DSLs
in the context of
Agile

Markus Voelter
independent/itemis
voelter@acm.org

www.voelter.de
voelterblog.blogspot.de
@markusvoelter
+Markus Voelter

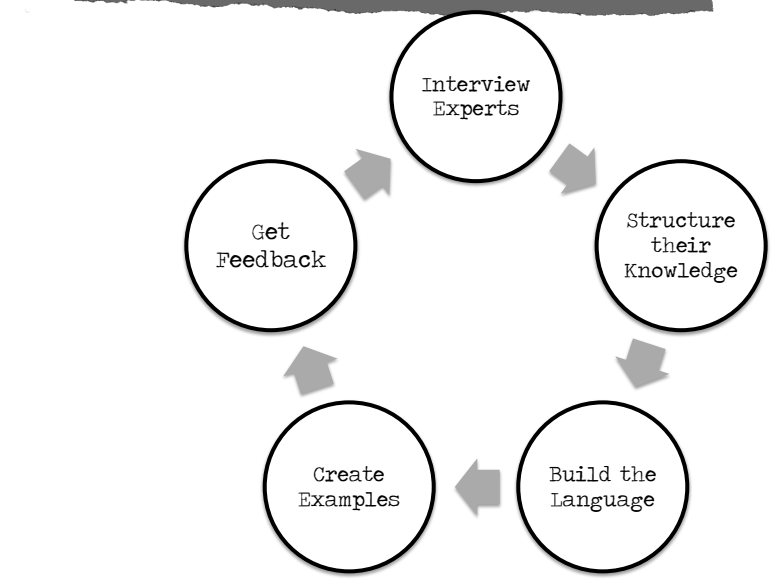
Introduction

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.

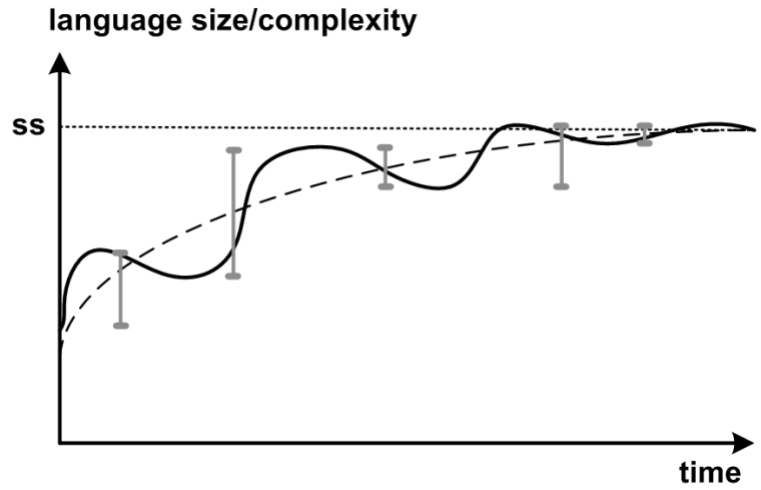
	more in GPLs	more in DSL
Domain Size	large and complex	smaller and well-defined
Designed by	guru or committee	a few engineers and domain experts
Language Size	large	small
Turing-completeness	almost always	often not
User Community	large, anonymous and widespread	small, accessible and local
In-language abstraction	sophisticated	limited
Lifespan	years to decades	months to years (driven by context)
Evolution	slow, often standardized	fast-paced
Incompatible Changes	almost impossible	feasible

Process

Domain Analysis



Iterate to goal

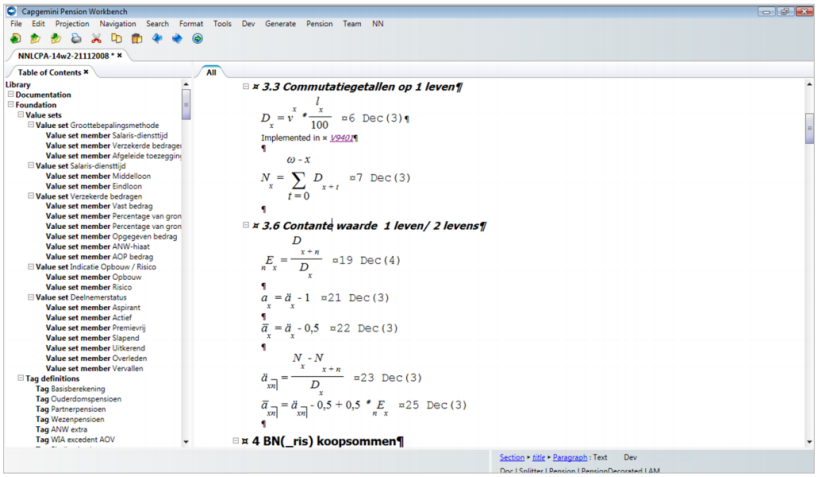


Documentation

Create example-based tutorials!

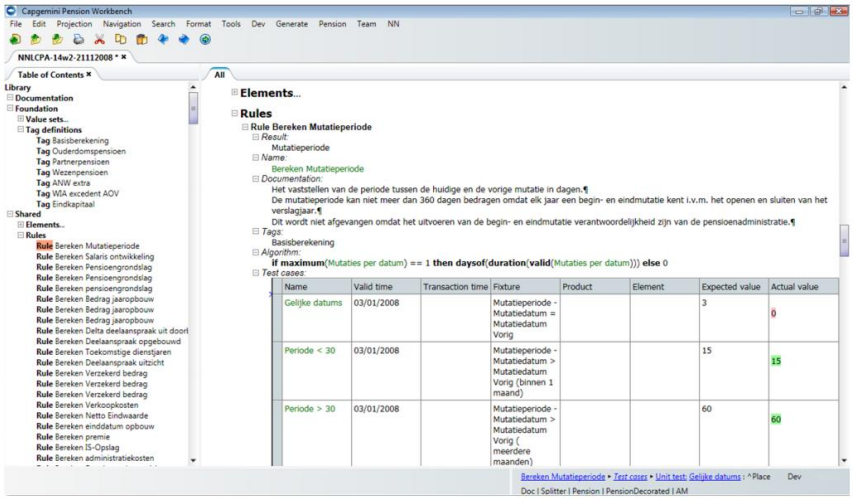
Domain Folks Programming?

Precision vs. Algorithmics!



Example

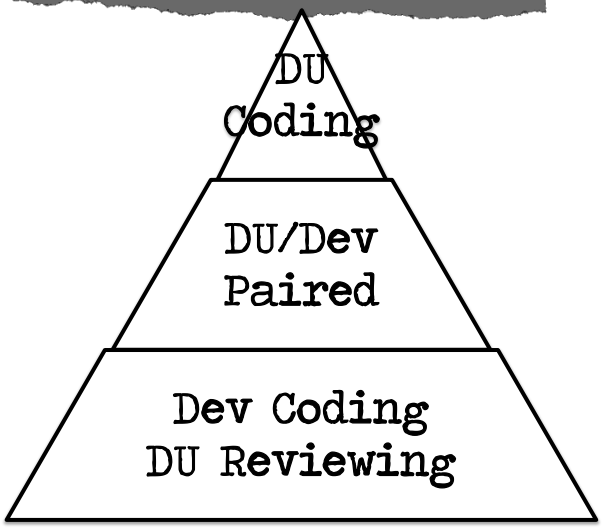
Pension Plans



Example

Pension Plans

Domain Folks Programming?

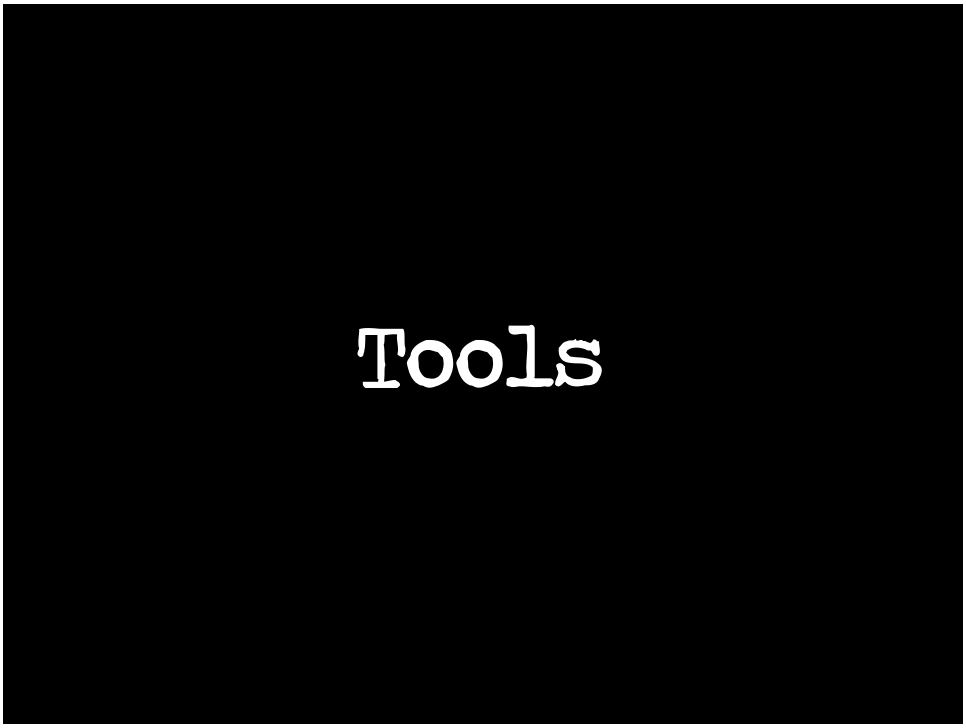


DSL as a Product

Release Plan
Bug Tracker
Testing!
Support
Documentation

Reviews

Reviews become easier --- less code, more domain-specific


Tools

Language Workbenches

Define, use, evolve and debug a set of integrated langauges

xtext

Language Workbenches

Define, use, evolve and debug a set of integrated langauges

xtext

http://languageworkbenches.net

Expressivity

Shorter
Programs

More
Accessible
Semantics

For a limited
Domain!

Domain Knowledge
encapsulated in
language

Smaller Domain

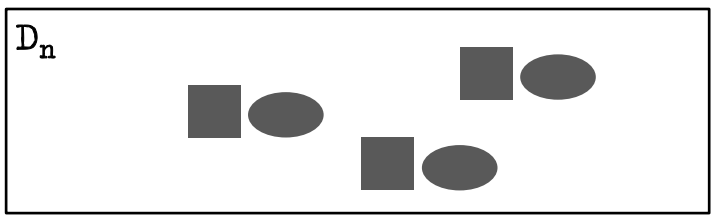
⇩

More Specialized Language

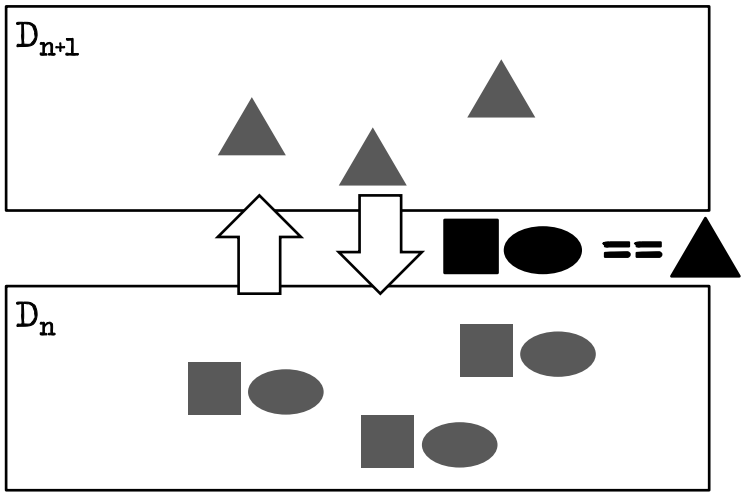
⇩

Shorter Programs

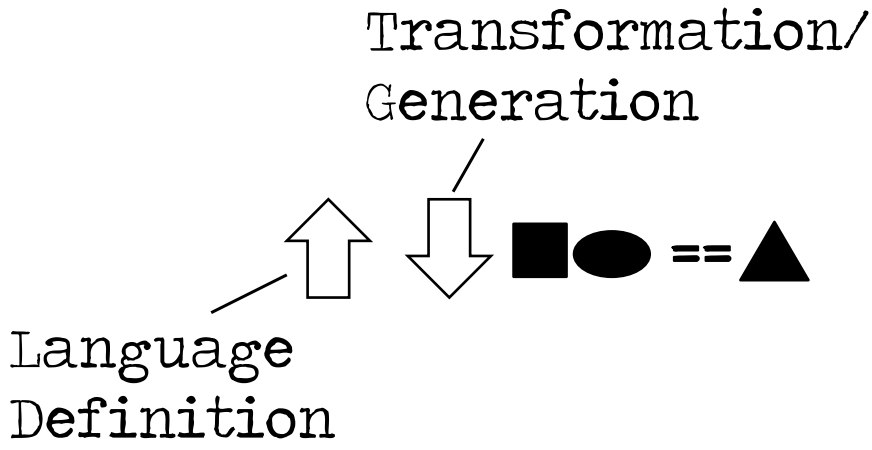
Reification



Reification



Reification



```
int[] arr = ...
for (int i=0; i<arr.size(); i++) {
    sum += arr[i];
}
```

```
int[] arr = ...
List<int> l = ...
for (int i=0;  i<arr.size(); i++) {
    l.add( arr[i] );
}
```

Overspecification!
Requires Semantic Analysis!

```
int[] arr = ...
for (int i=0; i<arr.size(); i++) {
    sum += arr[i];
}
```

```
int[] arr = ...
List<int> l = ...
for (int i=0;  i<arr.size(); i++) {
    l.add( arr[i] );
}
```

Linguistic Abstraction

```
for (int i in arr) {
    sum += i;
}
```

Declarative!
Directly represents Semantics.

```
seqfor (int i in arr) {
    l.add( arr[i] );
}
```

Def: DSL

A DSL is a **language** at D that provides **linguistic abstractions** for **common patterns and idioms** of a language at D-1 when used within the domain D.

Def: DSL cont'd

A good DSL does **not** require the use of patterns and idioms to express **semantically interesting** concepts in D. Processing tools do not have to do "semantic recovery" on D programs.

Declarative!

Semantics & Execution

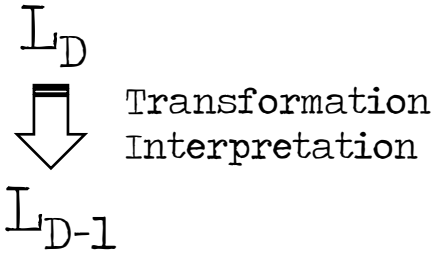
What does it all mean?

Execution Semantics

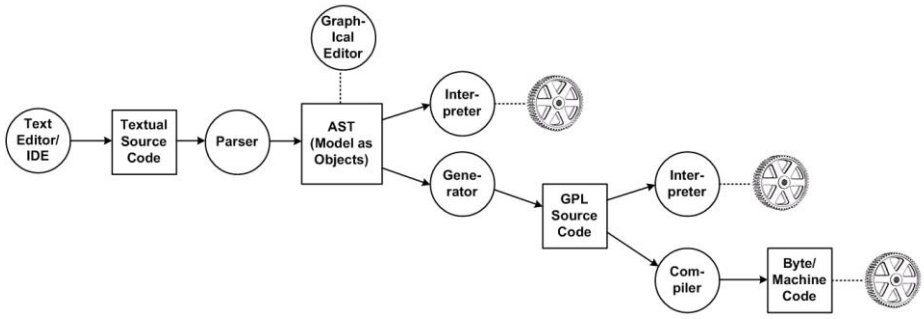
Def: Semantics

… via mapping to lower level

$$semantics(p_{L_D}) := q_{L_{D-1}}$$
$$where\ OB(p_{L_D}) == OB(q_{L_{D-1}})$$



Trafo vs. Interpreter



Transformation

+ Code Inspection
+ Debugging
+ Performance & Optimization
+ Platform Conformance

Interpretation

+ Turnaround Time
+ Runtime Change

Name	Documentation	Tags	Valid time	Transaction time	Fixture	Product	Element	Expected value	Actual value
Accrued right at retireme			2006-12-31	2007-9-24	Jan De Jong	Old Age Pension	Accrued right	761.0402	761.0402
Accrued Right last final pay			2004-1-1	2007-9-24	Jan De Jong	Old Age Pension	Accrued right	705.0589	705.0589
premium last year			2006-1-1	2007-9-24	Jan De Jong	Old Age Pension	Premium old age pension	329.0625	329.0625
Accrued right at retireme 2)			2006-12-31	2007-9-24	Piet Van Dijk	Old Age Pension	Accrued right	740.94	724.7658
			1985-12-31	2007-9-24	Jan De Jong	Old Age Pension	Accrued Right in service period	73.661	73.661
			1985-12-31	2007-9-24	Jan De Jong	Old Age Pension	Years of service in service period	3.7534	3.7534
			1987-12-31	2007-9-24	Jan De Jong	Old Age Pension	Pension base average FP	7750	7750
			1998-12-31	2007-9-24	Jan De Jong	Old Age Pension	Accrued Right in service period	387.7449	387.7449
			1998-12-31	2007-9-24	Jan De Jong	Old Age Pension	Years of service in service period	10.8082	10.8082
			1998-12-31	2007-9-24	Jan De Jong	Old Age Pension	Pension base average FP	8250	

Example

Pension Plans

```
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->stehz
        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
        }
    }
```

```
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
```

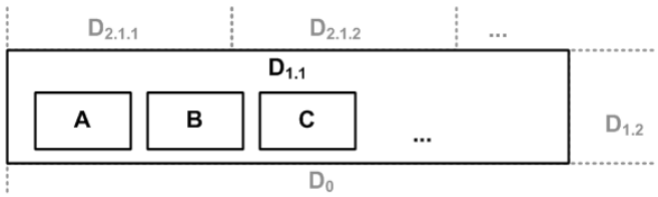
Example

Refrige rators

Separation of Concerns

Several Concerns

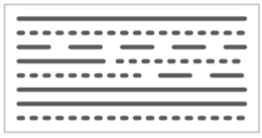
… in one domain



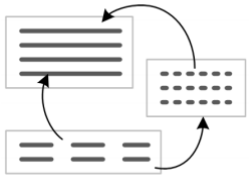
Several Concerns

… in one domain

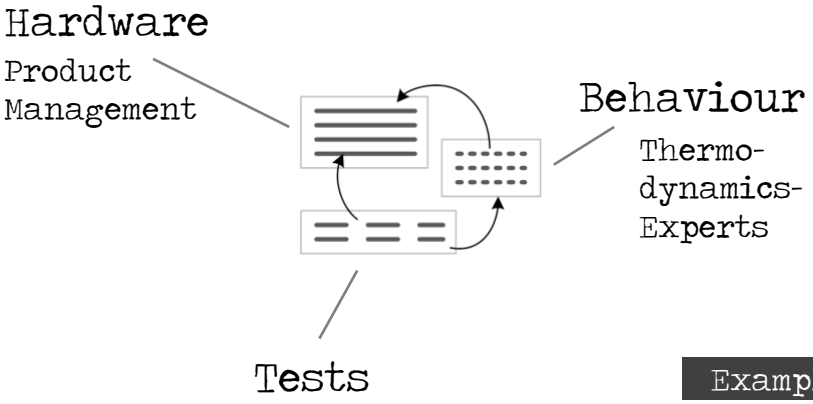
integrated into
one fragment



separated into
several fragments



Viewpoints

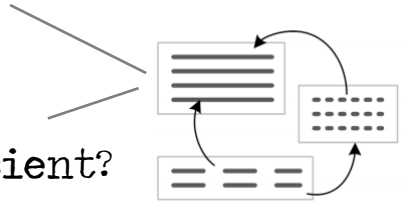


Example

Refrige
rators

Viewpoints

independent

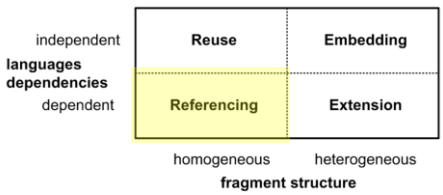


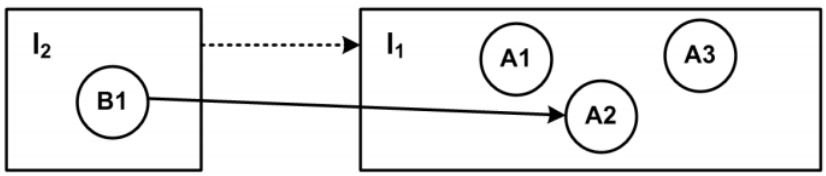
sufficient?

contains all the data for running a meaningful transformation

Language Modularity

Referencing



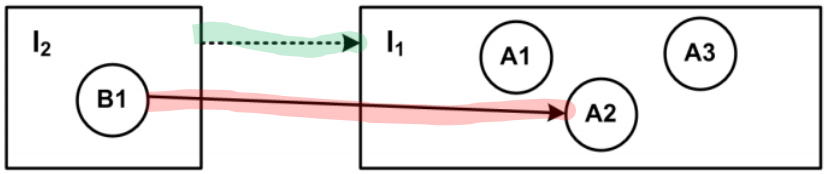


Referencing

languages dependencies	homogeneous	heterogeneous
independent	Reuse	Embedding
dependent	Referencing	Extension

fragment structure

Dependent

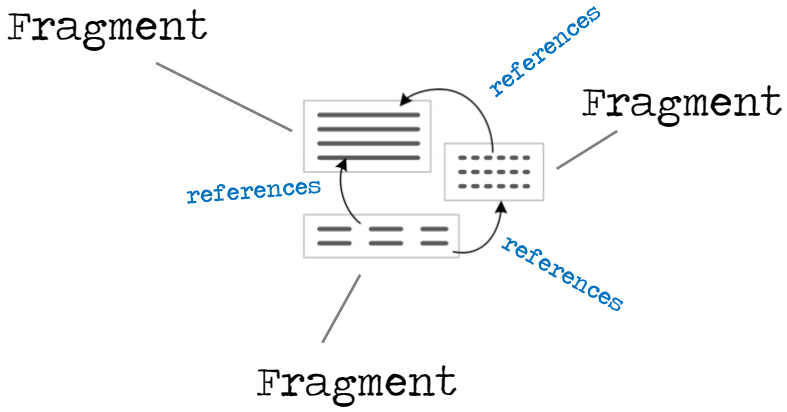


No containment

Referencing

Used in Viewpoints

Referencing



Referencing

```
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->stehz
        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
        }
    }
```

```
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
```

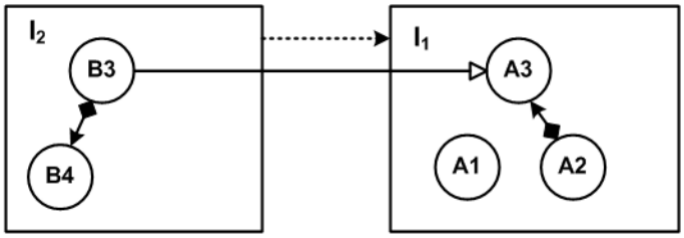
Example

Refrige rators

Extension

languages dependencies	homogeneous	heterogeneous
independent	Reuse	Embedding
dependent	Referencing	Extension

fragment structure

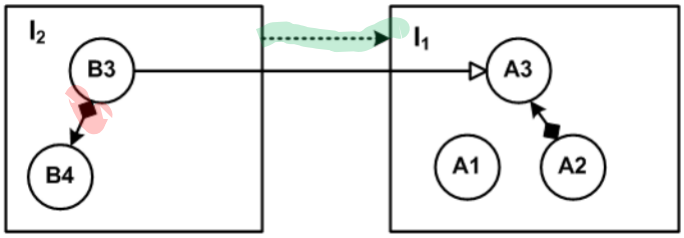


Extension

languages dependencies	homogeneous	heterogeneous
independent	Reuse	Embedding
dependent	Referencing	Extension

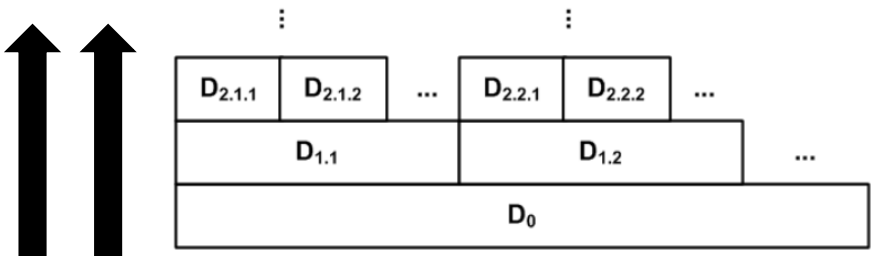
fragment structure

Dependent



Containment

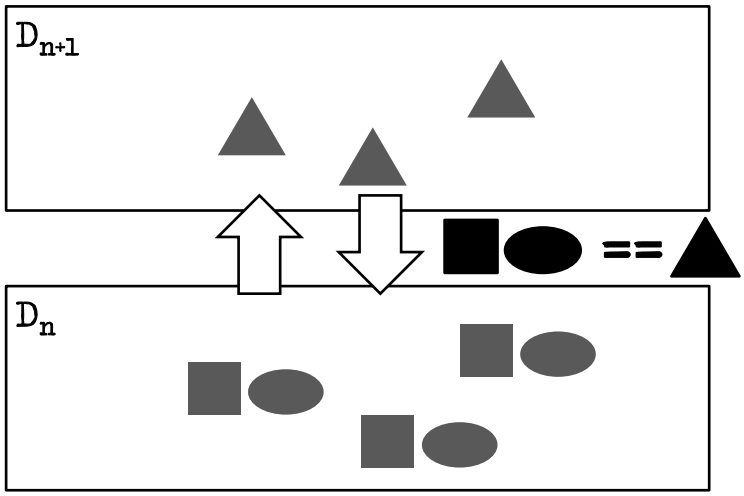
Extension



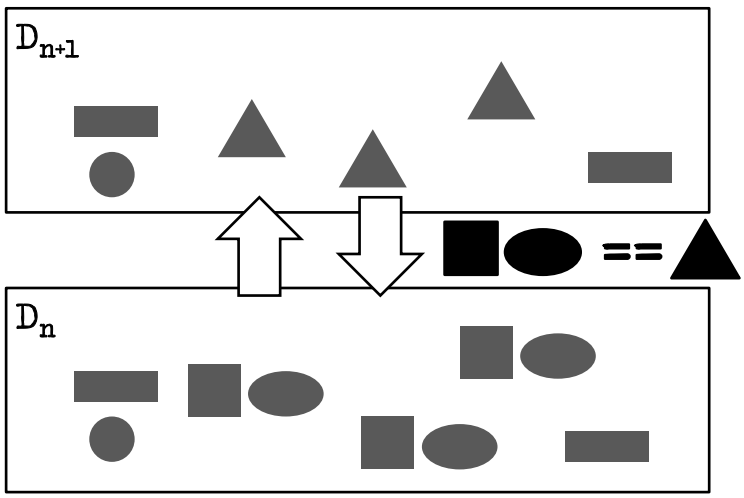
more specialized domains

more specialized languages

Extension

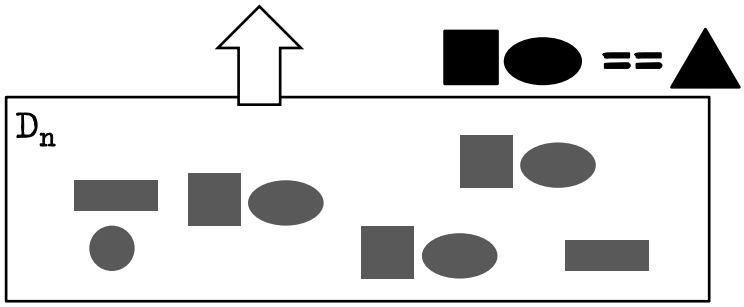


Extension



Extension

Good for **bottom-up** (inductive) domains, and for use by **technical** DSLs (people)

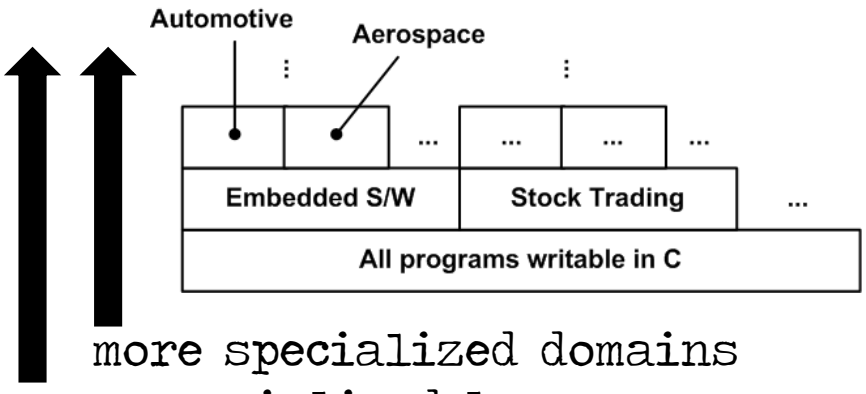


Extension

Drawbacks

tightly bound to base
potentially hard to analyze the combined program

Extension



more specialized domains
more specialized languages

```
module UnitTestDemo from cdesignpaper.unittest imports nothing {

  int32_t main(int32_t argc, int8_t*[ ] argv) {
    return test testMultiply;
  } main (function)

  exported test case testMultiply {
    assert(0) times2(21) == 42;
    if ( 1 > 2 ) {
      fail(1);
    } if

  } testMultiply(test case)

  int8_t times2(int8_t a) {
    return 2 * a;
  } times2 (function)
}
```

```
exported c/s interface Orienter on contract error MultibotMessages.prePostconditionFailed {
  int16_t heading()
    post(0) result >= 0 && result <= 359
  void orientTowards(int16_t heading, uint8_t speed, DIRECTION dir)
    pre(0) heading >= 0 && heading <= 359
}
```


```
exported c/s interface Orienter on contract error MultibotMessages.prePostconditionFailed {
  int16_t heading()
    post(0) result >= 0 && result <= 359
  void orientTowards(int16_t heading, uint8_t speed, DIRECTION dir)
    pre(0) heading >= 0 && heading <= 359
}
```


```
exported component OrienterImpl extends nothing {
  ports:
    provides Orienter orienter
    requires EcRobot_Compass compass
    requires EcRobot_Motor motorLeft
    requires EcRobot_Motor motorRight
  contents:
    field int16_t[5] headingBuffer

    void orienter_orientTowards(int16_t heading, uint8_t speed, DIRECTION dir) <-
        op orienter.orientTowards {
      int16_t currentDir = compass.heading();
      if ( dir == COUNTERCLOCKWISE ) {
        motorLeft.set_speed(-1 * ((int8_t) speed));
        motorRight.set_speed(((int8_t) speed));
        while ( currentDir != heading ) { currentDir = compass.heading(); } while
      } else {
        motorLeft.set_speed(((int8_t) speed));
        motorRight.set_speed(-1 * ((int8_t) speed));
        while ( currentDir != heading ) { currentDir = compass.heading(); } while
      } if
      motorLeft.stop();
      motorRight.stop();
    }

    int16_t orienter_heading() <- op orienter.heading {
      return compass.heading();
    }
}
```

```
exported test case testDriveTrain {
  initialize instances;
  assert(0) dt.currentSpeed() == 0;
  dt.driveContinouslyForward(50);
  dt.stop();
  validate mock motorLeft;
  validate mock motorRight;
} testDriveTrain(test case)
```

```
instance configuration instances extends nothing {
  instances:
  instance MotorLeftMock motorLeft
  instance MotorRightMock motorRight
  instance DriveTrainImpl driveTrain
  instance EcUtil util
  connectors:
  connect driveTrain.motorLeft to motorLeft.motor
  connect driveTrain.motorRight to motorRight.motor
  connect driveTrain.util to util.util
  adapter:
  << ... >>
}
```

```
mock component MotorLeftMock {
  report messages: true
  ports:
    provides EcRobot_Motor motor
  expectations:
    total no. of calls is 2
    sequence {
      0: motor.set_speed {
          0: parameter speed: speed == 50
        }
      1: motor.stop
    }
}
```

```
mock component MotorRightMock {
  report messages: true
  ports:
    provides EcRobot_Motor motor
  expectations:
    total no. of calls is 2
    sequence {
      0: motor.set_speed {
          0: parameter speed: speed == 50
        }
      1: motor.stop
    }
}
```

```
void testBasicUnits() {
  int8_t/N n = 3 N;
                      -2
  int8_t/N n3 = 3 kg m s ;

  int8_t/m length;
  int8_t/s time;
                -1
  int8_t/[m s ] speed = length / time;

  int8_t/v thisShouldNotWork = length + time;

  length l1 = length + length;
  length l2 = length * 33;
  length l3 = length / 3;
  length l4 = length + 3;
} testBasicUnits (function)
```

```
verifiable
statemachine Counter {
  in start() <no binding>
     step(int[0..10] size) <no binding>
  out someEvent(int[0..100] x, boolean b) => handle_someEvent
      resetted() => resetted
  vars int[0..100] currentVal = 0
       int[0..100] LIMIT = 10
  states (initial = initialState)
    state initialState {
      on start [ ] -> countState { send someEvent(100, true && false || true); }
    }
    state countState {
      on step [currentVal + size > LIMIT] -> initialState { send resetted(); }
      on step [currentVal + size <= LIMIT] -> countState { currentVal = currentVal + size; }
      on start [ ] -> initialState {  }
    }
} end statemachine
```

Testing DSLs

Syntax

```
@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'");
	}

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

http://code.google.com/a/eclipselabs.org/p/xtext-utils/wiki/Unit_Testing

Type System

```
model subtyping

calc int t1 = 2;       // works
calc int t2 = 2.2;     // error: float cannot be assigned to int
calc float t3 = 2.2;   // works
calc float t4 = 2;     // works; int can be assigned to float
calc int t5 = 2 + 2;   // works; type of plus is int
calc int t6 = 2 + 2.3; // error: common type of 2 and 2.3 is float
                                                    signed to int
```

incompatible type Int and Float (on a Formula)

Press 'F2' for focus

tests

```
@Test
public void testTArray() throws Exception {
    testFileNoSerializer("4-array.expr2");
    assertConstraints( issues.errorsOnly().sizeIs(1) );
    assertConstraints( issues.forType(Formula.class).named("anotherOne2").
            theOneAndOnlyContains("incompatible") );
}
```

Type System

```
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)
}
```

Type System

```
nodes
  ( [ <check types 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 = <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>;

                  } dummy (function)
                }
```

Type System

```
  ( [ <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.isInstanceOf(ArrayAccessExpr);
    assert true arrayAccessPrioTest2.isInstanceOf(MultiExpression);
    assert true arrayAccessPrioTest2.left.isInstanceOf(ArrayAccessExpr);
    assert true arrayAccessPrioTest2.right.isInstanceOf(NumberLiteral);
    assert true arrayAccessPrioTest3.isInstanceOf(MultiExpression);
    assert true arrayAccessPrioTest3.left.isInstanceOf(ArrayAccessExpr);
    assert true arrayAccessPrioTest3.right.isInstanceOf(PostIncrementExpression);
    assert true arrayAccessPrioTest4.isInstanceOf(UnaryMinusExpression);
    assert true arrayAccessPrioTest4.expression.isInstanceOf(ArrayAccessExpr);
  }
```

Semantics

```
var tuerNachlaufSchwelle: int = 0

start:
    entry { state noCooling }

state noCooling:
    check ( (RC->needsCooling) && (cc.c1->stehz
        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
        }
    }
```

```
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
```

Semantics

```
re.pointers imports nothing {

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)
}
```

Editor

Default × TestCreatingNewVariable × InitializingNewVariable ×

```
Editor test case InitializingNewVariable
description: no description
before: Variables:
          <cell var <no name> : <no alias>>
result: Variables:
          var myVariale : boolean
code:
  type "myVariable"
  press keys <any>+<VK_TAB>;
  press keys <ctrl>+<VK_SPACE>;
  type "boo"
  press keys <any>+<VK_ENTER>;
```

The End.

This material is part of my upcoming (early 2013) book **DSL Engineering**. Stay in touch, it may become a free eBook ☺

www.voelter.de
voelterblog.blogspot.de
@markusvoelter
+Markus Voelter