Searching for Optimal Models: Comparing Two Encoding Approaches

Data Summary

This archive contains the raw files generated by MDEOptimiser and MOMoT and generated summaries.

Raw Data

Raw files are included in the `data` directory. One archive is included for each tool, with the tool name as the name of the archive.

Summaries

Generated data summaries are included in the `summary` directory.

Problem instance summaries

Each experiment has been repeated 30 times. We include a number of summaries for each of the runs executed:

- Average Steps
- Elasped Time
- Hypervolume (Single objective value for single objective problems)
- Median Hypervolume (Single objective value for single objective problems)

For each individual problem we include a set of statistics for the hypervolume (Single objective value for single objective problems).

Two plots are generated in the `plots` directory for each problem instance.

Overall summaries

We include overall summaries for each:

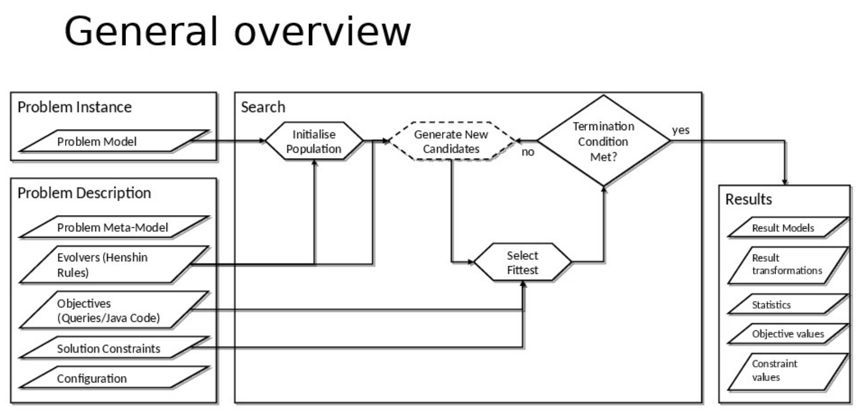
- Elapsed Time
- Statistical Testing Results (U, P, Cohen's D)

Tool architectures

In this section we include diagrams showing the overall architecture of MDEOptimiser and MOMoT.

General overview

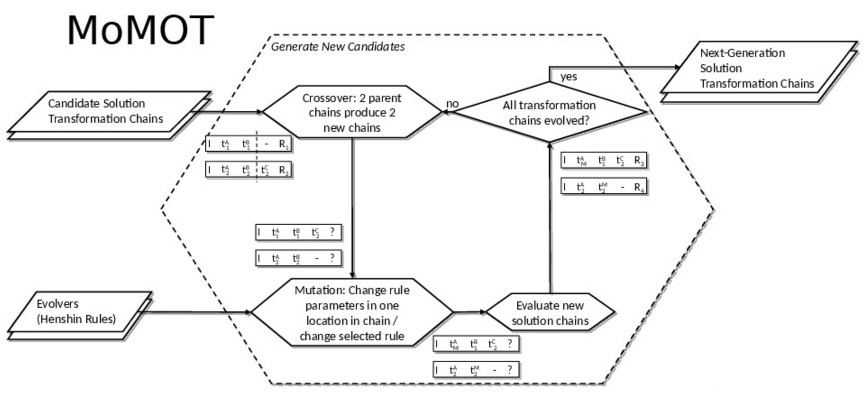
General overview diagram showing the similarities between the two tools. Both tools require the user to provide similar inputs through a DSL and return optimal models along with search results information.



Tool Architectures General overview

MOMoT

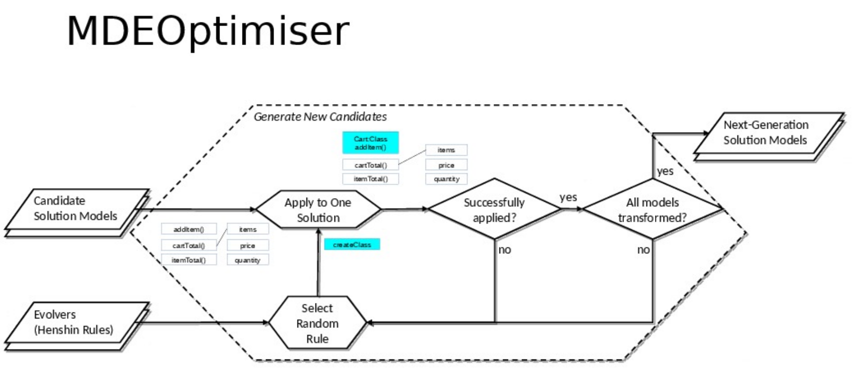
MOMoT tool architecture and solution encoding example.


MoMOT
Generate New Candidates
Candidate Solution Transformation Chains
Crossover: 2 parent chains produce 2 new chains
no
All transformation chains evolved?
yes
Next-Generation Solution Transformation Chains
Evolvers (Henshin Rules)
Mutation: Change rule parameters in one location in chain / change selected rule
Evaluate new solution chains

Tool Architecture MOMoT

MDEOptimiser

MDEOptimiser tool architecture and solution encoding example.



Tool Architecture MDEOptimiser

Problem Specifications

Problem specifications for all case studies have been included for both tools in the `specifications` folder.

The MDEOptimiser problem specification files are included in the `mdeo` folder. The MOMoT problem specification files are included in the `momot` folder.

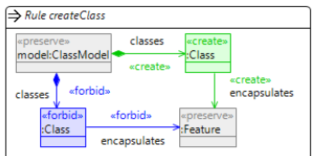
Henshin Mutation Operators

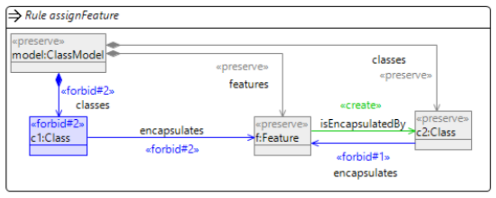
Henshin mutation operators are inlcuded in the `specifications\henshin` folder.

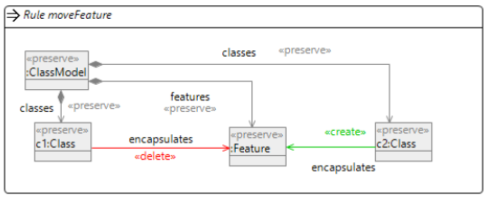
Below, we show graphical renderings of all rule sets for ease of reference.

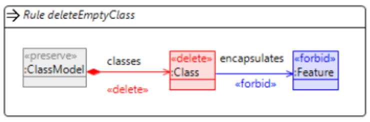
Class Responsibility Assignment

Mutation operators used for the CRA case study:

- 

- 

- 

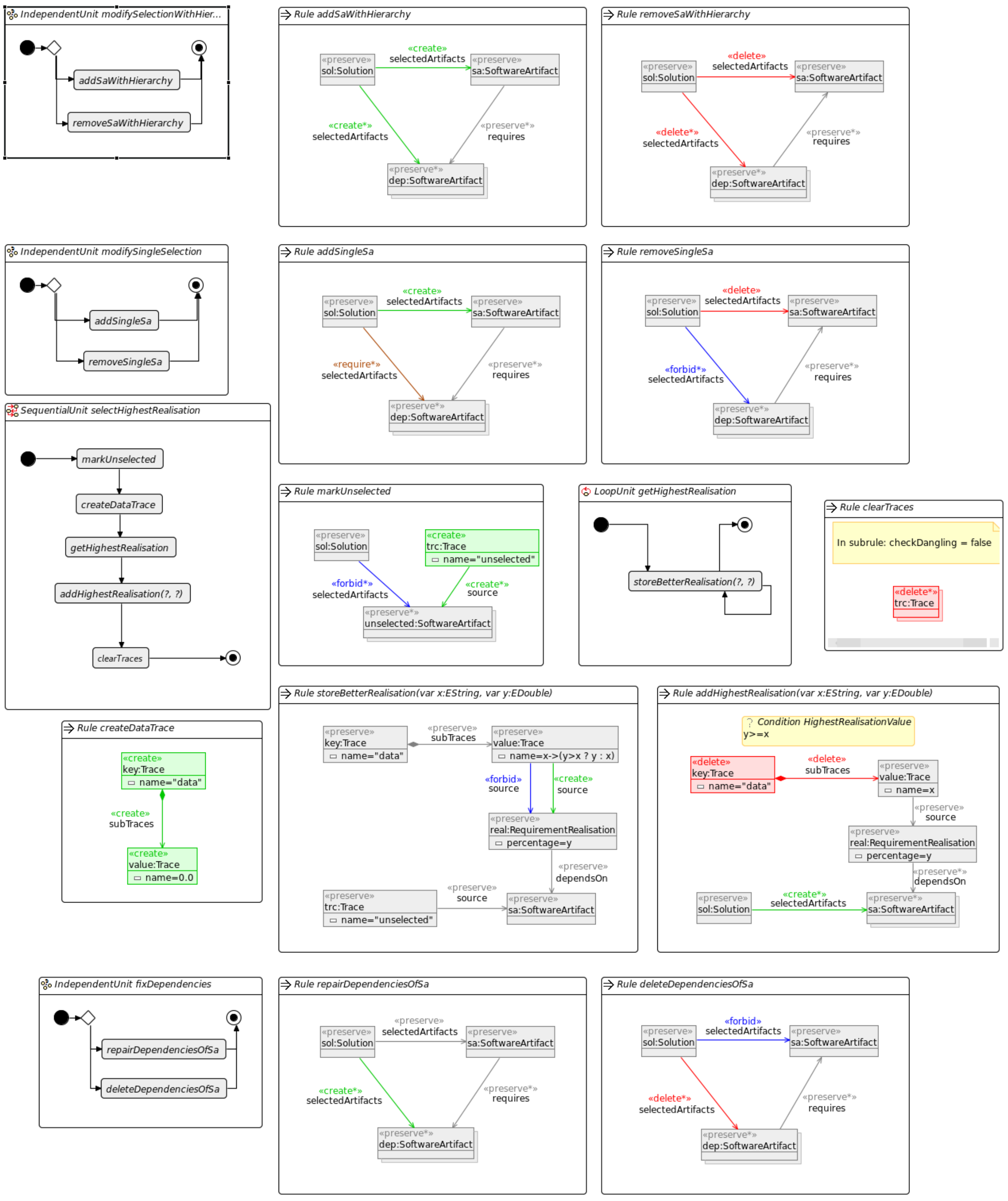
- 

Next Release Problem

Mutation operators used for the NRP case study. In this case study, the operators are encoded as Henshin units. The main operators used are the following:

- modifySelectionWithHierarchy
- modifySingleSelection
- selectHighestRealisation
- fixDependencies

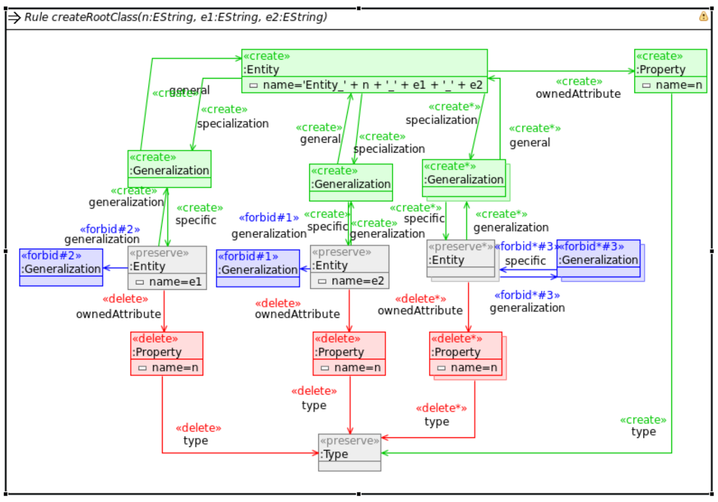
The rest of the transformations are helper transformations for the transformations used as operators.

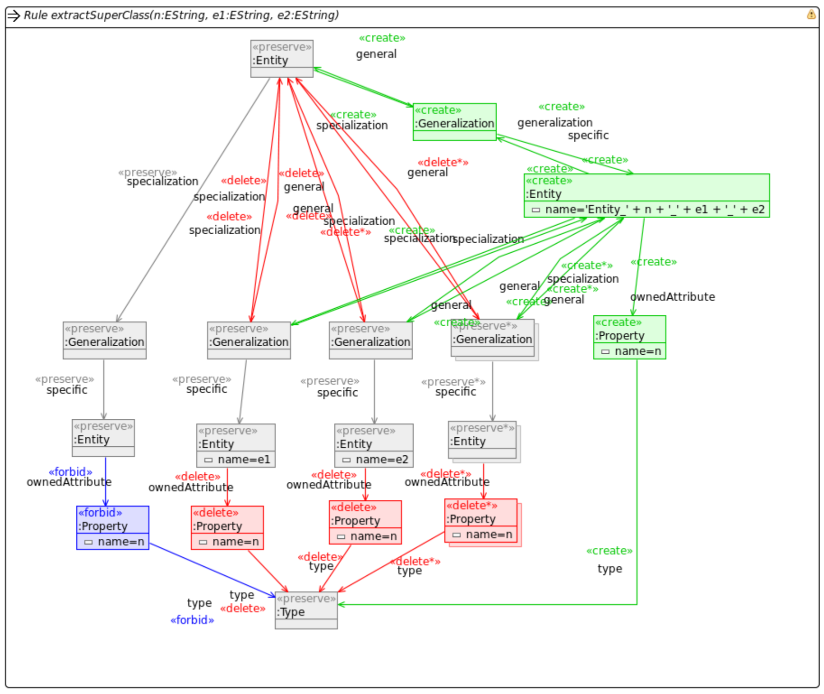

IndependentUnit modifySelectionWithHier...
addSaWithHierarchy
removeSaWithHierarchy
Rule addSaWithHierarchy
«preserve» sol:Solution
«create» selectedArtifacts
«preserve» sa:SoftwareArtifact
«create*» selectedArtifacts
«preserve*» requires
«preserve*» dep:SoftwareArtifact
Rule removeSaWithHierarchy
«delete» selectedArtifacts
«delete*» selectedArtifacts
IndependentUnit modifySingleSelection
addSingleSa
removeSingleSa
Rule addSingleSa
«require*» selectedArtifacts
Rule removeSingleSa
«forbid*» selectedArtifacts
SequentialUnit selectHighestRealisation
markUnselected
createDataTrace
getHighestRealisation
addHighestRealisation(?, ?)
clearTraces
Rule markUnselected
«create» trc:Trace
name="unselected"
«forbid*» selectedArtifacts
«create*» source
«preserve*» unselected:SoftwareArtifact
LoopUnit getHighestRealisation
storeBetterRealisation(?, ?)
Rule clearTraces
In subrule: checkDangling = false
«delete*» trc:Trace
Rule createDataTrace
«create» key:Trace
name="data"
«create» subTraces
«create» value:Trace
name=0.0
Rule storeBetterRealisation(var x:EString, var y:EDouble)
«preserve» key:Trace
name="data"
«preserve» subTraces
«preserve» value:Trace
name=x->(y>x ? y : x)
«forbid» source
«create» source
«preserve» real:RequirementRealisation
percentage=y
«preserve» dependsOn
«preserve» trc:Trace
name="unselected"
«preserve» source
«preserve» sa:SoftwareArtifact
Rule addHighestRealisation(var x:EString, var y:EDouble)
Condition HighestRealisationValue
y>=x
«delete» key:Trace
name="data"
«delete» subTraces
«preserve» value:Trace
name=x
«preserve» source
«preserve» real:RequirementRealisation
percentage=y
«preserve*» dependsOn
«preserve» sol:Solution
«create*» selectedArtifacts
«preserve*» sa:SoftwareArtifact
IndependentUnit fixDependencies
repairDependenciesOfSa
deleteDependenciesOfSa
Rule repairDependenciesOfSa
«preserve» selectedArtifacts
«create*» selectedArtifacts
«preserve*» requires
Rule deleteDependenciesOfSa
«forbid» selectedArtifacts
«delete*» selectedArtifacts
«preserve*» requires

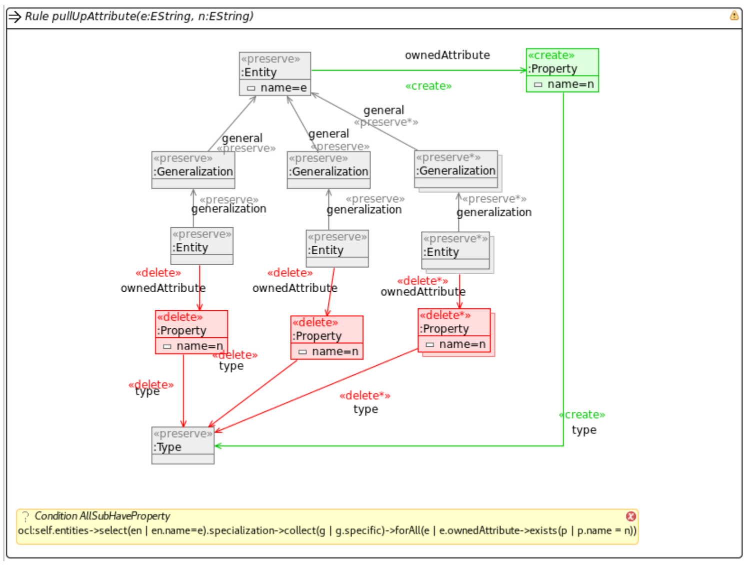
-

Refactoring

Mutation operators used for the Refactoring case study.







Condition AllSubHaveProperty

ocl:self.entities->select(en | en.name=e).specialization->collect(g | g.specific)->forAll(e | e.ownedAttribute->exists(p | p.name = n))