DSL Design
A conceptual framework for building good DSLs

Markus Voelter
independent/itemis
voelter@acm.org

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

based on material from a paper written with Eelco Visser

E.Visser@tudelft.nl
http://eelcovisser.org/
This material is part of my upcoming (Feb. 2013) book

DSL Engineering
Designing, Implementing and Using Domain-Specific Languages

Stay in touch; it will be cheap :-)  
http://dslbook.org
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.
Modular Language

with many optional, composable modules
Case Studies
namespace com.mycompany {
    namespace datacenter {
        component DelayCalculator {
            provides aircraft: IAircraftStatus
            provides console: IManagementConsole
            requires screens[0..n]: IInfoScreen
        }
        component Manager {
            requires backend[1]: IManagementConsole
        }
        public interface IInfoScreen {
            message expectedAircraftArrivalUpdate (id: ID, time: Time)
            message flightCancelled(flightID: ID)
        }
        public interface IAircraftStatus ...
        public interface IManagementConsole ...
    }
}
namespace com.mycompany.test {
    system testSystem {
        instance dc: DelayCalculator
        instance screen1: InfoScreen
        instance screen2: InfoScreen
        connect dc.screens to
            (screen1.default, screen2.default)
    }
}
appliance KIR {
    compressor compartment cc {
        static compressor c1
        fan ccfan
    }
    ambient temsensor at
    cooling compartment RC {
        light rclight
        superCoolingMode
door rcdoor
        fan rcfan
evaporator temsensor rceva
    }
}
```plaintext
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->stehzeit > 333) ) {
    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
    }
  }
```
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
module main imports OsekKernel, EcAPI, BitLevelUtilities {

    constant int WHITE = 500;
    constant int BLACK = 700;
    constant int SLOW = 20;
    constant int FAST = 40;

    statemachine linefollower {
        event initialized;
        initial state initializing {
            initialized [true] -> running
        }
        state running { }
    }

    initialize {
        ecrobot_set_light_sensor_active
            (SENSOR_PORT_T::NXT_PORT_S1);
        event linefollower:initialized
    }

    terminate {
        ecrobot_set_light_sensor_inactive
            (SENSOR_PORT_T::NXT_PORT_S1);
    }

task run cyclic prio = 1 every = 2 {
    statemachine linefollower
    state running
        int32 light = 0;
        light = ecrobot_get_light_sensor
            (SENSOR_PORT_T::NXT_PORT_S1);
        if ( light < ( WHITE + BLACK ) / 2 ) {
            updateMotorSettings(SLOW, FAST);
        } else {
            updateMotorSettings(FAST, SLOW);
        }
    default
        <noop>;
}

void updateMotorSettings ( int left, int right ) {
    nxt_motor_set_speed(MOTOR_PORT_T::NXT_PORT_C, left, 1);
    nxt_motor_set_speed(MOTOR_PORT_T::NXT_PORT_B, right, 1);
}
3.3 Commutatiegetallen op 1 leven

\[ D_x = \frac{x}{l} \frac{x}{100} = 6 \text{ Dec}(3) \]

\[ \omega - x \]

\[ N = \sum_{t=0}^{x} D_{x-t} = 7 \text{ Dec}(3) \]

3.6 Contante waarde 1 leven/2 levens

\[ D = \frac{x+n}{D_x} = 19 \text{ Dec}(4) \]

\[ a_x = \frac{a}{x} - 1 = 21 \text{ Dec}(3) \]

\[ \bar{a}_x = \bar{a} - 0,5 = 22 \text{ Dec}(3) \]

\[ \frac{N-N}{N} = \frac{x+n}{D_x} = 23 \text{ Dec}(3) \]

\[ \bar{a}_x = \frac{\bar{a}_x - 0,5 + 0,5 \times E_x}{x} = 25 \text{ Dec}(3) \]

4 BN(_ris) koopsommen
## Example Pension Plans

### Rules

- **Rule Bereken Mutatieperiode**
  - **Result:** Mutatieperiode
  - **Name:** Bereken Mutatieperiode
  - **Documentation:** Het vaststellen van de periode tussen de huidige en de vorige mutatie in dagen.
  
  De mutatieperiode kan niet meer dan 360 dagen bedragen omdat elk jaar een begin- en eindmutatie kent i.v.m. het openen en sluiten van het verslagjaar.
  
  Dit wordt niet afgevangen omdat het uitvoeren van de begin- en eindmutatie verantwoordelijkheid zijn van de pensioenadministratie.

- **Tags:** Basisberekening
- **Algorithm:**

\[
\text{if maximum}(\text{Mutaties per datum}) = \text{1 then } \text{days of (duration (valid(Mutaties per datum)))) else 0}
\]

### Test cases

<table>
<thead>
<tr>
<th>Name</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>Gelijke datums</td>
<td>03/01/2008</td>
<td></td>
<td></td>
<td>Mutatieperiode - Mutatiedatum = Mutatiedatum Vorig</td>
<td>(een maand)</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Periode &lt; 30</td>
<td>03/01/2008</td>
<td></td>
<td></td>
<td>Mutatieperiode - Mutatiedatum &gt; Mutatiedatum Vorig (binnen 1 maand)</td>
<td></td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>Periode &gt; 30</td>
<td>03/01/2008</td>
<td></td>
<td></td>
<td>Mutatieperiode - Mutatiedatum &gt; Mutatiedatum Vorig (meerder maanden)</td>
<td></td>
<td>60</td>
<td>60</td>
</tr>
</tbody>
</table>
entity Post {
    key :: String (id)
    blog -> Blog
    urlTitle :: String
    title :: String (searchable)
    content :: WikiText (searchable)
    public :: Bool (default=false)
    authors -> Set<User>
}

function isAuthor(): Bool {
    return principal() in authors
}

function mayEdit(): Bool {
    return isAuthor();
}

function mayView(): Bool {
    return public || mayEdit();
}

access control rules

rule page post(p: Post, title: String) {
    p.mayView()
}

rule template newPost(b: Blog) {
    b.isAuthor()
}

section posts

define page post(p: Post, title: String) {
    title{ output(p.title) }
    bloginfo(p.blog){
        placeholder view { postView(p) }
        postComments(p)
    }
}

define permalink(p: Post) {
    navigate post(p, p.urlTitle) { elements }
}
Terms & Concepts
Domain
Domain

deductive top down

body of knowledge in the real world

Example
Penion Plans

Example
Refrigermators
Domain

existing software (family)

inductive bottom up

Example
Extrended C

Example
Component
Programs are trees

```
    M
   / \    
  A   B    
    / \    
   C   D    
      / \  
     E   F  
        /  
         G
```
Programs are graphs, really.
Fragments are subtrees with root.
Languages are sets of concepts

$L = \{C_1, C_2, C_3, \ldots, C_n\}$
Languages are sets of concepts

$L \rightarrow \text{concept} \rightarrow \text{language}$
Programs and languages

\[
M \rightarrow A \rightarrow C \rightarrow D \rightarrow E \rightarrow F \rightarrow G
\]

\[
\text{LocalVariableDeclaration} \rightarrow \text{element} \rightarrow \text{concept}
\]

\[x = 3;\]
Language: concept inheritance

\[ \text{Language: concept inheritance} \]
Independence

Language does not depend on any other language

Fragment does not depend on any other fragment

\[ \forall r \in \text{Refs}_l \mid \text{lo}(r.\text{to}) = \text{lo}(r.\text{from}) = l \]

\[ \forall s \in \text{Inh}_l \mid \text{lo}(s.\text{super}) = \text{lo}(s.\text{sub}) = l \]

\[ \forall c \in \text{Cdn}_l \mid \text{lo}(c.\text{parent}) = \text{lo}(c.\text{child}) = l \]

\[ \forall r \in \text{Refs}_f \mid \text{fo}(r.\text{to}) = \text{fo}(r.\text{from}) = f \]

\[ \forall e \in E_f \mid \text{lo}(\text{co}(e)) = l \]
Independence

Hardware:

```c
compressor compartment cc {
  static compressor c1
  fan ccfan
}
```

Cooling Algorithm

```python
macro kompressorAus {
  set cc.c1->active = false
  perform ccfanabschaltet after 10 {
    set cc.ccfan->active = false
  }

}
Homogeneous Fragment

everything expressed with one language

\[ \forall e \in E_f \mid lo(e) = l \]
\[ \forall c \in Cdn_f \mid lo(c.parent) = lo(c.child) = l \]
module CounterExample from counterd imports nothing {

var int theI;
var boolean theB;
var boolean hasBeenReset;

statemachine Counter {
  in start() <no binding>
  step(int[0..10] size) <no binding>
  out someEvent(int[0..100] x, boolean b) <no binding>
  reseted() <no binding>
  vars int[0..10] 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 reseted(); }
    on step [currentVal + size <= LIMIT] -> countState { currentVal = currentVal + size; }
    on start [ ] -> initialState { }
  }
}

var Counter c1;

exported test case test1 {
  initsm(c1);
  assert(0) isState<Counter, initialState>;
  trigger(c1, start);
  assert(1) isState<Counter, countState>;
  test1(test case)
}
Domain Hierarchy
Domain Hierarchy

all programs

embedded software

automotive

avionics

Example Extended C
Design Dimensions

expressivity
coverage
semantics
separation of concerns

| completeness | paradigms |
| modularity | concrete |
| syntax |

process
Expressivity

expressivity
coverage
semantics
separation of concerns

process

completeness
paradigms
modularity
concrete
syntax
Shorter Programs
More Accessible Semantics
For a limited Domain!

Domain Knowledge encapsulated in language
Smaller Domain → More Specialized Language → Shorter Programs
The do-what-I-want language
Single Program vs. Class/Domain

No Variability!
Domain Hierarchy

more specialized domains
more specialized languages
Reification

\[ D_n \]
Reification

\[ D_{n+1} \]

\[ \square \] \[ \text{ellipse} \] \[ \triangle \] \[ == \]

\[ D_n \]

\[ \square \] \[ \text{ellipse} \]
Reification

Transformation/Generation

Language Definition

\[ \square = \triangle \]
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!

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

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

Declarative!
Directly represents Semantics.

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

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

expressivity  completeness
coverage  paradigms
semantics  modularity
separation of  concrete
concerns  syntax

process
Static Semantics

Execution Semantics
Static Semantics

Execution Semantics
Unique State Names
Unreachable States
Dead End States

Example
Extended C
Unique State Names
Unreachable States
Dead End States
...
Easier to do on a declarative Level!
Unique State Names
Unreachable States
Dead End States
Easier to do on a declarative Level!

Thinking of all constraints is a coverage problem!
What does a type system do?

var int x = 2 * someFunction(sqrt(2));

Assign fixed types

Derive Types

Calculate Common Types

Check Type Consistency
var int x = 2 * someFunction(sqrt(2));
Intent + Check

```
var int x = 2 * someFunction(sqrt(2));
```

More code
Better error messages
Better Performance

Derive

```
var x = 2 * some Function(sqrt(2));
```

More convenient
More complex checkers
Harder to understand for users
What does it all mean?
Execution Semantics
Def: Semantics

... via mapping to lower level

\[ \text{semantics}(p_{L_D}) := q_{L_{D-1}} \]

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

OB: Observable Behaviour (Test Cases)
**Def: Semantics**

... via mapping to lower level

\[ \text{semantics}(p_{LD}) := q_{LD-1} \]

where \( OB(p_{LD}) = OB(q_{LD-1}) \)

\[ \mathbb{L}_D \xrightarrow{\text{Transformation}} \mathbb{L}_{D-1} \]
Transformation

$D_{n+1}$

$D_n$

Diagram showing the transformation from $D_{n+1}$ to $D_n$.
Transformation

```plaintext
module impl imports <<imports>> {

    int speed( int val ) {
        return 2 * val;
    }

    robot script stopAndGo
    block main on bump
        accelerate to 12 + speed(12) within 3000
        drive on for 2000
        turn left for 200
        decelerate to 0 within 3000
        stop
    }

Example
Extended C
```
Transformation

$\mathbf{L}_D \xrightarrow{\text{Transformation}} \mathbf{L}_{D-1}$

Known Semantics!
Transformation

Correct!?  

$\mathbf{L}_D$  

Transformation  

$\mathbf{L}_{D-1}$
Transformation

Tests (D) \downarrow L_D \downarrow Transformation

Tests (D-1) \downarrow L_{D-1}

Run tests on both levels; all pass.
Coverage Problem!
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 rccoooling
    )
    on isDown ( RC.rcdoor->open ) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        perform rcfanAbsschalttask after 10 {
            set RC.rcfan->active = false
        }
    }

state rccoooling:
    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 rcfanAbsschalttask 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 abtau
assert-value cc.c1->active is false
mock: set RC->accumulatedRuntime = 0
step 5
assert-currentstate-is abtau
assert-value cc.c1->active is false
step 15
assert-currentstate-is noCooling
<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>1985-12-31</td>
<td></td>
<td></td>
<td></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>1985-12-31</td>
<td></td>
<td></td>
<td></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>1987-12-31</td>
<td></td>
<td></td>
<td></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>1998-12-31</td>
<td></td>
<td></td>
<td></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>1998-12-31</td>
<td></td>
<td></td>
<td></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>

**Example Pension Plans**
Behavior

Refrigerators

Example

Refrigerators
Multi-Stage

\[ \mathcal{L}_3 \rightarrow \mathcal{L}_2 \rightarrow \mathcal{L}_1 \rightarrow \mathcal{L}_0 \]

Modularization
Multi-Stage: Reuse

L_3 \rightarrow L_2 \rightarrow L_1 \rightarrow L_0

Reusing Later Stages
Optimizations!
Multi-Stage: Reuse

Robot Control State Machine

Components

C (MPS tree)

C Text

Example

Extended C
Multi-Stage: Reuse

Robot Control
State Machine

Consistency
Model Checking

Efficient Mappings

C Type System

Syntactic Correctness,
Headers

C Text

Example
Extended C
Multi-Stage: Reuse

Reusing Early Stages
Portability
Multi-Stage: Reuse

$L_3$ → $L_2$

$L_1$ → $L_0$

$L_{1b}$ → $L_{0b}$

Java

C#
Multi-Stage: Preprocess

Adding an optional, modular emergency stop feature
A program at $D_0$ that acts on the structure of an input program at $D_{>0}$.
Interpretation

A program at $D_0$ that acts on the structure of an input program at $D_0$.

- imperative $\rightarrow$ step through
- functional $\rightarrow$ eval recursively
- declarative $\rightarrow$ ? solver ?
<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 retirement</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 retirement 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>Years of service in period</td>
<td>73.661</td>
<td>73.661</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 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>

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->steht) )
    state rccooling
    on isDown ( RC.rcdoor->open ) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        perform rcfanabschalittask 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 rcfanabschalittask 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 Refrigerators
Transformation

+ Code Inspection

Interpretation
Transformation

- Code Inspection
- OSGi Generators

Interpretation
Transformation

+ Code Inspection
+ Debugging

Interpretation
Transformation

+ Code Inspection
+ Debugging

Interpretation

Platform Interactions
Transformation

+ Code Inspection
+ Debugging
+ Performance & Optimization

Interpretation
Transformation

+ Code Inspection
+ Debugging
+ Performance & Optimization

Efficiency for Real-Time S/w
Memory Use
Transformation

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

Interpretation
<table>
<thead>
<tr>
<th>Transformation</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ Code Inspection</td>
<td></td>
</tr>
<tr>
<td>+ Debugging</td>
<td></td>
</tr>
<tr>
<td>+ Performance &amp; Optimization</td>
<td></td>
</tr>
<tr>
<td>+ Platform Conformance</td>
<td></td>
</tr>
<tr>
<td>Web Frameworks and Standards</td>
<td></td>
</tr>
</tbody>
</table>

Example

Web DSL
Transformation

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

Interpretation

+ Turnaround Time
Transformation

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

Interpretation

+ Turnaround Time Testing

Example

Pension Plans

Example

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

Interpretation
+ Turnaround Time
+ Runtime Change
Transformation

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

Interpretation

+ Turnaround Time
+ Runtime Change Business Rules without Redeployment
Def: Semantics

... via mapping to lower level

\[ \mathbb{L}_D \downarrow \]

Transformation

\[ \mathbb{L}_{D-1} \]
Multiple Mappings
... at the same time

\[ L_D \]

\[ L_x \quad L_y \quad L_z \]

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

$\mathbf{L}_D \circ \mathbf{T}$

$\mathbf{L}_x \circ \mathbf{T}$  $\mathbf{L}_y \circ \mathbf{T}$  $\mathbf{L}_z \circ \mathbf{T}$

all green!

Similar Semantics?
<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>1985-12-31</td>
<td></td>
<td></td>
<td>2007-9-24</td>
<td></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>1985-12-31</td>
<td></td>
<td></td>
<td>2007-9-24</td>
<td></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>1987-12-31</td>
<td></td>
<td></td>
<td>2007-9-24</td>
<td></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>1998-12-31</td>
<td></td>
<td></td>
<td>2007-9-24</td>
<td></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>1998-12-31</td>
<td></td>
<td></td>
<td>2007-9-24</td>
<td></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>1998-12-31</td>
<td></td>
<td></td>
<td>2007-9-24</td>
<td></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>
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

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

bad? maybe.
good? maybe!
Reduced Expressiveness

bad? maybe.
good? maybe!

Model Checking
SAT Solving

Exhaustive Search, Proof!
Unique State Names
Unreachable States
Dead End States
Guard Decidability
Reachability

Exhaustive Search, Proof!
c/s interface Decider {
    int decide(int x, int y) pre
}

component AComp extends nothing {
    ports:
        provides Decider decider
    contents:
        int decide(int x, int y) <- op decider.decide {
            return int, 0 |
            \begin{array}{|c|c|c|}
                \hline
                x & y \leq 0 & y > 0 \\
                \hline
                x \leq 0 & 0 & 1 \\
                x > 0 & 1 & 2 \\
                \hline
            \end{array}
        }

}
Separation of Concerns

expressivity
coverage
semantics
separation of concerns

process

completeness
paradigms
modularity
concrete
syntax
Several Concerns

... in one domain
Several Concerns
... in one domain

integrated into one fragment

separated into several fragments
Viewpoints

independent

\[ \forall r \in \text{Refs}_f \mid \text{fo}(r.\text{to}) = \text{fo}(r.\text{from}) = f \]

\[ \forall e \in E_f \mid \text{lo}(\text{co}(e)) = l \]

dependent
Viewpoints: Why?

Sufficiency
Different Stakeholders
Different Steps in Process - VCS unit!
Viewpoints

Hardware
Product Management

Tests

Behaviour
Thermo-dynamics-Experts

Example
Refrigerators
Viewpoints

sufficient?
contains all the data for running a meaningful transformation

independent
Viewpoints

sufficient
Hardware structure
documentation

Example
Refrigerators

sufficient implementation code
Viewpoints: Why?

1:n Relationships
Viewpoints

Hardware 1:n Behaviour

Tests n:1

Example Refrigerators
Viewpoints

Well-defined Dependencies

No Cycles!

Avoid Synchronization!
(unless you use a projectional editor)
<table>
<thead>
<tr>
<th>Language Modularity</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>expressivity</td>
<td>completeness</td>
</tr>
<tr>
<td>coverage</td>
<td>paradigms</td>
</tr>
<tr>
<td>semantics</td>
<td>modularity</td>
</tr>
<tr>
<td>separation of concerns</td>
<td>concrete</td>
</tr>
<tr>
<td></td>
<td>syntax</td>
</tr>
<tr>
<td>process</td>
<td></td>
</tr>
</tbody>
</table>
Language Modularity, Composition and Reuse (LMR&C) increase efficiency of DSL development
Language Modularity, Composition and Reuse (LMR&C) increase efficiency of DSL development

4 ways of composition:
- Referencing
- Reuse
- Extension
- Reuse
Language Modularity, Composition and Reuse (LMR&C) increase efficiency of DSL development.

4 ways of composition: distinguished regarding dependencies and fragment structure.
Dependencies: do we have to know about the reuse when designing the languages?
Dependencies: do we have to know about the reuse when designing the languages?

Fragment Structure: homogeneous vs. heterogeneous ("mixing languages")
Dependencies & Fragment Structure:

<table>
<thead>
<tr>
<th>Independent Languages Dependencies</th>
<th>Reuse</th>
<th>Embedding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Homogeneous</td>
<td>Referencing</td>
<td>Extension</td>
</tr>
<tr>
<td>Heterogeneous</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fragment structure</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Referencing

Referencing is independent of languages and dependent on dependencies.

<table>
<thead>
<tr>
<th>Reuse</th>
<th>Embedding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Referencing</td>
<td>Extension</td>
</tr>
<tr>
<td>homogeneous</td>
<td>heterogeneous</td>
</tr>
<tr>
<td>fragment structure</td>
<td></td>
</tr>
</tbody>
</table>

Diagram:
- I₁: A₁ → A₂
- I₂: B₁

References:
- B₁ refers to I₂
- A₁ references I₁
Referencing

Dependent

No containment
Referencing Used in Viewpoints
Referencing

Fragment

references

Fragment

references

Fragment
Referencing

```haskell
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
```
Extension

<table>
<thead>
<tr>
<th>Reuse</th>
<th>Embedding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Referencing</td>
<td>Extension</td>
</tr>
<tr>
<td>homogeneous</td>
<td>heterogeneous</td>
</tr>
<tr>
<td>fragment structure</td>
<td></td>
</tr>
</tbody>
</table>

Graph:

- **I₂**: B4 → B3
- **I₁**: A1 → A3
- **I₂** references **I₁** via **B3**

Independent languages, dependent dependencies.
Extension

Dependent

Containment

<table>
<thead>
<tr>
<th>Reuse</th>
<th>Embedding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Referencing</td>
<td>Extension</td>
</tr>
</tbody>
</table>

independent languages dependencies dependent

homogeneous heterogeneous fragment structure
Extension

more specialized domains
more specialized languages
Extension

\[ D_{n+1} \]

\[ \Rightarrow \]

\[ D_n \]

==
Extension

\[ D_{n+1} \]

\[ D_n \]

\[ \Rightarrow \]

==
Extension

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

Drawbacks

tightly bound to base
potentially hard to analyze
the combined program

Extension
module main imports OsakKernel, EcAPI, BitLevelUtilities {

constant int WHITE = 500;
constant int BLACK = 700;
constant int SLOW = 20;
constant int FAST = 40;

statemachine linefollower {
    event initialized;
    initial state initializing {
        initialized [true] -> running
    }
    state running {
    }
}

initialize {
    ecrobot_set_light_sensor_active
        (SENSOR_PORT_T::NXT_PORT_S1);
    event linefollower:initialized
}

terminate {
    ecrobot_set_light_sensor_inactive
        (SENSOR_PORT_T::NXT_PORT_S1);
}

task run cyclic prio = 1 every = 2 {
    stateswitch linefollower
        state running
            int32 light = 0;
            light = ecrobot_get_light_sensor
                (SENSOR_PORT_T::NXT_PORT_S1);
            if ( light < ( WHITE + BLACK ) / 2 ) {
                updateMotorSettings(SLOW, FAST);
            } else {
                updateMotorSettings(FAST, SLOW);
            }
        default
            <noop>;
    }

void updateMotorSettings( int left, int right ) {
    nxt_motor_set_speed(MOTOR_PORT_T::NXT_PORT_S1, left);
    nxt_motor_set_speed(MOTOR_PORT_T::NXT_PORT_S1, right);
}
Embedding

<table>
<thead>
<tr>
<th>Reuse</th>
<th>Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>Referencing</td>
<td>Embedding</td>
</tr>
<tr>
<td>homogeneous</td>
<td>heterogeneous</td>
</tr>
<tr>
<td>fragment structure</td>
<td></td>
</tr>
</tbody>
</table>
Embedding

<table>
<thead>
<tr>
<th>Reuse</th>
<th>Embedding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Referencing</td>
<td>Extension</td>
</tr>
<tr>
<td>homogeneous</td>
<td>heterogeneous</td>
</tr>
<tr>
<td>fragment structure</td>
<td>fragment structure</td>
</tr>
</tbody>
</table>
Embedding

Independent

Containment

<table>
<thead>
<tr>
<th>Reuse</th>
<th>Embedding</th>
</tr>
</thead>
<tbody>
<tr>
<td>homogeneous</td>
<td>heterogeneous</td>
</tr>
<tr>
<td>fragment structure</td>
<td></td>
</tr>
</tbody>
</table>

Independent languages
dependencies
dependent
Embedding

Example Pension Plans
Embedding often uses Extension to extend the embedded language to adapt it to its new context.
Concrete Syntax

expressivity
coverage
semantics
separation of concerns

process

completeness
paradigms
modularity
concrete syntax
UI for the language!
Important for acceptance by users!

Textual
Symbolic
Tabular
Graphical
Reuse existing syntax of domain, if any!

Tools let you freely combine all kinds.
Default: Text

Editors simple to build
Productive
Easy to integrate w/ tools
Easy to evolve programs
Editors simple to build
Productive
Easy to integrate w/ tools
Easy to evolve programs
... then add other forms, if really necessary
Graphical in case... Relationships
Graphical in case...

Flow and Dependency
Graphical in case...

Causality and Timing
Symbolic
Either Mathematical, or often highly inspired by domain
<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 retirem</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>Piet Van Dijk</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 retirem 2</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>740.94</td>
<td>724.7658</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Accrued Right in service period</td>
<td>73.661</td>
<td>73.661</td>
</tr>
<tr>
<td>1985-12-31</td>
<td></td>
<td></td>
<td>2007-9-24</td>
<td></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>1987-12-31</td>
<td></td>
<td></td>
<td>2007-9-24</td>
<td></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>1998-12-31</td>
<td></td>
<td></td>
<td>2007-9-24</td>
<td></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></td>
<td></td>
<td></td>
<td></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></td>
<td></td>
<td></td>
<td></td>
<td>Pension base average FP</td>
<td>8250</td>
<td>8250</td>
</tr>
</tbody>
</table>
Combinations

c/s interface Decider {
    int decide(int x, int y) pre
}

component AComp extends nothing {
    ports:
        provides Decider decider
    contents:
        int decide(int x, int y) <- op decider.decide {
            return int, 0
        }
    }

    \[
    \begin{array}{|c|c|c|}
    \hline
    & x = 0 & x > 0 \\
    \hline
    y = 0 & 0 & 1 \\
    y > 0 & 1 & 2 \\
    \hline
    \end{array}
    \]
Combinations
Combinations

<table>
<thead>
<tr>
<th>Description</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>system  SHALL display speed</td>
<td></td>
</tr>
<tr>
<td>system  SHALL display rpm</td>
<td></td>
</tr>
<tr>
<td>delay is less than “5”</td>
<td></td>
</tr>
<tr>
<td>rpm is greater than</td>
<td></td>
</tr>
</tbody>
</table>

**SHALL**
- and
- is disabled
- is enabled
- is equal to
- is greater than
- is less than
- is not equal to
- or
- xor
Combinations

CallHandling
interface user:
in event hangup
in event accept

interface phone:
in event callIncoming : string
in event callFinished
out event acceptCall
out event hangupCall

internal:
event finished = callFinished || hangup

var timer : integer

CallCycle

Waiting

IncomingCall
popup Phone.CallFinished
accept

callIncoming

Active

popup Phone.Call

scene Phone.ActiveCall
entry /
raise acceptCall;
timer = 0;
after 1 s / timer = timer + 1;

accept
acceptCall

callFinished

finished

hangup

hangupCall

timer
The End.

DSL Engineering
Designing, Implementing and Using Domain-Specific Languages

Markus Voelter
with  Sebastian Benz
Christian Dietrich
Birgit Engelmann
Mats Helander
Lennart Kats
Eelco Visser
Guido Wachsmuth

www.dslbook.org

@markusvoelter
+Markus Voelter