Mutations in ATL Transformations and their Identification with Matching Tables –Technical Report–

Javier Troya[1], Loli Burgueño[2], Manuel Wimmer[1], and Antonio Vallecillo[2]

[1] Business Informatics Group, Vienna University of Technology, Austria
{troya,wimmer}@big.tuwien.ac.at
[2] Dpto Lenguajes y Ciencias de la Computación, Universidad de Málaga, Spain
{loli,av}@lcc.uma.es

Abstract. In this technical report we explain with detail the experiments realized in the context of tractable fault localization in model transformations with regards to the different mutations performed in different model transformations. For each mutation realized, we show the computed matching tables as well as how to identify the guilty rule.

1 Introduction

This document reports different mutations, and how such mutations affect the matching tables and the identification of the guilty rule, in different model transformation scenarios. It is created to backup part of the assertions presented in our paper [3]. In such work, a light-weight approach to automatically checking model transformations is presented. It is based on matching functions that establish alignments between specifications and implementations using the metamodel footprints, i.e., the metamodel elements used. The approach is implemented for the combination of Tracts and ATL, both residing in the Eclipse Modeling Framework, and is supported by the corresponding toolkit[3].

When a transformation is mutated, the types extracted from such transformation might be different, producing a different alignment between such types and those obtained from the Tracts. The purpose of a mutated transformation is to emulate a possible mistaken transformation that could have been created by the user. Then, by analyzing the matching tables obtained with our approach, we would like to obtain the rule(s) that make certain constraints fail.

In order to identify possible mutations of an ATL transformation, we have used the information presented in a paper focused on co-evolution of model transformations [2]. In such paper, an evolution in an ATL model transformation is classified according to several changes performed in different elements of the ATL metamodel – in fact, we have considered the subset of classes and relationships that represent the declarative part, as shown in Figure 1. Table 1

[3] This toolkit is available from http://atenea.lcc.uma.es/index.php/Main_Page/Resources/MTB

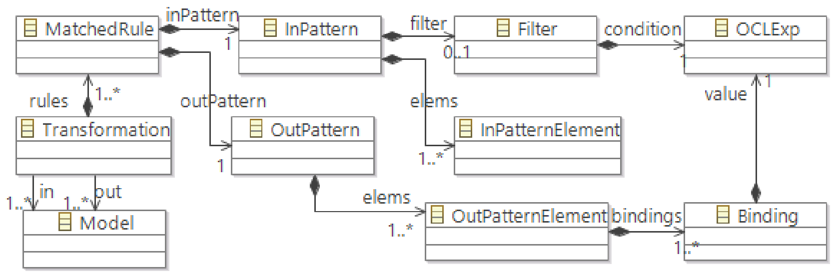


Fig. 1. Simplified ATL metamodel (from [2])

displays all the possible mutations as well as what they would mean in the output models in terms of elements that are missing or that should not be there. In the table, in order to keep the explanation short, we only mention elements, although of course relationships between them must also be taken into account.

Table 1. Possible Mutations

Concept	**Mutation**	**Meaning in the Produced Models**
MatchedRule	Addition	More elements than necessary are created
	Deletion	There are some elements that are not created
	Modification (*name* feature)	It does not affect the output model, only the trace model
InPatternEl.	Addition	More elements than necessary are created by the rule where it is added
	Deletion	Less elements than necessary are created by the rule where it is added
	Modification (*class* feature)	This means changing the type of an InPatternElement. It is the same as removing one and adding a different one. For this reason, there will be a higher or lower number of elements created and all of them with the features improperly set
	Modification (*name* feature)	It does not affect the output model, only the trace model
Filter	Addition	This makes the matching of the rule more restrictive, so less elements than expected are generated
	Deletion	This makes the matching of the rule less restrictive, so more elements than expected are generated
	Modification (*condition* feature)	This makes the matching of the rule either more or less restrictive, having the same consequence as one of the situations described above
OutPatternEl.	Addition	More elements than expected are created
	Deletion	Less elements than expected are created
	Modification (*class* feature)	This is considered as the Deletion and Addition of an OutPatternElement
	Modification (*name* feature)	It does not affect the output model, only the trace model
Binding	Addition	The feature represented by the binding is initialized and it should not be
	Deletion	A feature that should be initialized is not
	Modification (*value* feature)	The feature represented by the binding is improperly initialized
	Modification (*feature* feature)	This is considered as the Deletion and Addition of the binding

2 Mutation Experiments

As mentioned in the introduction, and in order to identify possible mutations of an ATL transformation, we have used the information presented in a paper focused on co-evolution of model transformations [2]. In such paper, an evolution in an ATL model transformation is classified according to several changes (Table 1). These changes are described according to the addition/modification/deletion of several concepts in a simplification of the ATL package of the ATL metamodel[4] that represents the declarative part of the rules (Fig. 1).

2.1 CPL2SPL Case Study

This transformation deals with behavioral models. Models conforming to CPL (Call Processing Language) [8] are transformed to models conforming to SPL (Session Processing Language) [5] (the transformation is called *CPL2SPL* for short). The *CPL2SPL* transformation [7] is a relatively complex example available from the ATL zoo[5]. It consists of 15 rules and 6 helpers. In total, they mean 348 lines of code, and they use 497 elements, 1114 links and 73 bindings.

The matching tables obtained for the original transformation are presented in Figure 2 – the cells that are below the threshold have been set to 0. As reported in [3], they have a precision of 0.8 and recall of 0.97 with the constraints used [6]. In the following subsections, when we write Cn we refer to constraint number n, and same thing for the rules, Rn.

Mutation CPL2SPL_1: Addition of a SimpleInPatternElement in R1. In this mutation, we have added a SimpleInPatternElement to the first rule (CPL2Program). The result is that, now, more than one Program are created for each CPL. The excerpt of the rule which is modified is shown in the Listing below.

```
rule CPL2Program { --R1
  from
    s : CPL!CPL,
    n : CPL!NodeContainer -- SimpleInPatternElement added
  to ...
```

Since the first rule has been modified, now there are some variations in the column representing such rule in the matching tables (Figure 3(a) – the values that change with respect to the original tables have been highlighted). By checking the constraints with our *TractsTool* [1,4,3], we observe that $C1$, $C2$, $C3$ and $C11$ are not satisfied. The first one checks if the number of CPL and Program instances is the same. By looking at the CC table, we easily check that $R1$ is

[4] A snapshot of the ATL package of the ATL metamodel is available from `http://atenea.lcc.uma.es/Descargas/ATL.png` (the references to the OCL package are not displayed)

[5] `http://www.eclipse.org/atl/atlTransformations`

[6] Such constraints can be found in `http://atenea.lcc.uma.es/index.php/Main_Page/Resources/MTB/CPL2SPL`

CC	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
C1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C3	0,5	0,63	0	0	0	0	0	0	0	0,63	0	0	0	0	0
C4	0	0	0,88	0	0	0	0	0	0	0	0	0	0	0	0
C5	0	0	0	0,18	0,45	0	0	0	0,27	0	0	0,3	0,18	0,27	0
C6	0	0	0	0,3	0,4	0,2	0,2	0,2	0	0,25	0,23	0,2	0,2	0	0
C7	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
C8	0	0	0	0	0	0	0	0	0,2	0	0	0,8	0	0,2	0
C9	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
C10	0	0	0	0,24	0	0	0	0,26	0	0	0,76	0	0	0	0
C11	0,688	0,25	0,5	0	0	0	0	0	0	0	0	0	0	0	0
C12	0	0	0	0	0,56	0,44	0,44	0,44	0	0	0	0	0,44	0	0
C13	0	0	0	0	0	0	0	0	0,5	0	0	0,8	0	0,5	0
C14	0	0	0	0	0,71	0	0	0	0	0	0	0	0	0	0
C15	0	0	0	0	0	0	0	0,9	0	0	0,8	0	0	0	0
C16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

RC	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
C1	0,111	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C2	0,167	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C3	0,111	0,21	0	0	0	0	0	0	0	0,21	0	0	0	0	0
C4	0	0	0,39	0	0	0	0	0	0	0	0	0	0	0	0
C5	0	0	0	0,17	0,12	0	0	0	0,5	0	0	0,2	0,2	0,6	0
C6	0	0	0	0,5	0,19	0,29	0,29	0,29	0	0,42	0,17	0,2	0,4	0	0
C7	0	0	0	0	0	0	0	0	0	0	0	0	0,4	0	0
C8	0	0	0	0	0	0	0	0	0,33	0	0	0,4	0	0,4	0
C9	0	0	0	0	0	0	0	0	0	0,25	0	0	0	0	0
C10	0	0	0	0,33	0	0	0	0,32	0	0	0,5	0	0	0	0
C11	0,306	0,17	0,22	0	0	0	0	0	0	0	0	0	0	0	0
C12	0	0	0	0	0,12	0,29	0,29	0,29	0	0	0	0	0,4	0	0
C13	0	0	0	0	0	0	0	0	0,42	0	0	0,2	0	0,5	0
C14	0	0	0	0	0,12	0	0	0	0	0	0	0	0	0	0
C15	0	0	0	0	0	0	0	0,32	0	0	0,15	0	0	0	0
C16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

RCR	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
C1	0,111	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C2	0,167	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C3	0,1	0,18	0	0	0	0	0	0	0	0,18	0	0	0	0	0
C4	0	0	0,37	0	0	0	0	0	0	0	0	0	0	0	0
C5	0	0	0	0,1	0,1	0	0	0	0,21	0	0	0,1	0,11	0,23	0
C6	0	0	0	0,23	0,15	0,13	0,13	0,13	0	0,19	0,11	0,1	0,15	0	0
C7	0	0	0	0	0	0	0	0	0	0	0	0	0,4	0	0
C8	0	0	0	0	0	0	0	0	0,13	0	0	0,4	0	0,14	0
C9	0	0	0	0	0	0	0	0	0	0,25	0	0	0	0	0
C10	0	0	0	0,16	0	0	0	0,17	0	0	0,43	0	0	0	0
C11	0,262	0,11	0,18	0	0	0	0	0	0	0	0	0	0	0	0
C12	0	0	0	0	0,11	0,21	0,21	0,21	0	0	0	0	0,27	0	0
C13	0	0	0	0	0	0	0	0	0,28	0	0	0,2	0	0,31	0
C14	0	0	0	0	0,11	0	0	0	0	0	0	0	0	0	0
C15	0	0	0	0	0	0	0	0,3	0	0	0,15	0	0	0	0
C16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Fig. 2. Matching tables for the original CPL2SPL transformation

causing the failure of this constraint – since it is actually the only rule that covers such constraint. Same thing happens with $C2$. There is only a value in cell $C2/R1$ for this constraint, so we can again quickly realize that $R1$ is causing the non compliance. As for $C11$, there are three candidate guilty rules (according to its row in the CC table). $R1$ has the highest value, so we start checking this rule and realize that it is the one making the constraint fail. Finally, the value of cell $C3/R1$ is below the threshold, having a false negative. In this case, we cannot directly realize that the non satisfaction of $C11$ is due to $R1$. However, by having already checked that $R1$ is the guilty one in the other constraints, $C11$ can have been already resolved too.

Mutation CPL2SPL_2: Modification of the value feature of a Binding in R3. In this mutation, there is an attribute that has been incorrectly initialized in a Binding (see Listing below), making $C4$ fail. In this case, the matching tables are the same as those in the correct transformation (Figure 2). By looking at the CC table, the user can quickly check that $R3$ is the guilty one. In this case $R3$ is also covered by $C11$, as shown in table RC, but the constraint is satisfied in this case.

```
rule Incoming2Method { --R3
  from s : CPL!Incoming
  to   t : SPL!Method (
          ...
          direction <- #"out", -- Binding modified
          ...
```

	CC R1	RC R1	RCR R1
C1	1	0,11	0,11
C2	1	**0,16**	**0,16**
C3	**0**	**0**	**0**
C4	0	0	0
C5	0	0	0
C6	0	0	0
C7	0	0	0
C8	0	0	0
C9	0	0	0
C10	0	0	0
C11	0,69	**0,29**	**0,25**
C12	0	0	0
C13	0	0	0
C14	0	0	0
C15	0	0	0
C16	0	0	0

(a) Mutation CPL2SPL_1

	CC R5	RC R5	RCR R5
C1	0	0	0
C2	0	0	0
C3	0	0	0
C4	0	0	0
C5	**0,55**	**0,14**	**0,12**
C6	**0,45**	**0,2**	**0,16**
C7	0	0	0
C8	0	0	0
C9	0	0	0
C10	0	0	0
C11	0	0	0
C12	0,56	**0,11**	**0,1**
C13	0	0	0
C14	0,71	**0,11**	0,11
C15	0	0	0
C16	0	0	0

(b) Mutation CPL2SPL_3

Fig. 3. Mutation CPL2SPL_1: Addition of a SimpleInPatternElement in R1. Mutation CPL2SPL_3: Modification of the Filter in R5

Mutation CPL2SPL_3: Modification of the Filter in R5. Here, we have modified the condition feature of the Filter in $R5$ by making the matching of this rule more restrictive. This is, now, a lower number of elements of type Proxy satisfy the condition that matches this rule. The code is shown in the Listing below.

```
rule Proxy2Select { --R5
  from
    s : CPL!Proxy ( --Filter modified
      not s.isSimple and not s.busy.contents.oclIsUndefined())
  to ...
```

Figure 3(b) shows the values that have been modified in the three tables for column $R5$. In this mutation, constraints $C5$, $C6$ and $C14$ are not satisfied. The highest value in the rows $C5$ and $C6$ in the CC table are $C5$/R5 and $C6$/R5, respectively (Figures 2 and 3(b)). Consequently, the user would start checking $R5$ in both cases and would realize that this one is the guilty rule. Regarding $C14$, it is only covered by $R5$, so, again, the user quickly discovers which the guilty rule is.

Mutation CPL2SPL_4: Modification of a Binding and addition of a SimpleOutPatternElement in R6. The first change in this mutation is the modification of the value feature of a Binding of the SimpleOutPatternElement of type SPL!SelectCase. This modification consists of the addition of a new element of type SPL!Constant in a sequence. Such element is actually added as a SimpleOutPatternElement of type SPL!StringConstant, along with its Binding. This mutation is shown in the Listing below.

```
rule Busy2SelectCase { --R6
  from s : CPL!Busy
  to ...
    t : SPL!SelectCase (
      commentsBefore <- Sequence {'// busy'},
```

```
    values <- Sequence {v,sc}, -- Binding modified
    statements <- Sequence {s.contents.statement}),
  ...
  sc : SPL!StringConstant ( -- SimpleOutPatternElement added
    value <- 'unspecified')
```

The modifications reflected in the matching tables caused by this mutation are shown in Figure 4(a). Although, as shown in Figure 2, $R6$ covers constraints $C6$ and $C12$, this mutation over $R6$ only causes $C12$ to fail. By having a look at the row for $C12$ in the CC table (Figures 2 and 4(a)), we see that cells $C12/R5$ and $C12/R6$ have the highest value, 0.56. Since there is a draw, we need to have a look at table RCR, where the value of $C12/R6$ is higher. Thereby, the user would start checking if $R6$ is the rule that has caused $C12$ to fail, discovering that it is.

	CC R6	RC R6	RCR R6
C1	0	0	0
C2	0	0	0
C3	0	0	0
C4	0	0	0
C5	0	0	0
C6	0,2	**0,25**	0,13
C7	**0,5**	**0,13**	**0,11**
C8	0	0	0
C9	0	0	0
C10	0	0	0
C11	0	0	0
C12	**0,56**	**0,31**	**0,25**
C13	0	0	0
C14	0	0	0
C15	0	0	0
C16	0	0	0

(a) Mutation CPL2SPL_4

	CC R8	RC R8	RCR R8
C1	0	0	0
C2	0	0	0
C3	0	0	0
C4	0	0	0
C5	**0,18**	**0,22**	**0,11**
C6	0,2	**0,44**	**0,16**
C7	0	0	0
C8	0	0	0
C9	0	0	0
C10	**0**	**0**	**0**
C11	0	0	0
C12	0,44	**0,44**	**0,29**
C13	0	0	0
C14	0	0	0
C15	**0,3**	**0,17**	**0,12**
C16	0	0	0

(b) Mutation CPL2SPL_5

Fig. 4. Mutation CPL2SPL_4: Modification of a Binding and addition of a SimpleOutPatternElement in R6. Mutation CPL2SPL_5: Deletion of a Binding and a SimpleOutPatternElement, along with its Binding, in R8.

Mutation CPL2SPL_5: Deletion of a Binding and a SimpleOutPatternElement, along with its Binding, in R8. This mutation emulates the circumstance in which the user has forgotten to create an element of type SPL!RedirectionErrorResponse in a rule, and there is a constraint that identifies this mistake. In this way, the mutation consists of the deletion of a SimpleOutPatternElement, along with its Binding, and yet a second Binding, as shown in the Listing below.

```
rule Redirection2SelectCase { --R8
  from  s : CPL!Redirection
  to
    t : SPL!SelectCase (...),
    v : SPL!ResponseConstant (
      --response <- r -- Binding deleted )
    --r : SPL!RedirectionErrorResponse ( -- SimpOutPattElement deleted
      --errorKind <- OclUndefined -- Binding deleted)
```

This modification changes some values of the columns belonging to $R8$, as shown in Figure 4(b). Out of the three constraints covered by $R8$ – $C6$, $C12$ and $C15$ –, only the last one is not satisfied when this mutation is applied. By looking at row $C15$ in table CC, the user would first check whether $R11$ is the guilty one. After she realizes it is not, the next rule to check is $R8$, which is in this case the guilty one.

Mutation CPL2SPL_6: Addition of a Filter in R9. In this mutation we insert a Filter in $R9$, as shown in the Listing below. By doing so, we are making the condition to match this rule be more restrictive, so less elements will be created in the output model.

```
rule Default2SelectDefault { --R9
  from
    s : CPL!Default (s.contents.oclIsUndefined()) -- Filter added
  to ...
```

$R9$ covers both $C5$ and $C13$, and they are both not satisfied in this case. The effects of this mutation in the matching tables is shown in Fig. 5(a). If the user starts having a look at row $C5$ in the CC table, she first starts checking $R5$ and realizing it is not the one that made the constraint fail. Then, $C5/R9$ and $C5/R14$ have the same value, so we check the RCR table to resolve the tie. In such table, the value of the latter cell is slightly higher, so the user checks $R14$ and realizes it is not the one causing the non compliance of the constraint. Thereby, she ends up discovering the guilty rule, in the third step. As for the other constraint, $C13$, there are three rules that could be making it fail: $R9$, $R12$ and $R14$. Cell $C13/R12$ has the highest value, so it is the one the user would check in the first place, realizing it is not the guilty one. The cells in the other two rules have the same value, so we need to check the RCR table, where the value of $C13/R14$ is a bit higher. Consequently, the user would realize which the guilty rule is in the first step.

Mutation CPL2SPL_7: Modification of Class feature in a SimpleOutPatternElement and deletion of Binding in R11. In this mutation, we have modified the class feature of a SimpleOutPatternElement of type SPL!HeadedMessageField, being the new type SPL!ReasonMessageField, and have removed a binding because the attribute initialized in it is not included in the new type. The transformation still works properly because, even if this SimpleOutPatternElement is referenced from a Binding of another SimpleOutPatternElement, both classes have a common super type. The mutated rule is shown in the Listing below.

```
rule Redirect2Return { --R11
  from ...
  to ...
    withExp : SPL!WithExp (
      exp <- v,
      msgFields <- Sequence {reason, contact}),
    ...
    contact : SPL!ReasonMessageField ( -- SimpOutPattElement modified
      --headerId <- '#contact:', -- Binding deleted
      exp <- contactConstant),
    ...
```

	CC R9	RC R9	RCR R9
C1	0	0	0
C2	0	0	0
C3	0	0	0
C4	0	0	0
C5	0,27	**0,43**	**0,2**
C6	0	0	0
C7	0	0	0
C8	0,2	**0,29**	0,13
C9	0	0	0
C10	0	0	0
C11	0	0	0
C12	0	0	0
C13	0,5	**0,36**	**0,25**
C14	0	0	0
C15	0	0	0
C16	0	0	0

(a) Mutation CPL2SPL_6

	CC R11	RC R11	RCR R11
C1	0	0	0
C2	0	0	0
C3	0	0	0
C4	0	0	0
C5	0	0	0
C6	0,23	**0,2**	**0,12**
C7	0	0	0
C8	0	0	0
C9	0	0	0
C10	**0,68**	0,5	**0,4**
C11	0	0	0
C12	0	0	0
C13	0	0	0
C14	0	0	0
C15	0,8	**0,17**	**0,17**
C16	0	0	0

(b) Mutation CPL2SPL_7

Fig. 5. Mutation CPL2SPL_6: Addition of a Filter in R9. Mutation CPL2SPL_7: Modification of class feature in a SimpleOutPatternElement and deletion of Binding in R11.

The columns of the matching tables affected by the changes performed are shown in Fig. 5(b). The modified rule covers $C10$ and $C15$, although the changed applied only makes the former be non complied. By having a look in table CC to row $C10$, we see we need to check first $R11$, so we quickly discover the guilty rule.

Mutation CPL2SPL_8: Deletion of R1. This is a particular case of mutation that has side effects on the shape of the matching tables.

The procedure to detect the guilty rule is different because there is no constraint violated. In order to detect the guilty rule, the CC and RCR matching tables have to be checked to make sure that there is no constraint that do not match any rule.

In CPL2SPL_8, R1 is removed and, therefore, the matching tables shown in Figure 2 are missing their first column. The sign that alerts the user that there is a mistake in the transformation is that, in tables CC and RCR, the rows that involve C1 and C2 do not have any value.

2.2 UML2ER Case Study

This transformation consists of the project that resides in the field of structural modeling and deals with the generation of Entity Relationship (ER) Diagrams from UML Class Diagram Models. The transformation receives simplified versions of UML class diagrams as inputs and generates entity-relationship diagrams as output. We have extended the metamodels for the UML2ER case study presented in [10]. They are illustrated in Fig. 6. This transformation can be considered as one of size between medium and small, consisting of 8 rules and no helper. In total, the rules involve 77 lines of code, and they use 86 elements,

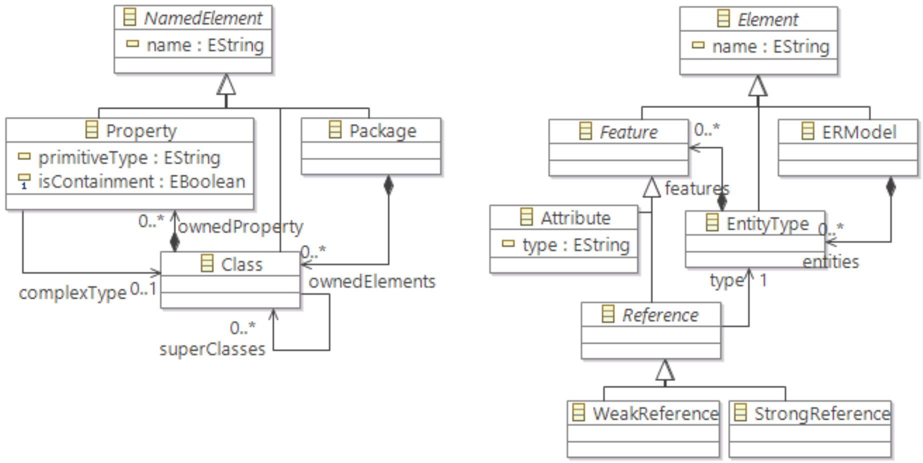


Fig. 6. The UML and ER metamodels.

201 links and 5 bindings. Finally, there are many rules inheriting others in this example. In fact, all rules are either an abstract rule or extend an abstract one.

The matching tables obtained for the original transformation are presented in Fig. 7 – the cells that are below the threshold have been set to 0. As we have reported in [3], they have a precision of 0.84, recall of 1, f-measure of 0.91 and utility average of 0.9, with the constraints used [7]. In the following subsections, when we write Cn we refer to constraint number n, and same thing for the rules, Rn.

In the following we present the different mutations that we have applied on this example. For each of them, we only show those columns of the matching tables where the values change – we have highlighted those values. The constraints that are violated for each mutation are computed with our *TractsTool* [1,4,3]. Please not that this transformation has a high degree of inheritance between rules. For this reason, a small mutation in one rule may cause the failure of several constraints.

Mutation UML2ER_1: Modification of the value feature of a Binding in R1. In this mutation, the only attribute that is initialized in $R1$ has been modified. Concretely, now the value in the Binding is always set to “name” (see Listing below).

```
abstract rule NamedElement{ --R1
  from s : SimpleUML!NamedElement
  to t : ER!Element(name <- 'name') -- Binding modified
}
```

[7] Such constraints can be found in http://atenea.lcc.uma.es/index.php/Main_Page/Resources/MTB/UML2ER

CC	R1	R2	R3	R4	R5	R6	R7	R8
C1	1	0,5	0	0	0	0	0	0
C2	0,8	0,4	0,4	0	0	0,4	0	0
C3	0,67	0	0,25	0,5	0,38	0,5	0,5	0,5
C4	1	1	1	0,67	0,42	1	0,67	0,67
C5	1	1	0	0	0	0	0	0
C6	1	1	1	0	0	1	0	0
C7	0,67	0	0,5	1	0,75	1	1	1
C8	0,36	0	0,17	0,33	0,5	0,25	0,25	0,25
C9	0,37	0	0,22	0,22	0,17	0,33	0,33	0,22
C10	0,37	0	0,22	0,22	0,17	0,33	0,33	0,22

RC	R1	R2	R3	R4	R5	R6	R7	R8
C1	1	0,33	0	0	0	0	0	0
C2	1	0,33	0,5	0	0	0,33	0	0
C3	0,67	0	0,25	1	0,38	0,33	0,67	1
C4	0,5	0,33	0,5	0,67	0,21	0,33	0,44	0,67
C5	0,5	0,33	0	0	0	0	0	0
C6	0,5	0,33	0,5	0	0	0,33	0	0
C7	0,33	0	0,25	1	0,38	0,33	0,67	1
C8	0,54	0	0,25	1	0,75	0,25	0,5	0,75
C9	0,83	0	0,5	1	0,38	0,5	1	1
C10	0,83	0	0,5	1	0,38	0,5	1	1

RCR	R1	R2	R3	R4	R5	R6	R7	R8
C1	0,5	0,25	0	0	0	0	0	0
C2	0,44	0,22	0,29	0	0	0,22	0	0
C3	0,33	0	0,14	0,5	0,21	0,22	0,33	0,4
C4	0,5	0,25	0,33	0,33	0,14	0,25	0,27	0,33
C5	0,33	0,33	0	0	0	0	0	0
C6	0,33	0,33	0,5	0	0	0,33	0	0
C7	0,22	0	0,2	1	0,3	0,29	0,5	0,67
C8	0,22	0	0,11	0,33	0,43	0,14	0,19	0,21
C9	0,26	0	0,18	0,22	0,13	0,25	0,33	0,2
C10	0,26	0	0,18	0,22	0,13	0,25	0,3	0,22

Fig. 7. Matching tables for the original UML2ER transformation

Since many rules in the transformation inherit from this one, this mutation makes more than half of the constraints fail, specifically $C1$, $C2$, $C3$, $C8$, $C9$ and $C10$. Figure 8(a) shows the variations in the columns of the first rule in the matching tables. For $C1$, $C2$ and $C3$, the cell in the first column contains the highest value, so the user quickly identifies the guilty rule. For $C8$, the user would first check $R4$, after which she would check the guilty rule, $R1$. For $C9$ and $C10$, there are three cells with the same value – the ones for $R1$, $R6$ and $R7$. For this reason, the user needs to check the RCR table. She would first check $R7$ and realize it is not the guilty rule. After this, she would check either $R1$ or $R6$, since they have the same value in the three tables.

Mutation UML2ER_2: Addition of an OutPatternElement with two Bindings in R3. In this situation, now two elements of type EntityType are created for each Class. The features assigned to the new EntityType are those of the super classes of the input Class. The new rule is shown in the listing below.

```
rule Class extends NamedElement{ --R3
  from s: SimpleUML!Class
  to t: ER!EntityType (
       features <- s.ownedProperties),
     t2: ER!EntityType( -- Added OutPatternElement and Bindings
       features <- s.superClasses -> collect (sc | sc.ownedProperties),
       name <- 'fromSuper'
       )}
```

This mutation makes some values of the third column in the matching tables change, as shown in Fig. 8(b). Also, with the addition of the OutPatternElement,

	CC R1	RC R1	RCR R1
C1	**0,75**	1	**0,43**
C2	**0,6**	1	**0,38**
C3	**0,58**	**0,78**	0,33
C4	1	**0,67**	**0,67**
C5	1	**0,67**	**0,4**
C6	1	**0,67**	**0,4**
C7	0,67	**0,44**	**0,27**
C8	**0,31**	**0,61**	**0,2**
C9	**0,33**	**1**	**0,25**
C10	**0,33**	**1**	**0,25**

(a) Mutation UML2ER_1

	CC R3	RC R3	RCR R3
C1	0	0	0
C2	**0,6**	**0,6**	**0,43**
C3	0,25	**0,2**	**0,13**
C4	1	**0,4**	**0,29**
C5	0	0	0
C6	1	**0,4**	**0,4**
C7	0,5	**0,2**	**0,17**
C8	0,17	**0,2**	**0,1**
C9	0,22	**0,4**	**0,17**
C10	0,22	**0,4**	**0,17**

(b) Mutation UML2ER_2

Fig. 8. Mutation UML2ER_1: Modification of the value feature of a Binding in R1. Mutation UML2ER_2: Addition of an OutPatternElement with two Bindings in R3

$C6$ fails. Having a look at the CC table, we observe that in the row corresponding to $C6$ there are four cells with the same value, 1. These are $R1$, $R2$, $R3$ and $R6$. Consequently, we then need to have a look at the RCR table, where we can see that the highest value is in the cell corresponding to $R3$, so the guilty rule is found.

Mutation UML2ER_3: Modification of a Filter in R8. In this mutation we have modified the Filter of the last rule. With this change, we make the application of this rule more restrictive, i.e., now less elements of type Property will fire the rule. In particular, we add to the previous constraint that the name of the Property has to begin with the character 's'. It is shown in the next Listing.

```
rule StrongReferences extends References{ -- R8
  from
    s: SimpleUML!Property
     (s.isContainment and not s.name.oclIsUndefined() -- Modified Filter
        and s.substring(1,1) = 'c')
    to t: ER!StrongReference }
```

The modification in this rule makes four constraints fail: $C3$, $C4$, $C7$ and $C10$. Having a look at the matching tables (Fig. 9(a)), we see that for $C3$ and $C10$, the user quickly finds the guilty rule, since it has the highest value in table CC. Regarding $C4$, the cells belonging to four different rules, none of them being the guilty one, have the highest value, so they are checked first. Then, the cell in $R4$, $R7$ and $R8$ share the same value, so the user has to check the RCR table. There, $R8$ has the lowest value, so the guilty rule is found in this case after 7 steps. Finally, for $C7$, four rules check the highest value in the CC table: $R4$, $R6$, $R7$ and $R8$. After consulting the RCR table, the guilty rule is identified in the first step, after having a look at $R4$ and $R7$

Mutation UML2ER_4: Modification of the class feature in an OutPatternElement and deletion of a Binding in R5 This mutation consists of changing

	CC R8	RC R8	RCR R8
C1	0	0	0
C2	0	0	0
C3	**0,75**	**0,75**	**0,5**
C4	0,67	**0,33**	**0,22**
C5	0	0	0
C6	0	0	0
C7	1	**0,5**	**0,4**
C8	**0,42**	**0,63**	**0,31**
C9	**0,44**	1	**0,4**
C10	**0,44**	1	**0,44**

(a) Mutation UML2ER_3

	CC R5	RC R5	RCR R5
C1	0	0	0
C2	0	0	0
C3	**0,5**	**0,67**	**0,33**
C4	**0,67**	**0,44**	**0,27**
C5	0	0	0
C6	0	0	0
C7	**1**	**0,67**	**0,5**
C8	**0,42**	**0,83**	**0,36**
C9	**0,22**	**0,67**	**0,2**
C10	**0,22**	**0,67**	**0,18**

(b) Mutation UML2ER_4

Fig. 9. Mutation UML2ER_3: Modification of a Filter in R8. Mutation UML2ER_4: Modification of the class feature in an OutPatternElement and deletion of a Binding in R5

the class feature of the only OutPatternElement in R5. This way, instead of creating elements of type Attribute, elements of type WeakReference are created. The original Binding has also been deleted, since the attribute it was initializing does not exist in elements of type WeakReference. The new rule is shown below.

```
rule Attributes extends Property{
  from s: SimpleUML!Property (not s.primitiveType.oclIsUndefined())
  to t: ER!WeakReference ( -- OutPatternElement modified
    --type <- s.primitiveType Binding deleted
  )}
```

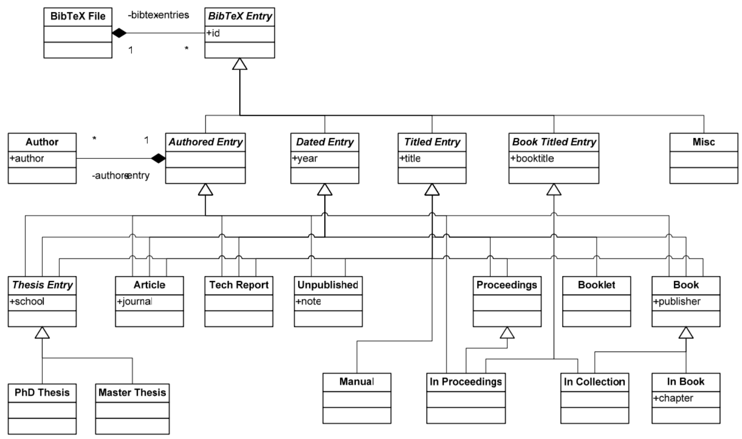
This mutation makes C8 fail. Having a look at the CC matching table (Fig. 9(b)), we can see that the cell with the highest value is precisely the one belonging to $R5$. Consequently, the guilty rule is quickly found.

2.3 BibTex2DocBook Case Study

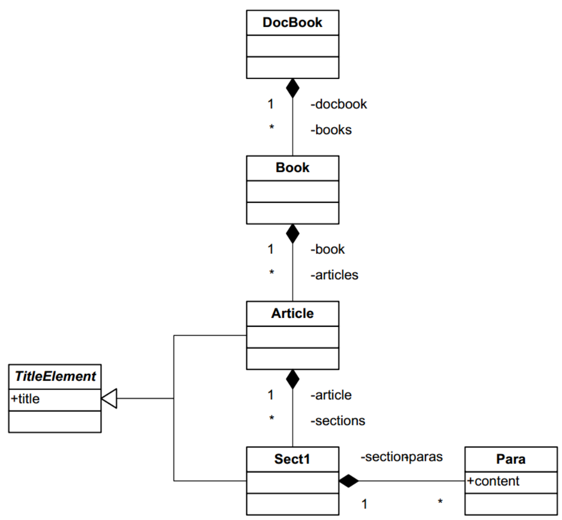
The BibTex2DocBook transformation, from here on BT2DB for short, is a model transformation that does not operate on modeling languages but on markup languages. This transformation is present in the ATL zoo. BibTeXML is an XML-based format for the BibTeX bibliographic tool, whose metamodel is shown in Fig 10(a). DocBook, in turn, is an XML-based format for document composition, its metamodel is shown in Fig. 10(b). The transformation consists of 9 rules and 4 helpers, what involve 286 lines of code, 449 elements, 1052 links and 25 bindings.

The matching tables for the original transformation are shown in Fig. 11 – the cells that are below the threshold have been set to 0. The accuracy of this case study is worse than for the previous ones [3]. Thus, it has a precision of 0.24, recall of 0.97, f-measure of 0.39 and utility-average of 0.54 [8]. In the following subsections, when we write Cn we refer to constraint number n, and same thing for the rules, Rn.

[8] This case study, results, and the constraints used can be found in `http://atenea.lcc.uma.es/index.php/Main_Page/Resources/MTB/BibTeXML2DocBook`



(a) BibTex Metamodel



(b) DocBook Metamodel

Fig. 10. Metamodels in the BibTex2DocBook case study

In the following we present the different mutations that we have applied on this example. For each of them, we only show those columns of the matching tables where the values change – we have highlighted those values. The constraints that are violated for each mutation are computed with our *TractsTool* [1,4,3].

Mutation BT2DB_1: Modification of the value feature of a Binding and deletion of an OutPatternElement together with its two Bindings in R1. This mutation emulates the circumstance where the user forgets to create one of

CC	R1	R2	R3	R4	R5	R6	R7	R8	R9
C1	1	0	0	0	0	0	0	0	0
C2	0,67	0	0	0	0	0	0	0	0
C3	0	1	0,5	0,5	0,5	0,5	0,5	0,5	0,5
C4	0,5	0,67	0,33	0,33	0,33	0,33	0,33	0,33	0,33
C5	0,57	0,29	0,36	0,29	0,29	0,29	0,29	0,29	0,29
C6	0,33	0,36	0,24	0,3	0,24	0,18	0,18	0,18	0,18
C7	0,29	0,33	0,21	0,25	0,21	0,17	0,17	0,17	0,17
C8	0,29	0,33	0,21	0,25	0,21	0,17	0,17	0,17	0,17
C9	0,29	0,33	0,21	0,25	0,21	0,17	0,17	0,17	0,17
C10	0,28	0,33	0,2	0,23	0,2	0,17	0,17	0,17	0,17
C11	0,33	0,36	0,24	0,3	0,24	0,18	0,18	0,18	0,18
C12	0,57	0,29	0,43	0,57	0,43	0,29	0,29	0,29	0,29
C13	0,46	0,25	0,33	0,42	0,33	0,25	0,25	0,25	0,25
C14	0,29	0,33	0,21	0,25	0,21	0,17	0,17	0,17	0,17
C15	0,28	0,33	0,2	0,23	0,2	0,17	0,17	0,17	0,17
C16	0,29	0,33	0,21	0,29	0,25	0,42	0,33	0,33	0,25

RC	R1	R2	R3	R4	R5	R6	R7	R8	R9
C1	0,14	0	0	0	0	0	0	0	0
C2	0,14	0	0	0	0	0	0	0	0
C3	0	1	0,5	0,4	0,5	0,4	0,5	0,5	0,67
C4	0,21	1	0,5	0,4	0,5	0,4	0,5	0,5	0,67
C5	0,29	0,5	0,63	0,4	0,5	0,4	0,5	0,5	0,67
C6	0,26	1	0,67	0,67	0,67	0,4	0,5	0,5	0,67
C7	0,25	1	0,63	0,6	0,63	0,4	0,5	0,5	0,67
C8	0,25	1	0,63	0,6	0,63	0,4	0,5	0,5	0,67
C9	0,25	1	0,63	0,6	0,63	0,4	0,5	0,5	0,67
C10	0,24	1	0,6	0,56	0,6	0,4	0,5	0,5	0,67
C11	0,26	1	0,67	0,67	0,67	0,4	0,5	0,5	0,67
C12	0,29	0,5	0,75	0,8	0,75	0,4	0,5	0,5	0,67
C13	0,26	0,5	0,67	0,67	0,67	0,4	0,5	0,5	0,67
C14	0,25	1	0,63	0,6	0,63	0,4	0,5	0,5	0,67
C15	0,24	1	0,6	0,56	0,6	0,4	0,5	0,5	0,67
C16	0,25	1	0,63	0,7	0,75	1	1	1	1

RCR	R1	R2	R3	R4	R5	R6	R7	R8	R9
C1	0,14	0	0	0	0	0	0	0	0
C2	0,13	0	0	0	0	0	0	0	0
C3	0	1	0,33	0,29	0,33	0,29	0,33	0,33	0,4
C4	0,16	0,67	0,25	0,22	0,25	0,22	0,25	0,25	0,29
C5	0,2	0,22	0,28	0,2	0,22	0,2	0,22	0,22	0,25
C6	0,15	0,36	0,21	0,24	0,21	0,14	0,15	0,15	0,17
C7	0,14	0,33	0,18	0,2	0,18	0,13	0,14	0,14	0,15
C8	0,14	0,33	0,18	0,2	0,18	0,13	0,14	0,14	0,15
C9	0,14	0,33	0,18	0,2	0,18	0,13	0,14	0,14	0,15
C10	0,14	0,33	0,17	0,19	0,17	0,13	0,14	0,14	0,15
C11	0,15	0,36	0,21	0,24	0,21	0,14	0,15	0,15	0,17
C12	0,2	0,22	0,33	0,4	0,33	0,2	0,22	0,22	0,25
C13	0,17	0,2	0,27	0,3	0,27	0,18	0,2	0,2	0,22
C14	0,14	0,33	0,18	0,2	0,18	0,13	0,14	0,14	0,15
C15	0,14	0,33	0,17	0,19	0,17	0,13	0,14	0,14	0,15
C16	0,14	0,33	0,18	0,25	0,23	0,42	0,33	0,33	0,25

Fig. 11. Matching tables for the original BT2DB transformation

the sections of a new book. The modifications made in the first rule are shown in the Listing below.

```
rule Main { --R1
  from bib : BibTeX!BibTeXFile
  to    ...
       art : DocBook!Article (
         title <- 'BibTeXML to DocBook',
         sections_1 <- Sequence{se1, se3, se4} --Binding Modified
       ),
     ...
       --se2 : DocBook!Sect1 ( --OutPatternElement deleted
         --title <- 'Authors list', --Binding deleted
         --paras <- thisModule.authorSet --Binding deleted
       --),
      ... }
```

This mutation makes $C2$, $C3$ and $C4$ fail. However, with this mutation the matching tables do not change, so they are the same as with the original transformation (Fig. 11). Regarding $C2$, it is only covered by $R1$, so the guilty rule is quickly found. As for $C4$, the user would first check if $R2$ is the guilty rule, after which she would check $R1$. In the cell $C3/R1$ there is a false negative, what means that the utility of such row is 0. Consequently, in this case the guilty rule cannot be found.

Mutation BT2DB_2: Modification of the value feature of a Binding in R4. In this mutation, the value of a Binding is modified, so that the string which is initialized in such Binding contains now more data, as shown in the Listing below.

```
rule TitledEntry_Title_NoArticle { --R4
  from ...
  to   ...
       title_para : DocBook!Para (
         content <- e.title + e.id -- Binding modified
       )}
```

Many constraints check for the correct initialization of the content attribute of different classes. Due to the high level of inheritance in this case study, a lot of constraints are not satisfied after this simple mutation. In particular, they are $C6$, $C7$, $C8$, $C9$, $C10$, $C11$, $C12$, $C13$, $C14$ and $C15$. For some of these constraints, the guilty rule is found in only one step, like for $C12$ for instance. There are others where the user would need two steps, as is the case of $C6$, or three steps, as with $C10$. They are all summarized in Table 2, explained in the Conclusion section.

Mutation BT2DB_3: Modification of the Filter in R6. This mutation implies the modification of the Filter in R6, by making the matching of the rule more restrictive. Now, less elements of type Article are going to fire the rule. Thus, al those Articles with only one author will not cause the firing of this rule. The rule is shown in the Listing below.

```
rule Article_Title_Journal { --R6
  from
    e : BibTeX!Article (
      thisModule.titledEntrySet->includes(e) and
      thisModule.articleSet->includes(e)
      and e.authors -> size() >= 2 -- code added
    )
  to ... }
```

The modification of this rule makes the constraint $C16$ fail. In this case, the highest value in the CC table for row $C16$ is in $R6$, so the guilty rule is found only in one step.

2.4 Ecore2Maude Case Study

This transformation is integrated in the e-Motions [9] tool. e-Motions is a Domain-Specific Modelling Language (DSML) that supports the specification, simulation and formal analysis of real-time systems. Internally, e-Motions runs Maude [6], a reflective language and system supporting both equational and rewriting logic specification and programming. As e-Motions deals with models conforming t

CC	R4
C1	0
C2	0
C3	0,5
C4	0,33
C5	0,29
C6	**0,36**
C7	**0,29**
C8	**0,29**
C9	**0,29**
C10	**0,27**
C11	**0,36**
C12	**0,71**
C13	**0,5**
C14	**0,29**
C15	**0,27**
C16	**0,33**

RC	R4
C1	0
C2	0
C3	**0,33**
C4	**0,33**
C5	**0,33**
C6	0,67
C7	**0,58**
C8	**0,58**
C9	**0,58**
C10	**0,53**
C11	0,67
C12	**0,83**
C13	0,67
C14	**0,58**
C15	**0,53**
C16	**0,67**

RCR	R4
C1	0
C2	0
C3	**0,25**
C4	**0,2**
C5	**0,18**
C6	**0,27**
C7	**0,22**
C8	**0,22**
C9	**0,22**
C10	**0,2**
C11	**0,27**
C12	**0,45**
C13	**0,33**
C14	**0,22**
C15	**0,2**
C16	**0,27**

(a) Mutation BT2DB_2

CC	R6
C1	0
C2	0
C3	**0,75**
C4	**0,5**
C5	0,29
C6	**0,27**
C7	**0,25**
C8	**0,25**
C9	**0,25**
C10	**0,25**
C11	**0,27**
C12	0,29
C13	0,25
C14	**0,25**
C15	**0,25**
C16	**0,5**

RC	R6
C1	0
C2	0
C3	**0,43**
C4	**0,43**
C5	**0,29**
C6	**0,43**
C7	**0,43**
C8	**0,43**
C9	**0,43**
C10	**0,43**
C11	**0,43**
C12	**0,29**
C13	**0,29**
C14	**0,43**
C15	**0,43**
C16	**0,86**

RCR	R6
C1	0
C2	0
C3	**0,38**
C4	**0,3**
C5	**0,17**
C6	**0,2**
C7	**0,19**
C8	**0,19**
C9	**0,19**
C10	**0,19**
C11	**0,2**
C12	**0,17**
C13	**0,15**
C14	**0,19**
C15	**0,19**
C16	**0,46**

(b) Mutation BT2DB_3

Fig. 12. Mutation BT2DB_2: Modification of the value feature of a Binding in R4. Mutation BT2DB_3: Modification of the Filter in R6

the Ecore metamodel, they need to be transformed to the Maude specification by means of the Ecore2Maude model transformation.

The Ecore2Maude transformation has 40 rules, where 7 of them are lazy rules and 27 are called rules, and 40 helpers in a total of 1397 lines of code.

Due to the size of the metamodels, they cannot be shown in this technical report but they can be accessed and downloaded, as well as the transformation and the constraints, on our webpage[9]. We have only written three constraints for this example that focus in a subset of the whole model transformation.

The matching tables for the original transformation and its constraints are the shown in Figure 13 where only the columns for those rules affected by the constraints (i.e., the columns with a value different from zero) are shown.

CC	R1	R4	R9	R10	R20	R29	R32	R38	R39
C1		0,21	0,41		0,21	0,21	0,21	0,14	0,29
C2			0,75	0,75			0,5		
C3	1			0,4			0,4		

RC	R1	R4	R9	R10	R20	R29	R32	R38	R39
C1		0,38	0,29		0,38	0,33	0,2	0,33	0,44
C2			0,15	0,5			0,13		
C3	0,2			0,33			0,13		

RCR	R1	R4	R9	R10	R20	R29	R32	R38	R39
C1		0,16	0,2		0,16	0,15	0,12	0,11	0,21
C2			0,14	0,43			0,12		
C3	0,2			0,2			0,11		

Fig. 13. Matching tables for the original E2M transformation

[9] http://atenea.lcc.uma.es/index.php/Main_Page/Resources/MTB/Ecore2Maude

Mutation E2M_1: Addition of a SimpleInPatternElement in R9. In this mutation, a SimpleInPatternElement has been added to rule Class2Sort so that for every combination between Class entities and EObject entities, the elements specified in the to part are created in the output model.

```
rule Class2Sort { -- R9
  from
    clazz : Ecore!EClass,
    mutation : Ecore!EObject  -- SimpleInPatternElement added
  to ...
```

The changes are reflected in the corresponding column (R9 – Class2Sort) in the matching tables. Figure 14(a) shows the values that have changed in bold. The increase in the number of types in the rule is reflected by the decrease of the metrics for the first constraint. In spite of that, the guilty rule for C1 is still detected on the first attempt because in its row, R9 is the cell with the higher value. C2 has not been affected by the mutation so its situation remains the same. As R10 is now more related to C2 (for R9 and C1 CC is 0.75, RC is 0.5 and RCR is 0.43; while for R9 and C2 CC is 0.75, RC is 0.15 and RCR is 0.14), R10 will be checked first and, secondly, R9, which is the guilty rule, will be checked.

CC	R9
C1	**0,36**
C2	0,75
C3	

RC	R9
C1	**0,25**
C2	0,15
C3	

RCR	R9
C1	**0,17**
C2	0,14
C3	

(a) Mutation E2M_1

CC	R9
C1	**0,36**
C2	0,75
C3	

RC	R9
C1	**0,26**
C2	**0,16**
C3	

RCR	R9
C1	**0,18**
C2	**0,15**
C3	

(b) Mutation E2M_2

Fig. 14. Mutation E2M_1: Addition of a SimpleInPatternElement in R9. Mutation E2M_2: Modification of a binding in Sort in R9.

Mutation E2M_2: Modification of a Binding in R9. In this mutation, the binding name has been modified in rule Class2Sort and now the string "mutation" is assigned to it instead of the name of the Class from which it is created. The following excerpt of code presents the code that has been modified.

```
rule Class2Sort { -- R9
  ...
  to
    sort : Maude!Sort (
      name <- 'mutation2', -- Binding modified
      ...
```

Figure 14(b) presents the columns from the matching tables that have changed after having done the mutation. Given that R9 is the guilty rule, C1 fails as well as C2. Both constraints are checked on the first attempt as their corresponding values to R9 in the CC table are the highest in their row.

Mutation E2M_3: Filter condition removed from R10. This mutation consists on the removal of the filter condition in the tenth rule. This means that not only the entities that fulfil the condition exposed in the filter match the left-hand side of this rule but all the entities of type EClassifier. Thus, the target model will contain elements that should not have been created.

```
unique lazy rule createSort{ -- R10
  from
    class : Ecore!EClassifier -- Filter condition removed
  to ...
```

Figure 15(a) presents the columns that correspond to R10, where the numbers that have changed have been emphasized. C3 has not been affected by the mutation. CC value for C2 decreased from 0.75 to 0.67 and RCR from 0.38 to 0.43, while the RC increased from 0.5 to 0.53. When C2 reports a failure, R10, which is the guilty rule, is checked in second place. And when C3 fails, it is the first or second rule to be checked, followed or preceded by R32 that has the same value in the CC table (0.4).

CC	R10
C1	
C2	**0,67**
C3	0,40

RC	R10
C1	
C2	**0,53**
C3	0,40

RCR	R10
C1	
C2	**0,38**
C3	0,22

(a) Mutation E2M_3

CC	R20
C1	**0,21**
C2	
C3	

RC	R20
C1	**0,30**
C2	
C3	

RCR	R20
C1	**0,14**
C2	
C3	

(b) Mutation E2M_4

Fig. 15. Mutation E2M_3: Filter condition removed from a SimpleInPatternElement in R10. Mutation E2M_4: SimpleOutPatternElement added with one binding in R20.

Mutation E2M_4: SimpleOutPatternElement added to R20 with one binding. The Listing below shows the SimpleOutPatternElement that has been inserted. After the mutation, for every Reference in the input model, an Operation is added to the output model as it should be, but in addition to that, a Parameter is added too.

```
rule Reference2Operation { -- R20
  ...
  to
    ...
    mutation : Maude!Parameter(
      label <- 'mutation'
    )
    ...
```

In Figure 15(b) the columns for the rule affected by the mutation are shown. Only the constraint C1 has a match with R20. CC did not suffer any change, RC went from 0.38 to 0.30 and RCR from 0.16 to 0.14. In spite of the differences in the values for the matching tables, the guilty rule is detected as it was before introducing the mutation, since table CC has not changed.

Mutation E2M_5: SimpleOutPatternElement deleted from R29. This mutation consists of removing a SimpleOutPattern from rule 29 which means that there will be missing elements in the output model. The following Listing shows the lines of code that have been commented.

```
rule Attribute2Operation { -- R29
  ...
  to
    -- opAtt : Maude!Operation(
      -- name <- att.maudeName(),
      -- "module" <- thisModule.sModule,
      -- coarity <- thisModule.sortAtt
    -- )
    ...
```

The side effects in the matching tables after the mutation are presented in Figure 16(a) where we can see that the relation between C1 and R29 is missing. This leads to a false negative (FN) and the impossibility to detect the guilty rule. This happens in this concrete case because R29 has very few types, so removing some of them implies a significant loss of information.

CC	R9
C1	*Missing*
C2	
C3	

RC	R9
C1	*Missing*
C2	
C3	

RCR	R9
C1	*Missing*
C2	
C3	

(a) Mutation E2M_5

CC	R1
C1	
C2	
C3	**0,80**

RC	R1
C1	
C2	
C3	**0,19**

RCR	R1
C1	
C2	
C3	**0,18**

(b) Mutation E2M_6

Fig. 16. Mutation E2M_5: SimpleOutPatternElement removed in R29. Mutation E2M_6: SimpleOutPatternElement removed in R1.

Mutation E2M_6: SimpleOutPatternElement deleted from R1. This mutation represents the same case as E2M_5 but the SimpleOutPatternElement is removed from rule 1, which has much more types in comparison with R29. The aim of E2M_6 is to show that not only the concrete mutation affects the results but also the concrete case in which it is applied. In the Listing below, the code that has been commented can be found.

```
entrypoint rule Initialize(){ -- R1
  ...
  to
    -- mSpec : Maude!MaudeSpec(
      -- els <- Sequence{sModuleEcore},
      -- printableEls <- Sequence{}
      -- ),
    ...
```

R1 is related to C3, and as they have plenty of types in common. The loss of several of them is barely reflected in the tables, so the results still allow the user to find the guilty rule. The value for CC went from 1 to 0.8, in the case of RC from 0.20 to 0.19 and RCR from 0.18 to 0.20. While the alignment is missing in

E2M_5, in this case the guilty rule is the first one to be checked despite having removed the SimpleOutPatternElement.

Mutation E2M_7: Filter added to R39. The mutation E2M_7 adds a filter condition to rule 39. This mutation is translated into missing entities in the output model because some of the elements that should match the left-hand side of the rule and create entities in the output element will not fulfil the filter so the rule will not be applied to them. The following Listing shows the filter added.

```
rule EnumLiteral2Operation { -- R39
  from
    enumLit : Ecore!EEnumLiteral (Ecore!EAttribute.allInstances->select(
        ↪att | att.iD)
    ...
```

The new values for the matching tables are in Figure 17 where we can see that after the mutation, the CC value for the guilty rule has increased and, therefore, it is detected even more clearly. The only inconvenience is that a false positive appears for C3, but as it is not higher than any other value in the row, it does not introduce noise when finding the guilty rule.

CC	R9
C1	**0,36**
C2	
C3	

RC	R9
C1	**0,45**
C2	
C3	

RCR	R9
C1	**0,25**
C2	
C3	

Fig. 17. Mutation E2M_7: Filter condition added to R39.

3 Conclusion

In Table 2 we show all the mutations that have been carried out. For each of them, we show which constraints are violated when applied such mutation, if it was able for the user to find the guilty rule, and the number of steps for finding such rule. By number of steps we mean the number of rules that the user needs to check in order to find the guilty one (including the latter).

As a summary, we injected a total of 21 mutations, causing 48 constraints to fail. All mutants were killed, i.e., all guilty rules were correctly identified by our approach. Only for three constraints that failed we could not identify the rule causing it but, in all cases, these rules caused the violation of several constraints, and the guilty rule was already identified as the one responsible for the violation of a different constraint that failed with the same mutation, such is the case with C3 in CPL2SPL_1, so the guilty rule was eventually identified. Regarding how many rules need to be checked before identifying the guilty one, our proposed approach needed an average of 1.78 rules to be checked.

Table 2. Table with the summary of mutations

Mutation	Constraints Violated	Guilty Rule Found?	Number of Steps
CPL2SPL_1	C1	Y	1
	C2	Y	1
	C3	N	-
	C11	Y	1
CPL2SPL_2	C4	Y	1
CPL2SPL_3	C5	Y	1
	C6	Y	1
	C14	Y	1
CPL2SPL_4	C12	Y	1
CPL2SPL_5	C15	Y	2
CPL2SPL_6	C5	Y	3
	C13	Y	3
CPL2SPL_7	C10	Y	1
UML2ER_1	C1	Y	1
	C2	Y	1
	C3	Y	1
	C8	Y	2
	C9	Y	2--3
	C10	Y	2--3
UML2ER_2	C6	Y	1
UML2ER_3	C3	Y	1
	C4	Y	7
	C7	Y	3
	C10	Y	1
UML2ER_4	C8	Y	1
BT2DB_1	C2	Y	1
	C3	N	-
	C4	Y	2
BT2DB_2	C6	Y	2
	C7	Y	2
	C8	Y	2
	C9	Y	2
	C10	Y	3
	C11	Y	2
	C12	Y	1
	C13	Y	1
	C14	Y	2
	C15	Y	3
BT2DB_3	C16	Y	1
E2M_1	C1	Y	1
	C2	Y	1--2
E2M_2	C1	Y	1
	C2	Y	1--2
E2M_3	C2	Y	2
E2M_4	C1	Y	3--4--5--6--7
E2M_5	C1	N	-
E2M_6	C3	Y	2
E2M_7	C1	Y	1

References

1. Atenea Research Group (Universidad de Málaga) in collaboration with Business Informatics Group (Vienna University of Technology): TractsTool (2013), `http://atenea.lcc.uma.es/index.php/Main_Page/Resources/Tracts`
2. Bergmayr, A., Troya, J., Wimmer, M.: From Out-Place Transformation Evolution to In-Place Model Patching. In: 29th IEEE/ACM International Conference on Automated Software Engineering (ASE 2014) (2014), accepted for publication
3. Burgueno, L., Troya, J., Wimmer, M., Vallecillo, A.: Static Fault Localization in Model Transformations. Submitted for Publication (2013)
4. Burgueño, L., Wimmer, M., Vallecillo, A.: Towards tracking *guilty* transformation rules. In: Proc. of the 1st Workshop on the Analysis of Model Transformations (AMT) @ MODELS'12. pp. 1–6. ACM DL (2012)
5. Burgy, L., Consel, C., Latry, F., Lawall, J., Palix, N., Reveillere, L.: Language technology for internet-telephony service creation. In: Proc. of ICC'06. pp. 1795–1800 (2006)

6. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.L.: All About Maude, LNCS, vol. 4350. Springer (2007)
7. Jouault, F., Bézivin, J., Consel, C., Kurtev, I., Latry, F.: Building DSLs with AMMA/ATL, a Case Study on SPL and CPL Telephony Languages. In: ECOOP Workshop on Domain-Specific Program Development (2006)
8. Lennox, J., Wu, X., Schulzrinne, H.: Call Processing Language (CPL): A language for user control of internet telephony services (2004), `http://www.ietf.org/rfc/rfc3880.txt`
9. Rivera, J.E., Durn, F., Vallecillo, A.: emotions: A graphical approach for modeling timedependent behavior of domain specific languages. available at http://atenea.lcc.uma.es/index.php/main_page/resources/e-motions (2009)
10. Wimmer, M., Martínez, S., Jouault, F., Cabot, J.: A catalogue of refactorings for model-to-model transformations. Journal of Object Technology 11(2), 1–40 (2012)