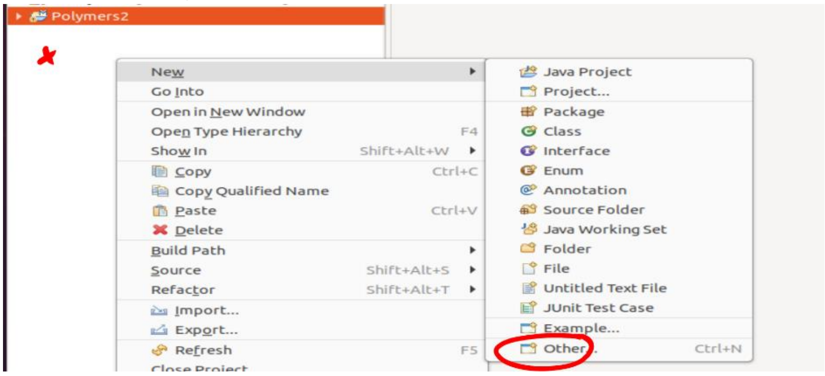
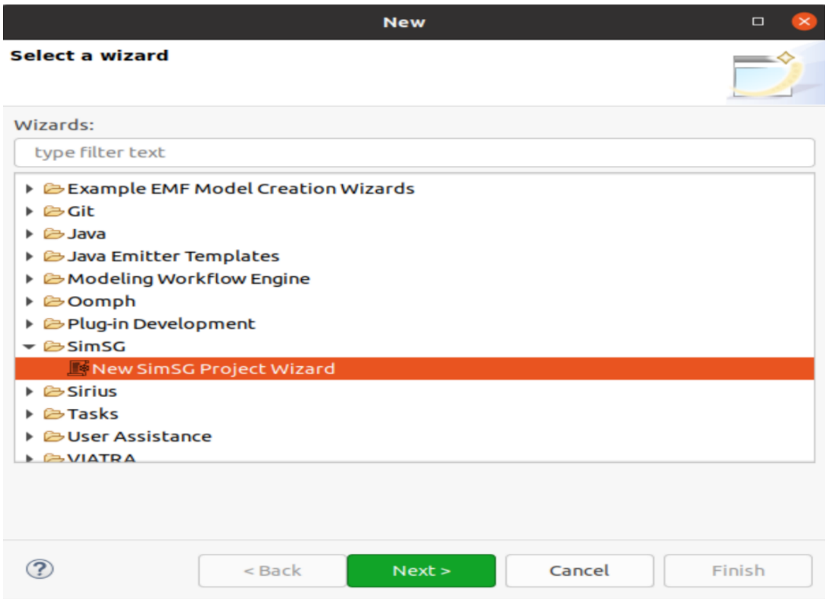
Creating Simulation Models

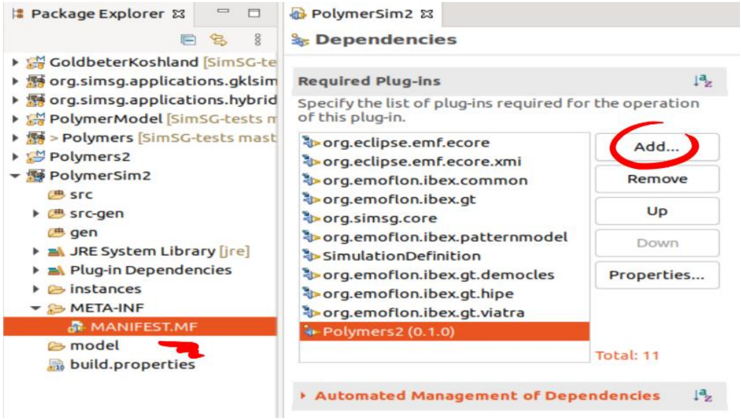
and how to run your own simulations

1. You need to have a metamodel prepared beforehand. If you don’t know how to do that, please refer to the Ecore Metamodeling – Tutorial.
2. Create a new SimSG project, by right-clicking in package explorer: New -> Other… -> SimSG -> New SimSG Project Wizard -> Next



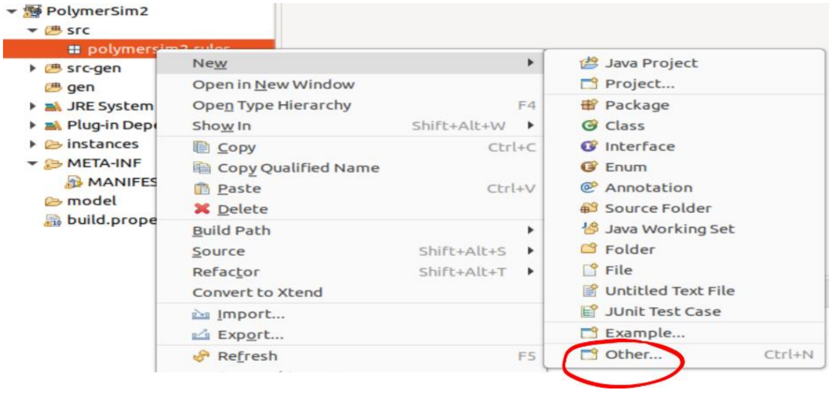


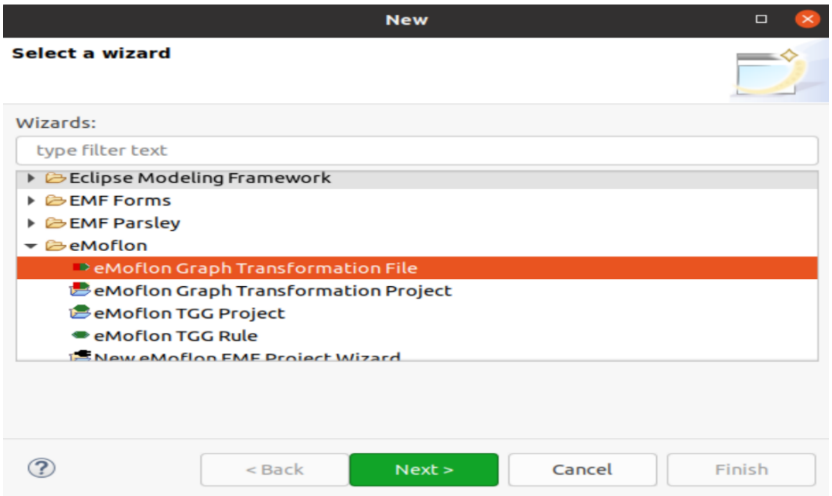
3. Select a Name for your Project, if necessary, set your custom project location and click on “Finish”.
4. Open your MANIFEST.MF file and add your metamodel to the project by using the “Dependencies” tab.



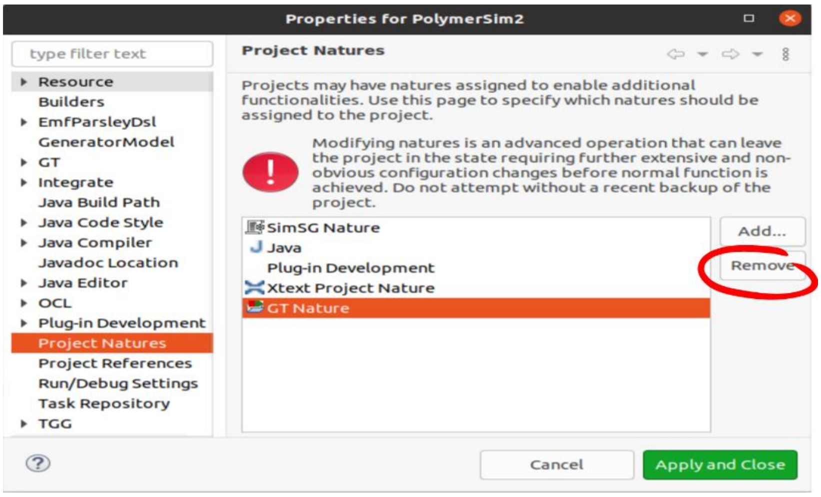
5. **Example: Create your own rules**

   a. Add a package below your src-folder (e.g. polymersim2.rules) and create a new GT-Rule file within the package: Rick-Click on package -> New -> Other … -> eMoflon -> eMoflon Graph Transformation File -> Next
   After that, name your new file and click on “Finish”.





   b. eMoflon automatically adds the emoflon nature to your project. This nature causes conflicts in the build process and, therefore, must be removed: Right-Click on your SimSG project -> Properties -> Project Natures. In this window select GT Nature, click on the “Remove” button, confirm with “Ok” and click on “Apply and Close”.

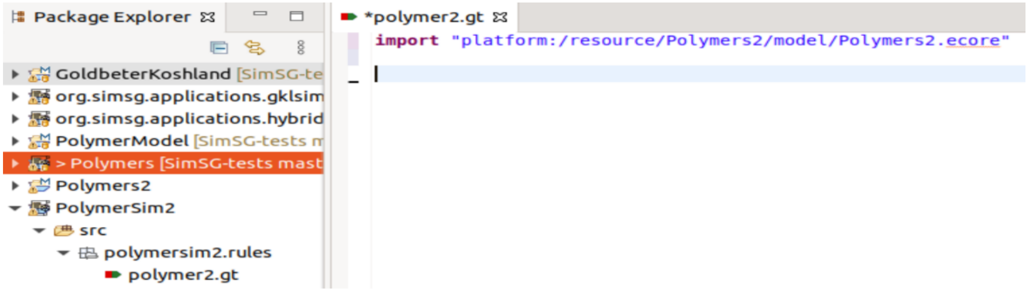


Projects may have natures assigned to enable additional functionalities. Use this page to specify which natures should be assigned to the project.

Modifying natures is an advanced operation that can leave the project in the state requiring further extensive and non-obvious configuration changes before normal function is achieved. Do not attempt without a recent backup of the project.

   Select your project in the package explorer and press F5.

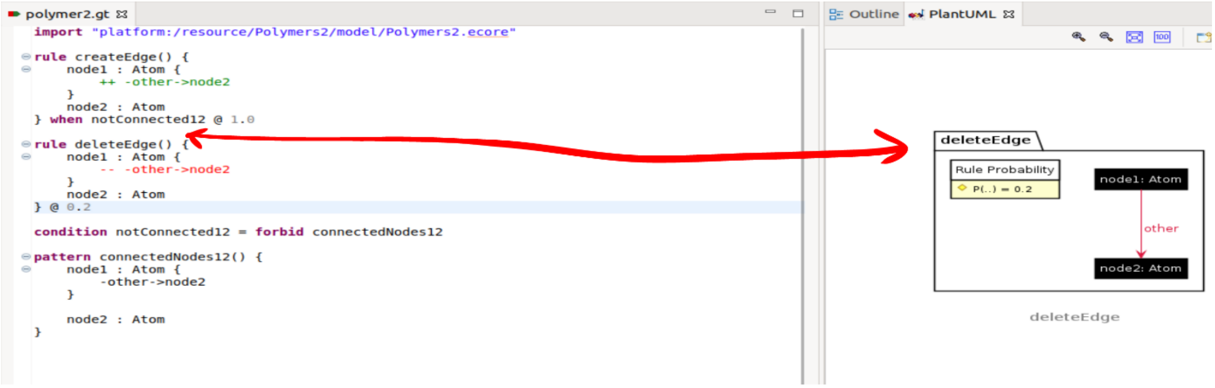
c. Open your GT-File (if it isn’t already opened) and change the default import to match the URI of your metamodel.



Package Explorer
GoldbeterKoshland [SimSG-te
org.simsg.applications.gklsim
org.simsg.applications.hybrid
PolymerModel [SimSG-tests m
> Polymers [SimSG-tests mast
Polymers2
PolymerSim2
src
polymersim2.rules
polymer2.gt
*polymer2.gt

```
import "platform:/resource/Polymers2/model/Polymers2.ecore"
```

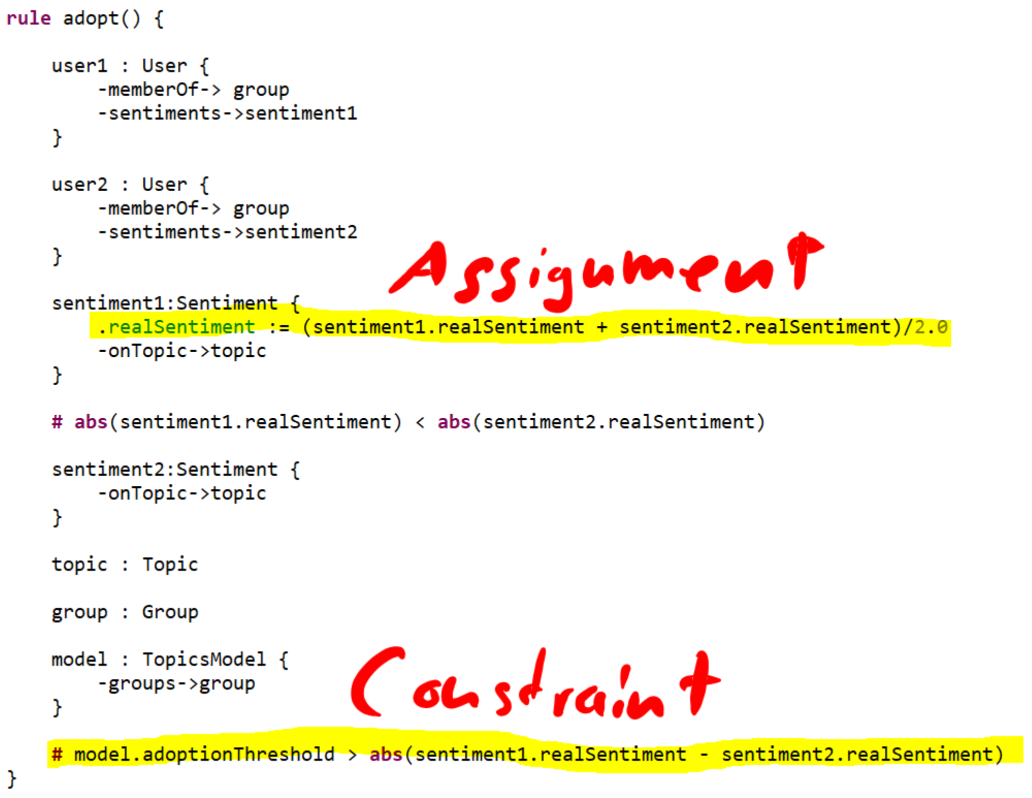
d. Add your rules, for example, a rule that creates edges and a rule that deletes edges. You can see the graph representation of your rules in the PlantUML view.



```
import "platform:/resource/Polymers2/model/Polymers2.ecore"

rule createEdge() {
    node1 : Atom {
        ++ -other->node2
    }
    node2 : Atom
} when notConnected12 @ 1.0

rule deleteEdge() {
    node1 : Atom {
        -- -other->node2
    }
    node2 : Atom
} @ 0.2

condition notConnected12 = forbid connectedNodes12

pattern connectedNodes12() {
    node1 : Atom {
        -other->node2
    }

    node2 : Atom
}
```

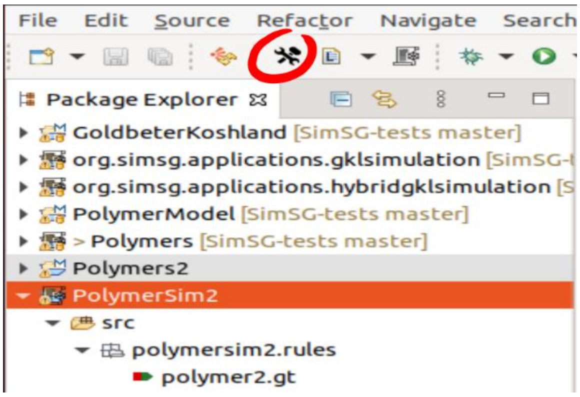
To create new elements or edges, use the **++** prefix. For deletions use the **--** prefix. The **when** keyword denotes a positive or negative pattern invocation, which can be defined using the **condition** keyword. The **forbid** or **enforce** keywords can be used to define negative or positive conditions. Rule rates used by the SimSG simulation can be defined using the **@**-Symbol. Currently SimSG only support static rule rates.
For more syntax examples please have a look at the other projects in the workspace (e.g. Polymers).

e. You may also define attribute constraints (**#**-Symbol) or attribute assignments (**:=**), with or without arithmetic expressions.

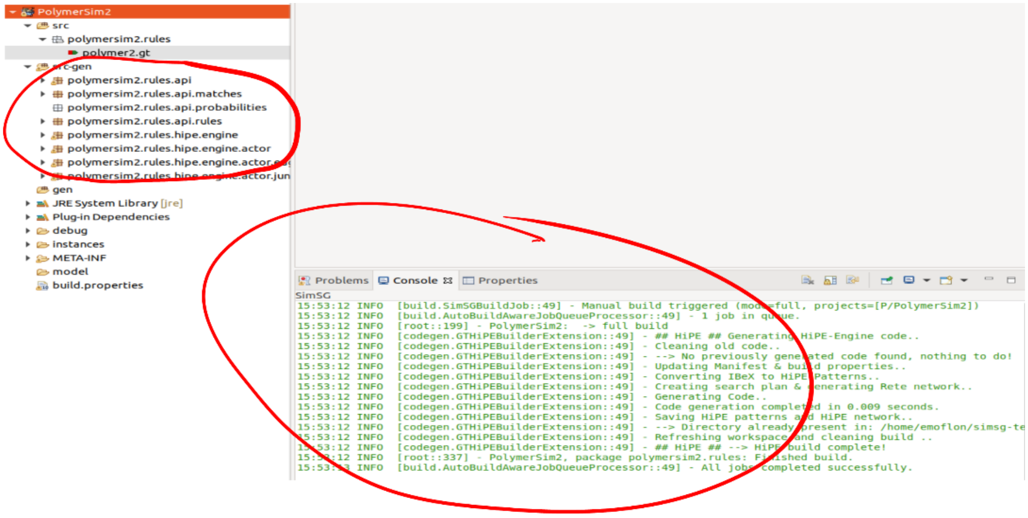


```
rule adopt() {

    user1 : User {
        -memberOf-> group
        -sentiments->sentiment1
    }

    user2 : User {
        -memberOf-> group
        -sentiments->sentiment2
    }

    sentiment1:Sentiment {
        .realSentiment := (sentiment1.realSentiment + sentiment2.realSentiment)/2.0
        -onTopic->topic
    }

    # abs(sentiment1.realSentiment) < abs(sentiment2.realSentiment)

    sentiment2:Sentiment {
        -onTopic->topic
    }

    topic : Topic

    group : Group

    model : TopicsModel {
        -groups->group
    }

    # model.adoptionThreshold > abs(sentiment1.realSentiment - sentiment2.realSentiment)
}
```

f. Save your GT-file, close it, and click on the build button to generate the necessary SimSG api java code.


File
Edit
Source
Refactor
Navigate
Search
Package Explorer
GoldbeterKoshland [SimSG-tests master]
org.simsg.applications.gklsimulation [SimSG-t
org.simsg.applications.hybridgklsimulation [S
PolymerModel [SimSG-tests master]
> Polymers [SimSG-tests master]
Polymers2
PolymerSim2
src
polymersim2.rules
polymer2.gt

If everything went well, the console output will report that all files have been generated and the src-gen folder should be populated by numerous packages.


PolymerSim2
src
polymersim2.rules
polymer2.gt
src-gen
polymersim2.rules.api
polymersim2.rules.api.matches
polymersim2.rules.api.probabilities
polymersim2.rules.api.rules
polymersim2.rules.hipe.engine
polymersim2.rules.hipe.engine.actor
polymersim2.rules.hipe.engine.actor.e
polymersim2.rules.hipe.engine.actor.jun
gen
JRE System Library [jre]
Plug-in Dependencies
debug
instances
META-INF
model
build.properties
Problems
Console
Properties
SimSG

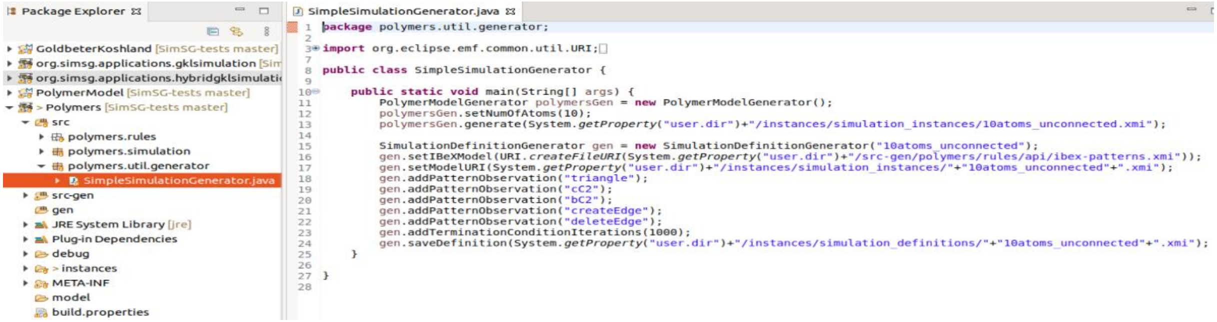
```
15:53:12 INFO  [build.SimSGBuildJob::49] - Manual build triggered (mode=full, projects=[P/PolymerSim2])
15:53:12 INFO  [build.AutoBuildAwareJobQueueProcessor::49] - 1 job in queue.
15:53:12 INFO  [root::199] - PolymerSim2:  -> full build
15:53:12 INFO  [codegen.GTHiPEBuilderExtension::49] - ## HiPE ## Generating HiPE-Engine code..
15:53:12 INFO  [codegen.GTHiPEBuilderExtension::49] - Cleaning old code..
15:53:12 INFO  [codegen.GTHiPEBuilderExtension::49] - --> No previously generated code found, nothing to do!
15:53:12 INFO  [codegen.GTHiPEBuilderExtension::49] - Updating Manifest & build properties..
15:53:12 INFO  [codegen.GTHiPEBuilderExtension::49] - Converting IBeX to HiPE Patterns..
15:53:12 INFO  [codegen.GTHiPEBuilderExtension::49] - Creating search plan & generating Rete network..
15:53:12 INFO  [codegen.GTHiPEBuilderExtension::49] - Generating Code..
15:53:12 INFO  [codegen.GTHiPEBuilderExtension::49] - Code generation completed in 0.009 seconds.
15:53:12 INFO  [codegen.GTHiPEBuilderExtension::49] - Saving HiPE patterns and HiPE network..
15:53:12 INFO  [codegen.GTHiPEBuilderExtension::49] - --> Directory already present in: /home/emoflon/simsg-te
15:53:12 INFO  [codegen.GTHiPEBuilderExtension::49] - Refreshing workspace and cleaning build ..
15:53:12 INFO  [codegen.GTHiPEBuilderExtension::49] - ## HiPE ## --> HiPE build complete!
15:53:12 INFO  [root::337] - PolymerSim2, package polymersim2.rules: Finished build.
15:53:13 INFO  [build.AutoBuildAwareJobQueueProcessor::49] - All jobs completed successfully.
```

If there are any errors, that report some missing *.ecore file or anything in that direction, please make sure every GT-File editor window is closed. Then close your dynamic workspace, reopen it and try to build your project again. Usually that should fix it, since Xtext produces this bug every time the project is being build while an editor window is open at the same time.

6. Example: Create your own SimSG simulation definion model

a. SimSG needs to know where to find your rules, models, which patterns to observe and when to terminate. A simulation definition model is used to store and convey that information. You could create it using the graphical editor, similar to creating dynamic instances of your metamodel, but it is much safer doing it programmatically.

b. The easiest way is to have a look at the SimpleSimulationGenerator.java file in the Polymers project inside your workspace, copy & paste it into your project and modify it to suit your needs.


Package Explorer
GoldbeterKoshland [SimSG-tests master]
org.simsg.applications.gklsimulation [Sim
org.simsg.applications.hybridgklsimulati
PolymerModel [SimSG-tests master]
> Polymers [SimSG-tests master]
src
polymers.rules
polymers.simulation
polymers.util.generator
SimpleSimulationGenerator.java
src-gen
gen
JRE System Library [jre]
Plug-in Dependencies
debug
> instances
META-INF
model
build.properties
SimpleSimulationGenerator.java

```java
package polymers.util.generator;

import org.eclipse.emf.common.util.URI;

public class SimpleSimulationGenerator {

    public static void main(String[] args) {
        PolymerModelGenerator polymersGen = new PolymerModelGenerator();
        polymersGen.setNumOfAtoms(10);
        polymersGen.generate(System.getProperty("user.dir")+"/instances/simulation_instances/10atoms_unconnected.xmi");

        SimulationDefinitionGenerator gen = new SimulationDefinitionGenerator("10atoms_unconnected");
        gen.setIBeXModel(URI.createFileURI(System.getProperty("user.dir")+"/src-gen/polymers/rules/api/ibex-patterns.xmi"));
        gen.setModelURI(System.getProperty("user.dir")+"/instances/simulation_instances/"+"10atoms_unconnected"+".xmi");
        gen.addPatternObservation("triangle");
        gen.addPatternObservation("cC2");
        gen.addPatternObservation("bC2");
        gen.addPatternObservation("createEdge");
        gen.addPatternObservation("deleteEdge");
        gen.addTerminationConditionIterations(1000);
        gen.saveDefinition(System.getProperty("user.dir")+"/instances/simulation_definitions/"+"10atoms_unconnected"+".xmi");
    }

}
```

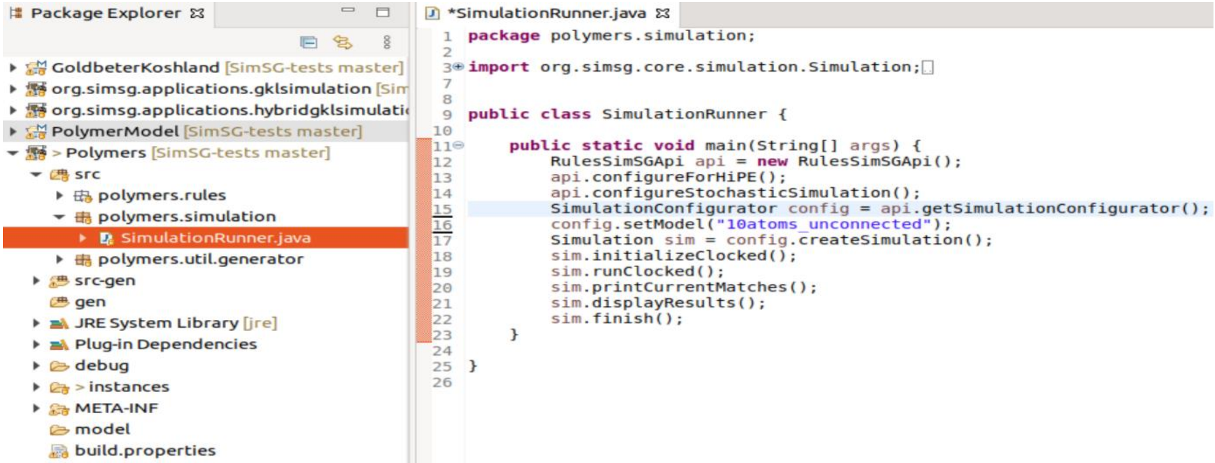
i. Line 11 creates a new model generator object, that creates a model instance of your metamodel of choice. In case of the example in the metamodeling tutorial, this would be your modified model generator (step 8) creating instances that conform to the Polymers2 metamodel.
ii. Line 12 sets the number of initially created Atoms to 10.
iii. Line 13 sets the path to where the new model instance is saved to. To avoid conflicts or errors, it is recommended to always store these models in the "instances/simulation_instances/" folder of your respective SimSG project.
iv. Line 15 creates a new SimulationDefinitionGenerator object, that is used to create new simulation definition instances. This generator is provided by SimSG, which means that you don't have to create one by yourself.
v. Line 16 sets the IBeX-Pattern location. These patterns are automatically generated from your GT-File and are always located in "src-gen/<name of the package that contains your GT-File, with dots replaced by slashes>/api/ibex-patterns.xmi".
vi. Line 17 sets the location of your desired model instance, e.g., the model created in lines 11 through 13.
vii. Lines 18 through 22 add the names of patterns to be observed. During the simulation, match populations of these patterns or rule preconditions will be tracked and can be displayed at the end of each simulation run.
viii. Line 23 sets the termination condition of your simulation. In this example the simulation will terminate after 1000 iterations. You may also use simulation time as a termination condition ***addTerminationConditionTime(double time)*** or use match counts of a certain pattern as a termination condition ***addTerminationConditionPattern(String patternName, int count)***.
ix. Line 24 sets the path to where the new simulation defintion instance is saved to. To avoid conflicts or errors, it is recommended to always store these models in the "instances/simulation_definitions/" folder of your respective SimSG project.

c. If you are checking-in your models to git and use your project cross-platform (e.g. Windows and Linux) you might receive path errors, since the generators generate and store full paths in your model. To avoid these errors, either refrain from checking-in your models or make sure to run your simulation definition generator beforehand, to make sure your paths are updated.

7. **Example: Run your new simulation**

   a. To run your simulation, you need to create a class where you call your SimSG api, configure the simulation, start it, and display your results.

   b. The easiest way is to have a look at the SimulationRunner.java file in the Polymers project inside your workspace, copy & paste it into your project and modify it to suit your needs.



```
1   package polymers.simulation;
2
3⊕  import org.simsg.core.simulation.Simulation;□
7
8
9   public class SimulationRunner {
10
11⊖      public static void main(String[] args) {
12          RulesSimSGApi api = new RulesSimSGApi();
13          api.configureForHiPE();
14          api.configureStochasticSimulation();
15          SimulationConfigurator config = api.getSimulationConfigurator();
16          config.setModel("10atoms_unconnected");
17          Simulation sim = config.createSimulation();
18          sim.initializeClocked();
19          sim.runClocked();
20          sim.printCurrentMatches();
21          sim.displayResults();
22          sim.finish();
23      }
24
25  }
26
```

      i. Line 12 creates a new SimSG Api object. The name of this class depends on the name of the package where your GT-File is located. In case of the PolymerSim2 example, it will be located in “src-gen/polymersim2.rules.api/” and will be called RulesSimSGApi.java

      ii. Line 13 configures SimSG to use our proprietary pattern matcher HiPE. This is recommended for maximum performance, since HiPE is massively parallelized.

      iii. Line 14 sets the simulation to stochastic mode, which makes use of your static rules rates within a continuous time Markov-chain. You can also chose to ignore static rules rates and simply let the simulation apply any random rule with matching preconditions, by selecting ***api.configureSimpleSimulation(false)***.

      iv. Line 15 generates a simulation configurator object.

      v. Line 16 configures the simulation to use your simulation definition of choice. As a parameter you must enter the name of the simulation definition model (step 6.b.iv) not its path.

      vi. Line 19 creates a simulation object.

      vii. Line 20 initializes the simulation. In this case the duration of the initialization phase will be measured and logged. If you do not want that to happen, please use ***sim.initlialize()***.

      viii. Line 21, finally, starts the simulation. In this case the duration of the simulation run will be measured and logged. If you do not want that to happen, please use ***sim.run()***.

      ix. Line 22 outputs match counts to the console.

      x. Line 23 plots your results. Here you will find the match counts of your observation patterns.

      xi. Line 24 cleans the state of your simulation and removes all matches.

   c. You can also save the final state of your model after the simulation finishes using ***sim.saveModelGraph()***.