Java Refactoring Case: a VIATRA Solution*

Dániel Stein Gábor Szárnyas Ábel Hegedüs István Ráth

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

daniel.stein@inf.mit.bme.hu, {szarnyas, abel.hegedus, rath}@mit.bme.hu

This paper presents a solution for the Java Refactoring Case of the Transformation Tool Contest 2015. The solution utilises Eclipse JDT for creating the program graph; while EMF-INCQUERY, VIATRA and the Xtend programming language are used for defining and performing the model transformations.

1 Introduction

The use of automated model transformations is a key factor in modern model-driven system engineering. Model transformations allow to query, derive and manipulate large industrial models, including models based on existing systems, e.g. source code models created with reverse engineering techniques. Since such transformations are frequently integrated to modeling environments, they need to feature both high performance and a concise programming interface to support software engineers.

Refactoring operations are often used in software engineering to improve the readability, maintainability of existing source code without altering the behaviour of the software.

1.1 EMF-INCQUERY

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, 4] framework is to provide a declarative way to define queries over EMF models. EMF-INCQUERY extended the pattern language of VIATRA2 with new features (including transitive closure, role navigation, match count) and tailored it to EMF models [3]. EMF-INCQUERY is developed with a focus on *incremental query evaluation*.

1.2 VIATRA

The VIATRA framework supports the development of model transformations with specific focus on event-driven, reactive transformations [7]. Building upon the incremental query support of the EMF-INCQUERY project, VIATRA offers a language to define transformations and a reactive transformation engine to execute certain transformations upon changes in the underlying model.

The VIATRA project provides:

- An internal DSL over the Xtend [8] language to specify both batch and event-driven, reactive transformations.

*This work was partially supported by the MONDO (EU ICT-611125) project.

Submitted to:
TTC 2015

- A complex event-processing engine over EMF models to specify reactions upon detecting complex sequences of events.
- A rule-based design space exploration framework to explore design candidates as models satisfying multiple criteria.
- A model obfuscator to remove sensitive information from a confidential model (e.g. to create bug reports).

The current VIATRA project is a full rewrite of the previous VIATRA2 framework, now with full compatibility and support for EMF models. The history of the VIATRA family is described in [6].

2 Case Description

The goal of the Java Refactoring Case [9] is to use model transformation tools to perform refactoring operations on Java source code. The main challenges of the case are the following:

1. Transform the *Java source code* to a *program graph* (PG). The source code and the program graph must be synchronised using a bidirectional transformation.
2. Perform the refactoring transformation on the program graph.

The source code is defined in a restricted sub-language of Java 1.4. The EMF metamodel of the PG is provided in the case description. The case considers two refactoring operations:

- Pull Up Method
- Create Superclass

3 Implementation

The source code of the solution is available as an open-source project.[1]

The solution was developed in the Eclipse IDE. For setting up the development environment, please refer to the readme file. The projects are not tied to the Eclipse environment and can be compiled with the Apache Maven [1] build automation tool. This offers a number of benefits, including easy portability and the possibility of continuous integration.

The solution is written in Java 8 and Xtend [8].

ide kellene egy abra sorszamokkal, amikre lehet hivatkozni a kovetkezokben

3.1 Parsing the Source Code to the ASG

The solution uses the parser of Eclipse Java Development Tools [5], which is also used in the Eclipse Java IDE. The parser generates the Abstract Syntax Graph (ASG) from the provided source code files.

[1] https://github.com/FTSRG/java-refactoring-ttc-viatra

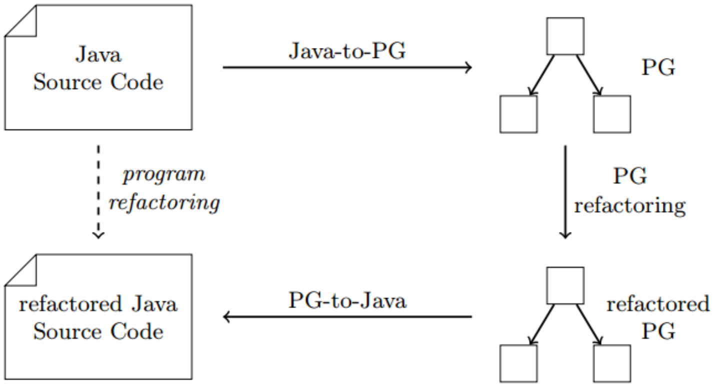

Figure 1: Caption.

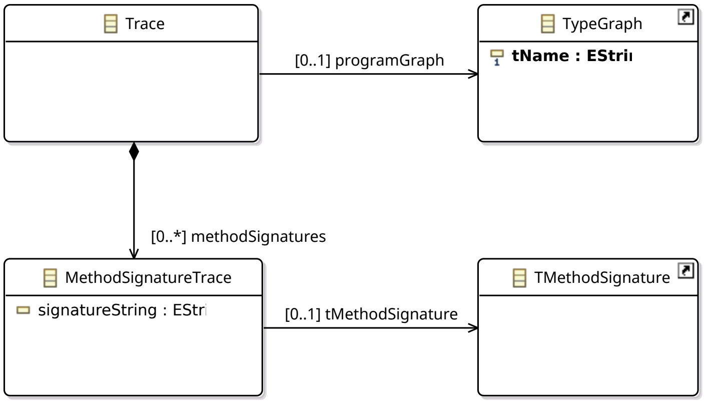

Figure 2: Metamodel of the trace model.

3.2 Synchronising the ASG and the PG

As VIATRA does not support bidirectional transformations, the JDT ASG–PG transformation was implemented as two separate transformations:

JDT ASG to PG The JDT model is traversed using the Visitor pattern.

PG to JDT ASG VIATRA rules are used to detect the changes and execute the appropriate actions to keep the ASG in sync.

3.3 Transforming the PG

The refactoring operations are implemented as model transformations on the PG. Each model transformation is defined in VIATRA: the LHS is defined with an EMF-INCQUERY pattern and the RHS is defined with an imperative Xtend code.

3.4 Transforming the ASG to Source Code

The ASG is transformed using JDT's CompilationUnit.rewrite() method which converts the changes of the abstract syntax graph to a set of text manipulation operations (TextEdit class).

4 Evaluation

The benchmark were conducted on a 64-bit Arch Linux virtual machine running on SHARE.[2]

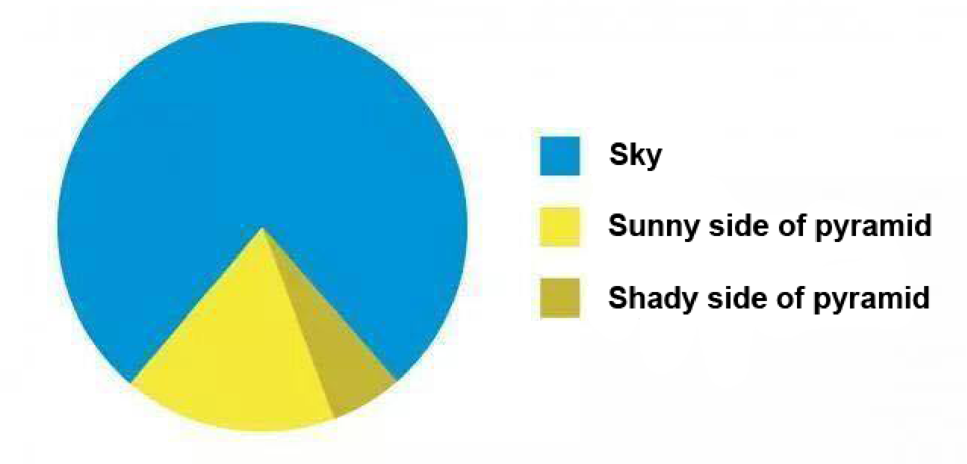

Figure 3: Benchmark results.

5 Summary

The paper presented a solution for the Java Refactoring case of the 2015 Transformation Tool Contest. The solution addresses both challenges (bidirectional synchrosiation and program refactoring) and both refactoring operations (pull up method, create superclass) defined in the case.

The framework is flexible to allow the user to define new refactoring operations, e.g. Extract Class or Pull Up Field.

References

[1] Apache.org: *Maven*. `http://maven.apache.org`.

[2] Gábor Bergmann, Ákos Horváth, István Ráth, Dániel Varró, András Balogh, Zoltán Balogh & András Ökrös (2010): *Incremental Evaluation of Model Queries over EMF Models*. In: *Model Driven Engineering Languages and Systems, 13th International Conference, MODELS2010*, Springer, Springer, doi:http://dx.doi.org/10.1007/978-3-642-16145-2_6. Acceptance rate: 21%.

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

[4] Eclipse.org: *EMF-IncQuery*. `http://eclipse.org/incquery/`.

[5] Eclipse.org: *Java Development Tools (JDT)*. `https://eclipse.org/jdt/`.

[6] Eclipse.org: *VIATRA History*. `https://wiki.eclipse.org/VIATRA/History`.

[7] Eclipse.org: *VIATRA Project*. `https://www.eclipse.org/viatra/`.

[8] Eclipse.org: *Xtend – Modernized Java*. `https://www.eclipse.org/xtend/`.

[2] `https://is.ieis.tue.nl/staff/pvgorp/share/`

[9] Géza Kulcsár, Sven Peldszus & Malte Lochau (2015): *The Java Refactoring Case*. In: *8th Transformation Tool Contest (TTC 2015)*.

A Appendix

```
/**
 * 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;
  /**
   * Helper function to add elements to the target resource.
   * @param
   */
  def addElementToResource(ContainedElement containedElement) {
    root.children.add(containedElement)
  }
  def addElementsToResource(Collection<? extends ContainedElement> containedElements) {
    root.children.addAll(containedElements)
  }
  def getElementsFromResource() {
    root.children
  }
  ////////////////////////////

  // to help with model manipulation
  extension MoviesFactory = MoviesFactory.eINSTANCE
  extension Imdb = Imdb.instance

  // create couples
  public def createCouples() {
    val engine = AdvancedIncQueryEngine.createUnmanagedEngine(r)
    val coupleMatcher = engine.personsToCouple
    val commonMoviesMatcher = engine.commonMoviesToCouple
    val personNameMatcher = engine.personName

    val newCouples = new LinkedList<Couple>
    coupleMatcher.forEachMatch [
      val couple = createCouple()
      val p1 = personNameMatcher.getAllValuesOfp(p1name).head
      val p2 = personNameMatcher.getAllValuesOfp(p2name).head
      couple.setP1(p1)
      couple.setP2(p2)
      val commonMovies = commonMoviesMatcher.getAllValuesOfm(p1name, p2name)
      couple.commonMovies.addAll(commonMovies)

      newCouples += couple
    ]

    println("# of couples = " + newCouples.size)
    engine.dispose
    addElementsToResource(newCouples);
  }

  // calculate the top group by rating
  def topGroupByRating(int size) {
    println("Top-15 by Average Rating")
    println("========================")
    val n = 15;

    val engine = IncQueryEngine.on(r)
    val coupleWithRatingMatcher = engine.groupSize
    val rankedCouples = coupleWithRatingMatcher.getAllValuesOfgroup(size).sort(
      new GroupAVGComparator)

    printCouples(n, rankedCouples)
  }

  // calculate the top group by common movies
  def topGroupByCommonMovies(int size) {
    println("Top-15 by Number of Common Movies")
    println("=================================")

    val n = 15;
    val engine = IncQueryEngine.on(r)
    val coupleWithRatingMatcher = engine.groupSize

    val rankedCouples = coupleWithRatingMatcher.getAllValuesOfgroup(size).sort(
      new GroupSizeComparator
    )
    printCouples(n, rankedCouples)
  }

  // pretty-print couples
  def printCouples(int n, List<Group> rankedCouples) {
    (0 .. n - 1).forEach [
      if(it < rankedCouples.size) {
        val c = rankedCouples.get(it);
        println(c.printGroup(it))
      }
    ]
  }

  // pretty-print groups
  def printGroup(Group group, int lineNumber) {
    if(group instanceof Couple) {
      val couple = group as Couple
      return '''«lineNumber». Couple avgRating «group.avgRating», «group.commonMovies.size» movies («couple.p1.name»; «couple.p2.name»)'''
    }
    else {
      val clique = group as Clique
      return '''«lineNumber». «clique.persons.size»-Clique avgRating «group.avgRating», «group.commonMovies.size» movies («
        FOR person : clique.persons SEPARATOR ", "»«person.name»«ENDFOR»)'''
    }
  }

  // calculate average ratings
  def calculateAvgRatings() {
    getElementsFromResource.filter(typeof(Group)).forEach[x|calculateAvgRating(x.commonMovies, x)]
  }

```

```
// calculate average rating
protected def calculateAvgRating(Collection<Movie> commonMovies, Group group) {
  var sumRating = 0.0

  for (m : commonMovies) {
    sumRating = sumRating + m.rating
  }
  val n = commonMovies.size
  group.avgRating = sumRating / n
}

// create cliques
public def createCliques(int cliques) {
  val engine = AdvancedIncQueryEngine.createUnmanagedEngine(r)
  val personMatcher = getPersonName(engine)
  var Collection<Clique> newCliques

  if(cliques == 3) {
    val clique3 = getPersonsTo3Clique(engine)

    newCliques = clique3.allMatches.map[x|generateClique(
      personMatcher.getOneArbitraryMatch(null,x.p1).p,
      personMatcher.getOneArbitraryMatch(null,x.p2).p,
      personMatcher.getOneArbitraryMatch(null,x.p3).p)].toList;
  }
  else if(cliques == 4) {
    val clique4 = getPersonsTo4Clique(engine)

    newCliques = clique4.allMatches.map[x|generateClique(
      personMatcher.getOneArbitraryMatch(null,x.p1).p,
      personMatcher.getOneArbitraryMatch(null,x.p2).p,
      personMatcher.getOneArbitraryMatch(null,x.p3).p,
      personMatcher.getOneArbitraryMatch(null,x.p4).p)].toList;
  }
  else if(cliques == 5) {
    val clique5 = getPersonsTo5Clique(engine)
    newCliques = clique5.allMatches.map[x|generateClique(
      personMatcher.getOneArbitraryMatch(null,x.p1).p,
      personMatcher.getOneArbitraryMatch(null,x.p2).p,
      personMatcher.getOneArbitraryMatch(null,x.p3).p,
      personMatcher.getOneArbitraryMatch(null,x.p4).p,
      personMatcher.getOneArbitraryMatch(null,x.p5).p)].toList;
  }

  println("# of "+cliques+"-cliques = " + newCliques.size)

  engine.dispose
  newCliques.forEach[x|x.commonMovies.addAll(x.collectCommonMovies)]
  addElementsToResource(newCliques);
}

// generate cliques
protected def generateClique(Person... persons) {
  val c = createClique
  c.persons += persons
  return c
}

// collect common movies
protected def collectCommonMovies(Clique clique) {
  var Set<Movie> commonMovies = null;
  for(personMovies : clique.persons.map[movies]) {
    if(commonMovies == null) {
      commonMovies = personMovies.toSet;
    }
```

```
      else {
        commonMovies.retainAll(personMovies)
      }
    }
    return commonMovies
  }
}
```

```
/** Group g0 is a subset of Group gx. */
pattern subsetOfGroup(g0 : Group, gx : Group) {
  neg find notSubsetOfGroup(p0, g0, gx);
}

/** This pattern returns is a helper for the subsetOfGroup pattern. */
pattern notSubsetOfGroup(p0 : Person, g0 : Group, gx : Group) {
  find memberOfGroup(p0, g0);
  neg find memberOfGroup(p0, gx);
}

/** Person p is a member of Group g. A Group is either a Couple or a Clique. */
pattern memberOfGroup(p, g) {
  Couple.p1(g, p);
} or {
  Couple.p2(g, p);
} or {
  Clique.persons(g, p);
}
```