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

E.Visser@tudelft.nl
http://eelcovisser.org/

based on material from a paper written with Eelco Visser
This material is based on this book:

http://dslbook.org

available Feb 2013
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**.
Concepts (abstract syntax)
(concrete) Syntax
semantics (generators)
Tools and IDE
<table>
<thead>
<tr>
<th>Feature</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>
C

Components

State Machines

Sensor Access

LEGO Robot Control

Domain Specific

General Purpose
Big Language

with many first class concepts!
Small Language

with a few, orthogonal and powerful concepts
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 tempsensor at

    cooling compartment RC {
        light rclight
        superCoolingMode
doord rcdoor
        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 rccoolling
  }
  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 rccoolling:
  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 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
    }
  }

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
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 {
    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_C, left, 1);
        nxt_motor_set_speed(MOTOR_PORT_T::NXT_PORT_B, right, 1);
    }
}
**3.3 Commutatiegetallen op 1 leven**

\[ D_x = \frac{1}{100} \sum_{t=0}^x \frac{1}{x-t} \]

Example

**3.6 Contante waarde 1 leven/2 levens**

\[ E_x = \frac{D_x}{D_{x+n}} \]

\[ a_x = \frac{1}{x} - 1 \]

\[ a_x = \frac{1}{x} - 0.5 \]

\[ \frac{N_x}{D_x} = \frac{1}{x} \]

\[ a_x = \frac{1}{x} - 0.5 + 0.5 \cdot E_x \]

**4 BN(-ris) koopsommen**
### Elements...

**Rules**

- **Rule Bereken Mutatieperiode**
  - **Result:**
  - **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:**
  
  ```
  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 datum</td>
<td>03/01/2008</td>
<td></td>
<td>Mutatieperiode -</td>
<td>Mutatieperiode = Mutatieperiode Vorig</td>
<td></td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Mutatieperiode = Mutatie</td>
<td>Mutatieperiode Vorig (binnen 1 maand)</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>
<tr>
<td>Periode &lt; 30</td>
<td>03/01/2008</td>
<td></td>
<td>Mutatieperiode -</td>
<td>Mutatieperiode &gt; Mutatieperiode Vorig</td>
<td></td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Mutatieperiode &gt;</td>
<td>Mutatieperiode 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>Mutatieperiode -</td>
<td>Mutatieperiode &gt; Mutatieperiode Vorig</td>
<td></td>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Mutatieperiode &gt;</td>
<td>Mutatieperiode Vorig (meerdere maanden)</td>
<td></td>
<td></td>
<td></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) }
  bloglayout(p.blog){
    placeholder view { postView(p) }
    postComments(p)
  }
}
define permalink(p: Post) {
  navigate post(p, p.urlTitle) { elements }
}
Terms & Concepts
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

A representation of a set of components of a process, system, or subject area, generally developed for understanding, analysis, improvement, and/or replacement of the process.
an abstraction or simplification of reality
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!
Model Purpose

- code generation
- analysis and checking
- platform independence
- stakeholder integration
Model Purpose

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

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

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

- **body of knowledge in the real world**
- **deductive top down**
- **inductive bottom up**

**existing software (family)**
Domain

body of knowledge in the real world

deductive top down

Example Penion Plans
Example Refrigerators
existing software (family)
A DSL $L_D$ for D is a language that is specialized to encoding $P_D$ programs.

more efficient

smaller
Programs are trees.
Fragments are subtrees w/ root

```
  f
  /\  
 M  A B
  /   /
 C  D  
 /    /
 E    F
     /  
      G
```
Parent-Child Relation

```
M
  |
  A
  |
  B
  |
C   D
  |
  E  F
  |
  G
```
Programs and Fragments

\[ f \quad M \quad \]  
\[ A \quad B \quad \]  
\[ C \quad D \quad \]  
\[ E \quad F \quad G \quad \]

\[ fo \Rightarrow element \Rightarrow fragment \]
Programs are graphs, really.

```
M
  /\    \\
 A  B     D
  /\    /\    /\    \\
 C  E  F  G

reference
```
Programs are graphs, really.
Languages are sets of concepts

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

\[ L \xrightarrow{lo} \text{concept} \xrightarrow{} \text{language} \]
Programs and languages

```
M
 A  B
 C  D
 E  F  G
```

```
C_1  C_2  C_n
```

```
LocalVariableDeclaration
```

```
x = 3;
```

```
co ⟷ element ⟷ concept
```
Language: concept inheritance

Diagram:

- **Language:** $L_1$ and $L_2$
- **Concepts:** $C_1$, $C_2$, $C_3$
- **Inheritance:** $Inh_1$, $super$, $sub$
- **Statements:** $Statement$, $LocalVariableDeclaration$
Language does not depend on any other language

\[ \forall r \in \text{Refs}_l \mid \text{lo}(r.to) = \text{lo}(r.from) = l \]
\[ \forall s \in \text{Inh}_l \mid \text{lo}(s.super) = \text{lo}(s.sub) = l \]
\[ \forall c \in \text{Cdn}_l \mid \text{lo}(c.parent) = \text{lo}(c.child) = l \]

Independence

Fragment does not depend on any other fragment

\[ \forall r \in \text{Refs}_f \mid \text{fo}(r.to) = \text{fo}(r.from) = f \]
\[ \forall e \in \text{Ef} \mid \text{lo}(\text{co}(e)) = l \]
Independence

Hardware:

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

Cooling Algorithm

```c
macro kompressorAUS {
    set cc.c1->active = false
    perform ccfanabschalttask 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 {
                }
            }
    } end statemachine

    var Counter c1;

    exported test case test1 {
        initsm(c1);
        assert(0) isInState<c1, initialState>;
        trigger(c1, start);
        assert(1) isInState<c1, countState>;
        } test1(test case)
    }
}
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 { }
    }
} end statemachine

var Counter c1;

exported test case test1 {
    initsm(c1);
    assert(0) isInState<c1, initialState>;
    trigger(c1, start);
    assert(1) isInState<c1, 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

completeness
paradigms
modularity
concrete syntax

process
Shorter Programs

More Accessible Semantics
For a limited Domain!

Domain Knowledge encapsulated in language
Def: Expressivity

A language $L_1$ is more expressive in domain $D$ than a language $L_2$ if for each $p \in P_D \cap P_{L_1} \cap P_{L_2}$, $|p_{L_1}| < |p_{L_2}|$
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

\[ D_{n+1} \]

\[ D_n \]

\[ == \]
Reification

Language Definition

Transformation/Generation

\[ \square \otimes \equiv \triangle \]
```java
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

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!
Another Example

```java
if (isConnected(port)) {
    port.doSomething();
}
```
Another Example

```java
if (isConnected(port) || true) {
    port.doSomething();
}
```

Turing Complete!

Requires Semantic Analysis!
with port (port) {
    port.doSomething();
}
Linguistic Abstraction

Example Extended C

```c
exported component AnotherDriver extends Driver {
    ports:
    requires optional ILogger logger
    provides IDriver cmd
    contents:
    field int count = 0

    int setDriverValue(int addr, int value) <- op cmd.setDriverValue {
        with port (logger) {
            logger.log("some error message");
        } with port
        return 0;
    }
}
```
Linguistic Abstraction

In-Language Abstraction

Libraries
Classes
Frameworks
Linguistic Abstraction

In-Language Abstraction

User-Definable Language

Simpler Language

Analyzable

Better IDE Support

Abstraction
Linguistic Abstraction

In-Language Abstraction

User-Definable Language

Simpler Language

Better IDE Support

Analyzeable

Special Treatment!
Linguistic Abstraction

Std Lib

In-Language Abstraction
lib stdlib {
  command compartment::coolOn
  command compartment::coolOff
  property compartment::totalRuntime: int readonly
  property compartment::needs Cooling: bool readonly
  property compartment::couldUseCooling: bool readonly
  property compartment::targetTemp: int readonly
  property compartment::currentTemp: double readonly
  property compartment::isCooling: bool readonly
}
Language
Evolution
Support
Customization vs. Configuration
Precision vs. Algorithmics
Coverage

expressivity  completeness
coverage  paradigms
semantics  modularity
separation of  concrete
concerns  syntax
process
Domain $D_L$ defined inductively by $L$
(the domain that can be expressed by $L$)

$$C_L(L) = 1 \text{ (by definition)}$$

not very interesting!
Def: Coverage

to what extent can a language $L$ cover a domain $D$

$$C_D(L) = \frac{\text{number of } P_D \text{ programs expressable by } L}{\text{number of programs in domain } D}$$
Def: Coverage

why would $C_D(L)$ be $\neq 1$?

1) $L$ is deficient

2) $L$ is intended to cover only a subset of $D$, corner cases may make $L$ too complex

Rest must be expressed in $D_{-1}$
Def: Coverage

Coverage is full.
You call always write C.
Def: Coverage

Only a particular style of web apps are supported.

Many more are conceivable.
Def: Coverage

DSLs are continuously evolved so the relevant parts of the deductive domain are supported.
Semantics & Execution

expressivity
coverage
semantics
separation of concerns
completeness
paradigms
modularity
concrete syntax

process
Static Semantics

Execution Semantics
Static Semantics

Execution Semantics
Static Semantics

Constraints

Type Systems
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!
var int x = 2 * someFunction(sqrt(2));

Assign fixed types

What does a type system do?
Assign fixed types

Derive Types

What does a type system do?
var int x = 2 * someFunction(sqrt(2));

Assign fixed types

Derive Types

Calculate Common Types

What does a type system do?
What does a type system do?
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
macro kompressorAus {
    set cc.c1->active = "aString"
    perform ccfanausschalt 
    set cc.ccfan->active = "aString"
}
What does it all mean?

Execution Semantics
Def: Semantics
... via mapping to lower level

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

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

OB: Observable Behaviour (Test Cases)
Def: Semantics

... via mapping to lower level

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

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

\[ L_D \downarrow \]

Transformation

\[ L_{D-1} \]

Known Semantics!
Transformation

Correct!?

$L_D$

$L_{D-1}$

Transformation
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 rcfanabschalttask 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 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
## Transformation

<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>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></td>
<td>1985-12-31</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></td>
<td>1987-12-31</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>387.7449</td>
</tr>
<tr>
<td></td>
<td>1998-12-31</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>1998-12-31</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></td>
<td>1998-12-31</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>

### Example Pension Plans
### Property Values

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

### Queue

<table>
<thead>
<tr>
<th>Event</th>
<th>Data</th>
</tr>
</thead>
</table>

### Commands

- St... Command

### Variable Values

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

### Running Tasks

- Task | Sinc...
Multi-Stage

Modularization
Multi-Stage: Reuse

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

Components
Efficient Mappings

C Type System

C (MPS tree)

Syntactic
Correctness,
Headers

C Text

Example

Extended C
Multi-Stage: Reuse

Reusing Early Stages

Portability
Multi-Stage: Reuse

```
L_3

\downarrow

L_2

\downarrow

L_1

\downarrow

L_0

\downarrow

L_{1b}

\downarrow

L_{0b}

\downarrow

Java

C#
```

Example

Extended C
Multi-Stage: Preprocess

Adding an optional, modular emergency stop feature
Platform

Generated Application
- Domain Frameworks
- Libraries
- Middleware
- Drivers
- Operating System
Platform

Temporal Data Versioning
Database Access
Transactions
Distribution (JEE)
Platform

- Generated Application
- Domain Frameworks
- Libraries
- Middleware
- Drivers
- Operating System

Data Collection
HAL
Device Drivers

Example Refrigerators
Interpretation

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 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></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></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></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></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->status < 0) )
  state rccooling
  on isDown ( RC.rcdoor->open ) {
    set RC.rcfan->active = true
    set RC.rclight->active = false
    perform rcfanabschallassnachlauf 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 rcfanabschallassnachlauf 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

Example
Refrigerators
### Property Values

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

### Queue

<table>
<thead>
<tr>
<th>Event</th>
<th>Data</th>
</tr>
</thead>
</table>

### Commands

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>St... Command</td>
</tr>
</tbody>
</table>

### Variable Values

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

### Assert Selected Variable

### Assert Selected Property

### Running Tasks

<table>
<thead>
<tr>
<th>Task</th>
<th>Sinc...</th>
</tr>
</thead>
</table>
A program at $D_0$ that acts on the structure of an input program at $D_{>0}$
Interpretation

An interpreter :-)
Transformation

+ Code Inspection

Interpretation
Transformation

+ Code Inspection

OSGi Generators

Interpretation

Example

Components
Transformation

+ Code Inspection
+ Debugging

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>Platform Interactions</td>
<td></td>
</tr>
</tbody>
</table>
Transformation

+ Code Inspection
+ Debugging
+ Performance & Optimization

Interpretation
Transformation

+ Code Inspection
+ Debugging
+ Performance & Optimization

Interpretation

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>
Transformation

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

Interpretation

+ Turnaround Time
Transformation

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

Interpretation

+ Turnaround Time Testing
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

\[ \mathcal{L}_D \downarrow \text{Transformation} \]

\[ \mathcal{L}_{D-1} \]
Multiple Mappings

... at the same time

\[ L_D \]

\[ L_x \quad L_y \quad L_z \]

Similar Semantics?
Multiple Mappings

... at the same time

$L_D \circ T$

$L_x \circ T$

$L_y \circ T$

$L_z \circ T$

all green!

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

\[ \mathbf{L}_D \circ \mathbf{T} \]

\[ \mathbf{L}_x \circ \mathbf{T} \]
\[ \mathbf{L}_y \circ \mathbf{T} \]
\[ \mathbf{L}_z \circ \mathbf{T} \]

Similar Semantics?

all green!
<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>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>
Refrigerators

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 abtau
assert-value cc.c1->active is false

step 5
assert-currentstate-is abtau
assert-value cc.c1->active is false
step 15
assert-currentstate-is noCooling

Example
Refrigerators
Multiple Mappings

... alternatively, selectably

\[ L_D \]

Extend \( L_D \) to include explicit data that determines transformation

\[ L_x \]
\[ L_y \]
\[ L_z \]
Multiple Mappings

... alternatively, selectably

\[ L_D \]

Extend \( L_D \) to include explicit data that determines transformation

Example

Restricted Port leads to reduced overhead C
Multiple Mappings

... alternatively, selectably

External Data:
- Switches
- Annotation Model

\[
L_D \\
\downarrow\downarrow\downarrow \downarrow \\
L_x \quad L_y \quad L_z
\]
Multiple Mappings
... alternatively, selectably

External Data:
- Switches
- Annotation Model

Switch Control
Java vs. C

Example
Pension Plans
Multiple Mappings

... alternatively, selectably

Heuristics: Analyze model to try to decide

\[ L_D \]

\[ L_x \]
\[ L_y \]
\[ L_z \]
Multiple Mappings

... alternatively, selectably

\[ L_D \]

\[ L_x \]

\[ L_y \]

\[ L_z \]

TESTING!
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!
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[-..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(2, false, true);
                    counter = green_val;
                }
                on step [counter > 0] -> green {
                    send out(0, false, true);
                    counter = counter - 1;
                }
            }
    }
}
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
            | x == 0 x > 0 |
            | y == 0 0 1 |
            | y > 0 1 2 |
        }
    }
}
Separation of Concerns

expressivity
coverage
semantics
separation of concerns

completeness
paradigms
modularity
concrete
syntax

process
Several Concerns

... in one domain
Several Concerns

... in one domain

integrated into one fragment

separated into several fragments
Viewpoints

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

independent

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

dependent
Viewpoints

Hardware

Behaviour

Tests
Viewpoints: Why?

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

Hardware
Product Management

Behaviour
Thermodynamics Experts

Tests
Viewpoints

independent

sufficient?
contains all the data for running a meaningful transformation
Viewpoints

sufficient
Hardware structure
documentation

sufficient
implementation
code

Example
Refrigerators
Viewpoints: Why?

1:n Relationships
Viewpoints

Hardware

Tests

1:n

Behaviour

n:1

Example

Refrigerators
Viewpoints

- Well-defined Dependencies
- No Cycles!
- Avoid Synchronization!

(unless you use a projectional editor)
Viewpoints: Why?
Progressive Refinement
Views on Programs

Achmea demo plan

- T-OP65-TOTAAL-2006
  - Het totale ouderdomspensioen, opgebouwd in de oude of de nieuwe regeling

- 1999-01-01

- 2006-01-01
  - T-OP-TOTAAL → T-OP65-VTZ-2006
  - T-OP65-WOV-2006
  - J-IDX
  - J-IDC-RGL
Completeness

expressivity  completeness
coverage  paradigms
semantics  modularity
separation of  concrete
concerns  syntax

process
Can you generate 100% of the code from the DSL program?

More generally: all of $D_{-1}$
Semantics:

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

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

Introduce \( G \) ("generator"):

\[ OB(p) =: OB(G(p)) =: OB(q) \quad \text{complete} \]
\[ OB(G(p)) \subset OB(p) \quad \text{incomplete} \]
Incomplete: What to do?

\[
\text{OB}(F_D) \neq F_{D-1}
\]
Incomplete: What to do?

Manually written code!

\[ F_D \]

\[ \text{OB}(F_D) = F_{D-1} + F_{D-1, \text{man}} \]
Manually written code?

Call "black box" code
(foreign functions)
Manually written code?

Call "black box" code
(foreign functions)

Embed $L_{D-1}$ code in $L_D$ program
Manually written code?

Call "black box" code
(foreign functions)

Embed $L_{D-1}$ code in $L_D$ program

Embed C statements
in components, state machines
or decision tables
Manually written code?

Call "black box" code (foreign functions)

Embed $L_{D-1}$ code in $L_D$ program

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

Manually written code?

Call "black box" code (foreign functions)

Embed $L_{D-1}$ code in $L_D$ program

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

Generate base classes with abstract methods; implement in subclass
Manually written code?

Call "black box" code (foreign functions)

Embed $L_{D-1}$ code in $L_D$ program

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

Use protected regions (if you really have to...)
Manually written code?

Call "black box" code (foreign functions)

Embed $L_{D-1}$ code in $L_D$ 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?
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?
Prevent Breaking Promises!

class B extends BBase {
    public void doSomething() {
        Registry.get("A").someMethod();
    }
}

```java
class B extends BBase {
    public void doSomething() {
        Registry.get("A").someMethod();
    }
}
```
Prevent Breaking Promises!

Better:

Dependency Injection
Static analysis tools
Roundtripping

\[ L_D \quad \downarrow \quad L_{D-1} \quad \cdots \quad L'_{D-1} \quad L'_D \]
Roundtripping - Don't!

 Semantic Recovery!
Fundamental Paradigms

expressivity
coverage
semantics
separation of concerns

completeness
paradigms
modularity
concrete syntax

process
Structure

Modularization, Visibility

Namespaces, public/private importing
Structure

Modularization, Visibility

Namespaces, public/private importing

divide & conquer reuse

stakeholder integration
Structure

Partitioning (Files)

VCS Unit
Unit of sharing
Unit of IP

!= logical modules
may influence language design
Structure
Modulearization, Visibility

```c
module Module1 from HPL.main imports Module2 {

    exported var int aReallyGlobalVar;

    struct aLocallyVisibleStruct {
        int x;
        int y;
    };

    exported int anExportedFunction() {
        return anImportedFunction/Module2();
    } anExportedFunction (function)
}
```
Structure

Modularization, Visibility

```java
namespace com.mycompany.test {
  system testSystem {
    instance dc: DelayCalculator
    instance screen1: InfoScreen
    instance screen2: InfoScreen
    connect dc.screens to
      (screen1.default, screen2.default)
  }
}
```
Structure

Partitioning (Files)

change impact
link storage
model organization
tool integration
Spec vs. Implementation

plug in different Impls
different stakeholders
Structure

Spec vs. Impl.

```c
exported component AnotherDriver extends Driver {
    ports:
        provides IDiagnostic diag
        requires optional ILogger logger
        requires ILowLevel lowlevel restricted to LowLevelCode
    contents:
        field int8_t count = 0

        override int8_t setDriverValue(int8_t addr, int8_t value) <- op cmd.setDriverValue {
            with port (logger) { logger.log("SomeMessage"); } with port
            lowlevel.doSomeLowlevelStuff(10);
            count++;
            return 1;
        }

        int8_t diag_getCount() <- op diag.getCount {
            return count;
        }
}
```

```c
exported c/s interface ITrafficLights {
    int8_t setColor(TLCommand cmd);
}

c/s interface IDriver {
    int8_t setDriverValue(int8_t addr, int8_t value);
}

c/s interface IDiagnostic {
    int8_t getCount();
}

c/s interface ILogger {
    void log(string message);
}

c/s interface ILowLevel {
    int8_t doSomeLowlevelStuff(int8_t y);
}
```
Specialization

Liskov substitution P leaving holes ("abstract")
Structure

Specialization

Liskov substitution P leaving holes ("abstract")

variants (in space) evolution (over time)
Pension Plans can inherit from other plans.

Rules can be abstract;

Plans with abstract rules are abstract.
Structure

Superposition, Aspects
merging
overlay
AOP

modularize cross-cuts
Structure

Superposition, Aspects

```java
component DelayCalculator {
    ...
}
component AircraftModule {
    ...
}
component InfoScreen {
    ...
}

aspect (*) component {
    provides mon: IMonitoring
}
```
Behavior

Not all DSLs specify behavior
Some just declare behavior

This section is not for those!
Behavior

Imperative

sequence of statements changes program state

<table>
<thead>
<tr>
<th></th>
<th>write</th>
<th>understand</th>
<th>debug</th>
<th>analyze</th>
<th>performance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>simple</td>
<td>simple -</td>
<td>simple</td>
<td>hard</td>
<td>good</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(step)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Behavior

Imperative

sequence of statements
changes program state

Example
Refrigerators
Behavior

Functional

functions call other functions. no state. No aliasing.

<table>
<thead>
<tr>
<th>write</th>
<th>understand</th>
<th>debug</th>
<th>analyze</th>
<th>performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>simple -</td>
<td>simple</td>
<td>simple (tree)</td>
<td>good</td>
<td>good -</td>
</tr>
</tbody>
</table>
Behavior

Functional

functions call other functions. no state. No aliasing.

Example Pension Plans
Behavior

Functional

Example Pension Plans
Behavior

Functional

pure expressions are a subset of functional (operators hard-wired)
guards
preconditions
derived attributes
Behavior

Declarative
only facts and goals.
no control flow.
eval engine, solver (several)

<table>
<thead>
<tr>
<th>write</th>
<th>understand</th>
<th>debug</th>
<th>analyze</th>
<th>performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>simple</td>
<td>simple -</td>
<td>hard</td>
<td>depends</td>
<td>often bad</td>
</tr>
</tbody>
</table>
Behavior

Declarative

concurrency

constraint programming

solving

logic programming
Behavior

Declarative
only facts and goals.
no control flow.
eval engine, solver (several)

```log
section posts
  define page post(p: Post, title: String) {
    title{ output(p.title) }
    bloglayout(p.blog){
      placeholder view { postView(p) }
      postComments(p)
    }
  }
  define permalink(p: Post) {
    navigate post(p, p.urlTitle) { elements }
  }
```

Example

Web
DSL
Behavior

Declarative

Example

Extended C
Behavior

Reactive

reactions to events, more events are produced.
Communication via events and channels/queues

<table>
<thead>
<tr>
<th>write</th>
<th>understand</th>
<th>debug</th>
<th>analyze</th>
<th>performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>simple</td>
<td>simple/hard</td>
<td>hard</td>
<td>simple</td>
<td>can be good</td>
</tr>
</tbody>
</table>
**Behavior**

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

**Refrigerators**

**Example**

<table>
<thead>
<tr>
<th>Event</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>St...</td>
<td>Command</td>
</tr>
</tbody>
</table>

| Variable Values

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

| Running Tasks

<table>
<thead>
<tr>
<th>Task</th>
<th>Sinc...</th>
</tr>
</thead>
</table>

**Simulate View**

- **Status**
  - Current Test: KIRAabtauen
  - Current State: -
  - Current Step: -

- **Control**
  - Autorun
  - Single Step
  - Enable Breakpoints

- **Property Values**
  - RC.accumulatedRun: 80
  - RC.needsCooling: false
  - c1.active: false
  - ccfan.active: false
  - rcdoor.open: false
  - rceva.evaTemp: 20
  - rcfan.active: false

- **Queue**
  - Event: St...
  - Data: Command

- **Commands**
  - St...

- **Variable Values**
  - Variable: tuerNachlaufSchwel
  - Value: 0
Behavior

Data Flow

chained blocks consume continuous data that flows from block to block

<table>
<thead>
<tr>
<th>write</th>
<th>understand</th>
<th>debug</th>
<th>analyze</th>
<th>performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>simple</td>
<td>simple/hard</td>
<td>hard</td>
<td>simple</td>
<td>can be good</td>
</tr>
</tbody>
</table>
Behavior

Data Flow

continuous, calc on change
quantized, calc on new data
time triggered, calc every x
Behavior

Data Flow

Embedded Programming

Enterprise ETL & CEP
Behavior

State Based

states, transitions, guards, reactions

event driven, timed

<table>
<thead>
<tr>
<th>write</th>
<th>understand</th>
<th>debug</th>
<th>analyze</th>
<th>performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>simple -</td>
<td>simple/hard</td>
<td>s/h</td>
<td>simple +</td>
<td>can be good</td>
</tr>
</tbody>
</table>
Behavior

State Based

```
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
    }
  }
```
Behavior

Combinations

data flow uses functional, imperative or declarative lang inside block
Behavior

Combinations

state machines use expressions in guards and often an imperative lang in actions
Behavior

Combinations

```
start:
  entry { state noCooling }

state noCooling:
  check ( (RC\rightarrow needsCooling) \&\& (cc.c1\rightarrow stehzeit > 333) ) {
    state rccooling
  }  
  on isDown ( RC.rcdoor\rightarrow open ) {
    set RC.rcfan\rightarrow active = true
    set RC.rclight\rightarrow active = false
    perform rcfanabschalttask after 10 {
      set RC.rcfan\rightarrow active = false
    }
  }

state rccooling:
  entry { set RC.rcfan\rightarrow active = true }
  check ( !(RC\rightarrow needsCooling) ) {
    state noCooling
  }  
  on isDown ( RC.rcdoor\rightarrow open ) {
    set RC.rcfan\rightarrow active = true
    set RC.rclight\rightarrow active = false
    set tuerNachlaufSchwelle = currStep + 30
  }  
  exit {
    perform rcfanabschalttask after max( 5, tuerNachlaufSchwelle-currStep ) {
      set RC.rcfan\rightarrow active = false
    }
  }
```
Behavior

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

CallHandling

CallCycle

Waiting

callIncoming

IncomingCall

popup Phone.CallFinished

hangup

accept

Active

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

Finish

callIncoming

callFinished

popup Phone.CallFinished

finished

hangup

hangupCall

timer

EventDefinition callIncoming
Behavior

Combinations

purely structural languages often use expressions to specify constraints

```c/s interface IDriver {
    int setDriverValue(int addr, int value);
    pre value > 0
}
```
Language Modularity

expressivity  completeness
coverage  paradigms
semantics  modularity
separation of concerns  concrete
concerns  syntax

process
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</th>
<th>Reuse</th>
<th>Embedding</th>
</tr>
</thead>
<tbody>
<tr>
<td>languages</td>
<td>Referencing</td>
<td>Extension</td>
</tr>
<tr>
<td>dependencies</td>
<td>homogeneous</td>
<td>heterogeneous</td>
</tr>
</tbody>
</table>

fragment structure
Dependencies & Fragment Structure:

Referencing

Reuse

Extension

Embedding
Referencing

<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 languages  
-dependencies

-homogeneous fragment structure  
-heterogeneous fragment structure
Referencing

Dependent

No containment
Referencing

Used in Viewpoints
Referencing
Referencing

```java
// Example

parameter t_abtauStart: int
parameter t_abtaudauer: int
parameter t_abtauEnde: int

var turNachlaufSchwelle: 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 rcfanabschalfltask 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 turNachlaufSchwelle = currStep + 30
    }
    exit {
        perform rcfanabschalfltask after max( 5, turNachlaufSchwelle-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.nceva->evaTemp = 10
assert-currentstate-is abtauSen
assert-value cc.c1->active is false
mock: set RC->accumulatedRuntime = 0
step 15
assert-currentstate-is abtauSen
assert-value cc.c1->active is false
```

Refrigerators
Containment

Dependent

Extension

Independent
- languages
- dependencies
- dependent

Reuse

Embedding

Referencing

Extension

Homogeneous

Heterogeneous

Fragment structure
more specialized domains
more specialized languages
Extension

$D_{n+1}$

$D_n$
Extension

\[ D_{n+1} \]

\[ D_n \]
**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
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_S1);
        nxt_motor_set_speed(MOTOR_PORT_T::NXT_PORT_S1);
    }
}
Extension

Diagram:

- Module
- IModule Content
  - Struct
  - Function
- State Machine

Example Extended C
**Reuse**

- Independent

---

**No containment**

---

**Table:**

<table>
<thead>
<tr>
<th>Reuse</th>
<th>Embedding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Languages</td>
<td>dependent</td>
</tr>
<tr>
<td>Dependencies</td>
<td>dependent</td>
</tr>
</tbody>
</table>

- Referencing
- Extension

- Homogeneous
- Heterogeneous

- Fragment structure
Often the referenced language is built expecting it will be reused.

Hooks may be added.
<table>
<thead>
<tr>
<th>Reuse</th>
<th>Embedding</th>
</tr>
</thead>
<tbody>
<tr>
<td>heterogeneous</td>
<td>dependent language</td>
</tr>
<tr>
<td>homogeneous</td>
<td>dependencies</td>
</tr>
</tbody>
</table>

Diagram:

- **Embedding**
  - **I_2**
    - B3
    - B4
  - **I_A**
    - B5
  - **I_1**
    - A1
    - A2
    - A3

- Arrows indicate dependencies and relationships between elements.
Embedding

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

Independent languages dependencies dependent
### Elements

#### 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.

#### Algorithm

```plaintext
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 - Mutatiedatum = Mutatiedatum Vorig</td>
<td>3</td>
<td></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>15</td>
<td></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 (meerdere maanden)</td>
<td>90</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Embedding often uses Extension to extend the embedded language to adapt it to its new context.
Challenges - Syntax

Extension and Embedding requires modular concrete syntax

Many tools/formalisms cannot do that
Challenges - Type Systems

Extension: the type system of the base language must be designed to be extensible/overridable
Reuse and Embedding: Rules that affect the interplay can reside in the adapter language.
Challenges - Trafo & Gen Referencing (I)

Two separate, dependent single-source transformations

Written specifically for the combination

Can be Reused
Challenges - Trafo & Gen

Referencing (II)

A single multi-sourced transformation
Challenges - Trafo & Gen

Referencing (III)

A preprocessing trafo that changes the referenced frag in a way specified by the referencing frag.
Challenges - Trafo & Gen Extension

Transformation by assimilation, i.e. generating code in the host lang from code expr in the extension lang.
Challenges - Trafo & Gen Extension

```
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
Challenges - Trafo & Gen

Reuse (I)

Reuse of existing transformations for both fragments plus generation of adapter code
Challenges - Trafo & Gen
Reuse (II)

composing transformations
Challenges - Trafo & Gen

Reuse (III)

generating separate artifacts plus a weaving specification
Challenges – Trafo & Gen Embedding (I)

a purely embeddable language may not come with a generator.

Assimilation (as with Extension)
Challenges - Trafo & Gen Embedding (II)

Adapter language can coordinate the transformations for the host and for the embedded languages.
### Concrete Syntax

<table>
<thead>
<tr>
<th>expressivity</th>
<th>completeness</th>
</tr>
</thead>
<tbody>
<tr>
<td>coverage</td>
<td>paradigms</td>
</tr>
<tr>
<td>semantics</td>
<td>modularity</td>
</tr>
<tr>
<td>separation of concerns</td>
<td>concrete syntax</td>
</tr>
</tbody>
</table>

---

process
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

... then add other forms, if really necessary
Editors simple to build
Productive
Easy to integrate w/ tools
Easy to evolve programs
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 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>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 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 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
        }
    }

    |   | x == 0 | x > 0 |
    ---|-------|-------|
    y == 0 | 0     | 1     |
    y > 0  | 1     | 2     |
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

### Keywords:
- and
- is disabled
- is enabled
- is equal to
- is greater than
- is less than
- is not equal to
- or
- xor
Combinations
Process

expressivity
coverage
semantics
separation of concerns

completeness
paradigms
modularity
concrete syntax

process
Domain Analysis

Interview Experts

Get Feedback

Structure their Knowledge

Create Examples

Build the Language
Iterate to goal

language size/complexity

ss

time
Create example-based tutorials!
Domain Folks
Programming?

Precision vs. Algorithmics!
Domain Folks Programming?

DU
Coding

DU/Dev
Paired

Dev Coding
DU Reviewing
DSL as a Product

Release Plan
Bug Tracker
Testing!
Support
Documentation
Reviews become easier --- less code, more domain-specific
The End.

www.voelter.de
voelterblog.blogspot.de
@markusvoelter
+Markus Voelter
This material is based on this book:

http://dslbook.org

available Feb 2013