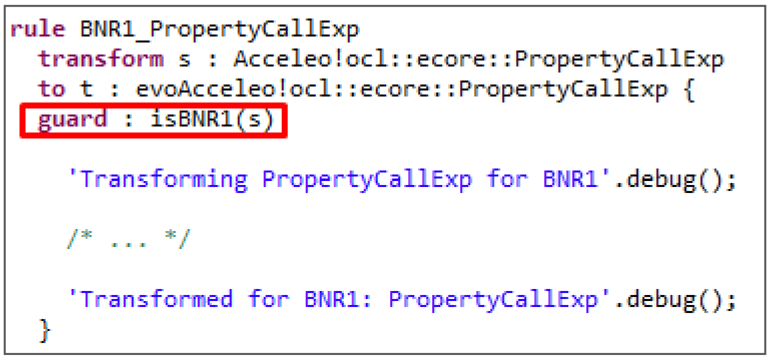
```
operation existChangedReferencedClass(referenceName : String, parentClassName : String) : Boolean {
  return Delta!ChangedEClass.allInstances()->exists(changedClass |
    changedClass.name == parentClassName and changedClass.eStructuralFeatures->exists(feature |
      feature.isTypeOf(Delta!ChangedEReference) and feature.name == referenceName and
      feature.eType.name <> changedClass.updatedElement.at(0).eStructuralFeatures->selectOne(otherFeature |
        otherFeature.isTypeOf(Delta!ecoreDiff::EReference) and otherFeature.name == referenceName
      ).eType.name
    )
  );
}
```

Check for EReference with changed referenced class (BNR3)

Fixing BNR1: “Referenced EAttribute changed into EReference”

The BNR-change fixing process is divided into two parts: a detection phase and a transformation phase.

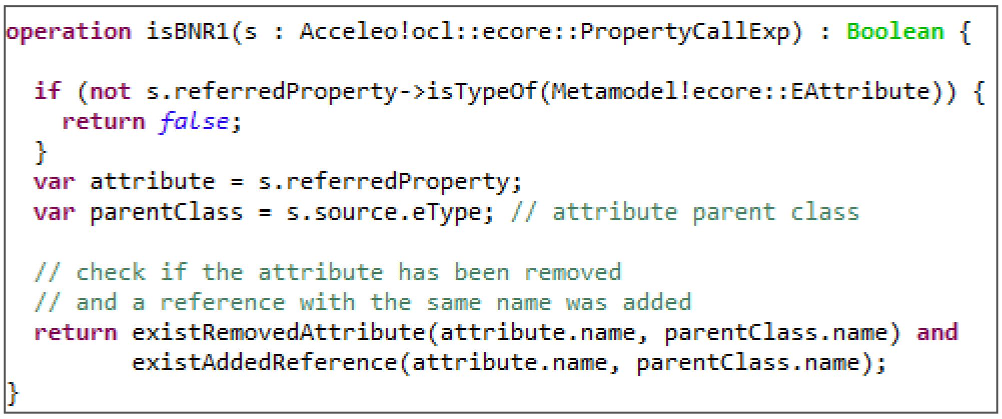
The detection phase uses the delta utilities previously shown alongside with other methods to check if an Acceleo object that could contain the change actually has it. The detection is triggered as a guard on the transformation rule(s) for the affected Acceleo object class(es).



```
rule BNR1_PropertyCallExp
  transform s : Acceleo!ocl::ecore::PropertyCallExp
  to t : evoAcceleo!ocl::ecore::PropertyCallExp {
  guard : isBNR1(s)

    'Transforming PropertyCallExp for BNR1'.debug();

    /* ... */

    'Transformed for BNR1: PropertyCallExp'.debug();
  }
```

BNR guard

For BNR1, a PropertyCallExp referencing an EAttribute must exists, moreover, the referenced attribute must have been removed from its parent class in the evolved metamodel, while a reference with the same name must have been added in it. Practically speaking, this means to replace the EAttribute with an EReference:

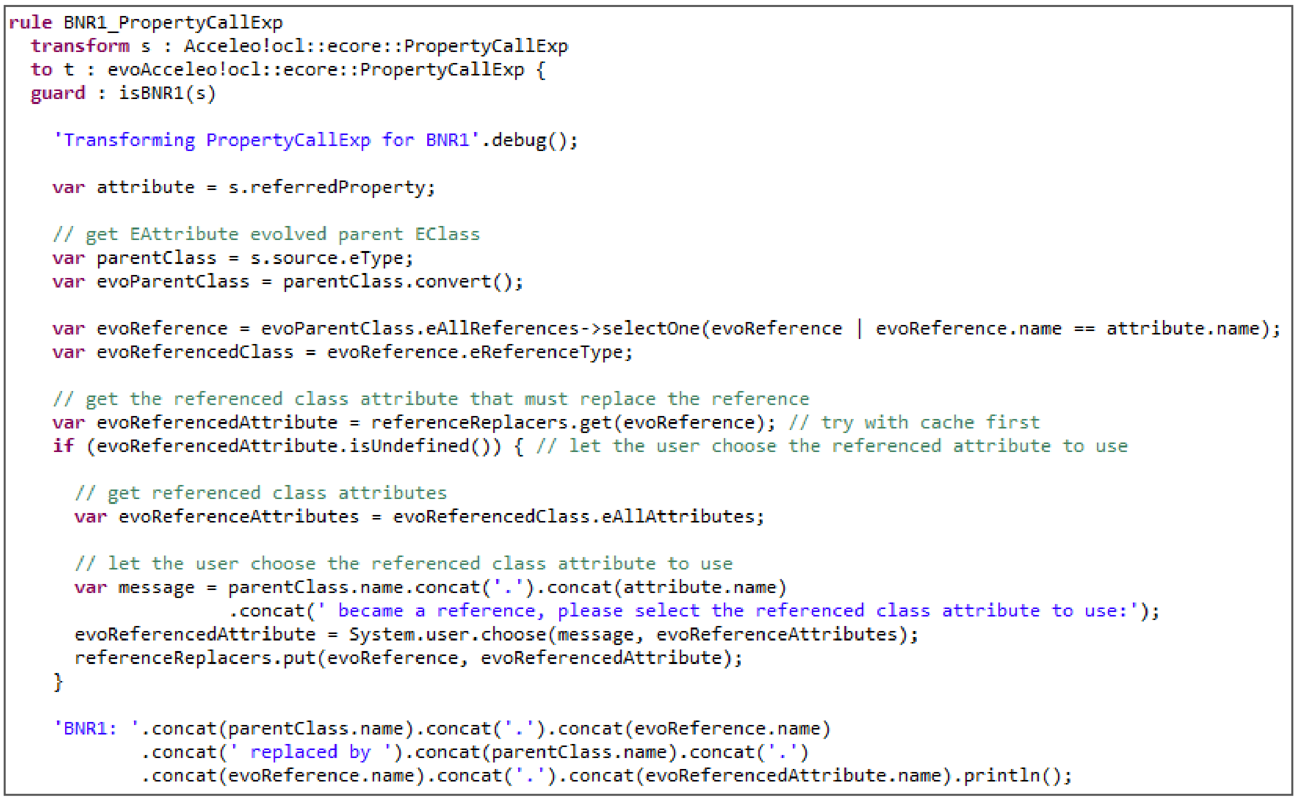


```
operation isBNR1(s : Acceleo!ocl::ecore::PropertyCallExp) : Boolean {

  if (not s.referredProperty->isTypeOf(Metamodel!ecore::EAttribute)) {
    return false;
  }
  var attribute = s.referredProperty;
  var parentClass = s.source.eType; // attribute parent class

  // check if the attribute has been removed
  // and a reference with the same name was added
  return existRemovedAttribute(attribute.name, parentClass.name) and
         existAddedReference(attribute.name, parentClass.name);
}
```

BNR1: detection phase

In the transformation phase, the interactive dialog is created, to ask the user for the required action to perform, based on the type of breaking change. In case of BNR1, the user must choose what attribute of the referenced class to use.

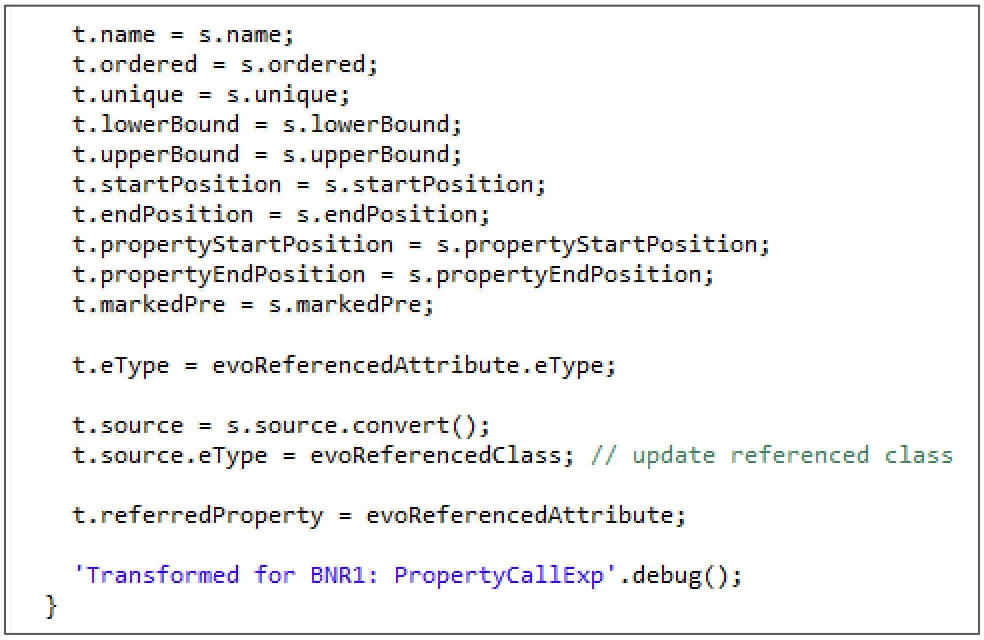
To make the transformation more comfortable, a cache is created to remember the user choice, in order to don’t ask the same question over and over again.



```
rule BNR1_PropertyCallExp
  transform s : Acceleo!ocl::ecore::PropertyCallExp
  to t : evoAcceleo!ocl::ecore::PropertyCallExp {
  guard : isBNR1(s)

    'Transforming PropertyCallExp for BNR1'.debug();

    var attribute = s.referredProperty;

    // get EAttribute evolved parent EClass
    var parentClass = s.source.eType;
    var evoParentClass = parentClass.convert();

    var evoReference = evoParentClass.eAllReferences->selectOne(evoReference | evoReference.name == attribute.name);
    var evoReferencedClass = evoReference.eReferenceType;

    // get the referenced class attribute that must replace the reference
    var evoReferencedAttribute = referenceReplacers.get(evoReference); // try with cache first
    if (evoReferencedAttribute.isUndefined()) { // let the user choose the referenced attribute to use

      // get referenced class attributes
      var evoReferenceAttributes = evoReferencedClass.eAllAttributes;

      // let the user choose the referenced class attribute to use
      var message = parentClass.name.concat('.').concat(attribute.name)
                    .concat(' became a reference, please select the referenced class attribute to use:');
      evoReferencedAttribute = System.user.choose(message, evoReferenceAttributes);
      referenceReplacers.put(evoReference, evoReferencedAttribute);
    }

    'BNR1: '.concat(parentClass.name).concat('.').concat(evoReference.name)
        .concat(' replaced by ').concat(parentClass.name).concat('.')
        .concat(evoReference.name).concat('.').concat(evoReferencedAttribute.name).println();
```

BNR1: asking for user intervention

Once the EAttribute to use has been chosen by the user, the evolved PropertyCallExp can be created:



```
    t.name = s.name;
    t.ordered = s.ordered;
    t.unique = s.unique;
    t.lowerBound = s.lowerBound;
    t.upperBound = s.upperBound;
    t.startPosition = s.startPosition;
    t.endPosition = s.endPosition;
    t.propertyStartPosition = s.propertyStartPosition;
    t.propertyEndPosition = s.propertyEndPosition;
    t.markedPre = s.markedPre;

    t.eType = evoReferencedAttribute.eType;

    t.source = s.source.convert();
    t.source.eType = evoReferencedClass; // update referenced class

    t.referredProperty = evoReferencedAttribute;

    'Transformed for BNR1: PropertyCallExp'.debug();
  }
```

BNR1: target creation

Fixing BNR2: “Referenced EAttribute replaced by other EAttribute features”

Similar to the previous, since also affects PropertyCallExp objects, happens when an attribute that is referenced is removed, while other attributes are added in the same parent class.

```
operation isBNR2(s : Acceleo!ocl::ecore::PropertyCallExp) : Boolean {

  if (not s.referredProperty->isTypeOf(Metamodel!ecore::EAttribute)) {
    return false;
  }
  var attribute = s.referredProperty;
  var parentClass = s.source.eType; // attribute parent class

  return existRemovedAttribute(attribute.name, parentClass.name) and
         existAddedAttributes(parentClass.name);
}
```

BNR2: detection phase

Here, the user must be asked for the added attribute to use:

```
rule BNR2_PropertyCallExp
  transform s : Acceleo!ocl::ecore::PropertyCallExp
  to t : evoAcceleo!ocl::ecore::PropertyCallExp {
  guard : isBNR2(s)

    'Transforming PropertyCallExp for BNR2'.debug();

    var attribute = s.referredProperty;

    // get EAttribute evolved parent EClass
    var parentClass = s.source.eType;
    var evoParentClass = parentClass.convert();

    // get the evolved attribute from the evolved class.
    // Note: the attribute could be in a super class, so we use eAllAttributes to get also super class attributes.
    var evoAttribute = evoParentClass.eAllAttributes->selectOne(evoAttribute | evoAttribute.name == attribute.name);

    var newEvoAttributes = evoParentClass.eAllAttributes->excluding(parentClass.eAllAttributes);

    var evoReplacerAttribute = deletedAttributeReplacers.get(attribute); // try with cache first
    if (evoReplacerAttribute.isUndefined()) {
      // let the user choose the attribute to use
      var message = parentClass.name.concat('.').concat(attribute.name)
                    .concat(' was replaced by new attributes, please select the one to use:');
      evoReplacerAttribute = System.user.choose(message, newEvoAttributes);
      deletedAttributeReplacers.put(attribute, evoReplacerAttribute);
    }

    'BNR2: '.concat(parentClass.name).concat('.').concat(attribute.name)
           .concat(' replaced by ').concat(parentClass.name).concat('.')
           .concat(evoReplacerAttribute.name).println();
```

BNR2: asking for user intervention

Finally, the chosen attribute can be setted as the referred property in the target PropertyCallExp:

```
   t.name = s.name;
   t.ordered = s.ordered;
   t.unique = s.unique;
   t.lowerBound = s.lowerBound;
   t.upperBound = s.upperBound;
   //t.many = s.many; // is just a Boolean but gives java.lang.NullPointerException
   //t.required = s.required; // as above
   t.startPosition = s.startPosition;
   t.endPosition = s.endPosition;
   t.propertyStartPosition = s.propertyStartPosition;
   t.propertyEndPosition = s.propertyEndPosition;
   t.markedPre = s.markedPre;

   t.eType = evoReplacerAttribute.eType; // new attribute type

   t.source = s.source.convert();

   t.referredProperty = evoReplacerAttribute;

   'Transformed for BNR2: PropertyCallExp'.debug();
}
```

BNR2: target creation

Fixing BNR3: “EReference used for a template invocation changed referenced class, making the invocated template unusable”

Differently from the previous breaking-not-resolvable changes, the last one affects the Acceleo TemplateInvocation class, in fact, the change happens when a template is called by using as argument a reference whose referenced class changed in the evolved metamodel, moreover, the new referenced class must be incompatible with the template:

```
operation isBNR3(s : Acceleo!TemplateInvocation) : Boolean {

  var argument = s.argument.at(0); // argument is actually an OrderedSet

  if (not argument->isTypeOf(Acceleo!ocl::ecore::PropertyCallExp)) {
    return false;
  }
  var propertyCallExp = argument;

  if (not propertyCallExp.referredProperty->isTypeOf(Metamodel!ecore::EReference)) {
    return false;
  }
  var reference = propertyCallExp.referredProperty;

  var parentClass = propertyCallExp.source.eType; // reference parent class

  if (existChangedReferencedClass(reference.name, parentClass.name)) { // the reference changed type
    // check if the current template is incompatible with the referenced class

    var evoReference = reference.convertFromPropertyCallExp(propertyCallExp);
    var evoReferencedClass = evoReference.eReferenceType;
    var template = s.definition;
    if (not template.isCompatibleWith(evoReferencedClass)) {
      return true;
    }
  }

  return false;
}
```

BNR3: detection phase

A template is compatible with an EClass if it is of the same kind of the template class, i.e. it is the same class or one if its subclasses. An apposite operation has been created to check for that compatibility:

```
// a template for a class A is compatible with a class B if B is either A or a subclass of A
operation Acceleo!Template isCompatibleWith(class : Metamodel!ecore::EClass) : Boolean {
  var templateClass = self.parameter.eType.at(0); // Template parameter eType is a Sequence
  return class.name == templateClass.name or class.eSuperTypes->includes(templateClass);
}
```

The template “isCompatibleWith” operation

In the transformation phase, the same helper is used to retrieve the collection of templates that are compatible with the new referenced class, letting the user choose the one to use:

```
rule BNR3_TemplateInvocation
  transform s : Acceleo!TemplateInvocation
  to t : evoAcceleo!TemplateInvocation {
  guard : isBNR3(s)

    'Transforming TemplateInvocation for BNR3'.debug();

    var propertyCallExp = s.argument.at(0);
    var template = s.definition;
    var reference = propertyCallExp.referredProperty;

    var evoReference = reference.convertFromPropertyCallExp(propertyCallExp);
    var evoReferencedClass = evoReference.eReferenceType;
    var parentClass = propertyCallExp.source.eType; // reference parent class

    var replacerTemplate = templateReplacers.get(reference); // try with cache first
    if (replacerTemplate.isUndefined()) {
      // let the user choose the compatible template to use

      var compatibleTemplates = Acceleo!Template.allInstances()->select(otherTemplate |
        otherTemplate.isCompatibleWith(evoReferencedClass)
      );

      var message = template.name.concat(' is not compatible anymore with ')
                       .concat(parentClass.name).concat('.').concat(reference.name)
                      .concat(', please select the template to use:');
      replacerTemplate = System.user.choose(message, compatibleTemplates);
      templateReplacers.put(reference, replacerTemplate);
    }

    'BNR3: '.concat(template.name).concat('(').concat(parentClass.name)
            .concat('.').concat(reference.name).concat(')')
            .concat(' replaced by ').concat(replacerTemplate.name).concat('(')
            .concat(parentClass.name).concat('.').concat(reference.name).concat(')').println();
```

BNR3: asking for user intervention

As always, once the required information has been collected, the target TemplateInvocation object can be created:

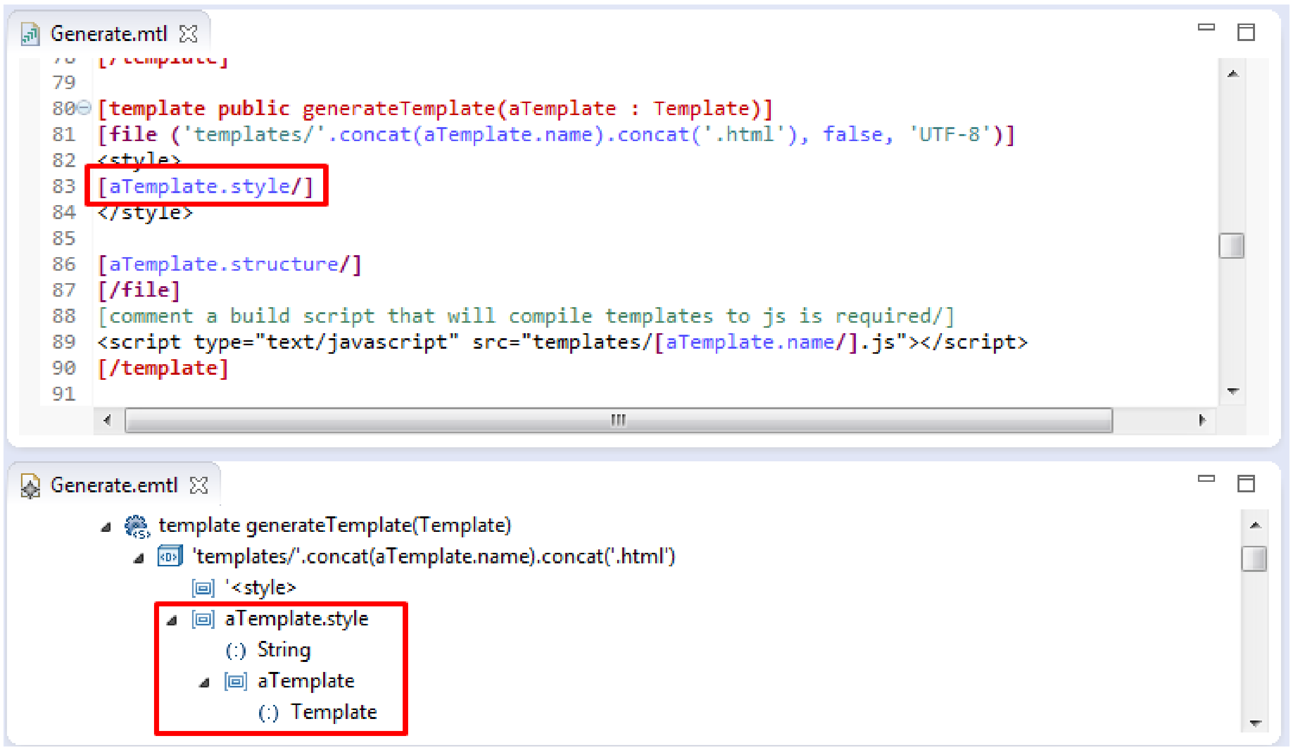
```
  t.name = s.name;
  t.ordered = s.ordered;
  t.unique = s.unique;
  t.lowerBound = s.lowerBound;
  t.upperBound = s.upperBound;
  t.startPosition = s.startPosition;
  t.endPosition = s.endPosition;
  t.super = s.super;

  t.argument = s.argument.convert();
  t.argument.at(0).eType = evoReferencedClass; // the reference changed type => it must be updated

  t.definition = replacerTemplate.convert();

  t.eType = s.eType.convert();

  'Transformed for BNR3: TemplateInvocation'.debug();
}
```

BNR3: target creation

Co-evolved Acceleo transformation

In this section, the result of executing the transformation is shown, comparing the original Acceleo module with its automatically evolved and fixed version.

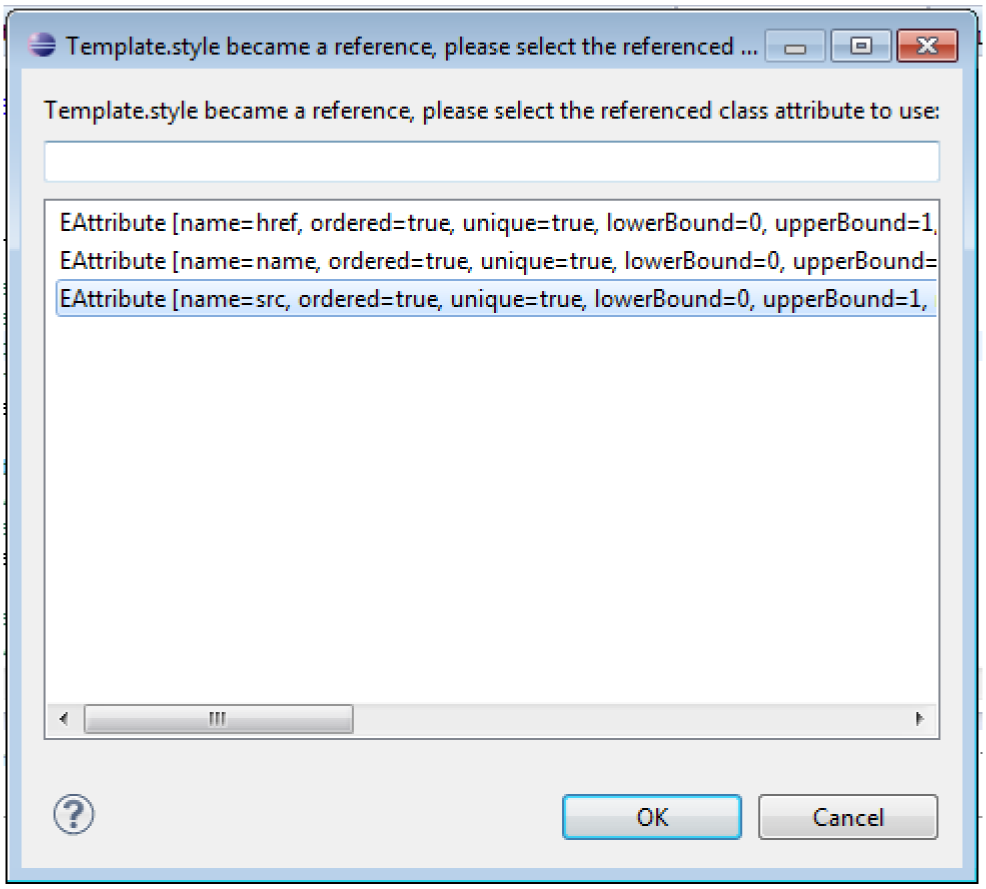
- For BNR1, the referenced Template "style" attribute in the generateTemplate template must be changed into the Style "src" attribute:



```
[/template]

[template public generateTemplate(aTemplate : Template)]
[file ('templates/'.concat(aTemplate.name).concat('.html'), false, 'UTF-8')]
<style>
[aTemplate.style/]
</style>

[aTemplate.structure/]
[/file]
[comment a build script that will compile templates to js is required/]
<script type="text/javascript" src="templates/[aTemplate.name/].js"></script>
[/template]
```

```
template generateTemplate(Template)
  'templates/'.concat(aTemplate.name).concat('.html')
    '<style>
    aTemplate.style
      String
      aTemplate
        Template
```

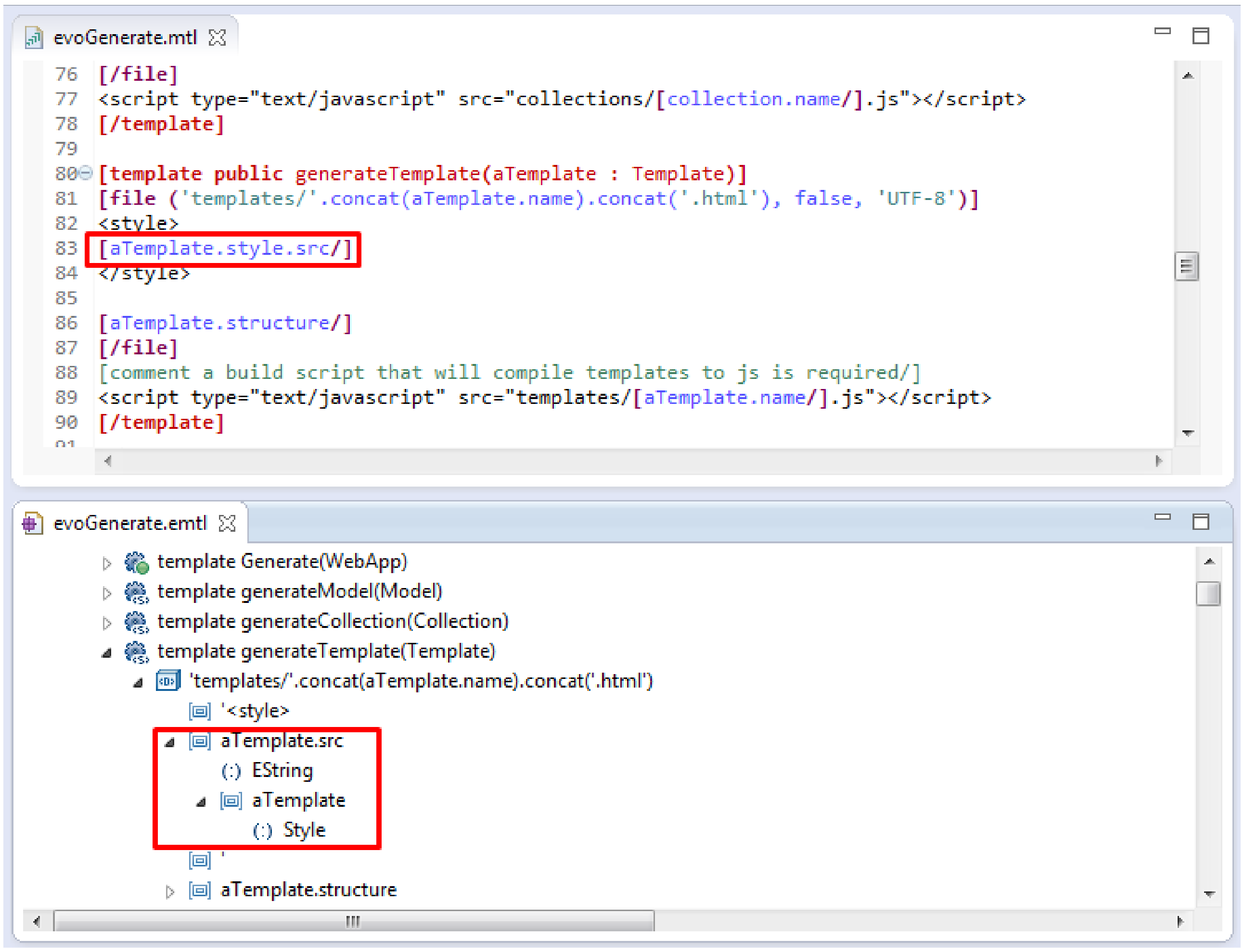
BNR1: before


Template.style became a reference, please select the referenced ...

Template.style became a reference, please select the referenced class attribute to use:

```
EAttribute [name=href, ordered=true, unique=true, lowerBound=0, upperBound=1,
EAttribute [name=name, ordered=true, unique=true, lowerBound=0, upperBound=
EAttribute [name=src, ordered=true, unique=true, lowerBound=0, upperBound=1,
```

BNR1: asking for user intervention


evoGenerate.mtl

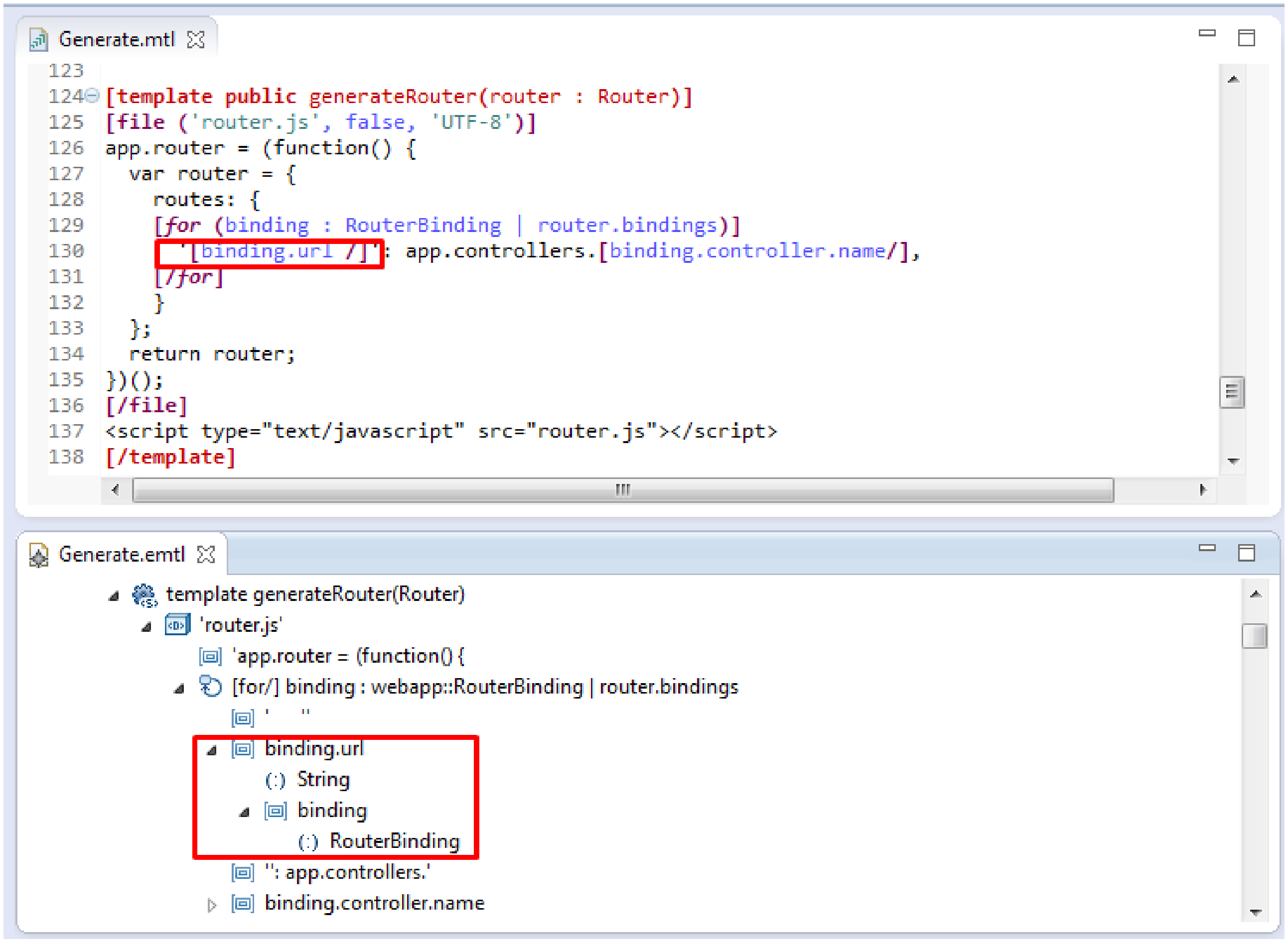
```
[/file]
<script type="text/javascript" src="collections/[collection.name/].js"></script>
[/template]

[template public generateTemplate(aTemplate : Template)]
[file ('templates/'.concat(aTemplate.name).concat('.html'), false, 'UTF-8')]
<style>
[aTemplate.style.src/]
</style>

[aTemplate.structure/]
[/file]
[comment a build script that will compile templates to js is required/]
<script type="text/javascript" src="templates/[aTemplate.name/].js"></script>
[/template]
```

evoGenerate.emtl

```
template Generate(WebApp)
template generateModel(Model)
template generateCollection(Collection)
template generateTemplate(Template)
  'templates/'.concat(aTemplate.name).concat('.html')
    '<style>
    aTemplate.src
      (:) EString
      aTemplate
        (:) Style
    '
    aTemplate.structure
```

BNR1: after

- For BNR2, the referenced RouterBinding “url” attribute in the generateRouter template must be changed into the RouterBinding “requestURL” attribute:



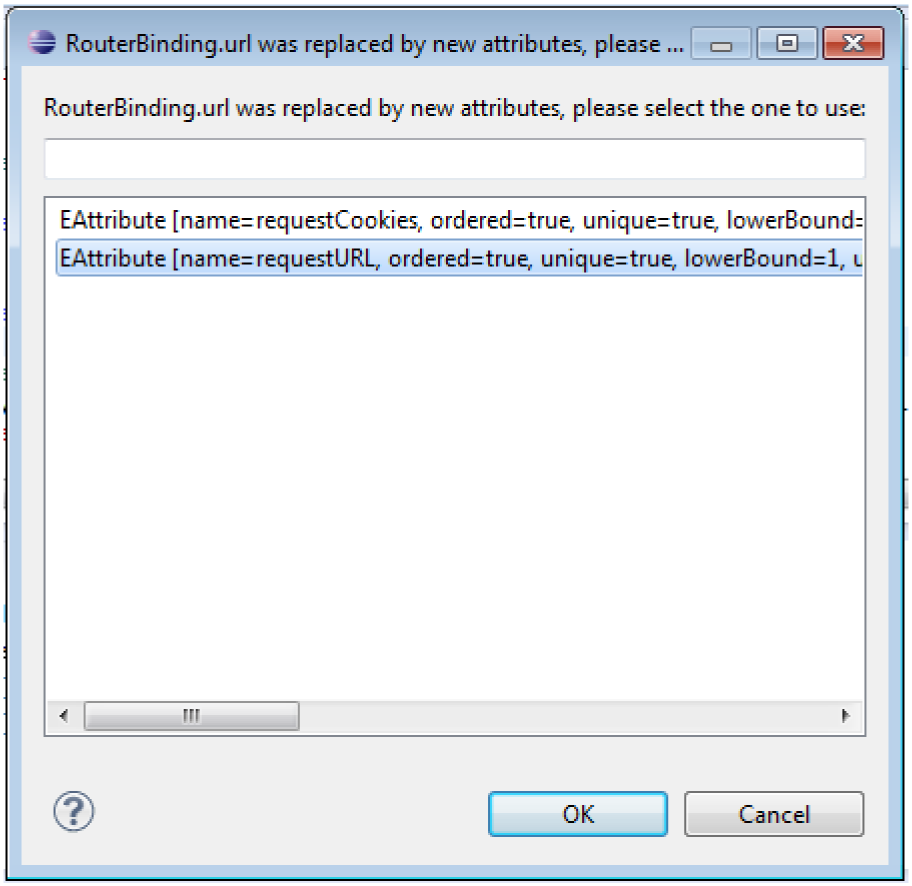
Generate.mtl

```
[template public generateRouter(router : Router)]
[file ('router.js', false, 'UTF-8')]
app.router = (function() {
  var router = {
    routes: {
    [for (binding : RouterBinding | router.bindings)]
      '[binding.url /]': app.controllers.[binding.controller.name/],
    [/for]
    }
  };
  return router;
})();
[/file]
<script type="text/javascript" src="router.js"></script>
[/template]
```

Generate.emtl

```
template generateRouter(Router)
  'router.js'
    'app.router = (function() {
    [for/] binding : webapp::RouterBinding | router.bindings
      '    ''
      binding.url
        (:) String
        binding
          (:) RouterBinding
      '': app.controllers.'
      binding.controller.name
```

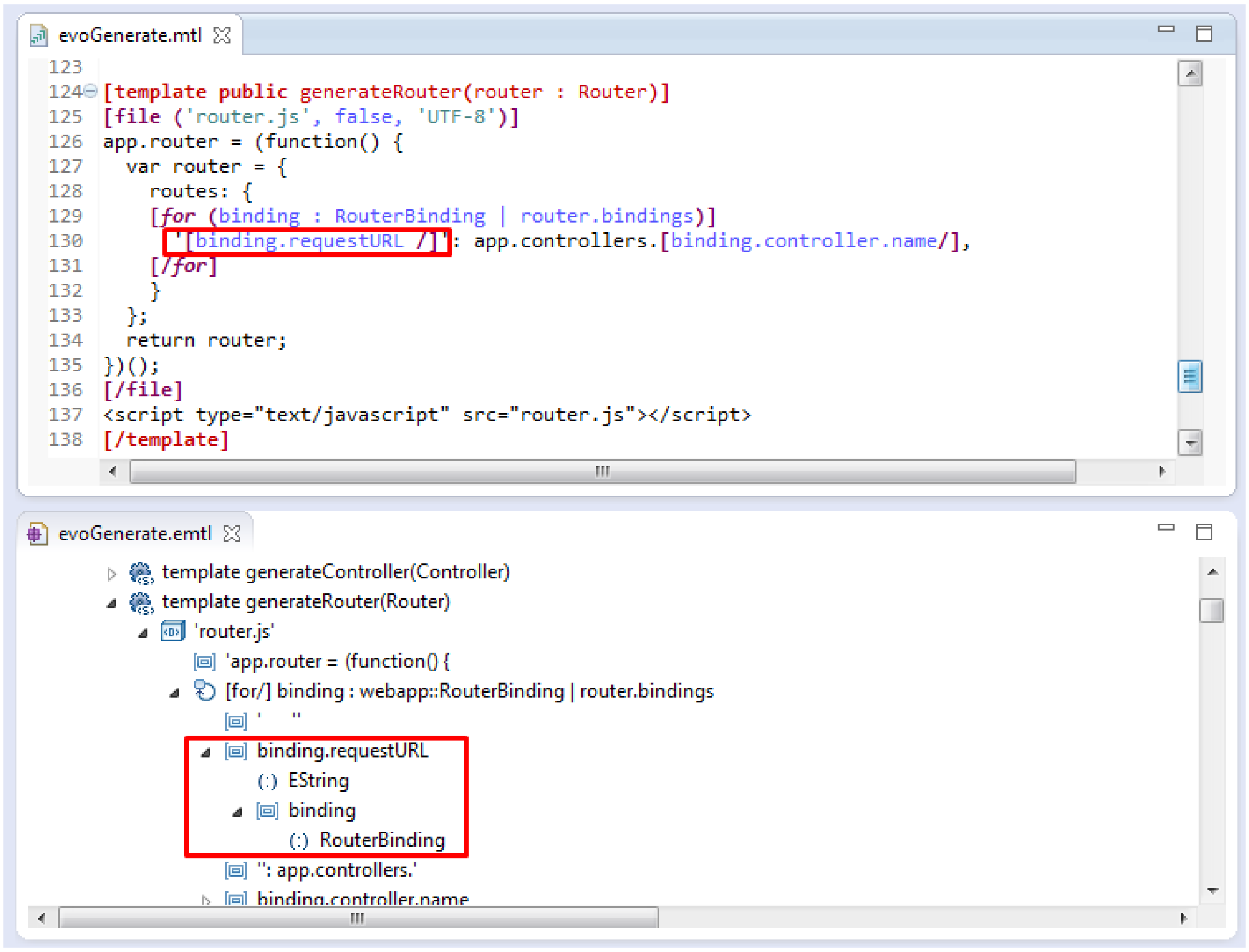
BNR2: before


RouterBinding.url was replaced by new attributes, please ...

RouterBinding.url was replaced by new attributes, please select the one to use:

```
EAttribute [name=requestCookies, ordered=true, unique=true, lowerBound=
EAttribute [name=requestURL, ordered=true, unique=true, lowerBound=1, u
```

BNR2: asking for user intervention

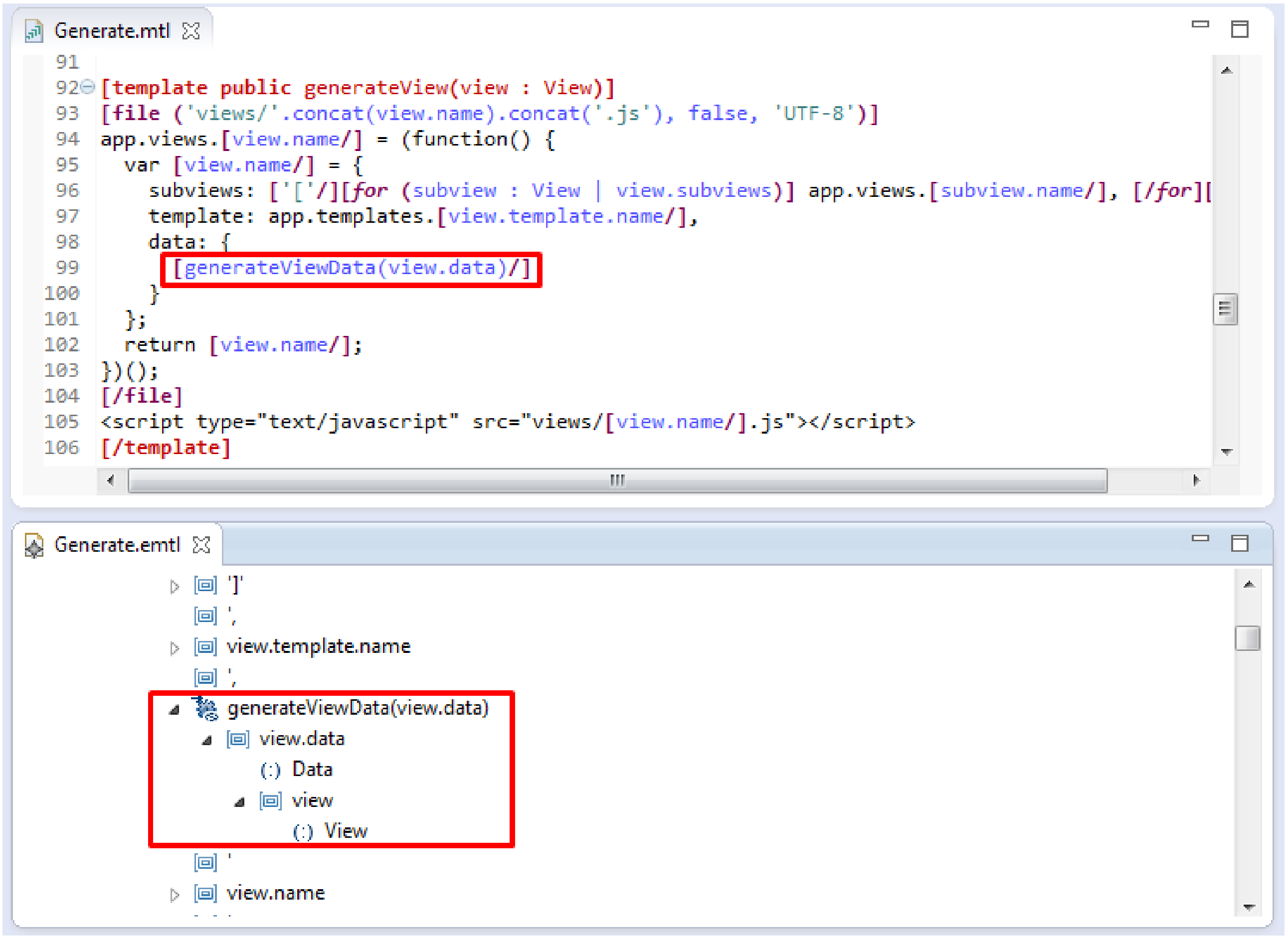


```
[template public generateRouter(router : Router)]
[file ('router.js', false, 'UTF-8')]
app.router = (function() {
  var router = {
    routes: {
    [for (binding : RouterBinding | router.bindings)]
      '[binding.requestURL /]': app.controllers.[binding.controller.name/],
    [/for]
    }
  };
  return router;
})();
[/file]
<script type="text/javascript" src="router.js"></script>
[/template]
```

```
template generateController(Controller)
template generateRouter(Router)
  'router.js'
    'app.router = (function() {
    [for/] binding : webapp::RouterBinding | router.bindings
      '    '
      binding.requestURL
        EString
        binding
          RouterBinding
      '': app.controllers.'
      binding.controller.name
```

BNR2: after

- For BNR3, the generateViewData template called with the View “data” attribute as argument in the generateView template, must be changed into the generateAttribute template, which is compatible with the evolved View “data” reference:



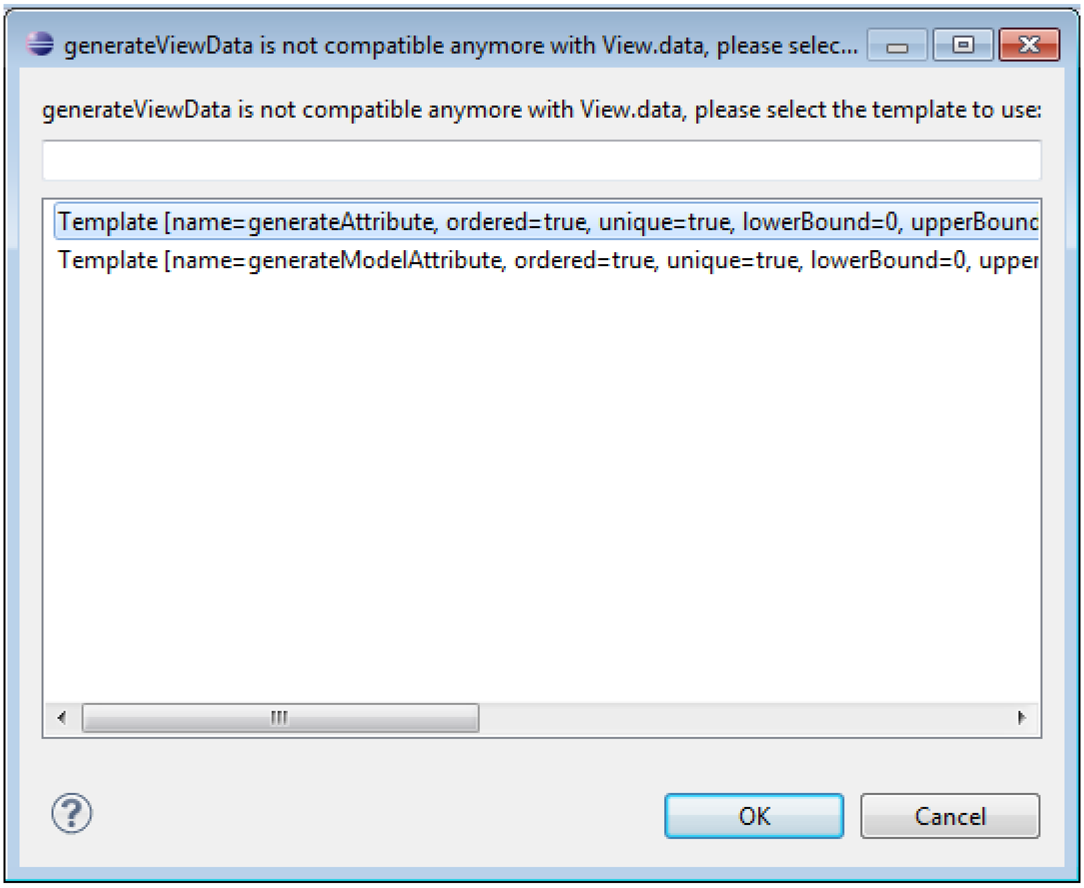
Generate.mtl

```
[template public generateView(view : View)]
[file ('views/'.concat(view.name).concat('.js'), false, 'UTF-8')]
app.views.[view.name/] = (function() {
  var [view.name/] = {
    subviews: ['['/][for (subview : View | view.subviews)] app.views.[subview.name/], [/for][
    template: app.templates.[view.template.name/],
    data: {
      [generateViewData(view.data)/]
    }
  };
  return [view.name/];
})();
[/file]
<script type="text/javascript" src="views/[view.name/].js"></script>
[/template]
```

Generate.emtl

```
']'
','
view.template.name
','
generateViewData(view.data)
  view.data
    Data
    view
      View
'
view.name
```

BNR3: before

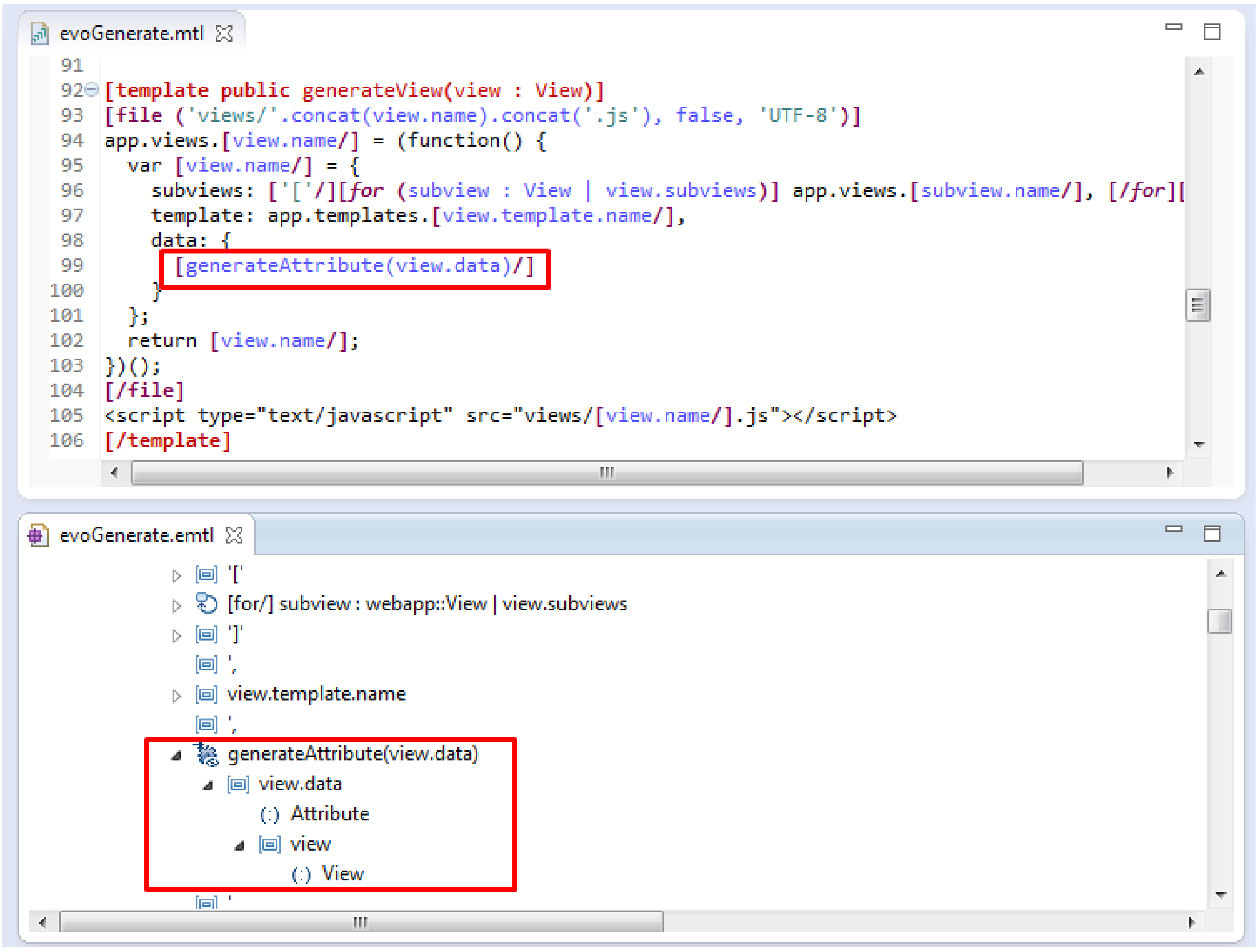


generateViewData is not compatible anymore with View.data, please selec...

generateViewData is not compatible anymore with View.data, please select the template to use:

```
Template [name=generateAttribute, ordered=true, unique=true, lowerBound=0, upperBounc
Template [name=generateModelAttribute, ordered=true, unique=true, lowerBound=0, upper
```

BNR3: asking for user intervention

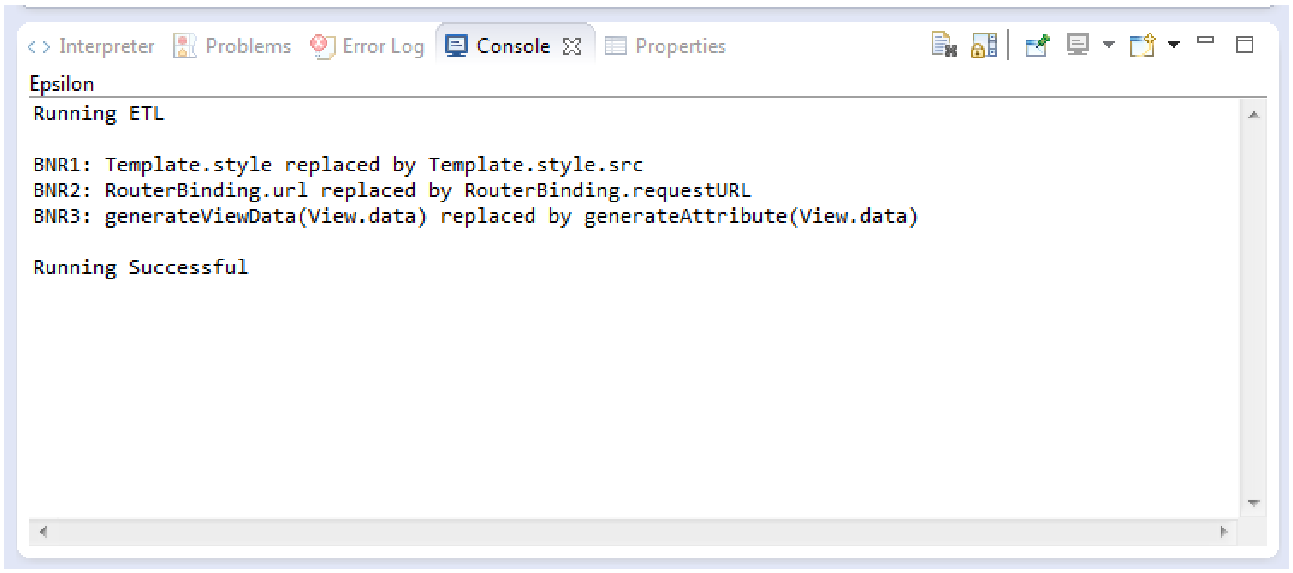


```
[template public generateView(view : View)]
[file ('views/'.concat(view.name).concat('.js'), false, 'UTF-8')]
app.views.[view.name/] = (function() {
  var [view.name/] = {
    subviews: ['['/][for (subview : View | view.subviews)] app.views.[subview.name/], [/for][
    template: app.templates.[view.template.name/],
    data: {
      [generateAttribute(view.data)/]
    }
  };
  return [view.name/];
})();
[/file]
<script type="text/javascript" src="views/[view.name/].js"></script>
[/template]
```

```
'['
[for/] subview : webapp::View | view.subviews
']'
','
view.template.name
','
generateAttribute(view.data)
  view.data
    Attribute
    view
      View
'
```

BNR3: after

Moreover, the status of the transformation is logged into the console:



```
Running ETL

BNR1: Template.style replaced by Template.style.src
BNR2: RouterBinding.url replaced by RouterBinding.requestURL
BNR3: generateViewData(View.data) replaced by generateAttribute(View.data)

Running Successful
```

Possible improvements

The following improvements should be studied to check for their feasibility in automatically adapting Acceleo transformations:

- **Support for objects affected by multiple breaking-not-resolvable changes.**
  After having fixed an object for a BNR change, the adaptor should recursively check if the resulting object is still affected by other BNR changes and fix them.
  In order to do this, the target class in the BNR-transformation-rules could be setted to be the same of the source class, so to add new objects in the source model, keeping sure to not store them on adaptation end (uncheck "Store on disposal" option on ETL run configuration), and thus calling the existing transformation rules on the new created source objects. When no other BNR-change is detected, the conservative copy transformation rule would be called, finally copying the transformed and fully fixed object into the adapted Acceleo model.
  Note: this could be tricky, since the source Acceleo model and the transformation rules / converts expects references to be versus the original metamodel, not to the evolved metamodel, however such issues can probably be ignored, thanks to the equivalent operation that keeps objects as they are if no convert or transformation rule applies and to the fact that metamodel-type checks are actually performed only when compiling the final Acceleo transformation, not while editing an emtl model.

- **Providing default user actions for fixing breaking-not-resolvable changes.**
  Suppose the user has to fix the BNR2 change, where it has to choose the new attribute to reference, but actually none of the added attributes really represent the old one, e.g. if it was just deleted; or the BNR3 change, if there are no compatible templates at all, generally speaking , all the situations where the standard fixing choices do not apply.
  The adaptor should provide a default action that should be feasible for all the BNR changes, for example, by removing the broken object.
  Unfortunately, that's not a straightforward problem, since the object could be part of a larger external context, thus removing it could lead to broken transformations.
  The challenge here is as such to find the minimum-size dependent context, which could be transformation-semantic dependent, and removing it, either by not calling the transformation rule at all, or by generating as output a StringLiteralExp with an empty string.