Testing DSLs
How to test DSLs, their IDEs and Programs

Markus Voelter
independent/itemis
voelter@acm.org

www.voelter.de
voelterblog.blogspot.de
@markusvoelter
+Markus Voelter
A DSL is a focussed, processable language for describing a specific concern when building a system in a specific domain. The abstractions and notations used are natural/suitable for the stakeholders who specify that particular concern.
Limited Expressiveness.
Reduced Need For Tests.
Constraint Checks.
A Form of Test.
Test Semantics, not Structure!

Example Models ➞ Generator ➞ Code

Based On

Test Cases ➞ Tests ➞ Binary
This tests only the generators!
Separate test models and generated test code

Model → Generator → Code

based on

Test Model (Test Language) → Generator → Test Code

tests
Separate test models and generated test code

Model — Generator — Code

based on

Test Model (Test Language)

Generator — Test Code

Mocks

tests
Testing the Language

Testing Programs
Does the DSL cover the domain it is intended to cover?

Build Examples and discuss with Stakeholders.
Interview Experts

Get Feedback

Create Examples

Build the Language

Structure their Knowledge
Can the language definition process all of the relevant notation? (Parser Test)

Create and maintain example programs and keep trying to parse them
@RunWith(XtextRunner.class)
@InjectWith(Expr2DslInjectorProvider.class)
public class TestModels extends XtextTest {

    @Test
    public void testBasics() {
        testFileNoSerializer("1-basic.expr2");
    }
}

http://code.google.com/a/eclipselabs.org/p/xtexxlim/utils/wiki/Unit_Testing
```java
@RunWith(XtextRunner2.class)
@InjectWith(DomainmodelInjectorProvider.class)
public class ParserAndLexerTest extends XtextTest {
    @Test
    public void id() {
        testTerminal("bar", /* token stream: */ "ID");
        testTerminal("bar3", /* token stream: */ "ID");
        testTerminal("_bar_", /* token stream: */ "ID");
        testTerminal("$bar$", /* token stream: */ "ID");
        testNotTerminal("3bar", /* unexpected */ "ID");
        testNotTerminal("#bar", /* unexpected */ "ID");

        // token streams with multiple token
        testTerminal("foo.bar", "ID", ".", "ID");
        testTerminal("foo.*", "ID", ".", "*.", ".*");
    }

    @Test
    public void qualifiedName() {
        testParserRule("foo.bar", "QualifiedName");
        testParserRuleErrors("3foo.bar", "QualifiedName", "extraneous input '3'"avanaugh
    }

    @Test
    public void qualifiedNameWithWildcard() {
        testParserRule("foo.*", "QualifiedNameWithWildCard");
    }
}
```

http://code.google.com/a/eclipselabs.org/p/xtext-utils/wiki/Unit_Testing
Do the scopes work?
Do the constraints and typesystem work?
Are there any error messages associated with certain program elements?

Create positive and negative examples and check for correct error annotations.
```java
tested model basic

var int a = 1;
var int b = 2;
var int c = 3;

calc int x = 1;
calc int y = a + c;
calc int z = a * a + plus(1,2);

assert a is 1
assert b is 2
assert x is 1
assert y is 4
assert z is 4

function int plus(int a, int b) {
    return a+b;
}

@Test
public void testBasics() {
    testFileNoSerializer("1-basic.expr2");
    assertConstraints( issues.errorsOnly().sizeIs(0) );
}
```
@Test
public void testTArray() throws Exception {
    testFileNoSerializer("4-array.expr2");
    assertConstraints( issues.errorsOnly().sizeIs(1) );
    assertConstraints( issues.forType(Formula.class).named("anotherOne2").
                        theOneAndOnlyContains("incompatible") );
}
<table>
<thead>
<tr>
<th>The method...</th>
<th>returns a new IssueCollection that...</th>
</tr>
</thead>
<tbody>
<tr>
<td>forType( t )</td>
<td>contains only those issues that are attached to an instance of t</td>
</tr>
<tr>
<td>get( index )</td>
<td>are at position index in the IssueCollection</td>
</tr>
<tr>
<td>inLine( line )</td>
<td>are in line line in the model file</td>
</tr>
<tr>
<td>withStringFeatureValue( n, v )</td>
<td>whose feature named n has the value (toString()) v</td>
</tr>
<tr>
<td>errorsOnly()</td>
<td>contains no warnings</td>
</tr>
<tr>
<td>named( n )</td>
<td>contains only those issues that are attached to an element with name property value n</td>
</tr>
<tr>
<td>forElement( t, n )</td>
<td>contains only those issues that are attached to elements that have the name n</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>The method...</th>
<th>asserts that...</th>
</tr>
</thead>
<tbody>
<tr>
<td>sizeIs( s )</td>
<td>the size of the current collection is s</td>
</tr>
<tr>
<td>oneOfThemContains( t )</td>
<td>the collection has any size, and one of the error messages contains the substring t</td>
</tr>
<tr>
<td>allOfThemContain( t )</td>
<td>the collection has any size, and all the error messages contains the substring t</td>
</tr>
<tr>
<td>theOneAndOnlyContains( s( t )</td>
<td>the collection is of size 1 and the message of the single error contains the substring t</td>
</tr>
</tbody>
</table>
```java
@Check()
public void runAssertStatements(Model m) {
    if (!m.isIsTested()) return;
    try {
        MessageList errors = new ExprModelInterpreter().runModel(m, ts);
        for (MessageList.MessageItem o: errors.getMessages()) {
            error(o.getMessage(), o.element, null, -1, INTERPRETERFAILED);
        }
    } catch (InterpreterException e) {
        if (e.getFailedObject() != null) {
            error(e.getMessage(), e.getFailedObject(), null, -1, INTERPRETERFAILED);
        } else {
            error(e.getMessage(), m, null, -1, INTERPRETERFAILED);
        }
    } e.printStackTrace();
}
```
module Dummy from tests.ts.core.tests1 imports nothing {

    void dummy() {
        int8_t[2] andererArray = {10, 20};
        int8_t[3] intarr = {10, 20, 30};
        int8_t[2] two = {1, 2, 3};
        float[2] fa = {1, 2};
        int8_t[2] ia = {1.1, 2.1};
        int8_t* intp = andererArray;
        int8_t* intp2 = intarr;
        int8_t[ ] xx = {1, 2.1, "Hello"};
        int8_t x = xx[2];
        float ff = xx[2];
        int8_t fi = fa[2];

    } dummy (function)
}

Test case Arrays
nodes

```c
(void dummy()
{
    int8_t[2] andererArray = {10, 20};
    int8_t[3] intarr = {10, 20, 30};
    int8_t[2] two = <node {1, 2, 3} has error>;
    float[2] fa = {1, 2};
    int8_t[2] ia = <node {1.1, 2.1} has error>;
    int8_t* intp = andererArray;
    intp = <node fa has error>;
    int8_t* intp2 = intarr;
    int8_t[ ] xx = <node {1.2.1, “Hello”} has error>;
    int8_t x = xx[2];
    float ff = xx[2];
    int8_t fi = <node fa[2] has error>;
}
}
```
Test case Arrays

nodes

<check types module Dummy from tests.ts.core.tests1 imports nothing { > }

exported test case arrayAccessPrioTest {
  int8_t[2] array = {10, 20};
  <arrayAccessPrioTest1 array[1]>;
  <arrayAccessPrioTest2 array[1] * 2>;
  int8_t i = 2;
  <arrayAccessPrioTest3 array[1] * i++>;
  <arrayAccessPrioTest4 -array[1]>;
}
arrayAccessPrioTest(test case)

test methods
test prioTestArrayAccess {
  assert true arrayAccessPrioTest1.isArrayOf(ArrayAccessExpr);
  assert true arrayAccessPrioTest2.isArrayOf(MultiExpression);
  assert true arrayAccessPrioTest2.left.isArrayOf(ArrayAccessExpr);
  assert true arrayAccessPrioTest2.right.isArrayOf(NumberLiteral);
  assert true arrayAccessPrioTest3.isArrayOf(MultiExpression);
  assert true arrayAccessPrioTest3.left.isArrayOf(ArrayAccessExpr);
  assert true arrayAccessPrioTest3.right.isArrayOf(PostIncrementExpression);
  assert true arrayAccessPrioTest4.isArrayOf(UnaryMinusExpression);
  assert true arrayAccessPrioTest4.expression.isArrayOf(ArrayAccessExpr);
}
Do the generators, transformations or interpreters work?

Run them; execute test cases against the running code.
You can write test cases in the target language manually.

Or you can express test cases on the model level.
```prolog
prolog {
    set RC->accumulatedRuntime = 80
}

step 10
assert-currentstate-is noCooling

mock: set RC->accumulatedRuntime = 110

step
mock: set RC.rceva->evaTemp = 10
assert-currentstate-is abtauen
assert-value cc.c1->active is false

mock: set RC->accumulatedRuntime = 0
step 5
assert-currentstate-is abtauen
assert-value cc.c1->active is false
step 15
assert-currentstate-is noCooling
```

```prolog
parameter t_abtaustart: int
parameter t_abtaudauer: int
parameter T_abtauEnde: int

var tuerNachlaufSchwelle: int = 0

start:
    entry { state noCooling }

state noCooling:
    check ( (RC->needsCooling) && (cc.c1->steh) 
        state rccooling
    )

    on isDown ( RC.rcdoor->open ) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        perform rcfanabschalttask after 10 {
            set RC.rcfan->active = false
        }
    }

state rccooling:
    entry { set RC.rcfan->active = true }
    check ( !(RC->needsCooling) ) {
        state noCooling
    }

    on isDown ( RC.rcdoor->open ) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        set tuerNachlaufSchwelle = currStep + 30
    }

    exit {
        perform rcfanabschalttask after max( 5, tuerNachlaufSchwelle-currStep ) {
            set RC.rcfan->active = false
        }
    }
```
Interpreting the tests directly in the IDE reduced turn around time.
## Simulation View

### Status
- Current Test: KIRAAbtauen
- Current State: 
- Current Step: 

### Control
- Autorun
- Single Step
- Enable Breakpoints

### Property Values
<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC.accumulatedRun</td>
<td>80</td>
</tr>
<tr>
<td>RC.needsCooling</td>
<td>false</td>
</tr>
<tr>
<td>c1.active</td>
<td>false</td>
</tr>
<tr>
<td>ccfan.active</td>
<td>false</td>
</tr>
<tr>
<td>rcdoor.open</td>
<td>false</td>
</tr>
<tr>
<td>rceva.evaTemp</td>
<td>20</td>
</tr>
<tr>
<td>rcfan.active</td>
<td>false</td>
</tr>
</tbody>
</table>

### Queue
- Event
- Data

### Commands
- St...
- Command

### Variable Values
<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tuerNachlaufSchwel...</td>
<td>0</td>
</tr>
</tbody>
</table>

### Assert Selected Variable

### Running Tasks
- Task
- Sinc...
<table>
<thead>
<tr>
<th>Name</th>
<th>Documentation</th>
<th>Tags</th>
<th>Valid time</th>
<th>Transaction time</th>
<th>Fixture</th>
<th>Product</th>
<th>Element</th>
<th>Expected value</th>
<th>Actual value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accrued right at retireme</td>
<td></td>
<td></td>
<td>2006-12-31</td>
<td>2007-9-24</td>
<td>Jan De Jong</td>
<td>Old Age Pension</td>
<td>Accrued right</td>
<td>761.0402</td>
<td>761.0402</td>
</tr>
<tr>
<td>Accrued Right last final pay</td>
<td></td>
<td></td>
<td>2004-1-1</td>
<td>2007-9-24</td>
<td>Jan De Jong</td>
<td>Old Age Pension</td>
<td>Accrued right</td>
<td>705.0589</td>
<td>705.0589</td>
</tr>
<tr>
<td>premium last year</td>
<td></td>
<td></td>
<td>2006-1-1</td>
<td>2007-9-24</td>
<td>Jan De Jong</td>
<td>Old Age Pension</td>
<td>Premium old age pension</td>
<td>329.0625</td>
<td>329.0625</td>
</tr>
<tr>
<td>Accrued right at retireme 2</td>
<td></td>
<td></td>
<td>2006-12-31</td>
<td>2007-9-24</td>
<td>Piet Van Dijk</td>
<td>Old Age Pension</td>
<td>Accrued right</td>
<td>740.94</td>
<td>724.7658</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1985-12-31</td>
<td>2007-9-24</td>
<td>Jan De Jong</td>
<td>Old Age Pension</td>
<td>Accrued Right in service period</td>
<td>73.661</td>
<td>73.661</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1985-12-31</td>
<td>2007-9-24</td>
<td>Jan De Jong</td>
<td>Old Age Pension</td>
<td>Years of service in service period</td>
<td>3.7534</td>
<td>3.7534</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1987-12-31</td>
<td>2007-9-24</td>
<td>Jan De Jong</td>
<td>Old Age Pension</td>
<td>Pension base average FP</td>
<td>7750</td>
<td>7750</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1998-12-31</td>
<td>2007-9-24</td>
<td>Jan De Jong</td>
<td>Old Age Pension</td>
<td>Accrued Right in service period</td>
<td>387.7449</td>
<td>387.7449</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1998-12-31</td>
<td>2007-9-24</td>
<td>Jan De Jong</td>
<td>Old Age Pension</td>
<td>Years of service in service period</td>
<td>10.8082</td>
<td>10.8082</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1998-12-31</td>
<td>2007-9-24</td>
<td>Jan De Jong</td>
<td>Old Age Pension</td>
<td>Pension base average FP</td>
<td>8250</td>
<td>8250</td>
</tr>
</tbody>
</table>
exported test case logicalOperators {
    assert(0) (true || true) == true;
    assert(1) (true || false) == true;
    assert(2) (false || false) == false;
    assert(3) (true && true) == true;
    assert(4) (true && false) == false;
    assert(5) (false && false) == false;
    assert(6) (true && false || true) == true;
    assert(7) (true && false || false) == false;
} logicalOperators(test case)
module functionPointerStuff from test.ex.core.pointers imports nothing {

  var (int8_t, int8_t)=>(int8_t) globalRef;

typedef (int8_t, int8_t)=>(int8_t) as twoIntArgsReturningInt;

int8_t add(int8_t x, int8_t y) {
    return x + y;
} add (function)

exported test case testFC {
    globalRef = :add;
    int8_t res = globalRef(1, 2);
    assert(0) res == 3;

    (int8_t, int8_t)=>(int8_t) localRef;
    localRef = :add;
    int8_t localres = localRef(10, 22);
    assert(1) localres == 32;
    assert(2) -anotherHOF(:add) == -9;
    assert(3) higherOrderFunction(globalRef) == 3;
    assert(4) higherOrderFunction(:add) == 3;
    assert(5) anotherHOF(:add) == 9;

} testFC(test case)

int8_t higherOrderFunction((int8_t, int8_t)=>(int8_t) fun) {
    return fun(1, 2);
} higherOrderFunction (function)

int8_t anotherHOF(twoIntArgsReturningInt fun) {
    return fun(4, 5);
} anotherHOF (function)
Multiple Mappings
... at the same time

$L_D$

$L_x \quad L_y \quad L_z$

Similar Semantics?
Multiple Mappings
... at the same time

\[ L_D \]

Similar Semantics?

all green!
Multiple Mappings

... alternatively, selectably

Extend $L_D$ to include explicit data that determines transformation

$L_D$
Multiple Mappings
... alternatively, selectably

TESTING!
Generators:
What do I do if I cannot execute the programs?
Generators:
What do I do if I cannot execute the programs?

Then (and only then!) use structure analysis
Does the actual editor perform its services correctly?

This is highly tool specific...
Editor test case InitializingNewVariable

description: no description

before: Variables:

<cell var <no name> : <no alias>>

result: Variables:

  var myVariable : boolean

code:

  type "myVariable"
  press keys <any>+<VK_TAB>;
  press keys <ctrl>+<VK_SPACE>;
  type "boo"
  press keys <any>+<VK_ENTER>;}
Moritz Eysholdt’s Xpect
Can I do proofs, and not just "trial and error" testing?

Yes. Formal Methods.
Model Checking

Model

Property Specifications

Is the model correct?

This is what I mean by „correct“

Tool

OK

COUNTER EXAMPLE

Out Of Memory
Challenges

Model

+ 

Property Specifications

Formalism

Input Language

Property Language

Tool

Algorithm

OK

COUNTER EXAMPLE

Out Of Memory

Interpretation
State Based

Model

+ 

Property Specifications

State Machines

NuSMV

LTL / CTL

Tool

„Magic“

OK

COUNTER EXAMPLE

Out Of Memory
mbeddr Approach

- High-Level State Machine
- High-Level Properties

---

NuSMV Model

- generate

- CTL

---

NuSMV

highlight errors

---

Text File
mbeddr Approach

easier to use

hopefully used more

full power: write CTL/LTL if you want to
Model Checking

module TrafficLights from trafficlights imports nothing

```plaintext
verifiable
statemachine TrafficLights {
    in timePassed() <no binding>
    pedestrianButtonPressed() <no binding>
    out << ... >>
    vars int[0..2] pedLights = 0
    int[0..2] carLights = 0
    states (initial = bothRed)
    state bothRed {
        on timePassed [ ] -> carsGreen {
            carLights = 2;
            pedLights = 0;
        }
    }
    state carsGreen {
        on timePassed [ ] -> pedGreen {
            carLights = 0;
            pedLights = 2;
        }
    }
    state pedGreen {
        on timePassed [ ] -> carsGreen {
            carLights = 0;
            pedLights = 2;
        }
    }
} end statemachine
```

verification conditions
never carLights == 2 && pedLights == 2
module TrafficLights from trafficlights imports nothing

verifiable

state machine TrafficLights {
    in timePassed() < no binding>
        pedestrianButtonPressed() < no binding>
    out <<< >>
    vars Int[0..2] pedLights = 0
    Int[0..2] carLights = 0
    states (initial = bothRed)
        state bothRed {
            on timePassed [ ] -> carsGreen {
                carLights = 2;
                pedLights = 0;
            }
        }
        state carsGreen {
            on timePassed [ ] -> pedGreen {
                carLights = 0;
                pedLights = 2;
            }
        }
        state pedGreen {
            on timePassed [ ] -> carsGreen {
                carLights = 2;
                pedLights = 2;
            }
        }
    end state machine

inspectors

com.mbeddrt.ostatemachines.structuredState

[states] State "pedGreen"[1162046401483365036] in trafficLights

- State "bothRed" can be reached
- State "carsGreen" can be reached
- State "pedGreen" can be reached
- Variable 'pedLights' is always between its....
- Variable 'carsLights' is always between its....
- State "bothRed" has deterministic transitions
- State "carsGreen" has deterministic transitions
- State "pedGreen" has deterministic transitions
- Transition 0 of state 'bothRed' is not dead
- Transition 0 of state 'carsGreen' is not dead
- Transition 0 of state 'pedGreen' is not dead

- Condition 'carLights' >= 2 && 'pedLights' >= 2... FAIL
Model Checking

```plaintext
module TrafficLights from trafficlights imports nothing

verifiable

statemachine TrafficLights {
  in timePassed() <no binding>
  pedestrianButtonPressed() <no binding>
  out << ... >>
  vars int[0..2] pedLights = 0
  int[0..2] carLights = 0
  states (initial = bothRed)
    state bothRed {
      on timePassed [] -> carsGreen {
        carLights = 2;
        pedLights = 0;
      }
    }
    state carsGreen {
      on timePassed [] -> pedGreen {
        carLights = 0;
        pedLights = 2;
      }
    }
    state pedGreen {
      on timePassed [] -> carsGreen {
        carLights = 2;
        pedLights = 0;
      }
    }
  }
}
```

---

http://mbeddr.com
Model Checking

Finds problems in state machines.

... even ones you didn’t think of!

Much more complete than manual testing.
Model Checking

Systems and Software Verification
Model-Checking Techniques and Tools
mbeddr Future

Higher Level, Non-State Machine DSL

High-Level State Machine

High-Level Properties

NuSMV Model

CTL

NuSMV

Text File
SAT Solving for
PLE Variability
Decision Table
Completeness
Executing a Program using all possible values and execution paths at the same time
Abstract Execution

\{ [1..3] \} + \{ [10..15], 30 \} = \\
\{ [11..18], [31..33] \}
Abstract Execution

Klaus Birken
Abstract Execution

Klaus Birken

```
a := { [7..20] }

T

a < 13 ?

F

b := a

a := { [7..12] }  

b := 2011

b := { [7..12], 2011 }
```
Abstract Execution

Klaus Birken

- **customer requirements**
  - validated against / based on
  - validated against / based on

- **software model**
  - online validation with abstract execution

- **abstract testcases model**
Abstract Execution

---

**customer requirements**
- validated against / based on
  - software model
    - online validation with abstract execution
    - transformation
    - (generated) system
      - automated black-box regression tests
  - abstract testcases model
    - transformation
    - testcases
Abstract Execution

Klaus Birken

Test for implementation of Audio_Routing_Control service

Abstract parameter specification

Specify reaction of underlying subsystem (with timings!)

Expected test result: positive response with $0 \leq \$Result < 100$

Warning: implementation will respond with concrete value
Abstract Execution

Klaus Birken

```cpp
266
267  /**
268  * Implementing service 0x2F_1210_03_Audio_Routing_Control
269  */
270
271 service 0x2F_1210_03_Audio_Routing_Control Audio_Routing_Control
272 {
273     action
274     {
275         controlAudioRouting : dsirequest DAVTAudioController.DAVTAudioDiagnosis.controlAudioRouting
276             {
277                 audioInput = cast<UInt8>($Input);
278                 audioOutput = cast<UInt8>($Output);
279             }
280         // add further event handlers and functions here
281     on controlAudioRouting
282     {
283         if ( #responseCode == DAVTAudioDiagnosis.EResponseCode::eOK )
284             { $Result = cast<Audio_Routing_Result>(#audioResult); posResponse; }
286         elseif ( #responseCode == DAVTAudioDiagnosis.EResponseCode::eSERVICE_NOT_SUPPORTED_IN_DIAG )
288             { negResponse (0x80); }
290         else {
292             negResponse (0x72); }
293     }
294 }
```
fully explore the state space
with abstract test cases

maximise coverage,
increase confidence,
reduce risk

tightly integrate development and test

iteratively develop implementation
and test cases, each uncovering incompleteness in the other
The End.

This material is part of my upcoming (early 2013) book

**DSL Engineering with Language Workbenches**

Stay in touch; it will be cheap or maybe even free :-)  

[www.voelter.de/dslbook](http://www.voelter.de/dslbook)

@markusvoelter  
+Markus Voelter