The Edapt Solution for the Hello World Case

Markus Herrmannsdoerfer

Institut für Informatik, Technische Universität München
Boltzmannstr. 3, 85748 Garching b. München, Germany
herrmama@in.tum.de

Abstract. This paper gives an overview of the Edapt solution to the hello world case of the Transformation Tool Contest 2011.

1 Edapt in a Nutshell

Edapt[1] is a transformation tool tailored for the migration of models in response to metamodel adaptation.

Modeling the Coupled Evolution. As depicted by Figure 1, Edapt specifies the metamodel adaptation as a sequence of operations in an explicit history model. The operations can be enriched with instructions for model migration to form so-called coupled operations. Edapt provides two kinds of coupled operations according to the automatability of the model migration [1]: reusable and custom coupled operations.

Reuse of recurring migration specifications allows to reduce the effort associated with building a model migration [2]. Edapt thus provides *reusable coupled operations* which make metamodel adaptation and model migration independent of the specific metamodel through parameters and constraints restricting the applicability of the operation. An example for a reusable coupled operation is *Enumeration to Sub Classes* which replaces an enumeration attribute with subclasses for each literal of the enumeration. Currently, Edapt comes with a library of over 60 reusable coupled operations [3]. By means of studying real-life metamodel histories, we have shown that, in practice, most of the coupled evolution can be covered by reusable coupled operations [2, 4].

Migration specifications can become so specific to a certain metamodel that reuse makes no sense [2]. To express these complex migrations, Edapt allows the user to define a custom coupled operation by manually encoding a model migration for a metamodel adaptation in a Turing-complete language [5]. By softening the conformance of the model to the metamodel within a coupled operation, both metamodel adaptation and model migration can be specified as in-place transformations, requiring only to specify the difference. A transaction mechanism ensures conformance at the boundaries of the coupled operation.

[1] http://www.eclipse.org/edapt

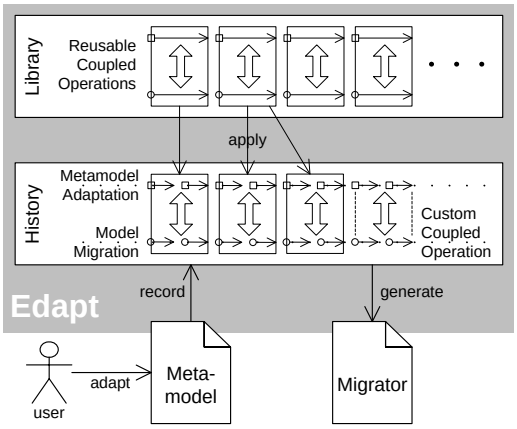

Fig. 1. Overview of Edapt

Recording the Coupled Evolution. To not lose the intention behind the metamodel adaptation, Edapt is intended to be used already when adapting the metamodel. Therefore, Edapt's user interface which is depicted in Figure 2 is directly integrated into the existing EMF *metamodel editor*. The user interface provides access to the *history model* in which Edapt records the sequence of coupled operations. An initial history can be created for an existing metamodel by invoking *Create History* in the *operation browser* which also allows the user to *Release* the metamodel.

The user can adapt the metamodel by applying reusable coupled operations through the *operation browser*. The operation browser allows to set the parameters of a reusable coupled operation, and gives feedback on its applicability based on the constraints. When a reusable coupled operation is executed, its application is automatically recorded in the history model. Figure 2 shows the operation Sub Classes to Enumeration being selected in the operation browser and recorded to the history model.

The user needs to perform a custom coupled operation only, in case no reusable coupled operation is available for the change at hand. First, the metamodel is directly adapted in the metamodel editor, in response to which the changes are automatically recorded in the history. A migration can later be attached to the sequence of metamodel changes. Figure 2 shows the *migration editor* to encode the custom migration in Java.

2 Hello World Case

Table 1 gives an overview of the tasks and their solution. The tasks are given a name and are grouped similar to the case description. For the solutions to these tasks, we indicate whether custom or reusable coupled operation are required.

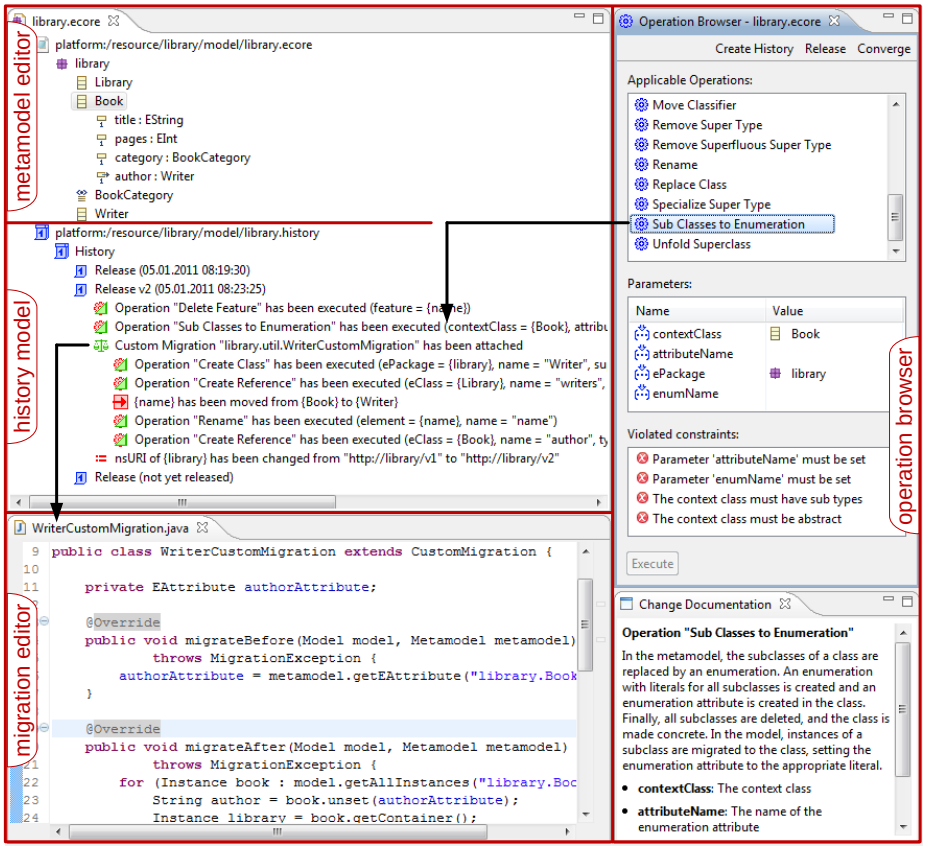

Fig. 2. User interface of Edapt

Figure 3 shows how the history model looks like for all tasks of this case that are solved using custom coupled operations. In this case, the custom coupled operation always consists of a custom migration which is attached to an empty metamodel adaptation. The custom migration is implemented as a Java class that inherits from a special super class.

The complete solutions are available in the appendix and through the repository of the Eclipse Edapt project[2]. In the following, we briefly explain the main characteristics of the solutions for the different tasks.

2.1 Hello World! Figure 3 shows how the constant transformation is implemented using the migration language provided by Edapt. Since Edapt is a migration tool, the transformation is always performed in-place. To store the

[2] `http://dev.eclipse.org/svnroot/modeling/org.eclipse.emft.edapt/trunk/examples/ttc_helloworld`

Table 1. Tasks of the Hello World case

sec.	task	coupled operations
2.1	constant transformation	custom
	constant transformation to create model with references	custom
	model-to-text-transformation	custom
2.2	count the number of nodes	custom
	count the number of looping edges	custom
	count the number of isolated nodes	custom
	count the number of circles consisting of three nodes	custom
	count the number of dangling edges	custom
2.3	reverse edges	custom
2.4	simple migration	reusable
	topology-changing migration	reusable
2.5	delete node with name and incident edges	custom
2.6	insert transitive edges	custom

result at another location, we use helper methods that are provided by the superclass HelloWorldCustomMigration. The task to perform an extended constant transformation is solved in a similar way. For the model-to-text-transformation, we also have to include the result metamodel in the history and provide helper methods to store instances of the classes defined by this metamodel.

2.2 Count Matches with certain Properties. All the count tasks require the result metamodel to be part of the history and a helper method to store the integer result which is provided by the superclass HelloWorldCustomMigration. The solutions of the tasks to count the number of nodes, looping edges, isolated nodes and dangling edges are straightforward. For the solution of the task to count the number of circles, we implemented the helper method getReachable to get the nodes reachable from a node through directed edges. For this helper method, we used the function getInverse to navigate the inverse of an association.

2.3 Reverse Edges. The solution to this task is straightforward, since we only have to exchange src and trg of each Edge.

2.4 Simple Migration. Figure 4 shows the history model for the simple migration which can be solved completely using already available reusable coupled operations. Note that the custom coupled operation is only necessary to store the result of a transformation in a different file. First, the common super class GraphComponent is created for classes Node and Edge. Then, the associations nodes and edges are united into the association gcs. Finally, the attribute name is pulled up from class Node to GraphComponent and renamed to text.

Figure 5 shows the history model for the topology-changing migration which can also be solved using reusable coupled operations. The operation ClassToAs-

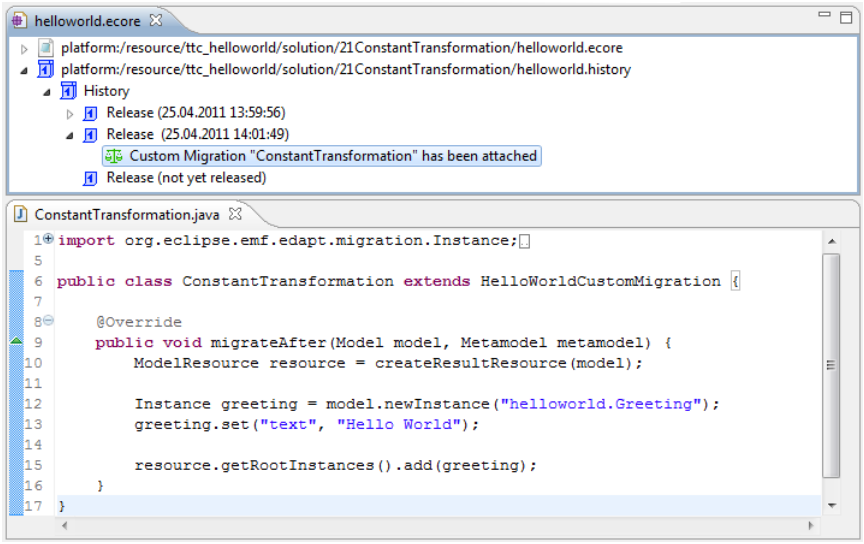

Fig. 3. History model with a custom migration

sociation is applied to replace the class Edge by the association linksTo. Finally, also a Rename of an attribute is required to complete the migration.

2.5 Delete Node with Specific Name and its Incident Edges. This task can also be implemented quite easily, since Edapt provides a method to delete instances of classes. To also delete all incident edges, we can again use the method getInverse to navigate to the edges which have the node as source or target.

2.6 Insert Transitive Edges. The solution to this task is a little bit more involved. To not let the newly created edges influence the result, we first determine the pairs of nodes for which edges need to be created. Here, we can again rely on our helper method getReachable to obtain the nodes reachable from a node. Finally, we create the edges for these nodes.

3 Conclusion

As Edapt is a transformation tool targeted at model migration, it clearly shows its strengths in the migration tasks. The migration tasks can be solved by applying only reusable coupled operations. Thereby, not a single line of custom migration code needs to be written.

Although a degenerated case, the other tasks can be solved by attaching custom migrations to an empty metamodel adaptation. The custom migrations are implemented in Java based on the API provided by Edapt to navigate and modify models. Even though the Java solutions are quite concise and clear, a

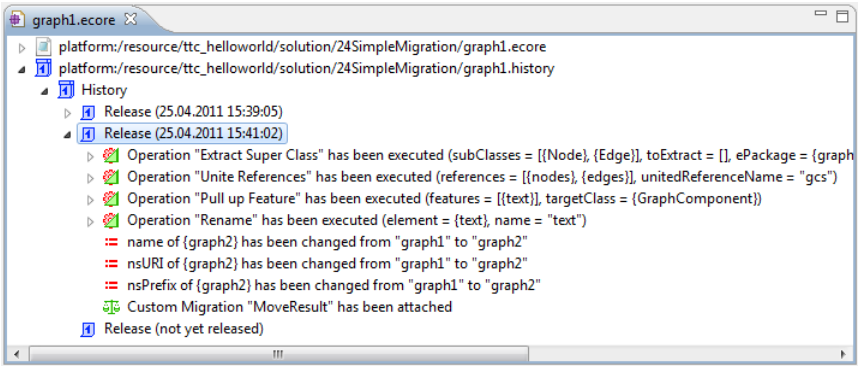

Fig. 4. History model for the simple migration

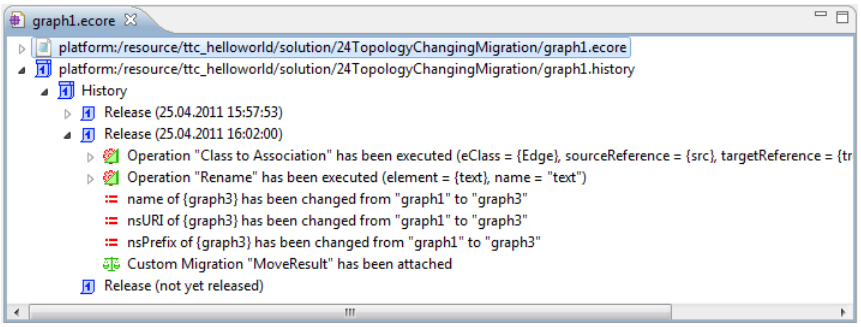

Fig. 5. History model for the topology-changing migration

specialized DSL could further improve conciseness and clarity. However, we can rely on Java's abstraction mechanisms to organize the implementation, and on the strong Java tooling to implement, refactor and debug the solution.

Acknowledgments. This work was funded by the German Federal Ministry of Education and Research (BMBF), grants "SPES2020, 01IS08045A" and "Quamoco, 01IS08023B".

References

1. Herrmannsdoerfer, M., Benz, S., Juergens, E.: COPE - automating coupled evolution of metamodels and models. In: ECOOP 2009 - Object-Oriented Programming. Volume 5653 of Lecture Notes in Computer Science., Springer Berlin / Heidelberg (2009) 52–76
2. Herrmannsdoerfer, M., Benz, S., Juergens, E.: Automatability of coupled evolution of metamodels and models in practice. In Czarnecki, K., Ober, I., Bruel, J.M., Uhl, A., Völter, M., eds.: Model Driven Engineering Languages and Systems (MODELS 2008). Volume 5301/2008 of Lecture Notes in Computer Science., Springer Berlin / Heidelberg (October 2008) 645–659

3. Herrmannsdoerfer, M., Vermolen, S., Wachsmuth, G.: An extensive catalog of operators for the coupled evolution of metamodels and models. In: Software Language Engineering. (2010)
4. Herrmannsdoerfer, M., Ratiu, D., Wachsmuth, G.: Language evolution in practice: The history of GMF. In: Software Language Engineering. Volume 5969 of Lecture Notes in Computer Science., Springer Berlin / Heidelberg (2009) 3–22
5. Herrmannsdoerfer, M., Benz, S., Juergens, E.: COPE: A language for the coupled evolution of metamodels and models. In: 1st International Workshop on Model Co-Evolution and Consistency Management. (2008)

A Solution

2.1-2.6 Base Class for Custom Migrations for the Hello World Case

```
1  import org.eclipse.emf.common.util.URI;
2  import org.eclipse.emf.edapt.migration.CustomMigration;
3  import org.eclipse.emf.edapt.migration.Instance;
4  import org.eclipse.emf.edapt.migration.Model;
5  import org.eclipse.emf.edapt.migration.ModelResource;
6
7  public abstract class HelloWorldCustomMigration extends CustomMigration {
8
9      /** Create the resource in which to store the result of the transformation. */
10     protected ModelResource createResultResource(Model model) {
11         URI resultUri = getResultURI(model);
12         return model.newResource(resultUri);
13     }
14
15     /** Change the location in which the model is stored to the result location.
           */
16     protected void moveResult(Model model) {
17         model.getResources().get(0).setUri(getResultURI(model));
18     }
19
20     /** Get the location in which the result should be stored. */
21     private URI getResultURI(Model model) {
22         URI uri = model.getResources().get(0).getUri();
23         URI resultUri = uri
24                 .trimSegments(1)
25                 .appendSegment(uri.trimFileExtension().lastSegment() + "result")
26                 .appendFileExtension(uri.fileExtension());
27         return resultUri;
28     }
29
30     /** Create the result resource and save a result of type integer in it. */
31     protected void saveResult(Model model, int i) {
32         ModelResource resource = createResultResource(model);
33         Instance instance = model.newInstance("result.IntResult");
34         instance.set("result", i);
35         resource.getRootInstances().add(instance);
36     }
37
38     /** Save a result of type String in a resource. */
39     protected void saveResult(ModelResource resource, String s) {
40         Instance instance = resource.getModel().newInstance(
41                 "result.StringResult");
42         instance.set("result", s);
43         resource.getRootInstances().add(instance);
44     }
45 }
```

2.1 Constant Transformation

```
1  import org.eclipse.emf.edapt.migration.Instance;
2  import org.eclipse.emf.edapt.migration.Metamodel;
3  import org.eclipse.emf.edapt.migration.Model;
4  import org.eclipse.emf.edapt.migration.ModelResource;
5
6  public class ConstantTransformation extends HelloWorldCustomMigration {
7
8      @Override
9      public void migrateAfter(Model model, Metamodel metamodel) {
10         ModelResource resource = createResultResource(model);
11
12         Instance greeting = model.newInstance("helloworld.Greeting");
13         greeting.set("text", "Hello World");
14
15         resource.getRootInstances().add(greeting);
16     }
17 }
```

2.1 Constant Transformation to create Model with References

```
1  import org.eclipse.emf.edapt.migration.Instance;
2  import org.eclipse.emf.edapt.migration.Metamodel;
3  import org.eclipse.emf.edapt.migration.Model;
4  import org.eclipse.emf.edapt.migration.ModelResource;
5
6  public class ConstantTransformationReferences extends HelloWorldCustomMigration {
7
8      @Override
9      public void migrateAfter(Model model, Metamodel metamodel) {
10         ModelResource resource = createResultResource(model);
11
12         metamodel.setDefaultPackage("helloworldext");
13
14         Instance greeting = model.newInstance("Greeting");
15
16         Instance message = model.newInstance("GreetingMessage");
17         message.set("text", "Hello");
18         greeting.set("greetingMessage", message);
19
20         Instance person = model.newInstance("Person");
21         greeting.set("person", person);
22         person.set("name", "TTC Participants");
23
24         resource.getRootInstances().add(greeting);
25     }
26 }
```

2.1 Model-to-Text-Transformation

```
1  import org.eclipse.emf.edapt.migration.Instance;
2  import org.eclipse.emf.edapt.migration.Metamodel;
3  import org.eclipse.emf.edapt.migration.Model;
4  import org.eclipse.emf.edapt.migration.ModelResource;
5
6  public class ModelToTextTransformation extends HelloWorldCustomMigration {
7
8      @Override
9      public void migrateBefore(Model model, Metamodel metamodel) {
10         ModelResource resource = createResultResource(model);
11         metamodel.setDefaultPackage("helloworldext");
12         for (Instance greeting : model.getAllInstances("Greeting")) {
```

```
            String greetingText = greeting.getLink("greetingMessage").get(
                    "text");
            Object personName = greeting.getLink("person").get("name");
            String text = greetingText + "␣" + personName + "!";
            saveResult(resource, text);
        }
    }
}
```

2.2 Base Class for Custom Migrations for the Graph1 Metamodel

```
import java.util.ArrayList;
import java.util.List;

import org.eclipse.emf.edapt.migration.Instance;

public abstract class Graph1CustomMigration extends HelloWorldCustomMigration {

    /** Get the nodes reachable from a nodes through directed edges. */
    protected List<Instance> getReachable(Instance node) {
        List<Instance> reachable = new ArrayList<Instance>();
        for (Instance edge : node.getInverse("graph1.Edge.src")) {
            reachable.add(edge.getLink("trg"));
        }
        return reachable;
    }
}
```

2.2 Count the Number of Nodes

```
import org.eclipse.emf.edapt.migration.Metamodel;
import org.eclipse.emf.edapt.migration.Model;

public class CountNodes extends HelloWorldCustomMigration {

    @Override
    public void migrateBefore(Model model, Metamodel metamodel) {
        int nodes = model.getAllInstances("graph1.Node").size();
        saveResult(model, nodes);
    }
}
```

2.2 Count the number of looping Edges

```
import org.eclipse.emf.edapt.migration.Instance;
import org.eclipse.emf.edapt.migration.Metamodel;
import org.eclipse.emf.edapt.migration.Model;

public class CountLoopingEdges extends HelloWorldCustomMigration {

    @Override
    public void migrateAfter(Model model, Metamodel metamodel) {
        int loops = 0;
        for (Instance edge : model.getAllInstances("graph1.Edge")) {
            if (edge.get("src") == edge.get("trg")) {
                loops++;
            }
        }
        saveResult(model, loops);
    }
}
```

2.2 Count the number of isolated Nodes

```
import org.eclipse.emf.edapt.migration.Instance;
import org.eclipse.emf.edapt.migration.Metamodel;
import org.eclipse.emf.edapt.migration.Model;

public class CountIsolatedNodes extends HelloWorldCustomMigration {

    @Override
    public void migrateBefore(Model model, Metamodel metamodel) {
        metamodel.setDefaultPackage("graph1");
        int isolated = 0;
        for (Instance node : model.getAllInstances("Node")) {
            if (node.getInverse("Edge.src").isEmpty()
                    && node.getInverse("Edge.trg").isEmpty()) {
                isolated++;
            }
        }
        saveResult(model, isolated);
    }
}
```

2.2 Count the Number of Circles consisting of three Nodes

```
import org.eclipse.emf.edapt.migration.Instance;
import org.eclipse.emf.edapt.migration.Metamodel;
import org.eclipse.emf.edapt.migration.Model;

public class CountCircles extends Graph1CustomMigration {

    @Override
    public void migrateBefore(Model model, Metamodel metamodel) {
        metamodel.setDefaultPackage("graph1");
        int circles = 0;
        for (Instance n1 : model.getAllInstances("Node")) {
            for (Instance n2 : getReachable(n1)) {
                if (n1 != n2) {
                    for (Instance n3 : getReachable(n2)) {
                        if (n2 != n3 && n1 != n3) {
                            if (getReachable(n3).contains(n1)) {
                                circles++;
                            }
                        }
                    }
                }
            }
        }
        saveResult(model, circles);
    }
}
```

2.2 Count the Number of dangling Edges

```
import org.eclipse.emf.edapt.migration.Instance;
import org.eclipse.emf.edapt.migration.Metamodel;
import org.eclipse.emf.edapt.migration.Model;

public class CountDanglingEdges extends HelloWorldCustomMigration {

    @Override
    public void migrateBefore(Model model, Metamodel metamodel) {
        int dangling = 0;
        for (Instance edge : model.getAllInstances("graph1.Edge")) {
```

```
            if (edge.get("src") == null || edge.get("trg") == null) {
                dangling++;
            }
        }
        saveResult(model, dangling);
    }
}
```

2.3 Reverse Edges

```
import org.eclipse.emf.edapt.migration.Instance;
import org.eclipse.emf.edapt.migration.Metamodel;
import org.eclipse.emf.edapt.migration.Model;

public class ReverseEdges extends HelloWorldCustomMigration {

    @Override
    public void migrateBefore(Model model, Metamodel metamodel) {
        moveResult(model);

        for (Instance edge : model.getAllInstances("graph1.Edge")) {
            Instance src = edge.get("src");
            Instance trg = edge.get("trg");
            edge.set("src", trg);
            edge.set("trg", src);
        }
    }
}
```

2.4 Custom Migration to Move the Result

```
import org.eclipse.emf.edapt.migration.Metamodel;
import org.eclipse.emf.edapt.migration.MigrationException;
import org.eclipse.emf.edapt.migration.Model;


public class MoveResult extends HelloWorldCustomMigration {

    @Override
    public void migrateBefore(Model model, Metamodel metamodel)
            throws MigrationException {
        moveResult(model);
    }
}
```

2.4 Simple Migration

see Figure 4

2.4 Topology-Changing Migration

see Figure 5

2.5 Delete Node with name and incident Edges

```java
import org.eclipse.emf.edapt.migration.Instance;
import org.eclipse.emf.edapt.migration.Metamodel;
import org.eclipse.emf.edapt.migration.MigrationException;
import org.eclipse.emf.edapt.migration.Model;

public class DeleteNodeWithName extends HelloWorldCustomMigration {

    @Override
    public void migrateBefore(Model model, Metamodel metamodel)
            throws MigrationException {
        moveResult(model);

        metamodel.setDefaultPackage("graph1");
        for (Instance node : model.getAllInstances("Node")) {
            if ("n1".equals(node.get("name"))) {
                for (Instance edge : node.getInverse("Edge.src")) {
                    model.delete(edge);
                }
                for (Instance edge : node.getInverse("Edge.trg")) {
                    model.delete(edge);
                }
                model.delete(node);
            }
        }
    }
}
```

2.6 Insert Transitive Edges

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import org.eclipse.emf.edapt.migration.Instance;
import org.eclipse.emf.edapt.migration.Metamodel;
import org.eclipse.emf.edapt.migration.Model;

public class InsertTransitiveEdges extends Graph1CustomMigration {

    @Override
    public void migrateBefore(Model model, Metamodel metamodel) {
        moveResult(model);

        metamodel.setDefaultPackage("graph1");
        List<List<Instance>> pairs = new ArrayList<List<Instance>>();
        for (Instance n1 : model.getAllInstances("Node")) {
            for (Instance n2 : getReachable(n1)) {
                for (Instance n3 : getReachable(n2)) {
                    pairs.add(Arrays.asList(n1, n3));
                }
            }
        }
        Instance graph = model.getAllInstances("Graph").get(0);
        for (List<Instance> pair : pairs) {
            Instance n1 = pair.get(0);
            Instance n3 = pair.get(1);
            if (!getReachable(n1).contains(n3)) {
                Instance edge = model.newInstance("Edge");
                edge.set("src", n1);
                edge.set("trg", n3);
                graph.add("edges", edge);
            }
        }
    }
}
```