# Test strategy

We have written tests covering the following parts of our project:

* Meta model: Tested through dynamic model instances.
* Parser: Testing grammar syntax.
* Constraints: Testing syntax that satisfies/violates the constraint in question.
* Code generators: Testing that different elements returns expected generated code.

We have written unit-tests for each part, which are all based on a known initial state / input (i.e. a test bench with a fixed input), and a confirmation that the tested element returns the expected output.

We have written unit tests which validates valid input, or (correctly) invalidates invalid input. Thus, we have both positive and negative test cases.

We have focused on making each unit test as small as possible, in order to give a detailed overview of the test results. This gives a clear indication for any possible test errors.

We are aware that unit tests cannot stand alone as a full test of the developed feature. An easy way extend the system tests would be to perform a compilation of the generated code (if the generated code needs to be compiled), subsequently performing an exploratory test of the final application which the user sees.

/

# Metamodel test case examples

**class** TestConstraints {

**def** **static** **dispatch** constraint(Configurator **it**) {

!parameters.empty && !name.empty

}

**def** **static** **dispatch** constraint(Parameter **it**) {

!name.empty

}

**def** **static** **dispatch** constraint(ParameterGroup **it**) {

!parameters.empty && !name.empty

}

**def** **static** **dispatch** constraint(Bounded **it**) {

**var** lBound = (lowerBound **as** dk.itu.smdp2015.church.model.configurator.Integer)

**var** uBound = (upperBound **as** dk.itu.smdp2015.church.model.configurator.Integer)

lBound.value < uBound.value

}

....

// Fallback

**def** **static** **dispatch** constraint(EObject **it**) {

**true**

}

}

# Grammar test case examples

**package** dk.itu.smdp2015.church.configurator.syntax.tests

// Imports removed

@RunWith(XtextRunner)

@InjectWith(ConfiguratorInjectorProvider)

**class** ConfiguratorGrammarTest {

@Inject **extension** ParseHelper<Configurator>

@Inject **extension** ValidationTestHelper

@Before

**def** **void** before() {

ConfiguratorPackage.*eINSTANCE*.eClass

}

@Test

**def** **void** testInvalidModelNoParameters() {

**var** model = '''configurator Empty'''.parse

model.assertError(ConfiguratorPackage.Literals.*CONFIGURATOR*, Diagnostic.*SYNTAX\_DIAGNOSTIC*, "mismatched input")

}

@Test

**def** **void** testValidBoundedRange() {

**var** model = '''configurator Bicycle "Bicycle configuration" { parameter wheel\_size values [16;24] }'''.parse

*assertEquals*("Bicycle configuration", model.description)

**var** param = model.parameters.get(0) **as** Parameter

*assertEquals*("wheel\_size", param.name)

**var** valueRange = param.valueRange **as** Bounded

*assertEquals*(16, (valueRange.lowerBound **as** IntegerImpl).value)

*assertEquals*(24, (valueRange.upperBound **as** IntegerImpl).value)

model.assertNoErrors

}

@Test

**def** **void** testInvalidBoundedRangeReverse() {

**var** model = '''configurator Bicycle { parameter wheel\_size values [24;16] }'''.parse

model.assertError(ConfiguratorPackage.Literals.*BOUNDED*, ConfiguratorValidator.*INVALID\_BOUND*, "Lower bound should be less than upper bound")

}

@Test

**def** **void** testInvalidBoundedRangeWrongTypes() {

**var** model = '''configurator Bicycle { parameter wheel\_size values [b;16] }'''.parse

model.assertError(ConfiguratorPackage.Literals.*BOUNDED*, ConfiguratorValidator.*WRONG\_TYPE*, "expected the same type")

}

@Test

**def** **void** testValidEnum() {

**var** model = '''configurator Car { parameter Variant values ("Standard", "Sport", "Luxury") }'''.parse

*assertNull*(model.description)

**var** param = model.parameters.get(0) **as** Parameter

*assertEquals*("Variant", param.name)

*assertFalse*(param.optional)

**var** enumerated = param.valueRange **as** Enumerated

*assertEquals*(3, enumerated.values.*length*)

*assertEquals*("Standard", (enumerated.values.get(0) **as** StringImpl).value)

*assertEquals*("Sport", (enumerated.values.get(1) **as** StringImpl).value)

*assertEquals*("Luxury", (enumerated.values.get(2) **as** StringImpl).value)

model.assertNoErrors

}

@Test

**def** **void** testInvalidEnumWrongElement() {

**var** model = '''configurator Car { parameter Variant values ("Standard", "Sport", Luxury) }'''.parse

model.assertError(ConfiguratorPackage.Literals.*ENUMERATED*, ConfiguratorValidator.*INVALID\_ENUMERATION*, "Enumerated item should be a constant")

}

// .....

}

# Code generators test case examples

[TODO]