EvolveDB - Documentation for Developer

Author: Torben Eckwert, M.Sc.
E-Mail: torben.eckwert@zdh.thm.de

Subject Area: Computer Science

Supervisors: Prof. Dr. rer. nat. Michael Guckert
Prof. Dr. ing. Gabriele Taentzer

December 8, 2022 , Wetzlar

JUSTUS-LIEBIG-UNIVERSITÄT GIESSEN

Philipps Universität Marburg

THM TECHNISCHE HOCHSCHULE MITTELHESSEN

FORSCHUNGSCAMPUS MITTELHESSEN

Contents

1 Introduction

EvolveDB is an Eclipse-based framework for schema evolution in relational databases. The user specifies the evolution steps by freely editing a database model extracted by reverse engineering. EvolveDB analyzes the differences between the status quo and the evolved model structures and generates a data migration script. This user manual includes installation instructions and introductory tutorials on how to use EvolveDB as an end user.

2 Prerequisites and Installation

EvolveDB is a plug-in for recent versions of the Eclipse Modeling Tools (last tested version: 2022-06). EvolveDB is open source and licensed with the Apache license 2.0. The source code is available via our GitHub repository.

Before downloading the source code of EvolveDB, we need to install some additional Eclipse plugins. The following list contains the mandatory plugins.

- Eclipse OCL is available via the eclipse marketplace.
- The Henshin project provides a state-of-the-art model transformation language for the Eclipse Modeling Framework.
- ATL 3.5 (ATL Transformation Language) is a model transformation language and toolkit.
- ATL/EMFTVM 4.2.1 The EMF Transformation Virtual Machine (EMFTVM) is a runtime engine for the ATL Transformation Language (ATL).
- Xtend is a statically-typed programming languages for Java developers.

SiLift is a generic model comparison environment for EMF-based models. The SiLift update site is no longer available, but SiLift is included in the release version of EvolveDB. **Help** ⇒ **Install New Software...** (fig. 1).

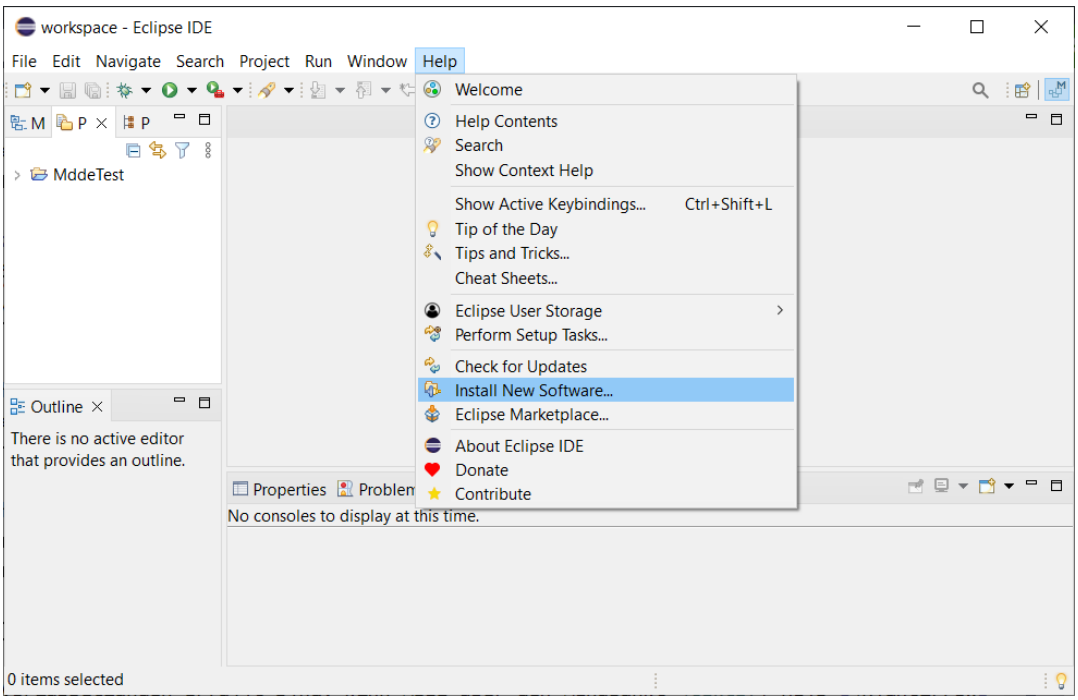

Figure 1: Eclipse: Install New Software...

The repository contains multiple categories (fig. 2). For installing SiLift, select *SiLift* and *SiLift Matcher*.

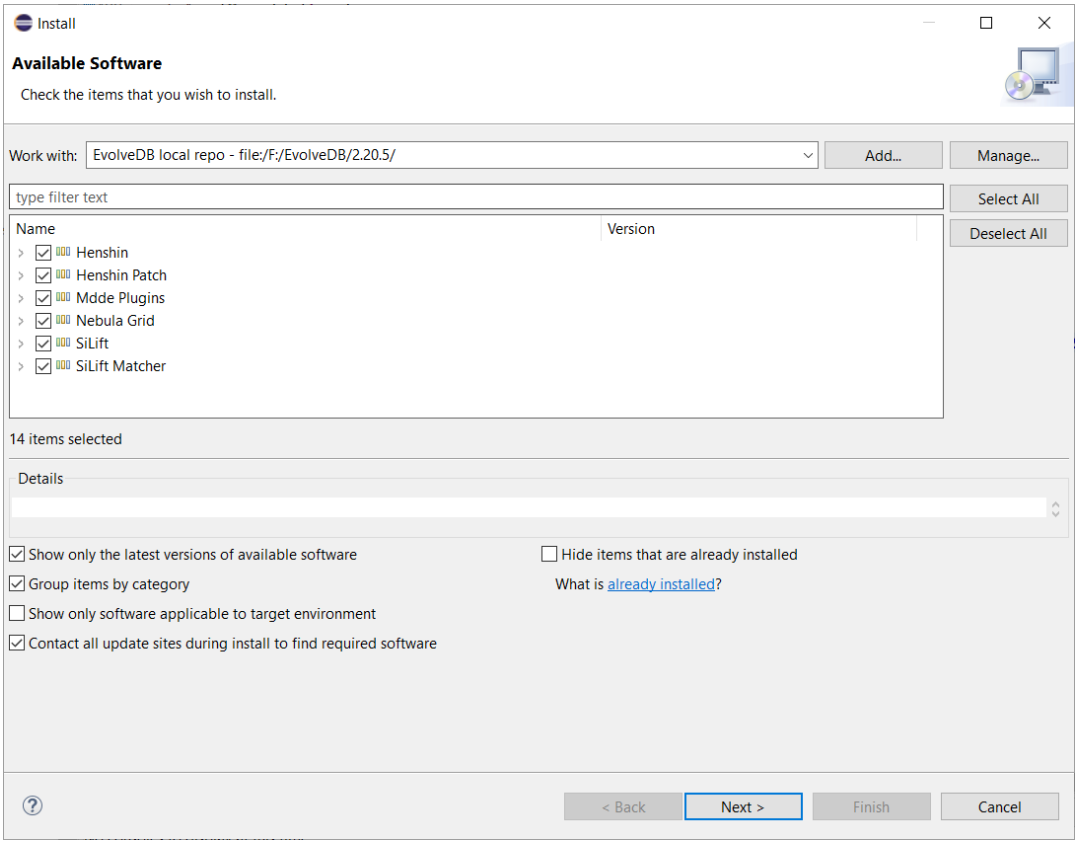

Figure 2: Eclipse: EvolveDB local repository

If all requirements are fulfilled, the source code of EvolveDB can be cloned and imported.

3 EvolveDB

The schema evolution with EvolveDB is a model-driven reengineering process that includes three phases, reverse engineering, restructuring, and forward engineering. In the first step, we start by creating a representation of a database schema through reverse engineering. In the restructuring phase, we edit this model resulting in a second model version. Afterwards, EvolveDB analyzes the differences between the status quo and the evolved model structures. From the delta between the two model versions, SQL migration scripts are generated, which can be used to migrate the schema and the associated data. EvolveDB includes a data source and a migration script generator which supports MySQL databases. If other data sources or generators should be used, EvolveDB provides two extension points. By contributing to these extension points, custom data sources or generators can be integrated.

3.1 Create a new datasource

As already mentioned, custom drivers can be integrated into EvolveDB. To do this, create a plug-in via **File** ⇒ **New** ⇒ **Other** ⇒ **Plug-in Development** ⇒ **Plug-in Project** and open the MANIFEST.MF file. Switch to the tab *Dependencies* and add the dependencies marked in Figure 4.

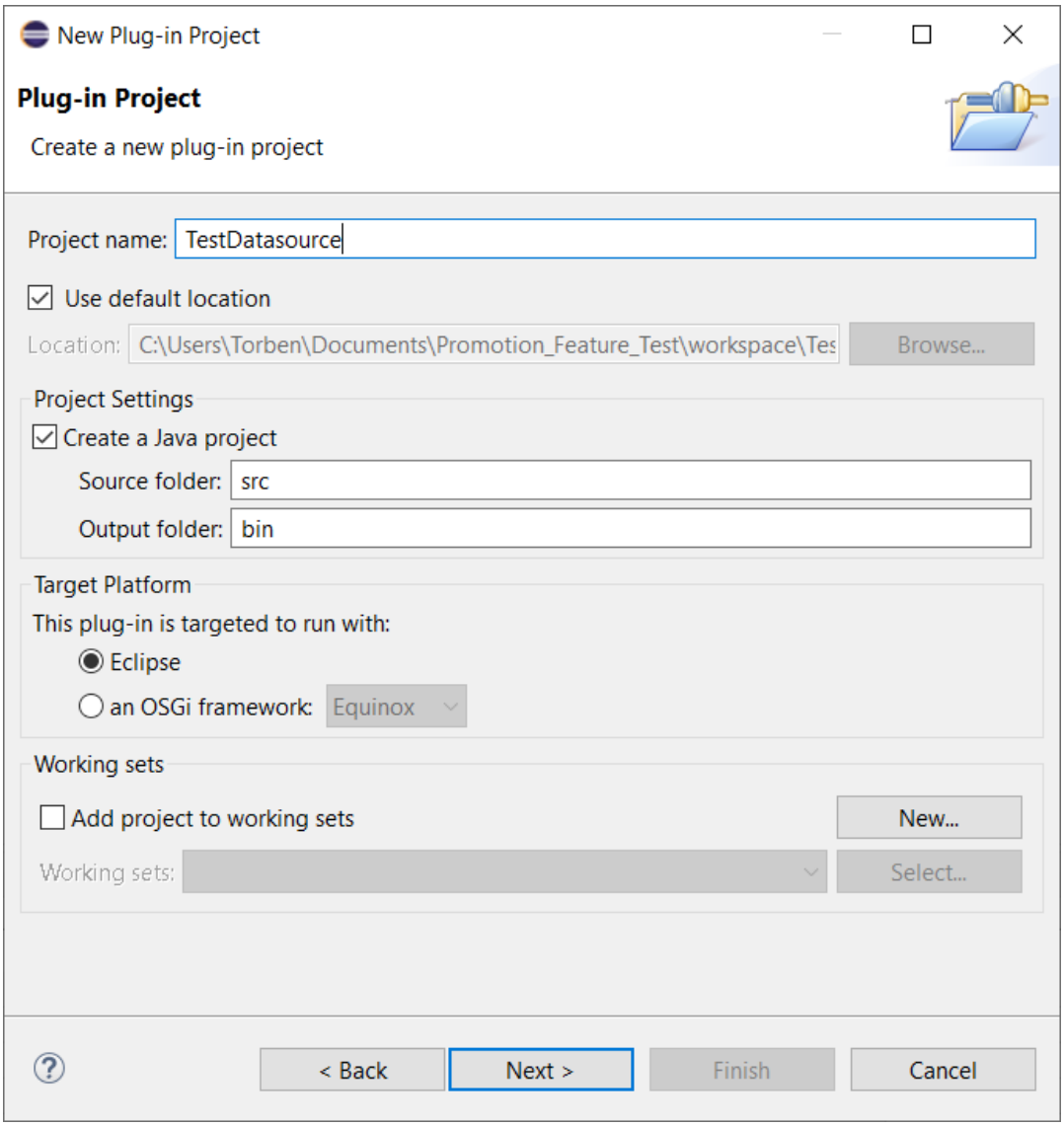

Figure 3: Eclipse: Create New Plug-in Project

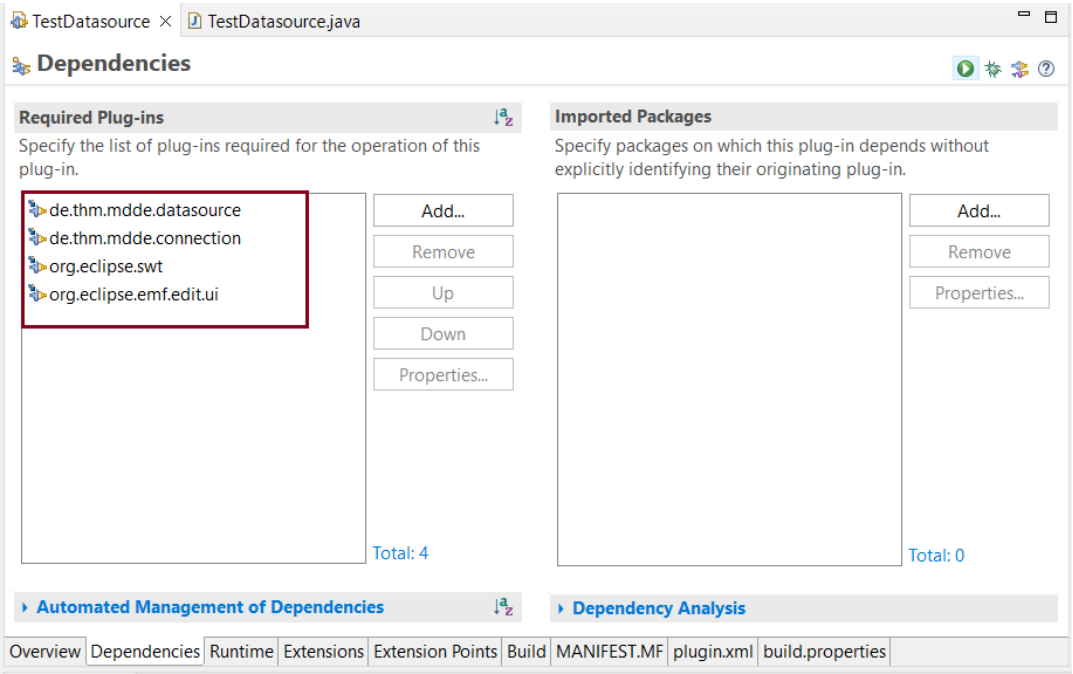

Figure 4: Plug-in Project Manifest.mf

Next we have to create a class that implements the *EDBDataSource* (fig. 5) interface. Figure 6 shows the class TestDatasource that implements the interface.

```java
import org.eclipse.emf.ecore.EObject;

public interface EDBDataSource {

    /**
     * Datasource name
     * @return the name of the datasource
     */
    String getName();


    /**
     * Connection icon
     * @return Database Icon
     */
    Image getImage();


    /**
     * DBPDriver class
     * @return Driver class
     */
    DBPDriver getDriver();



    /**
     * DBPDriverDependencies
     * @return the corresponding driver dependencies
     */
    DBPDriverDependencies getDriverDependencies();


    /**
     * The UI for entering the connection details.
     * @return
     */
    void openConnectionUi();


    /**
     * Returns the root element of the newly created model;
     * @return
     */
    EObject getRootObject();


    /**
     * Returns true if the user left the wizard via the cancel button.
     * @return
     */
    boolean isCanceled();


}
```

Figure 5: EDBDataSource Interface

TestDatasource × TestDatasource.java ×

```java
package de.thm.mdde.datasource;

import org.eclipse.emf.ecore.EObject;
import org.eclipse.swt.graphics.Image;

import de.thm.mdde.connection.model.DBPDriver;
import de.thm.mdde.connection.model.DBPDriverDependencies;

public class TestDatasource implements EDBDataSource{

    @Override
    public DBPDriver getDriver() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public DBPDriverDependencies getDriverDependencies() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public Image getImage() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public String getName() {
        // TODO Auto-generated method stub
        return "TestDriver";
    }

    @Override
    public EObject getRootObject() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public boolean isCanceled() {
        // TODO Auto-generated method stub
        return false;
    }

    @Override
    public void openConnectionUi() {
        // TODO Auto-generated method stub

    }

}
```

Figure 6: Class TestDatasource

Next we have to add the new class as an extension for EvolveDB. Switch back to the *MANIFEST.MF* and open the Extension tab. Click on add and select the extension point *de.thm.mdde.datasource*. (fig. 7)

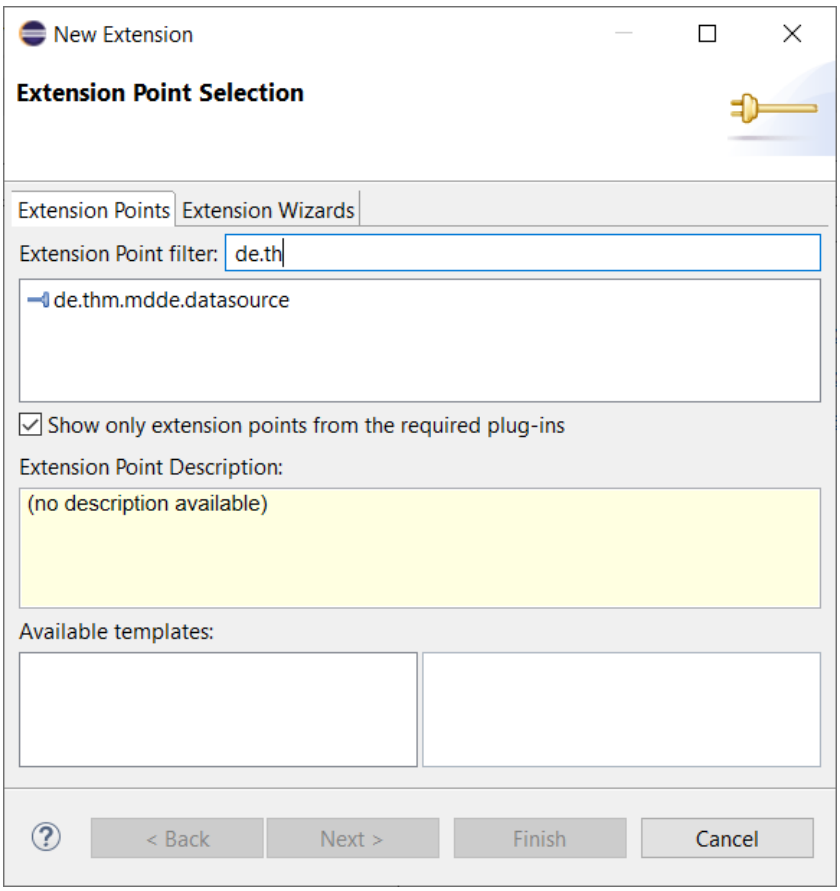

Figure 7: de.thm.mdde.datasource extension point

Select the *plugin.xml* and add the path to the class we created. Figure 8 shows the edited *plugin.xml*.

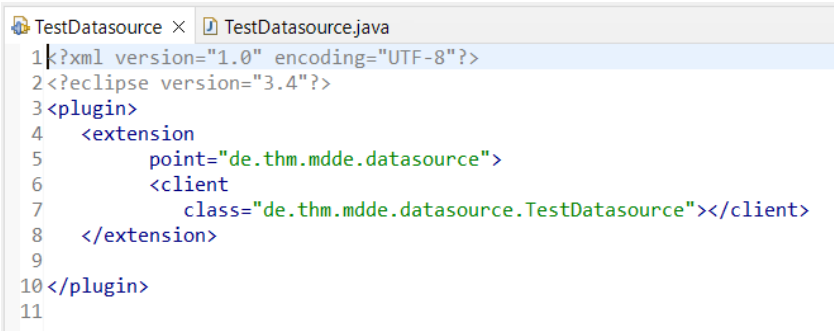

Figure 8: Plugin.xml

Finally, we have to deploy the new driver. Select the project in the package or file explorer and open the context menu with a right-click. In the context menu we choose export (9).

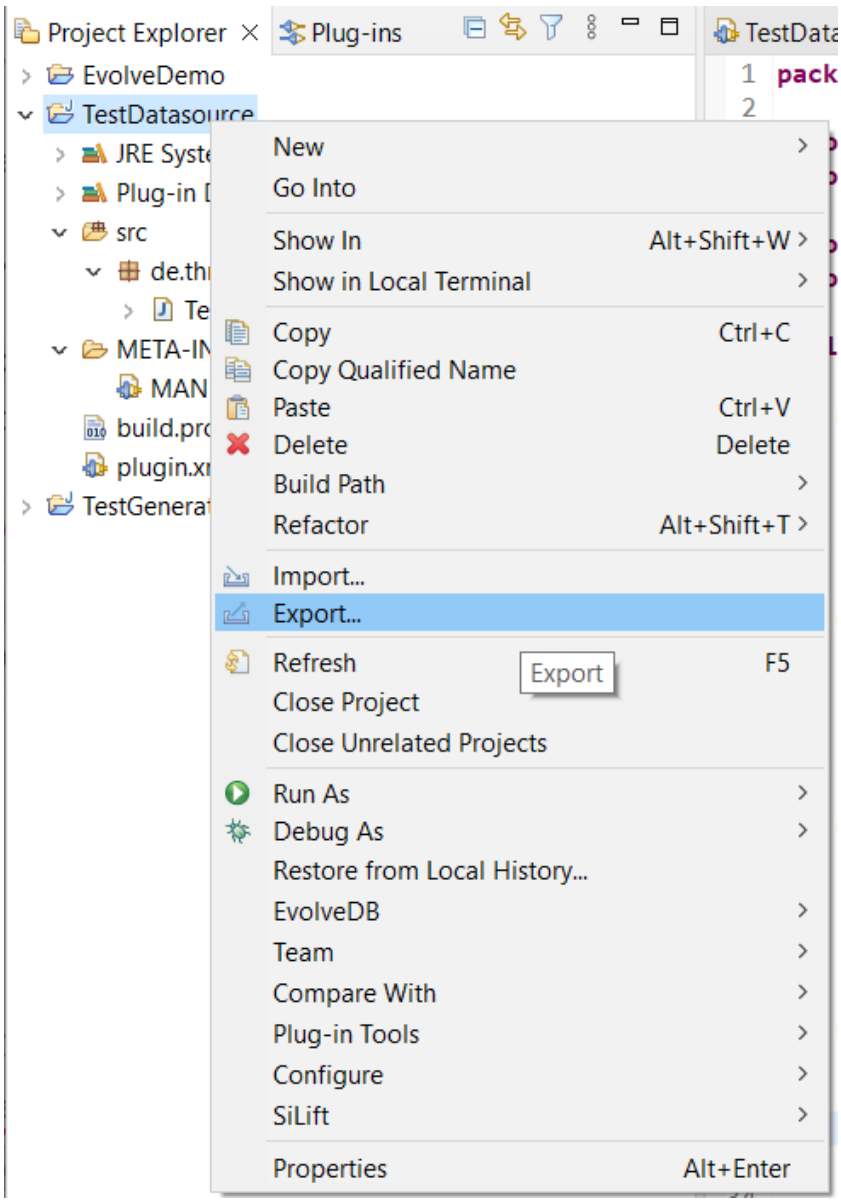

Figure 9: Export a plugin

A dialog opens. In the dialog, we select **Plug-in Development** ⇒ **Deployable plug-ins and fragements** and click next (fig. 10).

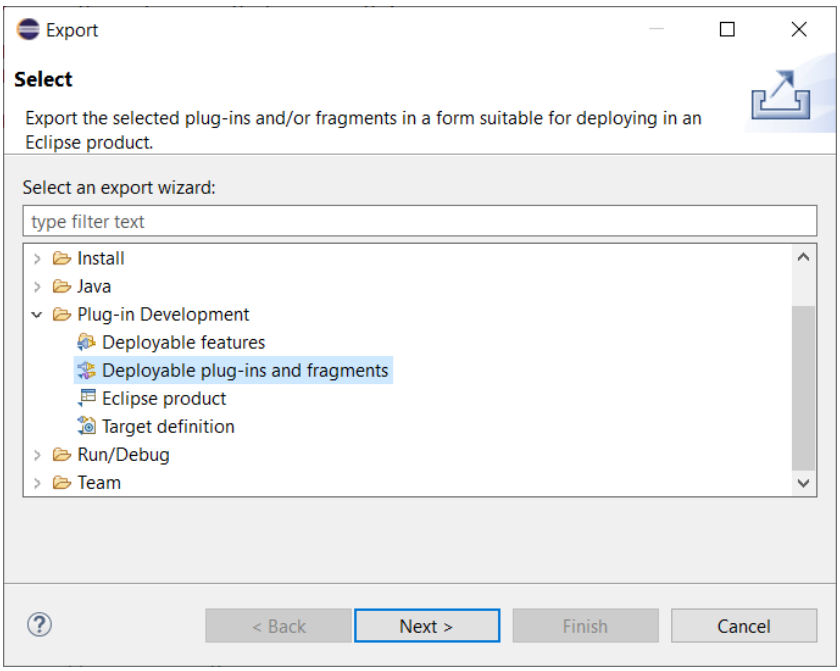

Figure 10: Export Wizard: page 1

Next, select *install into host.Repository* and specify the path to the plug-in folder of your Eclipse installation (fig. 11). Click Finish and restart your Eclipse after successful installation. The new data source can now be used.

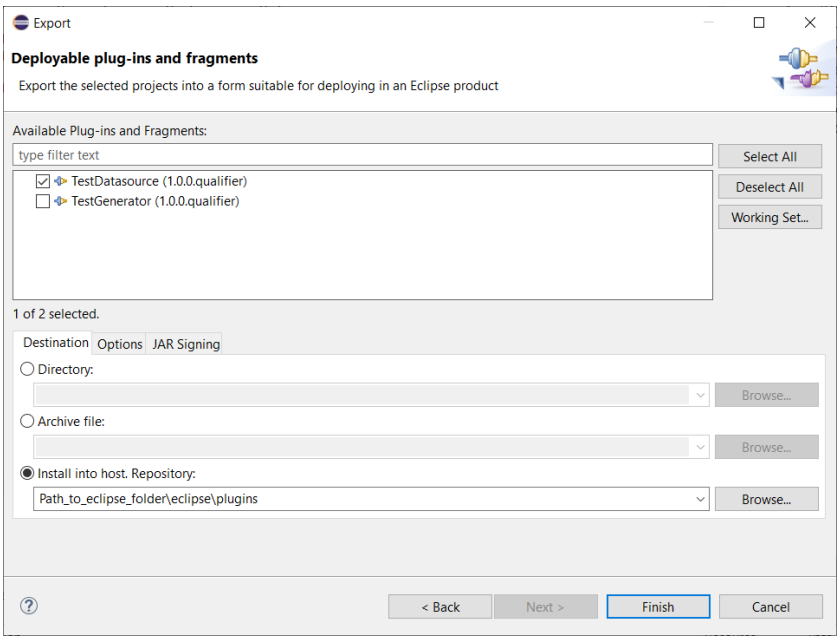

Figure 11: Export wizard: page 2

3.2 Create a new migration script generator

The default installation EvolveDB includes a migration script generator for MySQL version 5.7 or higher. It is also possible to add custom generators via an extension point. To do this, create a plug-in via **File** ⇒ **New** ⇒ **Other** ⇒ **Plug-in Development** ⇒ **Plug-in Project** and open the MANIFEST.MF file. Switch to the tab *Dependencies* and add the dependencies marked in Figure 13.

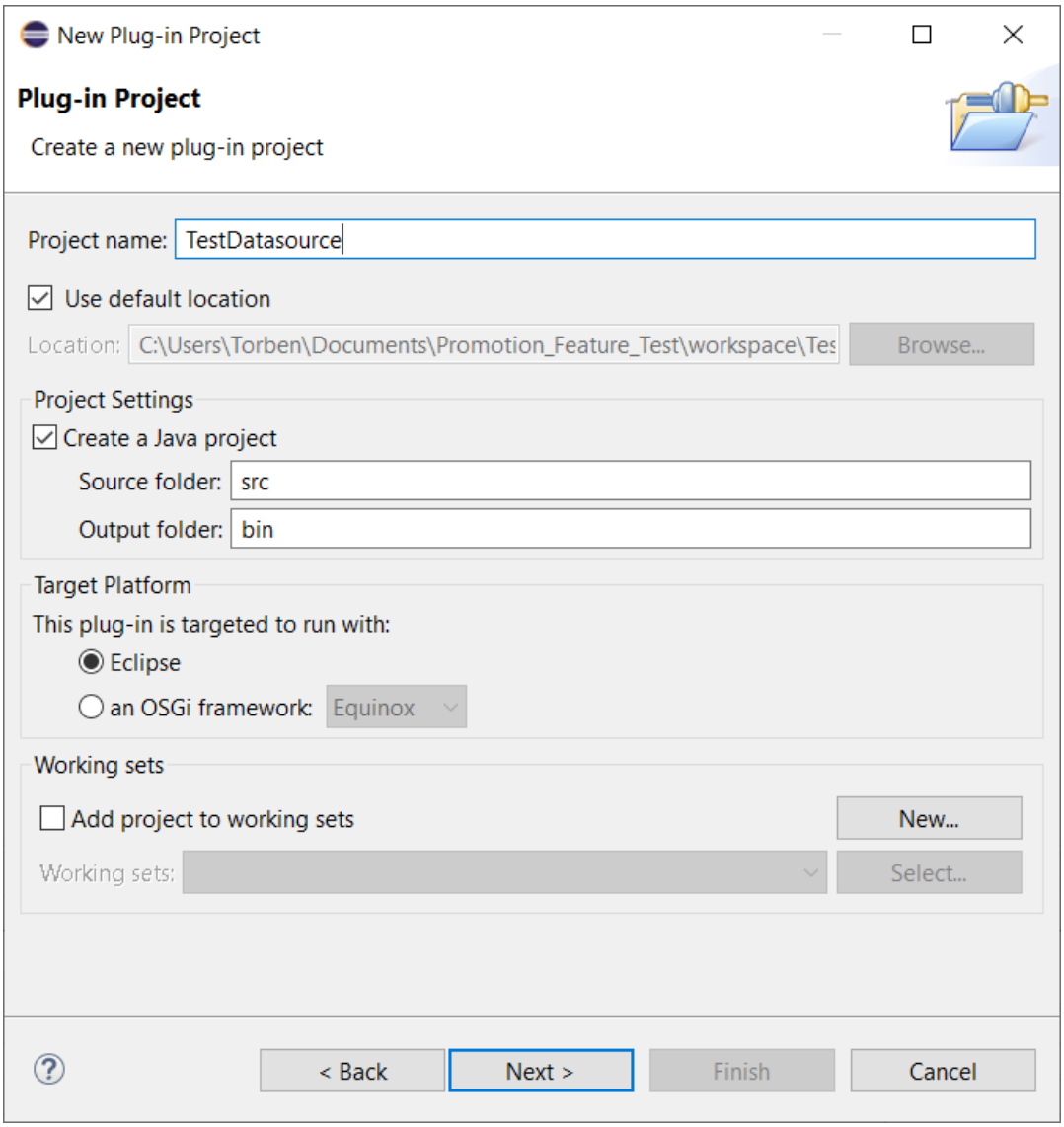

Figure 12: Eclipse: Create New Plug-in Project

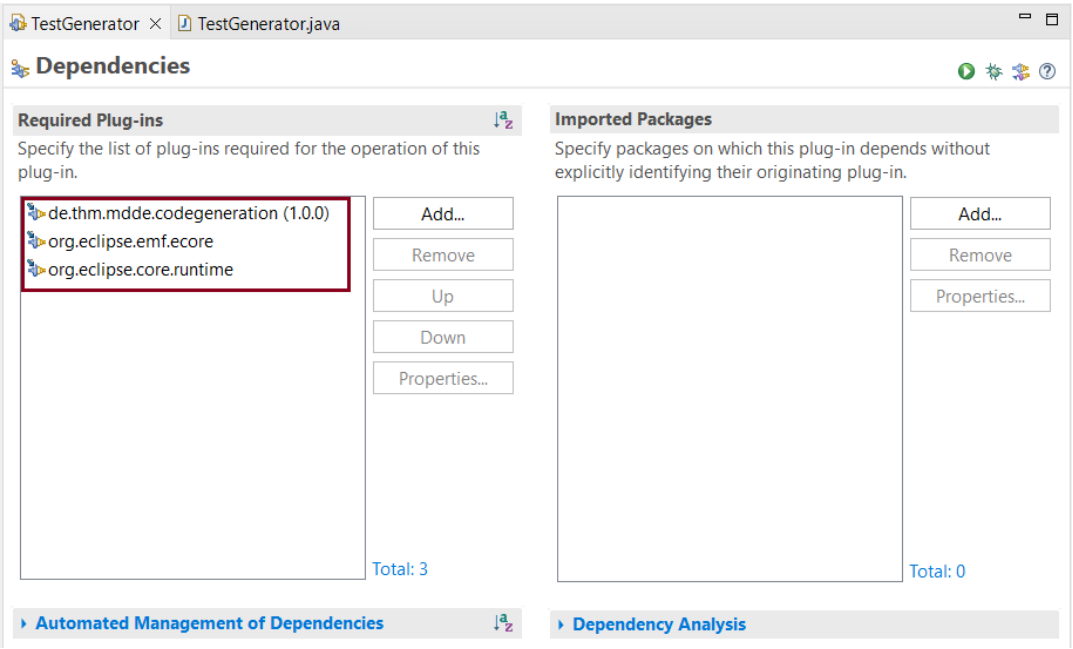

Figure 13: Plug-in Project Manifest.mf

Next, we have to create a class that implements the *ISQLGenerator* (fig. 14) interface. Figure 15 shows the class TestGenerator that implements the interface.

```java
import org.eclipse.core.runtime.IProgressMonitor;

public interface ISQLGenerator {

    /**
     * Returns the display name of the extension.
     * @return
     */
    String getDisplayName();


    /**
     * Generate the migrations.
     * @param resEcoreFile --> The matching model.
     * @param project --> The currently selected project.
     * @param generator --> The generator chosen by the user.
     * @param monitor --> ProgressMonitor
     */
    void generate(Resource resEcoreFile, IProgressMonitor monitor);



    /**
     * Returns the content for the migration script.
     * @return
     */
    String getContent();

}
```

Figure 14: ISQLGenerator Interface

```java
TestGenerator    TestGenerator.java
1  package de.thm.mdde.generator;
2
3  import org.eclipse.core.runtime.IProgressMonitor;
4  import org.eclipse.emf.ecore.resource.Resource;
5
6  import de.thm.extensionpoint.interfaces.ISQLGenerator;
7
8  public class TestGenerator implements ISQLGenerator{
9
10     @Override
11     public void generate(Resource arg0, IProgressMonitor arg1) {
12         // TODO Auto-generated method stub
13
14     }
15
16     @Override
17     public String getContent() {
18         // TODO Auto-generated method stub
19         return null;
20     }
21
22     @Override
23     public String getDisplayName() {
24         return "TestGenerator";
25     }
26
```

Figure 15: Class TestGenerator

Next, we have to add the new class as an extension for EvolveDB. Switch back to the *MANIFEST.MF* and open the Extension tab. Click on add and select the extension point *de.thm.mdde.extensionpoint.SQLGenerator*. (fig. 16)

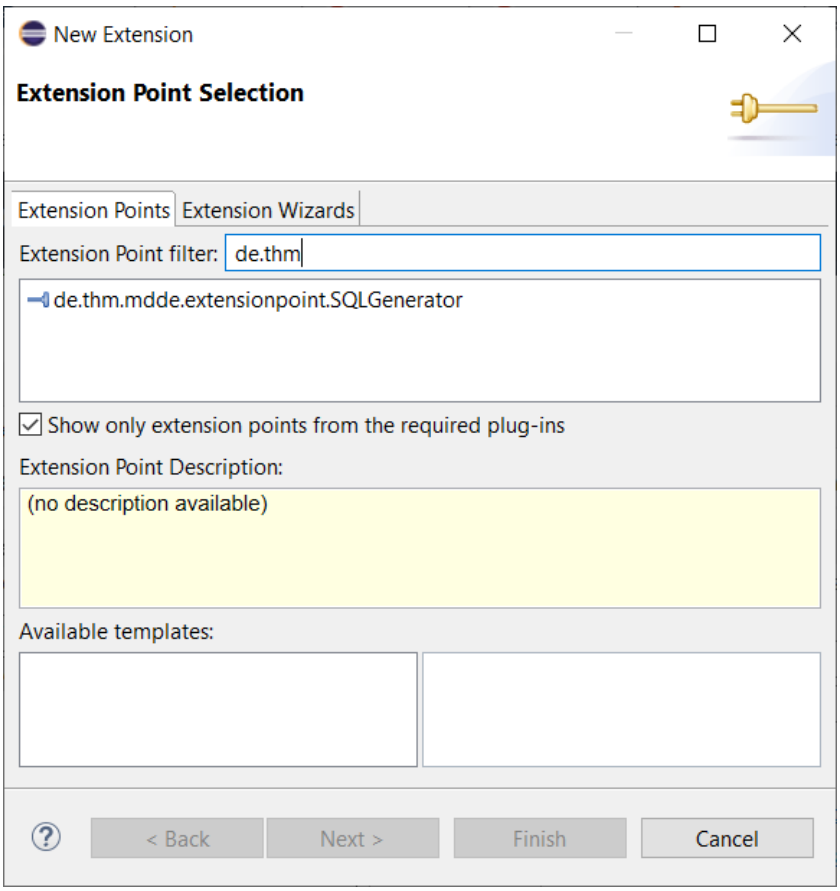

Figure 16: de.thm.mdde.extensionpoint.SQLGenerator extension point

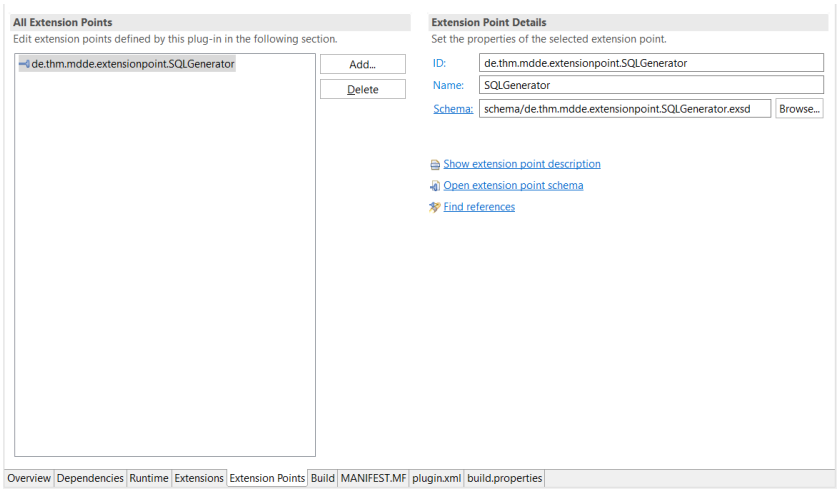

Figure 17: de.thm.mdde.extensionpoint.SQLGenerator extension point

Select the *plugin.xml* and add the path to the class we created. Figure 18 shows the edited *plugin.xml*.

TestGenerator × TestGenerator.java

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <?eclipse version="3.4"?>
3 <plugin>
4    <extension
5          point="de.thm.mdde.extensionpoint.SQLGenerator">
6          <client
7             class="de.thm.mdde.generator.TestGenerator"></client>
8    </extension>
9
10 </plugin>
11
12
```

Overview | Dependencies | Runtime | Extensions | Extension Points | Build | MANIFEST.MF | plugin.xml | build.properties

Figure 18: Plugin.xml

Finally, we have to deploy the new driver. Select the project in the package or file explorer and open the context menu with a right-click. In the context menu, we choose export (19).

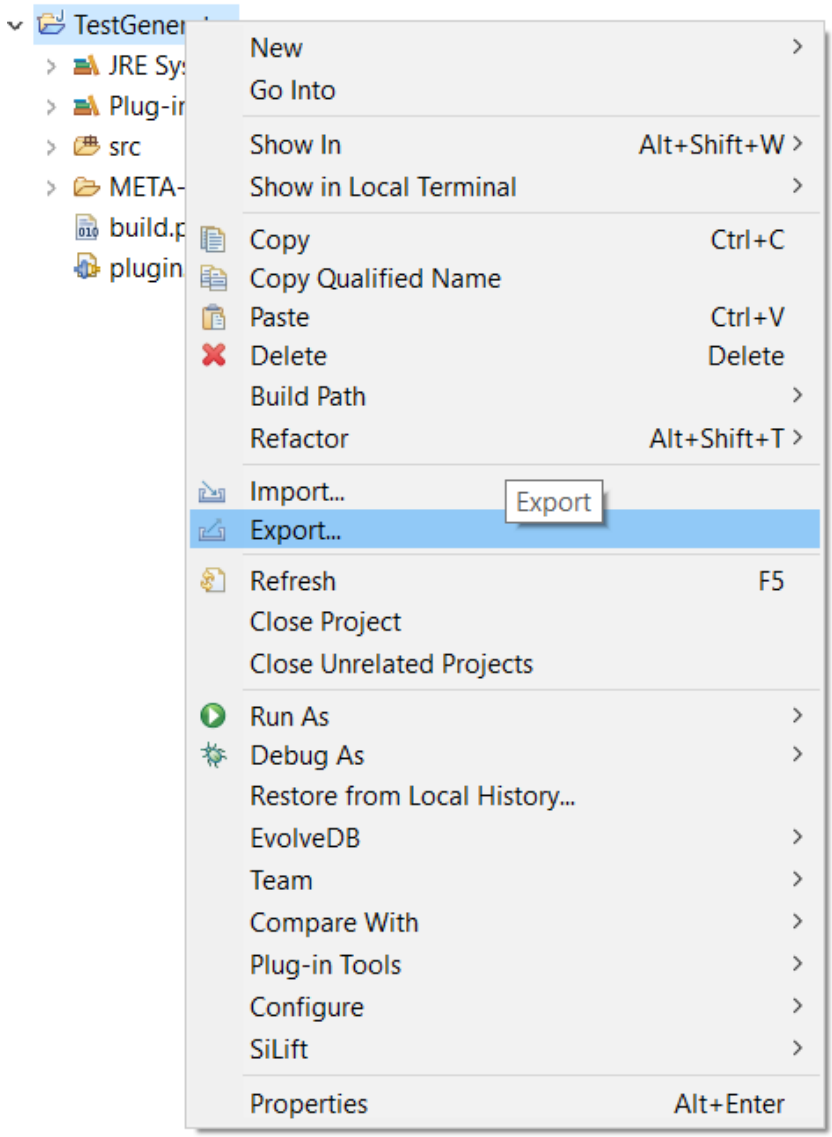

Figure 19: Export a plugin

A dialog opens. In the dialog, we select **Plug-in Development** ⇒ **Deployable plug-ins and fragements** and click next (fig. 20).

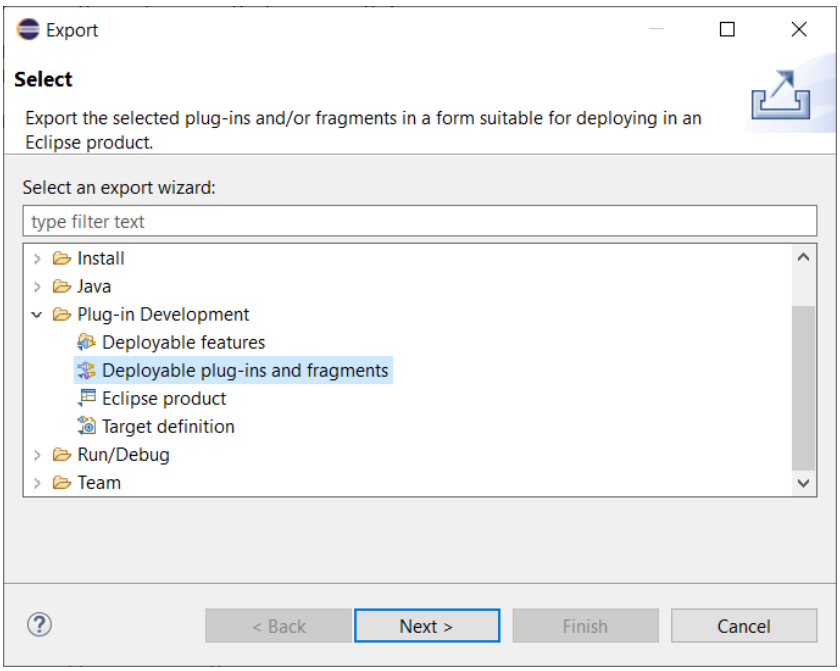

Figure 20: Export Wizard: page 1

Next, select *install into host.Repository* and specify the path to the plug-in folder of your Eclipse installation (fig. 21). Click Finish and restart your Eclipse after successful installation. The new migration script generator can now be used.

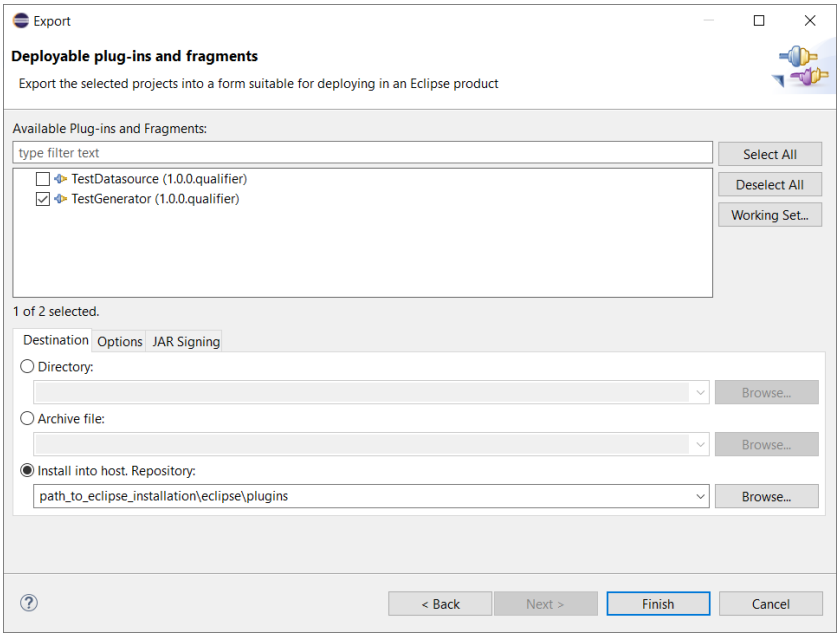

Figure 21: Export wizard: page 2

4 Metamodel

EvolveDB uses EMF for the database (MDDE model) and the migration metamodel. This section gives a short overview of both metamodels.

4.1 MDDE metamodel

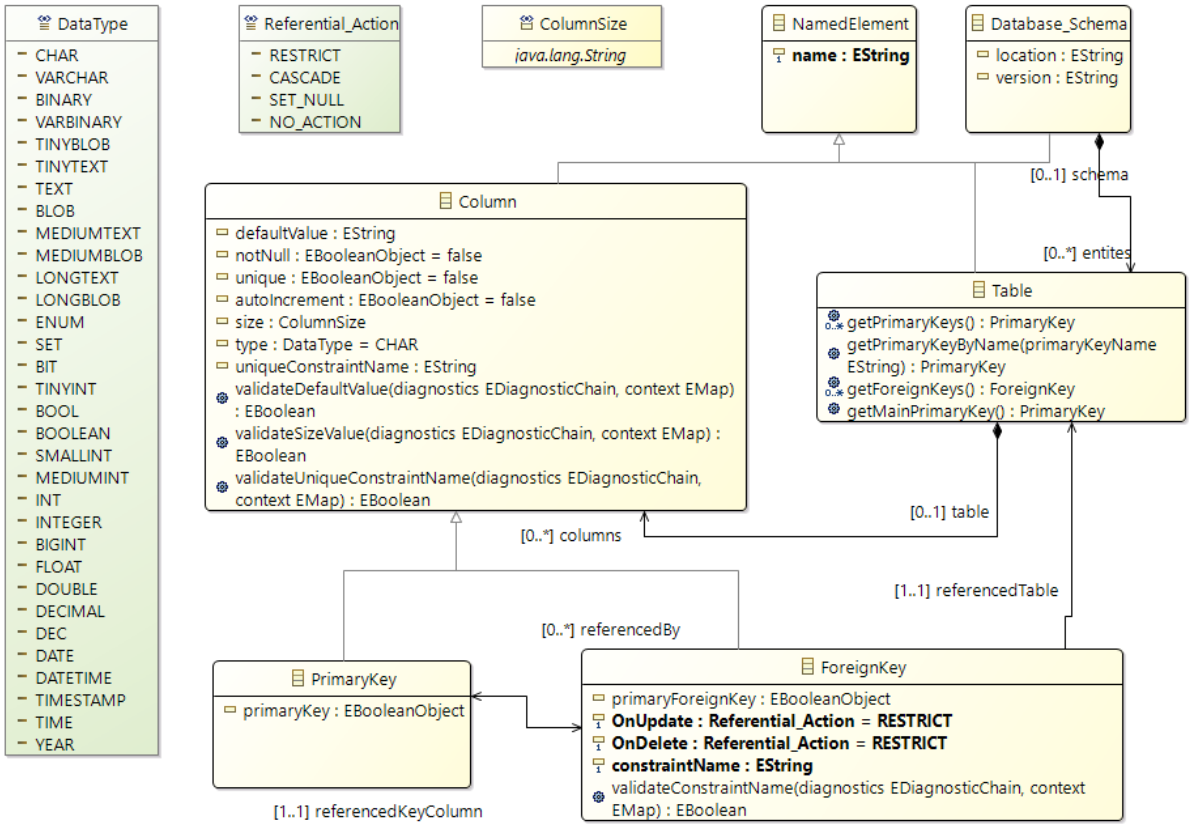

Figure 22: MDDE metamodel

The abstract syntax of the MDDE metamodel (fig. 22) is similar to the relational model. When creating a new data source, the method *getRootObject()* from the *EDBDataSource* interface has to return the root object from an instance of this metamodel. Typically, the root *EObject* is an instance of class *Database_Schema*.

4.2 Migration metamodel

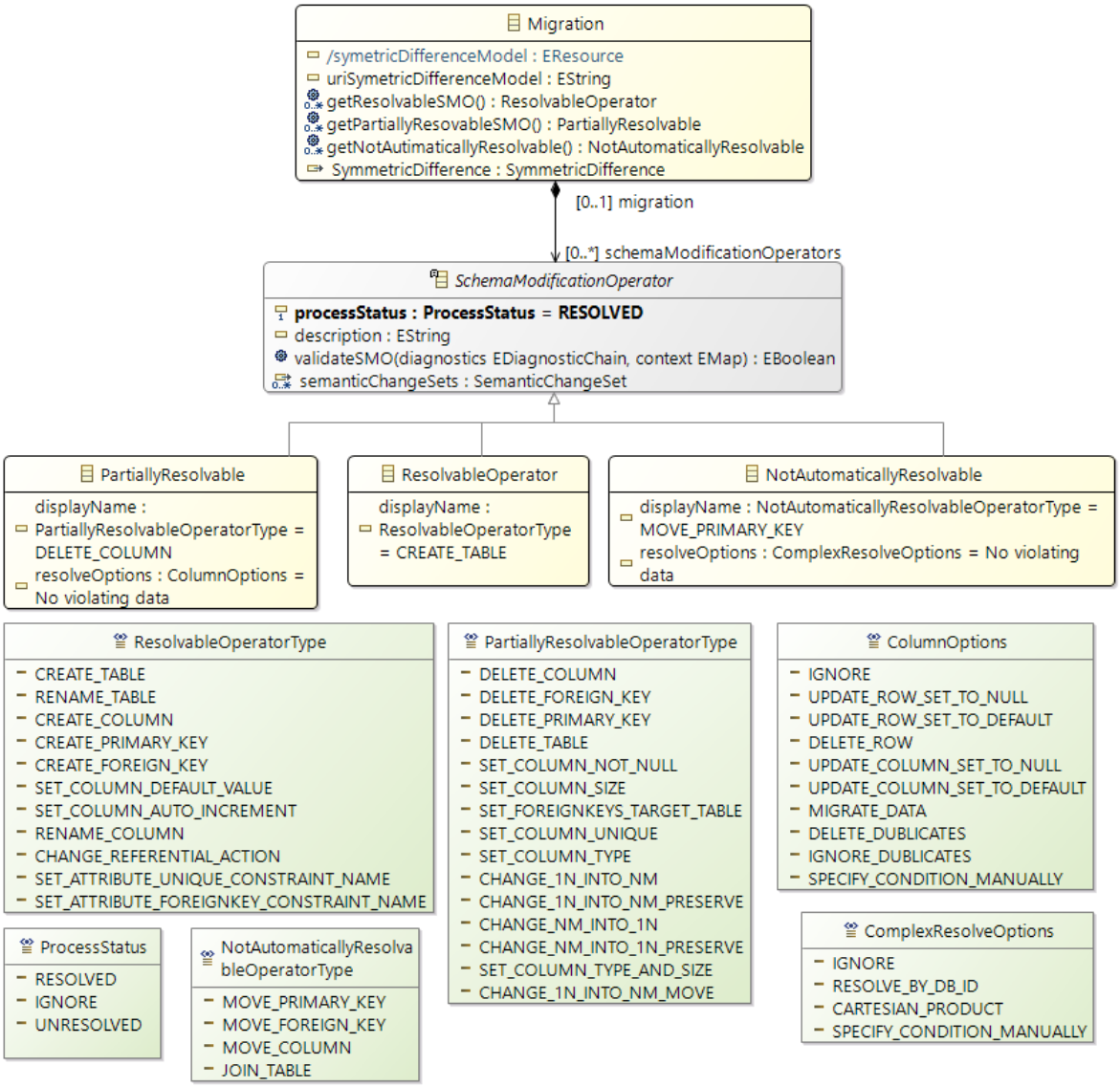

Figure 23: Migration metamodel

The symmetric difference model created by SiLift during the model matching phase is only used to represent the difference between the two model versions. The difference model is unsuitable for adding the additional information required for the migration. For this reason, we convert the *difference.symmetric* model into a migration model. Figure 23 shows the metamodel for the migration model. The migration model references the symmetric difference model and both mdde models.