

# Automatic Generation of Ecore Models for Testing ATL Transformations

Jesús M. Almendros-Jiménez<sup>(✉)</sup> and Antonio Becerra-Terón

Department of Informatics, University of Almería, 04120 Almería, Spain  
{jalmen, abecerra}@ual.es

**Abstract.** Model transformation testing is crucial to detect incorrect transformations. Buggy transformations can lead to incorrect target models, either violating target meta-model requirements or more complex target model properties. In this paper we present a tool for testing ATL transformations. This tool is an extension of a previously developed tool for testing XML-based languages. With this aim an Ecore to XML Schema transformation is defined which makes to automatically generate random Ecore models possible. These randomly generated Ecore models are used to test ATL transformations. Properties to be tested are specified by OCL constraints, describing input and output conditions on source and target models, respectively.

## 1 Introduction

*Model transformation* is a key component of *Model Driven Engineering (MDE)* [19]. Several transformation languages (*ATL*, *QVT*, *AGG*, *VIATRA*, *Fujaba*, among others) have been proposed to define transformations making the definition of (*M2M*) *Model to Model* and (*M2T*) *Model to Text* transformations possible. Transformation languages work on source and target meta-models establishing a mapping from source to target models. Transformation programs are based on rules and range from imperative to declarative software.

*Testing* [17] is essential for ensuring software quality. The automation of testing enables the programmer to reduce time of testing and also makes to repeat testing after each modification to a program possible. A testing tool should determine whether a test is passed or failed. When failed, the testing tool should provide evidences of failures, that is, counterexamples of the properties to be checked. Additionally, a testing tool should generate test cases automatically [2]. Fully random generation of tests could not be suitable for an effective and efficient tool. Distribution of test data should be controlled, by providing user-defined test cases, that is, data distribution should be put under the human tester's control.

Testing of model transformations has been studied in several recent works [3–9, 11, 12, 14, 16, 18]. The quoted works have basically the same goal: specification of properties on transformations and meta-models in order to ensure correct

---

This work was supported by the EU (FEDER) and the Spanish MINECO Ministry (*Ministerio de Economía y Competitividad*) under grant TIN2013-44742-C4-4-R.

transformations. Properties on transformations range from termination, determinism, rule independence, rule applicability and reachability of states, while properties on meta-models establish input (respectively, output) properties on source (respectively, target) meta-models, as well as input-output properties on both source and target models. Here we will focus on meta-model properties, and our goal will be to build a testing tool able to detect buggy transformations from automatically generated test models.

In our research group we have developed a tool [1] to test XML-based applications. In particular, this tool has been previously used to test XQuery programs from automatically generated XML data. In this paper, we adapt the tool to model based transformation languages. In particular, we are able to automatically generate *Ecore* models, which are the selected format of ATL (ATLAS transformation language) [13] programs for source/target models. Additionally, we have extended the tool to test ATL applications, in such a way that given a source meta-model  $\mathcal{S}$ , an ATL transformation  $\mathcal{TR}$ , a set of input properties  $\mathcal{IP}$  on the source meta-model and a set of output properties  $\mathcal{OP}$  on the target meta-model, the tool is able to determine whether the output models of  $\mathcal{TR}$  satisfy  $\mathcal{OP}$  for each randomly generated test model of the meta-model  $\mathcal{S}$  satisfying  $\mathcal{IP}$ . In case of success, that is, each output model of  $\mathcal{TR}$  satisfies the properties  $\mathcal{OP}$ , then the tool answers “Ok”, otherwise the tool shows counterexamples, that is, input test models which do not satisfy  $\mathcal{OP}$ , together with the result of the transformation for the counterexamples.

Test models are randomly generated from source meta-models in an automatic way. The input of the testing is an Ecore source meta-model which is automatically transformed into an XML Schema. The resulting XML Schema is used by the tool to generate XML data tests. However, the human tester has to select the XML elements and attributes, and the number and value of them, for which test cases are randomly generated. In terms of Ecore models, the human tester has to prune the meta-model (similarly to [15]), selecting classes, attributes and associations to which generate instances. The idea is to select the smallest subset of the input meta-model that is relevant for the transformation (i.e., the classes, attributes and associations accessed by the transformation code). Thus, even when the generation of test models is fully random, the human tester can control the size and diversity of test models. The choice depends on the transformation to be tested as well as the input/output properties to be checked. Test models are randomly generated as combinations of values and a number of classes, attributes and associations, enabling a high level of diversity in test cases. Since the number of test cases can be potentially infinite, the human tester can select a limit of the size, and moreover, the test cases are generated in increasing size. In fact, most of ATL programming bugs can be detected from a small set of test models in a short time. The tool has been designed to interrupt testing when the output property is not satisfied by a test model, and thus even when the number of test models selected by the human tester can be bigger, the tool stops when a counterexample has been found.

The tool (i.e., Ecore to XML Schema transformation, test models generation, and property-based testing) has been implemented in the XQuery language. We use the *BaseX* XQuery interpreter to test ATL transformations. We have integrated the ATL EMFTVM virtual machine in XQuery, using the Java binding mechanism, in order to execute ATL transformations from XQuery and to test input/output properties.

The structure of the paper is as follows. Section 2 will define the transformation of Ecore meta-models to XML Schemas and will show examples of randomly generated test models. Section 3 will report several examples of ATL transformation testing. Section 4 will describe related work. Finally, Sect. 5 will present conclusions and future work.

## 2 Automatic Generation of Ecore Models

The process of automatic generation of Ecore test models is as follows. Firstly, the Ecore meta-model is transformed into an XML Schema. Next, the human tester selects items (elements and attributes) and number of items of the XML Schema (setting values for *minOccurs*/*maxOccurs* of elements, and values optional/required in *use* of attributes), and provides values for each type of the XML Schema (adding values to the *enumeration* section of each type). Finally, the test case generator reports random combinations of values and items.

With this aim, an Ecore to XML Schema transformation has been defined. Basically, *EClass* elements are mapped to *XML elements*, and *EAttribute* elements are mapped to *XML attributes*. Moreover, *EReference* elements are mapped to *XML elements* and *XML Attributes*. In the case of *EReference* elements with *containment* set to *true* the are mapped to *XML elements*, otherwise to *XML attributes*. In the second case, *XML Attributes* of XML Schemas have been modified enabling *minOccurs* and *maxOccurs* attributes. A similar solution was previously adopted in other works<sup>1</sup>. Thus XML Schemas have been extended to cover with Ecore references. Additionally, Ecore *lowerBound* and *upperBound* have been mapped to XML Schema *minOccurs* and *maxOccurs*, respectively. Finally, Ecore *datatypes* are mapped to XML *datatypes*.

For example, the *class* meta-model of the well-known *Class2Relational* transformation<sup>2</sup> of the *ATL Zoo*, is translated into the XML Schema of Fig. 1.

There are two types of XML attributes. The first one is the standard XML attribute: `<xs:attribute name="name" type="nameType" use="required"/>` and the second one is a reference to an element: `<xs:attribute name="super" type="Class" class="yes" minOccurs="1" maxOccurs="unbounded"/>` in which *class* = “*yes*” means that *type* is a reference to an XML element (in the example, the XML element *Class*), and *minOccurs* and *maxOccurs* represent the number of allowed references. The test case generator will generate values `//@Class.1`, `//@Class.2`, etc., of references to elements. For instance, this is an example of test model generated by the tool:

<sup>1</sup> <https://www.eclipse.org/modeling/emf/docs/overviews/XMLSchemaToEcoreMapping.pdf>.

<sup>2</sup> <http://www.eclipse.org/atl/atlTransformations/#Class2Relational>.

```

<xs:complexType name="attr">
  <xs:attribute name="name" type="nameType" use="required"/>
  <xs:attribute name="type" type="Classifier" class="yes"
    minOccurs="1" maxOccurs="1"/>
  <xs:attribute name="multiValued" type="multiValuedType" use="required"/>
</xs:complexType>
<xs:element name="xmi:XMI">
  <xs:complexType><xs:sequence>
    <xs:element name="Classifier" minOccurs="1" maxOccurs="unbounded">
      <xs:complexType>
        <xs:attribute name="name" type="nameType" use="required"/>
      </xs:complexType></xs:element>
    <xs:element name="DataType" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
        <xs:attribute name="name" type="nameType" use="required"/>
      </xs:complexType></xs:element>
    <xs:element name="Class" minOccurs="1" maxOccurs="unbounded">
      <xs:complexType>
        <xs:attribute name="name" type="nameType" use="required"/>
        <xs:attribute name="super" type="Class" class="yes"
          minOccurs="1" maxOccurs="unbounded"/>
        <xs:attribute name="isAbstract" type="isAbstractType" use="required"/>
        <xs:sequence>
          <xs:element ref="attr" minOccurs="1" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType></xs:element>
    </xs:sequence></xs:complexType>
  </xs:element>

```

Fig. 1. XML Schema of class meta-model

```

<xmi:XMI>
  <Classifier name="Default"/>
  <Class super="//@Class.1" isAbstract="true" name="Default"/>
  <Class isAbstract="true" name="Default">
    <attr type="//@Classifier.0" multiValued="true" name="Default"/>
  </Class>
</xmi:XMI>

```

The test case generator of [1] has been modified to include the second kind of attributes, and values of references to elements. The tool generates Ecore models of increasing size. Basically, starting from an initial step (step 0) with minimal models according to *minOccurs* values for elements and *optional/required* for attributes, it generates new models in each step (step  $n + 1$ ) adding new attributes and elements to models of step  $n$ , up to *maxOccurs* for elements and *required* for attributes is reached. In other words, the test case generator adds new classes, attributes and associations in each step, increasing the size of Ecore models. Additionally, it randomly takes values for attributes from the *enumeration* section. These values are manually added by the human tester, and the test case generator randomly combines values to produce different models. In the case a certain type is not defined in the *enumeration* section, the test case generator assigns as values “Default” for strings, “0” for numbers and “true” for Boolean. The number of steps  $n$  is a parameter of the tester, in order to limit the size of test models. The human tester can play with this parameter generating a large number of models. Also, the human tester can play with *minOccurs* (and *maxOccurs*) values for XML elements/XML Attributes, in order to generate a large number of classes and associations for each model. Playing with steps and

```

module Book2Publication;
create OUT : Publication from IN : Book;

rule Book2Publication1 {
  from
    b : Book!Book (b.getSumPages() > 100 and b.keyword='Biology')
  to
    out : Publication!Publication (
      title <- b.title,
      authors <- b.getAuthors(),
      nbPages <- b.getSumPages()
      keyword <- b.keyword)
}
rule Book2Publication2 {
  from
    b : Book!Book (b.getSumPages() < 100 and b.keyword='Romance')
  to
    out : Publication!Publication (
      title <- b.title,
      authors <- b.getAuthors(),
      nbPages <- b.getSumPages(),
      keyword <- b.keyword)
}

```

**Fig. 2.** *Book2Publication* transformation

*minOccurs*, the human tester can have a stronger confidence about the soundness of the program.

### 3 Testing of ATL Transformations

Let us consider the following (buggy) transformation *Book2Publication* defined by the code of Fig. 2. The transformation tries to map *book* into *publication* classes with two rules for *Biology* and *Romance* books, respectively. *Book* class is defined as a set of *chapters* each one with a *title*, an *author* and a number of pages *nbPages*. The transformation summarizes *books* as a *publication* in which *title* is the same, *authors* of *chapters* are concatenated and the total number of pages is computed. ATL helpers *getAuthors* and *getSumPages* have been defined with this end (omitted in the Figure). However, the mapping is only required for a total number of pages greater than 100.

The human tester can now define OCL constraints for the transformation (see Fig. 3). The OCL constraints on the source meta-model describe the required properties on the source model. In this case, source model constraints require that all the *books* and *chapters* have a title, and all the *books* have a *keyword*. The target meta-model OCL constraints describe the required properties on the target model. They require that the *title* of *publications* is not empty, all the *publications* are *Biology* or *Romance* *publications*, and finally, the number of pages *nbPages* is greater than 100. The Ecore source meta-model and the corresponding XML Schema are shown in Figs. 4 and 5, respectively.

The human tester can now edit the XML Schema in order to select relevant elements (XML elements and attributes) and values for the transformation and testing. In the *Book2Publication* transformation, the elements *book* and *chapters*

- (1) `Book!Book->allInstances()->select(b | b.title->size()->isEmpty())`
- (2) `Book!Chapter->allInstances()->select(ch | ch.title->size()->isEmpty())`
- (3) `Book!Book->allInstances()->select(b | b.keyword->size()->isEmpty())`
- (4) `Publication!Publication->allInstances()->select(p | p.title->size()->isEmpty())`
- (5) `Publication!Publication->allInstances()->select(p | not(p.keyword='Biology' or p.keyword='Romance'))->isEmpty()`
- (6) `Publication!Publication->allInstances()->select(p | not(p.nbPages>100))->isEmpty()`

**Fig. 3.** Input and output properties of *Book2Publication* transformation

```
<eClassifiers xsi:type="Ecore:EClass" name="Book">
<eStructuralFeatures
  xsi:type="Ecore:EAttribute" name="title" lowerBound="1" eType="EString"/>
<eStructuralFeatures
  xsi:type="Ecore:EAttribute" name="keyword" eType="EString"/>
<eStructuralFeatures
  xsi:type="Ecore:EReference" name="chapters" upperBound="-1"
  eType="##/Chapter" containment="true" eOpposite="##/Chapter/book"/>
</eClassifiers>
<eClassifiers xsi:type="Ecore:EClass" name="Chapter">
<eStructuralFeatures
  xsi:type="Ecore:EAttribute" name="title" lowerBound="1" eType="EString"/>
<eStructuralFeatures
  xsi:type="Ecore:EAttribute" name="nbPages" lowerBound="1" eType="EInt"/>
<eStructuralFeatures
  xsi:type="Ecore:EAttribute" name="author" lowerBound="1" eType="EString"/>
<eStructuralFeatures
  xsi:type="Ecore:EReference" name="book" lowerBound="1"
  eType="##/Book" eOpposite="##/Book/chapters"/>
</eClassifiers>
```

**Fig. 4.** Source meta-model of *Book2Publication* transformation

are relevant, and the same can be said for attributes *title*, *author*, *keyword* and *nbPages*. In order to force the generation of *books* with at least one *chapter* the human tester can set *minOccurs* to 1:

```
<x:element ref="chapters" minOccurs="1" maxOccurs="unbounded"/>
```

Next, the human tester selects relevant values for the transformation. In this case, the idea is to generate test models in which books have as *keyword* “Biology” and “Romance”, and some other value (for instance “Computers”) in order to validate (3) and (5) of the OCL constraints. Additionally, it would be useful to have chapters with different number of pages, greater than 100, and smaller than 100, in order to validate (6). A good choice would be to generate test models with chapters of size 50, 100, 150, etc. Thus the values for *nbPages* will be selected to be “50” and “150”. With regard to *title*, and the validation of (1), (2) and (4), it is only required to have at least one value, for instance “a”. *Author* element is not required by the OCL constraints, but required by the transformation, thus we can add just one value “b”. Values for attributes and elements are added to the XML Schema in the *enumeration* section as shown in Fig. 6. The tester call is as follows:

```

<xs:complexType name="chapters">
  <xs:attribute name="title" type="titleType" use="required"/>
  <xs:attribute name="nbPages" type="nbPagesType" use="required"/>
  <xs:attribute name="author" type="authorType" use="required"/>
</xs:complexType>
<xs:element name="xmi:XMI">
  <xs:complexType><xs:sequence>
    <xs:element name="Book" minOccurs="1" maxOccurs="unbounded">
      <xs:complexType><xs:sequence>
        <xs:element ref="chapters" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="title" type="titleType" use="required"/>
      <xs:attribute name="keyword" type="keywordType" use="required"/>
    </xs:complexType></xs:element>
  </xs:sequence></xs:complexType>
</xs:element>

```

**Fig. 5.** Source XML Schema of *Book2Publication* transformation

```

<xs:simpleType name="titleType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="a"/>
  </xs:restriction></xs:simpleType>
<xs:simpleType name="keywordType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Biology"/>
    <xs:enumeration value="Romance"/>
    <xs:enumeration value="Computers"/>
  </xs:restriction></xs:simpleType>
<xs:simpleType name="nbPagesType">
  <xs:restriction base="xs:integer">
    <xs:enumeration value="50"/>
    <xs:enumeration value="150"/>
  </xs:restriction></xs:simpleType>
<xs:simpleType name="authorType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="b"/>
  </xs:restriction>
</xs:simpleType>

```

**Fig. 6.** Values for source model of *Book2Publication* transformation

```

atl:tester("Schema.xsd","Book.Ecore","Book","Publication.Ecore",
  "Publication","MyPath","Book2Publication","Input_prop","Output_prop",1)

```

where the number “1” is the number of steps, and *Input\_prop* and *Output\_prop* are selected to be (3) and (6), respectively, of Fig. 3. The ATL tester reports in 639 ms the following answer:

```

Output Property Falsifiable after 2 tests.
Counterexample:

```

```

<xmi:XMI>
  <Book keyword="Romance" title="a">
    <chapters author="b" nbPages="50" title="a"/>
  </Book>
</xmi:XMI>

```

**Result:**

```

<publication:Publication title="a" authors="b" nbPages="50"
  keyword="Romance"/>

```

which means that after two test models, the ATL tester found that the output property cannot be satisfied. The tester shows a counterexample, that is, a test model fulfilling the input properties, but violating the output property. In addition, the tester shows the result of the transformation for the test model. In the counterexample, we can see that given as source model an *Romance* book with one chapter of 50 pages, the target model does not satisfy the output property. This is due to the (intentionally added) *bug* in the transformation: *Romance* books are transformed when the number of pages is smaller than 100. Once the bug is removed, the ATL tester answers (in 8,722 ms) as follows: **Ok: passed 54 tests, 54 valid**. Which means that 54 models have been proven, and all of them satisfy the output property, and in addition, it reports the number of models that satisfy the input property (i.e., valid models). When the number of models satisfying the input property is zero, the ATL tester answers: **Unable to test the property**.

```

module ER2RL;
create OUT : REL from IN : ER;

rule S2S { from s : ER!ERSchema
           to t : REL!RELSchema (name<-s.name,relations <- s.entities,
                                relations <- s.relships)}
rule E2R { from s : ER!Entity
           to t : REL!Relation ( name <- s.name) }

rule R2R { from s : ER!Relship
           to t : REL!Relation ( name <- s.name ) }

rule EA2A { from att : ER!ERAttribute, ent : ER!Entity (att.entity = ent)
            to t : REL!RELAttribute (name <- att.name, isKey <- att.isKey,
                                     relation <- ent) }

rule RA2A { from att : ER!ERAttribute, rs : ER!Relship (att.relship = rs)
            to t : REL!RELAttribute (name <- att.name, isKey <- att.isKey,
                                     relation <- rs ) }

rule RA2AK { from att : ER!ERAttribute, rse : ER!RelshipEnd
              (att.entity = rse.entity and att.isKey = true)
            to t : REL!RELAttribute (name <- att.name, isKey <- att.isKey,
                                     relation <- rse.relship )}

```

**Fig. 7.** *Class2Relational* transformation

Let us now consider the *Class2Relational* transformation defined in Fig. 7. This ATL program transforms entity-relationship (*ERSchema*) schemas into relational (*RELSchema*) ones, in which each *Entity* and relationship (*Relship*) is transformed into a *Relation*. Additionally, attributes (*ERAttribute*) of entities and relationships are transformed into attributes (*RELAttribute*) of relations. Finally, key attributes of entities (*isKey* is set to *true*) become attributes of the relations in which entities participate (*RelshipEnd*). This is a simplified version of the *Class2Relational* transformation of the ATL Zoo<sup>3</sup>, but enough for showing several examples of testing.

<sup>3</sup> <http://www.eclipse.org/atl/atlTransformations/#Class2Relational>.

```

(1) ER!Entity->allInstances()->forAll(e | e.attrs->forAll(a1,a2 | a1.name=
    a2.name implies a1=a2));
(2) ER!Relship->allInstances()->forAll(r | r.attrs->forAll(a1,a2 | a1.name =
    a2.name implies a1=a2));
(3) ER!Relship.allInstances()->forAll(e | e.attrs->collect(a | a.isKey)->
    size()=1) and ER!Entity.allInstances()->forAll(e | e.attrs->collect(a |
    a.isKey)->size()=1);
(4) REL!Relation->allInstances()->forAll(r | r.attrs->forAll(a1,a2 | a1.name
    =a2.name implies a1=a2));
(5) REL!Relation->allInstances()->forAll(r | r.attrs->collect(a | a.isKey)->
    size()=3 or r.attrs->collect(a | a.isKey)->size()=1);

```

**Fig. 8.** Input and output properties of *Class2Relational* transformation

Let us also consider the following set of input and output properties defined as OCL constraints (see Fig. 8). The OCL constraints on the source model establish that (1) all entities have attributes with distinct names, (2) all relationships have attributes with distinct names and (3) all entities and relationships have exactly one key. The OCL constraints on the target model establish that (4) all attributes of a relation have distinct names and (5) all relations have one or three keys (one for relations coming from entities, and three for relations coming from relationships). Let us suppose the human tester uses our tool to test the previous OCL constraints. Firstly, the human tester can select (1) and (2) as input properties and try to test (4) as output property. For the selected properties, the XML Schema should be modified in order to ensure that each entity and relationship have at least one attribute (setting *minOccurs* in *attrs* of *entities* and *relships* to one). Otherwise, entities and relationships will be generated without attributes, and the tester could not be able to test the properties (see example bellow). Now, the human tester calls the tester adding a couple of values for attribute names (“a” and “b”) to *enumeration* section of the XML Schema in the *nameType* simpleType. The tester answers (in 5,249 ms) as follows:

Output Property Falsifiable after 33 tests.  
Counterexample:

```

<xmi:XMI>
  <ERSchema name="a">
    <entities name="a">
      <attrs isKey="true" name="a"/>
    </entities>
    <relships name="a">
      <attrs isKey="true" name="a"/>
      <ends entity="//@ERSchema.0/@entities.0" name="a"/>
      <ends entity="//@ERSchema.0/@entities.0" name="a"/>
    </relships>
  </ERSchema>
</xmi:XMI>

```

Result:

```

<xmi:XMI>
  <rel:RELSchema name="a">
    <relations name="a">
      <attrs isKey="true" name="a"/>
      <attrs isKey="true" name="a"/>
      <attrs isKey="true" name="a"/>
    </relations>
  </rel:RELSchema>
</xmi:XMI>

```

```

    </relations>
  </rel:RELSchema>
  <rel:Relation name="a">
    <attrs isKey="true" name="a"/>
  </rel:Relation>
</xmi:XMI>

```

The test model illustrates that when entities and relationships share an attribute with the same name, the output property is violated. Thus, a stronger condition than (1) and (2) is required. Let us focus now on properties (3) and (5). In this case, names are not relevant and thus the human tester can use a different version of the XML Schema, in which he or she introduces two values (*true* and *false*) for *isKey*, adding them to *enumeration* section of the XML Schema, in the *isKeyType* simpleType. Additionally, *minOccurs* of *attrs* in *entities* and *relships* is set to one. In this case, the ATL tester answers (in 15,233 ms and number of steps 2):

Output Property Falsifiable after 109 tests.  
Counterexample:

```

<xmi:XMI>
  <ERSchema name="a">
    <entities name="a">
      <attrs isKey="true" name="a"/>
    </entities>
    <relships name="a">
      <attrs isKey="true" name="a"/>
      <ends entity="//@ERSchema.0/@entities.0" name="a"/>
      <ends entity="//@ERSchema.0/@entities.0" name="a"/>
      <ends entity="//@ERSchema.0/@entities.0" name="a"/>
    </relships>
  </ERSchema>
</xmi:XMI>

```

which means that the output property is violated after 109 tests when the number of relationship *ends* is three. Thus, the human tester should restrict the number of relationship ends to ensure the output property. Setting *maxOccurs* of *ends* in *relships* to two, the following answer (in 170,740ms and number of steps 3) is reported: Ok: passed 1296 tests, 464 valid. which means that 464 models satisfy the input and output properties from 1,296 randomly generated examples.

Let us now suppose that *attrs* is set to zero in *entities* and *relships*, and the number of steps is set to zero. In this case the ATL tester answers as follows: Unable to check the property, which means that from the models generated none of them satisfies the input property. This kind of answer is in most of cases reported when the selected number of steps is not enough to get valid source models. In this case, the solution is to increase the number of steps, or more appropriately and efficiently, to increase the number of elements, by setting a greater value of *minOccurs*. Sometimes, it can be solved by adding more values to types. For instance, in case of checking distinct values of a certain attribute at least two values are required. On the other hand, in the case of a buggy rule (of Fig. 7) is defined as follows:

```

rule RA2A { from att : ER!ERAttribute , rs : ER!Relship
  to t : REL!RELAttribute
  ( name <- att.name , isKey <- att.isKey , relation <- rs ) }

```

in which the condition  $att.relship = rs$  is omitted, the ATL tester answers (in 528 ms and number of steps 1) as follows:

Output Property Falsifiable after 5 tests.  
Counterexample:

```
<xmi:XMI>
  <ERSchema name="a">
    <entities name="a">
      <attrs isKey="true" name="a"/>
    </entities>
    <relships name="a">
      <attrs isKey="true" name="a"/>
      <ends entity="//@ERSchema.0/@entities.0" name="a"/>
      <ends entity="//@ERSchema.0/@entities.0" name="a"/>
    </relships>
  </ERSchema>
</xmi:XMI>
```

### 3.1 Benchmarks

Finally, we would like to show the benchmarks of ATL testing for several examples. Table 1 shows the execution time of the testing for four examples: Book2Publication (B2P), Families2Persons (F2P), Composed2Simple (CSP) and ER2RL transformation. Code of transformations is omitted. The execution times have been measured (in milliseconds, ms) for different number of steps (2, 3, 4 and 5). Additionally, the table shows the number of test models (m), and which of them are valid (v). We have not modified the XML Schema generated from the Ecore model, except in the XML Schema of Entity Relationship meta-model in which  $maxOccurs$  of  $ends$  is set to two (in order to satisfy source OCL constraints). We can see in this table that the ATL tester is able to generate up to 5,998 models in reasonable time (ER2RL transformation), and test input and output properties of 1,202 models in short time (F2P transformation). Let us remark that modifying XML Schema elements (i.e.,  $minOccurs$ ,  $maxOccurs$  and values of types of enumerations) execution time can be drastically altered. To take the original XML Schema is not usually recommended, but for validating the performance of our test model generator we have decided to leave XML Schema unchangeable. Usually, XML Schema should be modified to avoid large and useless models (non valid models). Obviously, when the output property is not satisfied the ATL tester is faster.

Table 1. Benchmarks

| Transformation | Steps = 2            | Steps = 3              | Steps = 4                | Steps = 5                    |
|----------------|----------------------|------------------------|--------------------------|------------------------------|
| B2P            | 7 m. 7 v. 1,068 ms.  | 16 m. 16 v. 1,331 ms.  | 38 m. 38 v. 3,363 ms.    | 97 m. 97 v. 14,631 ms.       |
| F2P            | 7 m. 7 v. 553 ms.    | 32 m. 32 v. 2,585 ms.  | 177 m. 177 v. 18,382 ms. | 1202 m. 1202 v. 207,670 ms.  |
| C2S            | 14 m. 4 v. 1,945 ms. | 61 m. 9 v. 4,360 ms.   | 274 m. 23 v. 19,024 ms.  | 1374 m. 69 v. 95,741 ms.     |
| ER2RL          | 16 m. 0 v. 4,205 ms. | 144 m. 4 v. 15,567 ms. | 841 m. 22 v. 106,848 ms. | 5998 m. 306 v. 1,089,177 ms. |

## 4 Related Work

From related works about test modeling we can distinguish two main lines of research. Those ones generating source test models using black-box techniques, and others using white-box techniques. Black box techniques do not use the transformation code or the specification (i.e., properties) to generate source test models. White box techniques use the transformation code or the specification to generate source test models. Most of white box techniques are based on OCL analysis and use constraint solvers.

In case of black-box approach, the authors of [18] use tracts to certify that a transformation works for some source test models generated from a script defined with the ASSL language. Tracts are pieces of specification focused on a particular scenario and each transformation can be specified by a set of tracts, enabling the partition of the full input space into smaller units. Tracts are specified by OCL constraints. The authors of [6] introduce model transformation testing in which an expected target model is used to validate transformations in which testing results are difference models. MANTra tool has been proposed in [4] to define test models for QVTO (QVT Operational) in terms of a transformation. The testing report is therefore also a model. MANTra checks output properties on target models. Test models are manually defined for each transformation, and they are used to show testing results. The authors of [8] adapt the classical category-partition method to qualify source models of transformations. They established partitions on meta-models in order to automatically generate test models. The specification language PAMOMO is used in [11] to express contracts: input, output properties as well as input-output properties (called invariants). These contracts are compiled into executable QVT transformations which are run to certify transformations. In case of properties are violated, detailed information (parts of the model in which a contract fails) is given to the user.

Our approach follows the same proposal as [18]: that is, it is able to partially validate a transformation concentrating on specific input and output properties. Due to the automation of our approach, the human tester can play with several combinations of input and output properties. The difference of our approach with this work is that we are able to automatically generate test models from the meta-model, while in [18] the human tester has to program a test model generator for each transformation using the ASSL language. Even more, the human tester in our approach can play with several sizes and values for model elements enabling the definition of a high diversity of test models. Properties on target models are specified in our approach by OCL constraints instead of using an expected output model like in [6]. Nevertheless, we plan to study the possibility of providing other techniques for checking output properties. Source test models are, in our approach, automatically generated from the source meta-model, but we plan to consider the testing procedure as a transformation, similarly to [4]. In our approach the human tester is responsible of the customization of test models, selecting from the source meta-model the elements and number of elements required to test the transformation. Thus, the partition is defined by the human tester, compared to the work [8]. This is an advantage given that the

partition can be more precise and suitable in some examples. We use OCL to express properties, and we are able to certify ATL transformations for the test models automatically generated. So far, the information provided to the human tester is limited to “Ok” and in the case of fail, the tool shows counterexamples, which are source models, together with the result of the transformation for the counterexamples. The human tester uses the counterexamples in order to detect the bugs in the transformation, enough in most of cases. However, we plan to study the possibility of extending our work in the same line as [11], providing richer information to the human tester about rules and parts of the target model in which properties are not satisfied.

With regard to white box approaches, the authors of [7] use a constraint solver to generate test models from the source meta-model OCL constraints, and similarly the proposal of [9], using also the dependence graph of ATL transformation. The authors of [10] use a category-partition based method adapted to OCL and an EMFtoCSP tool to generate test models. Generation of test models using SAT-solving techniques to complete hand-crafted partial models from the source meta-model is proposed in [16]. The authors of [12] derive test models from the transformation. The source test models are computed using SAT-solving techniques on OCL expressions generated from the specification. Typing errors of ATL transformations are studied in [5], and the authors are able analyzing ATL code to generate test cases for most common typing errors. OCL constraints are used to generate test cases from constraint solving. Finally, fault analysis in ATL rules is carried out in [3] by using OCL constraints. Our approach is black box, but we would like to extend our work in the future to white box techniques. In our case, white box testing means to be able to automatically select the number and value of elements (classes, attributes and associations) to which test models are generated. In other words, automatize the selection made by the human tester. It involves to analyze ATL code and OCL constraints like in the quoted approaches.

## 5 Conclusions and Future Work

In this paper we have presented how to automatically generate random Ecore models for testing ATL transformations. We have described an Ecore to XML Schema transformation that makes to generate Ecore models according to human tester’s choices possible. We have showed how the developed ATL tester is able to show counterexamples of OCL constraints on target models, when the testing fails, and to certify ATL transformations when OCL constraints on source and target models are satisfied. As future work, we would like to extend our approach as follows. Firstly, we would like to extend our tool with input-output properties (called invariants in the context of model transformation). Input-output property testing is of great interest since transformations can fulfill input and output properties but fail in source and target model mapping, which is only detected from input-output properties. Secondly, we plan to work in the improvement of the ATL tester, showing better results of tests: which property is not satisfied,

and which part of the model is involved. Thirdly, we would like to work on white box testing, using ATL code to automatically generate a pruned XML Schema with values taken from the transformation code. Our approach is black-box, and the human tester designs the test models by selecting items, number of them and values. Thus, test coverage, for instance, is decided by the human tester. In white-box testing, we will study test coverage. Finally, we would like to implement a Web tool to test ATL programs.

## References

1. Almendros-Jiménez, J.M., Becerra-Terón, A.: XQuery testing from XML schema based random test cases. In: Chen, Q., Hameurlain, A., Toumani, F., Wagner, R., Decker, H. (eds.) DEXA 2015. LNCS, vol. 9262, pp. 268–282. Springer, Heidelberg (2015)
2. Anand, S., Burke, E.K., Chen, T.Y., Clark, J., Cohen, M.B., Grieskamp, W., Harman, M., Harrold, M.J., McMinn, P., et al.: An orchestrated survey of methodologies for automated software test case generation. *J. Syst. Softw.* **86**(8), 1978–2001 (2013)
3. Burgueno, L., Troya, J., Wimmer, M., Vallecillo, A.: Static fault localization in model transformations. *IEEE Trans. Softw. Eng.* **41**(5), 490–506 (2015)
4. Ciancone, A., Filieri, A., Mirandola, R.: Testing operational transformations in model-driven engineering. *Innovations Syst. Softw. Eng.* **10**(1), 19–32 (2014)
5. Cuadrado, J.S., Guerra, E., de Lara, J.: Uncovering errors in ATL model transformations using static analysis and constraint solving. In: 2014 IEEE 25th International Symposium on Software Reliability Engineering, pp. 34–44. IEEE (2014)
6. Finot, O., Mottu, J.-M., Sunyé, G., Attiogbé, C.: Partial test oracle in model transformation testing. In: Duddy, K., Kappel, G. (eds.) ICMB 2013. LNCS, vol. 7909, pp. 189–204. Springer, Heidelberg (2013)
7. Fiorentini, C., Momigliano, A., Ornaghi, M., Poernomo, I.: A constructive approach to testing model transformations. In: Tratt, L., Gogolla, M. (eds.) ICMT 2010. LNCS, vol. 6142, pp. 77–92. Springer, Heidelberg (2010)
8. Fleurey, F., Baudry, B., Muller, P.-A., Le Traon, Y.: Qualifying input test data for model transformations. *Softw. Syst. Model.* **8**(2), 185–203 (2009)
9. González, C.A., Cabot, J.: ATLTest: a white-box test generation approach for ATL transformations. In: France, R.B., Kazmeier, J., Breu, R., Atkinson, C. (eds.) MODELS 2012. LNCS, vol. 7590, pp. 449–464. Springer, Heidelberg (2012)
10. González, C.A., Cabot, J.: Test data generation for model transformations combining partition and constraint analysis. In: Di Ruscio, D., Varró, D. (eds.) ICMT 2014. LNCS, vol. 8568, pp. 25–41. Springer, Heidelberg (2014)
11. Guerra, E., de Lara, J., Wimmer, M., Kappel, G., Kusel, A., Retschitzegger, W., Schönböck, J., Schwinger, W.: Automated verification of model transformations based on visual contracts. *Autom. Softw. Eng.* **20**(1), 5–46 (2013)
12. Guerra, E., Soeken, M.: Specification-driven model transformation testing. *Softw. Syst. Model.* **14**(2), 623–644 (2015)
13. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: ATL: a model transformation tool. *Sci. Comput. Program.* **72**(1), 31–39 (2008)
14. Selim, G.M.K., Cordy, J.R., Dingel, J.: Model transformation testing: the state of the art. In: Proceedings of the First Workshop on the Analysis of Model Transformations, pp. 21–26. ACM (2012)

15. Sen, S., Moha, N., Baudry, B., Jézéquel, J.-M.: Meta-model pruning. In: Schürr, A., Selic, B. (eds.) MODELS 2009. LNCS, vol. 5795, pp. 32–46. Springer, Heidelberg (2009)
16. Sen, S., Mottu, J.-M., Tisi, M., Cabot, J.: Using models of partial knowledge to test model transformations. In: Hu, Z., de Lara, J. (eds.) ICMT 2012. LNCS, vol. 7307, pp. 24–39. Springer, Heidelberg (2012)
17. Utting, M., Pretschner, A., Legeard, B.: A taxonomy of model-based testing approaches. *Softw. Test. Verification Reliab.* **22**(5), 297–312 (2012)
18. Vallecillo, A., Gogolla, M., Burgueño, L., Wimmer, M., Hamann, L.: Formal specification and testing of model transformations. In: Bernardo, M., Cortellessa, V., Pierantonio, A. (eds.) SFM 2012. LNCS, vol. 7320, pp. 399–437. Springer, Heidelberg (2012)
19. Völter, M., Stahl, T., Bettin, J., Haase, A., Helsen, S.: *Model-Driven Software Development: Technology, Engineering, Management*. Wiley, Chichester (2013)



<http://www.springer.com/978-3-319-45546-4>

Model and Data Engineering

6th International Conference, MEDI 2016, Almería,  
Spain, September 21-23, 2016, Proceedings

Bellatreche, L.; Pastor, Ó.; Almendros Jiménez, J.M.;  
Ait-Ameur, Y. (Eds.)

2016, XVII, 360 p. 121 illus., Softcover

ISBN: 978-3-319-45546-4