Movie Database Case: An EMF-INCQUERY Solution*

Gábor Szárnyas Oszkár Semeráth Benedek Izsó Csaba Debreceni

Ábel Hegedüs Zoltán Ujhelyi Gábor Bergmann

Budapest University of Technology and Economics,
Department of Measurement and Information Systems,
H-1117 Magyar tudósok krt. 2., Budapest, Hungary

{szarnyas, semerath, izso, debreceni, abel.hegedus, ujhelyiz, bergmann}@mit.bme.hu

This paper presents a solution for the Movie Database Case of the Transformation Tool Contest 2014, using EMF-INCQUERY and Xtend for implementing the model transformation.

1 Introduction

Automated model transformations are frequently integrated to modeling environments, requiring both high performance and a concise programming interface to support software engineers. The objective of the EMF-INCQUERY [2] framework is to provide a declarative way to define queries over EMF models. EMF-INCQUERY extended the pattern language of VIATRA with new features (including transitive closure, role navigation, match count) and tailored it to EMF models [1].

EMF-INCQUERY is developed with a focus on *incremental query evaluation*. The latest developments extend this concept by providing a preliminary rule execution engine to perform transformations. As the engine is under heavy development, the design of a dedicated rule language (instead of using the API of the engine) is currently subject to future work. Conceptually, the environment relies on graph transformation (GT) rules: conditions are specified as EMF-INCQUERY patterns, while model manipulation and environment configuration is managed using the Xtend language [3].

One case study of the 2014 Transformation Tool Contest describes a movie database transformation [4]. The main characteristics of the transformation related to the application of EMF-INCQUERY are that i) it only adds new elements to the input model (i.e. couple and group are does not modify the input model), and ii) it is non-incremental (i.e. creating a new group will not affect rule applicability).

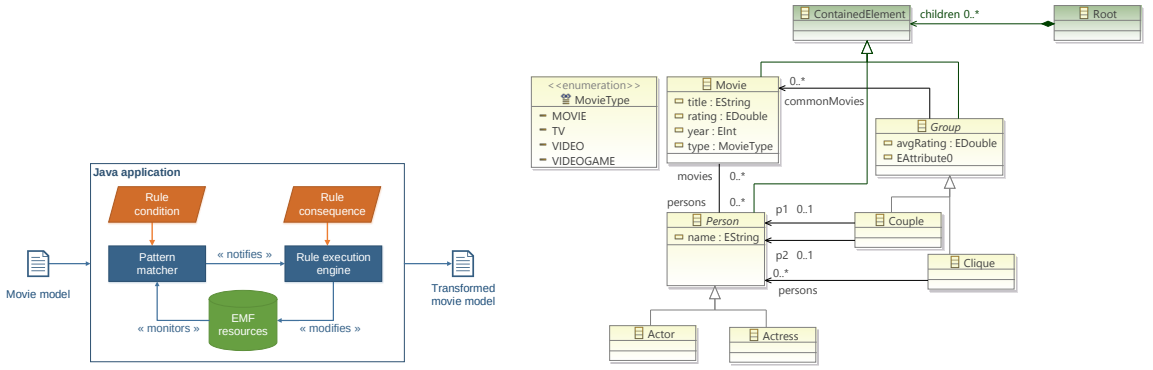
The rest of the paper is structured as follows: Section 2 gives an overview of the implementation, Section 3 describes the solution including measurement results, and Section 4 concludes our paper.

2 Architecture Overview

The overview of the rule-based solution is illustrated in Figure 1a. The input of the transformation is a *movie model*. The result is a *transformed movie model* including various groups (couples and *n*-cliques) and their average rating [4]. The transformation runs in a Java application, that uses *pattern matchers* provided by EMF-INCQUERY and model manipulation specified in Xtend. The pattern matcher *monitors*

*This work was partially supported by the MONDO (EU ICT-611125) and TÁMOP (4.2.2.B-10/1–2010-0009) projects. This research was realized in the frames of TÁMOP 4.2.4. A/1-11-1-2012-0001 „National Excellence Program – Elaborating and operating an inland student and researcher personal support system". The project was subsidized by the European Union and co-financed by the European Social Fund.

Submitted to:
TTC 2014



(a) The specification and runtime

(b) Our extended metamodel

Figure 1: Overview of our approach

the resources to incrementally update match sets. The application initially reads the input movie database, creates the output resources, then executes the transformation, and finally serializes the results into files.

In the Ecore model of the specification, no containment hierarchy is used and all objects are held in the `contents` list of the EMF resource. However, the performance of the transformation was affected by the resource implementation used (since it will determine the implementation of the list operations). To avoid this issue, we have extended the metamodel by a `Root` object (see Figure 1b). This object serves as a container for all `Group`, `Movie` and `Person` objects. According to our experiments, this increases the speed of the pattern matching by a factor of two. For compatibility with the specification, we ensured that our solution works with the models provided and persists outputs in a format without this root element.

3 Solution

3.1 Patterns and Transformations

Task 1: Generating Test Data The synthetic test data is generated in Xtend (see Listing A.2.1). The code tightly follows the specification defined in the case description [4].

Task 2: Finding Couples Couples are listed with the following pattern:

```
1 pattern personsToCouple(p1name, p2name) {
2   find cast(p1name, M1); find cast(p2name, M1);
3   find cast(p1name, M2); find cast(p2name, M2);
4   find cast(p1name, M3); find cast(p2name, M3);
5   M1 != M2; M2 != M3; M1 != M3;
6   check(p1name < p2name);
7 }
8 pattern cast(name, M) { Movie.persons.name(M, name); }
9 pattern personName(p, pName) { Person.name(p, pName); }
```

Note that the `cast` pattern returns the names of persons that play in a given movie. This is important since the names of the persons can be used to avoid symmetric matches in the `personsToCouple` pattern by sorting. The `Couple` objects are created and configured in Xtend (see `createCouples` in line 45 of Listing A.2.2). This includes setting the `p1` and `p2` references using a `personName` pattern and computing the `commonMovies` by simple set intersection operators (`retainAll`).

Task 3: Computing Average Rankings The average rankings are computed in Xtend by calculating the mean of the rating attributes of a couple's common movies (see calculateAvgRatings in line 122 of Listing A.2.2). The movies are enumerated with the following pattern:

```
1 pattern commonMoviesOfCouple(c, m) { Couple.commonMovies(c, m); }
```

Extension Task 1: Compute Top-15 Couples This task is mostly implemented in Xtend (see topGroupByRating in line 70 and topGroupByCommonMovies in line 84 of Listing A.2.2), however, it uses the groupSize pattern in order to filter the groups with the particular number of members.

```
1 pattern groupSize(group, S) {
2   Group(group);
3   S == count find memberOfGroup(_, group);
4 }
```

This pattern uses the count find construct which computes the number of matches for a given pattern. Additionally, specific comparators are used to sort and determine the top-15 lists by rating or number of common movies (see Listing A.2.4).

Extension Task 2: Finding Cliques The pattern for finding cliques is implemented similarly to the personsToCouple pattern 3.1. The pattern for 3-cliques is defined as follows:

```
1 pattern personsTo3Clique(P1, P2, P3) {
2   find cast(P1, M1); find cast(P2, M1); find cast(P3, M1);
3   find cast(P1, M2); find cast(P2, M2); find cast(P3, M2);
4   find cast(P1, M3); find cast(P2, M3); find cast(P3, M3);
5   M1 != M2; M2 != M3; M1 != M3;
6   check(P1 < P2); check(P2 < P3);
7   check(P1 < P3);
8 }
```

The creation of cliques is done similarly to couples (see createCliques in line 138 of Listing A.2.2). However, this pattern has a redundant check constraint, as $P_1 < P_2$ and $P_2 < P_3$ already imply $P_1 < P_3$. This works as a hint for the query engine and allows it to filter the permutation of the results (e.g. $(a_2, a_1, a_3), (a_1, a_3, a_2), \ldots$) earlier.

For performance considerations, additional patterns were defined manually for 4- and 5-cliques. For larger cliques ($n > 5$), patterns could be automatically generated using code generation techniques.

General solution for n-cliques. We also provide the outline for a more general solution (for arbitrary n values). For the sake of clarity, we will refer to couples as 2-cliques. In this approach, the cliques are built iteratively. Suppose we already have all k-cliques in the graph (e.g. we already added the 2-, 3-, 4- and 5-cliques with the previous patterns). To get the $(k+1)$-cliques, we look for a group g_0 and a person p_0 that (i) have at least 3 movies in common, (ii) $g = g_0 \cup \{p_0\}$ is a group that is not a subset of any other groups (see Figure 2).

Formally, (ii) can be expressed as $(\nexists g') : g \subseteq g'$. Using $g = g_0 \cup \{p_0\}$, we derive the following expression $(\nexists g') : (g_0 \subseteq g') \wedge (p_0 \in g')$. The $g_0 \subseteq g'$ expression can be formulated as follows: $(\forall p \in g_0) : p \in g'$. As the EMF-IncQuery Pattern Language does not have a universal quantifier, we rewrite this using the existential quantifier: $(\nexists p \in g_0) : p \notin g'$.

The resulting expression for condition (ii) is the following: $(\nexists g') : ((\nexists p \in g_0) : p \notin g') \wedge (p_0 \in g')$. We have implemented this general solution (see Listing A.2.3). Pattern subsetOfGroup implements condition (ii), while nextClique pattern is capable of determining the $(k+1)$-cliques given a model

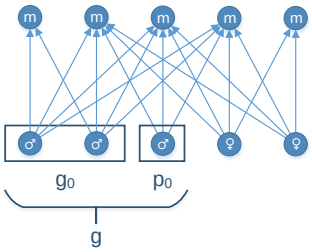


Figure 2: Matching 3-clique groups in the positive test pattern. g_0 is a couple.

containing all *k*-cliques. This approach is functionally correct, however, it only works for very small input models and hence is omitted from our measurements.

Extension Task 3: The average rankings are computed the same way as in *task 3*.

Extension Task 4: The top 15 average rankings are computed the same way as in *extension task 2*.

3.2 Optimizations

To increase the performance of the transformations, we carried out some optimizations. (1) The common movies of the two `Person` objects are computed from Xtend instead of EMF-INCQUERY. (2) The patterns for 3-, 4- and 5-cliques are implemented manually. (3) *Common subpatterns* were identified and extracted them into separate patterns, as the engine can reuse the pattern for each occurrence, and makes the query definition file easier to maintain. For an example, see the `cast` pattern in A.1.

3.3 Benchmark Results

The implementation was benchmarked in the SHARE cloud, on an Ubuntu 12.04 64-bit operating system running in a VirtualBox environment. The virtual machine used one core of an Intel Xeon E5-2650 CPU and had 6 GB of RAM. The transformations were ran in a timeout window of 10 minutes.

3.4 Synthetic model

Results are displayed in Figure 3. The diagram shows the transformation times for creating couples and cliques for synthetic models. The results show that the transformations run in near linear time.

The dominating factor of the running time is the initialization of the query engine. However, after initialization, creating groups can be carried out efficiently. Furthermore, our experiments showed that the limiting factor for our solution is the memory consumption of the incremental query engine. Given more memory, the solution is capable of transforming larger models as well.

3.5 IMDb model

In the given time range and memory constraints, the transformation of the IMDb model could only generate the couples and 3-cliques for the smallest instance model. Finding the couples took 3 minutes,

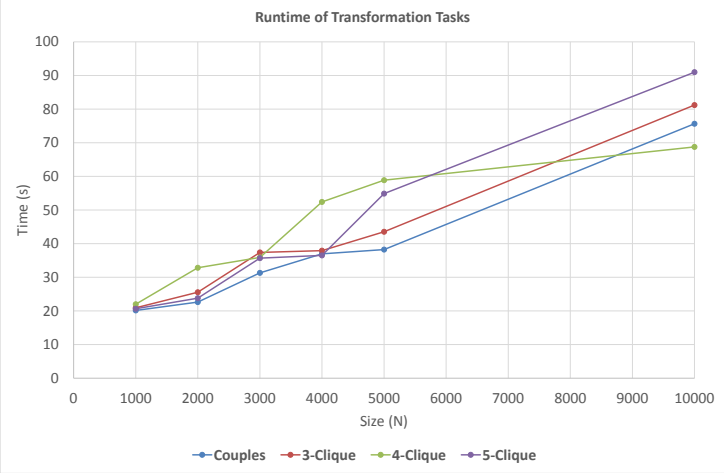


Figure 3: Benchmark results

while finding 3-cliques took 6. However, in case of a live and evolving model, our solution is capable of incrementally running the transformation which in practice results in near instantaneous response time.

3.6 Transformation Correctness and Reproducibility

Our solution was developed as Eclipse plug-ins, however, it is also available as a command line application compiled using the Apache Maven.The transformation runs correctly for the provided test cases on SHARE[1], and the source code is also available in a Git repository[2]. The results of the transformations were spot-checked for both synthetic and IMDb models.

4 Conclusion

In this paper we have presented our implementation of the Movie Database Case. The solution uses EMF-IncQuery as a model query engine: the transformation is specified using declarative graph pattern queries over EMF models for rule preconditions, and Xtend code for model manipulations. The main conclusion of the performance evaluation is that EMF-IncQuery's incremental approach is not a good fit for this case study as the transformation is very model manipulation dominant.

References

[1] Gábor Bergmann, Zoltán Ujhelyi, István Ráth & Dániel Varró (2011): *A Graph Query Language for EMF models*. In: *Theory and Practice of Model Transformations, Fourth Int. Conf., LNCS* 6707, Springer.

[2] Eclipse.org (2014): *EMF-IncQuery*. `http://eclipse.org/incquery/`.

[3] Eclipse.org (2014): *Xtend – Modernized Java*. `https://www.eclipse.org/xtend/`.

[4] Matthias Tichy Tassilo Horn, Christian Krause (2014): *The TTC 2014 Movie Database Case*. In: *7th Transformation Tool Contest (TTC 2014)*, EPTCS.

[1] `http://is.ieis.tue.nl/staff/pvgorp/share/?page=ConfigureNewSession&vdi=Ubuntu12LTS_TTC14_64bit_TTC14-EIQ-imdb.vdi`

[2] `https://git.inf.mit.bme.hu/w?p=projects/viatra/ttc14-eiq.git` (username: `anonymous`, no password).

A Appendix – Movie Database Case Transformation Code

A.1 EMF-INCQUERY Graph Patterns

```
package hu.bme.mit.ttc.imdb.queries

import "http://movies/1.0"


// Shorthand patterns
pattern personName(p, pName) {
  Person.name(p, pName);
}

// Actor with name is part of the case of movie M
pattern cast(name, M) {
  Movie.persons.name(M, name);
}

// Movie m is a common movie of Couple c
pattern commonMoviesOfCouple(c, m) {
  Couple.commonMovies(c, m);
}

/**
 * This pattern determines if a person is a member of a group.
 */
pattern memberOfGroup(person, group) {
  Couple.p1(group,person);
} or {
  Couple.p2(group,person);
} or {
  Clique.persons(group, person);
}

/**
 * This pattern determines the size of a group.
 */
pattern groupSize(group, S) {
  Group(group);
  S == count find memberOfGroup(_, group);
}

// Couple patterns
/**
 * This pattern looks for two person names (p1name, p2name), who were in the cast of
 * three different movies (M1, M2, M3).
 * The names are ordered lexicographically in order to list the same pair only one
 * (the match set contains only {(a1, a2)} instead of {(a1, a2), (a2, a1)}.
 */
pattern personsToCouple(p1name, p2name) {
  find cast(p1name, M1); find cast(p2name, M1);
  find cast(p1name, M2); find cast(p2name, M2);
  find cast(p1name, M3); find cast(p2name, M3);

  M1 != M2; M2 != M3; M1 != M3;

  check(p1name < p2name);
}

/**
 * This pattern looks for the common movies of a couple.
 * The couple is determined with the personsToCouple pattern.
 */
```

```
61 pattern commonMoviesToCouple(p1name, p2name, m) {
62   find personsToCouple(p1name, p2name);
63
64   Person.movies(p1, m);
65   Person.movies(p2, m);
66   Person.name(p1,p1name);
67   Person.name(p2,p2name);
68
69   check(p1name < p2name);
70 }
71
72 /**
73  * Returns with the number of common movies of a couple.
74  */
75 pattern countOfCommonMoviesOfCouple(p1, p2, n) {
76   Couple.p1(c, p1);
77   Couple.p2(c, p2);
78   n == count find commonMoviesOfCouple(c, _m);
79 }
80
81 // Clique patterns
82 /**
83  * Similarly to the couple pattern, this pattern looks for 3-cliques.
84  */
85 pattern personsTo3Clique(P1, P2, P3) {
86   find cast(P1, M1); find cast(P2, M1); find cast(P3, M1);
87   find cast(P1, M2); find cast(P2, M2); find cast(P3, M2);
88   find cast(P1, M3); find cast(P2, M3); find cast(P3, M3);
89
90   M1 != M2; M2 != M3; M1 != M3;
91
92   check(P1 < P2); check(P2 < P3);
93 }
94
95 /**
96  * Similarly to the couple pattern, this pattern looks for 4-cliques.
97  */
98 pattern personsTo4Clique(P1, P2, P3, P4) {
99   find cast(P1, M1); find cast(P2, M1); find cast(P3, M1); find cast(P4, M1);
100  find cast(P1, M2); find cast(P2, M2); find cast(P3, M2); find cast(P4, M2);
101  find cast(P1, M3); find cast(P2, M3); find cast(P3, M3); find cast(P4, M3);
102
103  M1 != M2; M2 != M3; M1 != M3;
104
105  check(P1 < P2); check(P2 < P3); check(P3 < P4);
106 }
107
108 /**
109  * Similarly to the couple pattern, this pattern looks for 5-cliques.
110  */
111 pattern personsTo5Clique(P1, P2, P3, P4, P5) {
112   find cast(P1, M1); find cast(P2, M1); find cast(P3, M1); find cast(P4, M1); find cast(
        P5, M1);
113   find cast(P1, M2); find cast(P2, M2); find cast(P3, M2); find cast(P4, M2); find cast(
        P5, M2);
114   find cast(P1, M3); find cast(P2, M3); find cast(P3, M3); find cast(P4, M3); find cast(
        P5, M3);
115
116  M1 != M2; M2 != M3; M1 != M3;
117
118  check(P1 < P2); check(P2 < P3); check(P3 < P4); check(P4 < P5);
119 }
```

A.2 Xtend Code

A.2.1 Generator Code

```
/**
 * This class implements the test model generator logic.
 */
class Generator {

  // The EMF resource on which the transformation operates
  public Resource r

  // We define this extension to help with model element creation
  extension MoviesFactory = MoviesFactory.eINSTANCE

  // method to generate an example of size N
  def generate(int N) {
    createExample(N);
  }

  // create N test cases in the model
  def createExample(int N) {
    (0 .. N - 1).forEach[createTest(it)]
  }

  // create a test cases in the model with parameter n
  def createTest(int n) {
    createPositive(n)
    createNegative(n)
  }

  // create a positive match for the test case
  // initialize some movies and actors/actresses
    // create interconnections according to a logic that will yield a positive match
  def createPositive(int n) {
    val movies = newArrayList()
    (0 .. 4).forEach[movies += createMovie(10 * n + it)]

    val a =   createActor("a" + (10 * n))
    val b =   createActor("a" + (10 * n + 1))
    val c =   createActor("a" + (10 * n + 2))
    val d = createActress("a" + (10 * n + 3))
    val e = createActress("a" + (10 * n + 4))

    val actors =   #[a, b, c, d, e]
    val firstTwo = #[a, b]
    val lastTwo =  #[         d, e]

    movies.get(0).persons += firstTwo;
    (1 .. 3).forEach[movies.get(it).persons += actors]
    movies.get(4).persons += lastTwo

    r.contents += actors
    r.contents += movies
  }

  // create a positive match for the test case
  // initialize some movies and actors/actresses
    // create interconnections according to a logic that will yield a negative match
  def createNegative(int n) {
    val movies = newArrayList()
    (5 .. 9).forEach[movies += createMovie(10 * n + it)]

    val a =   createActor("a" + (10 * n + 5))
    val b =   createActor("a" + (10 * n + 6))
```

```
    val c = createActress("a" + (10 * n + 7))
    val d = createActress("a" + (10 * n + 8))
    val e = createActress("a" + (10 * n + 9))

    val actors =              #[a, b, c, d, e]
    movies.get(0).persons += #[a, b]
    movies.get(1).persons += #[a, b, c]
    movies.get(2).persons += #[   b, c, d]
    movies.get(3).persons += #[      c, d, e]
    movies.get(4).persons += #[         d, e]

    r.contents += actors
    r.contents += movies
  }

  // create a movie with the given rating
  def createMovie(int rating) {
    val movie = createMovie
    movie.rating = rating
    movie
  }

  // create an actor with the given name
  def createActor(String name) {
    val actor = createActor
    actor.name = name
    actor
  }

  // create an actress with the given name
  def createActress(String name) {
    val actress = createActress
    actress.name = name
    actress
  }

}
```

A.2.2 Transformation Code

```
/**
 * This class implements the transformation logic.
 */
class Transformation {

  /**
   * Initialize the transformation processor on a resource.
   * The runtime of the transformation steps are logged.
   * @param r The target resource of the transformation.
   * @param bmr The benchmark logger.
   */
  new (Resource r, BenchmarkResults bmr) {
    this.r = r;
    this.bmr = bmr;
    this.root = r.contents.get(0) as Root
  }

  // to store the benchmark results
  protected val BenchmarkResults bmr;
  // to store the model
  protected Resource r

  ////// Resources Management
  protected val Root root;
  /**
```

```
26    * Helper function to add elements to the target resource.
27    * @param
28    */
29   def addElementToResource(ContainedElement containedElement) {
30     root.children.add(containedElement)
31   }
32   def addElementsToResource(Collection<? extends ContainedElement> containedElements) {
33     root.children.addAll(containedElements)
34   }
35   def getElementsFromResource() {
36     root.children
37   }
38   ////////////////////////////
39
40   // to help with model manipulation
41   extension MoviesFactory = MoviesFactory.eINSTANCE
42   extension Imdb = Imdb.instance
43
44   // create couples
45   public def createCouples() {
46     val engine = AdvancedIncQueryEngine.createUnmanagedEngine(r)
47     val coupleMatcher = engine.personsToCouple
48     val commonMoviesMatcher = engine.commonMoviesToCouple
49     val personNameMatcher = engine.personName
50
51     val newCouples = new LinkedList<Couple>
52     coupleMatcher.forEachMatch [
53       val couple = createCouple()
54       val p1 = personNameMatcher.getAllValuesOfp(p1name).head
55       val p2 = personNameMatcher.getAllValuesOfp(p2name).head
56       couple.setP1(p1)
57       couple.setP2(p2)
58       val commonMovies = commonMoviesMatcher.getAllValuesOfm(p1name, p2name)
59       couple.commonMovies.addAll(commonMovies)
60
61       newCouples += couple
62     ]
63
64     println("# of couples = " + newCouples.size)
65     engine.dispose
66     addElementsToResource(newCouples);
67   }
68
69   // calculate the top group by rating
70   def topGroupByRating(int size) {
71     println("Top-15 by Average Rating")
72     println("========================")
73     val n = 15;
74
75     val engine = IncQueryEngine.on(r)
76     val coupleWithRatingMatcher = engine.groupSize
77     val rankedCouples = coupleWithRatingMatcher.getAllValuesOfgroup(size).sort(
78       new GroupAVGComparator)
79
80     printCouples(n, rankedCouples)
81   }
82
83   // calculate the top group by common movies
84   def topGroupByCommonMovies(int size) {
85     println("Top-15 by Number of Common Movies")
86     println("=================================")
87
88     val n = 15;
89     val engine = IncQueryEngine.on(r)
90     val coupleWithRatingMatcher = engine.groupSize
```

```
91
92      val rankedCouples = coupleWithRatingMatcher.getAllValuesOfgroup(size).sort(
93        new GroupSizeComparator
94      )
95      printCouples(n, rankedCouples)
96    }
97
98    // pretty-print couples
99    def printCouples(int n, List<Group> rankedCouples) {
100     (0 .. n - 1).forEach [
101       if(it < rankedCouples.size) {
102         val c = rankedCouples.get(it);
103         println(c.printGroup(it))
104       }
105     ]
106   }
107
108   // pretty-print groups
109   def printGroup(Group group, int lineNumber) {
110     if(group instanceof Couple) {
111       val couple = group as Couple
112       return '''«lineNumber». Couple avgRating «group.avgRating», «group.commonMovies.
          size» movies («couple.p1.name»; «couple.p2.name»)'''
113     }
114     else {
115       val clique = group as Clique
116       return '''«lineNumber». «clique.persons.size»-Clique avgRating «group.avgRating», «
          group.commonMovies.size» movies («
117         FOR person : clique.persons SEPARATOR ", "»«person.name»«ENDFOR»)'''
118     }
119   }
120
121   // calculate average ratings
122   def calculateAvgRatings() {
123     getElementsFromResource.filter(typeof(Group)).forEach[x|calculateAvgRating(x.
          commonMovies, x)]
124   }
125
126   // calculate average rating
127   protected def calculateAvgRating(Collection<Movie> commonMovies, Group group) {
128     var sumRating = 0.0
129
130     for (m : commonMovies) {
131       sumRating = sumRating + m.rating
132     }
133     val n = commonMovies.size
134     group.avgRating = sumRating / n
135   }
136
137   // create cliques
138   public def createCliques(int cliques) {
139     val engine = AdvancedIncQueryEngine.createUnmanagedEngine(r)
140     val personMatcher = getPersonName(engine)
141     var Collection<Clique> newCliques
142
143     if(cliques == 3) {
144       val clique3 = getPersonsTo3Clique(engine)
145
146       newCliques = clique3.allMatches.map[x|generateClique(
147         personMatcher.getOneArbitraryMatch(null,x.p1).p,
148         personMatcher.getOneArbitraryMatch(null,x.p2).p,
149         personMatcher.getOneArbitraryMatch(null,x.p3).p)].toList;
150     }
151     else if(cliques == 4) {
152       val clique4 = getPersonsTo4Clique(engine)
```

```
153
154       newCliques = clique4.allMatches.map[x|generateClique(
155         personMatcher.getOneArbitraryMatch(null,x.p1).p,
156         personMatcher.getOneArbitraryMatch(null,x.p2).p,
157         personMatcher.getOneArbitraryMatch(null,x.p3).p,
158         personMatcher.getOneArbitraryMatch(null,x.p4).p)].toList;
159     }
160     else if(cliques == 5) {
161       val clique5 = getPersonsTo5Clique(engine)
162       newCliques = clique5.allMatches.map[x|generateClique(
163         personMatcher.getOneArbitraryMatch(null,x.p1).p,
164         personMatcher.getOneArbitraryMatch(null,x.p2).p,
165         personMatcher.getOneArbitraryMatch(null,x.p3).p,
166         personMatcher.getOneArbitraryMatch(null,x.p4).p,
167         personMatcher.getOneArbitraryMatch(null,x.p5).p)].toList;
168     }
169
170     println("# of "+cliques+"-cliques = " + newCliques.size)
171
172     engine.dispose
173     newCliques.forEach[x|x.commonMovies.addAll(x.collectCommonMovies)]
174     addElementsToResource(newCliques);
175   }
176
177   // generate cliques
178   protected def generateClique(Person... persons) {
179     val c = createClique
180     c.persons += persons
181     return c
182   }
183
184   // collect common movies
185   protected def collectCommonMovies(Clique clique) {
186     var Set<Movie> commonMovies = null;
187     for(personMovies : clique.persons.map[movies]) {
188       if(commonMovies == null) {
189         commonMovies = personMovies.toSet;
190       }
191       else {
192         commonMovies.retainAll(personMovies)
193       }
194     }
195     return commonMovies
196   }
197 }
```

A.2.3 General Clique Patterns

The resulting expression for condition (ii) is the following: $(\nexists g') : ((\nexists p_0 \in g_0) : p_0 \notin g') \wedge (p \in g')$. This is equivalent to the following EMF-INCQUERY pattern:

```
1  /** Group g0 is a subset of Group gx. */
2  pattern subsetOfGroup(g0 : Group, gx : Group) {
3    neg find notSubsetOfGroup(p0, g0, gx);
4  }
5
6  /** This pattern returns is a helper for the subsetOfGroup pattern. */
7  pattern notSubsetOfGroup(p0 : Person, g0 : Group, gx : Group) {
8    find memberOfGroup(p0, g0);
9    neg find memberOfGroup(p0, gx);
10 }
11
12 /** Person p is a member of Group g. A Group is either a Couple or a Clique. */
13 pattern memberOfGroup(p, g) {
14   Couple.p1(g, p);
```

```
} or {
  Couple.p2(g, p);
} or {
  Clique.persons(g, p);
}
```

Based on the subsetOfGroup pattern, we may implement the nextClique pattern like follows:

```
/** the nextCliques pattern */
pattern nextCliques(g : Group, p : Person)  {
  neg find alphabeticallyLaterMemberOfGroup(g, p);
  n == count find commonMovieOfGroupAndPerson(g, p, m);
  check(n >= 3);
  neg find union(g, p);
}

/** p is a member of g for which another alphabetically previous member exists */
pattern alphabeticallyLaterMemberOfGroup(g : Group, p : Person)  {
  find memberOfGroup(m, g);
  Person.name(p, pName);
  Person.name(m, mName);
  check(mName >= pName);
}

/** m is a common movie of g and p */
pattern commonMovieOfGroupAndPerson(g, p, m) {
  find commonMoviesOfGroup(g, m);
  Person.movies(p, m);
}

/** m is a common movie of g */
pattern commonMoviesOfGroup(g, m) {
  Group.commonMovies(g, m);
}

/** p is in g0 */
pattern union(g0, p) {
  find memberOfGroup(p, gx);
  find subsetOfGroup(g0, gx);
}
```

A.2.4 Comparator Code for Top-15

```
class GroupSizeComparator implements Comparator<Group>{

  override compare(Group arg0, Group arg1) {
    if (arg0.commonMovies.size < arg1.commonMovies.size) {return 1}
    else if (arg0.commonMovies.size == arg1.commonMovies.size) {return 0}
    else return -1;
  }
}

class GroupAVGComparator implements Comparator<Group>{

  override compare(Group arg0, Group arg1) {
    if(arg0.avgRating<arg1.avgRating) {return 1;}
    else if (arg0.avgRating == arg1.avgRating) {return 0;}
    else return -1;
  }
}
```