Models, DSLs and Code Generation

\[ E = \sum_{-\infty}^{+\infty} \sum_{-\infty}^{+\infty} \sum_{i=1}^{41} \sum_{\theta=0}^{2\pi} \int_{t=-\infty}^{+\infty} \int_{y_{+\infty}}^{y_{-\infty}} \int_{x_{+\infty}}^{x_{-\infty}} \hat{A}_{\theta \phi} d\theta d\phi dx \]

Markus Voelter
independent/itemis
voelter@acm.org

www.voelter.de
voelterblog.blogspot.de
@markusvoelter
+Markus Voelter
Models,
Languages,
DSLs
Model

A schematic description of a system, theory, or phenomenon that accounts for its known or inferred properties and may be used for further study of its characteristics.
Model

an abstraction or simplification of reality

ecosurvey.gmu.edu/glossary.htm
an abstraction or simplification of reality

which ones?

what should we leave out?
Model Purpose

... code generation
... analysis and checking
... platform independence
... stakeholder integration

"drives design of language!"
A Domain Specific Language is a focussed, processable language for describing software in a specific domain. The abstractions and notations used are suitable for the stakeholders who use the language.
This is a DSL!
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 tempsensor at

  cooling compartment RC {
    light rclight
    superCoolingMode
doор rcdoor
    fan rcfan
    fan rcfan
    evaporator tempsensor rceva
  }

}
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 rcfanabschalltask 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 rcfanabschalltask 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->steht)
        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
module ADemoModule from cdesignpaper.screenshot imports nothing {

enum MODE { FAIL; AUTO; MANUAL; }

statemachine Counter {
    in start() <no binding>
        [step(int[0..10] size) <no binding>] trace R2
    out started() <no binding>
        resetted() <no binding> (resettable)
        incremented(int[0..10] newVal) <no binding>
    vars int[0..10] currentVal = 0
        int[0..10] LIMIT = 10
    states (initial = start)
        state start {
            on start [ ] -> countState { send started(); }
            start ^inEvents (cdesignpaper.screenshot.ADemoModule)
            step ^inEvents (cdesignpaper.screenshot.ADemoModule)
        }
        on step [currentVal + size > LIMIT] -> start { send resetted(); }
        on step [currentVal + size <= LIMIT] -> countState {
            currentVal + size;
        }
        send incremented();
        on start [ ] -> start { send resetted(); }
        (resettable)
    }
    end statemachine

    MODE nextMode(MODE mode, int8_t speed) {
        return [MODE, FAIL]
            speed < 50 { mode == AUTO, mode == MANUAL }
            speed >= 50 { MANUAL, MANUAL }
        trace R1;
    }
}
```c
void testBasicUnits() {
    int8_t/N n = 3 N;
    int8_t/N n3 = 3 kg m s ;

    int8_t/m length;
    int8_t/s time;
    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)
```
simple product BasisHausrat

attributes
- groesse : int16_t
- alter : int16_t
- dritterParam : boolean

calculate
- int16_t, 0

<table>
<thead>
<tr>
<th>groesse &lt; 150</th>
<th>groesse &gt;= 150</th>
</tr>
</thead>
<tbody>
<tr>
<td>alter &lt; 18</td>
<td>10</td>
</tr>
<tr>
<td>alter &gt;= 18</td>
<td>30</td>
</tr>
</tbody>
</table>

tests
- groesse, alter, dritterParam

| 100 | 2   | true | 20   | 10  |
| 200 | 30  | true | 40   | 40  |
| 150 | 20  | true | 40   | 40  |
Example

Pension Plans

\[ D_x = \frac{x}{100} \times \frac{l_x}{D_x} \quad \text{Dec}(3) \]

\[ N_x = \sum_{t=0}^{\infty} D_{x+1} \quad \text{Dec}(3) \]

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

\[ a_x = \alpha - 1 \quad \text{Dec}(3) \]

\[ \bar{a}_x = \bar{a}_x - 0.5 \quad \text{Dec}(3) \]

\[ \bar{d}_x = \frac{N_x - N_{x+n}}{D_x} \quad \text{Dec}(3) \]

\[ \bar{a}_x = \frac{\bar{a}_x}{\bar{a}_x} - 0.5 \times E_{n+x} \quad \text{Dec}(3) \]

\[ 4 \text{ BN(_ris) koopsommen} \]
Elements...

Rules

- Rule Bereken Mutatieperiode
  - Result: Mutatieperiode
  - Name: Bereken Mutatieperiode
  - Documentation: Het vaststellen van de periode tussen de huidige en de vorige mutate 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:
    ```
    if maximum(Mutaties per datum) == 1 then daysof(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 -</td>
<td>3</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Mutatiedatum =</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Mutatiedatum</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Vorig</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Periode &lt; 30</td>
<td>03/01/2008</td>
<td></td>
<td></td>
<td>Mutatieperiode -</td>
<td>15</td>
<td></td>
<td>15</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Mutatiedatum &gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Mutatiedatum</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Vorig (binnen 1 maand)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Periode &gt; 30</td>
<td>03/01/2008</td>
<td></td>
<td></td>
<td>Mutatieperiode -</td>
<td>60</td>
<td></td>
<td>60</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Mutatiedatum &gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Mutatiedatum</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Vorig (meerdere maanden)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Concepts (abstract syntax)
(concrete) Syntax
semantics (generators)
Tools and IDE
<table>
<thead>
<tr>
<th></th>
<th>more in GPLs</th>
<th>more in DSL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain Size</td>
<td>large and complex</td>
<td>smaller and well-defined</td>
</tr>
<tr>
<td>Designed by</td>
<td>guru or committee</td>
<td>a few engineers and domain experts</td>
</tr>
<tr>
<td>Language Size</td>
<td>large</td>
<td>small</td>
</tr>
<tr>
<td>Turing-completeness</td>
<td>almost always</td>
<td>often not</td>
</tr>
<tr>
<td>User Community</td>
<td>large, anonymous and widespread</td>
<td>small, accessible and local</td>
</tr>
<tr>
<td>In-language abstraction</td>
<td>sophisticated</td>
<td>limited</td>
</tr>
<tr>
<td>Lifespan</td>
<td>years to decades</td>
<td>months to years (driven by context)</td>
</tr>
<tr>
<td>Evolution</td>
<td>slow, often standardized</td>
<td>fast-paced</td>
</tr>
<tr>
<td>Incompatible Changes</td>
<td>almost impossible</td>
<td>feasible</td>
</tr>
</tbody>
</table>
module ADemoModule from cdesignpaper.screenshot imports nothing {

enum MODE { FAIL: AUTO: MANUAL: }

statemachine Counter {
  in start() [no binding]
  [step(int[0..10] size) [no binding] ] trace R2
  out started() [no binding]
  reseted() [no binding] [resetable]
  incremented(int[0..10] newVal) [no binding]

  vars int[0..10] currentVal = 0
  int[0..10] LIMIT = 10

  states (initial = start)

  state start {
    on start [ ] -> countState { send started(); } [resetable]

    start +inEvents (cdesignpaper.screenshot.ADemoModule)
    state step +inEvents (cdesignpaper.screenshot.ADemoModule)
    on step [currentVal + size > LIMIT] -> start { send reseted(); }
    on step [currentVal + size <= LIMIT] -> countState {
      send incremented();
      on start [ ] -> start { send reseted(); } [resetable]
    }
  }

} end statemachine

MODE nextMode(MODE mode, int8_t speed) {
  return MODE, FAIL
  speed < 50 AUTO mode == AUTO MANUAL
  speed >= 50 MANUAL mode == MANUAL
IDE for Lang Developer

Language Workbench

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

xtexx

MPS
Meta Programming System
Design
Concerns
Expressivity
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!

ψ
Design
Concerns
Analyzability
```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] );
}
```
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]);
}
```
Reification
Reification

\[ D_{n+1} \]

\[ D_n \]
Reification

Transformation/Generation

Language Definition
Linguistic Abstraction

Declarative!
Directly represents Semantics.

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

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

bad? maybe.
good? maybe!

Model Checking
SAT Solving

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

Exhaustive Search, Proof!
extended C

```c
module Semaphore from semaphore imports nothing {

    verifiable statemachine statemachine {
        in request(boolean req) <no binding>
            step(int[-10..10] stepCount) <no binding>
            out out(int[0..2] traffic, boolean pedestrian, boolean indicator) => handleOut
    
        vars int[-1..10] counter = 0
                int[0..5] green_val = 2
                int[0..5] yellow_val = 2
                int[0..5] red_val = 4

        states (initial - Init)
            always reachable state Init {
                on step [counter == 0] -> green {
                    send out(2, false, true);
                    counter = 5;
                }

                on step [counter > -1 && counter < 1] -> green {
                    send out(2, false, true);
                    counter = 5;
                }
            }

            always reachable state green {
                on request [counter == -1] -> green {
                    send out(0, false, true);
                    counter = green_val;
                }
                on step [counter > 0] -> green {
                    send out(0, false, true);
                    counter = counter - 1;
                }
            }

    }
}
```
Design
Concerns
Execution
Execution
**Transformation**

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

**Interpretation**

+ Turnaround Time
+ Runtime Change
Transformation

```c
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  
}
```
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 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
<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</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>8250</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>
Transformation

Tests
Simulators
Documentation

$\mathcal{L}_D$

$\mathcal{L}_{D-1}$

Transformation
Behavior

Refrigerators

Example

Refrigerators
Design
Concerns
Completeness
Can you generate 100% of the code from the DSL program?
Manually written code?

Call "black box" code (foreign functions)

Embed code in DSL program

Use composition mechanisms of $L_{D-1}$ (inheritance, patterns, aspects, ...)

Use protected regions (if you really have to...) DON’T!
Incomplete: When?

**Good** for technical DSLs: Devs write $L_{D-1}$ code

**Bad** for business DSLs. Maybe use a $L_{D-1}$ std lib that $L_D$ code can call into?
Roundtripping

$L_D$  
\[\downarrow\]
$L_{D-1}$

$L'_D$

$L'_D$

$L'_{D-1}$
Roundtripping - Don’t!

\[ L_D \quad \downarrow \quad L_{D-1} \quad \ldots \quad L'_{D-1} \quad L'_D \]

Semantic Recovery!
Design Concerns
Modularity
Language Modularity, Composition and Reuse (LMR&C) increase efficiency of DSL development
Referencing
Referencing

Fragment

references

Fragment

references

Fragment

references
Referencing

Example Refrigerators

```c
parameter t_abtaustart: int
parameter t_abtaudauer: int
parameter T_abtauEnde: int

var tauernachlaufSchwelle: int = 0

start:
   entry { state noCooling }

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

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

   exit {
      perform rcfanabschalttask after max( 5, tauernachlaufSchwelle-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
more specialized domains
more specialized languages
Extension

Good for **bottom-up** (inductive) domains, and for use by **technical** DSLs (people)
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 {
        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>Embedding</th>
</tr>
</thead>
<tbody>
<tr>
<td>independent</td>
<td>homogeneous</td>
</tr>
<tr>
<td>languages</td>
<td>heterogeneous</td>
</tr>
<tr>
<td>dependencies</td>
<td>fragment structure</td>
</tr>
<tr>
<td>dependent</td>
<td></td>
</tr>
</tbody>
</table>
Embedding

Example Pension Plans
Design
Concerns
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.
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
## Tables

<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 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>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>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>
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{tabular}{c|c|c}
                & x = 0 & x > 0 \\
                \hline
                y = 0 & 0 & 1 \\
                y > 0 & 1 & 2 \\
            \end{tabular}
        }
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 &quot;5&quot;</td>
<td></td>
</tr>
<tr>
<td>rpm is greater than</td>
<td></td>
</tr>
</tbody>
</table>

- system SHALL display speed
- system SHALL display rpm
- delay is less than "5"
- rpm

- 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

callIncoming

IncomingCall

popup Phone.CallFinished

accept

Active

scene Phone.ActiveCall

entry /
raise acceptCall;
timer = 0;
after 1 s / timer = timer + 1;

after 1 s / timer = timer + 1;

accept

acceptCall

callFinished

hangup

hangupCall

timer

EventDefinition callIncoming
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